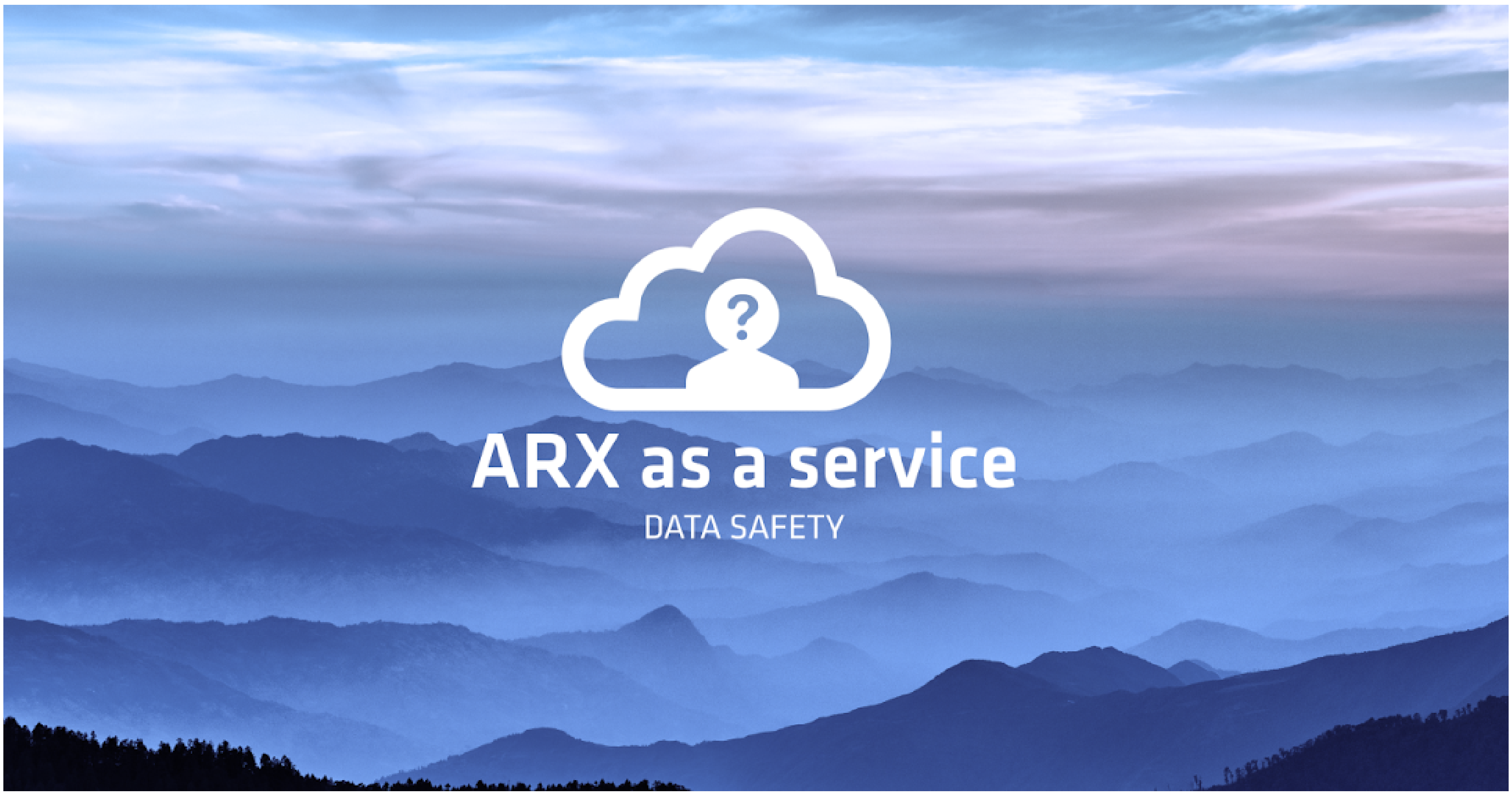


Lord André Groseth, Sondre Halvorsen, Viktor Vartdal Johansen,
Julian Sagen and Jeremiah Augie Salita Uy

OsloMet - storbyuniversitetet
Postboks 4, St. Olavs plass
0130 Oslo

Project Group: 19-08	
Availability: Open Source	

BACHELOR THESIS

Title: Anonymization as a Service	**Date:** 23.05.2019
	Total Pages: 218 **Appendix:** 32
Students: Lord André Groseth (s181365) Sondre Halvorsen (s305349) Viktor Vartdal Johansen (s315615) Julian Sagen (s315584) Jeremiah Augie Salita Uy (s181369)	**Supervisor:** Eva Hadler Vihovde

Collaborator: NAV Data and Insight	**Supervisor:** Robindra Prabhu

Summary:
Anonymization as a Service is a bachelor project thesis done at OsloMet in collaboration with NAV IT. The project team has developed a solution to provide data anonymization against re-identification risk, as a service for the data scientists working at NAV IT.

To accomplish this goal, a microservice was developed. The service is built using Spring Boot and utilizes the ARX de-identification library to implement the core functionality. The service is packaged and deployed as a Docker container.

A Python package was developed for the data scientists to interact with the microservice, and as a stretch goal, a web application was developed for a wider user group.

Three key expressions:

Data anonymization	Public sector	Data science

About the report

This report has been optimized for digital reading by utilizing links, footnotes, and code blocks. While the digital version is recommended, this report can also be read in paper format.

Throughout this report, the following expressions have the listed definitions:

- 'the product(s)':
  Subsystems of the solution developed during the course of this project. Note the expressions 'product documentation', 'product specification' and 'minimum viable product' do not concern the subsystems specifically, but the solution as a whole. Their use of 'product' are exceptions to the above definition, so they retain their innate meaning.
- 'the solution':
  Also referred to as 'Anonymization as a Service' in the documentation, consists of the products working together as a whole, altogether aspiring to provide the functionality requested in the product specification.
- 'the project':
  All work involved in the whole bachelor's project, including work process.
- 'the team':
  All five team members contributing to the project.

The project is *open source and is distributed with* MIT license. The client, NAV IT, is a government agency following a policy where all software that can be *open source*, should be *open source*.

The report has been written in English at the request of the client, as the solution developed has a potentially international user group.The intention is to make global consumption of, and contribution to the project as easy as possible.

Terminology is expressed in *italic text* throughout the report, and listed along with an explanation in chapter 7.1 Terminology of the Appendix. All sources are linked in footnotes.

See chapter 7.3 Testimonial (Norwegian) in the Appendix for the client's, NAV IT, description of the work the team has completed.

Chapter introduction

Following this paragraph is a list that introduces each chapter of the report with a short summary. Its purpose is to provide a sense of reading order. As chapter relevancy depends on the reader's approach, all readers should use this list for direction.

The chapter order is intended to provide a gradual understanding of the project. As such, it represents the report's recommended reading order from start to finish.

1 Presentation

Introduces the project, its background, and the *project stakeholders* involved. The document provides a good overview of the solution for readers not interested in reading the more detailed product documentation. It has been written with the expectation that this chapter is read before any other chapter. This chapter also includes the project's conclusion.

2 De-Identification

This chapter is intended for readers without prior knowledge of de-identification or anonymization. Furthermore, this chapter presents the problem domain of the project, providing essential information for understanding the project's purpose.

3 Process Documentation

This chapter provides insight into the team's work process, including method and tools. The document includes the teams decisions and experiences using new and modern tools and development processes, and has high value for users looking to gain insight in terms of estimation, planning and executing a software project. Readers interested in the team's experiences with the strategies and tools utilized should read this chapter.

4 Test Documentation

The purpose of this chapter is to give the reader a detailed description of each test and how the test is implemented. This document is written with the expectation that the reader has basic programming and testing knowledge, and has read the process documentation.

5 Product Documentation

This chapter describes the products developed on a architectural and technical level. People working with operations, maintenance, or future development of the project should read this chapter. The reader is expected to have moderate programming knowledge as it is advantageous for understanding the concepts of this document.

6 User Manual

The purpose of this chapter is to give the reader a step-by-step guide on how to set up the solution and how to benefit from the products. Moderate programming knowledge is advantageous in setting up the solution. It is recommended to read the presentation documentation and the de-identification chapter before reading the user manual.

7 Appendix

This chapter contains explanations of terminology, HTTP request and response body, the product specification and the testimonial from NAV IT.

Table of Contents

1 Presentation **13**
1.1 Introduction 13
1.2 Presentation of the group 14
1.3 Presentation of the client 15
1.3.1 Project stakeholders at NAV IT 15
1.4 Project background 16
1.4.1 Introduction to de-identification 17
1.4.2 Current situation 17
1.5 The solution - Anonymization as a Service 18
1.5.1 Solution description 18
1.5.1.1 ARXaaS - The service 19
1.5.1.2 Clients 21
1.5.1.2.1 PyARXaaS - the wrapper package 21
1.5.1.2.2 WebARXaaS - web application (stretch goal) 24
1.5.2 Deployment 25
1.5.3 License 26
1.6 Future Development 26
1.7 Conclusion 27
1.7.1 Value delivered 27

2 De-identification **29**
2.1 Introduction 29
2.2 De-identification overview 29
2.2.1 Pseudonymization 30
2.2.2 Anonymization 30
2.3 Main anonymization techniques 31
2.4 Risk assessment 31
2.5 Privacy Models 33
2.5.1 K-Anonymity 33
2.5.2 L-Diversity 33
2.5.3 T-Closeness 33
2.6 Generalization Hierarchies 34
2.7 Anonymizing a dataset 35

3 Process documentation **36**
3.1 Introduction 36
3.2 Planning and methods 36
3.2.1 Development methods 36
3.2.1.1 Agile work process 37
3.2.1.2 Continuous integration and Continuous delivery 40
3.2.1.2.1 CI/CD pipeline 41

3.2.1.3 Test driven development (TDD) 43
3.2.1.4 Code style 44
3.2.1.5 Code reviews 44
3.2.1.6 Feature slicing 45
3.2.2 Project phases 45
3.2.2.1 Initiation 45
3.2.2.2 Planning 45
3.2.2.3 Execution 47
3.2.2.4 Project conclusion and documentation 47
3.2.3 Planning tools 47
3.2.4 Competence building 49
3.2.4.1 Literature 49
3.2.4.2 Courses 49
3.2.4.2.1 Online Courses 49
3.2.4.2.2 Presentations and workshops 50
3.2.5 Budget 50
3.3 Development process 51
3.3.1 Descriptions of the sprints 51
3.3.1.1 Sprint 0 51
3.3.1.2 Sprint 1 52
3.3.1.3 Sprint 2 54
3.3.1.4 Sprint 3 57
3.3.1.5 Sprint 4 60
3.3.1.6 Sprint 5 63
3.3.1.7 Sprint 6 66
3.3.1.8 Sprint 7 69
3.3.2 Development tools 73
3.3.3 Lessons learned during development 74
3.4 Product specification 75
3.4.1 Main specifications 75
3.4.1.1 System Diagram 75
3.4.1.2 Requirements 76
3.4.1.2.1 Functional requirements 76
3.4.1.2.2 Non-Functional Requirements 77
3.4.1.2.2.1 Software Requirements 77
3.4.1.2.2.2 Design Decisions 77
3.4.1.3 User Stories 78
3.4.1.4 System Restriction 81
3.4.1.4.1 Security 81
3.4.1.4.2 Data Storage/Cache 81
3.4.1.4.3 Accessible API 81
3.4.1.5 Additional Requirements for System Construction 81

3.4.1.5.1 Process Requirements 81
3.4.1.5.1.1 Continuous Integration/Continuous Delivery (CI/CD) 81
3.4.1.5.1.2 System Development Framework 82
3.4.1.5.2 Technical Requirements 82
3.4.1.5.2.1 System Packaging 82
3.4.1.6 Additional Requirements for Documentation 82
3.4.1.6.1 System Documentation 82
3.4.2 Stretch goals 83
3.4.3 Delivery 83
3.5 Conclusion process documentation 84
4 Test documentation 85
4.1 Goal 85
4.2 Tools 85
4.3 Planning 86
4.4 Execution 86
4.4.1 Travis 88
4.4.2 Static code analysis 89
4.4.3 Vulnerability analysis 90
4.5 Test phases 91
4.6 ARXaaS 93
4.6.1 Unit testing 93
4.6.2 Integration testing 94
4.6.3 System testing 95
4.7 PyARXaaS 97
4.7.1 Unit testing 97
4.7.2 Integration testing 99
4.7.3 System testing 99
4.8 WebARXaaS 100
4.8.1 Data integrity 100
4.8.2 API Testing 102
4.9 Performance testing 102
4.9.1 Load test 103
4.9.2 Stress test 105
4.10 Acceptance testing 109
4.11 Conclusion test documentation 110
5 Product documentation 111
5.1 Introduction 111
5.2 ARXaaS 112
5.2.1 Short presentation 112
5.2.2 Release Pipeline 113
5.2.3 Technologies 115

5.2.3.1 Runtime 115
5.2.3.2 Building and packaging 116
5.2.3.2.1 Apache Maven 116
5.2.3.2.2 Docker 116
5.2.3.3 Libraries and Frameworks 116
5.2.3.3.1 Runtime 116
5.2.3.3.2 Development 117
5.2.4 Architecture 118
5.2.5 Endpoints 119
5.2.5.1 Index 119
5.2.5.2 Analyzation 119
5.2.5.3 Anonymization 120
5.2.5.4 Hierarchy 121
5.2.5.4.1 Redaction based hierarchy 121
5.2.5.4.2 Interval based hierarchy 122
5.2.5.4.3 Order based hierarchy 123
5.2.6 Functionality 125
5.2.6.1 Controller Layer 126
5.2.6.2 Service Layer 128
5.2.6.3 Domain Layer 129
5.2.7 Security 130
5.2.7.1 SSL Handshake 130
5.2.8 Monitoring 131
5.2.9 Logging 132
5.2.10 License 132
5.2.11 Error description 132
5.2.11.1 HTTP status codes 132
5.3 Client side introduction 134
5.4 PyARXaaS 134
5.4.1 Short presentation 135
5.4.2 Packaging and release 136
5.4.3 Release Pipeline 137
5.4.4 Technologies 139
5.4.5 Libraries 139
5.4.5.1 Runtime libraries 139
5.4.5.2 Development libraries 141
5.4.6 Functionality 142
5.4.6.1 Package components 142
5.4.6.1.1 Connecting to ARXaaS 142
5.4.6.1.2 Hierarchy Builders 145
5.4.6.1.3 Privacy Models 146
5.4.6.1.4 dataset class 148

5.4.6.1.5 Response objects 149
5.4.7 Security 152
5.4.8 Logging 152
5.4.9 License 152
5.5 WebARXaaS 153
5.5.1 Short presentation 153
5.5.2 Technologies 153
5.5.2.1 React 153
5.5.2.2 React-BootStrap 154
5.5.2.3 Papa Parse 154
5.5.3 Functionality 154
5.5.3.1 DataImport 154
5.5.3.2 Privacy model builder 155
5.5.3.3 Analyzation 155
5.5.3.4 Anonymization 155
5.5.4 Operations 156
5.5.5 License 156
5.6 Future development 157
5.6.1 ARXaaS 157
5.6.2 PyARXaaS 157
5.6.3 WebARXaaS 158
5.7 Product Documentation Conclusion 158

6 User manual 159
6.1 ARXaaS 159
6.1.1 Run ARXaaS 159
6.1.1.1 HTTP Configuration 159
6.1.1.1.1 Run ARXaaS from Docker image 159
6.1.1.1.2 Run ARXaaS from .jar 160
6.1.1.2 HTTPS Configuration 160
6.1.1.2.1 Recommended: Run ARXaaS with dynamic HTTPS parameters 160
6.1.1.2.2 Generating and correctly configuring a keystore for an ARXaaS project 160
6.1.1.2.3 Compile and run ARXaaS with pre-defined, non-dynamic SSL configuration 161
6.1.1.2.4 Running the server with dynamic HTTPS configuration for static HTTPS keystore/certificate(s). 162
6.2 PyARXaaS client 163
6.2.1 Introduction 163
6.2.2 Installing PyARXaaS Client 163
6.2.2.1 Pip install 164
6.2.2.2 Setup virtual environment 164
6.2.2.2.1 Mac/Linux 164

6.2.2.2.2 Windows 164
6.2.3 Quick start guide 165
6.2.3.1 Analyze the risk of a dataset 165
6.2.3.2 Anonymize a dataset 167
6.2.4 Connecting to and using ARXaaS 169
6.2.4.1 Creating an instance 169
6.2.4.2 Risk Profile 169
6.2.4.3 Anonymization 169
6.2.4.4 Hierarchy Generation 170
6.2.5 Using the dataset class 170
6.2.6 Construction 170
6.2.6.1 Dataset type conversion 171
6.2.6.2 Mutation 171
6.2.6.2.1 Attribute type 171
6.2.6.2.2 Hierarchies 171
6.2.7 Creating Hierarchies 172
6.2.7.1 Hierarchy Building 172
6.2.7.2 Redaction based hierarchies 172
6.2.7.2.1 Creating a simple redaction hierarchy 173
6.2.7.3 Interval based hierarchies 174
6.2.7.4 Order based hierarchy 175
6.3 WebARXaaS client 176
6.3.1 Starting the application 176
6.3.2 Deploying to production 176
6.3.3 Configuration 177
6.3.4 Analyzing 177
6.3.5 Anonymizing 179
6.3.6 Privacy model input field description 183
6.3.6.1 K-Anonymity 183
6.3.6.2 L-Diversity 183
6.3.6.2.1 Using non-recursive variants of L-diversity 183
6.3.6.2.2 Using Recursive variant of L-diversity 184
6.3.6.3 T-Closeness 185
7 Appendix 186
7.1 Terminology 186
7.2 Product Specification 190
7.3 Testimonial from NAV IT (Norwegian) 207
7.4 Analyzation HTTP JSON request body 208
7.5 Analyzation HTTP JSON response body 208
7.6 Anonymization HTTP JSON request body 211
7.7 Anonymization HTTP JSON response body 212

7.8 Redaction based hierarchy HTTP JSON request body 215
7.9 Redaction based hierarchy HTTP JSON response body 216
7.10 Interval based hierarchy HTTP JSON request body 216
7.11 Interval based hierarchy HTTP JSON response body 217
7.12 Order based hierarchy HTTP JSON request body 218
7.13 Order based hierarchy HTTP JSON response body 218

List Of Figures

- Figure 1 - System diagram
- Figure 2 - ARXaaS architecture layer diagram (DDD)
- Figure 3 - PyARXaaS package components overview
- Figure 4 - Example of PyARXaaS usage in Jupyter notebook
- Figure 5 - of PyARXaaS PyPI page
- Figure 6 - Screenshot of WebARXaaS application use
- Figure 7 - a ARXaaS CI/CD pipeline job on Travis CI
- Figure 8 - of the team completing a Sprint Review with the Product Owner, Robindra
- Figure 9 - of a sprint review with NAV IT stakeholders
- Figure 10 - of a SCRUM retrospective board from a team retrospective meeting
- Figure 11 - of a team Sprint Retrospective at NAV IT's offices
- Figure 12 - Diagram of a CI/CD pipeline
- Figure 13 - ARXaaS pull request waiting to be merged
- Figure 14 - Overview of ARXaaS CI/CD pipeline
- Figure 15 - showing the process of test driven development
- Figure 16 - Screen grab from a Github pull request waiting for review
- Figure 17 - Screen grab from a Github pull request that has been reviewed and accepted
- Figure 18 - Example figure of feature slicing
- Figure 19 - Example Asana Kanban board
- Figure 20 - Gantt Diagram
- Figure 21 - Image of Accelerate book
- Figure 22 - Sprint retro Start, Stop, and Continue
- Figure 23 - Sprint retro
- Figure 24 - System diagram
- Figure 25 - Checking the branch before being approved to merge
- Figure 26 - All checks passed
- Figure 27 - Failed Travis build
- Figure 28 - Passed Travis build
- Figure 29 - Travis script running Code Climate test reporter
- Figure 30 - Jacoco include/exclude configuration
- Figure 31 - Code Climate dashboard
- Figure 32 - Snyk dashboard
- Figure 33 - All unit test passed
- Figure 34 - All integration test passed
- Figure 35 - All system test passed
- Figure 36 - Risk profile form WebARXaaS
- Figure 37 - Risk profile form th ARX GUI
- Figure 38 - Suppression limit parameter
- Figure 39 - Response time for a dataset of given rows x columns during ARXaaS stress test
- Figure 40 - of response time for a dataset of size in MiB during ARXaaS stress test

- Figure 41 - ARXaaS tools and libraries diagram
- Figure 42 - Diagram showing the ARXaaS CI/CD pipeline
- Figure 43 - Screengrab from ARXaaS API documentation
- Figure 44 - Screengrab of ARXaaS on Maven Central
- Figure 45 - ARXaaS architecture overview
- Figure 46 - UML diagram of ARXaaS
- Figure 47 - ARXaaS controller layer class diagram
- Figure 48 - ARXaaS service layer class diagram
- Figure 49 - Screen grab of ARXaaS metrics dashboard when running on NAIS
- Figure 50 - Error response contents
- Figure 51 - Unable to anonymize error message
- Figure 52 - Invalid attribute type error message
- Figure 53 - Failed to create dataset error message
- Figure 54 - diagram showing abstract usage of PyARXaaS
- Figure 55 - PyARXaaS version control host repository
- Figure 56 - showing the PyARXaaS PyPI page.
- Figure 57 - Diagram showing the PyARXaaS CI/CD pipeline
- Figure 58 - PyARXaaS pull request view on Github
- Figure 59 - of the documentation badge
- Figure 60 - Class diagram for PyARXaaS
- Figure 61 - Class diagram for ARXaaS abstraction classes
- Figure 62 - Hierarchy Builder class diagram
- Figure 63 - Privacy models class diagram
- Figure 64 - dataset class diagram
- Figure 65 - ARXaaS response objects class diagram
- Figure 66 - Sequence diagram of WebARXaaS
- Figure 67 - Importing of dataset on WebARXaaS
- Figure 68 - dataset headers generated
- Figure 69 - WebARXaaS privacy model section
- Figure 70 - WebARXaaS download button for anonymized dataset
- Figure 71 - Image of PyARXaaS documentation page using Sphinx
- Figure 72 - Importing of dataset to analyze
- Figure 73 - Setting attribute type of dataset to analyze
- Figure 74 - Analyze the dataset
- Figure 75 - Importing of dataset to anonymize
- Figure 76 - Setting attribute type and importing transformation model
- Figure 77 - Setting the privacy model to anonymize the dataset
- Figure 78 - Setting the suppression limit for the dataset
- Figure 79 - Anonymizing the dataset
- Figure 80 - K- Anonymity input field
- Figure 81 - Input fields for Distinct, Grass-Berger- and Shannon-Entropy
- Figure 82 - Input fields for Recursive L-Diversity
- Figure 83 - Input fields for Order and Equal Distance T-Closeness
- Figure 84 - Testimony from NAV IT

1 Presentation

Introduces the project, its background, and the *project stakeholders* involved. The document provides gives an overview of the solution for readers not interested in reading the more detailed product documentation. It has been written with the expectation that this chapter is read before any other chapter. This chapter also includes the project's conclusion.

1.1 Introduction

Organizations worldwide store large amounts of user information, and with the current trend of digitalization, they are likely to keep storing more. They recognize the potential value in analyzing the stored information, but they are not always free to do so. Data privacy laws, while incredibly important for privacy protection, places significant limitations on information access. In this project we present a solution to how one can control the detriment to privacy when attempting to utilize sensitive data as a resource.

Anonymization as a Service is a bachelor thesis project completed by a group of students from OsloMet in cooperation with NAV IT. The goal of this project is to provide an integration of ARX[1]'s state-of-the-art anonymization functionality into NAV IT's *data-driven* development strategy. While quite useful, ARX's interface is demanding, and can be further streamlined. The solution aims to solve this problem by wrapping ARX's functionality in a web API application that offers ARX's functionality as a web service.

The planning for this bachelor project was initiated in June 2018. The development of the solution was officially started in January 2019 and lasted until May 2019. After the development completion , the team worked on this report until the deadline on May 23rd, 2019. The solution was deployed to NAV IT's internal platform on the 15th of April and has been continuously deployed to afterwards, in accordance with *continuous integration* and *continuous development* (*CI/CD)* practices.

The team has been working in Oslo, at OsloMet university and NAV headquarters in Sannergata 2, four days a week with fixed core work hours nine to four. After many hours spent on this project we are proud of the results. We have developed a solution, delivered it, and distributed it as an open source project. The team has also delivered on the project's stretch goals as described in chapter 3.4.2 Stretch goals.

The project could not have been completed without the excellent support from our stakeholders at NAV IT Robindra Prabhu, Paul Bencze, Gøran Berntsen and Erik Vattekar. The team also wishes to thank NAV IT for trusting us with this important and challenging assignment and for providing us with an excellent work environment. We also appreciate the opportunities granted to present the solution. This includes both presentations for NAV IT internally and other public sector agencies such as Agency for Public Management and e-Government (Difi[2]) and the Norwegian tax administration(Skatteetaten[3]). The team also

[1]ARX De-Identifier homepage - https://arx.deidentifier.org
[2]Difi homepage - https://www.difi.no/

wishes to acknowledge and thank the ARX team for developing a powerful anonymization library and distributing it *open source*. This project was made possible as a result of their magnificent work.

Finally, the team would like to thank the Bachelor project supervisor Eva Hadler Vihovde for guiding the team continuously throughout the project.

1.2 Presentation of the group

Sondre - IT, has worked part-time as a data engineer for NAV IT AI-lab since fall 2018. Has helped keep the team connected to different project stakeholders in NAV IT. Main competence is Java and Python development, but has experience in web development from previous work.

André - has a bachelor's degree in machine engineering. Currently finishing a bachelor's degree in software engineering. Worked as an intern at Aker Solutions prior to studying computer engineering. Main competence in Java and MVC architecture.

Julian - IT, Two years of work experience from Basefarm. Besides his studies, he has worked part-time for Medarbeiderne AS doing further development on their CRM system. Spent 6 months as an exchange student, while studying abroad in San Francisco. Main competence is Java, Javascript and web development.

Viktor - a programmer with passion for web development and understanding logical systems. Aspires to never stop absorbing knowledge and to eventually become a renowned full-stack developer. As of the publication of this report, he is especially fond of programming in the MVC architecture, utilizing either Java or C# in the Spring Boot and .Net frameworks respectively.

Jeremiah - has a bachelor's degree in machine engineering. Currently studying for a bachelor's degree in software engineering. Worked as an intern for OXX from January to May 2019. Main competence in Java, javascript, C#, MySql, and MVC architecture.

[3]Norwegian Tax Administration homepage - https://www.skatteetaten.no/

1.3 Presentation of the client

The Norwegian Labour and Welfare Administration (NAV) is the backbone of the Norwegian welfare state, administering a third of the national budget and servicing almost 2.8 million people. Responsibilities include, but are not limited to, unemployment benefit, work assessment allowance, sickness benefit, pension, child benefit, and cash-for-care benefit. In addition, NAV manages and retains stewardship of several important data sources containing information on its users and the services it provides. NAV IT is currently in the midst of a digital transformation, undergoing significant changes in team organization, work processes and harnessing new technologies. Its utilization of data in the development of new and improved services faces limitations from strict data privacy practices.

Data and Insight is a department within NAV IT. NAV IT Data and Insight delivers systems and operations purposed for data storage, data processing, data access and analytics.
Our client for this assignment is NAV IT Data and Insight, and will be hereby referred to as the client unless otherwise is specified.

1.3.1 Project stakeholders at NAV IT

Person of Contact

Name	**Role**	**Email**
Gøran Berntsen	Tech Lead - Open Data	Goran.Berntsen@nav.no

Product Owner

More about the product owner role can be read about in chapter 3.2.1.1 Agile work process.

Robindra Prabhu	Data Scientist - AI Lab	Robindra.Prabhu@nav.no

Other project stakeholders

Paul Bencze	Tech Lead - AI Lab	Paul.Bencze@nav.no
Erik Vattekar	Data Engineer - AI Lab	Erik.Vattekar@nav.no

1.4 Project background

The terms data and dataset are used continuously throughout the documentation. Unless anything else is specified, the term refers to tabular data/datasets containing population data, where one row corresponds to a record.

Within NAV IT Data and Insight, there is a team called AI Lab. One of its tasks is to function as an internal source of knowledge on machine learning and data science. Data anonymization, also referred to as de-identification and risk assessment, is one of the areas relevant to the AI Lab. This is a worldwide field involving dataset manipulation and analyzation.

The objective of data anonymization is to create processes of controlled data disposal.
AI Lab is currently utilizing both internal and external tools for data anonymization. One such tool is ARX[4], which is widely regarded as top-class anonymization software because of its utilization of well-established models and algorithms.

Data science today is typically conducted within programming languages like Python and R. The data scientists develop scripts, notebooks, and larger programs for extracting and analyzing data. The early stages of these tasks typically involve data cleaning and data transformation, which is often conducted with the help of ARX. However, ARX does not currently integrate seamlessly into the typical Python/R-based data science workflow. While ARX provides useful data anonymization functionality for an experienced user, there is a significant time investment involved in both learning how to use, and using the application. Perhaps the most significant is the latter, which if solved, could contribute to sparing AI Lab's data scientists for unnecessary work hours.

As a result, the AI-lab has requested the team to:

- Develop a solution that grants access to ARX functionality from modern data science toolsets and workflows

- Provide an extendable framework for making ARX's state-of-the-art anonymization methods accessible to a broader user group in NAV IT, by lowering barriers of use such as user requirements.

[4] Official ARX paper/article - https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4419984/

1.4.1 Introduction to de-identification

Following is a short description of de-identification. For more in-depth explanations see chapter 2 on de-identification.

De-Identification describes anonymization operations that serve to reduce the *re-identification risk* in a dataset containing *personally identifiable information,* otherwise known as *PII*. Transforming datasets containing *PII* into aggregates is a common practice within de-identification, but this often results in undesirably large amounts of information loss. This loss in information obstructs other researches from verifying the aggregate or doing additional research on the data. By transforming the data so as to reduce the *re-identification risk*, de-identification serves to provide legal opportunities for sharing and utilizing datasets containing *PII.*

ARX is is an *open source* (Apache License, Version 2.0) anonymization software tool. Programmed in Java, ARX offers its anonymization functionality through its library of state-of-the-art anonymization models and algorithms. ARX is utilized by medical researchers worldwide for anonymization of patient datasets prior to sharing them with the research community.

As of the time that this report is written, ARX is being continuously updated and improved by the ARX team. This has placed an important requisite on the solution in regards to updateability, which is elaborated on in chapter 5.6 on Future Development.

De-identification is further elaborated on in chapter 2 De-identification, while more information on ARX can be found in ARX's official papers, available in footnote 4.

1.4.2 Current situation

The majority of literature, practices, and frameworks in the field of anonymization are targeted on the distribution of sensitive data between researchers. The client's use cases for anonymization are similar, but differ slightly.

NAV IT main data de-identification use cases

- Risk assessment to determine the legal status of a dataset
- De-identification to ease internal sharing or processing of data
- Risk assessment and de-identification for external sharing of data

All processing of datasets containing *PII* is subject to privacy laws and regulations. Whenever the client finds a new use for *PII* data, it has to go through a Data Protection Impact Assessment or DPIA[5]. The process of de-identifying a dataset involves the processing of *PII* and is hence subject to the same legal restrictions. However, once sufficiently de-identified, the dataset will no longer be considered personal information, and consequently, it will no longer be bound by the initial level of strict legal compliance. As a result, sufficiently de-identified datasets may provide leeway for data scientists and

[5] e-Government DPIA - https://www.datatilsynet.no/regelverk-og-verktoy/veiledere/vurdering-av-personvernkonsekvenser/

developers in the utilization of internal data sources. Determining the proper legal status of a dataset depends on the dataset's *re-identification risk*. This project also aims to improve access to tooling that provides efficient methods and tools for determining that risk, while mainly aiming to provide increased availability to state-of-the-art anonymization functionality. Read more about the de-identifying process in chapter 2 De-identification.

The *data-driven*[6] revolution is becoming widely adopted all around the world. The core of the *data-driven* enterprise paradigm is the sharing of data internally in the enterprise. Easing access to anonymization would put NAV IT in a better position to become more *data-driven* in their pursuit to offer better services to their users.

Tools in the anonymization field mainly fall into one of the two following categories.

- Proprietary solutions that work as a 'walled garden' with limited integration towards other tooling
- *Open source* projects

Common to both categories is the fact that most solutions are mainly focused on Graphical User Interfaces(GUI). Such solutions are by their very nature limited in support for integration and further development.

1.5 The solution - Anonymization as a Service

This chapter describes the solution developed. For a more detailed description of the development process and its phases, please read chapter 3 Process documentation. For a more detailed description of the product, please read chapter 5 Product documentation.

1.5.1 Solution description

Anonymization as a Service provides access to anonymization tools for data scientists at NAV IT. A data scientist is able to analyze and anonymize tabular datasets based on user-provided configurations. The user can configure privacy models, attribute types, and transformation models known as hierarchies to use in the anonymization. See the 2.6 Generalization Hierarchies section in the De-Identification chapter for elaboration on hierarchies and their purpose.

The Anonymization as a Service ecosystem
Service
- ARXaaS

Clients
- PyARXaaS
- WebARXaaS (stretch goal)

Anonymization as a Service refers to the products that make up the Anonymization as a Service ecosystem. The centerpiece of this ecosystem is the microservice ARXaaS or "ARX as a Service". The core idea behind ARXaaS' microservice design is to create an

[6] Meaning of a Data-Driven Enterprise - https://dzone.com/articles/what-it-means-to-be-a-data-driven-enterprise-and-h

anonymization platform that offers clients seamless integration with state-of-the-art anonymization functionality.

Two ‘client’ products have been developed, named PyARXaaS and WebARXaaS. The purpose of PyARXaaS and WebARXaaS is to provide user interfaces to ARXaaS. PyARXaaS is a Python package meant for data scientists and WebARXaaS is a web application for a wider user group.

The main focus throughout the project was the development of ARXaaS and PyARXaaS, while WebARXaaS was a stretch goal of the project. This is reflected in the maturity of the products.

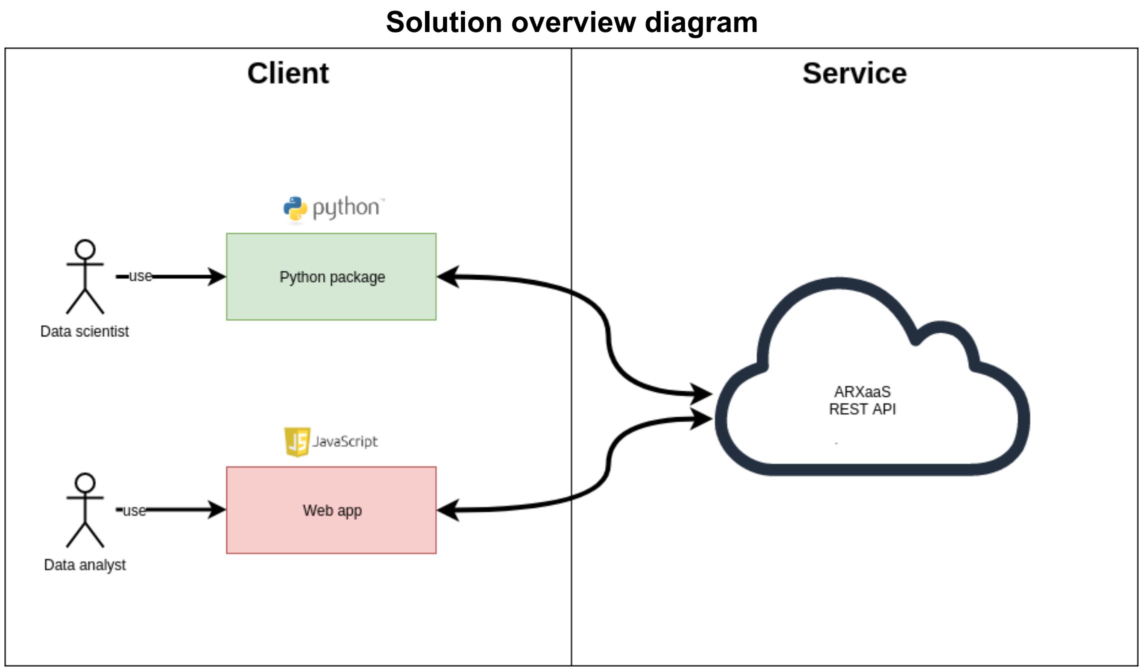


Figure 1 - System diagram

1.5.1.1 ARXaaS - The service

ARXaaS is named after ARX because the team felt it was appropriate to show attribution to the great work of the ARX team. ARXaaS is a web service developed using the Java Spring Boot framework. It utilizes the ARX[7] anonymization library to implement functionality in three key areas.

- Dataset risk analysis
- Dataset anonymization
- Generation of generalization hierarchies (used in anonymization)

The service implements *HTTP* endpoints to expose the functionality for client applications. The service endpoints follow a *RESTful design* and are documented extensively. The API

[7] ARX homepage - https://arx.deidentifier.org/

documentation is automatically generated and published to the project documentation page[8] on every new release. The service is packaged and distributed as a Docker container. See chapter 5.2 ARXaaS in the product documentation for elaboration on the technical implementation.

It is important to note that hierarchy generation was not part of the original product specification. The product owner did not believe the team capable of implementing it within the deadline. The team saw this as a challenge, and by mid-April, hierarchy generation with the exception of date hierarchies were implemented along with test coverage and documentation. For details on hierarchies see 5.2.5.3 Hierarchy section in the product documentation.

NAV IT has its own internal application platform named NAIS[9]. This platform handles encryption and security for all services deployed to it. With ARXaaS having use cases outside of NAV IT, the service's security is not sufficient with deployment to NAIS. As such, the team was also required by the client to implement stand-alone HTTPS support. Guide to setup the service with HTTPS can be read in chapter 6.1.1.2 HTTPS Configuration and on the ARXaaS Github page[10]. Elaboration on HTTPS implementation can be read in Product documentation chapter 5.1 ARXaaS.

The ARXaaS architecture has been inspired by *Domain Driven Design*(DDD). The different parts of the system are divided into separate layers according to responsibility. See chapter 5.2.4 Architecture in the product documentation for more details.

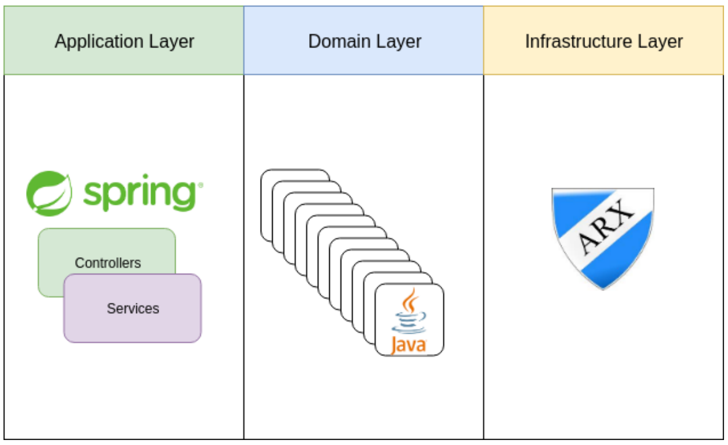


Figure 2 - ARXaaS architecture layer diagram (DDD)

Testing

To ensure the quality of the solution, the team has utilized various types of testing. As a domain concerning sensitive data of individuals, the quality of the product is of paramount importance. In order to ensure that the service can be entrusted with the sensitive data, extensive and robust testing of the product is necessary. ARXaaS currently(May 2019) has 93% test coverage.

[8] ARXaaS API documentation page - https://oslomet-arx-as-a-service.github.io/ARXaaS/
[9] NAIS - https://nais.io/
[10] ARXaas HTTPS guide - https://github.com/oslomet-arx-as-a-service/ARXaaS/blob/master/READMEHTTPS.md

Testing of ARXaaS can be divided into the following categories:

- **Testing of implementation correctness**
  Unit, integration, system and acceptance testing
- **ARX parity testing**
  Tests that verify that the output from the service is identical to the consuming client's output.
- **Performance testing**
  ARXaaS was load tested on the NAIS platform test environment. This contributed to fine tuning of the operational parameters for the service, such as RAM and CPU power needed for running the service.

For in-depth information, see chapter 4 Testing documentation.

1.5.1.2 Clients

The project team has built two clients for consumption of the ARXaaS service. One of them is a Python package purposed for the data scientist user group. The other one is a web app with a simplified user interface targeting a wider user group.

1.5.1.2.1 PyARXaaS - the wrapper package

PyARXaaS is a python wrapper package developed for NAV IT's data scientists. The product owner's main requirement demanded that anonymization functionality was to be made available in Python. A common use case would be in a workflow where the data scientist is manipulating a dataset, and requires dynamic analysis of the data anonymity metrics. Another use case could involve integrating the system in a data pipeline to provide data analytics and anonymization capabilities.

To fulfill its purpose, the package provides an easy to use Python *API* and makes *HTTP* calls to ARXaaS. See chapter 5.4 PyARXaaS product documentation, for a detailed description of the package, technologies used, package components, release pipeline, security and logging.

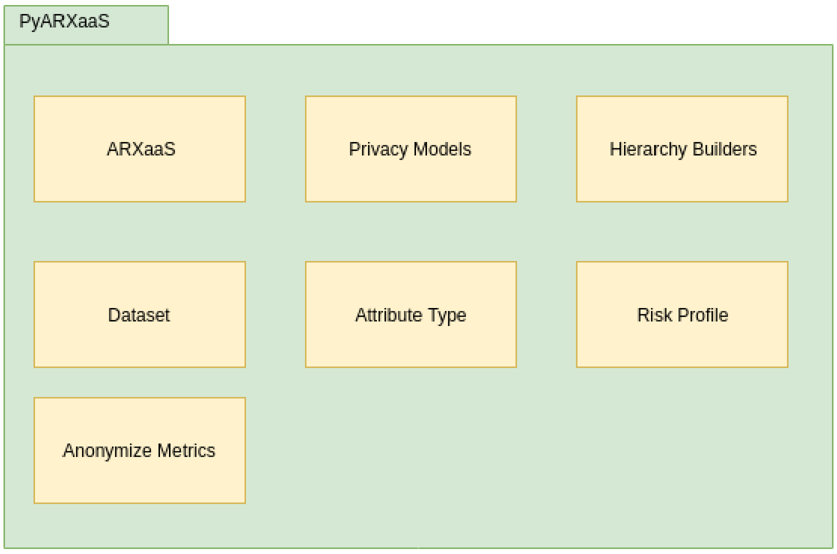
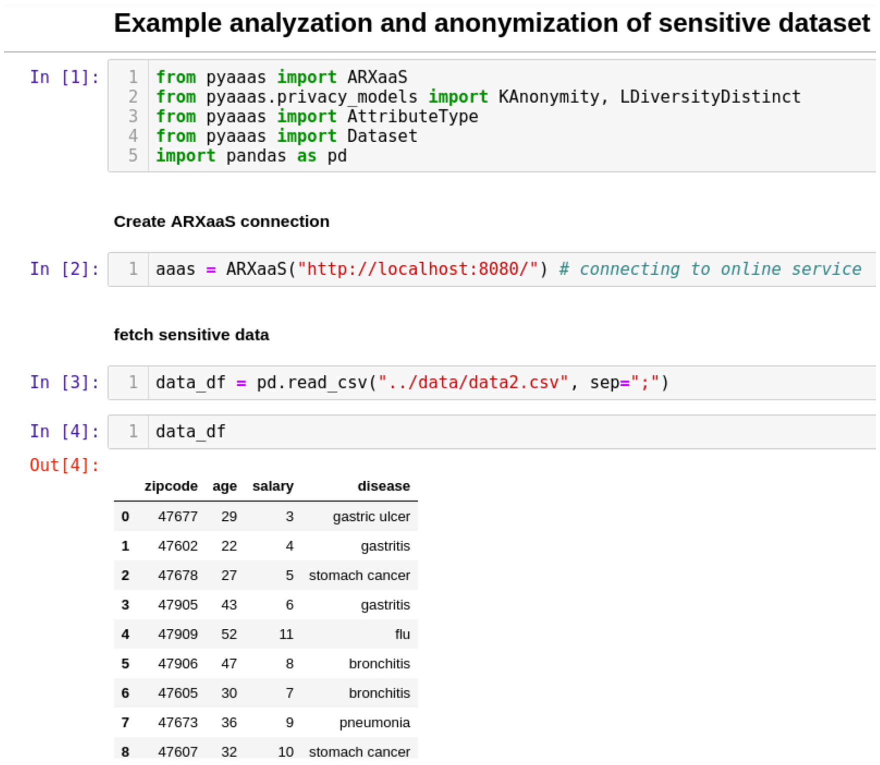


Figure 3 - PyARXaaS package components overview

Data scientists can use PyARXaaS to program Python scripts for re-identification analysis and anonymization of datasets. PyARXaaS's anonymization functionality is elaborated on in chapter 2 De-identification. Read about using PyARXaaS in 6.2 PyARXaaS user manual.

Jupyter notebook[11] is the preferred editor tool of NAV IT data scientists for Python programming. Jupyter notebook provides an interactive web editor for writing and executing Python code asynchronously in cells that can be interspersed with text cells for explanation. The Jupyter notebook environment has influenced the development of PyARXaaS specifically design decisions such as logging and feedback to the user. It is of course also possible to use PyARXaaS as a regular package in a Python script or application.

[11] The Jupyter Notebook Introduction - https://jupyter-notebook.readthedocs.io/en/latest/notebook.html



Example analyzation and anonymization of sensitive dataset

```
In [1]:  from pyaaas import ARXaaS
         from pyaaas.privacy_models import KAnonymity, LDiversityDistinct
         from pyaaas import AttributeType
         from pyaaas import Dataset
         import pandas as pd
```

Create ARXaaS connection

```
In [2]:  aaas = ARXaaS("http://localhost:8080/") # connecting to online service
```

fetch sensitive data

```
In [3]:  data_df = pd.read_csv("../data/data2.csv", sep=";")

In [4]:  data_df
Out[4]:
```

	zipcode	age	salary	disease
0	47677	29	3	gastric ulcer
1	47602	22	4	gastritis
2	47678	27	5	stomach cancer
3	47905	43	6	gastritis
4	47909	52	11	flu
5	47906	47	8	bronchitis
6	47605	30	7	bronchitis
7	47673	36	9	pneumonia
8	47607	32	10	stomach cancer

Figure 4 - Example of PyARXaaS usage in Jupyter notebook

PyARXaaS is continuously being published to the Python Package Index(PyPI)[12] as new versions are developed. This ensures constant availability of the package's latest version for the user group. Also, PyARXaaS user documentation[13] is built and published alongside every new version, which is essential for guiding the users of the product. See chapter 5.4.3 Release Pipeline for more in-depth description of the release process for PyARXaaS.

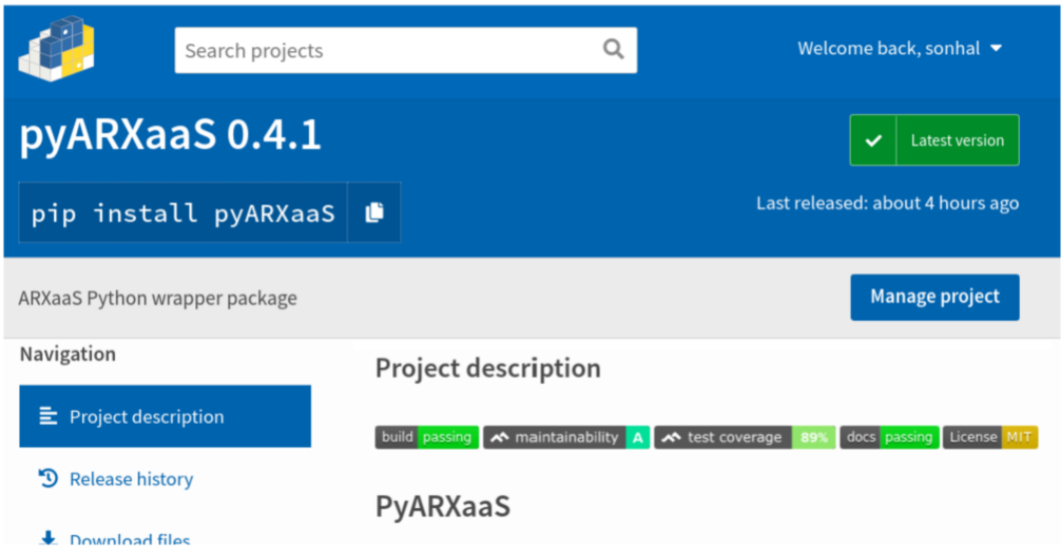
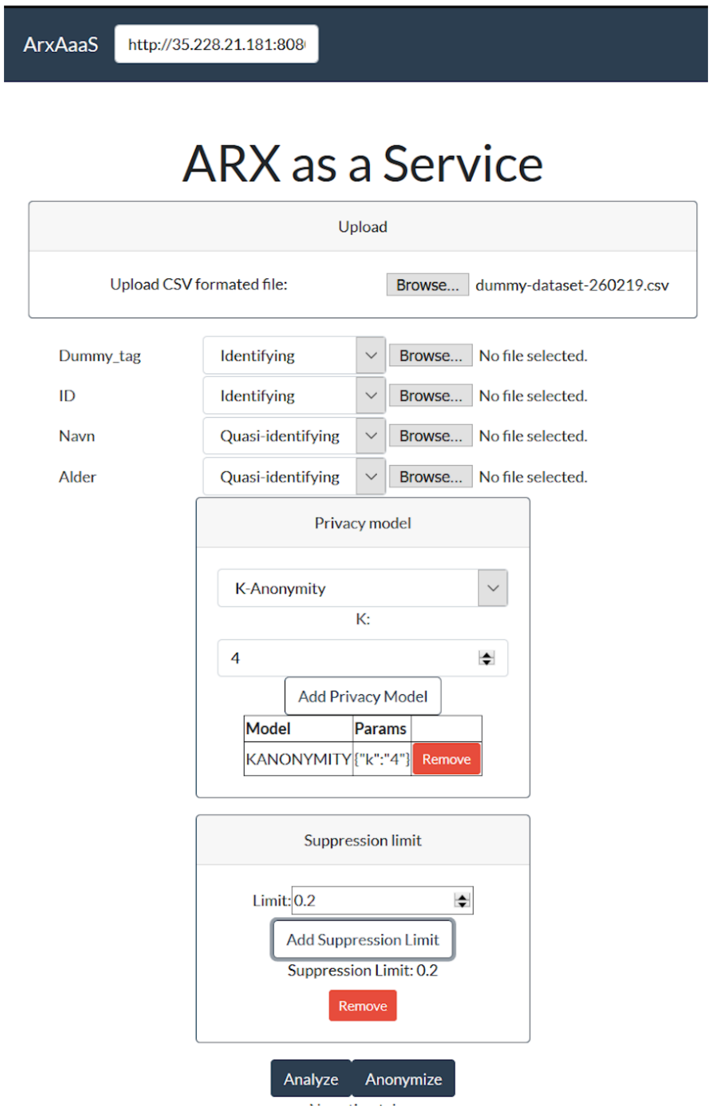


Figure 5 - of PyARXaaS PyPI page

[12] PyARXaaS PyPI page https://pypi.org/project/pyARXaaS/
[13] PyARXaaS user documentation - https://pyaaas.readthedocs.io/en/latest/notebooks/notebooks.html

1.5.1.2.2 WebARXaaS - web application (stretch goal)

The product owner made a request for a product that could offer access to ARXaaS's anonymization functionality for a wider user group. This request was a result of a need for streamlined access to data analysis and anonymization functionality without the requisite of Python programming knowledge. To fulfill this request, the team created the WebARXaaS product. WebARXaaS is a React web app that offers integration with ARXaaS through a user friendly GUI experience from any web browser. Since it was a stretch goal, work on the solution did not start until mid April. As such, efforts on WebARXaaS have been focused on implementing functionality at the expense of interface design.



ARX as a Service

http://35.228.21.181:808

Upload CSV formated file: dummy-dataset-260219.csv

Dummy_tag	Identifying	No file selected.
ID	Identifying	No file selected.
Navn	Quasi-identifying	No file selected.
Alder	Quasi-identifying	No file selected.

K-Anonymity

K: 4

Model	Params	
KANONYMITY	{"k":"4"}	Remove

Limit: 0.2

Suppression Limit: 0.2

Figure 6 - Screenshot of WebARXaaS application use

1.5.2 Deployment

NAV IT requested in the preliminary meetings with the team, that continuous integration and preferably continuous delivery was to be complied to and fulfilled throughout the project. See chapter 3.2.1.2 Continuous integration and Continuous delivery, for the concepts and the team usage in even closer detail.

The client wished for ARXaaS to be deployed to it's internal container orchestration platform *NAIS*[14]. This was part of the motivation for creating a complete *CI/CD pipeline* utilizing Travis CI[15] for every product of the solution. Having a mature release pipeline made it easier to get permission for deploying to NAIS, as the team could refer to the tests and vulnerability scans being done on every product build.

During the first few sprints, the team set up a *Kubernetes* cluster on *Google Cloud* to continuously deploy to, until the product was stabilized. Deployment to NAIS happened after the product got stabilized in the mid of April. The team feels strongly that the *CI/CD* pipeline has been instrumental to the project success.

When a commit is to be merged to the products master branch[16], the commit goes through the following steps before being accepted. If there are any failures the deployment is stopped, and the team is notified so they can correct the error.

- **Test**
  The full test suite with unit tests, integration tests and system tests.

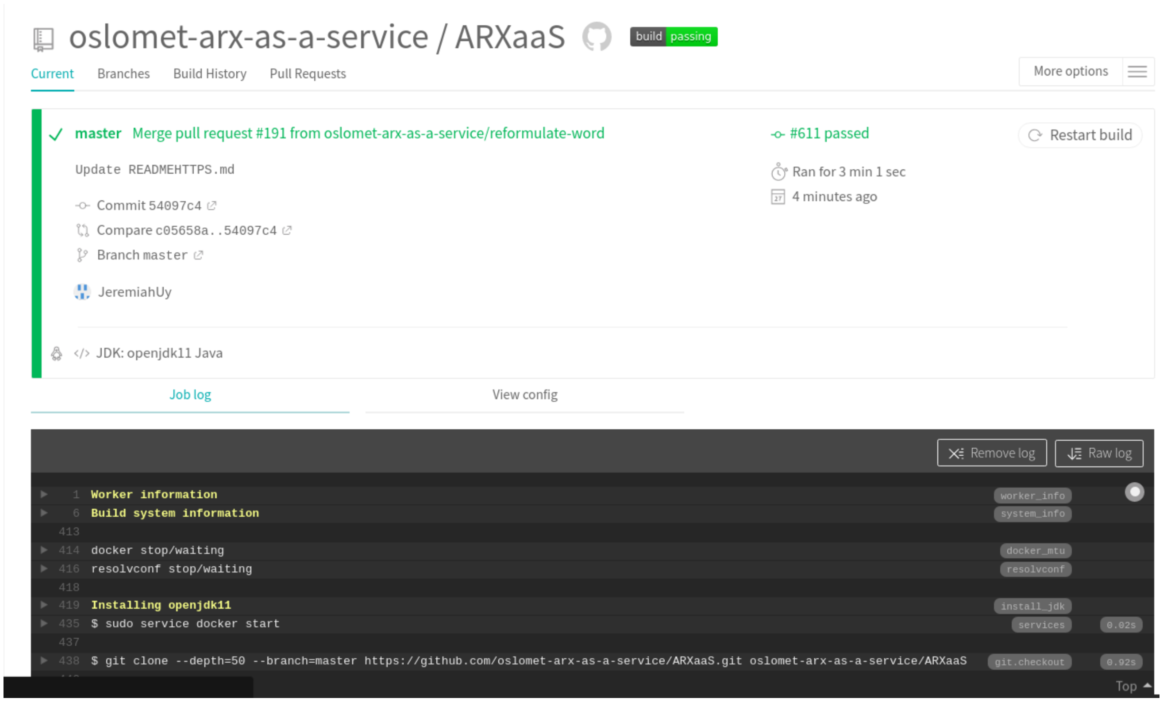
- **Documentation**
  API documentation for e.g *HTTP* endpoints are generated. This documentation is a combination of manually written text and auto-generated snippets of information about the various endpoints, accepted *HTTP* verbs, request and response body JSON structure.

- **Packaging**
  The product in question is packaged in different artifacts with different formats for the deployment and artifact hosting platforms. Main artifacts being Java JAR and Docker image for ARXaaS and bdist wheels[17] for PyARXaaS.

- **Publishing**
  The packaged artifacts such as documentation, executables and source code are published to different platforms such as Docker hub, Maven Central and PyPI.

- **Deployment (ARXaaS only)**
  After a new deployment has been completed, the team's internal *Kubernetes* cluster and the client's NAIS platform deploys the new ARXaaS version.

[14] NAIS homepage - https://nais.io/
[15] Travis CI homepage - https://travis-ci.com/
[16] Git branches - https://git-scm.com/book/en/v1/Git-Branching-What-a-Branch-Is
[17] Bdist wheels - https://pythonwheels.com/


oslomet-arx-as-a-service / ARXaaS

master Merge pull request #191 from oslomet-arx-as-a-service/reformulate-word

Update READMEHTTPS.md

Commit 54097c4

Compare c05658a..54097c4

Branch master

JeremiahUy

#611 passed

Ran for 3 min 1 sec

4 minutes ago

JDK: openjdk11 Java

```
Worker information                                                                                      worker_info
Build system information                                                                                system_info

docker stop/waiting                                                                                     docker_mtu
resolvconf stop/waiting                                                                                 resolvconf

Installing openjdk11                                                                                    install_jdk
$ sudo service docker start                                                                             services      0.02s

$ git clone --depth=50 --branch=master https://github.com/oslomet-arx-as-a-service/ARXaaS.git oslomet-arx-as-a-service/ARXaaS   git.checkout   0.92s
```

Figure 7 - a successful ARXaaS CI/CD pipeline job on Travis CI

Having a complete *CI/CD* pipeline setup from day one meant that the team where continuously gathering feedback on the products. Also, the client could use the features developed as soon as they were merged into the master branch.

1.5.3 License

All of the products developed for this project are distributed under the MIT License. This is because NAV IT, the client, is a government agency following a policy where all software that can be *open source* should be *open source*. The MIT license is the default license used by *open source* projects at NAV IT.

- ARXaaS License[18]
- PyARXaaS License[19]
- WebARXaaS License[20]

1.6 Future Development

Data anonymization is a rapidly moving problem, and the technical aspect is only a part of the bigger picture. The team is scheduled to participate in several meetings with NAV IT development teams and leadership, to work out how to integrate the solution best into the organization. Plans for future development, stewardship of the project, and legal framework are being worked out as this report is being written(May 2019). The work the team has delivered is viewed as an important piece to the further development of NAV IT, as a *data driven* organization. The team feels strongly that the combination of focus on software quality and cooperation, has resulted in a solution that NAV IT has taken seriously. This trust has resulted in summer internship contracts for all the team members. During the summer

[18] ARXaaS License - https://github.com/oslomet-arx-as-a-service/ARXaaS/blob/master/LICENSE
[19] PyARXaaS License - https://github.com/oslomet-arx-as-a-service/PyARXaaS/blob/master/LICENSE
[20]WebARXaaS License - https://github.com/oslomet-arx-as-a-service/WebARXaaS/blob/master/LICENSE

internship the team will continue the work on the solution. The decision to *open source* the project and open up for outside involvement and usage means the project has the legs to take on a life outside the walls of NAV IT. Other actors such as DiFi, DNB, The Norwegian Tax Administration(Skatteetaten) and the ARX development team in Munich, has showed interest in the project, and future collaboration is being discussed.

1.7 Conclusion

The Anonymization as a Service project has been successful in delivering on the client's requirements. The solution, consisting of three separate software products, is built with a high standard of quality. The team agrees that this is a result of following several acknowledged practices. The project's consistent fulfillment of *continuous delivery* and compliance to the *agile* guidelines of *Scrum* has been instrumental to the team's success.

Mona has been tasked with a analytics product that requires processing of personal data

Figure 8 - of the team completing a Sprint Review with the Product Owner, Robindra

1.7.1 Value delivered

The solution developed delivers value to the client by providing access to ARX's state-of-the-art anonymization functionality with integrations to the client's preferred data scientist tooling.

The solution is easily adoptable by other interested parties because of the *open source* licence. If adopted, the solution can potentially contribute to mitigate issues regarding data privacy and information leakage on a global scale.

Value for NAV IT
The solution is valuable for NAV IT as a whole, as it provides a well documented anonymization api to integrate other solutions against and build upon. The solution reduces friction and effort required in the client's anonymization process, effectively saving resources for the client. Providing better anonymization tooling also positively impacts the usage and quality of anonymization. See the testimonial in the 7.3 Appendix for the client's own statement on the impact of the work.

Value of this report
The development team has put a lot of time and effort into producing thorough documentation of the work process and the products developed. The user documentation offers detailed guidance for end users on setup and how to use the products. Any team aspiring to undertake similar projects are able to read about the work process of this project, especially the *continuous integration* aspects. Further, they can potentially adopt aspects to their own work process if they so desire.

Value for the team
During the development of Anonymization as a Service the team learned and utilized a variety of new technologies. We experienced professional product development in the domain of de-identification. A domain which puts a high premium on reliability and correctness. The whole team feels a strong sense of accomplishment from the project. We developed three separate technical products in synchronization with feedback from the *product owner*, and we have exceeded our client's expectation in both the solution's maturity and features implemented.

2 De-identification

This document is intended to present the problem domain this project aims to tackle. It is intended for readers without any knowledge of de-identification or anonymization. The team recommends reading this document before the product documentation, and user manual, as this document will provide the necessary context to the business domain of the solution.

2.1 Introduction

De-Identification, Anonymization and Pseudonymization are often confused and used with different intentions. In this document the team has the defined important de-identifications terms and concepts in accordance with the available literature on the subject, and with guidance from the product owner.

De-Identification is a common term for reducing the probability that a person can be identified from a microdata dataset. Microdata datasets are datasets that provide information on a set of variables for individuals. In practical terms, this means each row in the dataset represents an individual or entity. Summarizing microdata into aggregates is a common method to de-identify the data, but this result in high levels of information loss and hinders other researches from verifying the aggregate or do additional research.

"The aim of anonymizing microdata is to transform the datasets to achieve an "acceptable level" of disclosure risk. The level of acceptability of disclosure risk and the need for anonymization are usually at the discretion of the data producer and guided by legislation."

From Statistical Disclosure Control: A Practice Guide[21]

2.2 De-identification overview

De-identification is a critical component in research. It protects the privacy of individuals contained in a dataset by removing the connection to the individual. A side benefit of this is the legal ramifications, because once de-identified, a dataset is no longer considered *personally identifiable information* (*PII*). If a dataset does not include *PII,* GDPR does not apply or restrict its use[22].

De-identification invariably means removing or obscuring information in the original dataset. The goal when de-identifying is to remove enough data to be able to confidently classify a dataset as de-identified, whilst retaining a dataset that can be useful. This means that when de-identifying a dataset there is always a trade-off to be had between *data privacy* and *data utility*. How hard it is to identify an individual in the dataset (data protection) and how much value can be extracted from the dataset (data utility). These two interests are always at odds when de-identifying.

[21] Statistical Disclosure Control - https://buildmedia.readthedocs.org/media/pdf/sdcpractice/latest/sdcpractice.pdf
[22] Data privacy - https://gdpr.eu/data-privacy/

There are two main types of de-identification:

- Anonymization
- Pseudonymization

We will briefly talk about the pseudonymization before moving on to Anonymization which has been the focus of the teams work and the functionality implemented in the solution are anonymization features.

2.2.1 Pseudonymization

Pseudonymization is a type of de-identification where the association between data and person is removed. It achieves this by introducing a new bidirectional mapping between an individual in the dataset and his or her identifiers. Pseudonymization can be irreversible, in that case the mapping to the identifiers are deleted. Pseudonymization is regarded as a weak form of de-identification as the identifiers have only been moved to a separate mapping (making direct identification harder, but still possible through remaining variables in the dataset and background knowledge).

2.2.2 Anonymization

Anonymization intends to irreversibly reduce the linkage between an individual and identifying information in a dataset.

When anonymizing a dataset, attributes are categorized into the following categories:

Attribute Type	**Description**
Identifying	Attributes that are directly identifying. e.g. phone number, email, full name, bank account number
Quasi Identifying	Attributes that, while not directly identifying, could be identifying in combination with other information. e.g. birth date, zip code, workplace, GPS location data
Sensitive	Sensitive Attributes is information that could be damaging if released, about a person. e.g. voting, medical information, criminal record
Insensitive	These are all other attributes in the dataset. Data points that neither could identify, nor cause harm to an individual if released.

Additionally quasi identifying attributes are further categorized in the following categories:

Type	**Description**
Categorical	Categorical attributes are values in a finite set. e.g. gender, religion, home state.
Continuous	Continuous attributes are values in an infinite set. e.g. income, weight, height. Due to their intrinsic uniqueness, they may be transformed into categorical variables when anonymizing

2.3 Main anonymization techniques

Generalization	Reducing the precision of an attribute. e.g full date of birth becomes only month and year or home address becomes home state or even country.
Suppression	Replacing a value in a dataset with a null value. The name “Peter Peterson” becomes 'Peter *'
Micro-aggregation	Sets of numeric attribute values can be transformed into a common value by user-specified aggregation functions.
Subsampling	Releasing only a subset of the dataset instead of all the records

2.4 Risk assessment

Re-Identification is the reverse of de-identification and is the primary threat addressed by laws and regulation[23]. Quantifying the risk of re-identification associated with a dataset is of high importance. The key aspect for re-identification is the uniqueness of quasi identifying attributes and the uniqueness of the combinations of the attributes. Quasi attributes can be linked with additional data in the dataset or from external datasets to identify individuals.

[23] Privacy-enhancing ETL-processes - https://www.sciencedirect.com/science/article/pii/S1386505618307007

In the literature these are the three types of disclosures that are of importance[24]

Disclosure type	**Description**
Identity disclosure	Occurs if an attacker associates a known individual with a data record. e.g. If the attacker uses external information to in combination with a dataset to identify a person in the dataset.
Attribute disclosure	Occurs if an attacker is able to learn some new information about a person based on the information in the dataset. e.g. If a dataset show all males over 60 in Oslo has been on unemployment benefits and the attacker knows a specific male to fit this description.
Membership disclosure	Occurs if an attacker can determine if an individual is in a specific dataset without being able to identify the specific record.

Three different threat scenarios are commonly used by researchers[25]

- **Prosecutor attack model**
  Under this model the attacker is assumed to a target specific individual and know data concerning the individual is present in the dataset.

- **Journalist attack model**
  Under this model the attacker is assumed to target an arbitrary individual without knowing if the individual is present in the dataset. Regarded as a more realistic model than the prosecutor model.

- **Marketer attack model**
  Under this model the attacker is assumed to aim at re-identifying as many individuals as possible, without regard to false identifications. The risk of a successful attack can be expressed as the expected average number of re-identified individuals from the dataset.

[24] Statistical Disclosure Control: - https://buildmedia.readthedocs.org/media/pdf/sdcpractice/latest/sdcpractice.pdf
[25] Privacy-enhancing ETL-processes - https://www.sciencedirect.com/science/article/pii/S1386505618307007

2.5 Privacy Models

Privacy models also referred to as privacy criteria are used when anonymizing a dataset. In essence the models define mathematical criteria or properties the released dataset should satisfy. Anonymization tools such as ARX implement algorithms for transforming a raw dataset to conform to user-specified privacy models.

Many models have been implemented to prevent the different disclosure types. We will only introduce the most important here.

2.5.1 K-Anonymity

K-Anonymity ensures that the records for each person contained in the dataset cannot be distinguished from at least k-1 individuals whose records also appear in the dataset[26]. Groups of indistinguishable records are said to form an *equivalence class*.

2.5.2 L-Diversity

L-Diversity protects a dataset against attribute disclosure. It does this by ensuring that each sensitive attribute has at least "l" represented values in each *equivalence class*. Different variants exist which differ in how they measure diversity[27].

Types of L-Diversity[28]:

1. Distinct l-diversity – The simplest definition ensures that at least l distinct values for the sensitive field in each equivalence class exist.

2. Entropy l-diversity – The most complex definition defines Entropy of an equivalent class E to be the negation of summation of s across the domain of the sensitive attribute of p(E,s)log(p(E,s)) where p(E,s) is the fraction of records in E that have the sensitive value s. A table has entropy l-diversity when for every equivalent class E, Entropy(E) ≥ log(l).

3. Recursive (c-l)-diversity – A compromise definition that ensures the most common value does not appear too often while less common values are ensured to not appear too infrequently.

2.5.3 T-Closeness

T-Closeness also protects a dataset against attribute disclosure. It ensures that the distributions of values of a sensitive attribute within each equivalence class must have a

[26] K-anonymity - https://www.worldscientific.com/doi/abs/10.1142/S0218488502001648
[27] L-diversity: Privacy beyond k-anonymity - https://dl.acm.org/citation.cfm?doid=1217299.1217302
[28] Types of L-Diversity - https://en.wikipedia.org/wiki/L-diversity

‘distance’ of not more than t to the distribution of the attribute values in the input dataset. Also for T-Closeness there are several variants which differ in the way they measure the distance[29].

2.6 Generalization Hierarchies

Generalization hierarchies, simply referred to as hierarchies in the rest of the documentation, is the most important data transformation mechanism when anonymizing data. Hierarchies can either be used to directly reduce the uniqueness of attribute values or to form clusters that will be transformed using further methods, such as microaggregation. Hierarchies are in essence the rules on how a unique value in a dataset column is to be generalized. The more fine grained the hierarchy is, the better tools such as ARX or ARXaaS performs. Hierarchies is a matrix data structure composed of the original value in the first column and more and more generalized transformations column 1 to n.

Example hierarchy for a set of zip code values using a simple redaction strategy

	0	1	2	3	4	5
0	47677	4767*	476**	47***	4****	*****
1	47602	4760*	476**	47***	4****	*****
2	47678	4767*	476**	47***	4****	*****
3	47905	4790*	479**	47***	4****	*****
4	47909	4790*	479**	47***	4****	*****

[29] T-Closeness - https://ieeexplore.ieee.org/document/4221659

2.7 Anonymizing a dataset

Anonymization of dataset for public release has been thoroughly described, documented and in some cases put into law by many countries and governing bodies[30]. We will give a short recap of the process.

The process roughly breaks down into the following steps:

1. Determine the release model
2. Determine an acceptable re-identification risk threshold
3. Classify variables
4. Calculate the risk
5. Anonymize the data
6. Assess data utility

The release model is how and to whom the dataset will be made available. This will guide the risk threshold. A dataset that is to be used by internally employed data-scientists with a signed NDA has a different risk profile from a dataset which is to be released publicly. dataset fields are then classified into the before mentioned categories (identifying, quasi identifying, sensitive, insensitive). Risk is calculated according to the disclosure models (identity, attribute, membership). If the dataset risk is above the threshold the dataset is anonymized using privacy models. The act of choosing the right privacy model for the dataset is also context dependent. But if the dataset contains sensitive attributes a privacy model which handles sensitive attributes should be employed. Finally the utility is judged. Different tooling exists to help in this step, ARX also implements a utility analysis tool[31] but this project has not focused or implemented any features for this step of the process.

[30] De-identification - https://www.ipc.on.ca/wp-content/uploads/2016/08/Deidentification-Guidelines-for-Structured-Data.pdf
[31] ARX | Utility analysis - https://arx.deidentifier.org/anonymization-tool/analysis/

3 Process documentation

This chapter provides insight into the team's work process, including method and tools. The document includes the teams decisions and experiences using new and modern tools and development processes. This document has high value for users looking to gain insight in terms of estimation, planning and executing a software project. Readers interested in the team's experiences with the work strategy utilized should read this chapter.

3.1 Introduction

This chapter covers the process of the bachelor project work. The initial exploratory phase with the project owner at NAV IT, planning, execution, the tools and development methodologies that helped us produce a solution that meets the client's needs. Significant space in this document will be allocated to the tools and methodologies. The client signaled early on, that they wished the project work to follow *agile* practices. The team decided to use the *Scrum* framework as its development process framework.

3.2 Planning and methods

This chapter explains the creation of the team and planning of the project as well as the planning during the project. The chapter starts with development methods applied by the team, followed by documentation about project phases and conclusion.

3.2.1 Development methods

The client NAV IT is in the middle of their own *digital transformation,* there was an immediate demand for the project and the team members to follow *agile* practices. The first assignment the team got back in November was to read the book **Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations[32].** The book lays out in scientific terms, and with data to back it up, what differentiates high performing and low performing IT organizations. Key takeaways that were utilized in the project where:

- **Build quality into the solution from day one**
  Invest in tools and processes for detecting and resolving issues as soon as they appear. See the chapters on 3.2.1.3 Test driven development(TDD) and 3.2.1.5 Code reviews, for more on how the team handled this.

- **Work in small batches**
  Deliveries consisting of large chunks of work introduces much bigger risks of merge conflicts. It prolongs the feedback intervals and increases the risk of misimplementation. See chapter 3.2.1.6 Feature slicing for further elaboration.

[32] Accelerate - https://www.oreilly.com/library/view/accelerate/9781457191435/

- **Computers do repetitive tasks**
  Many bachelor projects are completed with limited thought to the testing, publishing of user documentation and deploying of the solution. The pitfalls of this can be seen in the vast collection of OsloMet IT bachelor rapports. Bugs sneak in when the solutions go too long without deploying to a production(like) environment. When the deploy process is manual, it will become rarer.

- **Continuous integration**
  Keep branches short lived. In conjunction with working in small batches, continuous integration is the practice of merging these batches of work into the master branch continuously. See the chapter on 3.2.1.2 Continuous integration and Continuous delivery, for more detail. Suffice to say this might be the most important takeaway the team had from the book.

- **Continuous testing**
  Instead of leaving the testing for the last part of a solution release. The codebase and every new feature should be continuously tested during development and lifetime. Without the practice of continuous testing, continuous integration would be reckless. See the chapter on 4 Test documentation, for a complete overview.

- **Continuous improvement**
  Foster a team environment where continuously asking questions and taking action to improve technical debt as well as process detail. See the chapter on 3.2.1.1 Agile work process, for more on how we implemented continuous improvement in the team.

Our approach was to deliver a minimum viable product, and deliver small features as increments in order to always have a running application. Close contact with the product owner was always a high priority, to ensure that the solution was developed to the product owner's requirements and needs.

3.2.1.1 Agile work process

The team decided to use the Scrum framework to guide the development process. Scrum is a framework in the agile process family[33], and as such it was in compliance with our product owner's demands and wishes for an agile development process. The Scrum framework was chosen because it best modeled the timeboxed nature of the project. Scrum emphasizes dividing the development into discrete timeboxes. Since the project had a very hard deadline, it was a natural fit.

[33] Agile methodologies - https://www.blueprintsys.com/agile-development-101/agile-methodologies

SCRUM includes concepts that the team uses, and configure them to the team's needs. The reader is expected to be familiar with these terms.

- **Product Owner**
  The role of the product owner as described in the Scrum Guide[34] is to develop and maintain the product backlog. The backlog is a prioritized list of user stories that aim to describe value providing features the development team is to implement. Robindra from NAV IT AI lab was the project's product owner.

- **Sprint**
  Sprint, a time-box of one month or less where work is planned, completed and delivered. The team worked in two week sprints. At the start of every sprint the team completed sprint planning and picked the user stories to add to the next sprint backlog. The team then added the user stories picked for the sprint to the Kanban board in Asana. The team members choose and assigned the tasks as a group.

- **Definition of Done (DoD)**
  A definition of done was specified for each sprint by the group. An example of this was that every feature had to be tested before merging into the master branch.

- **Daily Scrum**
  During each sprint the team had daily scrum meetings to keep the rest of the team updated on each individuals progress as well as to share knowledge in the team.

The following Scrum concepts have been given a little more space in the text. This is to make it clear how important these practices where to the team. The following two practices where absolutely instrumental to force the team to understand the client and to keep the team atmosphere good and spirits high.

[34] Scrum Guide - https://www.scrumguides.org/

Sprint Review
The team presented the new features to our product owner at the end of each sprint. Based on the feedback from the product owner and other NAV IT stakeholders, the team created new user stories and definitions of done for the next sprint.

Figure 9 - of a sprint review with NAV IT stakeholders

Sprint Retrospective
At the end of the sprint the team had a sprint retrospective, also referred to as a sprint retro. Starting back in January several team members voiced scepticism for this Scrum practice, but in the end this had become a team favourite. The goal of this meeting is to serve as a time and place for airing out team friction. The goal was that each team member leaves the meeting feeling heard and respected and that the team is strengthened by a better collective empathy. By the end of the sprint retrospective, the team should ideally have identified improvements that it could implement in the next sprint.

During the sprint retrospective, the team discussed
- What went well in the sprint?
- What could be improved?
- What will we commit to improve in the next sprint?

The way team team went about this was that each member made post-it notes labeled **start**, **stop** and **continue**. The start post-it notes where suggestions of team improvements to be made in the next sprint. The stop post-it notes where things that had not worked well for the team or an individual. The continue post-it notes where items that worked well in the sprint and the team member wanted to keep doing in the next sprint. For examples see the sprints documented in 3.3 Development process.

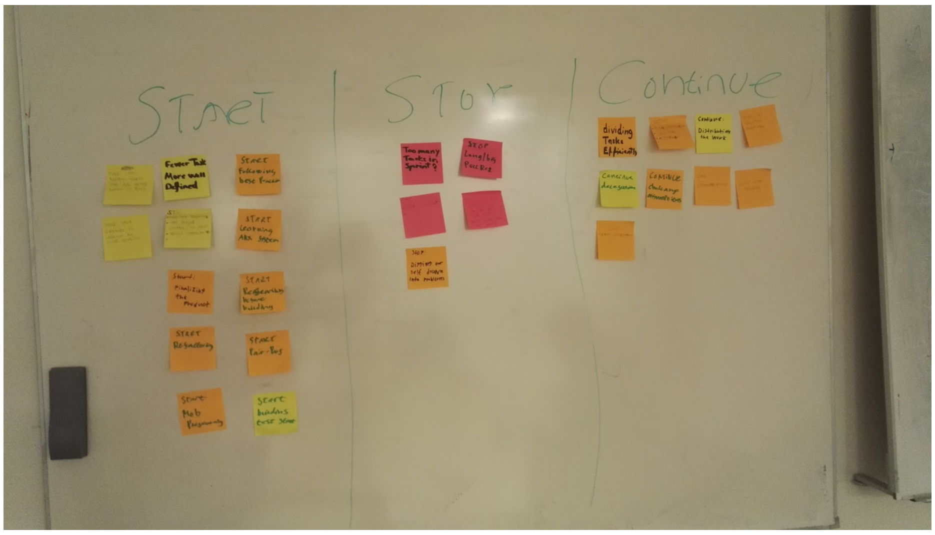


Figure 10 - of a Scrum retrospective board from a team retrospective meeting

Figure 11 - of a team Sprint Retrospective at NAV IT's offices

3.2.1.2 Continuous integration and Continuous delivery

Continuous integration is the practice of continuously integrating new work into a product's master branch, and it was the most important takeaway from the Accelerate book. Implementing continuous integration additionally makes a another practice possible; continuous delivery. Continuous delivery is the practice of automating the software release process. This fits nicely into the other takeaways such as continuous testing and letting computers do the repetitive work. When a team wants to practice continuous integration and continuous delivery they build a ***CI/CD* pipeline**. The responsibility of a *CI/CD* pipeline is to automate the complete process of integration and release, so that each new change to the software is efficiently and securely applied to the code base.

3.2.1.2.1 CI/CD pipeline

This is the overview of the *CI/CD* pipelines the team built for the two main products; ARXaaS and PyARXaaS. The *CI/CD* pipeline is separated into discrete stages, together making up a build. If a stage fails the whole build is regarded as failed and the development team is notified that a *CI/CD* job has failed.

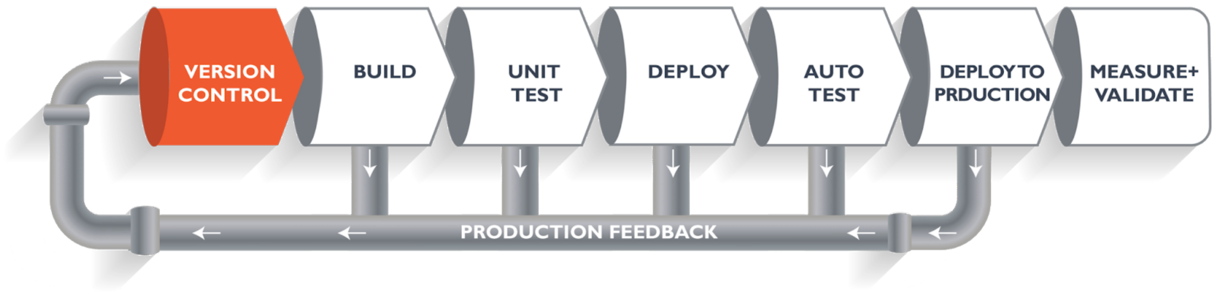
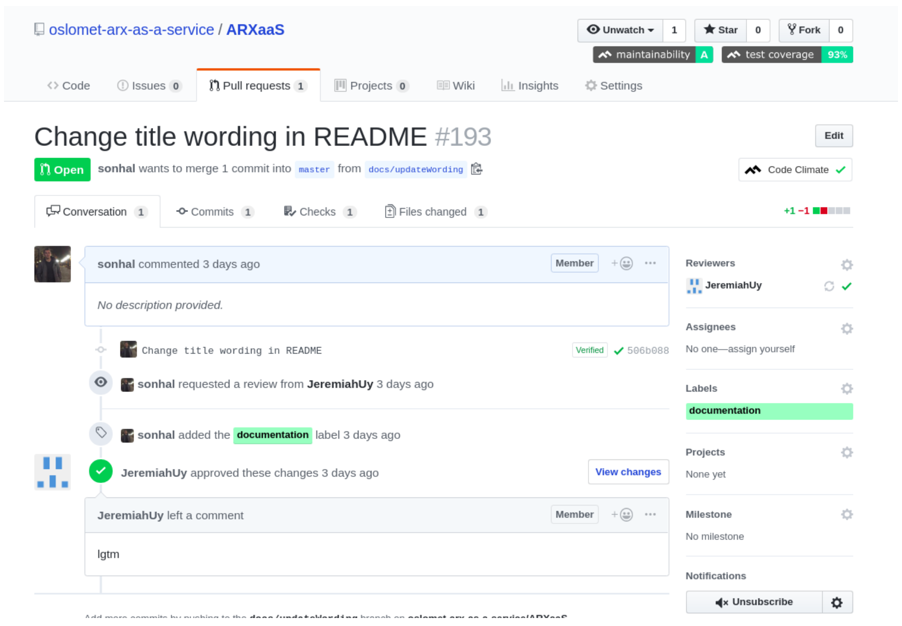


Figure 12 - Diagram of a CI/CD pipeline

For the *CI/CD* pipeline to be efficient, it needs to be highly available for all team members, provide accurate feedback on a build status, and integrate nicely with the version control host. The team's choice fell on Travis CI which is a feature rich and professional *CI/CD* platform. Travis CI also provides free use of their platform to *open source* projects.

A developer working on ARXaaS or PyARXaaS interacts with the *CI/CD* pipeline mainly through the Github page for the project. The team configured the Github repositories to block direct merging to master branch. The only way to merge new code into the master branch is to make a *pull request* on Github. On Github the requirement were for all *CI/CD* stages to pass successfully and for one other team member to accept the changes proposed. Travis CI spins up a virtual machine and runs the configured stages on the branch, this is referred to as a *CI/CD* job. If all the stages passes, it can be merged into to master branch.


oslomet-arx-as-a-service / **ARXaaS**

Unwatch 1 | Star 0 | Fork 0

maintainability A | test coverage 93%

Code | Issues 0 | Pull requests 1 | Projects 0 | Wiki | Insights | Settings

Change title wording in README #193

Edit

Open **sonhal** wants to merge 1 commit into `master` from `docs/updateWording`

Code Climate ✓

Conversation 1 | Commits 1 | Checks 1 | Files changed 1 | +1 −1

sonhal commented 3 days ago — Member

No description provided.

`Change title wording in README` Verified ✓ 506b088

sonhal requested a review from **JeremiahUy** 3 days ago

sonhal added the **documentation** label 3 days ago

JeremiahUy approved these changes 3 days ago — View changes

JeremiahUy left a comment — Member

lgtm

Reviewers: JeremiahUy ✓

Assignees: No one—assign yourself

Labels: documentation

Projects: None yet

Milestone: No milestone

Notifications: Unsubscribe

Figure 13 - ARXaaS pull request waiting to be merged

Afterwards, a team member would be notified that they have been requested to review a pull request. The team member could then approve or request changes to be made to the code.

CI/CD pipeline stages
The following stages is a overview applicable to PyARXaaS and ARXaaS for more detailed overview see the chapter on 5.4 PyARXaaS and 5.2 ARXaaS of the Product documentation.

- **Static analysis stage**
  This stage is closely related to testing so a more detailed description can be found in chapter 4.4.2 Static code analysis**.** The purpose of the analysis stage is to analyze the code and the dependencies of the products, and provide feedback to development team on metrics such as code complexity and vulnerabilities in the code and dependencies.

- **Test**
  The full test suite is with unit tests, integration tests and system tests are ran to verify that there are not any regressions and that the new tests are successful.

- **Documentation**
  Documentation is generated from human written text files, versioned with the product and documentation generated from the source code and tests

- **Packaging**
  The service is packaged in different artifacts with different formats for the deployment and artifact hosting platforms.

- **Publishing**
  The packaged artifacts are deployed to different platforms and hosts, to make it available to the users of the solution. This includes source, executable and documentation artifacts

- **Deployment**
  For ARXaaS, there is an additional step involving deployment of the service and execution on the platforms used, namely the team's *Kubernetes* cluster instance and the client's NAIS platform.

Figure 14 below shows the CI/CD pipeline for ARXaaS. Note that there are several steps and services involved that make up the pipeline. As the whole process is automated lots of otherwise manual labor is avoided and quality and stability of the service is improved.

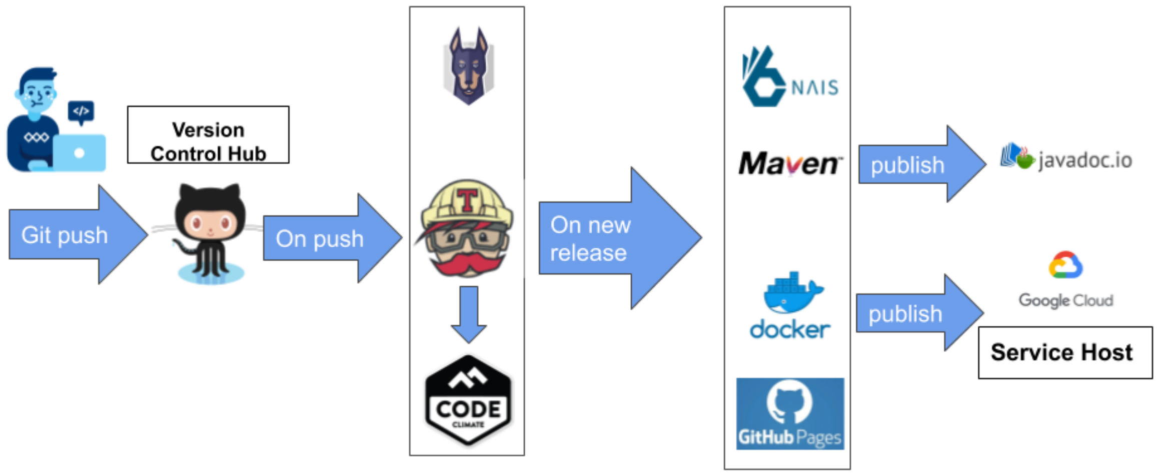


Figure 14 - Overview of ARXaaS CI/CD pipeline

3.2.1.3 Test driven development (TDD)

Test driven development entails writing the test before writing the code is to be tested. The practice is somewhat controversial, and the team has deviated from it on occasions. The team can attest to that the best code in the project codebase has been developed as a result of this practice.

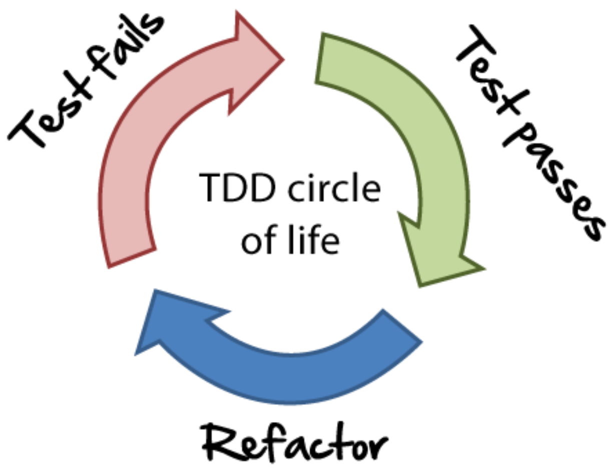


Figure 15 - showing the process of test driven development

A benefit of the practice has been an incredibly robust set of tests and large test coverage. As of writing this report this is the test coverage on the main products:

ARXaaS
- 93% coverage

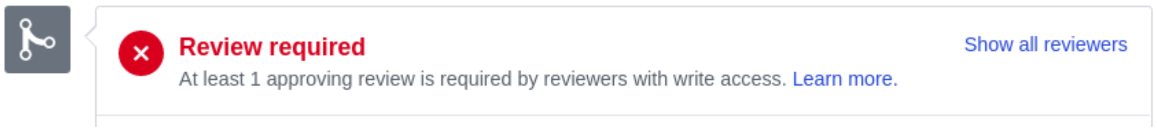
PyARXaaS
- 89% coverage

3.2.1.4 Code style

Used Sonarlint in the team members IDE's to make sure the code conforms to a uniform code style.

3.2.1.5 Code reviews

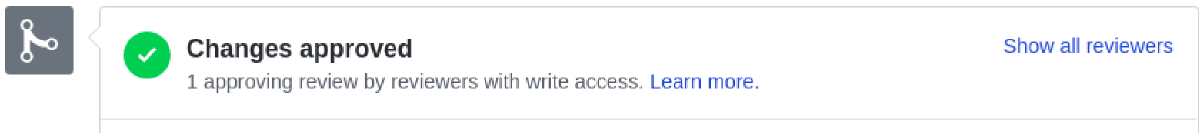
Git was chosen as version control host for this project. Before merging a branch into the master branch Travis, Code climate and SNYK tests had to be passed. When all tests pass a team member had to make a pull request to get a team member to review the branch in question before being merged into the master branch.



Review required

At least 1 approving review is required by reviewers with write access. Learn more.

Figure 16 - Screen grab from a Github pull request waiting for review



Changes approved

1 approving review by reviewers with write access. Learn more.

Figure 17 - Screen grab from a Github pull request that has been reviewed and accepted

3.2.1.6 Feature slicing

The team worked actively on slicing up big features requested by the product owner into smaller features. After slicing the features, the team evaluated the different features and decided with the product owner which features where best to implement first. This ensured that the product owner was in on the prioritization and frequency of deliveries. Deliveries were made in small increments that provided the most value as fast as possible.

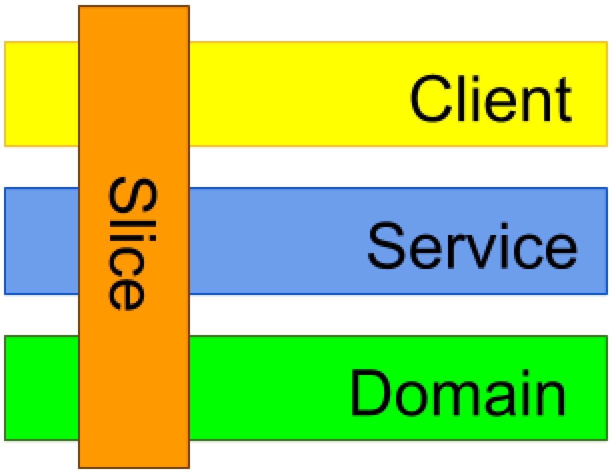


Figure 18 - Example figure of feature slicing

3.2.2 Project phases

3.2.2.1 Initiation

The project started with a get to know meeting at Starbucks Torggata 18.07.2018. Each team member discussed their expectations for the project and the way forward. Sondre had a contact person at NAV IT and had talked about a possible bachelor project with them. Our contact person at NAV IT had mentioned key words such as Kafka, Docker / Kubernetes, Java, Python, data lakes, data virtualization as possible areas to work with. During this meeting It was also suggested to recruit a fifth party to the team. The team then contacted Viktor and asked if he wanted to join the team. After a second get to know meeting Viktor joined the team.

3.2.2.2 Planning

An information meeting was held with the team and tech leads for Open Data, Tommy and Gøran from NAV IT. It was determined that the goal of NAV IT was to build a ‘pipeline’, that makes raw data that NAV sits on available to stakeholders outside NAV (media, private persons, etc.). This pipeline was to accommodate the entire range from integration with raw data systems at NAV, to visualization and typical front-end solutions.

The solution that Open Data project from NAV IT wants to deliver, is a common platform / toolbox for delivering open data to the Norwegian population. The idea was that the team will contribution to this project. The team was to be involved in developing the plan for a specific solution that was to be implemented to the Open Data project framework.

Concrete features in the Open Data project that the team could participate in was, API - backend pipeline, Portal - web platform (data.nav.no) and Meta-data system for automatic generation of visualizations.

The team was given a few task to complete before January 1st 2019. The team was asked to get to know Docker and Kubernetes and read the Accelerate book. Furthermore the team was asked to brainstorm ideas for solving 'How can NAV deliver specific datasets, in a good and user-friendly manner (user-friendly for developers, media houses, private persons)'.

A proposed idea was to focus on a specific data type. The initial idea was to focus on treatment time. What kind of issues have the longest treatment time and shed light on it. The goal was to streamline the assessment time spent on treatment.

At 18.10.18 the team had its first meeting with Eva Hadler to talk about guidance for the bachelor project. Eva was highly recommended and the team was happy she decided to be the supervisor for the project. Eva shared her experiences with undergraduate projects and what was expected.

The group had their first bachelor meeting in 2019. After discussing frameworks and tools, the team also decided to use Github, Asana, Travis and Slack. The team also decided on the work days and hours that best suited all the team members.

At this point the project objective was not set in stone, but at the next meeting at NAV IT, Robindra and Paul presented the data anonymity project to the team. The essence of the project was to make ARX's functionality available to NAV's systems, and build a framework that makes ARXs anonymization functionality available to NAVs data scientists. The functionalities sought after were submission of datasets to the service, and to receive an anonymized dataset of the desired anonymity level.

The group discussed different approaches to the problem and tools to use. Tentatively the team planned an overall architecture with Java / Spring backend and Python / Jupyter Notebooks, Javascript / React frontend and Docker as scalability. The group agreed that this was a good project with both enough content and room for stretch goals.

The team decided on an English working language for documentation and code to make the open source project available to everyone. Robindra was named *Product Owner* and Sondre got the title as *Scrum Master*. For the development of the solution, the team decided to use seven sprints with a duration of two weeks. The remaining four weeks will be used to write the project report.

3.2.2.3 Execution

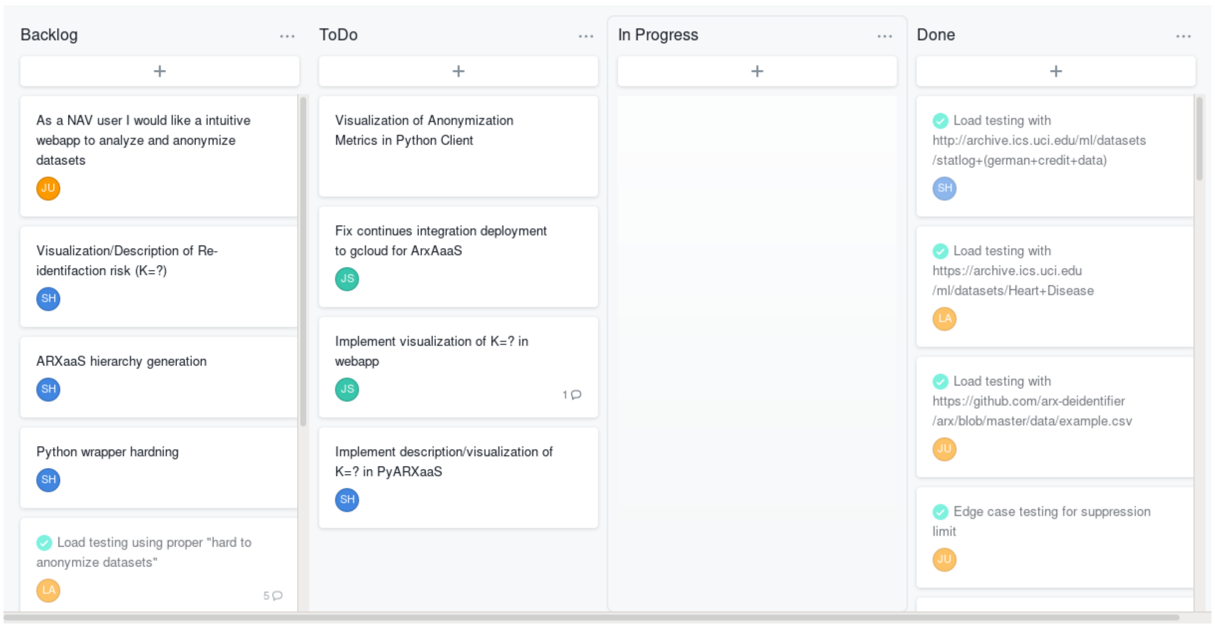
The project lasted from 08.01.2019 to 23.05.2019 and ended with a presentation of the solution 14.07.19. During this time the team had workshops and presentations for the NAV IT staff as well as a presentation for the Agency for Public Management and e-Government[35] (Difi). The solution was implemented in the NAV IT system and is now running on the NAIS platform, Google Cloud and hosted by Maven. The entire group was offered summer internships at NAV IT to continue development of the solution and all team members accepted.

3.2.2.4 Project conclusion and documentation

The solution handover took place at NAV IT headquarters at 26.04.2019 where the team presented the final solution to NAV IT. The team then worked on the project report as well as online documentation and user manuals until delivery 23.05.2019.

3.2.3 Planning tools

The team decided early to use Asana[36] for sprint planning and project management. The sprints were divided in 'backlog', 'to do', 'in progress' and 'done'. This was displayed in a Kanban board in Asana and allowed all team members to see each team members tasks and progress. The larger backlog items were divided into smaller subtask where each subtask was assigned to a team member. The tasks also had notes on possible solutions or tools to use for completing the task.



Backlog

As a NAV user I would like a intuitive webapp to analyze and anonymize datasets — JU

Visualization/Description of Re-identifaction risk (K=?) — SH

ARXaaS hierarchy generation — SH

Python wrapper hardning — SH

✓ Load testing using proper "hard to anonymize datasets" — LA — 5

ToDo

Visualization of Anonymization Metrics in Python Client

Fix continues integration deployment to gcloud for ArxAaaS — JS

Implement visualization of K=? in webapp — JS — 1

Implement description/visualization of K=? in PyARXaaS — SH

In Progress

Done

✓ Load testing with http://archive.ics.uci.edu/ml/datasets/statlog+(german+credit+data) — SH

✓ Load testing with https://archive.ics.uci.edu/ml/datasets/Heart+Disease — LA

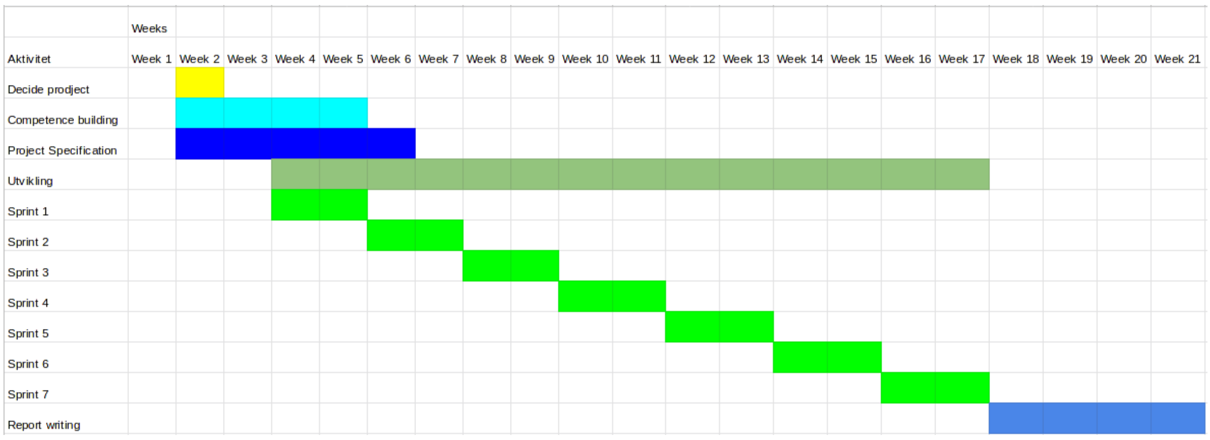
✓ Load testing with https://github.com/arx-deidentifier/arx/blob/master/data/example.csv — JU

✓ Edge case testing for suppression limit — JU

Figure 19 - Example Asana Kanban board

[35] Difi - https://www.difi.no/om-difi/about-difi
[36] Asana homepage - https://app.asana.com/

After each sprint retro meeting with the product owner the team created new user stories and tasks for the backlog. During sprint planning the team decided which task to bring to the new sprint as well as team members assigned to the task. The team tried to keep the tasks and features small enough to be completed in one sprint. To get an overview of the entire project the team created a Gantt diagram that gave the team a great tool to keep track of the sprint and the time left for the project.



	Weeks																				
Aktivitet	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10	Week 11	Week 12	Week 13	Week 14	Week 15	Week 16	Week 17	Week 18	Week 19	Week 20	Week 21
Decide prodject		🟨																			
Competence building		🟦	🟦	🟦																	
Project Specification		🟦	🟦	🟦	🟦																
Utvikling				🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩				
Sprint 1				🟩	🟩																
Sprint 2						🟩	🟩														
Sprint 3								🟩	🟩												
Sprint 4										🟩	🟩										
Sprint 5												🟩	🟩								
Sprint 6														🟩	🟩						
Sprint 7																🟩	🟩				
Report writing																		🟦	🟦	🟦	🟦

Figure 20 - Gantt Diagram

3.2.4 Competence building

The team used literature and courses, both online and through presentations held both by the product owner and Sondre Halvorsen for competence building.

3.2.4.1 Literature

Prior to the start of project the team was given an assignment, to read a book called ‘Accelerate’ by Nicole Forsgren. The team learned different working methods, that makes software development more efficient. These working methods can be found in chapter 3.2.1 Development methods.

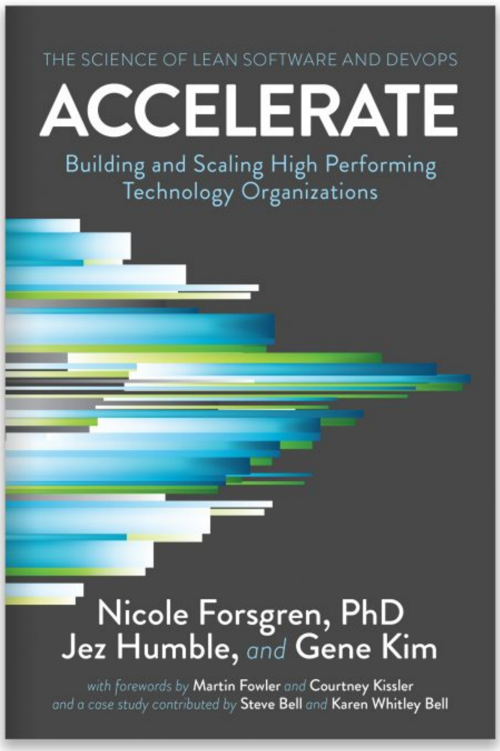

THE SCIENCE OF LEAN SOFTWARE AND DEVOPS
ACCELERATE
Building and Scaling High Performing Technology Organizations
Nicole Forsgren, PhD
Jez Humble, and Gene Kim
with forewords by Martin Fowler and Courtney Kissler
and a case study contributed by Steve Bell and Karen Whitley Bell

Figure 21 - Image of Accelerate book

3.2.4.2 Courses

Includes both online courses, presentations and workshop the team participated in.

3.2.4.2.1 Online Courses

The team used Pluralsight[37] and Safari books[38] for online competence building.

Pluralsight courses:

- Scrum fundamentals[39]
- Spring fundamentals[40]

[37] Pluralsight - https://www.pluralsight.com/
[38] Safari books - https://www.oreilly.com/?utm_source=my&utm_medium=referral&utm_campaign=classic
[39] Scrum fundamentals - https://www.pluralsight.com/courses/scrum-fundamentals
[40] Spring fundamentals - https://www.pluralsight.com/courses/spring-fundamentals

- Maven fundamentals[41]

Safari books:
- Spring with microservice[42]

The team used Pluralsight to improve their knowledge of Scrum, and to make sure that everyone in the team had the same understanding of Scrum.

To have a better understanding of Spring boot and microservice, the team utilized the 'Spring fundamentals course' on Pluralsight and 'Spring with microservice' on Safari books. This ensured that the team could correctly set up the service by using Spring boot.

The Maven Fundamentals course was used to learn about the build structure of Maven. The team decided to use Maven as the dependency management tool for Spring boot.

3.2.4.2.2 Presentations and workshops

To improve the team's knowledge on Python programming, Sondre held a Python programming workshop. This ensured that the team was up-to-date on Python programming and made sure that everyone had at least basic Python programming knowledge.

The Python programming course slides can be found here:
https://sonhal.github.io/googlecodelabs-python-kurs/nav-python-course/index.html#0

The product owner Robindra Prabhu held a introductory presentation on de-identification to improve the team's knowledge on data anonymizing. Through the presentation the team gained a better understanding of the different privacy models used, as well as the different re-identification risk involved when anonymizing a dataset.

3.2.5 Budget

The budget for the project was 200 NOK for the Google Cloud service. The team's software was either student licensed or free to keep the cost of the project to a minimum.

[41] Maven fundamentals - https://www.pluralsight.com/courses/maven-fundamentals
[42] Spring with microservice -https://www.safaribooksonline.com/library/view/spring-microservices-with/9781789132588/?ar&orpq

3.3 Development process

This chapter is meant to give the reader detailed insight into the sprints used to develop the solution.

3.3.1 Descriptions of the sprints

The team worked in two week sprints, and used seven sprints to develop the solution. There was also a sprint 0 before the team started work on the solution. This chapter contains detailed descriptions of the sprints. Each sprint is divided into:

- Duration
- Summary
- Backlog
- Goals
- Sprint Review
- Sprint retro

3.3.1.1 Sprint 0

Duration: Monday 07.01.19 - Friday 18.01.19
Summary: The team finalized the project assignment with NAV IT and started working on writing the pre-project report. The team created: ● A repository in the version control host Github was created, where the team members had the opportunity to start experimenting and exploring the ARX library. ● Created a repository in the version control host for the solution service and started preparing the REST controller. ● The project webpage.
Backlog:

[COMPLETED] Finalize the project with NAV IT
[COMPLETED] Write the pre-project rapport
[COMPLETED] Create ARX Playground project for Testing and exploring the ARX core

library. [COMPLETED] Create Java Spring boot repository on Github for development of the service. [COMPLETED] Publish the project webpage
Goals: ● We did not create any sprint goals for this sprint
Review: The team finalized the project with NAV IT. The team did group education in Scrum and Python programming. As well as completing the entire backlog. The team presented the pre-project rapport to Eva the teams Bachelor thesis supervisor. Eva approved the teams pre-project rapport, but pointed out important areas of improvement for the final rapport. Eva also provided guidance for the requirements document the group has to finalize by 20.01.19
Retro: In this first unofficial sprint we did not do a sprint retro

3.3.1.2 Sprint 1

Duration: 21.01.19 - 01.02.19
Summary: The team started working on implementing the continuous integration workflow on both the service and client repository in the version control host Github. Started with implementing a feature to anonymize a dataset with a k-anonymity privacy model. Finished implementing the spring controller class in the service, and started to build a docker image and published the service to google container registry.
Backlog:

[COMPLETED] Implement python wrapper API for k-anonymity
[COMPLETED] Implement Spring Controller class in the service
[COMPLETED] Build service in a docker image
[COMPLETED] Publish AaaS backend docker image to Google container registry
[COMPLETED] Setup travis for PyAaaS

Goals:
In sprint 1 we defined this definition of done (DoD) as part of our Sprint:

- Unit testing implemented:
  - Coverage for normal case
  - Coverage for sensible edge cases
- Testing running in Travis
- JavaDoc on Classes and methods
- Code Review completed, min=2
- Code is complaint to linter
- Code complaint to Static code review

Review:
Participants:

- Robindra
- Gøran
- Sondre
- André
- Viktor
- Jeremiah
- Julian

Delivery:
https://github.com/OsloMET-Gruppe-8/PyAaaS/releases/tag/Release-0.0.2
AaaS
The team presented the delivery from Sprint 1. A Python wrapper package and Spring boot backend for ARX functionality. The current feature set is anonymization of data (pandas.DataFrame, csv string) with k-anonymization.

Feedback from Product owner:

The delivery for sprint 1 has delivered to expectations. The choices regarding the public api exposed by the python package was approved. Requests for the next Sprint: (Not ordered by priority) ● Implement deploy pipeline to NAIS platform ● Implement support for L-Diversity Privacy Model ● Documentation and example cases for non-experienced users ● Implement Hierarchy Generator ● Create NAV/Norway specific Hierarchies ● Return metadata from anonymization with the anonymized dataset ● Provide easy integration for data package metadata format and metadata from anonymization ● Provide continuous feedback to user when completing subactions with Python wrapper
Retro: Good not ordered by priority: ● Good first increment from sprint 1 Challenging ● Frustration with new technologies ● Challenges with work documentation in Agile process ● Time for learning

3.3.1.3 Sprint 2

Duration: 04.02.19 - 15.02.19
Summary: The team started working on implementing new features , anonymizing a dataset with L-diversity privacy model and receiving a re-identification risk profile when analyzing a dataset. The team started working on creating a javadoc for the service.

Created a feature in the client where the different settings to be used to anonymize a dataset is shown before being sent to the service.

Backlog:

[INCOMPLETED] As a data-scientist, I would like to retrieve analytics of re-identification risk for my dataset.
[INCOMPLETED] As a data scientist I would like to be able to use L-Diversity as a Privacy Model for my dataset.
[INCOMPLETED] As a user of the system, I would like java-docs for easy functionality lookup.
[COMPLETED] As a developer using PyAaaS I would like to receive information about the configurations about my anonymization payload before I run the anonymization process.

Goals:

Test Coverage:

- AaaS test coverage 55%
- PyAaaS maintain 80 %
- PyAaaS Increase unit tests vs integration tests

Documentation

- AaaS Javadoc:
  - All Classes
  - All Controller methods
  - All Service methods
  - All ARX util methods
- PyAaaS docstring
  - All classes
  - All class methods
  - Complex internal functions/methods
  - public functions
- Refactor and reduce Technical debt
  - Split ARX wrapper in classes

Review:

Started working with implementing javadoc.
Started back-end implementation of analyzing against re-identification risk.
Started back-end implementation of l-diversity risk.
Implemented functionality to programmatically display information from my request object in the frontend.
Started implementing maven deployment and generation of PGP-key pair.

Retro:
Completed 18.02.19
We timeboxed 1 hour monday to complete Sprint Retro for Sprint 2 (week 6-7)

Concrete improvements:
- Every project member is responsible for writing a short summary of their day in the project diary
- We schedule a debrief meeting (10 mins per member) before each Sprint Review for sharing of concrete features they have worked on
- We will make ourselves available for the Product owner(Robindra) at least one day a week

Good:

- Fun at work
- Independent working
- On schedule - right amount of work
- Good increments
- Good motivation for work
- The team is good at giving feedback
- Stabile and good progression
- Good cooperation and sharing of knowledge
- Good teamwork when taking decisions
- The team uses Scrum well
- Good attendee's = 100%
- Good cooperation when developing solutions
- Realistic expectations

Improvements:

- Avoid blocking tasks
- Splitting in group of skills - risk of isolation
- Team members are arriving late for work in the morning
- Lots of interruptions
- Time is used for other school work meant for bachelor work.
- Challenges with Python skills
- Get better at proposing improvements to the entire team.

● Improve focus in time boxed events ● Improve effort in writing reports ● Improve handling of blocked task ● Improve focus on product owner

3.3.1.4 Sprint 3

Duration: 18.02.19 - 01.03.19
Summary: Finished implementing the analyzation feature for the service and client. Finished implementing the different L-diversity privacy model for the service. Continued to work on more documentation for the javadoc. Started working on implementing a feature in the continuous integration pipeline, to build a docker image of the service and publishing it to the google cloud repository. Started exploring the possibilities on implementing a feature for global value generalization as a transformation scheme.
Backlog:
[INCOMPLETED] As a data scientist, I would like to be able to set a global value generalization as my transformation scheme for column field.
[INCOMPLETED] As a data engineer supporting the ARXaaS service, I would like to be able to deploy the service to a docker container environment.
[COMPLETED] As a user of the system, I would like java-docs for easy functionality lookup.
[COMPLETED] As a data scientist, I would like to be able to use ℓ-Diversity as a Privacy Model for my dataset
[COMPLETED] As a data-scientist, I would like to retrieve analytics of re-identification risks for my dataset.
Goals:

- Test Coverage:
    - AaaS test maintain 80%+
    - PyAaaS test maintain 80%+
    - create AaaS integration tests for new and old functionality
- Documentation:
    - AaaS Javadoc
        - All Classes
        - All Controller methods
        - All Service methods
        - All ARX util methods
    - PyAaaS docstring
        - All classes
        - All class methods
        - Complex internal functions/methods
        - public functions

Goals from sprint 2 continued in sprint 3:

- Every project member is responsible for writing a short summary of their day in the project diary.
- We schedule a debrief meeting (10 mins per member) before each Sprint Review for sharing of concrete features they have worked on.
- We will make ourselves available for the product owner(Robindra) at least one day a week
- Refactor and reduce Technical debt
    - Split AaaS ARX wrapper in classes
    - Split PyAaaS
    - AnonymizationPayload class

Review:
Implemented the feature to retrieve analytics of re-identification risks for the dataset.
Implemented generation of javadoc in maven and automatically deploy it to an online service for hosting using travis.
Split the ArxWrapper back-end class into modules.
Fully implemented the use of L-diversity and its variants.
Successfully deploying to maven central with hosting of the javadoc.
Generate PGP-key pair and encrypt all the jar-s and pom file upon deployment.
Sonatype authentication upon deployment.

Retro:
We are going to keep concrete improvements from the last sprint. This has worked well for the team and the product owner.

Every project member is responsible for writing a short summary of their day in the project diary.

We schedule a debrief meeting (10 mins per member)before each Sprint Review for sharing of concrete features they have worked on.
We will make ourselves available for the Product owner(Robindra) at least one day a week.

The group did a Start-stop-continue session
Where each team member writes down at least one thing for each the three categories.

Start:

- If we have to change workplace or time. Try to do it in daytime the day before
- Start documenting completed features
- Star increasing documentation frequency?
- Python course?
- Start focus on building features

Stop:

- Stop coming a little late
- Stop overestimating how much we can do in a sprint
- Stop late meeting changes when we don’t need to. Try do it in the daytime
- Stop assuming that someone else will fix it

Continue:

- Continue being open and helping each other
- Continue working dynamically
- Continue planning daily schedules
- Continue Planning daily schedules
- Continue showing up at time
- Continue with good moral
- Continue daily individual project diary
- Continue to find errors
- Continue to take care of each outer
- Continue to keep product owner in focus

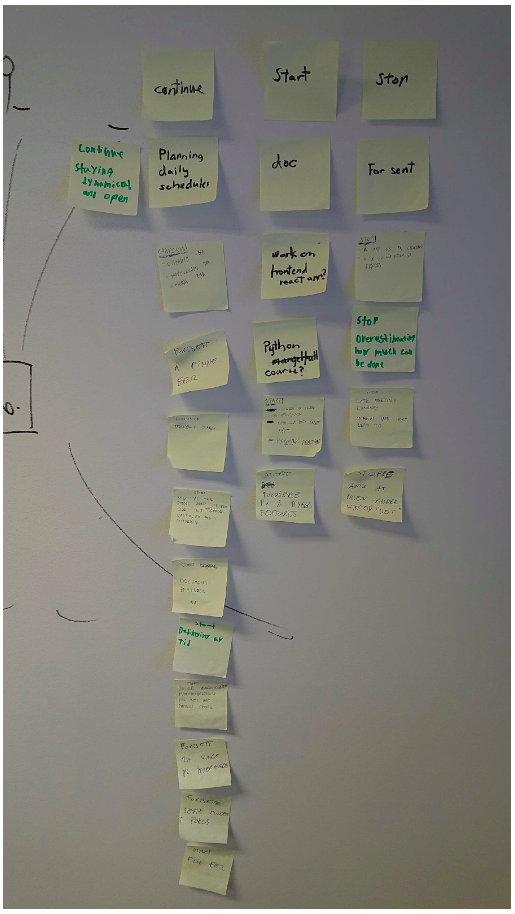

continue
Start
Stop
Continue staying dynamical and open
Planning daily schedule
doc
For sent
work on frontend react app?
Python course?
Stop overestimating how much can be done

Figure 22 - Sprint retro Start, Stop and Continue

3.3.1.5 Sprint 4

Duration: 03.03.19 - 15.03.19
Summary: Created a new increment of PyAaaS and ARXaaS with support for anonymizing datasets with Privacy Models and already generated hierarchies, and analyzing risk profile of a given dataset. ● Implemented the l-diversity feature for the python client. Fixed a bug when reading the python client where csv files were not read with utf 8. ● Implemented a feature to analyze a dataset before anonymizing. ● Started working on implementing a feature to receive anonymization analytics and better error messages when assigning a hierarchy on non-quasi-identifying attributes.

- Started working on a quick step-by-step guide on how to use the client and its features.

Backlog:

[INCOMPLETED] As a data scientist, I would like the possibility to choose more than one set of anonymized dataset output, Currently only showing best ranked result. Possibility to choose subsets of data ranked as 2 or 3 if the user needs less washed data-set.
[INCOMPLETED] As a data scientist, I would like to get anomyzation analytics for a given dataset.
[INCOMPLETED] As a data scientist, I would like to get a error message if we assign a hierarchy for a dataset field that has a sensitive attribute.
[INCOMPLETED] As a data scientist, I would like a documentation to guide me through the complete process of anonymizing a dataset using Pyaaas and ARXaaS
[INCOMPLETED] As a data scientist, I would like to be able to set a global value generalization as my transformation scheme (hierarchy) for column field
[COMPLETED] As a data scientist, I would like to be able to analyze my dataset before anonymization
[COMPLETED] As a data-scientist, I would like Pandas read functionality for utf8. Æ,Ø,Å is currently not recognized.
[COMPLETED] As a data-scientist, I would like the front-end to handle data field columns that contain spaces. l-diversity function currently splits the column names containing spaces.
[COMPLETED] [BUG] ARXaaS cannot handle CSV files using other than "," separator
[COMPLETED] As a project team, we would like Travis to deploy ARXaaS

Goals:

Test Coverage

- AaaS test maintain 80%+
- PyAaaS test maintain 80%+
- Create AaaS integration tests for new and old functionality

Documentation:

- AaaS Javadoc
  - All Classes
  - All Controller methods
  - All Service methods
  - All ARX util methods
- PyAaaS docstring
  - All classes
  - All class methods
  - Complex internal
- Functions/methods
  - public functions

Goals from Sprint 3 continued in Sprint 4:

- Every project member is responsible for writing a short summary of their day in the project diary.
- We schedule a debrief meeting (10 mins per member)before each Sprint Review for sharing of concrete features they have worked on.
- We will make ourselves available for the PO(Robindra) at least one day a week.
- Refactor and reduce Technical debt.
  - Split AaaS ARX wrapper in classes.
  - Split PyAaaS AnonymizationPayload class.

Review:

- Implemented a new model that fixed separator problem when making a Data object in the backend.
- Implemented a fix for hierarchies overwriting attribute types of dataset fields.
- Implemented a function to analyze a dataset before anonymizing.
- Importing csv with utf-8 support.
- Fixed a bug when setting L-diversity with white spaces on dataset field name.
- Splitting out ARXWrapper into domain specific classes.
- Refactored the backend model for better communication with the domain.

Retro:
The group did a Start-stop-continue session
Where each team member writes down at least one thing for each the three categories.

Start:

- Think about expanding to new features (logging, encryption, webpage).
- Take more tasks.
- Finalizing the project structure so we can working in parallel more easily.
- Start Landing PyaaaS.
- Start stretch goal planning(what can we get done in time).

Stop:

- Naming things imprecisely, if unsure ask?.
- Underestimating how much we can do in a sprint.
- Slacking on docs, logs and reports regarding the project.
- Need more structured sprints with a more thought out planning phase.

Continue:

- Communicating problems etc.
- Being motivated, showing up on time
- Looking for ways to improve
- Pair programming
- Motivation
- Attendance
- Challenge ideas, assumptions and decisions
- Listening to the users, and taking their needs into account
- Continue feedback loop with Robindra
- Continue with good work discipline(Don't get lazy) and stay focused.

3.3.1.6 Sprint 5

Duration: 18.03.19 - 31.03.19
Summary: ARXaaS went into production on the NAIS platform. The team implemented HTTPS support and configuration of HTTPS. Metrics from the service was made available through Spring boot actuator and prometheus endpoints. The team implemented logging service using Log4j. Exceptions are now returned to client with descriptive HTTP status. Risk profile has been made more rich with distributed risk as a new data point and attacker success rate. Metrics from anonymization like attribute generalization is now returned with

the anonymized data. Time elapsed when anonymizing is also returned. Properties of the Privacy Models used in anonymization are included. Service now also has a subpage for documentation of the service and it's endpoints using Swagger. Refactoring of the domain models is on-going.

Backlog:

[INCOMPLETED] As a data engineer, I would like to have proper logging from a running ARXaaS application
[INCOMPLETED] As a maintainer, I would like PyAaaS and ARXaaS to be as decoupled as possible
[INCOMPLETED] As a data scientist I would like metadata on the anonymization performed on a given dataset
[COMPLETED] As a data scientist, I would like ARXaaS to be OpenAPI compliant so functionality can easily be discovered and used
[COMPLETED] As a data scientist, I would like to have a richer risk profile for my dataset
[COMPLETED] As a data scientist, I would like descriptive exceptions when something went wrong.
[COMPLETED] As a data engineer i would like operational information about the running application
[COMPLETED] End to end SSL Encryption between ARXaaS and clients

Goals:

Used the same goals as in sprint 4

- Pair-program on features – Sondre is taskmaster
- Make sure product is shippable at end of Sprint – Julian is taskmaster
- Make sure decisions are documented – Viktor

Review:
The team presented the implemented features from Sprint 5 to Erik and Robindra. Robindra requested the team to present the project on Fagforum for kunstig intelligens/data science i offentlig sektor.

Wishes for next Sprint by Robindra :

- T-closeness (with more models if possible/easy)
- Create presentation
- System dynamic for changes in the ARX project

Notes from team:

- More integration tests
- Increase test coverage(jacoco to properly read junit5??)
- After implementing a new feature present it to team
- User documentation for setup of ARXaaS
- User documentation for PyAaaS(Sphinx)

Retro:
The group did a Start-stop-continue session
Where each team member writes down at least one thing for each the three categories.

Start:

- Researching before building
- Learning ARX system/data
- Learning de-identification
- Refactoring
- Following best practice
- Pair-programming
- More pair-programming
- More smaller commits/pull-request
- Precise commits/pull-request
- Invest more time into making tasks more well defined
- Take small breaks when the air in the room is bad.
- More Java courses to improve my code quality

Continue:

- Challenge assumptions
- Moral/motivation
- Questioning

- Attendance
- Efficiently dividing tasks among team members
- good code reviews
- good team cooperation
- good communication
- team is solution orientated

Stop:

- Long/big pullreqests
- Big pull-request
- Adding too many tasks to sprint?

3.3.1.7 Sprint 6

Duration: 01.04.19 - 12.04.19

Summary:
The project team renamed PyaaS to PyARXaaS, to better align the product naming with the service.

The group prepared and had a presentation at DiFi about the bachelor project, and had a workshop with NAV data scientist on analyzing and anonymizing a dataset.
Started working on the Web app, that can connect to the analyze end point. Implemented to types of T-closeness that can be used to anonymize a dataset. The team decided not to implement the 3rd type of T-closeness as this was a bigger task than expected and would take a whole new sprint to fulfill. Fully implemented SSL encryption and created documentation on how to use it. Finished implementing metadata on the anonymization performed on a given dataset. Started PyARXaaS documentation in sphinx. Implemented edge case test and integration test of the end points.

Backlog:

[COMPLETED] as a data interested employee in NAV I would like to be able to analyze the risk profile of a given dataset in a web app
[COMPLETED] As a data scientist I would like that ARXaaS has been stress tested and what load I can expect the system to handle

[COMPLETED] As a developer on the bachelor project I would like the endpoints to have integration tests with edge cases and data content correctness tests (using ARX GUI as fasit)
[COMPLETED] Implement T-closeness(++) Privacy Model in service and Python client
[COMPLETED] Create Presentation for AI/data science forum
[COMPLETED] as a data scientist I would like metadata on the anonymization performed on a given dataset
[COMPLETED] End to end SSL Encryption between ARXaaS and clients

Goals:

- By the end of the Sprint the team wants to be able to say the product is usable by data scientist in NAV
- Test coverage on ARXaaS 70%++, Integration tests for all main endpoints with end case coverage

Suggestions for Sprint 7 backlog:

- Create presentation for AI user forum - H
- Implement T-closeness(++) Privacy Model in service and Python client
- As a data scientist I would like that ARXaaS has been stress tested and what load I can expect the system to handle
- As a developer on the bachelor project I would like good test data to write tests with
- As a developer on the bachelor project I would like the end points to have integration tests with edge cases and data content correctness tests (using ARX GUI as fasit)
- As a NAV employee in I would like a web application to anonymize data with K-anonymity and L-diversity and imported csv hierarchies
- As a data scientist I would like to create hierarchies for my dataset using the ARXaaS service

Bugs to fix:

- Return data from anonymization does not have correct attribute types

Review:

The team completed the entire backlog and are pleased with the progress made.

Retro:

The group did a Start-stop-continue session
Where each team member writes down at least one thing for each the three categories.

Start:

- Thinking deployment/shipping (applications/packages/docs/ci pipeline)
- Thinking handover in regards to code/docs
- Reaching out for assistance on hard features
- Bringing in team members on feature work (pair-program)
- Better motivation after assigned task is done
- Better planning and feedback before and after presentation/workshop
- More peer programming
- More pictures
- Writing documentation more targeted at the user

Stop:

- Winging it (Maybe small practice round)
- Not logging enough in asana when performing a task
- Not using branches in webarx, must make it easier to work in parallel
- Workshop
    - not tested
    - no dry run

Continue:

- Take ownership of the product
- Learning about ARX system
- Learning about anonymization
- Pair-programming when possible
- Reviews and merging principles
- Attendance
- Motivation
- Teamwork
- Tasks done
- Good feedback
- Customer happy
- Group still positive and working hard
- Being productive, and limiting the scope of tasks

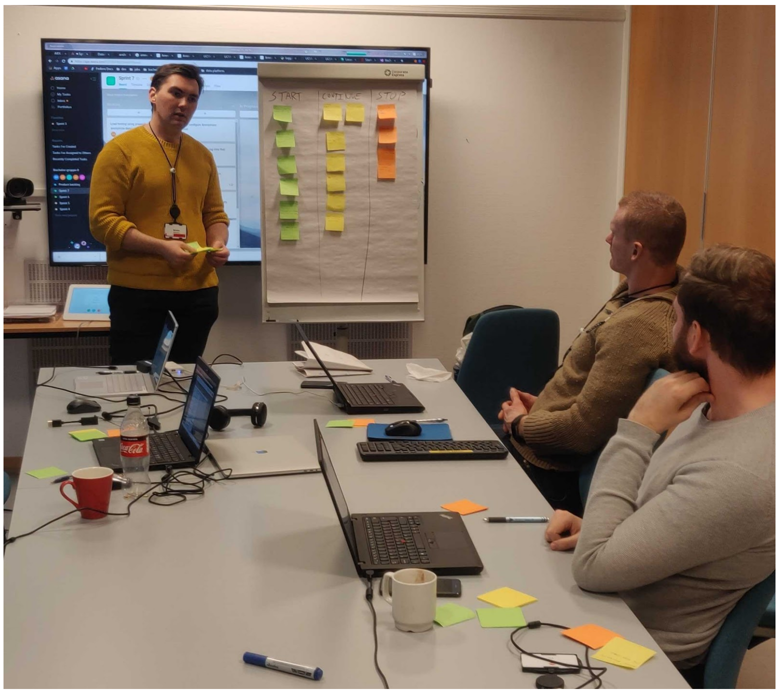


Figure 23 - Sprint retro

3.3.1.8 Sprint 7

Duration: 15.04.19 - 26.04.19
Summary: The group prepared a demo of the final presentation for Eva about the bachelor project. The group also had a presentation for the sprint review. Started implementing the different features for the Web app. All feature except for hierarchy generation are now available on the web app. The web app still needs a design overhaul. Implemented suppression limit to be taken in a parameter, as well as logging the limit used. The suppression limit is now available on both the python client and the web app. Privacy models used to anonymize the dataset are now logged as well. Prepared different datasets and hierarchies to be load tested on the service. Implemented hierarchy generation end-point on the service and is now available on the python client.
Backlog:

[INCOMPLETED] Visualization/Description of Re-identification risk (K=?)
[COMPLETED] As a NAV user I would like a intuitive webapp to analyze and anonymize datasets
[COMPLETED] ARXaaS hierarchy generation
[COMPLETED] Python wrapper hardening
[COMPLETED] Load testing using proper "hard to anonymize datasets"
[COMPLETED] As a Data scientist I would like the suppression limit to be passed as a parameter
[COMPLETED] As a data scientist, I would like ARXaaS to provide richer logging

Goals:

- The team prepares for handover on all technical deliveries - Andre
- At Least two persons has knowledge on a given feature - Sondre

Review:

The team is proud to hand over the solution to NAV IT. The teams focus will now be on the project report.

Retro:

The group did a sprint retro session for the entire project divided in:

- Positive Learnings
- Negative Learnings
- Surprises

Where each team member writes down at least one thing for each the three categories.

Positive Learnings:

- Continuous Integration worked really well (thanks, Accelerate!)

- Knowledge is well shared and spread across team members
- Sprint retro has worked well
- Solid agreement on decisions and productive discussions
- Spring Boot is a good framework for creating professional web applications
- Time spent on sprint planning makes time spent elsewhere more efficient. Time boxing worked well
- Great team
- Pair programming has worked really well
- Delivered a production worthy, whole product
- Test driven programming worked well
- Learning fullstack/professional product development
- Pair-programming
- The importance of keeping everyone up to date and engaged
- The importance of planning well ahead, and having everyone agreeing on the decisions along the way
- Scrum retro has worked well
- Software
- Share knowledge
- Continues delivery

Negative Learnings:

- Preparing for workshops, presentations has been underprioritized
- Creating software is time consuming, and cannot always be produced effectively by a single worker
- Taking time off in the middle of the semester is stressful
- Refactoring is tough
- Definition of Done and speccing is super hard, time consuming
- Estimating time requisites takes too much effort
- Learning new tech, especially Python is tough
- Creating good user stories is very demanding,
- Perhaps format could be less stale if it had been less static and repetitive. Also more knowledge on user stories from product owner, better collaboration on creating them.
- Need agile coach?
- Software takes time, nobody can do it all alone
- Prepare better for workshops and presentation

Surprises:

- Lists and maps are pretty useful
- Putting effort into aligning team is valuable
- Very little time loss/waste from unfortunate events
- Pair programming requires some scheduling
- Knowledge silos appear fast

- Effort when aligning team
- Meetings and admin takes a lot of time
- Sprint retro

3.3.2 Development tools

Asana[43] - Project management software

The team used Asana as an agile kanban board throughout the project. It proved useful for declaring the time box for sprints, dividing the sprints into user stories, and assigning the user stories as tasks for team members.

Git[44] - Version Control System

Git proved useful during the project's development phase, where multiple team members were altering the code base simultaneously. The team's knowledge of Git was already at an acceptable level prior to the project's initialization. As a result, Git contributed to allow for seamless and continuous cooperation and merging of the code base. From the perspective of *continuous integration*, Git shone bright during the development and report writing of the project.

GitHub[45] - Version Control Host

The entire code base of the solution is backed up in GitHub's cloud storage. *Pull requests* was perhaps the most useful feature that GitHub offered to the team. *Pull requests* place certain requisites that must be fulfilled before a merge can be made to the master branch. This way the team could assert the quality of the code by reviewing each other's work, and even utilize addons that could further verify the code quality before a merge. The team also utilized some of the useful GitHub Apps and Third Party Applications, namely Travis, CodeClimate and Snyk, and they are covered in the following tool declarations.

Travis CI[46] - *Continuous Integration* (CI) Service

Travis provided the team with a virtual machine that could run, test and deploy code following the desired sequence and configuration defined by the team, invoked by changes to the project's master branch. The invocation of the functionality could also be controlled, for instance so that all releases to the project's master branch were tested, but only tagged releases were deployed. This way, Travis' functionality proved extremely useful to the team in regards to *continuous delivery*.

CodeClimate[47] - Static code analysis service

One of the steps involved in Travis' *CI/CD* pipeline included running tests on every *pull request* to the project's master branch. CodeClimate would read the results from those tests and report on various discrepancies in the code with distinct feedback. For instance, the GitHub repositories were configured so that *pull requests* with failed

[43] What is Asana and its purpose - https://asana.com/developers/documentation/examples-tutorials/overview
[44] Git - https://git-scm.com/
[45] Github - https://github.com/
[46] Travis, *Continuous Integration* platform - Core Concepts https://docs.travis-ci.com/user/for-beginners/
[47] CodeClimate - https://codeclimate.com/quality/

tests or decreased *test coverage* would be rejected until acted upon. CodeClimate would also offer a visual presentation of the result, including information about *test coverage* percentage before and after, and a grading for code maintainability based on duplication, cyclomatic complexity, cognitive complexity, and structural issues. Read chapter 4.4.2 Static code analysis, involving static code analysis in the test documentation for specific clarification.

Snyk[48] - Analyzes the dependencies used in the project for known vulnerabilities.

Notified the team of any vulnerabilities and potential risks in the repo through automated updates to communications channels like Slack. Read more about SNYK and its' purpose in the project in chapter 4.4.3 Vulnerability analysis, of the test documentation.

Slack[49] - Professional team communication application

Slack offered the team a tidy and organized instant messenger experience, optimized for groups and organizations. The team's most common use case for Slack was quick coordination and information sharing.

JetBrains IntelliJ[50] - Integrated Development Environment [Java]

The team utilized Intellij as the GUI for Java programming, and it was primarily used for developing ARXaaS. In short, it offered the team powerful static code analysis and useful integrations with Git (VCS) and Maven (build/dependencies).

JetBrains PyCharm[51] - Integrated Development Environment [Python]

PyCharm is IntelliJ's counterpart for development in Python, and was utilized for building PyARXaaS. Its' integration with pypi (Python Package Index) allowed for seamless access to critical Python tools like Jupyter Notebooks.

3.3.3 Lessons learned during development

The team learned that setting up a Continuous Integration pipeline and developing in increments was very productive. This ensured that by the end of every sprint a stable product was delivered, this also made developing new features more efficient.

Working agile with Scrum was highly productive and ensured that the desired functionality of the product was consistently delivered. The continuous communication with the product owner ensured strong confidence within the team, while also ensuring that the end solution fulfilled the specifications. One of the biggest surprises was the Sprint retro meetings worked so well. The team had an opportunity to reflect on the last sprint and improve the next Sprint.

[48] Snyk - https://snyk.io/
[49] Slack - https://api.slack.com/#read_the_docs
[50] Intellij- https://www.jetbrains.com/idea/
[51] PyCharm - https://www.jetbrains.com/pycharm/

3.4 Product specification

Throughout the project, feedback was continuously exchanged with the product owner as a result of the team's agile working strategy. In order to fulfill the product owner's requests, the team needed to continuously verify, reconstruct and re-prioritize tasks, shaped as user stories. The user stories would eventually form the core of our product specification, which will be the topic at hand for this subchapter.

3.4.1 Main specifications

The system will provide access to anonymization tools for data scientists at NAV IT. A data scientist should be able to anonymize tabular datasets based on user-specific configurations. Configurability includes privacy models, column attribute types and transformation models that determine how much data will be lost in the resulting anonymized dataset.

An example use case could involve a workflow where the data scientist is manipulating a dataset, and requires dynamic analysis of the data's anonymity metrics. Another use case could involve integrating the system in a data pipeline to provide data analytics and anonymization capabilities.

Deliverables

- Python Package - working title: PyAaaS
  - Python package wrapper providing abstracted access to the backend service.
- Web Service - working title: AaaS
  - Java Spring web service.

3.4.1.1 System Diagram

Jupyter notebook is a common user interface among data scientists, and will be a important platform for the system to support. In a Jupyter Notebook a data scientist that wishes to anonymize or analyze a dataset will import a Python package which wraps and abstracts the backend service. The backend service utilizes the ARX library and Spring framework to deliver the anonymization and analytics functionality as a web service. The service is packaged as a Docker container.

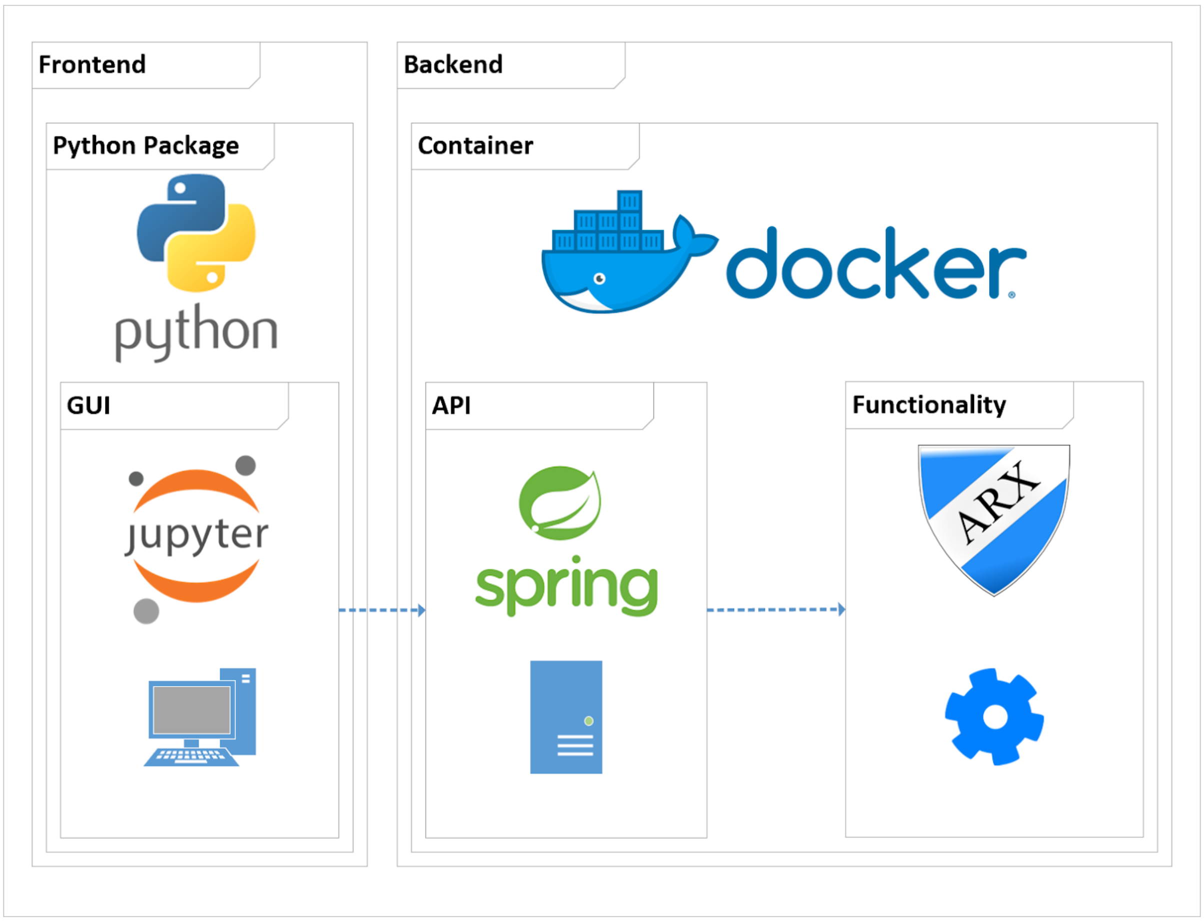


Figure 24 - System diagram

3.4.1.2 Requirements

The collection of system requirements defined in collaboration with the client(NAV IT - AI lab)

3.4.1.2.1 Functional requirements

- The system will provide the ability to complete data anonymization with the provided user configurations on tabular datasets.
- The system will provide the ability to analyze re-identification risks on tabular datasets.
- The system will provide the ability to configure the Privacy Models to use in the anonymization.
- The system will provide the ability to configure data Attribute Type to use in the anonymization.
- The system will provide the ability to configure the Transformation Models to use in the anonymization.
- The system will provide the ability to produce a visual presentation of data anonymity metrics.
- The system will provide the ability to compare data from before and after data anonymization.

- The system will be able to consume data in a variety of formats including (pandas.DataFrame, path to csv file, url to data resource, csv string, JSON).
- The system will be able to deliver the anonymized dataset in a variety of formats including (pandas.DataFrame, csv file, JSON).
- The system will be able to deliver metrics about the anonymization in a variety of formats including (pandas.DataFrame, csv file, JSON, Data Package).
- The system will provide the ability to produce data package metadata regarding the anonymization process that has been completed on the dataset and the relevant metrics.

3.4.1.2.2 Non-Functional Requirements

3.4.1.2.2.1 Software Requirements

- The client has requested that the team uses the ARX anonymizer library to implement anonymization functionality.
- The client has required that resulting anonymization process has to be more efficient than the previous and reduce the hours that are spent doing this manually.
- The client has required that the project is published as an *open source* project with an MIT licence.
- The client has required that the system backend will be packaged as a docker image so the service can be deployed to the NAIS/Kubernetes platforms.
- The client has required that the team develop a Python package to “wrap” the web service, it will provide easy integration and interaction between the web service and data scientist tools and processes.
- The client has requested that the Python package has to be designed for use in a Jupyter notebook.
- The client has required that the system will utilize end to end encryption for data in transit, to and from the web service backend.

3.4.1.2.2.2 Design Decisions

Design decisions made by the team in collaboration with the customer to achieve the stated goal of the system.

- English will be the main language used for both the documentation and programming to make it easier for the team to deliver on the open source requirement from the client.
- The team has decided to utilize Java as its runtime environment for the backend service. The ARX library that the client has requested to be used is packaged as a Java JAR file. Using Java was a logical choice.
- The team has decided to use a service architecture to decouple the different logical components of the project. A service architecture will also deliver on the clients wish to be able to scale the system dynamically according to use.

- The team has decided to utilize Spring as its backend framework to deliver a web service in accordance with the service architecture. Spring is the defacto standard for Java web applications and has great libraries for development of secure, scalable web applications.

3.4.1.3 User Stories

User stories are one of the primary development artifacts for Scrum project teams. A user story is a very high-level definition of a requirement, containing just enough information so that the developers can produce a reasonable estimate of the effort to implement it.

The team wrote the user stories at the beginning of the project. This document was written with the expectation that it would change in the future, since the team decided on implementing an agile workflow. The user stories has changed based on the feedback received from the product owner.

Actor	**Story**	**Priority**
Data Scientist	As a data-scientist, I would like to easily **visualize the anonymization metrics** for anonymized datasets Reasoning: Visualization is a powerful tool to get an understanding of complex data.	High
Data Scientist	As a data-scientist, I would like to **analyze the anonymization metrics (re-identification risks)** of my dataset Reasoning: Getting metrics of the anonymization level of a dataset is necessary to judge how safe the dataset is to use in production, and/or if the dataset should be anonymized further.	High
Data Scientist	As a data scientist, I would like to have a single source where to lookup the **documentation for the PyAaaS package** (AaaS Python wrapper package). Reasoning: To facilitate efficient use of the Python package it is critical to make available up-to-date documentation of both the package API and tutorials for common use cases.	Medium
Data Scientist	As a data scientist, I would like to be able to **configure the Privacy Models** to be used when anonymizing my dataset	High
Data Scientist	As a data scientist, I would like to be able to **configure the Transformation Models** to be used when anonymizing my dataset.	High

Data Scientist	As a data scientist, I would like to be able to use **K-Anonymization as a Privacy Model** for my dataset.	High
Data Scientist	As a data scientist, I would like to be able to **set a global transformation scheme for all records** in a column/field.	High
Data Scientist	As a data scientist, I would like to be able to **set a local transformation scheme for a column/field**. Meaning a unique transformation scheme for each individual row or subset of rows in a column/field.	Low
Data Scientist	As a data scientist, I would like to be able to use a **Value Generalization hierarchy as a Transformation Model** for a column/field.	High
Data Scientist	As a data scientist, I would like to be able to use **random sampling as a Transformation Model** for a column/field	Medium
Data Scientist	As a data scientist, I would like to be able to use **attribute suppression as a Transformation Model** for a column/field	Medium
Data Scientist	As a data scientist, I would like to use **microaggregation as a Transformation Model**.	Medium
Data Scientist	As a data scientist, I would like to use **Top- and bottom-coding as a Transformation Model.**	Medium
Data Scientist	As a data scientist, I would like to use **Categorization as Transform Model for a column/field.**	Medium
Data Scientist	As a data scientist, I would like to **identify rows affected by lowest risk** in a dataset.	Low
Data Scientist	As a data scientist, I would like to determine the **Lowest prosecutor re-identification risk.**	Low
Data Scientist	As a data scientist, I would like to determine **highest prosecutor re-identification risk.**	High
Data Scientist	As a data scientist, I would like to **identify rows affected by highest risk.**	High
Data Scientist	As a data scientist, I would like to **determine average prosecutor re-identification risk.**	High
Data Scientist	As a data scientist, I would like to **determine fraction of unique records.**	High

Data Scientist	As a data scientist, I would like to be able to **use K-map as a Privacy Model** for my dataset.	Low
Data Scientist	As a data scientist, I would like to be able to use **Average risk as a Privacy Model** for my dataset.	Low
Data Scientist	As a data scientist, I would like to be able to use **Population uniqueness as a Privacy Model** for my dataset.	Low
Data Scientist	As a data scientist, I would like to be able to use **Sample uniqueness as a Privacy Model** for my dataset.	Low
Data Scientist	As a data scientist, I would like to be able to use **δ-Disclosure privacy as a Privacy Model** for my dataset.	Low
Data Scientist	As a data scientist, I would like to be able to use **β-Likeness privacy as a Privacy Model** for my dataset.	Low
Data Scientist	As a data scientist, I would like to be able to use **δ-Presence privacy as a Privacy Model** for my dataset.	Low
Data Scientist	As a data scientist, I would like to be able to use **Profitability privacy as a Privacy Model** for my dataset.	Low
Data Scientist	As a data scientist, I would like to be able to use **Differential privacy as a Privacy Model** for my dataset.	Medium
Data Scientist	As a data scientist, I would like to be able to use **ℓ-Diversity as a Privacy Model** for my dataset	High
Data Scientist	As a data scientist, I would like to be able to **set presets for anonymization** e.g loss percentage, min/max risk of prosecution	Low
Data Scientist	As a data scientist, I would like to be able to **verify whether an anonymized dataset** has been anonymized from an original dataset	Low

Actor	**Story**	**Priority**
Data Engineer	As a data engineer supporting the AaaS service in NAV, I would like to be able to **deploy the service to a docker container** environment.	high
Data Engineer	As a data engineer supporting the AaaS service in NAV, I would like **continuous information about the build status of the AaaS web service** source code	medium
Data	As a data engineer supporting the AaaS service in NAV, I would	medium

Engineer	like a **single source/location for documentation,** for setup and deployment of the AaaS service.	🟧
Data Engineer	As a data engineer supporting the AaaS service in NAV, I would like a single **source/location for the AaaS projects JavaDoc**	low
Data Engineer	As a data engineer supporting the AaaS service in NAV,I would like to have **logging available from the AaaS service.**	high

3.4.1.4 System Restriction

This section explains the system and client defined restriction. In this section the team defines the limitations of the system to be developed.

3.4.1.4.1 Security

The client has requested that the system use end-to-end encryption between the backend service and consumers of the service (Python package, Third-party applications). The team has decided to use TLS/SSL provided by the Spring framework.

3.4.1.4.2 Data Storage/Cache

The developed system cannot store or implement caching due to the sensitive nature of the data used.

3.4.1.4.3 Accessible API

Our API must follow RESTful guidelines and strive to provide endpoints that allow for seamless interaction with the ARX library.

3.4.1.5 Additional Requirements for System Construction

Non-functional requirements defined by the development team in cooperation with the client. These requirements are meant to improve the quality and maintainability of the solution.

3.4.1.5.1 Process Requirements

3.4.1.5.1.1 Continuous Integration/Continuous Delivery (CI/CD)

CI platform
The system uses Travis CI and Code Climate as Continuous Integration tools. Travis ensures that the codes are tested, while code climate checks the code quality and test coverage before being pushed to the repository. Along with Travis and Code Climate, the team uses GitHub with merge rules to ensure that the master build stays stable.

Each new iteration of the master build is packaged into a jar file, which would be packaged again into a Docker image. This way we will have different stable builds that can be deployed easily as a Docker container.

Version Control System (VCS)
Travis CI along with GitHub Merge rules maintains the stability of each version before a release.
Github Merge rules does not allow directly pushing to the master build, along with not allowing to push to the repository unless all test class passes. Each time there is a new implementation a new branch needs to be made. This new branch is then tested before being pushed to the repository, which is then finally merged to the master build.
Travis CI instantiates a docker container that runs the build being pushed along with all the test classes. If a build passes Travis will then allow the build to pushed.

Static Code Analysis
Code Climate is used to ensure the maintainability and test coverage of the codes written. Travis generates a test report using Jacoco, which is then forwarded to Code Climate by using a unique ID. Code Climate reads through the report and generates a grade for test coverage of our system. Code Climate is directly link to the systems GitHub repository, granting access to check the quality of the codes written. Based on the quality of the code a grade will be generated for the maintainability of our system.

3.4.1.5.1.2 System Development Framework

The team is developing the system following the Scrum framework.

3.4.1.5.2 Technical Requirements

3.4.1.5.2.1 System Packaging

Backend Service
- Docker Image

Python Wrapper Package
- Python package wheel and source distribution

3.4.1.6 Additional Requirements for Documentation

Additional non-functional requirements defined by the team in cooperation with the client. These requirements are meant to improve the quality of the documentation.

3.4.1.6.1 System Documentation

The system backend service and python package will be delivered with documentation for the corresponding to the intended usage.

Backend Service Documentation
- Setup and deployment tutorial
- System Javadoc

Python Wrapper Package Documentation
- Installation
- Common usage tutorial
- Examples (notebooks)
- API docs

3.4.2 Stretch goals

The following stretch requirements are wishes from the customer that the development team would try to achieve if there was time left after delivering on the main requirements.

- The system will be able to auto generate a hierarchy level based on the column attribute type.
- Provide specific Transformation Model hierarchies for NAV specific use cases (eg. Norwegian geographical hierarchies, Norwegian zip code hierarchies).
- Provide an alternative web frontend that provides a lower barrier to entry, and a more user friendly interaction.
- Grafana dashboard for surveillance of the anonymization service.

As the main stretch goal, the product owner wished for a simplified way to access the ARX functionality, with a low requisite to user competence. It was therefore decided to implement an interactive React[52] web application. By taking advantage of the flexible REST API provided by the ARXaaS service, the team decided that this was easily feasible. The client was to offer the possibility to both analyze and anonymize data, without the need to install software on their local machine, and with a lower requisite to competence. To fulfill this stretch goal, the team ended up creating the WebARXaaS product which is documented in detail in chapter 5.5 WebARXaaS.

3.4.3 Delivery

On 26.04.19 the team did the handover of ARX as a service to NAV IT. The solution delivery provided all the functionality initially asked for by the product owner as well as the project stretch goal.

The delivered features where:
- ARXaaS REST web service
- PyARXaaS
- WebARXaas

[52] React introduction - https://reactjs.org/docs/getting-started.html

3.5 Conclusion process documentation

After many hours spent on this project the team is proud of the results. During this process the team delivered new and improved functionality to the product owner as well as creating a open source project anyone can use. The team is proud to deliver the solution that is beyond the project specifications and stretch goals.

The team can reaffirm that the Scrum development framework is well suited to timeboxed projects like this. Continuous integration and Continuous delivery showed themself as incredibly helpful practices that helped elevate the resulting product. Development of CI/CD pipeline for the different products likely saved the project hours of manual work and debugging. Developing new features fully according to the feature slicing practice meant the client could provide feedback contrously, again a very important practice the team members will continue to use on other projects.

During the course of this project all teams members have learned new skills and technologies that can be used and developed further during work in the years to come.
The team also learned a lot from the product owners feedback and advice. The teams is looking forward to developing the solution further this summer.

4 Test documentation

The purpose of this chapter is to give the reader a detailed description of the testing completed during the project. The document describes the purpose of the tests, tools used for testing, and how the tests are implemented. This document is written with the expectation that the reader has basic programming and testing knowledge, and has read the process documentation.

4.1 Goal

The goal for these tests is to ensure that the service and the clients work according to the clients requirements. The test should also catch any regressions in the products. This is also to ensure that in an event that a feature fails, the right error message is returned with a detailed explanation of what went wrong and, if applicable, how to correct the error.

4.2 Tools

These are the tools the team used for testing:

- **Postman**
  - The team used postman to craft network requests and send it towards the service end-points, in order to quickly test the service.
- **Travis**
  - The team used travis to automate the continuous delivery pipeline. Each new branch pushed to the repository is tested in a virtual environment and has to pass before being allowed to be merged in to the master branch.
- **Code Climate**
  - Code climate was used as a static code analyser that showed test coverage as well the maintainability the code.
- **Snyk**
  - Snyk is used to ensure that the dependencies used in the project does not have any vulnerabilities
- **JUnit**
  - Used for unit testing for the service.
- **Spring boot starter test**
  - Used to test our service web environment. Edge case testing to ensure that the correct error messages shows and for the integration testing.
- **Unittest(Python)**
  - Module for making unit test for python libraries
- **Pytest**
  - Used in Travis to run the unittest for Python and retrieve the generated code coverage from Coverage.py
  - Makes it easy to write small tests, yet scales to support complex functional testing for applications and libraries.

- **Coverage.py**
    - Python library used to measure test coverage of Python programs.
- **Locust**
    - Locust is an easy-to-use, distributed, user load testing tool. It is intended for load-testing web sites (or other systems) and figuring out how many concurrent users a system can handle.

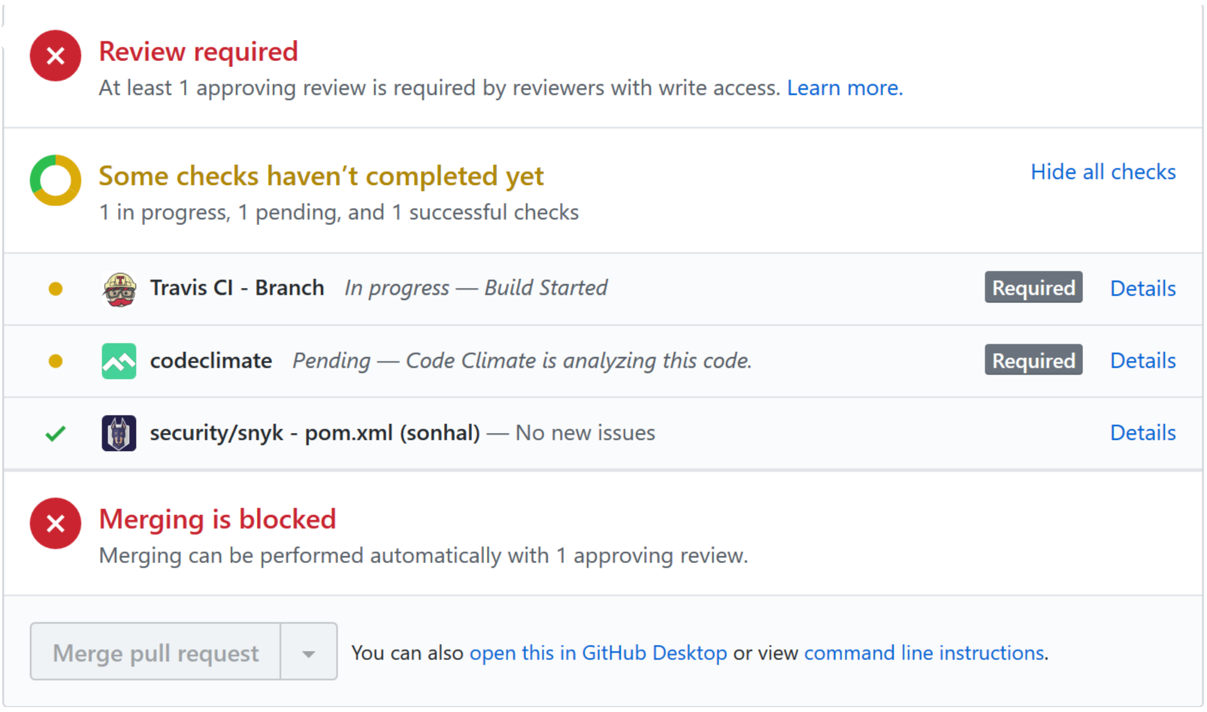
4.3 Planning

From the start of the project, the team used unit tests to explore the functionalities available in the ARX libraries.This ensured a good understanding of the feature and made it easier to integrate them in the service.

During development of the solution the project team decided to use test driven development on both the service and client products. The goal was to ensure good test coverage and test quality through the project. The test coverage was enforced by test coverage tools such as Code Climate, the plan was to unit test each new method and make it pass, before moving on with an integration test and thereafter system testing of the end-points. The assumption being that having a stable service endpoint will make it possible to work on the client-side in parallel. The project team decided on a test plan to ensure a stable build is always produced before merging with the master branch in the version control host. The project team decided on which test methods to use, test method naming, version control host merge rules and tools to use to enforce the merge rules.

4.4 Execution

Its is important to ensure that a stable build is produced at the end of each Sprint. Therefore testing was continuously done throughout the project. For each new feature implemented, a unit test must follow before being allowed to be merged to the master branch. Integration testing is done after all features in a sprint is implemented. Finally system testing and edge case testing was done to ensure that the service works properly and in an event of an error, show a detailed explanation of what occurred as well as make sure the correct error is shown.


Review required

At least 1 approving review is required by reviewers with write access. Learn more.

Some checks haven't completed yet

1 in progress, 1 pending, and 1 successful checks

Hide all checks

●	Travis CI - Branch *In progress — Build Started*	Required	Details
●	codeclimate *Pending — Code Climate is analyzing this code.*	Required	Details
✓	security/snyk - pom.xml (sonhal) — No new issues		Details

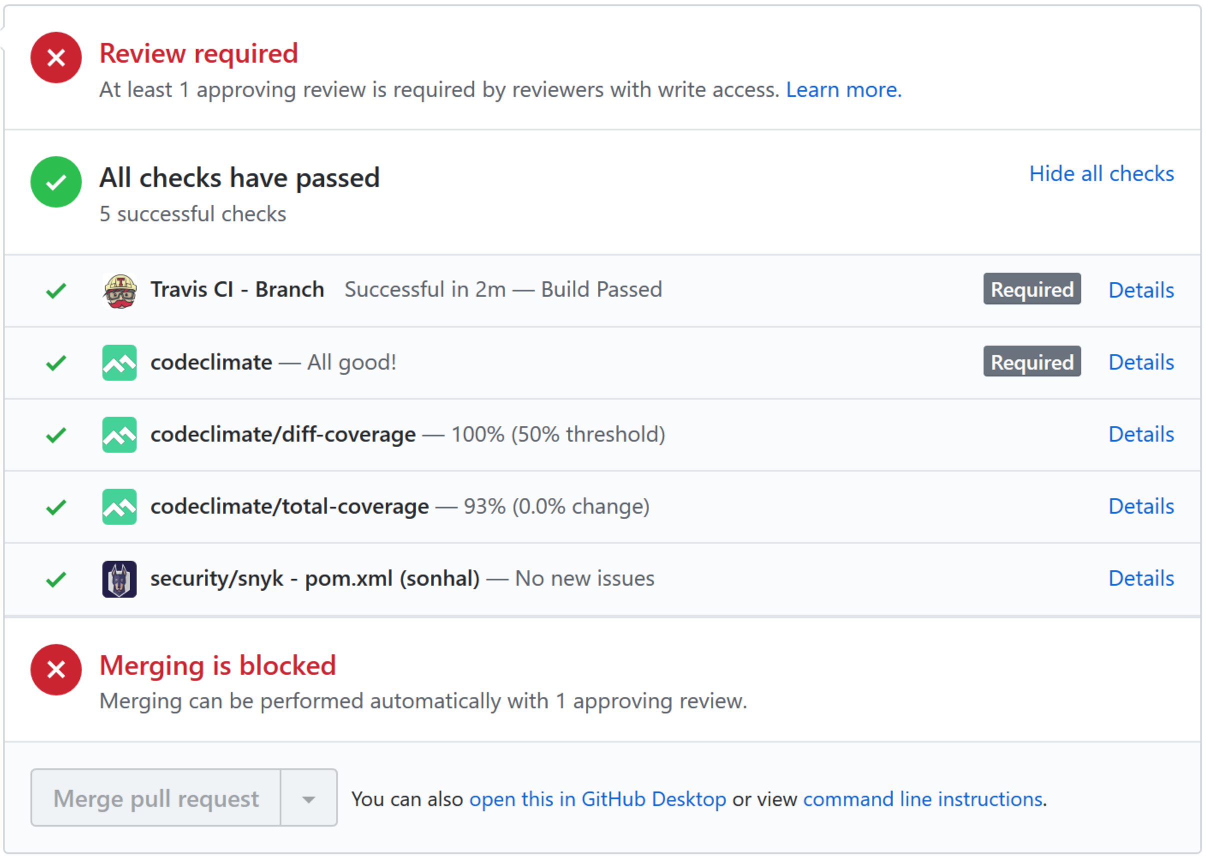
Merging is blocked

Merging can be performed automatically with 1 approving review.

Merge pull request

You can also open this in GitHub Desktop or view command line instructions.

Figure 25 - Checking the branch before being approved to merge


Review required

At least 1 approving review is required by reviewers with write access. Learn more.

All checks have passed

5 successful checks

Hide all checks

✓	Travis CI - Branch Successful in 2m — Build Passed	Required	Details
✓	codeclimate — All good!	Required	Details
✓	codeclimate/diff-coverage — 100% (50% threshold)		Details
✓	codeclimate/total-coverage — 93% (0.0% change)		Details
✓	security/snyk - pom.xml (sonhal) — No new issues		Details

Merging is blocked

Merging can be performed automatically with 1 approving review.

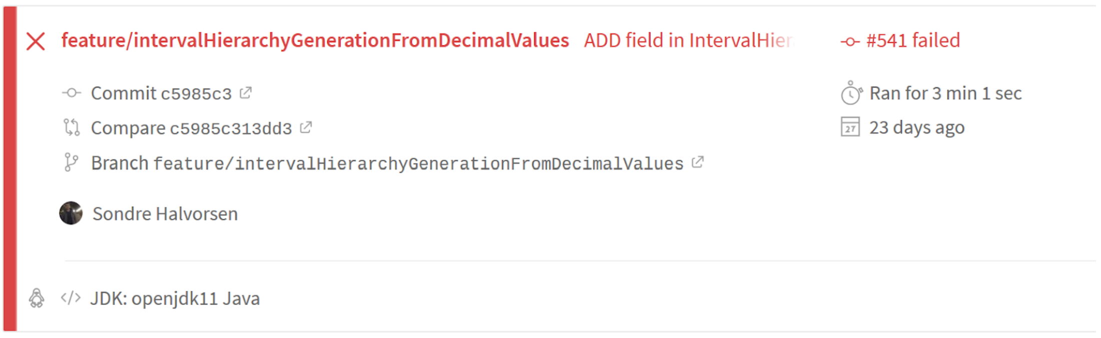
Merge pull request

You can also open this in GitHub Desktop or view command line instructions.

Figure 26 - All checks passed

4.4.1 Travis

For every push to Github a travis job is started, in this job a virtual machine will run the program and the tests. Each test must pass for travis to give a passing grade, this passing grade is used to restrict merging of unstable builds to the version control host.


feature/intervalHierarchyGenerationFromDecimalValues ADD field in IntervalHier

#541 failed

Commit c5985c3

Compare c5985c313dd3

Branch feature/intervalHierarchyGenerationFromDecimalValues

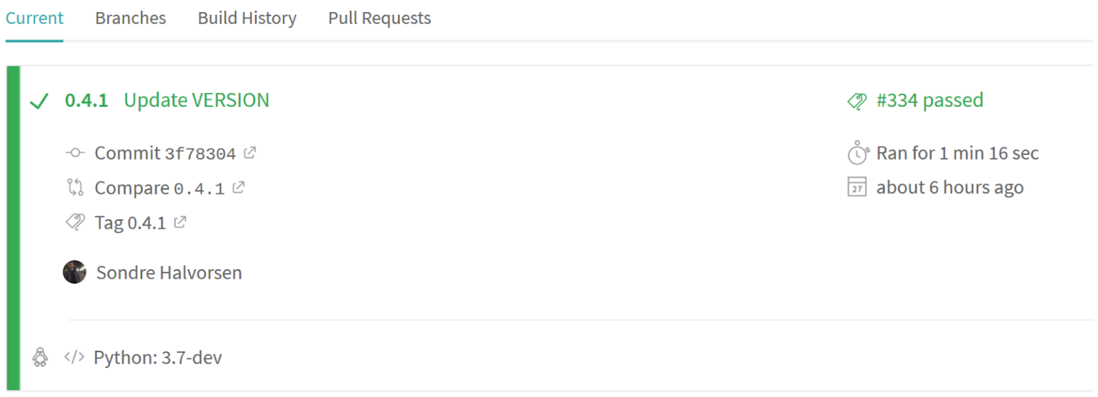
Ran for 3 min 1 sec

23 days ago

Sondre Halvorsen

JDK: openjdk11 Java

Figure 27 - Failed Travis build



0.4.1 Update VERSION

#334 passed

Commit 3f78304

Compare 0.4.1

Tag 0.4.1

Ran for 1 min 16 sec

about 6 hours ago

Sondre Halvorsen

Python: 3.7-dev

Figure 28 - Passed Travis build

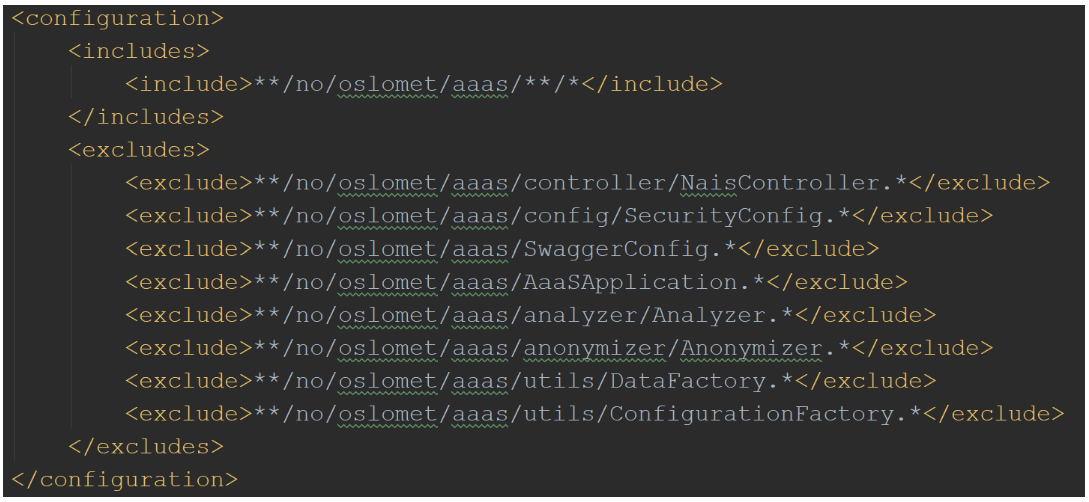
A set of scripts has been written to instruct travis on what to do upon a passing job. One of these scripts is running Code Climates test reporter. The test reporter publishes the test coverage report generated to Code Climate.

```
after_script:
 - JACOCO_SOURCE_PATH=src/main/java ./cc-test-reporter format-coverage -d
  ./target/site/jacoco/jacoco.xml --input-type jacoco

 - ./cc-test-reporter upload-coverage -d
```

Figure 29 - Travis script running Code Climate test reporter

For ARXaaS, Jacoco Maven plugin was used to generate a test coverage report. Jacoco has settings that tells it which classes to ignore when creating the test coverage report. These classes are ignored because they are not meant to be tested.

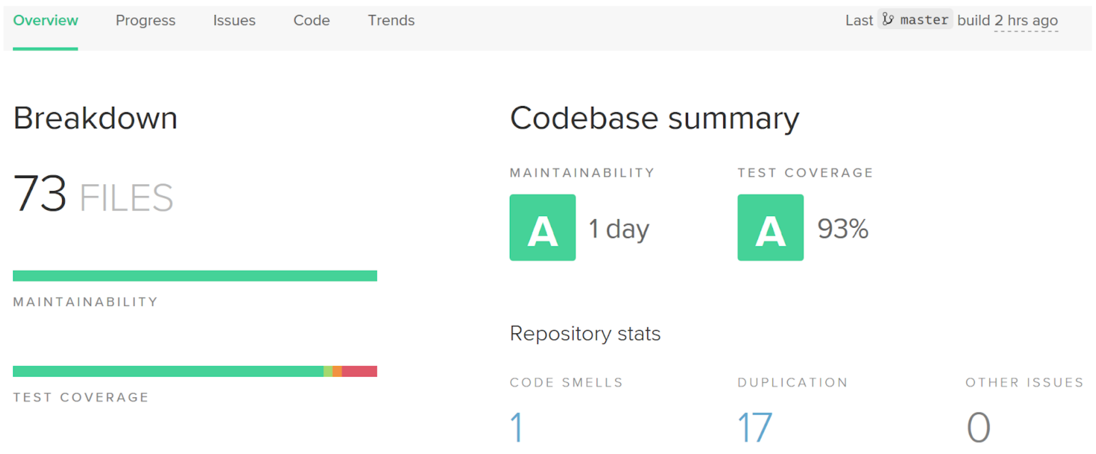


```
<configuration>
    <includes>
        <include>**/no/oslomet/aaas/**/*</include>
    </includes>
    <excludes>
        <exclude>**/no/oslomet/aaas/controller/NaisController.*</exclude>
        <exclude>**/no/oslomet/aaas/config/SecurityConfig.*</exclude>
        <exclude>**/no/oslomet/aaas/SwaggerConfig.*</exclude>
        <exclude>**/no/oslomet/aaas/AaaSApplication.*</exclude>
        <exclude>**/no/oslomet/aaas/analyzer/Analyzer.*</exclude>
        <exclude>**/no/oslomet/aaas/anonymizer/Anonymizer.*</exclude>
        <exclude>**/no/oslomet/aaas/utils/DataFactory.*</exclude>
        <exclude>**/no/oslomet/aaas/utils/ConfigurationFactory.*</exclude>
    </excludes>
</configuration>
```

Figure 30 - Jacoco include/exclude configuration

4.4.2 Static code analysis

The project has utilized Code Climate as its primary static code analysis tool in the *CI* pipeline. Code Climate jobs were invoked by every push to the Version Control System, triggering a scan of the code and generation of a detailed report. Maintainability and test coverage is graded based on cyclomatic complexity, cognitive complexity, duplication, and other structural issues. This subchapter covers explanations of the mentioned strategies that Code Climate uses for its' static code analysis. See Code Climate's documentation[53] for more on the setup and the use cases of the service.



Breakdown

73 FILES

MAINTAINABILITY

TEST COVERAGE

Codebase summary

MAINTAINABILITY	TEST COVERAGE
A 1 day	A 93%

Repository stats

CODE SMELLS	DUPLICATION	OTHER ISSUES
1	17	0

Figure 31 - Code Climate dashboard

[53] CodeClimate documentation - https://docs.codeclimate.com/docs/maintainability

Cyclomatic complexity[54]
Metric for the complexity of a program based on the quantity of independent paths available during runtime of the program's source code. Cyclomatic complexity is primarily dependent on the amount of if-else statements and loops. The amount of these statements directly affects the code's testability, following the logic that too many traversable paths is equivalent to less testable code. Code Climate processes the code with an algorithm to generate a *control-flow graph*[55] based on the sheer amount of traversable paths. Based on the control-flow graph Code Climate can generate a score on cyclomatic complexity.

Cognitive complexity[56]
Metric for code complexity from the perspective of readability. Code Climate interprets code flow, intuitive syntax and code structure through an algorithm to analyze the understandability and readability of the code.

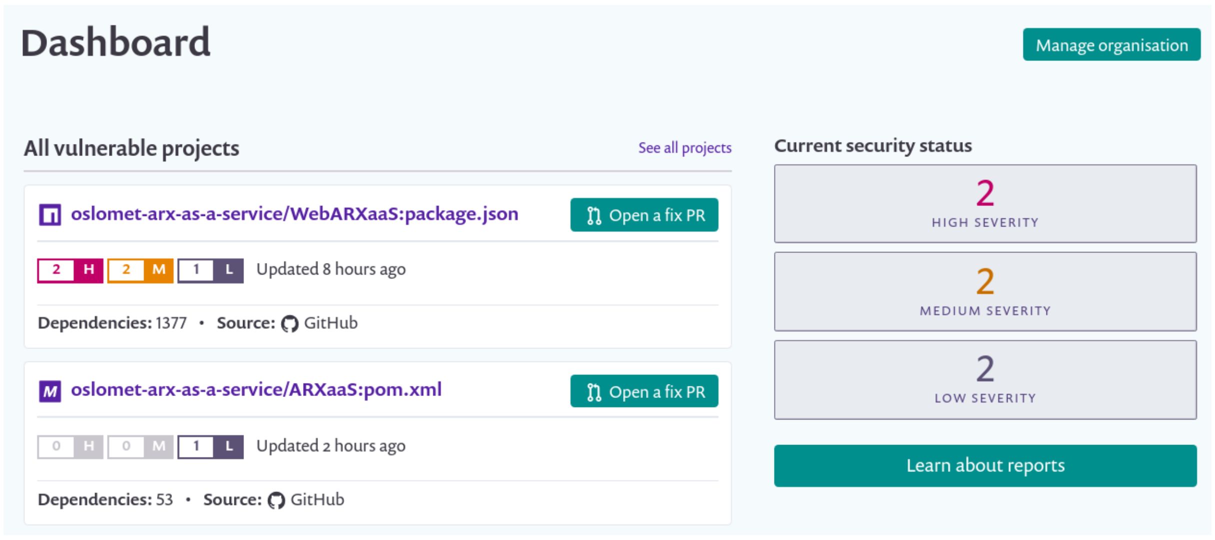
Duplication[57]
Duplication is quite simple; Code Climate looks for duplicate code blocks. With the help of a simple algorithm to point out duplication, Code Climate can provide helpful suggestions of relatively simple ways to improve the code quality, for instance suggesting to move duplicate code to a public function.

4.4.3 Vulnerability analysis

To mitigate the risk of introducing vulnerabilities or malicious code through product dependencies, vulnerability analysis is employed on all project products. Vulnerability analysis is the practice of scanning through a codebase dependency graph and check the dependencies against a database of known vulnerabilities.

Snyk is a open Source security platform, used to search the dependencies used in ARXaaS, PyARXaaS and WebARXaaS for known vulnerabilities. A report is sent to the project team when a vulnerability is detected. Vulnerabilities can usually be fixed by updating it to the newer version. In a case where an update doesn't fix the vulnerability, the project team will look at the dependency and how it affects the project. Depending on the extent of the effect, the team will either leave it be or completely replace the dependency.

[54] Cyclomatic complexity - https://en.wikipedia.org/wiki/Cyclomatic_complexity
[55] control-flow graph - https://en.wikipedia.org/wiki/Control-flow_graph
[56] Cognitive complexity - https://docs.codeclimate.com/docs/cognitive-complexity
[57] Duplication - https://docs.codeclimate.com/docs/duplication-concept


Dashboard

Manage organisation

All vulnerable projects

See all projects

oslomet-arx-as-a-service/WebARXaaS:package.json

Open a fix PR

2 H 2 M 1 L Updated 8 hours ago

Dependencies: 1377 • Source: GitHub

oslomet-arx-as-a-service/ARXaaS:pom.xml

Open a fix PR

0 H 0 M 1 L Updated 2 hours ago

Dependencies: 53 • Source: GitHub

Current security status

2 HIGH SEVERITY

2 MEDIUM SEVERITY

2 LOW SEVERITY

Learn about reports

Figure 32 - Snyk dashboard

4.5 Test phases

Test Design

Test design is a process that describes "how" testing should be done. It includes processes for the identifying test cases by enumerating steps of the defined test conditions. The testing techniques defined in test strategy or plan is used for enumerating the steps.

Unit testing

Unit tests are tests that test individual units or components in the system in isolation. A unit could be a class or a even a stand alone function.

Definition by ISTQB[58]
Component testing: The testing of individual software components.

Integration testing

Integration tests that tests the level where individual units of the system are combined. The purpose is to expose errors in the interactions between system units.

Definition by ISTQB
integration testing: Testing performed to expose defects in the interfaces and in the interactions between integrated components or systems.

System testing

System tests are tests that verify that the whole system works to specification.

[58] ISTQB - https://www.istqb.org/

Definition by ISTQB system testing: The process of testing an integrated system to verify that it meets specified requirements.

Acceptance testing

Acceptance tests are test that verify the systems compliance with business requirements. These are the tests that should verify that the user of the system is receiving the requested value.

Definition by ISTQB
acceptance testing: Formal testing with respect to user needs, requirements, and business processes conducted to determine whether or not a system satisfies the acceptance criteria and to enable the user, customers or other authorized entity to determine whether or not to accept the system.

Performance testing

Performance test are tests that intend to determine how a system performs. Important variables that are measured are:

- responsiveness
- stability

The team has primarily utilized these performance testing types:

- Load testing A type of performance test where the systems behaviors under expected load is tested.

- Stress testing A type of performance test where the system behavior under extreme load is tested

  Due to the user requirements and expected usage of the system spike and endurance testing was not completed.

  - Endurance testing The ARXaaS system does not use a database that could become overloaded or use other external services that could degrade over time.
  - Spike testing The ARXaaS system is deployed in a container orchestration service (NAIS) where spikes are managed by the service.

4.6 ARXaaS

The service has been unit tested, and integration tested using JUnit 4 and 5. System testing was done by using Spring boot starter test.

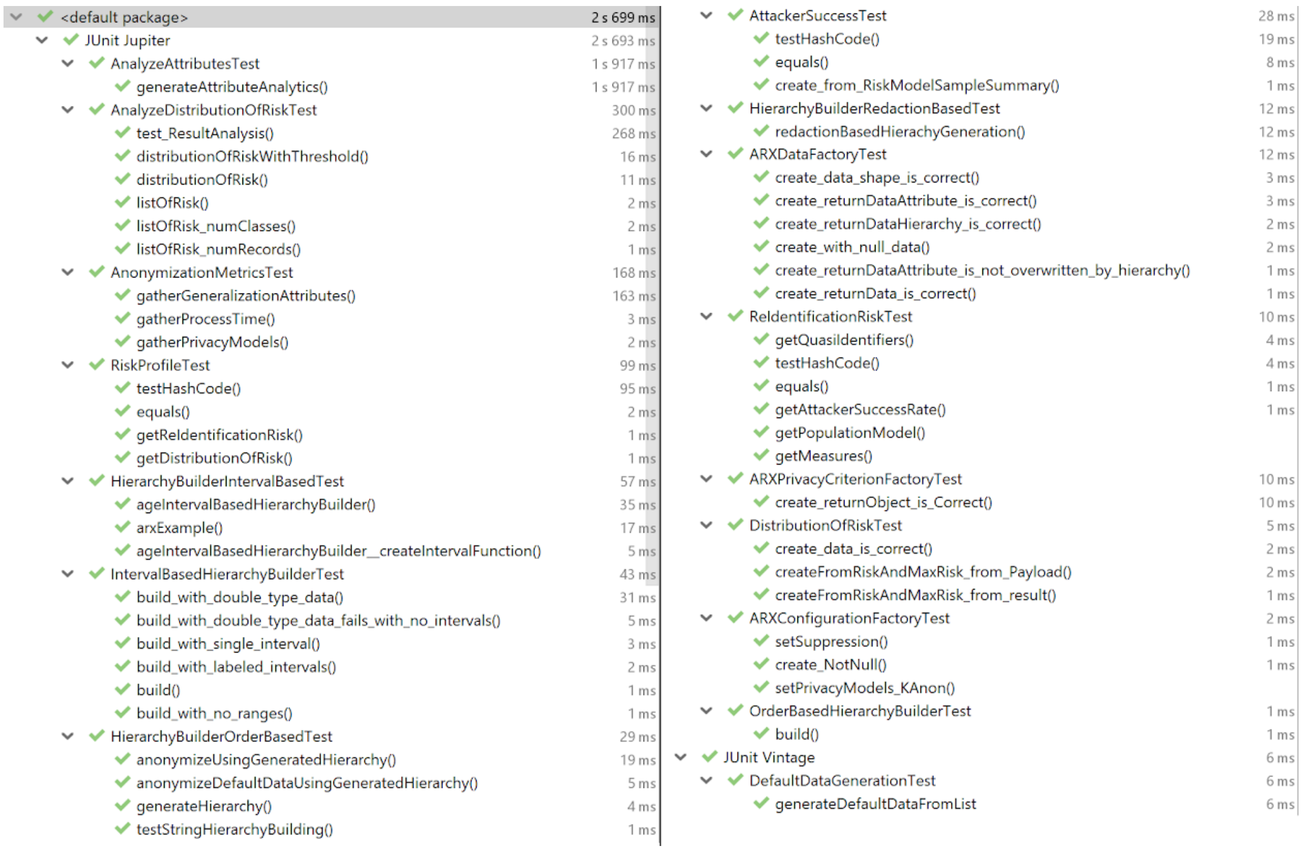
4.6.1 Unit testing

Each method that integrates a feature from the ARX library is unit tested. Along with these integrated methods, all the models and the most important components of the service has also been unit tested.

```java
@Test
void create_returnData_is_correct(){
   ARXDataFactory dataFactory = new ARXDataFactory();
   Data data = dataFactory.create(testPayload);
   DataHandle handle = data.getHandle();
   List<String[]> actual = new ArrayList<>();
   handle.iterator().forEachRemaining(actual::add);

   String[][] rawData = {{"age", "gender", "zipcode" },
           {"34", "male", "81667"},
           {"35", "female", "81668"},
           {"36", "male", "81669"},
           {"37", "female", "81670"},
           {"38", "male", "81671"},
           {"39", "female", "81672"},
           {"40", "male", "81673"},
           {"41", "female", "81674"},
           {"42", "male", "81675"},
           {"43", "female", "81676"},
           {"44", "male", "81677"}};
   List<String[]> expected = List.of(rawData);
   for(int x = 0; x<12;x++) {
       Assertions.assertArrayEquals(expected.get(x), actual.get(x));
   }
}
```

Unit testing is done by using a test data and sending it in as a parameter. The resulting data is then checked by comparing it to an expected result.



```
✓ <default package>                                              2 s 699 ms
  ✓ JUnit Jupiter                                                2 s 693 ms
    ✓ AnalyzeAttributesTest                                      1 s 917 ms
      ✓ generateAttributeAnalytics()                             1 s 917 ms
    ✓ AnalyzeDistributionOfRiskTest                                 300 ms
      ✓ test_ResultAnalysis()                                       268 ms
      ✓ distributionOfRiskWithThreshold()                            16 ms
      ✓ distributionOfRisk()                                         11 ms
      ✓ listOfRisk()                                                  2 ms
      ✓ listOfRisk_numClasses()                                       2 ms
      ✓ listOfRisk_numRecords()                                       1 ms
    ✓ AnonymizationMetricsTest                                      168 ms
      ✓ gatherGeneralizationAttributes()                            163 ms
      ✓ gatherProcessTime()                                           3 ms
      ✓ gatherPrivacyModels()                                         2 ms
    ✓ RiskProfileTest                                                99 ms
      ✓ testHashCode()                                               95 ms
      ✓ equals()                                                      2 ms
      ✓ getReIdentificationRisk()                                     1 ms
      ✓ getDistributionOfRisk()                                       1 ms
    ✓ HierarchyBuilderIntervalBasedTest                              57 ms
      ✓ ageIntervalBasedHierarchyBuilder()                           35 ms
      ✓ arxExample()                                                 17 ms
      ✓ ageIntervalBasedHierarchyBuilder__createIntervalFunction()    5 ms
    ✓ IntervalBasedHierarchyBuilderTest                              43 ms
      ✓ build_with_double_type_data()                                31 ms
      ✓ build_with_double_type_data_fails_with_no_intervals()         5 ms
      ✓ build_with_single_interval()                                  3 ms
      ✓ build_with_labeled_intervals()                                2 ms
      ✓ build()                                                       1 ms
      ✓ build_with_no_ranges()                                        1 ms
    ✓ HierarchyBuilderOrderBasedTest                                 29 ms
      ✓ anonymizeUsingGeneratedHierarchy()                           19 ms
      ✓ anonymizeDefaultDataUsingGeneratedHierarchy()                 5 ms
      ✓ generateHierarchy()                                           4 ms
      ✓ testStringHierarchyBuilding()                                 1 ms
    ✓ AttackerSuccessTest                                            28 ms
      ✓ testHashCode()                                               19 ms
      ✓ equals()                                                      8 ms
      ✓ create_from_RiskModelSampleSummary()                          1 ms
    ✓ HierarchyBuilderRedactionBasedTest                             12 ms
      ✓ redactionBasedHierachyGeneration()                           12 ms
    ✓ ARXDataFactoryTest                                             12 ms
      ✓ create_data_shape_is_correct()                                3 ms
      ✓ create_returnDataAttribute_is_correct()                       3 ms
      ✓ create_returnDataHierarchy_is_correct()                       2 ms
      ✓ create_with_null_data()                                       2 ms
      ✓ create_returnDataAttribute_is_not_overwritten_by_hierarchy()  1 ms
      ✓ create_returnData_is_correct()                                1 ms
    ✓ ReIdentificationRiskTest                                       10 ms
      ✓ getQuasiIdentifiers()                                         4 ms
      ✓ testHashCode()                                                4 ms
      ✓ equals()                                                      1 ms
      ✓ getAttackerSuccessRate()                                      1 ms
      ✓ getPopulationModel()
      ✓ getMeasures()
    ✓ ARXPrivacyCriterionFactoryTest                                 10 ms
      ✓ create_returnObject_is_Correct()                             10 ms
    ✓ DistributionOfRiskTest                                          5 ms
      ✓ create_data_is_correct()                                      2 ms
      ✓ createFromRiskAndMaxRisk_from_Payload()                       2 ms
      ✓ createFromRiskAndMaxRisk_from_result()                        1 ms
    ✓ ARXConfigurationFactoryTest                                     2 ms
      ✓ setSuppression()                                              1 ms
      ✓ create_NotNull()                                              1 ms
      ✓ setPrivacyModels_KAnon()
    ✓ OrderBasedHierarchyBuilderTest                                  1 ms
      ✓ build()                                                       1 ms
  ✓ JUnit Vintage                                                     6 ms
    ✓ DefaultDataGenerationTest                                       6 ms
      ✓ generateDefaultDataFromList                                   6 ms
```

Figure 33 - All unit test passed

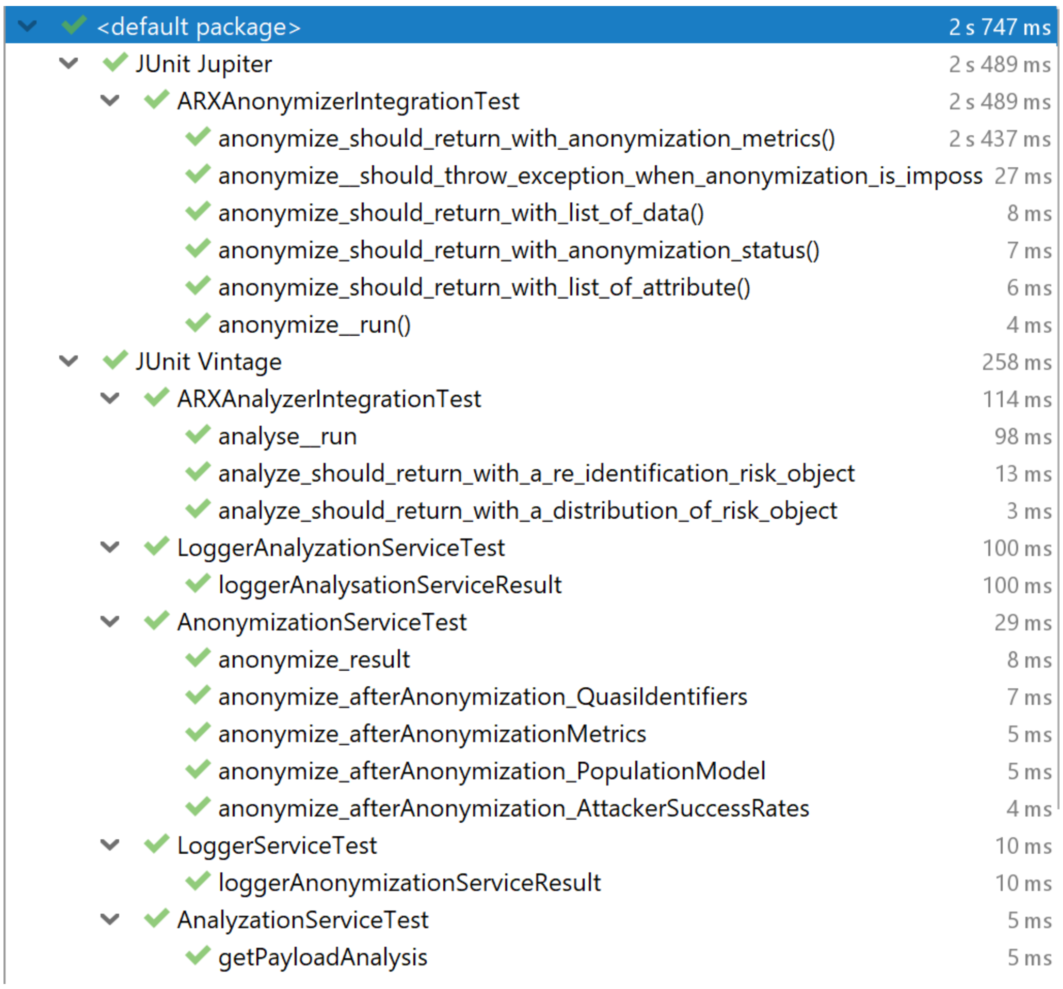
4.6.2 Integration testing

Integration testing is done on all the methods that uses the factory classes, and all the models used by the factory classes. A test object is generated and used as a parameter for integration testing. The resulting object from the integration tests is then checked if it managed to correctly created the response model object.

```
@Test
void anonymize_should_return_with_list_of_attribute(){
   AnonymizeResult result = testAnonymizer.anonymize(testRequestPayload);
   Assertions.assertNotNull(result.getAttributes());
}

@Test
void anonymize_should_return_with_list_of_data(){
   AnonymizeResult result = testAnonymizer.anonymize(testRequestPayload);
   Assertions.assertNotNull(result.getData());
}
```

The integration is about making sure that the different methods from different classes can work together and create the correct data object.



```
<default package>                                                            2 s 747 ms
  JUnit Jupiter                                                              2 s 489 ms
    ARXAnonymizerIntegrationTest                                             2 s 489 ms
      anonymize_should_return_with_anonymization_metrics()                   2 s 437 ms
      anonymize__should_throw_exception_when_anonymization_is_imposs           27 ms
      anonymize_should_return_with_list_of_data()                               8 ms
      anonymize_should_return_with_anonymization_status()                       7 ms
      anonymize_should_return_with_list_of_attribute()                          6 ms
      anonymize__run()                                                          4 ms
  JUnit Vintage                                                               258 ms
    ARXAnalyzerIntegrationTest                                                114 ms
      analyse__run                                                             98 ms
      analyze_should_return_with_a_re_identification_risk_object               13 ms
      analyze_should_return_with_a_distribution_of_risk_object                  3 ms
    LoggerAnalyzationServiceTest                                              100 ms
      loggerAnalysationServiceResult                                          100 ms
    AnonymizationServiceTest                                                   29 ms
      anonymize_result                                                          8 ms
      anonymize_afterAnonymization_QuasiIdentifiers                             7 ms
      anonymize_afterAnonymizationMetrics                                       5 ms
      anonymize_afterAnonymization_PopulationModel                              5 ms
      anonymize_afterAnonymization_AttackerSuccessRates                         4 ms
    LoggerServiceTest                                                          10 ms
      loggerAnonymizationServiceResult                                         10 ms
    AnalyzationServiceTest                                                      5 ms
      getPayloadAnalysis                                                        5 ms
```

Figure 34 - All integration test passed

4.6.3 System testing

System testing is done on the three main endpoints in the service. Spring boot starter test is used to start the service. A request object based on generated test data is created. The service endpoint is called with the request object as a parameter. The response object is returned for the service, and is checked if it is correctly created and if the values inside the object are correct.

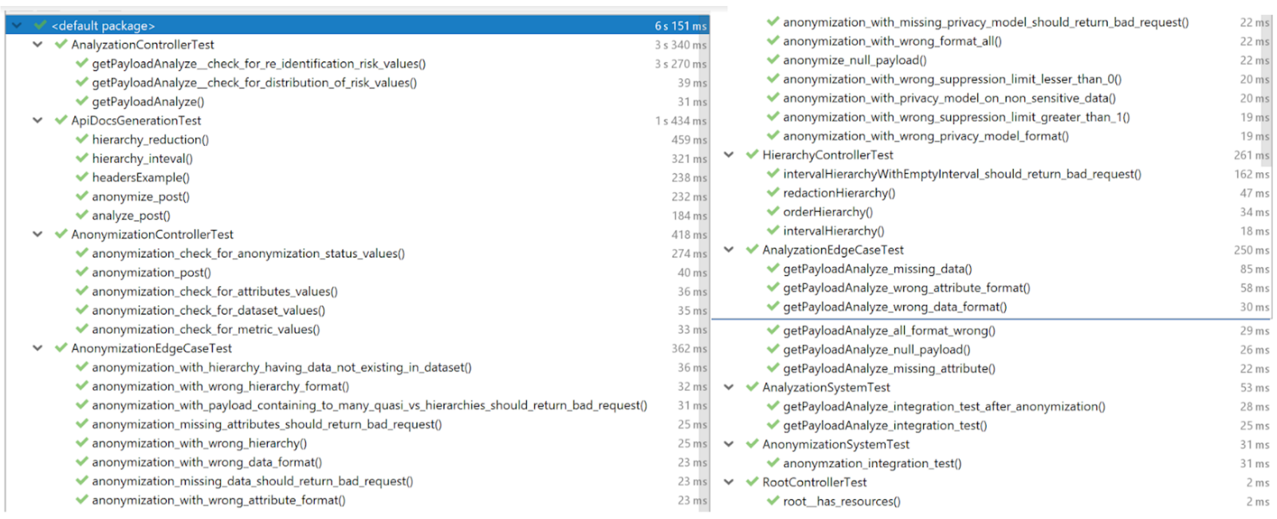
```
@Test
void getPayloadAnalyze_system_test(){
   Request testRequestPayload = GenerateIntegrationTestData.zipcodeRequestPayload();
   ResponseEntity<RiskProfile> responseEntity =
restTemplate.postForEntity("/api/analyze", testRequestPayload, RiskProfile.class);

   assertNotNull(responseEntity);
   assertSame(HttpStatus.OK , responseEntity.getStatusCode());
   RiskProfile actual = Objects.requireNonNull(responseEntity.getBody());
   RiskProfile expected = GenerateIntegrationTestData.zipcodeAnalyzation();
   Assertions.assertEquals(expected,actual);
}
```

Edge case testing was also done by generating request object with incorrect or invalid data. The end-points are then expected to throw an error exception, this exception is then compared to with an expected execution as wells making sure that end-points sends a detailed description of the error message and how to correct the error.

```java
@Test
void getPayloadAnalyze_wrong_attribute_format(){
   Request wrongAttributeFormat = 
GenerateEdgeCaseData.zipcodeRequestPayload_wrong_attribute_format();
   ResponseEntity<IllegalArgumentException> responseEntity = 
restTemplate.postForEntity("/api/analyze", wrongAttributeFormat, 
IllegalArgumentException.class);

   assertNotNull(responseEntity);
   assertSame(HttpStatus.BAD_REQUEST, responseEntity.getStatusCode());
   var resultData = responseEntity.getBody();
   assertNotNull(resultData);
   assertNotNull(resultData.getMessage());
}
```



```
✓ <default package>                                                                         6 s 151 ms
  ✓ AnalyzationControllerTest                                                               3 s 340 ms
    ✓ getPayloadAnalyze__check_for_re_identification_risk_values()                          3 s 270 ms
    ✓ getPayloadAnalyze__check_for_distribution_of_risk_values()                                 39 ms
    ✓ getPayloadAnalyze()                                                                        31 ms
  ✓ ApiDocsGenerationTest                                                                   1 s 434 ms
    ✓ hierarchy_reduction()                                                                     459 ms
    ✓ hierarchy_inteval()                                                                       321 ms
    ✓ headersExample()                                                                          238 ms
    ✓ anonymize_post()                                                                          232 ms
    ✓ analyze_post()                                                                            184 ms
  ✓ AnonymizationControllerTest                                                                 418 ms
    ✓ anonymization_check_for_anonymization_status_values()                                     274 ms
    ✓ anonymization_post()                                                                       40 ms
    ✓ anonymization_check_for_attributes_values()                                                36 ms
    ✓ anonymization_check_for_dataset_values()                                                   35 ms
    ✓ anonymization_check_for_metric_values()                                                    33 ms
  ✓ AnonymizationEdgeCaseTest                                                                   362 ms
    ✓ anonymization_with_hierarchy_having_data_not_existing_in_dataset()                         36 ms
    ✓ anonymization_with_wrong_hierarchy_format()                                                32 ms
    ✓ anonymization_with_payload_containing_to_many_quasi_vs_hierarchies_should_return_bad_request()  31 ms
    ✓ anonymization_missing_attributes_should_return_bad_request()                               25 ms
    ✓ anonymization_with_wrong_hierarchy()                                                       25 ms
    ✓ anonymization_with_wrong_data_format()                                                     23 ms
    ✓ anonymization_missing_data_should_return_bad_request()                                     23 ms
    ✓ anonymization_with_wrong_attribute_format()                                                23 ms
    ✓ anonymization_with_missing_privacy_model_should_return_bad_request()                       22 ms
    ✓ anonymization_with_wrong_format_all()                                                      22 ms
    ✓ anonymize_null_payload()                                                                   22 ms
    ✓ anonymization_with_wrong_suppression_limit_lesser_than_0()                                 20 ms
    ✓ anonymization_with_privacy_model_on_non_sensitive_data()                                   20 ms
    ✓ anonymization_with_wrong_suppression_limit_greater_than_1()                                19 ms
    ✓ anonymization_with_wrong_privacy_model_format()                                            19 ms
  ✓ HierarchyControllerTest                                                                     261 ms
    ✓ intervalHierarchyWithEmptyInterval_should_return_bad_request()                            162 ms
    ✓ redactionHierarchy()                                                                       47 ms
    ✓ orderHierarchy()                                                                           34 ms
    ✓ intervalHierarchy()                                                                        18 ms
  ✓ AnalyzationEdgeCaseTest                                                                     250 ms
    ✓ getPayloadAnalyze_missing_data()                                                           85 ms
    ✓ getPayloadAnalyze_wrong_attribute_format()                                                 58 ms
    ✓ getPayloadAnalyze_wrong_data_format()                                                      30 ms
    ✓ getPayloadAnalyze_all_format_wrong()                                                       29 ms
    ✓ getPayloadAnalyze_null_payload()                                                           26 ms
    ✓ getPayloadAnalyze_missing_attribute()                                                      22 ms
  ✓ AnalyzationSystemTest                                                                        53 ms
    ✓ getPayloadAnalyze_integration_test_after_anonymization()                                   28 ms
    ✓ getPayloadAnalyze_integration_test()                                                       25 ms
  ✓ AnonymizationSystemTest                                                                      31 ms
    ✓ anonymzation_integration_test()                                                            31 ms
  ✓ RootControllerTest                                                                            2 ms
    ✓ root__has_resources()                                                                       2 ms
```

Figure 35 - All system test passed

4.7 PyARXaaS

PyARXaaS tests are located under the **test/** directory in the project root[59]. The tests are all written using the **unittest[60]** testing framework included in the Python standard library. The tests are organized in a directory hierarchy mirroring the package directory hierarchy. The purpose of the tests are to verify that the package is behaving as intended and as a measure against future regressions.

```
tests/
├── __init__.py
└── pyarxaas/
    ├── data_generator.py
    ├── hierarchy/
    │   ├── __init__.py
    │   ├── test_IntervalHierarchyBuilder.py
    │   ├── test_Interval.py
    │   ├── test_OrderHierarchyBuilder.py
    │   └── test_ReductionHierarchyBuilder.py
    ├── __init__.py
    ├── models/
    │   ├── __init__.py
    │   ├── test_AnonymizationMetrics.py
    │   ├── test_AnonymizationResult.py
    │   ├── test_AnonymizeResult.py
    │   ├── test_Attribute.py
    │   ├── test_Data.py
    │   ├── test_Dataset.py
    │   ├── test_privacy_models_ldiversity.py
    │   ├── test_privacy_models_Tcloseness.py
    │   └── test_RiskProfile.py
    ├── test_AaaSConnector.py
    ├── test_ARXaaS.py
    ├── test_converters.py
    ├── test_data/
    │   ├── analyze_response_test_data.json
    │   └── anonymize_response_test_data.json
    ├── test_request_builder.py
    └── test_state_printer.py
```

Tests where ran for every new branch committed to the Github repository. This was intended to ensure that only stable code was merged into master and released to users. PyARXaaS development followed TDD practices and the **test coverage is at 89%** as of writing this rapport. See the chapter on 5.4.3 Release Pipeline for more on the PyARXaaS CI/CD pipeline.

4.7.1 Unit testing

[59] PyARXaaS test directory - https://github.com/oslomet-arx-as-a-service/PyARXaaS/tree/master/tests
[60] Python unittest - https://docs.python.org/3/library/unittest.html

For a package like PyARXaaS unit tests are key to ensure the users are provided a stable and well put together product. It's not enough or in some cases possible to fall back to integration test that test the package components together. User of the package have the possibility to pick and choose parts from the package in new and unanticipated combination. To ensure the package supports innovative use the individual parts must have good isolated unit tests.

Extract from the unit test for the Dataset class[61]

```
class DatasetTest(unittest.TestCase):

   def setUp(self):
       self.test_data = [['id', 'name'],
                         ['0', 'Viktor'],
                         ['1', 'Jerry']]
       self.test_attribute_type_mapping = {'id': AttributeType.IDENTIFYING,
                                           'name': AttributeType.QUASIIDENTIFYING}

   def test_init(self):
       Dataset(self.test_data, self.test_attribute_type_mapping)

   def test_equality(self):
       dataset_1 = data_generator.id_name_dataset()
       dataset_2 = data_generator.id_name_dataset()
       self.assertEqual(dataset_1, dataset_2)
       self.assertIsNot(dataset_1, dataset_2)
       dataset_2._set_attribute_type("id", AttributeType.QUASIIDENTIFYING)
       self.assertNotEqual(dataset_1, dataset_2)

   def test_hash(self):
       dataset_1 = data_generator.id_name_dataset()
       dataset_2 = data_generator.id_name_dataset()
       test_set = {dataset_1, dataset_2}
       self.assertEqual(1, len(test_set))
```

[61] Dataset unit tests - ttps://github.com/oslomet-arx-as-a-service/PyARXaaS/blob/master/tests/pyarxaas/models/test_Dataset.py

4.7.2 Integration testing

PyARXaaS contains many objects that have a collaborative relationship. Integration tests have been written to ensure the objects work together as intended.

Example from test_ARXaaS.py[62]

The tests verifies that the call to the method results in a return object. It does not verify that the content of the result object is correct. This is handled by other system tests.

```
   def test_analyze(self):
        aaas = ARXaaS('http://localhost', connector=MockAaasConnector)
        self.assertIsNotNone(aaas.risk_profile(self.test_dataset))

    def test_anaonymize(self):
        aaas = ARXaaS('http://localhost', connector=MockAaasConnector)
        self.assertIsNotNone(aaas.anonymize(self.test_dataset,
privacy_models=[KAnonymity(4)]))
```

4.7.3 System testing

System testing is mainly applicable to the ARXaaS class. This is the class that brings together and uses several different parts of the package. The ARXaaS class is responsible for making calls to the ARXaaS service. Tests should avoid dependency on outside elements as far as it is possible. The team has handled this in the ARXaaS test by *mocking* the ARXaaSConnector class in the ARXaaS system tests.

ARXaaSConnector Mock

```
class MockAaasConnector(ARXaaSConnector):

    def anonymize_data(self, payload: Body):
        return AnonymzationResponseStub()

    def risk_profile(self, payload: Body):
        return AnalyzationResponseStub()

    def hierarchy(self, payload: Body):
        return HierarchyResponseStub()

    def root(self):
        return RootResponseStub()
```

System test examples from ARXaaS class test The test verifies that the result have the correct attribute type.

[62] test_ARXaaS.py - ttps://github.com/oslomet-arx-as-a-service/PyARXaaS/blob/master/tests/pyarxaas/test_ARXaaS.py

```
def test_anonymize__dataset_attributes_are_correct(self):
   aaas = ARXaaS('http://localhost', connector=MockAaasConnector)
   anonymize_result = aaas.anonymize(self.test_dataset, [KAnonymity(4)])
   self.assertEqual(AttributeType.IDENTIFYING,
AttributeType(anonymize_result.dataset._attributes[0].type))
```

4.8 WebARXaaS

WebARXaaS was a stretch goal and the development process of the web application started late into the project. As a result of the limited time WebARXaaS lacks unit and component based testing. Although as most of the business logic is done on the backend by ARXaaS, most of the data operations are already being tested during the tests of ARXaaS. The fact that WebARXaaS mostly only handles the uploading, downloading, and visualization of data, makes it less prone for failure than ARXaaS. But nevertheless it is important that the team does their best to ensure that all the inputs gets supplied in the right format, and that the output is visualized correctly.

4.8.1 Data integrity

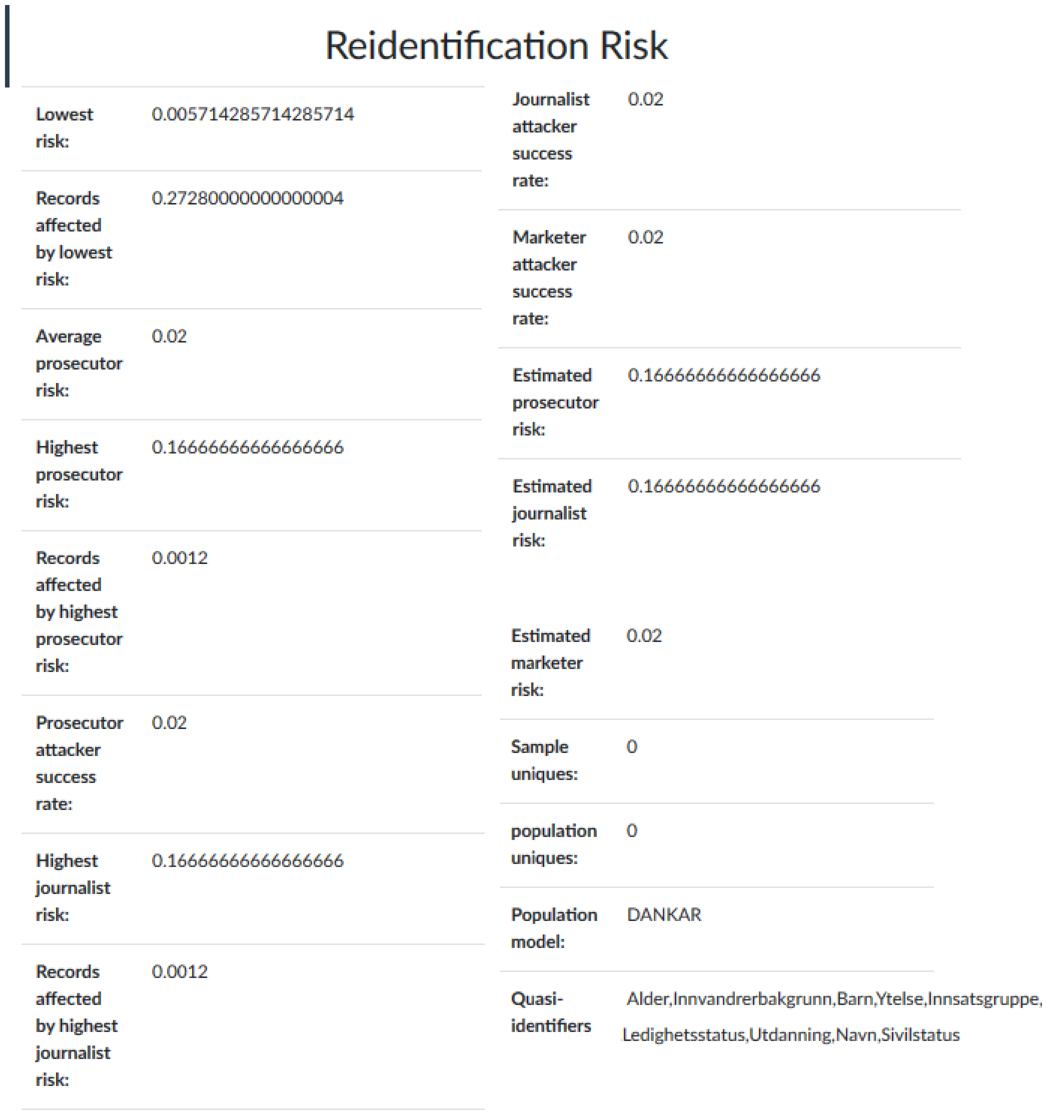
The primary way of testing data integrity in WebARXaaS is by manually checking if all data values are in place and are containing valid values.

Before the testing could start the team made sure that they had downloaded the *ARX* anonymization tool on their local machine, and ensure they had access to a dataset containing personal identifying tabular data or *microdata*[63]. To start the team first did a anonymization with the arx tool on the data they had supplied, making sure that they applied all four different attribute types. Then the team read the output metrics from both the analyzation and anonymization process and stored it through screenshots for it to be compared with WebARXaaS.

For testing WebARXaaS the team started up one instance each of both WebARXaaS and ARXaaS locally, they then used the WebARXaaS web interface and made sure to upload the exact same data through the web interface as they did in the arx application.

Anonymizing the data both in the ARX application and WebARXaaS at the same settings, then comparing the Re-identification risks. If every parameter have been assigned in the same way, all the data fields should hold identical values. If any of the data fields show unexpected values it could be an indicator of an error in either WebARXaaS, or the ARXaaS backend. It is especially important that the Highest prosecutor risk value matches both places as this value is usually used to determine wherever the dataset is sufficiently anonymous or not.

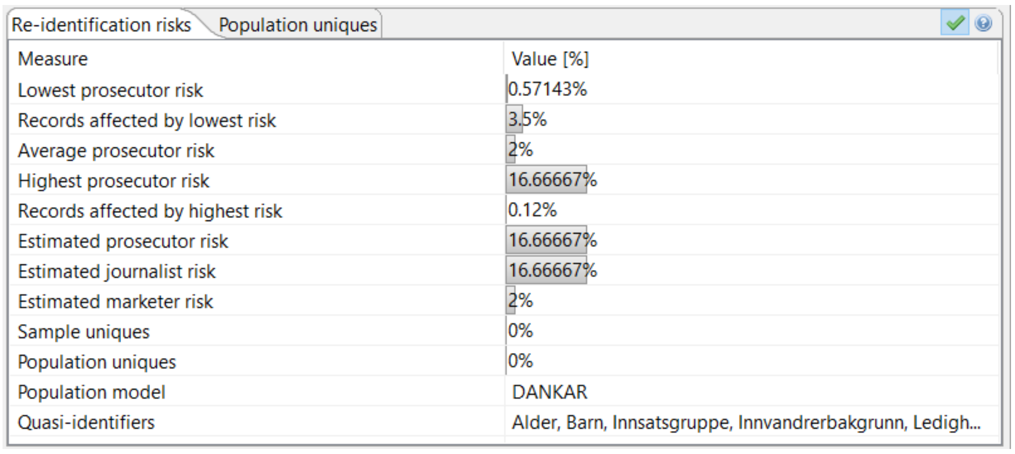
[63] Microdata - https://nsd.no/macrodataguide/macro_data.html



Reidentification Risk

Lowest risk:	0.005714285714285714	Journalist attacker success rate:	0.02
Records affected by lowest risk:	0.27280000000000004	Marketer attacker success rate:	0.02
Average prosecutor risk:	0.02	Estimated prosecutor risk:	0.16666666666666666
Highest prosecutor risk:	0.16666666666666666	Estimated journalist risk:	0.16666666666666666
Records affected by highest prosecutor risk:	0.0012	Estimated marketer risk:	0.02
Prosecutor attacker success rate:	0.02	Sample uniques:	0
Highest journalist risk:	0.16666666666666666	population uniques:	0
Records affected by highest journalist risk:	0.0012	Population model:	DANKAR
		Quasi-identifiers	Alder,Innvandrerbakgrunn,Barn,Ytelse,Innsatsgruppe, Ledighetsstatus,Utdanning,Navn,Sivilstatus

Figure 36 - Risk profile form WebARXaaS



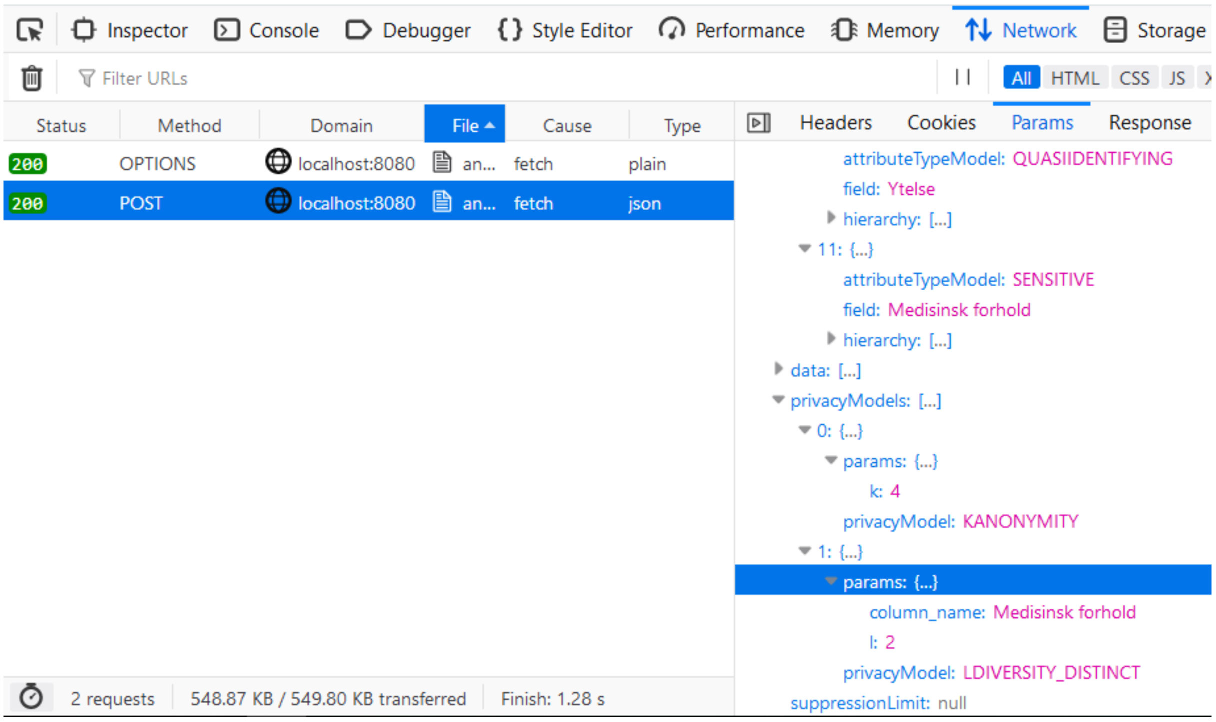
Re-identification risks | Population uniques

Measure	Value [%]
Lowest prosecutor risk	0.57143%
Records affected by lowest risk	3.5%
Average prosecutor risk	2%
Highest prosecutor risk	16.66667%
Records affected by highest risk	0.12%
Estimated prosecutor risk	16.66667%
Estimated journalist risk	16.66667%
Estimated marketer risk	2%
Sample uniques	0%
Population uniques	0%
Population model	DANKAR
Quasi-identifiers	Alder, Barn, Innsatsgruppe, Innvandrerbakgrunn, Ledigh...

Figure 37 - Risk profile form th ARX GUI

4.8.2 API Testing

In order to ensure that the metadata applied through the user interface gets sent onward to the ARXaaS service, the team inspected the json params from the outgoing request. By inspecting the json parameters they confirmed that all the attributes had been assigned correct attribute Type model. The team could also see each of the privacy models, and ensure that all the meta data for each of the models are in place and set correctly. This also confirmed that each of the rows in the dataset had been loaded. SuppressionLimit was an optional parameter, so a null value was acceptable.



```
Status  Method   Domain          File  Cause  Type
200     OPTIONS  localhost:8080  an... fetch  plain
200     POST     localhost:8080  an... fetch  json

2 requests   548.87 KB / 549.80 KB transferred   Finish: 1.28 s
```

```
        attributeTypeModel: QUASIIDENTIFYING
        field: Ytelse
      ▸ hierarchy: [...]
    ▾ 11: {...}
        attributeTypeModel: SENSITIVE
        field: Medisinsk forhold
      ▸ hierarchy: [...]
  ▸ data: [...]
  ▾ privacyModels: [...]
    ▾ 0: {...}
      ▾ params: {...}
          k: 4
        privacyModel: KANONYMITY
    ▾ 1: {...}
      ▾ params: {...}
          column_name: Medisinsk forhold
          l: 2
        privacyModel: LDIVERSITY_DISTINCT
  suppressionLimit: null
```

Figure 38 - Suppression limit parameter

4.9 Performance testing

The team completed different types of performance testing of ARXaaS in isolation with direct calls to the HTTP endpoints and with use of the PyARXaaS package. The Key performance indicator (KPI) for the performance tests are response time. The use case for the solution is not hundreds of calls a second. The much more likely use case is a single request with a large dataset. Scalability issues related to request per second is mitigated by the fact that ARXaaS can be scaled horizontally. The team worked from the assumption of a maximum of a single call per second to ARXaaS. The service was therefore tested with large but few requests. The goal of the different performance test where to discover defects or stability issues when the solution was put under normal load (load test) and extreme stress (stress test)

4.9.1 Load test

The load testing was completed using the Locust framework. Locust provides a Python API for constructing customized load testing jobs. The analyze and anonymize endpoints were tested in isolation. The locust tests where ran against the ARXaaS service running in NAIS pre production environment. The test simulates users making calls to the ARXaaS service endpoints. The test was for each test case configured to run for 1 minute with 1 request per second.

The test data can be found at this url:
https://github.com/oslomet-arx-as-a-service/ARXaaS-load-testing/blob/master/scripts/data/dummy-dataset-260219.csv

NAIS configuration file for the test:

```
replicas:
    min: 1
    max: 1

resources:
    limits:
        cpu: 1500m
        memory: 4800Mi

    requests:
        cpu: 200m
        memory: 512Mi
```

Locust CLI command

```
python -m locust.main -f scripts/analyze_locust_test.py --host http://localhost:8080
--no-web -c 2 -r 1 --run-time 1m --csv=example -t30s
```

Locust analyze test script

```
class UserBehavior(TaskSet):

    @task(1)
    def analyze(self):
        self.client.post("/api/anonymize", json=request, verify=False)


class WebsiteUser(HttpLocust):
    task_set = UserBehavior
    min_wait = 500
    max_wait = 1000
```

The data from the load test can be found in the Github repository[64]. Each test is ran with a dataset with the same dataset but copied n times to achieve a given size.

Result of analyze endpoint test with 100 000 rows

Method	Name	# requests	# failures	Median response time	Average response time	Min response time	Max response time	Average Content Size	Requests/s
POST	/api/analyze	25	0	1300	1400	729	2772	3312	0.43
None	Total	25	0	1300	1400	729	2772	3312	0.43

Result of analyze endpoint test with 1.2 million rows

Method	Name	# requests	# failures	Median response time	Average response time	Min response time	Max response time	Average Content Size	Requests/s
POST	/api/analyze	5	0	10000	9495	8290	10256	3589	0.10
None	Total	5	0	10000	9495	8290	10256	3589	0.10

The client has not put forward any requirements regarding response time. During Sprint review the load test result was presented and the feedback form the product owner was that the result was acceptable but that the team should monitor for changes.

The test result datas can be found on the github page:
https://github.com/oslomet-arx-as-a-service/ARXaaS-load-testing/tree/master/tests/analyze_stress_test/data

[64] Analyze load test - https://github.com/oslomet-arx-as-a-service/ARXaaS-load-testing/tree/master/tests/analyze_stress_test

4.9.2 Stress test

Stress testing is testing of the upper limits of a system. It puts the system under a extreme load the system was not designed or expected to handle. It is intended to reveal system behaviour outside the normal scope of operations and how it recovers from extreme load.

Windowing Stress test

Tool used: Jupyter notebooks

Link to test notebook:
https://github.com/oslomet-arx-as-a-service/ARXaaS-load-testing/blob/master/window-dataset-load-testing.ipynb

Stress test with increased rows and columns conducted 12.04.19. The test was completed at NAV IT NAIS test cluster. The NAIS team was informed of the test and gave the team a "OK" before the test was conducted. The purpose of the test was to find the limits of the ARXaaS system capabilities and when found, where the system would break. A secondary goal was to measure the response time for increasing size of datasets.

NAIS configuration file for the test:

```
replicas:
min: 1
max: 1


resources:
limits:

    cpu: 1500m
    memory: 4800Mi

requests:

    cpu: 200m
    memory: 512Mi
```

The test used randomly generated datasets consisting of the fields id, name, age, gender, location. Link to the script for generating test data:
https://github.com/oslomet-arx-as-a-service/ARXaaS-load-testing/blob/master/scripts/data/test_data.py

The generated datasets looked like this example:

	name	**age**	**gender**	**location**	**name**	**age**	**gender**	**location**
0	vtvkzewqgt	21	male	Miami	iceglyzyeu	80	female	London
1	ucfbturvpq	67	male	Oslo	nxctvwqhoj	79	male	Bergen
2	ldsrfybaai	78	female	London	zvnnpywesx	22	female	London
3	dytjgeyzdc	81	male	Tokyo	xugsuyqrme	16	female	Moscow
4	ptjkbfpria	40	male	Bergen	epvqcyedli	65	male	Miami
5	kpmgvyyceg	95	female	Miami	nheehiuyaf	99	female	Bergen
6	yjqhsdsgqm	78	female	Tokyo	szuohqxmyv	72	female	Tokyo
7	qykoujzcsv	57	male	Bergen	cywhpviqrr	23	female	Tokyo
8	bttfvkssnz	84	female	Bejing	xwosuqlpys	82	male	Tokyo
9	eoiqvrwitg	22	male	Miami	xihjismxqp	11	female	Bergen

Test dataset shape:

```
[(50000, 1),
 (100000, 2),
 (150000, 2),
 (200000, 3),
 (250000, 3),
 (300000, 4),
 (350000, 4),
 (400000, 5),
 (450000, 5),
 (500000, 6),
 (550000, 6),
 (600000, 7),
 (650000, 7),
 (700000, 8),
 (750000, 8),
 (800000, 9),
 (850000, 9),
 (900000, 10),
 (950000, 10),
 (1000000, 11)]
```

The largest dataset consisted off 1 million rows and 11 columns.

The test script
The test script uses the PyARXaaS to connect to ARXaaS. It uses the Python standard library module; timeit, to track the response time for the calls to the ARXaaS service.

```python
def dataset_window_analyze_stress_test(shapes: list, connector):
   global dataset
   global con
   con = connector

   for shape in shapes:
       result = {}
       dataset = test_window_dataset(shape[0], shape[1])
       size = sys.getsizeof(dataset.to_dataframe().to_csv())
       elapsed_time = timeit.timeit(f"analyze(dataset)",
                                    globals=globals(),
                                    number=1)
       result[str(shape[0])+"x"+str(shape[1])] = (elapsed_time, size)
       yield result
```

The Result
The result visualized as line graphs in figure 39 and figure 40 below shows a linear increase in response time from the service as the dataset size in the request increases. This is good news for the service as it indicates that the service can handle bigger requests with more machine power. If the request time increase would be exponential or worse adding more machine power would not scale with increased dataset sizes.

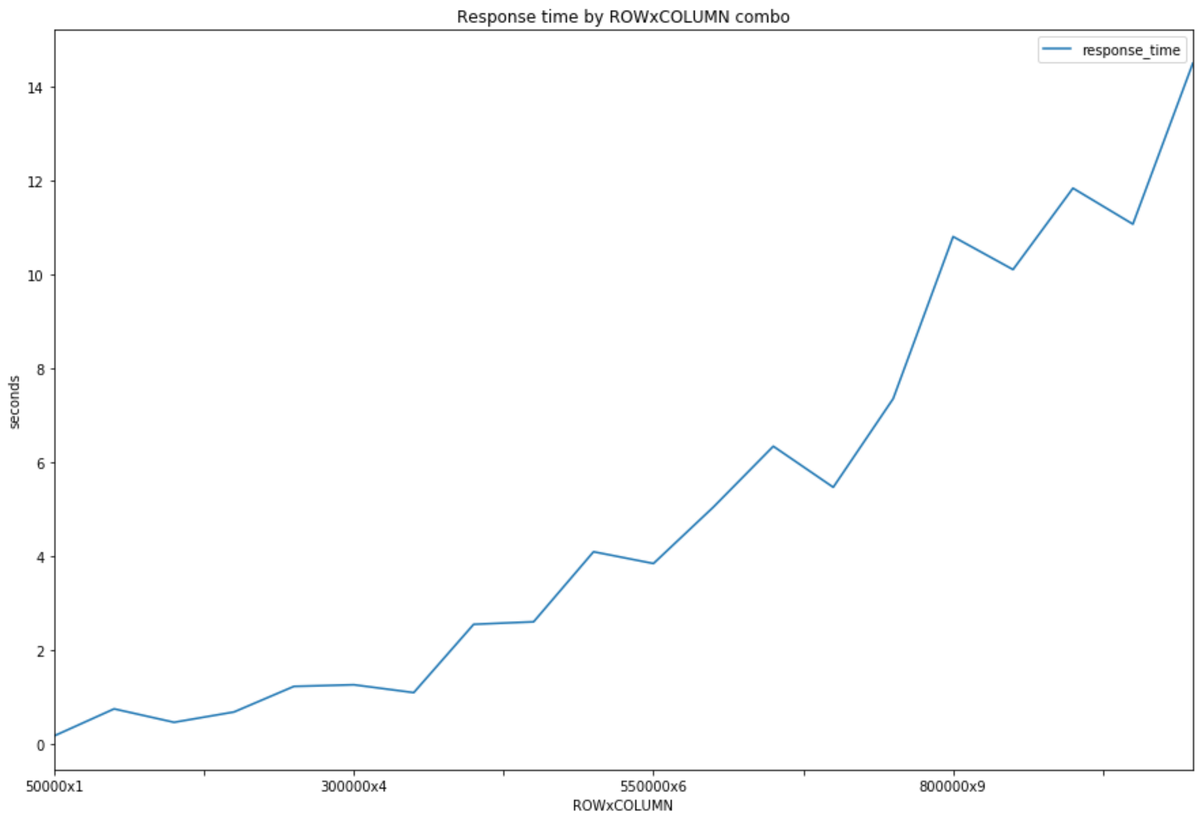


Figure 39 - Response time for a dataset of given rows x columns during ARXaaS stress test

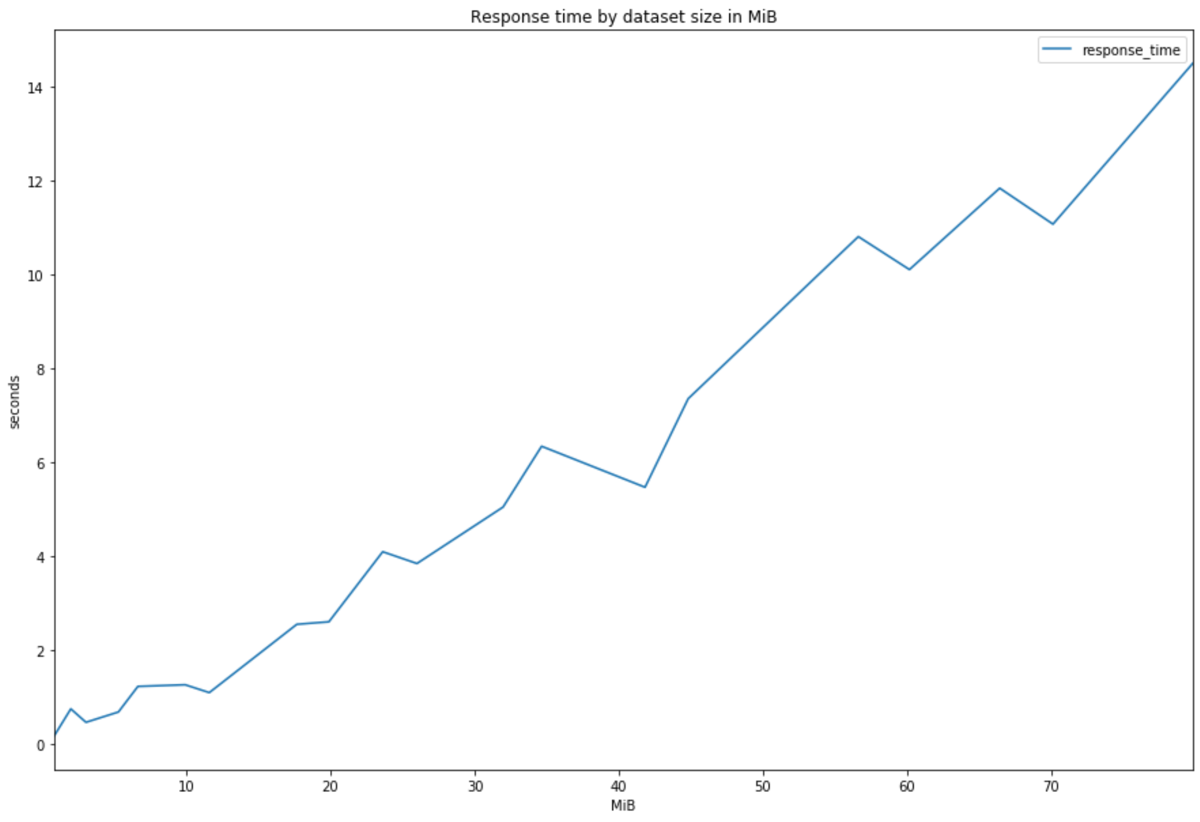


Figure 40 - of response time for a dataset of size in MiB during ARXaaS stress test

4.10 Acceptance testing

Acceptance testing was done by hosting a workshop at NAV IT. In this workshop the product owner, a data engineer, a data analyst and two data scientist from NAV IT was present. The product owner and the participants followed a step by step guide on how to start and use our solution.

The main focus of this workshop was to show both the anonymization and analyzation features, as well as show the different data the user receives from the process. Explore the different error messages was also tested to see if they were detailed enough on explaining the type of error and the solution to fixing the error. Feedback was collect on how to improve the service, as well as possible new features to be implemented.

Acceptance test report

Testers	Robindra, Gøran, Eirk, Paul, John Vegard
Facilitators	Sondre, Jeremiah, Andre, Julian
Test goal	Gather feedback on the usability and API ergonomics of the PyARXaaS package from NAV IT data scientist and analysts. The tester where to use the user guide to complete the work so a side effect of the test was gathering feedback on the quality of the user guide. And by transitive the ARXaaS service.
Test method	Test participants where given a test dataset to analyze re-identification risk on and then anonymize to k=4 using provided generalization hierarchies. Test data and hierarchies where supplied as csv files.
Time/place	Sannergata 2(NAV IT offices), 12 April
Result	Several defects were discovered in the user guides. Mostly related to the documentation being out of synch with the latest PyARXaaS version. Some confusion regarding the method naming and how similar named methods differed. But after the initial hurdles where overcome the test participants all managed to complete a successful risk analysis and anonymization using the solution. Features more related to a legal framework for anonymization was brought up, but such a solution not a part of the scope of this project. The testers consensus was that the solution delivered high value to the analyzation problems they are faced with.

4.11 Conclusion test documentation

By following a continuous integration workflow by continually testing each new feature on the service- and client-side, ensured that the new branch is stable before merging with the master branch in the version control host repository. The test provided a steady flow of feedback to the development team. Utilizing test driven development process ensured a the project was continually tested.

Test driven development made it easier to see if the new features were unstable or had problems long before it could enter the master branch. Conducting a workshop with the client also meant the project team received important feedback from both the product owner and other stakeholders on the solution.

Performance tests ensured that the solution could withstand normal and even extreme stress. The performance and edge tests meant the team could be confident that the solution would perform when faced with real user action.

5 Product documentation

The purpose of this document is to give the reader a technical view of how the system has been build and the functionalities available in both the service and clients. This document also shows how the client works with the service.

This document is written with the expectation of being used for working with operations, maintenance or future development, it is therefore expected that the reader has programming knowledge. For a better understanding of this document we recommend reading the presentation documentation first.

5.1 Introduction

The solution to the clients problem that the team developed consists of three different products. Acting as the service, the team created *ARX as a service*, or in short ARXaaS, which provides all the business logic using the ARX library. ARXaaS's design allows it to be packaged and deployed to a *container orchestration platform* and being interacted with, through a REST API. To fulfill our customers needs, the team created two clients, each designed to provide the same functionality, but for two very different groups of users.

Our main client *PyARXaaS* is a python package designed specifically with data scientists as the main user group in mind, which makes it possible to utilize ARX functionality from any python program or notebook. As well as integrating well in automated data pipelines.

The secondary client is *WebARXaaS* which is designed to be the more user-friendly alternative. The WebARXaaS provides the ARX functionality through a dynamic single-page web application. Meanwhile demanding no installation of software, or programming experience to use. The development of WebARXaaS was started later in the process than PyARXaaS as the need for it didn't become clear, until towards the end of the project, and even then was implemented as a stretch goal.

5.2 ARXaaS

5.2.1 Short presentation

ARXaaS re-packages the core ARX libraries as a data anonymization service.

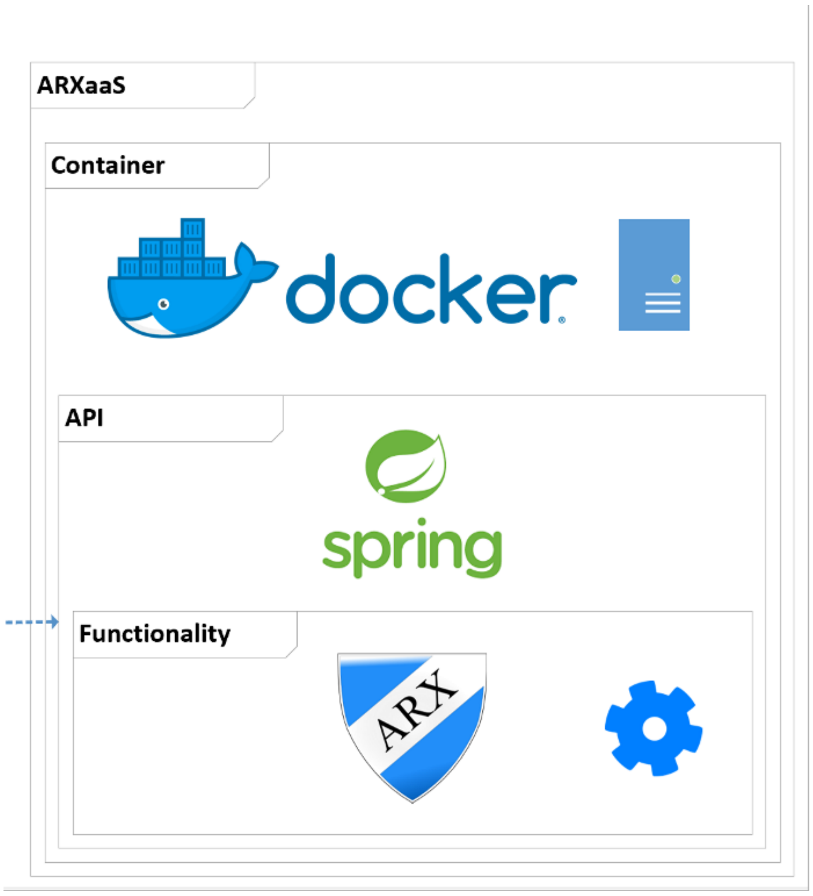


Figure 41 - ARXaaS tools and libraries diagram

A simplified rundown of the service's functionality, would be initiated upon receiving a POST HTTP request, to either the **/analyze**, **/anonymize** or **/hierarchy** endpoint. The request is sent to the /anonymize endpoint. This request would consist of a dataset to be anonymized and the anonymization parameters. The service's objective would be to return a transformed dataset with the specified level of anonymity. The service will have disposed of as little information as possible in order to reach the specified level, and the dataset's re-identification risks will have been reduced as a result.

The team's objective for ARXaaS has been to create a service capable of handling the described workload, and to offer it as a "Dockerizable", easily deployable and scalable service on platforms like Kubernetes.

5.2.2 Release Pipeline

ARXaaS utilizes a rich *CI/CD* pipeline to verify changes to the codebase, and to manage releases and deployments of new versions. See Chapter 3.2.1.2 Continuous integration and Continuous delivery, for a more detailed description on the reasoning and overview of the *CI/CD* pipelines developed during the project.

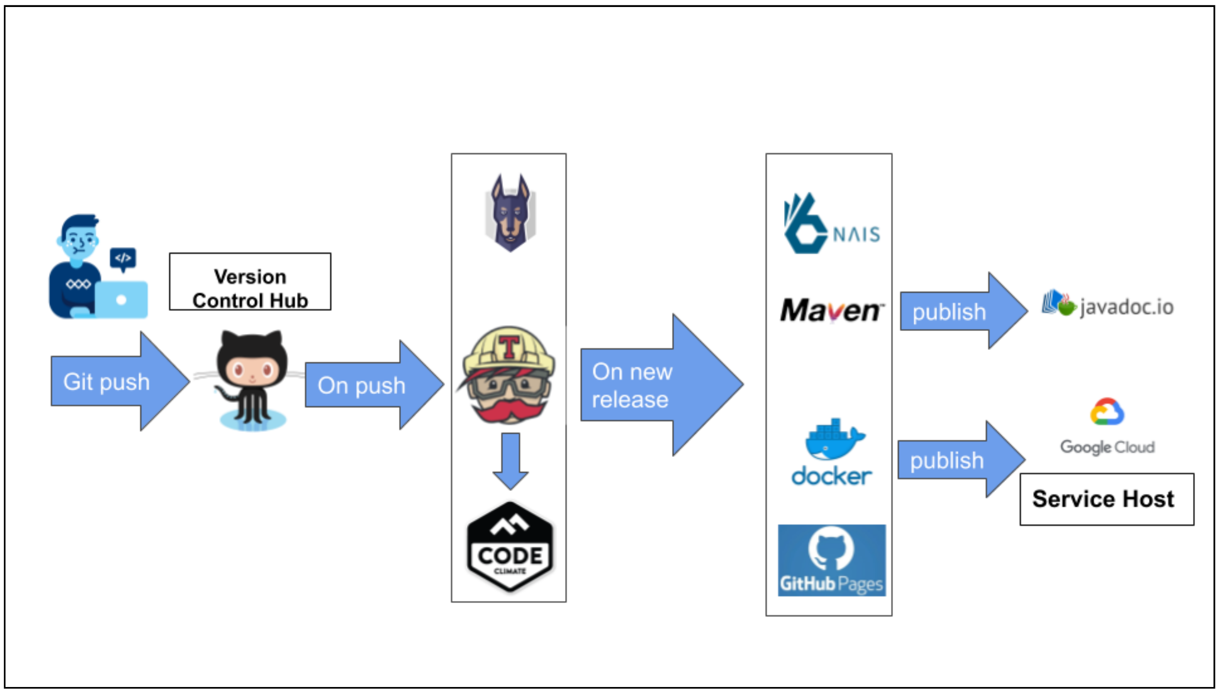


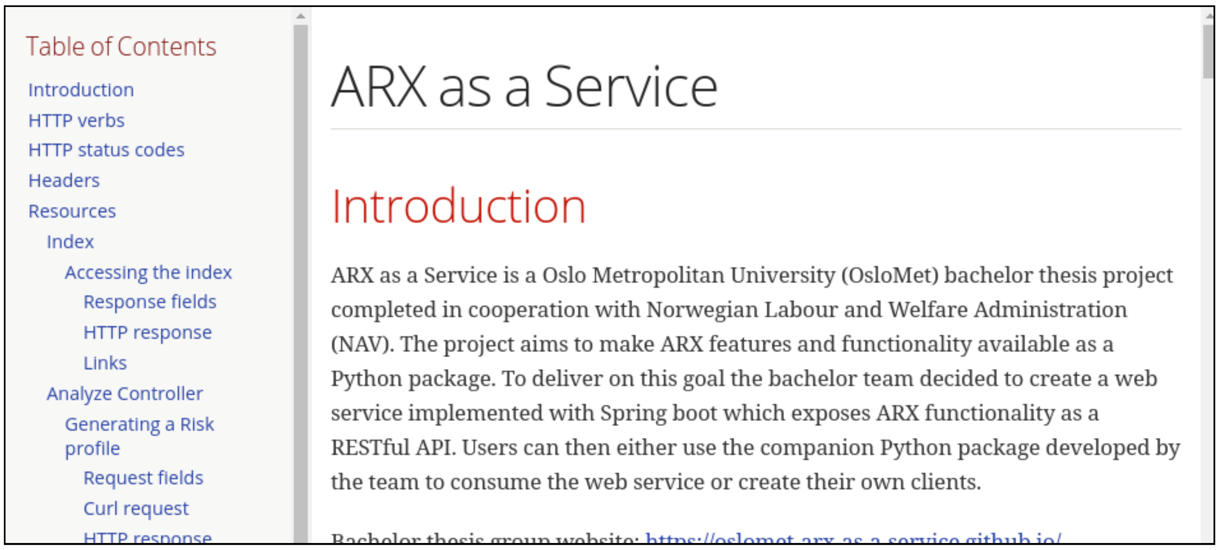
Figure 42 - Diagram showing the ARXaaS CI/CD pipeline

Configuration for each of the steps below can be seen in the .travis file in the ARXaaS Github repository[65], with the exception of the NAIS platform.

API Documentation

The ARXaaS API documentation is generated using the Spring REST docs library. Spring REST docs is a Maven plugin that uses tests to generate and validate documentation regarding a REST API. The benefit of Spring REST Docs over other API documentation tools like Swagger[66], is that the documentation is verified to be correct by a test. ARXaaS uses JUnit and Spring MockMvc to create system tests that test the API with test data and generate documentation from the tests. The generated documentation is published to Github pages[67] to host the documentation.

[65] ARXaaS travis file - https://github.com/oslomet-arx-as-a-service/ARXaaS/blob/master/.travis.yml
[66] Swagger homepage https://swagger.io/
[67] Github pages - https://pages.github.com/



ARX as a Service

Introduction

ARX as a Service is a Oslo Metropolitan University (OsloMet) bachelor thesis project completed in cooperation with Norwegian Labour and Welfare Administration (NAV). The project aims to make ARX features and functionality available as a Python package. To deliver on this goal the bachelor team decided to create a web service implemented with Spring boot which exposes ARX functionality as a RESTful API. Users can then either use the companion Python package developed by the team to consume the web service or create their own clients.

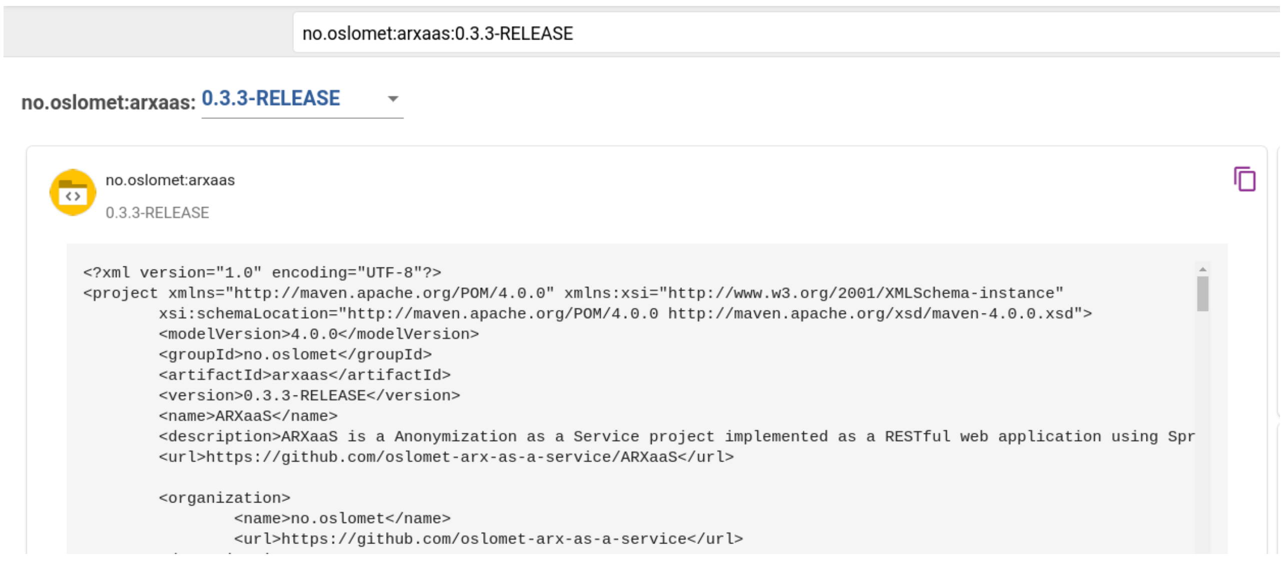
Figure 43 - Screengrab from ARXaaS API documentation[68]

Maven Central

Maven Central[69] is an online repository for sharing Java libraries. With the help of the *CI/CD* pipeline, the team can effortlessly upload every new version of the project to the Maven Central Repository. All projects uploaded to Maven Central are available world-wide as JARs and as Maven Dependencies[70]. As the Maven Central Repository is very accessible, it can be utilized to provide continuous world wide availability of the project's latest version, which the team used for the benefit of the *CI/CD* pipeline.

As a bonus, all libraries uploaded to Maven Central Repository automatically gain their own Javadoc web page hosted by javadoc.io[71]. Obviously, ARXaaS has its own Javadoc too[72].

[68] ARXaaS API docs page - https://oslomet-arx-as-a-service.github.io/ARXaaS/
[69] Maven Central Repository - https://search.maven.org/
[70] Maven Dependency - https://maven.apache.org/guides/introduction/introduction-to-dependency-mechanism.html
[71] Javadoc.io homepage - https://javadoc.io/
[72] ARXaaS Javadoc - https://javadoc.io/doc/no.oslomet/arxaas/0.3.3-RELEASE


no.oslomet:arxaas:0.3.3-RELEASE

no.oslomet:arxaas: 0.3.3-RELEASE

no.oslomet:arxaas
0.3.3-RELEASE

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <groupId>no.oslomet</groupId>
        <artifactId>arxaas</artifactId>
        <version>0.3.3-RELEASE</version>
        <name>ARXaaS</name>
        <description>ARXaaS is a Anonymization as a Service project implemented as a RESTful web application using Spr
        <url>https://github.com/oslomet-arx-as-a-service/ARXaaS</url>

        <organization>
                <name>no.oslomet</name>
                <url>https://github.com/oslomet-arx-as-a-service</url>
```

Figure 44 - Screengrab of ARXaaS on Maven Central[73]

Docker Hub

Every new release of ARXaaS is packaged as a container and released on Docker Hub[74]. Docker Hub is the hosting and distribution platform for docker containers. You can find the ARXaaS Docker Hub page here: https://hub.docker.com/r/arxaas/aaas

NAIS

The deployment to NAIS is handled by NAV IT AI-labs internal Jenkins server. Jenkins is a CI/CD automation tool, similar to Travis CI. NAIS have strict protections for its platform and the setup of the Jenkins server will not be described further in this report, ARXaaS is cloned from Github and re-compiled on the Jenkins server. After verifying the build, ARXaaS is deployed with the resources specified in the .nais file located in the root directory in the ARXaaS project.

5.2.3 Technologies

This section will describe the main technologies, libraries and dependencies used by the ARXaaS application. Some have been omitted for brevity see the ARXaaS pom.xml[75] file for the full list of dependencies and plugins.

5.2.3.1 Runtime

ARXaaS is developed with OpenJDK 11. OpenJDK 11 was decided on after consulting with NAV IT's application development teams. OpenJDK is a open source implementation of the Java language and runtime. The main reason for using OpenJDK is that Oracle, the main developers and owners of the Java programming language, have recently changed their licensing[76]. The new license is more restrictive and have implications for how ARXaaS could

[73] ARXaaS on Maven Central - https://search.maven.org/artifact/no.oslomet/arxaas/0.3.3-RELEASE/jar
[74] Docker Hub - https://hub.docker.com/
[75] ARXaaS pom file - https://github.com/oslomet-arx-as-a-service/ARXaaS/blob/master/pom.xml
[76] Oracle Java SE Licensing FAQ - https://www.oracle.com/technetwork/java/javase/overview/oracle-jdk-faqs.html

be run in production. The team therefore decided to use OpenJDK and its runtime, since it can be used in production for commercial use without paying a license fee. OpenJDK 11 is the latest Java Long Term Support (LTS) version and was is therefore a good balance of new features and longtime support.

5.2.3.2 Building and packaging

5.2.3.2.1 Apache Maven

Apache Maven is a complete software project tool for Java or JVM based software projects. From Apache Maven's homepage[77]:

Maven's primary goal is to allow a developer to comprehend the complete state of a development effort in the shortest period of time. In order to attain this goal, there are several areas of concern that Maven attempts to deal with:

Making the build process easy
- Providing a uniform build system
- Providing quality project information
- Providing guidelines for best practices development
- Allowing transparent migration to new features

ARXaaS uses Maven to run the test suite, generate the REST API documentation, generate the Javadoc, package the product artifacts, publish ARXaaS to Maven Central through the CI/CD pipeline.

5.2.3.2.2 Docker

Docker[78] is a independent platform and specification that enables organizations to seamlessly build, share and run any application. ARXaaS main deployment format is as a docker container. NAIS, the application platform in NAV IT is designed to run docker containers. The ARXaaS Dockerfile is built with **openjdk:11-jdk**[79] as the base image.

5.2.3.3 Libraries and Frameworks

5.2.3.3.1 Runtime

Libraries and frameworks that are primarily used when the service is running. ARXaaS uses several libraries, only the most important will be listed here. For the full list see the ARXaaS pom.xml dependencies section[80]

[77] What is Maven - https://maven.apache.org/what-is-maven.html
[78] Docker - https://docs.docker.com/get-started/
[79] Openjdk Docker Hub - https://hub.docker.com/_/openjdk/
[80] ARXaaS pom file - https://github.com/oslomet-arx-as-a-service/ARXaaS/blob/master/pom.xml

Spring Boot
ARXaaS uses **Spring Boot 2.1.2.RELEASE.** Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can "just run". The team has decided to utilize Spring as its backend framework to deliver a web service in accordance with the service architecture. Spring is the defacto standard for Java web applications and has great libraries for development of secure, scalable web applications.

Spring Boot Starter Web
Spring Starter for building web, including RESTful, applications using Spring MVC. It uses Tomcat as the default embedded Servlet/JSP container.

Spring Boot Actuator
Actuator endpoints let you monitor and interact with your application. It includes a number of built-in endpoints and lets you add your own. For example, the health endpoint provides basic application health information.

ARX
ARXaaS uses **ARX 3.7.1**. ARX is a comprehensive open source software for anonymizing sensitive personal data. It supports a wide variety of privacy and risk models, methods for transforming data and methods for analyzing the usefulness of output data. ARXaaS uses the ARX core library to implement all of the core functionality in the service.

5.2.3.3.2 Development

Libraries and frameworks that are primarily used during development and other purposes not directly related to running the service.

JUnit
JUnit is an open source Unit Testing Framework for the JVM.

Jacoco
JaCoCo maven plugin to generate code coverage reports for Java projects.

Spring REST docs
From Spring REST docs documentation[81]:

Spring REST Docs helps you to document RESTful services. It combines hand-written documentation written with Asciidoctor and auto-generated snippets produced with Spring MVC Test. This approach frees you from the limitations of the documentation produced by tools like Swagger. It helps you to produce documentation that is accurate, concise, and well-structured.

[81] Spring REST Docs - https://spring.io/projects/spring-restdocs

5.2.4 Architecture

ARXaaS architecture follows the normal three layers architecture with some deviation. A typical *CRUD application, u*ses a application layer, service layer, data layer architecture. Roughly dividing the controls accepting requests, the services handling the requests and the data layer which interacts with the necessary data from the database to complete the request. ARXaaS is not a CRUD application, it can be categorized as a data transform service. In all use cases it receives data from a service user, does some computation using the ARX library and returns the result. The architecture ARXaaS has been developed as a typical controller, service, data layer architecture, as it was the architecture the team was most familiar with. As the team understood more of the problem domain the architecture change to a more domain centric model with a fat domain model. The figure below is a overview the ARXaaS architecture. A more detailed figure is included in chapter 5.2.6 Functionality.

The **application layer** is responsible for transactions, keeping track of service context, creating and accessing *domain objects,* this layer uses the *Spring framework* extensively. It encompasses the **Controllers** and **Services** in the application

The **domain layer** is responsible for modeling the anonymization domain, it contains *domain objects* that encapsulates the core functionality of the service, this layers is implemented using regular java objects. This layer encompasses the **models**, **anonymize** and **analyzer** packages in the ARXaaS project.

The **infrastructure layer** is responsible for supporting the other layers. In ARXaaS this layer is mostly occupied by the ARX library that implements the algorithms and models the service uses.

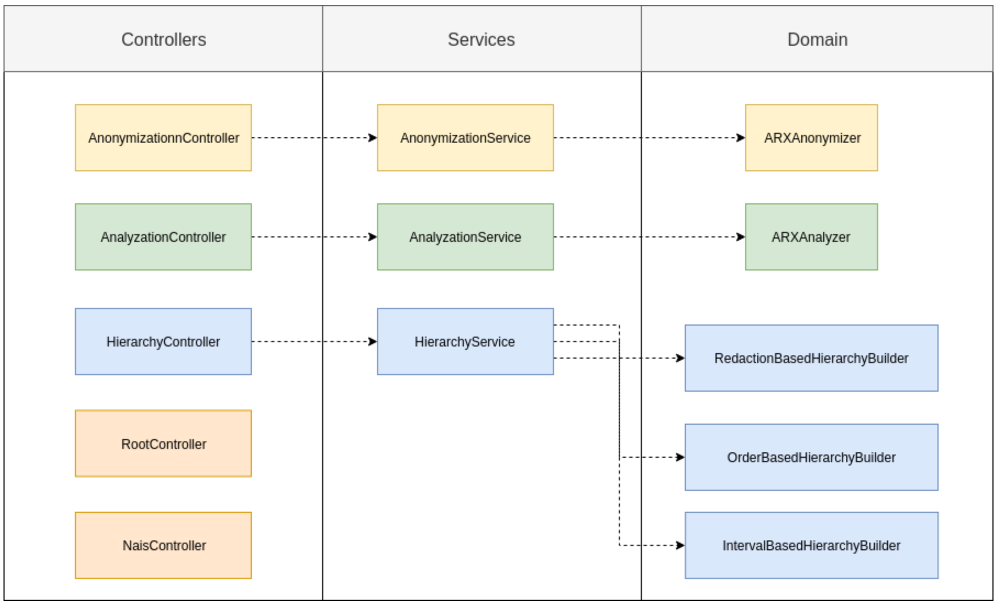


Figure 45 - ARXaaS architecture overview

5.2.5 Endpoints

This document will describe the functionality of each endpoint. This section will explain what each endpoint does and the response object it sends back to the user, as well as the different actions used with the request object sent to the endpoints.

5.2.5.1 Index

The index provides the entry point into the service. The response body consists of links to the available resources in accordance with the HATEOAS[82] protocol.

Accessing the index
A GET request is used to access the index

Links

Relation	Description
self	Link root resource
anonymize	Link anonymization controller
analyze	Link to analyze controller
hierarchy	Link to hierarchy controller

5.2.5.2 Analyzation

This endpoint can be reached by writing "{web address of the service}/api/analyze" and is an HTTP POST method.

The Analyze controller is used to generate risk profiles for a dataset. The REST controller receives a request object containing a dataset to be analyzed and the attribute type list of the dataset. The Controller returns an response object containing a risk profile that includes the re-identification risk and distribution of risk in a dataset.

Request fields

Path	Type	Description
data	Array	dataset to be anonymized

[82] REST cookbook HATEOAS - http://restcookbook.com/Basics/hateoas/

attributes	Array	Attributes types of the dataset

The risk profile object contains a re-identification risk that describes how anonymous the dataset is and the distribution of risk in the dataset.

Example of Analyzation HTTP request body:
This example can be found in the appendix in chapter 7.4 Analyzation HTTP JSON request body.

Example of Analyzation HTTP response body:
This example can be found in the appendix in chapter 7.5 Analyzation HTTP JSON response body.

5.2.5.3 Anonymization

This endpoint can be reached by writing "{web address of the service}/api/anonymize" and is an HTTP POST method.

The Anonymize controller is used to create new dataset anonymized according to provided privacy models and transformation models. The controller receives a request object containing a dataset to be anonymized, list of attribute types containing transformation models(hierarchies) and privacy models. The controller returns an response object containing a anonymized dataset, a risk profile, and metadata for the anonymization process.

Request fields

Path	**Type**	**Description**
data	Array	dataset to be anonymized
attributes	Array	Attributes types and transformation models to be applied to the dataset
privacyModels	Array	Privacy Models to be applied to the dataset
suppressionLimit	Number	Suppression limit to be applied to the dataset

Example of Anonymization HTTP request body:

This example can be found in the appendix in chapter 7.6 Anonymization HTTP JSON request body.

Example of Anonymization HTTP response body:

This example can be found in the appendix in chapter 7.7 Anonymization HTTP JSON response body.

5.2.5.4 Hierarchy

This endpoint can be reached by writing "{web address of the service}/api/hierarchy" and is a HTTP POST method.
The endpoint provides a interface to access ARX hierarchy builder features. This endpoint receives a request object containing the dataset column to create the hierarchy for, the builder type and builder specific attributes. This endpoint returns a response object containing the resulting hierarchy.
Currently the following builders are supported:

- Redaction based
- Interval based
- Order based

5.2.5.4.1 Redaction based hierarchy

This method builds hierarchies for categorical and non-categorical values using redaction. dataset items are:

1. aligned left-to-right or right-to-left,
2. differences in length are filled with a padding character.
3. Equally long values are redacted, character by character from left-to-right or right-to-left.

Request fields

Path	Type	Description
column	Array	List of values to create the hierarchy for
builder	Object	Object containing the different parameters on how to build the heirarchy for the dataset column
builder.type	String	Hierarchy builder type to use when creating the hierarchy
builder.paddingCharacter	String	Character to use when padding the values

builder.redactionCharacter	String	Character to use when redacting the values
builder.paddingOrder	String	Direction in which to pad the values in the column

Example of Redaction based hierarchy HTTP request body:
This example can be found in the appendix in chapter 7.8 Redaction based hierarchy HTTP JSON request body.
Example of Redaction based hierarchy HTTP response body:
This example can be found in the appendix in chapter 7.9 Redaction based hierarchy HTTP JSON response body.

5.2.5.4.2 Interval based hierarchy

This method builds hierarchies for non-categorical values by mapping them into given intervals.

Request fields

Path	**Type**	**Description**
column	Array	List of values to create the hierarchy for
builder	Object	Object containing the different parameters on how to build the heirarchy for the dataset column
builder.type	String	Hierarchy builder type to use when creating the hierarchy
builder.intervals	Array	List containing the different intervals to be generalized from and to
builder.intervals[].from	Number	Interval to generalize from
builder.intervals[].to	Number	Interval to generalize to
builder.intervals[].label	String	Optional label to replace the default generalized interval values
builder.levels	Array	List containing parameters on how to generalize the created intervals
builder.levels[].level	Number	Transformation level to create a generalization

builder.levels[].groups	Array	List containing parameters on how to group the generalized column new values
builder.levels[].groups[].grouping	Number	Number of items to be grouped from the new generalized column values
builder.levels[].groups[].label	Null	Optional label to replace the default generalized value
builder.lowerRange	Object	Object containing parameters on how to define the lower range interval
builder.lowerRange.snapFrom	Number	Value to snap from when a lower value than this defined value is discovered
builder.lowerRange.bottomTopCodingFrom	Number	Value to start bottom coding from
builder.lowerRange.minMaxValue	Number	If a value is discovered which is smaller than this value an exception will be raised.
builder.upperRange	Object	Object containing parameters on how to define the upper range interval
builder.upperRange.snapFrom	Number	Value to snap from when a higher value than this defined value is discovered
builder.upperRange.bottomTopCodingFrom	Number	Value to start top coding from
builder.upperRange.minMaxValue	Number	If a value is discovered which is larger than this value an exception will be raised.
builder.dataType	String	data type of the interval to generalize

Example of Interval based hierarchy HTTP request body:
This example can be found in the appendix in chapter 7.10 Interval based hierarchy HTTP JSON request body.

Example of Interval based hierarchy HTTP response body:
This example can be found in the appendix in chapter 7.11 Interval based hierarchy HTTP JSON response body.

5.2.5.4.3 Order based hierarchy

This method builds hierarchies for categorical and non-categorical values by ordering the dataset items and merging them into groups with the defined sizes.

Request fields

Path	Type	Description
column	Array	List of values to create the hierarchy for
builder	Object	Object containing the different parameters on how to build the hierarchy for the dataset column
builder.type	String	Hierarchy builder type to use when creating the hierarchy
builder.levels	Array	List containing parameters on how to generalize the dataset column
builder.levels[].level	Number	Transformation level to create a generalization
builder.levels[].groups	Array	List containing parameters on how to group the dataset column
builder.levels[].groups[].grouping	Number	Number of items to be grouped from the dataset column values
builder.levels[].groups[].label	String	Optional label to replace the default generalized value

Example of Order based hierarchy HTTP request body:
This example can be found in the appendix in chapter 7.12 Order based hierarchy HTTP JSON request body.

Example of Order based hierarchy HTTP response body:
This example can be found in the appendix in chapter 7.13 Order based hierarchy HTTP JSON response body.

5.2.6 Functionality

In this section the functionality and how it is implemented in the service. In the figure below the brown classes are classes with cross cutting concerns, Blue classes are responsible for hierarchy building, yellow classes are responsible for anonymization and green classes are responsible for risk analyzation. The figure is simplified to only include the most important classes. It does not include interfaces. Note that AnonymizationController and AnalyzeController both use the Request model class to represent requests to the controllers. The controllers needs are heavily overlapped so a single data structure is used to model a request.

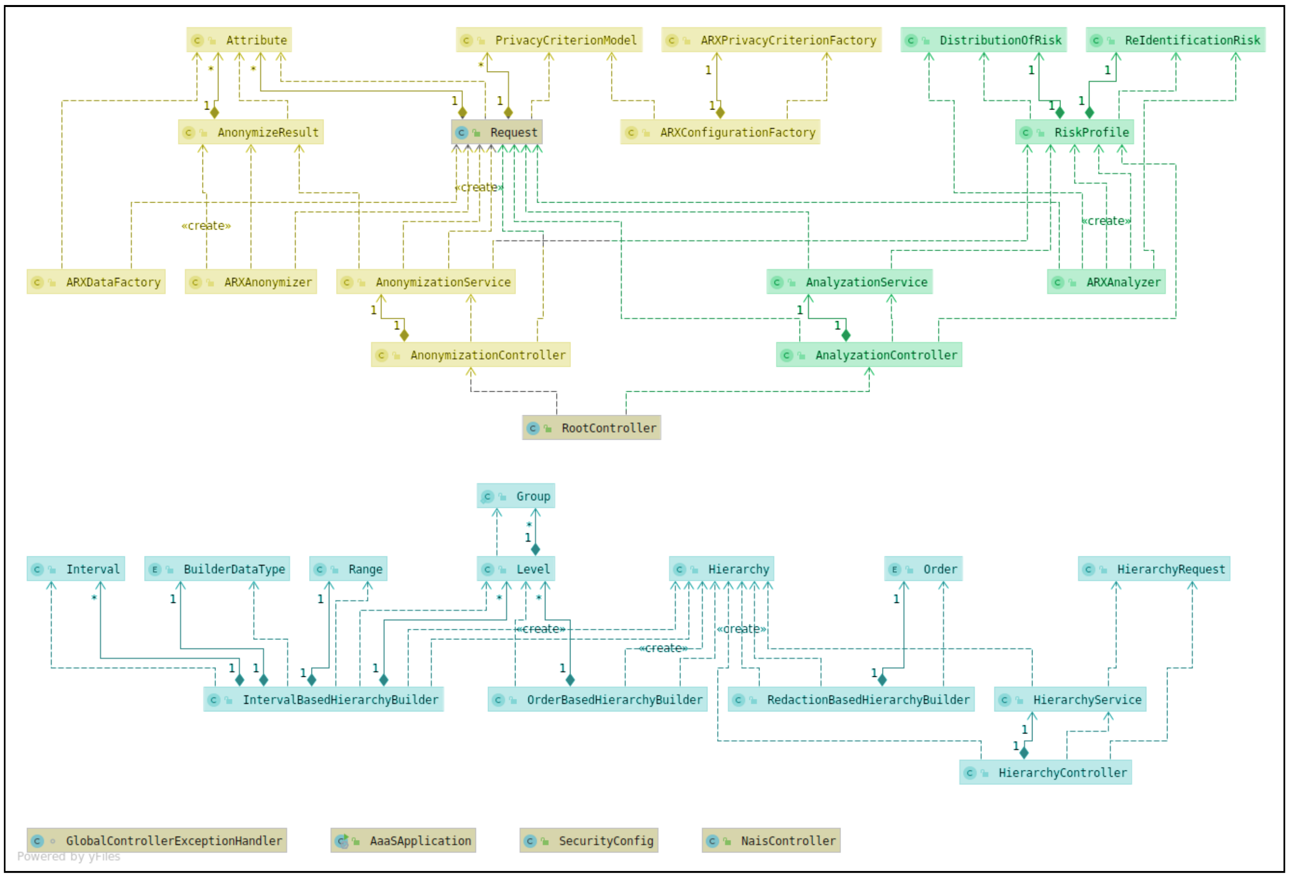


Figure 46 - UML diagram of ARXaaS

5.2.6.1 Controller Layer

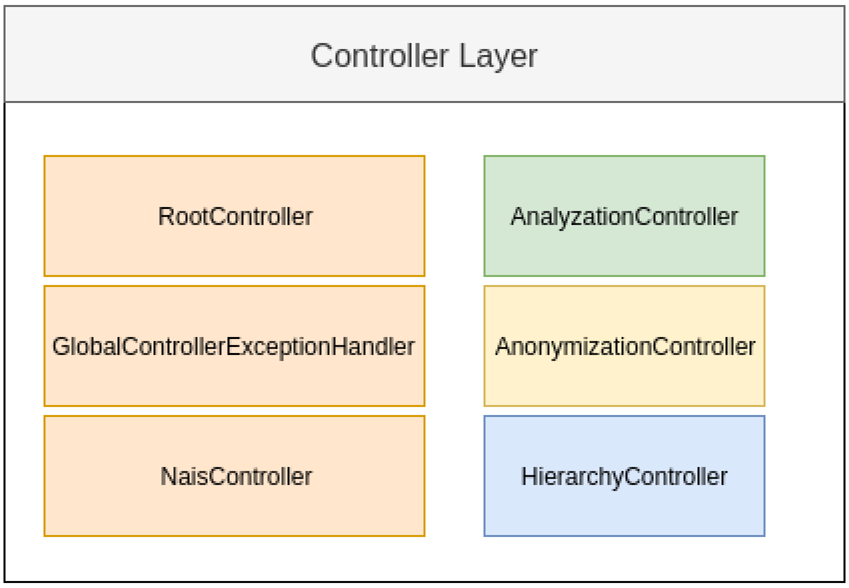


Figure 47 - ARXaaS controller layer class diagram

The service has three endpoints, to reach these endpoints a client must use an HTTP POST call to a web address that is running the service. These endpoints are written using RESTful design[83]

Example of a controller with a REST API endpoint:

```
@RequestMapping("/api/analyze")
public class AnalyzationController {

    @PostMapping
    public RiskProfile getPayloadAnalyze(@Valid @RequestBody Request         payload,
HttpServletRequest request) {
```

By following the REST architecture the web address is form in this format {web address of the service}/api/{function}. The resulting 3 end-points can then be reached by writing:

- {web address of the service}/api/analyze
- {web address of the service}/api/anonymize
- {web address of the service}/api/hierarchy

When an end-point receives a request object, it gets validated if it is correctly formatted. When the validation process fails the end-point will send a response in the form of an error message. This validation works as an extra safety net, because the clients are designed to always send a request object with the correct format. When the validation process succeeds the service will send a response object containing a JSON body that gets unpacked and

[83] REST - https://en.wikipedia.org/wiki/Representational_state_transfer

mapped by the clients. When the object is correctly formatted but contains invalid parameters, the end-point will send a response object containing the error message and how to correct the error.

Controller exception handling
ARXaaS uses a Exception controller to intercept exceptions thrown in the service when it propagates up to the controller layer.This Exception controller ensures that errors and thrown exceptions are handled and returned with correct HTTP status code and with a uniform message format.

```
/**
* Intercepts Exceptions thrown in the service. Ensures a uniform response format and that
a correct HTTP status is set
*/
@ControllerAdvice
class GlobalControllerExceptionHandler {

   private Logger logger = LoggerFactory.getLogger(this.getClass());

   /**
    * Handles all exceptions thrown unless cached by a more specific handler
    * @param ex Exception thrown
    * @param request WebRequest from client
    * @return ResponseEntity
    */
   @ExceptionHandler(Exception.class)
   public final ResponseEntity<Object> handleExceptionAllExceptions(Exception ex,
WebRequest request) {
       logger.error("Exception.class error, HttpStatus: INTERNAL_SERVER_ERROR,
ExceptionToString: ", ex);
       ExceptionResponse exceptionResponse = new ExceptionResponse(new Date(),
                       ex.getMessage(),
                       request.getDescription(false));
       return new ResponseEntity<>(exceptionResponse, HttpStatus.INTERNAL_SERVER_ERROR);
   }
```

5.2.6.2 Service Layer

Services are used to resolve queries from the controller layer.

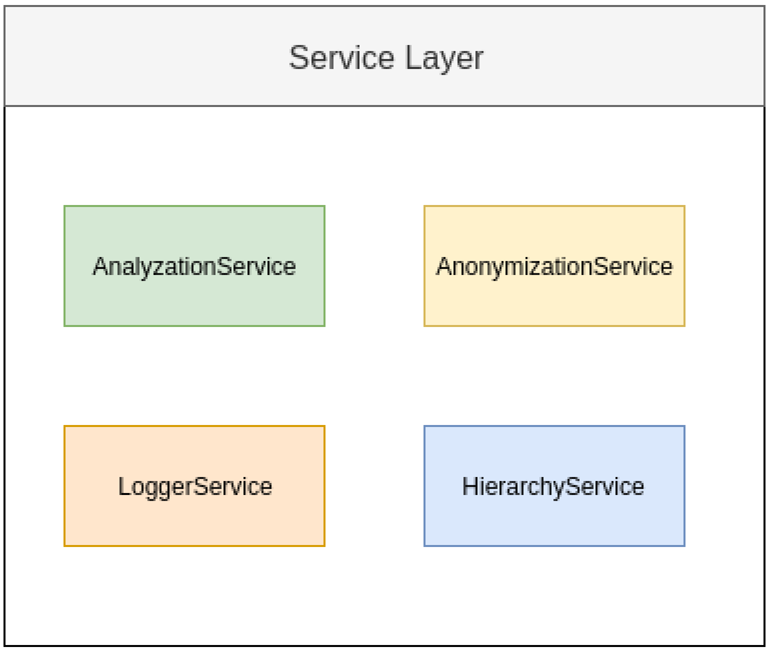


Figure 48 - ARXaaS service layer class diagram

The main services are relative light weight. Note that the Anonymization service is dependent on the Analyzation service. When resolving a anonymization request the services includes a risk profile for the anonymized dataset. This is done to reduce necessary calls to the ARXaaS service.

```
@Service
public class AnonymizationService {

   private final Anonymizer anonymizer;
   private final Analyzer analyzer;

   @Autowired
   public AnonymizationService(Anonymizer anonymizer, Analyzer analyzer){
       this.anonymizer = anonymizer;
       this.analyzer = analyzer;
   }

   public AnonymizationResultPayload anonymize(Request payload){

       AnonymizeResult result = this.anonymizer.anonymize(payload);
       Request afterAnalysisPayload =
               new Request(result.getData(), payload.getAttributes(), null, null);
       RiskProfile afterRiskProfile = analyzer.analyze(afterAnalysisPayload);
       return new AnonymizationResultPayload(result, afterRiskProfile);
   }
}
```

5.2.6.3 Domain Layer

A big part of the design of the domain layer was to keep a clean separation between ARXaaS domain models and ARX classes and components. This is to make a future developments that might even swap out ARX for another library, as easy as possible. The goal is to keep clear boundaries between the application, libraries and frameworks.

ARXaaS has interfaces for the domain services. Anonymizer which represents a objects that should do the actual anonymizing and Analyzer which should analyze re-identification risk.

Anonymizer interface

```
/**
*  Public Interface to be forfilled by data anonymizer classes
*/
public interface Anonymizer {

   /**
    * Method to run anonymization on data in the payload with the provided parameters in
the payload
    * @param payload {@link Request}  object containing the data to be anonymized and
params to use in anonymization
    * @return an {@link AnonymizeResult} object containing the best case anonymization
and statistics
    */
   AnonymizeResult anonymize(Request payload);
}
```

Classes that implemetents these interfaces can then use ARX to do their job.

Extract from the top of ARXAnonymizer class

```
/**
* Anonymizer class using the ARX library to implement the anonymization
*/
@Component
public class ARXAnonymizer implements Anonymizer {

   private final DataFactory dataFactory;
   private final ConfigurationFactory configFactory;
   private final Logger logger;
   private static final String exceptionError = "Exception error: %s";

   @Autowired
   public ARXAnonymizer(DataFactory dataFactory, ConfigurationFactory configFactory) {
       this.dataFactory = dataFactory;
       this.configFactory = configFactory;
       logger = LoggerFactory.getLogger(this.getClass());
   }
   /**
    * Method to run anonymization on data in the payload with the provided parameters in
the payload
    * @param payload {@link Request} object containing the data to be anonymized and
params to use in anonymization
    * @return an {@link AnonymizeResult} object containing the best case anonymization
and statistics
    */
   @Override
   public AnonymizeResult anonymize(Request payload) {
       Data data = dataFactory.create(payload);
       ARXConfiguration config = getARXConfiguration(payload);
       ARXResult result = getARXResult(data, config);
       return packageResult(result,payload);
   }
```

5.2.7 Security

ARXaaS is a service that is intended to process sensitive data, so it is of importance that its transactions are protected by end to end encryption. HTTPS is an acknowledged and robust protocol that offers suitable protection, but it places unique demands on every potential owner of a server running ARXaaS. HTTPS is not achieved by the push of a button, and that is why an unconfigured instance of ARXaaS will default to running with regular HTTP. In order to activate HTTPS protection, the owner must possess an SSL certificate provided by a CA (Certificate Authority), and it must be applied during the configuration of the ARXaaS instance. See the chapter on 6.1.1.2 HTTPS Configuration, for guide to setup ARXaaS with HTTPS.

5.2.7.1 SSL Handshake

This segment contains a general explanation of how the SSL Handshake works, and why it is essential to making HTTPS secure. Further, it explains how ARXaaS utilizes HTTPS for security.

1. A request sent from a client to a service supporting HTTPS is the first step in the SSL Handshake.
2. The service responds to the client's request with a new request containing the service’s certificate. The service will await a response from the client.
3. The client will verify the certificate authenticity with a request to the certificate's CA.
4. Response to the client is sent from the CA. The response contains information whether the service's certificate is valid and trusted.
5. The client will respond to the awaiting service accordingly, depending on the response content from the CA.

6. If the service reads OK from the response, the client and service will have established a secure connection, and they will proceed with execution of their originally intended transaction.

5.2.8 Monitoring

ARXaaS uses the Spring Actuator library to implement several endpoints in the service for gathering metrics on the application. The main type of endpoint ARXaaS uses is a prometheus[84] metrics endpoint. Prometheus is a popular and powerful open source metrics and logging solution.

Important metrics being recorded is:Important metrics being recorded is:

- Status of running containers
- Number of failed requests
- Memory usage
- CPU usage
- Response time

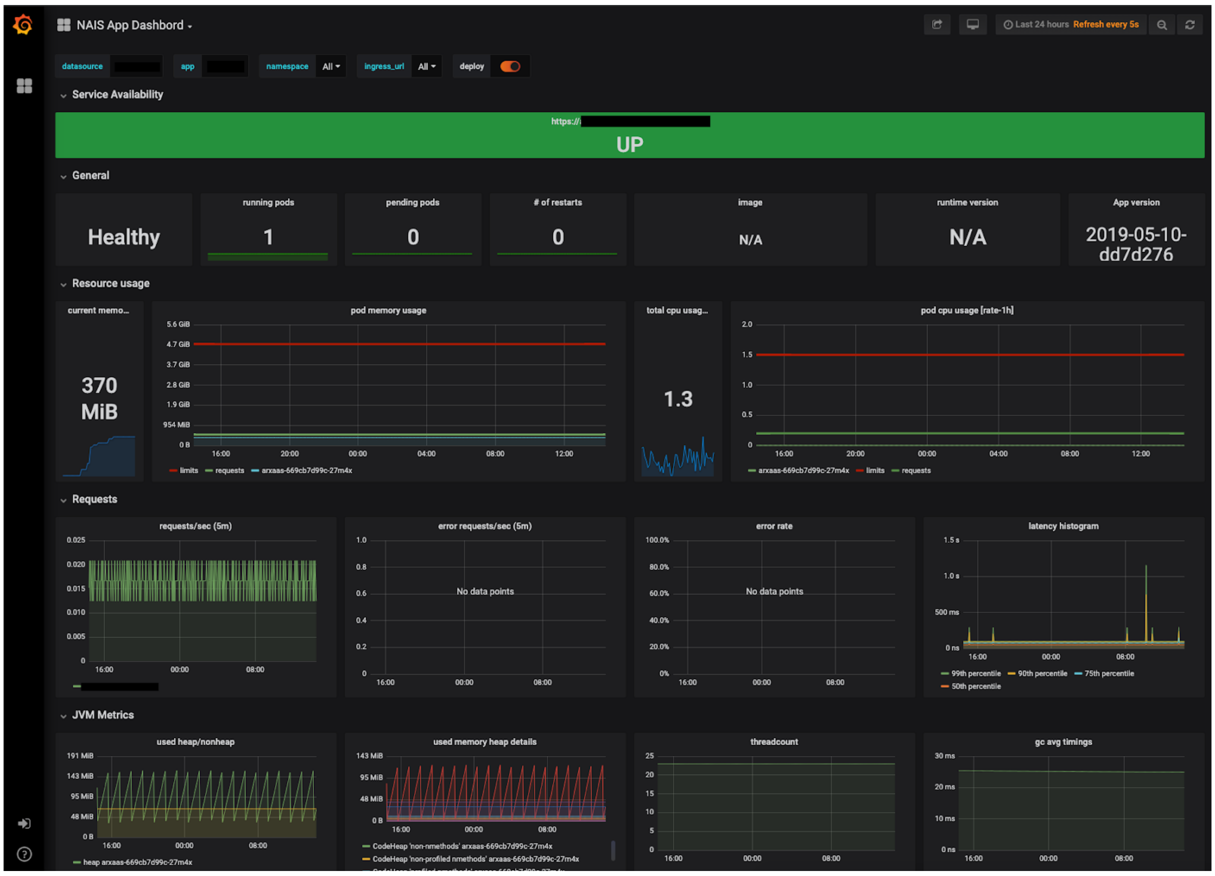


Figure 49 - Screen grab of ARXaaS metrics dashboard when running on NAIS

[84] Prometheus - https://prometheus.io/

5.2.9 Logging

ARXaaS has logging implemented using log4j. For every dataset that is analyzed and anonymized. The application provides metrics for received and completed requests. The log displays the size of the dataset, number of rows and columns, source IP, dataset bytesize, privacy model used, suppression limit and request processing time. In the case of an error or exception, a full stack trace is printed to make debugging faster and more efficient.

```
2019-05-10 11:48:39.237  INFO 29198 --- [nio-8080-exec-2]
n.o.a.c.AnonymizationController           : Request received, Size of
dataset: Number of rows = 12, Number of columns 3, Bytesize = 357,
Request Source IP = 0:0:0:0:0:0:0:1 Privacy models used = K-Anonymity,
Suppression Limit used = null
2019-05-10 11:48:40.403  INFO 29198 --- [nio-8080-exec-2]

n.o.a.c.AnonymizationController           : Request complete, Size of
dataset: Number of rows = 12, Number of columns 3, Bytesize = 360,
Request Source IP = 0:0:0:0:0:0:0:1 Request processing time = 1167
milliseconds
```

5.2.10 License

ARXaaS is distributed under the MIT license. See the ARXaaS License[85].

5.2.11 Error description

This segment contains a general explanation on the different error messages that ARXaaS sends as a response, and the format the message.

5.2.11.1 HTTP status codes

RESTful notes tries to adhere as closely as possible to standard HTTP and REST conventions in its use of HTTP status codes.

Status Code	Usage
200 OK	The request completed successfully
400 Bad Request	The request was malformed. The response body will include an error providing further information
404 Not Found	The requested resource did not exist
500 Internal Server Error	the server encountered an unexpected condition that prevented it from fulfilling the

[85] ARXaaS License - https://github.com/oslomet-arx-as-a-service/ARXaaS/blob/master/LICENSE

	request.

An error response contains a timestamp, a message containing the error and details telling which end-point the error originated.

```
"timestamp": "2019-05-16T14:53:02.326+0000",
"message": "no.oslomet.aaas.exception.UnableToAnonymizeDataException: Could not fulfill the pri
    attributes and hierarchies. A common cause of this error is more than one QUASIIDENTIFYING
"details": "uri=/api/anonymize"
```

Figure 50 - Error response contents

A 404 http status code happens when trying to reach an end-point that does not exist in the service.

A 400 http status code is shown, whenever ARXaaS wasn't able to fulfill the services it provides.

Example of 400 http responses:

- Unable to anonymize a dataset

```
"timestamp": "2019-05-16T15:20:15.458+0000",
"message": "no.oslomet.aaas.exception.UnableToAnonymizeDataException: Could not fulfill the privacy
    criterion set, Unable to anonymize the dataset with the provided attributes and hierarchies. A common
    cause of this error is more than one QUASIIDENTIFYING attribute without a hierarchy",
"details": "uri=/api/anonymize"
```

Figure 51 - Unable to anonymize error message

- Invalid attribute type

```
"timestamp": "2019-05-16T15:20:49.875+0000",
"message": "gender is not an attribute",
"details": "uri=/api/analyze"
```

Figure 52 - Invalid attribute type error message

- Invalid dataset or attribute type format

```
"timestamp": "2019-05-16T15:16:22.824+0000",
"message": "java.lang.IndexOutOfBoundsException: Column index out of range: -1. Valid: 0 - 1, Failed to
    create dataset. Check if dataset format and attribute dataset fields are correct",
"details": "uri=/api/anonymize"
```

Figure 53 - Failed to create dataset error message

5.3 Client side introduction

There are currently two clients available, that connect to the service. A python based client and a web application.

5.4 PyARXaaS

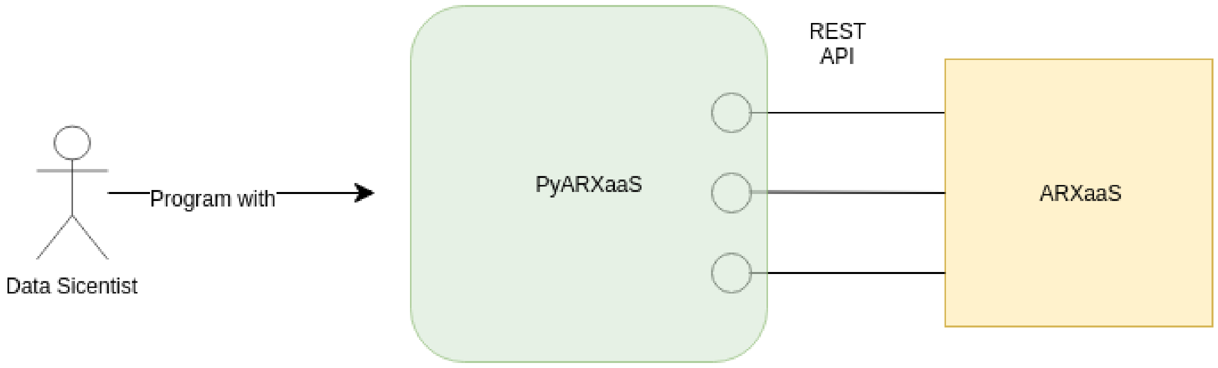


Figure 54 - diagram showing abstract usage of PyARXaaS

PyARXaaS is a Python client package that provides abstractions for interacting with a ARXaaS instance. It is inspired by other client packages like PyGithub[86]. It makes the integration of the risk analysis and de-identification functionality of ARXaaS as easy and intuitive as possible. The main user group of the package are data scientist that are familiar and accustomed to work with data in Python. The package delivers on the client requirement that the anonymization functionality was to be made available in Python. The package API has been developed with feedback from data scientist at NAV IT. The team notes that the final product has been very well received by the project stakeholders.

The package features

- ARXaaS class for connecting to and calling endpoints the ARXaaS service exposes.
- dataset class for encapsulating and configuring a dataset
- Privacy Model classes for creating and configuring the Privacy Models to use in anonymization.

Pandas integration

PyARXaaS was designed from day one with easy integration with pandas DataFrames[87] in mind. The pandas package is described in more detail in the runtime libraries section. It's important to note that pandas the most used library in the Python data science world. The main class in the package, the DataFrame[88], is a class that represents a table of data in a Python context. The DataFrame class is highly optimized and includes a lot of functionality for working with data. The team set a goal for for the PyARXaaS package to be as easy as

[86] PyGithub - https://github.com/PyGithub/PyGithub
[87] Pandas - https://pandas.pydata.org/
[88] Pandas DataFrame - https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html

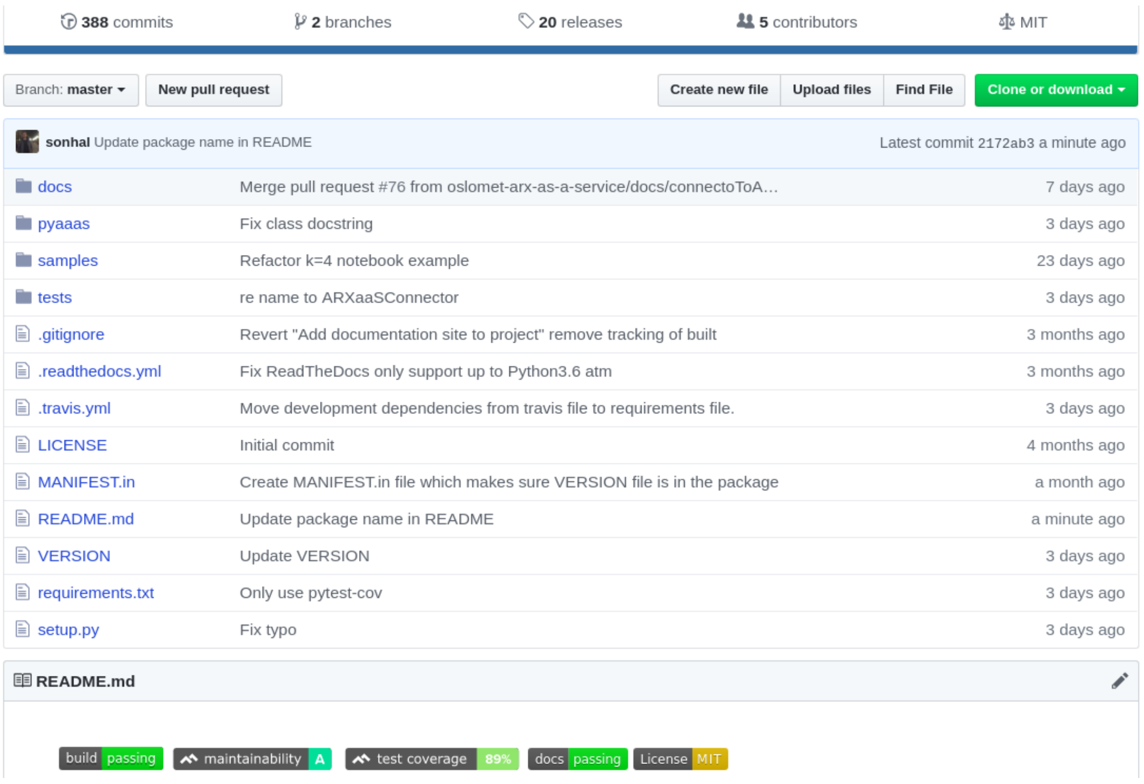
possible to work with pandas DataFrame and PyARXaaS support the conversion to and from DataFrame for several of the domain classes.

5.4.1 Short presentation

PyARXaaS is structured following the Python packaging use guide[89]. Below is a short summary of the package files and directories.

├── docs - **Contains text files and configuration scripts for the package documentation**
├── .gitignore
├── LICENSE
├── MANIFEST.in
├── pyarxaas - **Source directory, contains the packages Python source code**
├── README.md - **Project readme, contains a quick introduction and overview**
├── .readthedocs.yml
├── requirements.txt - **Contains development dependencies**
├── samples - **Contains sample Python scripts that use PyARXaaS**
├── setup.py - **Installation script for the package, contains package metadata**
├── tests - **Package test suite**
├── .travis.yml
└── VERSION

The package source files can be viewed in full on the PyARXaaS Githug page:
https://github.com/oslomet-arx-as-a-service/PyARXaaS


388 commits | 2 branches | 20 releases | 5 contributors | MIT

Branch: master | New pull request | Create new file | Upload files | Find File | Clone or download

sonhal Update package name in README — Latest commit 2172ab3 a minute ago

docs	Merge pull request #76 from oslomet-arx-as-a-service/docs/connectoToA…	7 days ago
pyaaas	Fix class docstring	3 days ago
samples	Refactor k=4 notebook example	23 days ago
tests	re name to ARXaaSConnector	3 days ago
.gitignore	Revert "Add documentation site to project" remove tracking of built	3 months ago
.readthedocs.yml	Fix ReadTheDocs only support up to Python3.6 atm	3 months ago
.travis.yml	Move development dependencies from travis file to requirements file.	3 days ago
LICENSE	Initial commit	4 months ago
MANIFEST.in	Create MANIFEST.in file which makes sure VERSION file is in the package	a month ago
README.md	Update package name in README	a minute ago
VERSION	Update VERSION	3 days ago
requirements.txt	Only use pytest-cov	3 days ago
setup.py	Fix typo	3 days ago

README.md

build passing | maintainability A | test coverage 89% | docs passing | License MIT

Figure 55 - PyARXaaS version control host repository

[89] Python packaging use guide - https://packaging.python.org/tutorials/packaging-projects/

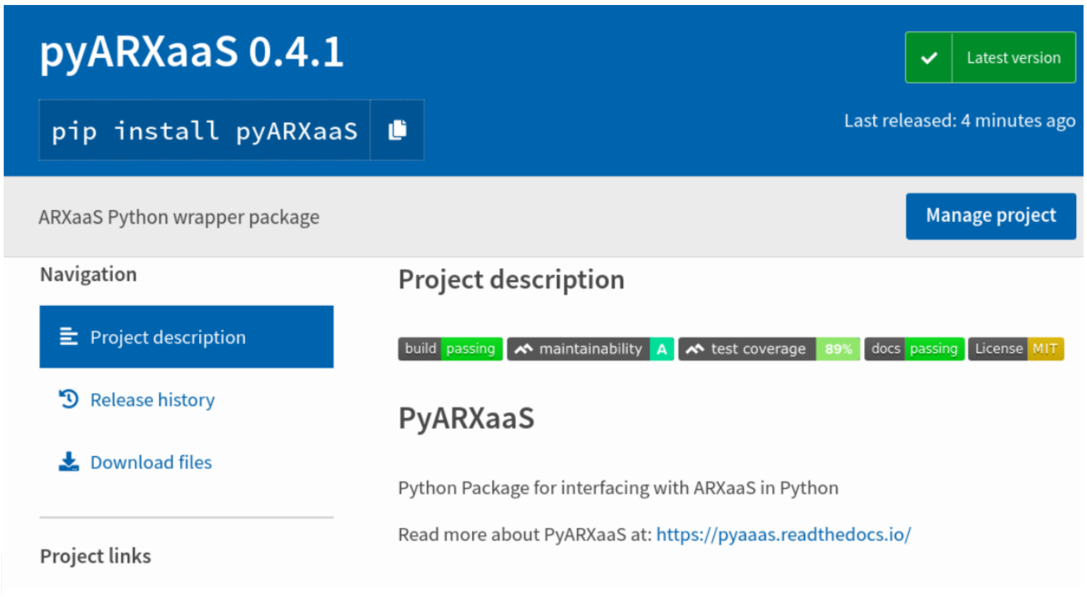
5.4.2 Packaging and release

Python packages can be released in several different distribution formats. A package release can consist of more than one format in the release artifact. PyARXaaS is packaged as source distribution and as a built distribution.

Source distribution
A distribution format commonly referred to as sdist is distribution of the Python source files and package metadata with only a simple compression step. A installation step is required on for users of the distribution.

Built distribution
A distribution format containing files and metadata that only needs to be moved to the correct location on the target system to be installed. PyARXaaS uses the wheel format[90] for built distribution.

Release
Every release of PyARXaaS is released to the Python Package Index (PyPI)[91]. The full version history with download links can be found on the package page https://pypi.org/project/PyARXaaS/.


pyARXaaS 0.4.1

pip install pyARXaaS

Latest version

Last released: 4 minutes ago

ARXaaS Python wrapper package

Manage project

Navigation

Project description

Release history

Download files

Project links

Project description

build passing | maintainability A | test coverage 89% | docs passing | License MIT

PyARXaaS

Python Package for interfacing with ARXaaS in Python

Read more about PyARXaaS at: https://pyaaas.readthedocs.io/

Figure 56 - showing the PyARXaaS PyPI page.

PyPI is the de-facto standard distribution platform for Python packages. Installing packages to be used in a Python project is simple. The Python installation comes with a tool; pip, for downloading and installing packages from PyPI to be used in the users script or application.

```
pip install pyarxaas
```

[90] PEP 427 -- The Wheel Binary Package - https://www.python.org/dev/peps/pep-0427/
[91] PyPI - https://pypi.org/

The team decided to use *semantic versioning* to version the package releases. Semantic versioning is used by most Python packages.

From Semantic Versioning 2.0.0[92]:
"Given a version number MAJOR.MINOR.PATCH, increment the:

MAJOR version when you make incompatible API changes,
MINOR version when you add functionality in a backwards-compatible manner, and
PATCH version when you make backwards-compatible bug fixes."

5.4.3 Release Pipeline

The reasoning and decision process behind the CI/CD pipeline is described in chapter 3.2.1.2 Continuous integration and Continuous delivery.

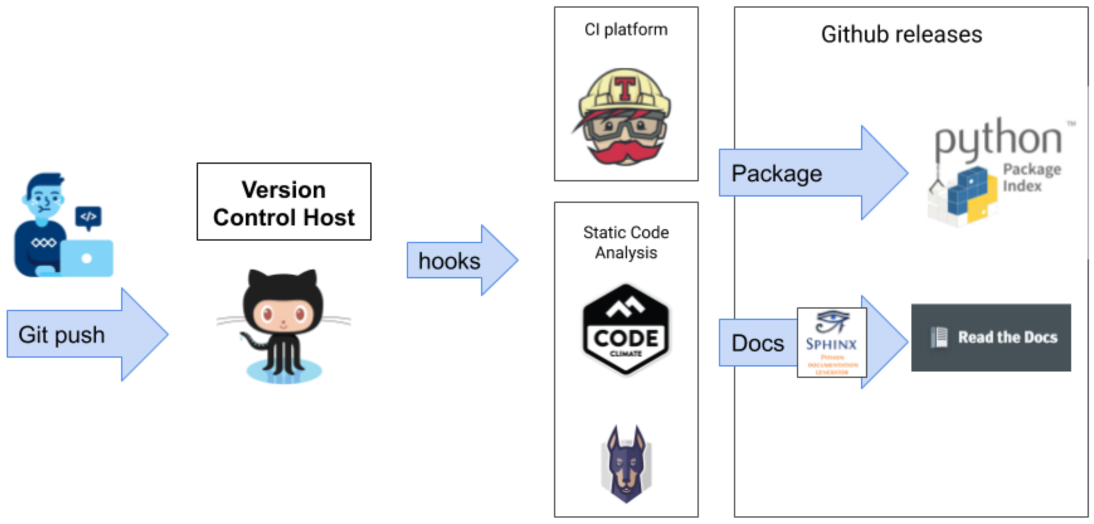


Figure 57 - Diagram showing the PyARXaaS CI/CD pipeline

The Diagram show the CI/CD pipeline for PyARXaaS. From the commit to Github to it is released as a package on PyPI. Developers on PyARXaaS publishes pull requests to the Github repository. Every pull request is then run through a Travis CI job[93]. The job consists of the following stages:

- Test
  The tests created under the **test/** directory in PyARXaaS is ran. If all tests are successful a test rapport is generated and published to Code Climate for processing.
- Deploy
  If the branch Travis CI is running is the master branch and the branch is tagged as a release Travis deploys the branch to PyPI as a new package version.

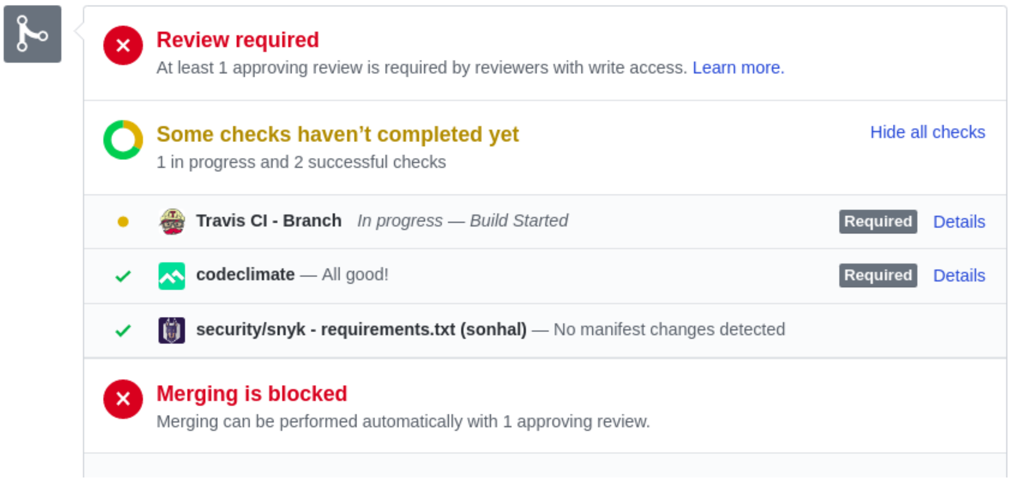
[92] Semantic Versioning 2.0.0 - https://semver.org/
[93] PyARXaaS Travis CI job script - https://github.com/oslomet-arx-as-a-service/PyARXaaS/blob/master/.travis.yml

In addition to the Travis CI job the following services are being ran at the branch.

- Code Climate
  Code Climate is triggered by the creation of a new pull request through a Github webhook[94]. It runs a code complexity test and flags problematic code. Code climate also receives the test rapport from the tests ran at Travis CI and provides feedback on the test coverage of the branch.

- Snyk
  Snyk runs a search of the dependencies defined in the requirements.txt file and flags any dependencies with a known vulnerability

The view the developer has of the status of their pull request looks like this.


Review required
At least 1 approving review is required by reviewers with write access. Learn more.

Some checks haven't completed yet
1 in progress and 2 successful checks
Hide all checks

●	Travis CI - Branch *In progress — Build Started*	Required	Details
✓	codeclimate — All good!	Required	Details
✓	security/snyk - requirements.txt (sonhal) — No manifest changes detected		

Merging is blocked
Merging can be performed automatically with 1 approving review.

Figure 58 - PyARXaaS pull request view on Github

New releases are controlled using the Github release system[95]. When a new release is created the code in master branch is packaged and published to PyPI.

Documentation

PyARXaaS uses the Read the Docs[96] service as its documentation hosting platform. Read the Docs simplifies software documentation by automating building, versioning, and hosting of documentation for projects. The hosting is free for open source projects like PyARXaaS. When a new commit is merged to master branch a new build of the documentation is triggered on Read the Docs. A badge showing the build status and serving as a link to the PyARXaaS documentation is available on the PyARXaaS Github page.

Figure 59 - of the documentation badge

[94] Github Web hook - https://developer.github.com/webhooks/
[95] Github releases - https://help.github.com/en/articles/creating-releases
[96] Read the Docs - https://readthedocs.org/

5.4.4 Technologies

PyARXaaS is written in Python 3. The project officially only supports Python 3.6 and upwards. Python is an interpreted, dynamically typed, object-oriented, high-level programming language used in many different software fields. Its main benefits is its expressive syntax and extensive standard library as well as one of the most extensive third-party library ecosystem.

“Often, programmers fall in love with Python because of the increased productivity it provides.” [97]

While Python got its start as a scripting and system automation tool it has gained popularity in the data-science field in recent years. NAV IT AI-lab data-scientists use python extensively in their day-to-day work.

5.4.5 Libraries

PyARXaaS leverages a couple of third-party packages. These packages are divided into runtime libraries that are used by PyARXaaS when executing and development libraries that are used for testing, documentation and other development related tasks.

5.4.5.1 Runtime libraries

The runtime libraries, referred to from here as dependencies, are set in the package setup.py file. Setup.py contains metadata for the project. The part that describes the package dependencies that must be installed to utilize the package is the *install_requires* field.

```
install_requires=["uplink==0.7.0",
                  "pandas==0.24.2",
                  "IPython==7.5.0"],
```

From the setup.py file https://github.com/oslomet-arx-as-a-service/PyARXaaS/blob/master/setup.py

The dependencies are installed automatically when installing the PyARXaaS package. The dependencies are versioned to increase stability between installations and to mitigate the risk of malicious software being installed through a corrupted dependency. Versioning combined with the Snyk tool described in the test documentation results in a robust risk mitigation.

[97] What is Python? Executive Summary - https://www.python.org/doc/essays/blurb/

Uplink[98]

Uplink is a python package intended to make creating specialized wrappers for web APIs much easier. It features a decorator oriented API for creating wrappers. Uplink is used in PyARXaaS for handling the HTTP connection to ARXaaS endpoints. This is abstracted in the ARXaaSConnector class.

```
from uplink import Consumer, get, headers, Path, Query, post, Body, json

class ARXaaSConnector(Consumer):
   """ Understands connection to ARXaaS endpoints"""

   def __init__(self, *args, **kwargs):
       super().__init__(*args, **kwargs)
       self._logger = logging.getLogger(__name__)

   @raise_for_status
   @json
   @post("api/anonymize")
   def anonymize_data(self, payload: Body):
       """Post data to AaaS Backend"""
```

pandas[99]

Pandas is an *open source*, high performance python package for data manipulation and analysis. As the pandas.DataFrame class is more or less the standard data structure for doing data analysis and manipulation PyARXaaS features easy conversion to and from the pandas.DataFrame class.

This is method in the Dataset class returns a pandas.DataFrame representation of the dataset object.

```
def to_dataframe(self) -> pandas.DataFrame:
   """
   Create pandas DataFrame of the dataset

   :return: pandas.DataFrame
   """

   return self._data.dataframe
```

IPython[100]

IPython is a toolkit package used to integrate with the Jupyter notebook editor. In PyARXaaS it is used to provide Jupyter notebook specific visualization.

[98] Uplink package documentation - https://uplink.readthedocs.io/en/stable/
[99] Pandas package documentation - https://pandas.pydata.org/
[100] Python package documentation - https://pypi.org/project/ipython/

5.4.5.2 Development libraries

Development libraries are dependencies used for testing, documentation generation or code analysis. These dependencies are not installed by the end user of the package, but by developers and continuous integration tools. These dependencies are managed and used by the package developers so they are not subject to the same strict versioning as the runtime dependencies. The dependencies are described in the package *requirements.txt* file. PyARXaaS requirements.txt can be found here:
https://github.com/oslomet-arx-as-a-service/PyARXaaS/blob/master/requirements.txt

At the writing of this document PyARXaaS contains the following development dependencies:

Sphinx[101]
Sphinx is a tool for creating beautiful documentation. It was originally created to generate the documentation for the Python language. It lets the users combine written documentation with the documentation generated from source code. PyARXaaS uses Sphinx to generate the user guides, tutorials and API documentation. Every new update of PyARXaaS triggers a new build of the documentation ensuring that the documentation stays up-to-date.

nbsphinx[102]
Nbshinx is a extension for Sphinx that makes it possible to include Jupyter notebooks in the Sphinx generated documentation. PyARXaaS documentation includes docs generated from Jupyter notebooks.

pytest[103]
Pytest is a testing framework for Python projects. It is a popular and widely used testing framework. Features a richer testing framework than the unittest library included in the Python standard library. PyARXaaS uses pytest in the continuous integration pipeline as pytest includes extensions for generating test reports used to generate the test coverage score displayed on the project Github page:
https://github.com/oslomet-arx-as-a-service/PyARXaaS

[101] Sphinx documentation - http://www.sphinx-doc.org/en/master/
[102] Jupyter Notebook Tools for Sphinx - https://nbsphinx.readthedocs.io/en/0.4.2/
[103] Pytest documentation - https://docs.pytest.org/en/latest/

5.4.6 Functionality

The package source code is located in the **pyarxaas** directory. It has the following structure:

```
pyarxaas/
├── aaas_connector.py - contains classes and functions for handling ARXaaS connection
├── arxaas.py - contains the public ARXaaS Wrapper class
├── converters.py - contains util code for conversion between different data formats
├── hierarchy - subpackage, contains hierarchy generation classes
├── __init__.py
├── models - subpackage, contains domain classes
└──  privacy_models.py - contains privacy model classes
```

5.4.6.1 Package components

Figure 60 shows a class diagram for the package. The purpose of the diagram is to give a overview of the functionality in the package and how it is structured. Green colored classes are classes used to connect to and interact with ARXaaS. The Orange are classes associated with hierarchy generation. Blue classes are associated with the dataset abstraction. Purple classes are model classes, abstractions for responses from ARXaaS. Red classes are Privacy Model classes used to configure a request to anonymize a dataset.

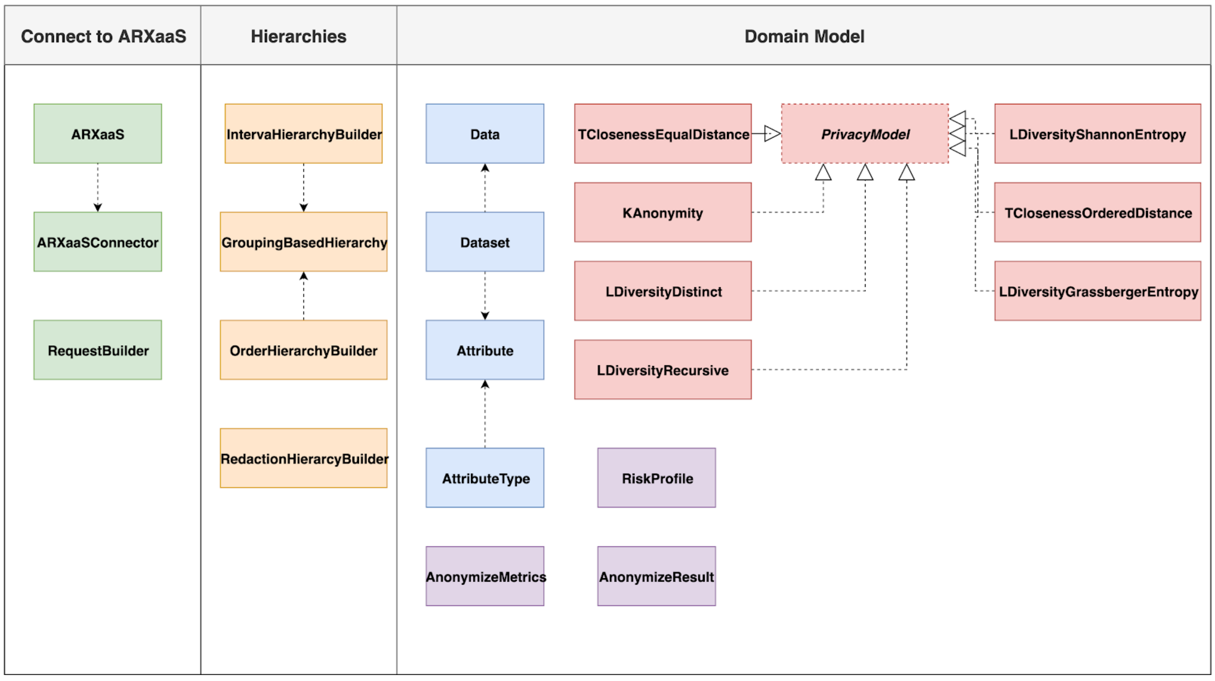


Figure 60 - Class diagram for PyARXaaS

5.4.6.1.1 Connecting to ARXaaS

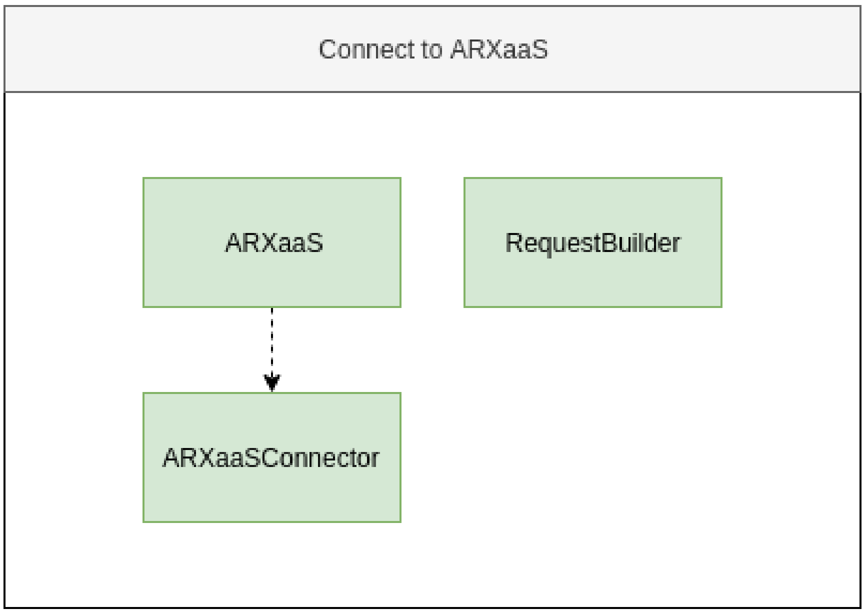


Figure 61 - Class diagram for ARXaaS abstraction classes

Figure 61 shows the classes associated with connecting to, and handling requests and responses from ARXaaS. The main class is the ARXaaS class which has the role of abstracting the service connection away for the user of the library. The ARXaaS class delegates to a ARXaaSConnector object for the HTTP request and response details. The RequestBuilder Class assists in building up a correctly formed JSON request object.

ARXaaS creation

```
# import the aaas module
from pyaaas import ARXaaS

# establishing a connection to the ARXaaS service using the URL
arxaas = ARXaaS("http://localhost:8080")
```

The ARXaaS class implements methods to anonymize a dataset according to provided Privacy Models, create a risk profile for a dataset object and generate hierarchies for from a dataset field/column.

risk_profile(dataset: dataset)
Returns a RiskProfile for the passed in dataset. See the chapter on 6.2.3.1 Analyze the risk of a dataset, for examples on how to use the method.

```
def risk_profile(self, dataset: dataset) -> RiskProfile:
    """
    Creates a risk profile for a provided dataset

    RiskProfile contains:
     - re-identifiaction risks
     - distributed risk
    :param dataset: dataset to create a risk profile for
    :return: RiskProfile
    """

    analyze_request = self._risk_profile_payload(dataset)
    response = self._risk_profile(analyze_request)
    metric_dict = json.loads(response.text)
    return RiskProfile(metric_dict)
```

anonymize(self, dataset: dataset, privacy_models, suppression_limit: float)
The method for doing anonymization of a dataset with the package. The method takes a dataset object to anonymize, privacy model(s) to apply and a optional supperions limit which configures the weighting of suppression vs data utility for the anonymization.

```
def anonymize(self, dataset: dataset, privacy_models,suppression_limit: float = None) ->
AnonymizeResult:
    """
    Attempt to anonymize a dataset with provided privacy models

    :param dataset: dataset to be anonymized
    :param privacy_models: privacy models to be used in the anonymization
    :param suppression_limit: suppression limit to be used in the anonymization
    :return: dataset with anonymized data
    """
    request_payload = self._anonymize_payload(dataset, privacy_models, suppression_limit)
    response = self._anonymize(request_payload)
    return self._anonymize_result(response)
```

hierarchy(self, redaction_builder, column)
The hierarchy method is called to create new generalization hierarchies for a dataset field/column. It accepts a HierarchyBuilder object and a column and passes them on to the ARXaaS service which creates the resulting hierarchy. The generated hierarchy is returned as a regular Python list[list].

```
def hierarchy(self, redaction_builder, column):
    """
    Creates a value generalization hierarchy with the passed in builder for the passed in
column.


    :param redaction_builder: a Hierarchy builder instance
    :param column: a list of values
    :return: list[list] containing the created hierarchy
    """

    request = redaction_builder._request_payload()
    request["column"] = column
    response = self._connector.hierarchy(request)
    response_dict = json.loads(response.text)
    return response_dict["hierarchy"]
```

5.4.6.1.2 Hierarchy Builders

Hierarchy builder Classes in the hierarchy sub-package represents different strategies for creating hierarchies. The classes in the hierarchy sub-package mirror the hierarchy builder classes in ARXaaS. The classes function is to provided objects representing a given hierarchy type. Letting the package user set hierarchy configurations through class constructors and methods. Figure 62 shows a class diagram for the hierarchy builder classes. Note that GroupingBasedHierarchy is a *abstract base class* (ABC).

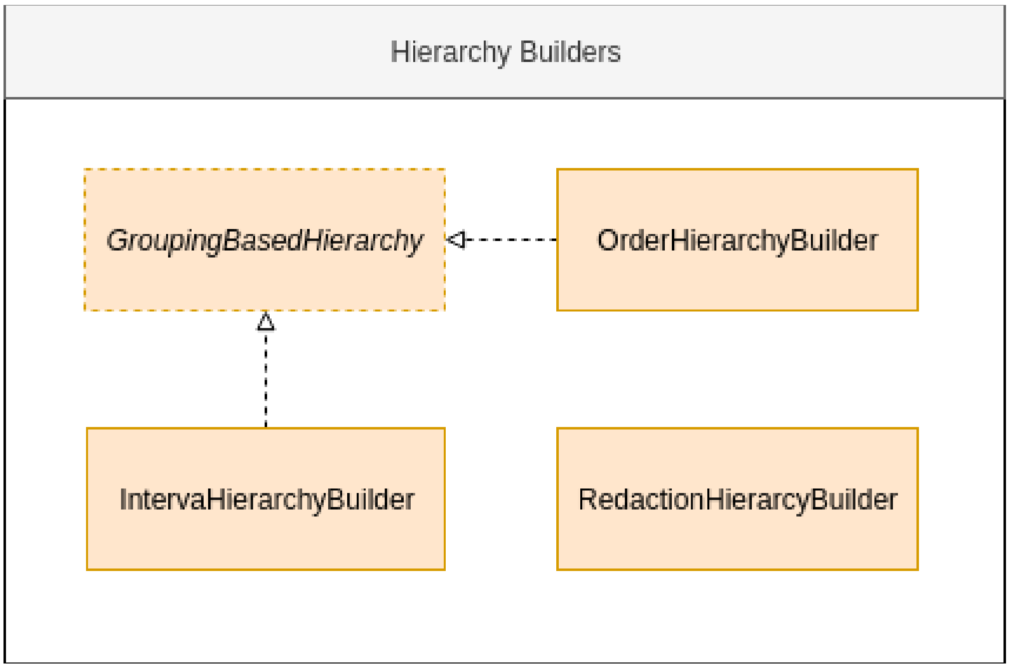


Figure 62 - Hierarchy Builder class diagram

RedactionHierarchyBuilder

RedactionHierarchyBuilder is one of the hierarchy builders available in the hierarchy package. It can be instantiated with configurations through its constructor. These configurations are:

- padding_char: The character to use when padding values in the generated hierarchy
- redaction_char: The character to use when redacting symbols from the column values in the generated hierarchy
- padding_order: The order in which to pad the values from the column, could be left to right or right to left
- redaction_order: The order to redact symbols from the column value in the generated hierarchy

The class uses default arguments for all the constructor parameters, making creating a default RedactionHierarchyBuilder simple. See the 6.2.4.4 Hierarchy Generation for more on the usage of RedactionHierarchyBuilder and the other hierarchy builder.

```
class RedactionHierarchyBuilder:
    """
    Understands building redaction based hierarchies
    """

    class Order(Enum):
        LEFT_TO_RIGHT = "LEFT_TO_RIGHT"
        RIGHT_TO_LEFT = "RIGHT_TO_LEFT"

    def __init__(self, padding_char:str = " ",
                 redaction_char: str = "*",
                 padding_order: Order = Order.RIGHT_TO_LEFT,
                 redaction_order: Order = Order.RIGHT_TO_LEFT):
        self._assert_padding_is_valid(redaction_char, padding_char)
        self._padding_char = padding_char
        self._reduction_char = redaction_char
        self._padding_order = padding_order
        self._redaction_order = redaction_order
```

5.4.6.1.3 Privacy Models

Privacy model classes represents the available Privacy models in ARXaaS service. The classes lets the package user instantiate Privacy models objects with user specified configuration through the class constructor. The Privacy Models objects can be passed along with a dataset object to the ARXaaS class anonymize method to anonymize a dataset according to the Privacy Model objects criterion and configuration.

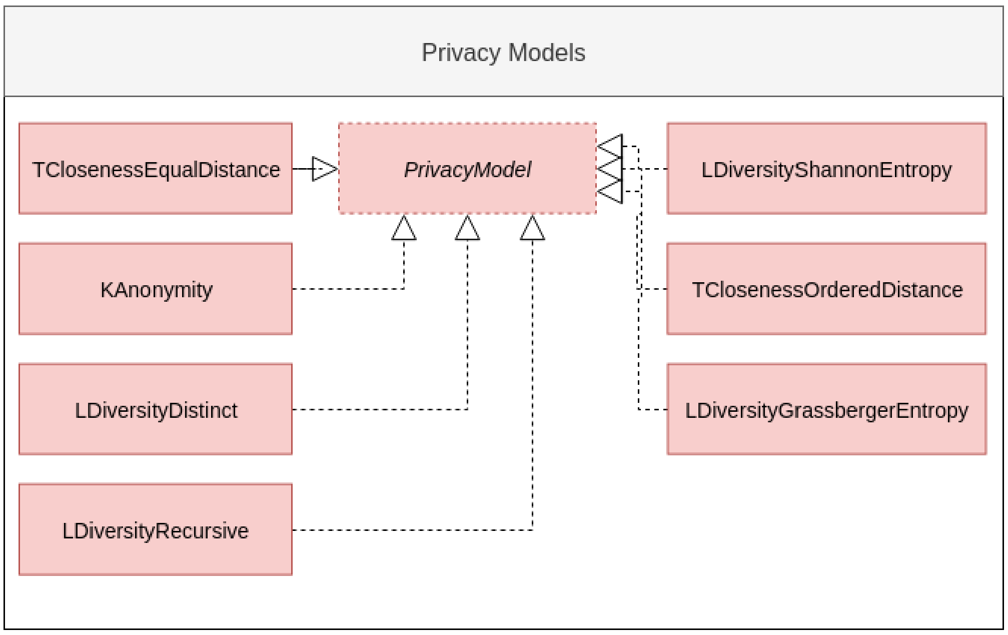


Figure 63 - Privacy models class diagram

PrivacyModel - ABC

PrivacyModel is the abstract base class (ABC) all other PrivacyModel subclass. In another language such as Java, PrivacyModel would most likely be implemented as a interface. Python has no concept of interface, so the closest thing was to implement the class as a ABC.

```
class PrivacyModel(ABC, Mapping):
   """
   Documentation of the privacy models implemented in the ARXaaS service and the
definition of the parameters
   each privacy model takes.
   """
   def __init__(self):
       self._anonymity_name = "Privacy Model"
       self._print_message = self._anonymity_name
       self._internal_dict = {}

   def __getitem__(self, item):
       return self._internal_dict[item]

   def __len__(self) -> int:
       return len(self._internal_dict)

   def __iter__(self):
       return iter(self._internal_dict)

   @property
   def name(self) -> str:
       return self._anonymity_name

   def __str__(self):
       return self._print_message

   def _payload(self):
       return {"privacyModel": self.name, "params": self._internal_dict}
```

KAnonymity

```
class KAnonymity(PrivacyModel):
   """
   Configuration class for K-Anonymity

   :param k: Value of K  to anonymize the dataset. K must have a value of 2 or higher to
take effect.

   """

   def __init__(self, k):
       super().__init__()
```

```
        self._internal_dict = {"k": k}
        self._anonymity_name = "KANONYMITY"
        self._print_message = f"KAnonymity(k={k})"
```

5.4.6.1.4 dataset class

The dataset class represents the concept of a tabular dataset containing fields referred to as attributes. The dataset attributes have the concept of AttributeType associated with them. See the chapter on 2.2.2 Anonymization, for more on attribute types. Optionally a attribute might have a generalization hierarchy associated with itself. The hierarchy describes how the attribute might be generalized in a anonymization process.

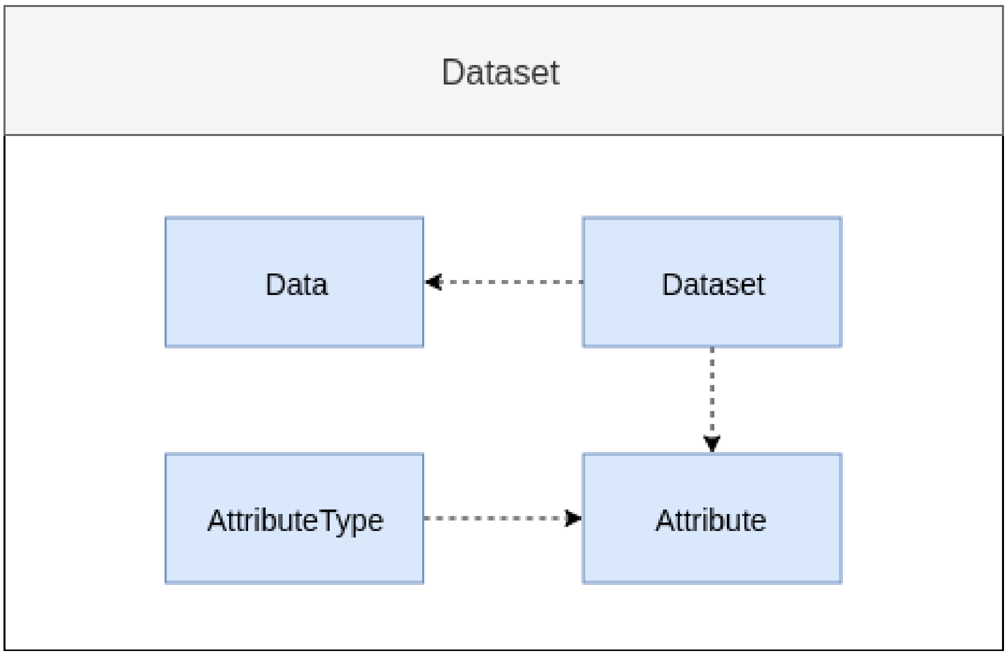


Figure 64 - dataset class diagram

Dataset construction

The Dataset class has a constructor that assures that the object created is well formed. If attribute_types are not passed they will be set to the default value; quasi identifying.

```
class dataset:
    """
    Understand tabular data containing personal data.
    """

    _DEFAULT_ATTRIBUTE_TYPE = AttributeType.QUASIIDENTIFYING

    def __init__(self, data: list, attribute_types: Mapping = None):
        if attribute_types is None:
            attribute_types = self._create_default_attribute_map(data[0])

        self._data = Data(data[0], data[1:])
        self._attributes = self._create_attributes(attribute_types)
```

To make construction of a Dataset more simple and to aid in creation from other Python objects *factory methods* have been implemented. The factory methods are class methods that are to be called on the class object to create a new instance of dataset.

Dataset factory methods

```python
@classmethod
def from_pandas(cls, dataframe: pandas.DataFrame):
    """
    Create a dataset from a pandas DataFrame

    :param dataframe: pandas Dataframe
    :return: dataset
    """

    headers = dataframe.columns.values.tolist()
    values = dataframe.values.tolist()
    data = [headers] + values
    return dataset(data=data, attribute_types=cls._create_default_attribute_map(headers))

@classmethod
def from_dict(cls, dictionary):
    """
    Create dataset from a python dictionary

    :param dictionary: Mapping object to create dataset from
    :return: dataset
    """

    df = pandas.DataFrame.from_dict(dictionary)
    return cls.from_pandas(df)
```

5.4.6.1.5 Response objects

ARXaaS response object are a group of objects representing the response data returned from calls to the ARXaaS service. These objects are instantiated by other objects the PyARXaaS package, usually the ARXaaS class. They are not intend for the user of the package to instantiate. The objects have properties to expose data generated by ARXaaS, and methods for easy conversion to other types such as pandas.DataFrame.

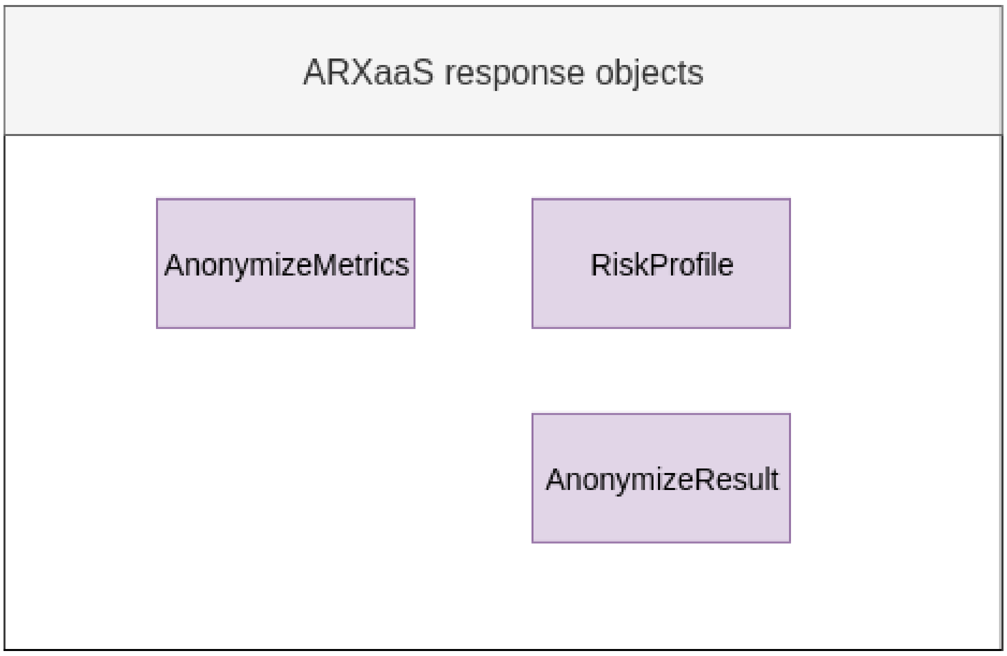


Figure 65 - ARXaaS response objects class diagram

RiskProfile

The RiskProfile class represents a re-identification risk profile for a Dataset. Returned when a user of PyARXaaS makes a call to the ARXaaS class `.risk_profile(dataset)` method. The methods accepts a dataset object and returns a RiskProfile object containing data describing different risk values calculated from the dataset.

```
class RiskProfile:
   """
   Represents the re-identification risks associated with a dataset
   """

   def __init__(self, metrics: Mapping):
       self._re_identification_of_risk =
copy.deepcopy(metrics["reIdentificationRisk"]["measures"])
       self._distribution_of_risk = copy.deepcopy(metrics["distributionOfRisk"])
       self._attacker_success_rate =
copy.deepcopy(metrics["reIdentificationRisk"]["attackerSuccessRate"]["successRates"])
       self._quasi_identifiers = metrics["reIdentificationRisk"]["quasiIdentifiers"]
       self._population_model = metrics["reIdentificationRisk"]["populationModel"]
```

RiskProfile exposes properties for risks, and attacker success rates. See the chapter on 2.4 Risk assessment, for more explanation on the different risks and attack models.

```
@property
def re_identification_risk(self):
    """
    Re-identification risk metrics for a given dataset

    :return: dict containing re-identification metrics
    """
    return copy.deepcopy(self._re_identification_of_risk)

@property
def distribution_of_risk(self):
    """
    Distribution of risk for a given dataset

    :return: dict containing the distribution of risks in a given dataset
    """
    return copy.deepcopy(self._distribution_of_risk)

@property
def attacker_success_rate(self):
    """
    Attacker success rates against re-identification for a given dataset

    :return: dict containing the attacker success rate.
    """
    return copy.deepcopy(self._attacker_success_rate)
```

RiskProfile implements methods for conversion of the risks to pandas.DataFrame for easy consumption and use of the data.

```
def re_identification_risk_dataframe(self) -> DataFrame:
    """
    Re-identification risk as a pandas.DataFrame

    :return: pandas.Dataframe with risk metrics
    """
    df = DataFrame([self._re_identification_of_risk])
    return df

def distribution_of_risk_dataframe(self) -> DataFrame:
    """
    Distribution of risk as a pandas.DataFrame

    :return: pandas.DataFrame
    """
    return DataFrame.from_dict(self._distribution_of_risk["riskIntervalList"])
```

5.4.7 Security

PyARXaaS being a package to be used in a users program or script, should not put to much restrictions and demands on the context the users uses the package in. It should not trick the use into making unsecure programs. The package should be configured by default to safe options, but always letting the user override where possible. How this materializes in the package is in the HTTPS certificate validation.

The service, ARXaaS, can be configured to use HTTPS or be located behind a proxy that adds HTTPS to the service calls. PyARXaaS uses the Uplink library, described in the library subchapter to complete HTTP(S) requests. By default this library does not accept self-signed certificates, certificates not issued by a trusted third-party CA authority. Often internal networks or test environments use HTTPS with self-signed certificates, so PyARXaaS should support this use case. By default PyARXaaS throws an exception if a call is made to a ARXaaS instance while using a self-signed certificate. Configured to allow self-signed certificates, PyARXaaS will resolve the request, logging a warning to the user.

5.4.8 Logging

PyARXaaS supports logging using the Python standard library logging module. This lets a user configure the logging for their script or program and PyARXaaS will follow those configurations.

5.4.9 License

PyARXaaS is distributed under the MIT license. See the PyARXaaS License[104]

[104] PyARXaaS License - https://github.com/oslomet-arx-as-a-service/PyARXaaS/blob/master/LICENSE

5.5 WebARXaaS

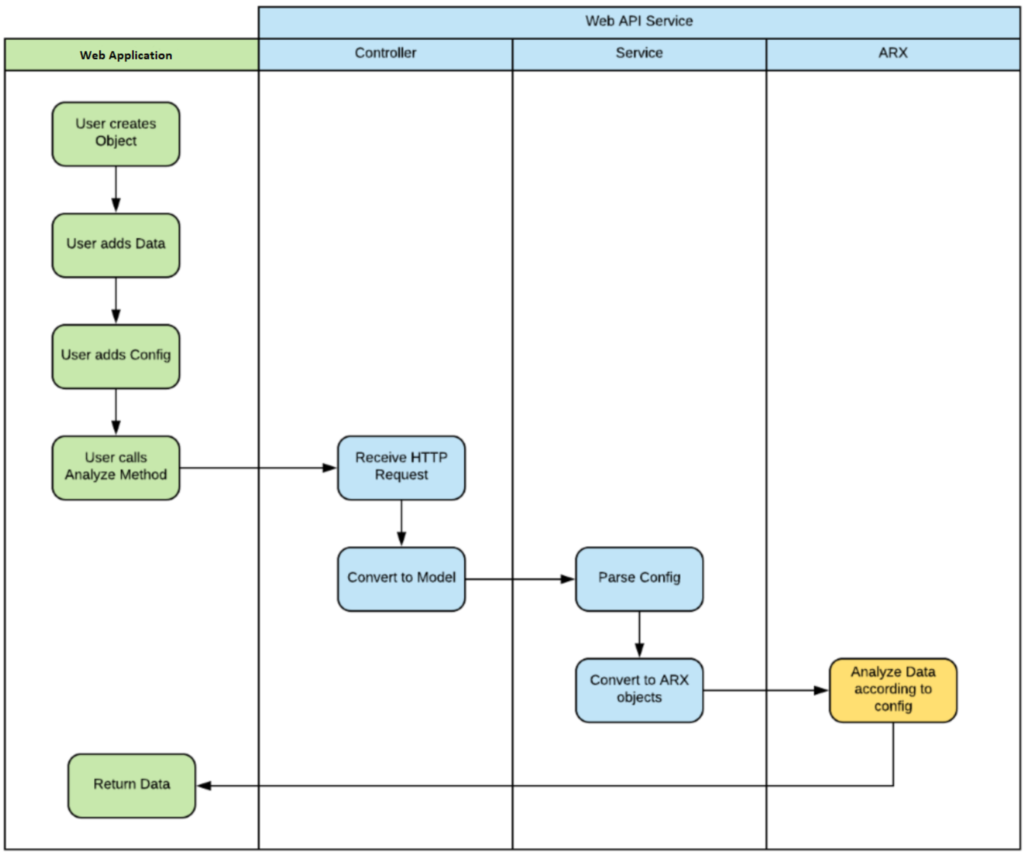


Figure 66 - Sequence diagram of WebARXaaS

5.5.1 Short presentation

As a stretch goal, the product owner wished for a way to quickly access the ARX functionality. It was therefore decided to implement an interactive web frontend, by taking advantage of the flexible REST API provided by the ARXaaS service. Making this client available will give the user the possibility to analyze or anonymize their data, without the need to install software on their local machine.

5.5.2 Technologies

The interactive Web service was implemented in React using multiple third-party frameworks in order to provide the best possible service

5.5.2.1 React

React is a Javascript library for building interactive web user interfaces. It optimizes the process of re-rendering the DOM of the loaded webpage, making dynamically re-rendering content on the webpage after its loaded. It also lets us split our code into many single-purpose components which speeds up the development process, and makes it more maintainable[105].

5.5.2.2 React-BootStrap

Well tested open source toolkit for making flexible interfaces. Which will fit most screen form factors out there. Also simplifies css styling, giving the page a more professional expression.

5.5.2.3 Papa Parse

Powerful tool for parsing and building CSV files with JavaScript. Allow us to provide state of the art support for all different forms of CSV files while taking care of edge cases. This also allows us to quickly export the anonymized data from the internal format so the user can download the result of the anonymization as a CSV file.

5.5.3 Functionality

All the functionality can be accessed through a single page application. Where the two main functionalities is *analyzation* and *anonymization*.

5.5.3.1 DataImport

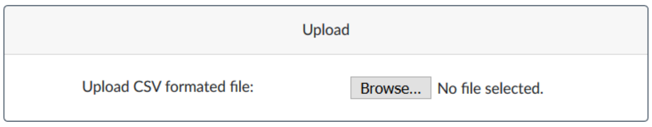
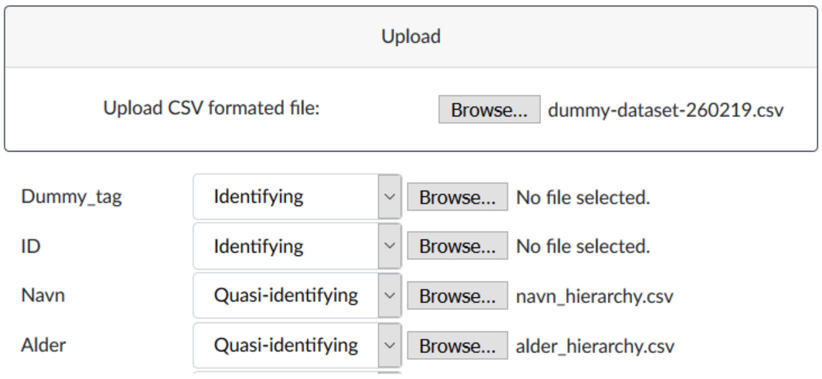


Figure 67 - Importing of dataset on WebARXaaS

The data import step is mandatory whenever the user wish to analyze or anonymize the data. To load data the user clicks the load button, and selects a *CSV* file. Once the *CSV* file is loaded, a automatically generated section will be displayed, showing each of the attribute headers from the csv file below the data import area.



Upload

Upload CSV formated file: Browse... dummy-dataset-260219.csv

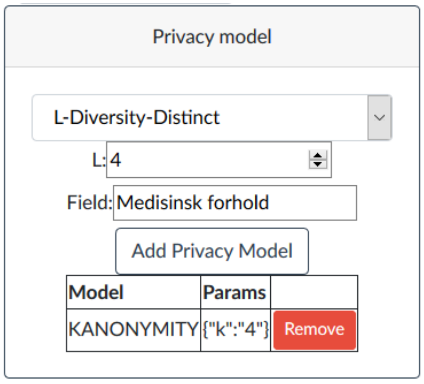
Dummy_tag	Identifying	Browse...	No file selected.
ID	Identifying	Browse...	No file selected.
Navn	Quasi-identifying	Browse...	navn_hierarchy.csv
Alder	Quasi-identifying	Browse...	alder_hierarchy.csv

Figure 68 - dataset headers generated

[105] Knowing reactJS - http://developer.ibm.com/recipes/tutorials/knowing-of-reactjs-with-advantages-limitations-challenges

5.5.3.2 Privacy model builder

When anonymizing data the user needs to add one or more privacy models describing which algorithm which will used to anonymizing the data. Each of the models has its own set of parameters which is used to configure how the anonymization process will behave.


Privacy model

L-Diversity-Distinct

L: 4

Field: Medisinsk forhold

Model	Params	
KANONYMITY	{"k":"4"}	Remove

Figure 69 - WebARXaaS privacy model section

5.5.3.3 Analyzation

The analyzation feature requires that the user already has loaded a *CSV* file, and set the correct *attribute types*. By pressing the *Analyze* button, the website will make a call to the backend service on /api/analyzation containing a JSON formatted payload, containing all the loaded data together with metadata. Once the response is received back from the service it will render multiple tables below containing metrics describing the analyzation quality.

Metric table	Content
Re Identification risk	Contains percentage likelihood on various *re-identification risks*
Risk interval	Gives metrics on how large portions of the entries in the data which is affected by each risk range

5.5.3.4 Anonymization

The anonymization feature requires that the user already loaded a *CSV* file, set the correct *attribute types*, and uploaded a *CSV* file containing a generalization hierarchy/transformation model for each of the *quasi-identifying* attributes.

By pressing the *Anonymize* button, the website will make a call to the backend service on /api/anonymization containing a JSON formatted payload, containing all the loaded data together with metadata. Once the response is received back from the server, it will display tables containing the following tables.

Metric table	Content
Anonymization data	The anonymized version of the dataset
Re Identification risk	Contains percentage likelihood on various *re-identification risks*
Risk interval	Gives metrics on how large portions of the entries in the data which is affected by each risk range
Process time	The time spent by the backend anonymizing the request in milliseconds.
privacy models	Containing metadata used by the service for each of the applied privacy models, and transformation model level used on the quasi-identifying attribute types.

After the data is anonymized the anonymized dataset can be downloaded as a csv file with the click of a button. This will make the web service package the as a csv file using ";" as a delimiter as it is the preferred style in most european countries.

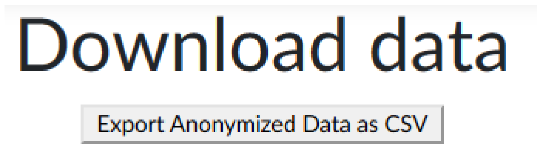


Figure 70 - WebARXaaS download button for anonymized dataset

5.5.4 Operations

This web application is built using *Node.js*. All the necessary dependencies for the project is specified inside the package.json file, in the root of the project directory.
Note that the ARXaaS service must be available via a server or running locally, in order to be utilizing the *analyzation* and *anonymization* functionality.

5.5.5 License

WebARXaaS is distributed under the MIT license. See the WebARXaaS License[106]

[106]WebARXaaS License - https://github.com/oslomet-arx-as-a-service/WebARXaaS/blob/master/LICENSE

5.6 Future development

NAV IT Data and Insight requested the solution to be maintainable, which presented several challenges in regards to future development. As ARX was, and is still receiving updates and new versions are to be released, the solution needs to retain compatibility with every new version of ARX. For instance, new versions of ARX have the potential to introduce new algorithms and strategies, which places a demand for continued developer support on every product.

Each product has challenges unique to them as well, and are presented on a product by product basis in the following subchapters.

5.6.1 ARXaaS

ARXaaS' API and wrapper define what ARX functionality is available to consumers of ARXaaS. When ARX is updated, ARXaaS will require maintenance in order to access and utilize the new functionality. Furthermore, ARXaaS' RESTful endpoints may need to be updated so that the new functionality can be offered to API consumers. Following fulfillment, the products consuming ARXaaS can be updated to call upon the new functionality and present the resulting output. The team has mitigated this by documenting the REST API thoroughly by writing unit tests on the ARX library to be run with the other ARXaaS tests to discover regressions in new ARX versions.

Throughout the project, the team learned that one of the largest challenges for efficient anonymization lied within the need for hierarchy diversity. Even though some datasets could be effectively anonymized with reused hierarchies, most datasets would present the need for at least a few unique tweaks for the proposed hierarchy to work. Unfortunately, customizing hierarchies is costly in both time and effort. To deal with this, the team is proposing to introduce automatic hierarchy generation powered by machine learning.

To promote more seamless integration with the client's IT systems, a proposed future challenge involves creating direct integrations with services at NAV. Specifically, introducing opportunities for products outside the project to benefit from calling the ARXaaS API.

5.6.2 PyARXaaS

Python has a rich ecosystem of data science tooling. Future development areas for PyARXaaS include building more integration to more of this ecosystem. Additionally the following areas could be a focus for future development.

- Functionality for comparison of machine learning model performance on datasets before and after anonymization
- Visualization of risk metrics contained in a risk profile
- Continuous improvement on the API design in collaboration with NAV IT data scientists

5.6.3 WebARXaaS

Although WebARXaaS at this stage is fully functional with all the core features, there is still room for future development on the platform.

The main functional feature which WebARXaaS is still lacking is the hierarchy builder, which would allow the user to construct new hierarchies inside the browser window. Once this feature is added, it will be possible to do the entire anonymization process inside of WebARXaaS without the usage of any external tools. It has also been requested more visualizations of the meta data displaying how well the data is anonymized. As well as the development of an anonymization threshold which would make it easier for the user to tell whether the data is sufficiently anonymized within the organizational standards. More usability related features could involve additional helping text boxes, guiding the use through the anonymization process.

- Hierarchy builder
- More visualizations
- Upload the image on Docker hub
- More user friendly help text which makes using best practices simpler for the user

5.7 Product Documentation Conclusion

Our project was started with the goal of providing data Scientists at NAV with state of the art anonymization functionality within their python work environment. In order to satisfy this need we have implemented a webservice with Java/Spring Boot together with a python package which seamlessly provides a programmatic interface for anonymizing tabular data. Furthermore we have also implemented a webclient in React.JS which was a stretch goal from our customer.

Overall both the team and the customer are happy the solution deliver beyond the initial product specification. NAV IT offered the entire team a work contracts for summer employment, and the entire team accepted. NAV IT is planning additional features they wish to be added to the core solution.

6 User manual

The purpose of this chapter is to give the reader a step-by-step guide on how to set up the solution, and how to benefit from the products. Moderate programming knowledge is advantageous in setting up the solution. It is recommended to read the product documentation and the de-identification chapter before utilizing the products.

6.1 ARXaaS

The API documentation can be found here:
https://oslomet-arx-as-a-service.github.io/ARXaaS/#analyze-controller

The javadoc can be found here: https://javadoc.io/doc/no.oslomet/arxaas/0.3.2-RELEASE

6.1.1 Run ARXaaS

This segment covers step-by-step guidance on how to run ARXaaS with both HTTP and HTTPS configuration. Docker offers the recommended environment for running ARXaaS. Docker is introduced and explained in the product documentation, under 5.2.3.2.2 Docker. Note that ARXaaS uses HTTP by default, reasoning can be found in the product documentation, under 5.4.7 Security.

In order to run ARXaaS as a Docker container, make sure

1. Docker is installed
2. Docker Desktop is running
3. Internet is available
4. If the previous three options are fulfilled, a command-line interface can be opened and the ARXaaS Docker image can be pulled from Docker Hub.

```
docker pull arxaas/aaas
```

5. Ready to move on to the following steps
    a. 6.1.1.1 HTTP Configuration
    b. 6.1.1.2 HTTPS Configuration.

6.1.1.1 HTTP Configuration

This is the default configuration for ARXaaS.
Before attempting to follow these steps, ensure that steps from 6.1.1 Run ARXaaS are fulfilled.

6.1.1.1.1 Run ARXaaS from Docker image

Run the ARXaaS Docker image as a Docker container

```
docker run arxaas/aaas
```

6.1.1.1.2 Run ARXaaS from .jar

Make sure that you have the latest version of java installed before going through these steps.

1. Download the latest version of the ARXaaS executable .jar from Maven Central.[107]
2. Open the command-line interpreter and run the jar file

```
java -jar <path to jar>
```

6.1.1.2 HTTPS Configuration

Before attempting to follow these steps, ensure that steps from 6.1.1 Run ARXaaS are fulfilled.

6.1.1.2.1 Recommended: Run ARXaaS with dynamic HTTPS parameters

This configuration applies when:

1. ARXaaS is being run as a Docker container
2. Keystore with SSL certificate(s) is passed from host machine to a mounted volume in the Docker container. For keystore generation see 6.1.1.2.2 Generating and correctly configuring a keystore for an ARXaaS project.
3. Spring Security HTTPS configuration is passed for locating said keystore upon running the Docker container

```
docker run -d -v <absolute path to keystore on host machine>:<relative
path from root directory in docker container to destination> -p 8080:8080
<docker image name> --server.ssl.key-store-type=<keystore type>
--server.ssl.key-store=<relative path to keystore file from root
directory in docker container> --server.ssl.key-store-password=<keystore
password> --server.ssl.key-alias=<name/alias of certificate in keystore>
```

NB: If your command application appears to stall after running this command, make sure to look for prompts from Docker concerning credential input

Working example:

```
docker run -d -v
C:/Users/vijo/git/ARXaaS/arxaas-keystore.p12:/app/arxaas-keystore.p12 -p
8080:8080 arxaas/aaas:latest --server.ssl.key-store-type=PKCS12
--server.ssl.key-store=/app/arxaas-keystore.p12
--server.ssl.key-store-password=password
--server.ssl.key-alias=arxaas-https
```

6.1.1.2.2 Generating and correctly configuring a keystore for an ARXaaS project

[107] Download ARXaaS executable jar from Maven Central - https://search.maven.org/search?q=a:arxaas

NB2: Option 2 and 3 require a keystore file containing a certificate inside the Spring project src/main/resources folder. Option 4 and 5 have the same requisites as option 2 and 3, plus compilation to jar / Docker image.

1. Create keystore and certificate. You will be prompted to set a password for the keystore

```
keytool -genkeypair -keystore <file name for new keystore, OPTIONAL:
preceed file name with absolute path to destination directory> -storetype
PKCS12 -alias <name for new certificate> -keyalg RSA -keysize 2048
-validity 360
```

2. OPTIONAL: Add more certificates to keystore

```
keytool -genkey -alias <name of new certificate> -keystore <path to
keystore> -storetype PKCS12 -keyalg RSA -storepass <keystore password>
-validity 730 -keysize 2048
```

NB3: For a non-dynamic HTTPS configuration, the keystore file should be placed in the Spring projects /src/main/resources folder. This is necessary for Spring to be able to find the certificate on the classpath with the settings that we have suggested

3. OPTIONAL**:** Verify that your certificate(s) are correctly stored inside the keystore

```
keytool -list -v -keystore <keystore file>
```

6.1.1.2.3 Compile and run ARXaaS with pre-defined, non-dynamic SSL configuration

Option 2 requires a keystore file inside the Spring projects src/main/resources folder required

1. Configuration should look like the following, change values after '='s to match user specific settings and uncomment the following settings from /src/main/resources/application.properties.

```
server.ssl.key-store=classpath:<full keystore file name>
server.ssl.key-store-type=<keystore type (PKCS12 recommended)>
server.ssl.key-store-password=<keystore password>
server.ssl.key-alias=<name/alias of certificate in keystore>
```

2. ARXaaS can now be run with HTTPS support enabled

- Docker:
    1. Build image

```
docker build -t <image name> <path to Dockerfile>
```

2. Run container

```
docker run -p 8080:8080 <image name>
```

- Jar:
  1. Compile the project

```
mvn clean install
```

  2. Run jar (after compiling it should be located inside the ARXaaS project's target folder)

```
java -jar <path to jar>
```

6.1.1.2.4 Running the server with dynamic HTTPS configuration for static HTTPS keystore/certificate(s).

The following options requires the keystore file to be inside the Spring projects src/main/resources folder).

Option 3: ...from Spring project

```
mvn spring-boot:run -Dserver.ssl.key-store-type=<keystore type>
-Dserver.ssl.key-store=classpath:<keystore file name>
-Dserver.ssl.key-store-password=<keystore password>
-Dserver.ssl.key-alias=<name/alias of certificate in keystore>
```

Option 4: ...from jar file

```
java -jar aaas-0.1.1-RELEASE.jar --server.ssl.key-store-type=<keystore
type> --server.ssl.key-store=classpath:<keystore file name>
--server.ssl.key-store-password=<keystore password>
--server.ssl.key-alias=<name/alias of certificate in keystore>
```

Option 5: ...from Docker image

```
docker run -p 8080:8080 -d arxaas/aaas
--server.ssl.key-store-type=<keystore type>
--server.ssl.key-store=classpath:<keystore file name>
--server.ssl.key-store-password=<keystore password>
--server.ssl.key-alias=<name/alias of certificate in keystore>
```

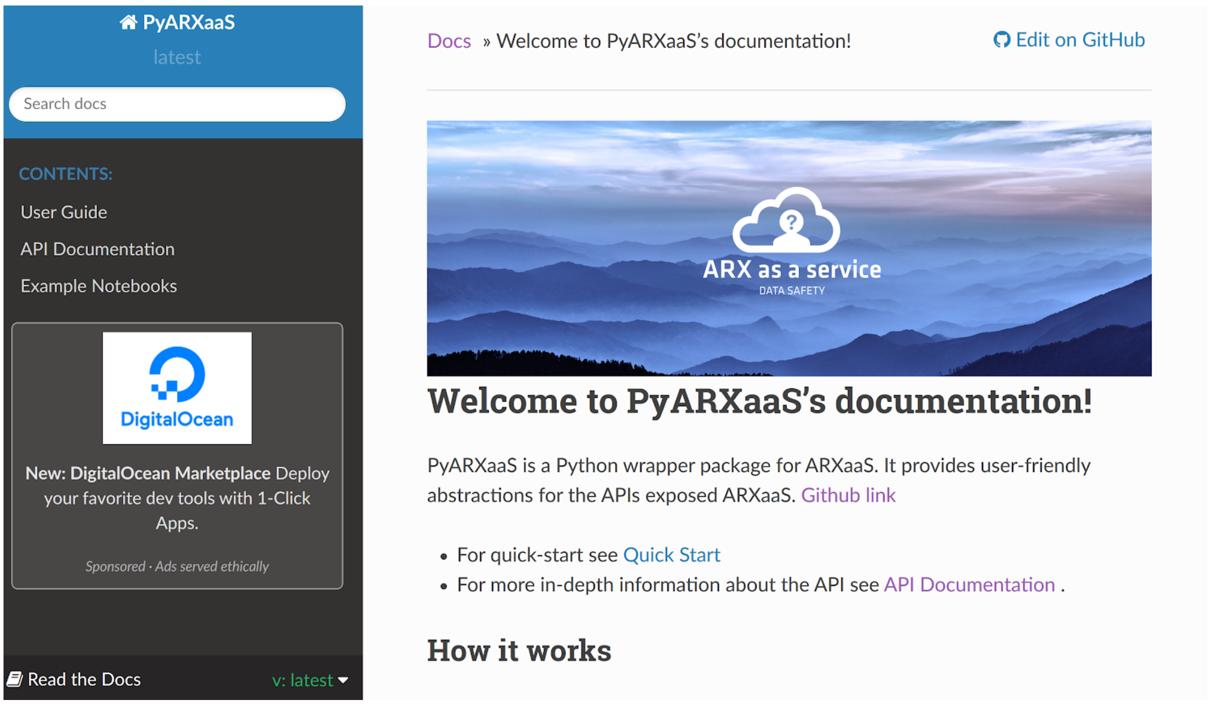
6.2 PyARXaaS client

6.2.1 Introduction

This document contains a step-by-step guide on how to start and use the Python Client. The python client was designed with the expectation that it would be used in Jupyter notebook, but it can be used on different IDE.

The team uses Sphinx to generate the PyARXaaS user guides, tutorials and API documentation. Published and hosted on the readthedocs.org[108]

- The API documentation can be found here: https://pyaaas.readthedocs.io/en/latest/



New: DigitalOcean Marketplace Deploy your favorite dev tools with 1-Click Apps.

Sponsored · Ads served ethically

Welcome to PyARXaaS's documentation!

PyARXaaS is a Python wrapper package for ARXaaS. It provides user-friendly abstractions for the APIs exposed ARXaaS. Github link

- For quick-start see Quick Start
- For more in-depth information about the API see API Documentation .

How it works

Figure 71 - Image of PyARXaaS documentation page using Sphinx

The user can access the different documentation by using the menu on the left side of the page.

6.2.2 Installing PyARXaaS Client

- PyAaas requires python 3.6 and up. Python download
- PyAaaS is available on PyPI PyPI link
- The source code can be found here: Github

[108] Read the Docs - https://readthedocs.org/

6.2.2.1 Pip install

Open the command-line interface and write:

```
pip install pyarxaas
```

6.2.2.2 Setup virtual environment

This is an optional step on how start a virtual environment to run PyARXaaS client on.

6.2.2.2.1 Mac/Linux

1. Create a new directory

```
mkdir pyarxaas-project
```

2. Change current directory to ‘pyaaas-project’

```
cd pyarxaas-project
```

3. Create a virtual environment

```
python3 -m venv c:\path\to\myenv
```

4. activate the virtual environment

```
source venv/bin/activate
```

6.2.2.2.2 Windows

1. Create a new directory

```
mkdir pyarxaas-project
```

2. Change current directory to ‘pyaaas-project’

```
cd .\pyarxaas-project
```

3. Create a virtual environment

```
python -m venv c:\path\to\myenv
```

4. activate the virtual environment

```
.\Scripts\activate
```

6.2.3 Quick start guide

This page gives a introduction in how to get started with PyAaaS
First, make sure that:

- PyARXaaS is installed
- PyARXaaS is up-to-date
- You have a tabular dataset to use
- You have a running ARXaaS instance to connect to.
    - Instructions on how to run ARXaaS can be found here: ARXaaS
- If you are going to anonymize a dataset, you need to have the required hierarchies. See anonymize section for more information

6.2.3.1 Analyze the risk of a dataset

Analyze the risk of a dataset using PyAaaS is very simple.

1. Begin by importing the dataset class and pandas which we are going to use to create a dataset.

```
from pyarxaas import dataset
import pandas as pd
```

Then we create a dataset from a local csv file.

Note
The dataset in this example contains the columns/fields id, name, gender.

```
dataframe = pd.read_csv("data.csv", sep=";")
# create dataset
dataset = dataset.from_pandas(dataframe)
```

The dataset class encapsulates the raw data, attribute types of the dataset fields and hierarchies.

2. Then we set the Attribute Type for the dataset fields.

```
 # import the attribute_type module
 from pyarxaas import AttributeType

# set attribute type
dataset.set_attribute_type(AttributeType.QUASIIDENTIFYING, 'name', 'gender')
dataset.set_attribute_type(AttributeType.IDENTIFYING, 'id')
```

3. To make a call to the ARXaaS instance we need to make a instance of the ARXaaS class.

   The ARXaaSConnector class needs a url to the ARXaaS instance. In this example we have ARXaaS running locally.

```
# import the aaas module
from pyarxaas import ARXaaS

# establishing a connection to the ARXaaS service using the URL
aaas = ARXaaS("http://localhost:8080")
```

4. After the ARXaaS object is created we can use it to call the ARXaaS instance to make a RiskProfile for our dataset.

```
# get the risk profile of the dataset
risk_profile = aaas.risk_profile(dataset)
```

   The RiskProfile contains two properties; re-identification risks and distributed risks. The two properties contains the different risks and the distribution of risks for the dataset.

```
# get risk metrics as a dictionary
re_indentifiation_risk = risk_profile.re_identification_risk
distribution_of_risk = risk_profile.distribution_of_risk

# get risk metrics as pandas.DataFrame
re_i_risk_df = risk_profile.distribution_of_risk_dataframe()
dist_risk_df = risk_profile.distribution_of_risk_dataframe()
```

6.2.3.2 Anonymize a dataset

Anonymizing a dataset using PyARXaaS.

1. Begin by importing the dataset class and pandas which we are going to use to create a dataset

```
from pyarxaas import dataset
import pandas as pd
```

2. Same as when in analyze we set the attribute type for the dataset fields:

```
 # import the attribute_type module
 from pyarxaas import AttributeType

# set attribute type
dataset.set_attributes(AttributeType.QUASIIDENTIFYING, 'name', 'gender')
dataset.set_attributes(AttributeType.IDENTIFYING, 'id')
```

3. In addition to setting attribute types we need to provide Transformation Models known as hierarchies for the dataset fields/columns with type **AttributeType.QUASIIDENTIFYING** Hierarchies can be added as pandas.DataFrame objects:

```
# importing the hierarchies from a local csv file. Specify the file path as the first
parameter
id_hierarchy = pd.read_csv("id_hierarchy.csv", header=None)
name_hierarchy = pd.read_csv("name_hierarchy.csv", header=None)

# setting the imported csv file. Specify the column name as the first parameter, and the
hierarchy as the second parameter
dataset.set_hierarchy('id', id_hierarchy)
dataset.set_hierarchy('name', name_hierarchy)
```

4. When anonymizing we need to supply a Privacy Model for ARXaaS to run on the dataset. You can read more about the models here ARX Privacy Models

```
# importing the privacy_models module
from pyarxaas.privacy_models import KAnonymity

# creating a privacy_models object
kanon = KAnonymity(4)
```

5. To make a call to the ARXaaS instance we need to make a instance of the AaaS class. The AaaS connector class needs a url to the ARXaaS instance. In this example we have ARXaaS running locally.

```
# import the aaas module
from pyarxaas import ARXaaS
```

```
# establishing a connection to the ARXaaS service using the URL
aaas = ARXaaS("http://localhost:8080")
```

6. After the ARXaaS object is created we can use it to call the ARXaaS instance. Back if the anonymization is successful we receive an AnonymizeResult

```
# specify the dataset as the first parameter, and privacy model list as the second
parameter
anonymize_result = aaas.anonymize(dataset, [kanon])
```

AnonymizeResult contains the new dataset, the RiskProfile for the new , the dataset, the anonymization status for the dataset and AnonymizeMetrics which contains metrics regarding the anonymization performed on the dataset.

```
# get the new dataset
anonymized_dataset = anonymize_result.dataset
anon_dataframe = anonymized_dataset.to_dataframe()

# get the risk profile for the new dataset
anon_risk_profile = anonymize_result.risk_profile

# get risk metrics as a dictionary
re_indentifiation_risk = anon_risk_profile.re_identification_risk
distribution_of_risk = anon_risk_profile.distribution_of_risk

# get risk metrics as pandas.DataFrame
re_i_risk_df = anon_risk_profile.distribution_of_risk_dataframe()
dist_risk_df = anon_risk_profile.distribution_of_risk_dataframe()

# get the anonymization metrics
anon_metrics = anonymize_result.anonymization_metrics
```

6.2.4 Connecting to and using ARXaaS

Calls to ARXaaS is made through the ARXaaS class. ARXaaS implements methods for the following functionality:

- Anonymize a Dataset object
- Analyze re-identification risk for a Dataset object
- Create generalization hierarchies (See chapter:6.2.7 Creating Hierarchies)

6.2.4.1 Creating an instance

When creating an instance of the ARXaaS class you need to pass a full url to the service running.

Example:

```
arxaas = ARXaaS("https://localhost:8080")
```

6.2.4.2 Risk Profile

Re-identification risk for prosecutor, journalist and marketer attack models can be obtained using the ARXaaS risk_profile method. The method takes a dataset object and returns a Risk Profile. See chapter 6.2.5 Using the dataset class, for more on the dataset class. More in depth information on re-identification risk ARX | risk analysis.

Example:

```
risk_profile = arxaas.risk_profile(dataset)
```

RiskProfile contains different properties containing analytics on the dataset re-identification risk. Most important is the re-identification risk property.

```
# create risk profile ...
risks = risk_profile.re_identification_risk
```

The property contains a mapping of risk => value. What is a acceptable risk depends entirely on the context of the dataset.

6.2.4.3 Anonymization

Anonymizing a dataset is as simple as passing a dataset containing the necessary hierarchies, a sequence of Privacy Model to use and optionally a suppression limit to the anonymize() method. The method, if successful returns a AnonymizeResult object containing the new dataset.

Example:

```
kanon = KAnonymity(2)
```

```
# in this example the dataset has a disease field
ldiv = LDiversityDistinct(2, "disease")
anonymize_result = arxaas.anonymize(dataset, [kanon, ldiv], 0.2)
anonymized_dataset = anonymize_result.dataset
```

6.2.4.4 Hierarchy Generation

Generalization hierarchies are a important part of anonymization. ARXaaS contains a hierarchy() method. It takes a configured Hierarchy Builders object and a dataset column represented as a common Python list. It returns a 2D list structure containing a new hierarchy.

Example making a redaction hierarchy:

```
redaction_builder = RedactionHierarchyBuilder()
zipcodes = [47677, 47602, 47678, 47905, 47909, 47906, 47605, 47673, 47607]
zipcode_hierarchy = arxaas.hierarchy(redaction_builder, zipcodes)
```

6.2.5 Using the dataset class

The dataset class represents a tabular dataset containing continuous or categorical attributes. Additionally each attribute has a Attribute Type describing the re-identification risk and sensitivity associated with the attribute.
In the case where a attribute is Quasi-identifying a hierarchy object can be added.

A Dataset contains:

- Tabular data
- *AttributeType*[109] for the data fields/attributes
- (optional) hierarchies for the quasi-identifying attributes

6.2.6 Construction

A *dataset* object can be made from a pandas, DataFrame or a python dict using the constructor class methods.

From Python dictionary

```
data_dict = {"id": [1,2,3], "name": ["Mike", "Max", "Larry"]}
new_dataset = dataset.from_dict(data_dict)
```

From pandas.DataFrame

```
dataframe = pd.read_csv("data.csv", sep=";")
new_dataset = dataset.from_pandas(dataframe)
```

[109] Python Docs for Attribute Type - https://pyaaas.readthedocs.io/en/latest/api/attribute_type.html#attribute-type

6.2.6.1 Dataset type conversion

The Dataset class possesses convenient methods for converting the contained tabular data to other data type, e.g. pandas dataframe.

To pandas.DataFrame Note: When you create a pandas DataFrame, from a Dataset[110] only the tabular data is included. The *Attribute Type* information and hierarchies are lost.

```
data_dict = {"id": [1,2,3], "name": ["Mike", "Max", "Larry"]}
new_dataset = dataset.from_dict(data_dict)
dataframe = new_dataset.to_dataframe()
#    id   name
#0   1   Mike
#1   2    Max
#2   3  Larry
```

6.2.6.2 Mutation

6.2.6.2.1 Attribute type

The default *Attribute Type* for attributes in a *dataset* is AttributeType.QUASIIDENTIFYING. The default is set to *quasi-identifying* so that new users will get an error that is easily understandable. You can change the type of a attribute with the set_attribute_type() method.:

```
from pyarxaas import AttributeType
new_dataset.set_attribute_type(AttributeType.IDENTIFYING, "id")
```

Above we have changed the *Attribute Type* of the *dataset* to Attribute Type.IDENTIFYING. This signals that the id attribute is a directly identifying attribute in this *dataset*. The id will be treated as such by ARXaaS if anonymization is applied to the dataset.
Read more about the different Attribute types here: Attribute Type

It is possible to pass *n* attributes following the Attribute Type parameter to set the attribute type to all the attribute.

```
# Here id and name are marked as insensitive attributes
new_dataset.set_attribute_type(AttributeType.INSENSITIVE, "id", "name")
```

6.2.6.2.2 Hierarchies

Hierarchy also referred to as *generalization hierarchies* represented either as pandas.DataFrames or a regular Python list, are the strategies ARXaaS will use when attempting to anonymize the dataset. Read more about them in chapter 6.2.7 Creating Hierarchies.

[110] Python Docs for dataset - https://pyaaas.readthedocs.io/en/latest/api/dataset.html#dataset

Setting a hierarchy on a dataset attribute

```
id_hierarchy = [["1", "*"], ["2", "*"], ["3", "*"]]
dataset.set_hierarchy("id", id_hierarchy)
```

You can also set several hierarchies in one call with the .set_hierarchies(hierarchies) method.

```
id_hierarchy = [["1", "*"], ["2", "*"], ["3", "*"]]
job_hierarchy = [["plumber", "manual-labour", "*"],
                 ["hairdresser", "service-industry", "*"]]
hierarchies = {"id": id_hierarchy, "job": job_hierarchy}
dataset.set_hierarchies(hierarchies)
```

6.2.7 Creating Hierarchies

After creating a dataset from some data source, you can set the hierarchies ARXaaS will use when attempting to anonymize the dataset. ARXaaS currently only support value generalization hierarchies. Read more about different transformation models in ARX documentation.

6.2.7.1 Hierarchy Building

ARXaaS offers an endpoint to use the ARX library hierarchy generation functionality. PyARXaaS implements abstractions to make this process as easy and intuitive as possible. Hierarchy generation that ARX offers falls into four different categories:

- Redaction based hierarchies
- Interval based hierarchies
- Order based hierarchies
- Date based hierarchies

ARXaaS and PyARXaaS currently only support Redaction, Interval and Order based hierarchy generation. In PyARXaaS all the hierarchy builders are importable from the **pyarxaas.hierarchy** package

6.2.7.2 Redaction based hierarchies

Redaction based hierarchies are hierarchies suited best for categorical but numeric values. Attributes such as zipcodes are a prime candidate. The hierarchy strategy is to delete one number at a time from the attribute column until the privacy model criteria is meet. The hierarchy builder can be configured to start deleting from either direction, but will default to RIGHT_TO_LEFT. Redaction based hierarchies are the hierarchies with the least effort to create.

Example:
In this example we will use a list of zip codes representing a column from a hypothetical dataset. The list could be generated from any source. Hierarchy building works on list of strings or numbers.

```
zipcodes = [47677, 47602, 47678, 47905, 47909, 47906, 47605, 47673, 47607]
```

We will then import the redaction hierarchy builder class

```
from pyarxaas.hierarchy import RedactionHierarchyBuilder
```

The RedactionHierarchyBuilder class is a simple class and all configuration is optional. When instantiating the class the user can pass in parameters to configure how the resulting hierarchy should be built. See RedactionHierarchyBuilder for more on the parameters.

6.2.7.2.1 Creating a simple redaction hierarchy

```
# Create builder
redaction_builder = RedactionHierarchyBuilder()
```

The builder defines a template to build the resulting hierarchy. Now that there is a list of a dataset field, and a builder to create a hierarchy. The client can connect to ARXaaS to make the hierarchy.

```
from pyarxaas import ARXaaS
# establishing a connection to the ARXaaS service using a url, in this case
ARXaaS is running locally on port 8080
arxaas = ARXaaS("http://localhost:8080")
```

With the connection to ARXaaS established we can create the hierarchy.

```
# pass builder and column to arxaas
redaction_hierarchy = arxaas.hierarchy(redaction_based, zipcodes)
```

The resulting hierarchy looks like this:

```
[['47677', '4767*', '476**', '47***', '4****', '*****'],
['47602', '4760*', '476**', '47***', '4****', '*****'],
['47678', '4767*', '476**', '47***', '4****', '*****'],
['47905', '4790*', '476**', '47***', '4****', '*****'],
['47909', '4790*', '476**', '47***', '4****', '*****'],
['47906', '4790*', '476**', '47***', '4****', '*****'],
['47605', '4760*', '476**', '47***', '4****', '*****'],
['47673', '4767*', '476**', '47***', '4****', '*****'],
['47607', '4760*', '476**', '47***', '4****', '*****']]
```

6.2.7.3 Interval based hierarchies

Interval based hierarchies are well suited for continuous numeric values. Attributes such as age, income or credit score are typically generalized with a interval based hierarchy. The Interval based hierarchy builder requires the user to specify intervals in which to generalize values in the attribute. Optionally these intervals can be labeled. In addition intervals can be grouped upwards using levels and groups to create a deeper hierarchy.

Example In this example we will use a list of ages representing a column from a hypothetical dataset.

```
ages = [29, 22, 27, 43, 52, 47, 30, 36, 32]
```

1. We import the IntervalHierarchyBuilder class from the hierarchy package.

```
from pyarxaas.hierarchy import IntervalHierarchyBuilder
```

2. Then we instantiate the builder. IntervalHierarchyBuilder takes no constructor arguments.

```
interval_based = IntervalHierarchyBuilder()
```

3. Add intervals to the builder. The intervals must be continuous(without gaps)

```
interval_based.add_interval(0,18, "child")
interval_based.add_interval(18,30, "young-adult")
interval_based.add_interval(30,60, "adult")
interval_based.add_interval(60,120, "old")
```

Optionally we add groupings. Groupings are added to a specific level and are order based according to the interval order.

```
interval_based.level(0)\
    .add_group(2, "young")\
    .add_group(2, "adult")
```

4. Call the ARXaaS service to create the hierarchy

```
interval_hierarchy = arxaas.hierarchy(interval_based, ages)
```

The hierarchy looks like this:

```
[['29', 'young-adult', 'young', '*'],
 ['22', 'young-adult', 'young', '*'],
 ['27', 'young-adult', 'young', '*'],
 ['43', 'adult', 'adult', '*'],
 ['52', 'adult', 'adult', '*'],
 ['47', 'adult', 'adult', '*'],
 ['30', 'adult', 'adult', '*'],
 ['36', 'adult', 'adult', '*'],
 ['32', 'adult', 'adult', '*']]
```

6.2.7.4 Order based hierarchy

OrderHierarchyBuilder are suited for categorical attributes. Attributes such as country, education level and employment status.

Order based hierarchies are built using groupings with optional labeling. This means that grouping is completed on the list of values as it is. This means the list has to be sorted according to ordering before a hierarchy can be made. Order based hierarchies are usually very reusable depending on the domain.

In this example we will use a column of diseases:

```
diseases = ['bronchitis',
            'flu',
            'gastric ulcer',
            'gastritis',
            'pneumonia',
            'stomach cancer']
```

In this case we will sort the diseases according to the disease location; *lung-disease* or *stomach-disease*. But this sorting can be as complex as the user wants.

```
unique_diseases[2], unique_diseases[4] = unique_diseases[4], unique_diseases[2]
unique_diseases

#['bronchitis',
# 'flu',
# 'pneumonia',
# 'gastritis',
# 'gastric ulcer',
# 'stomach cancer']
```

1. Import OrderHierarchyBuilder

```
from pyarxaas.hierarchy import OrderHierarchyBuilder
```

2. Create instance to use.

```
order_based = OrderHierarchyBuilder()
```

Group the values. Note that the groups are applied to the values as they are ordered in the list. Adding labels are optional, if labels are not set the resulting field will be a concatenation of the values included in the group.

```
order_based.level(0)\
    .add_group(3, "lung-related")\
    .add_group(3, "stomach-related")
```

3. Call the ARXaaS service to create the hierarchy

```
order_hierarchy = arxaas.hierarchy(order_based, diseases)
```

The resulting hierarchy looks like this:

```
[['bronchitis', 'lung-related', '*'],
 ['flu', 'lung-related', '*'],
 ['pneumonia', 'lung-related', '*'],
 ['gastritis', 'stomach-related', '*'],
 ['gastric ulcer', 'stomach-related', '*'],
 ['stomach cancer', 'stomach-related', '*']]
```

6.3 WebARXaaS client

The WebARXaaS client files can be found and downloaded here:
https://github.com/oslomet-arx-as-a-service/WebARXaaS

6.3.1 Starting the application

In order to start the application locally you must have a local installation of *NodeJS* newer than 10.15 and the packet manager *npm* installed.

1. Download the WebARXaaS client
2. Make sure the current directory of your terminal is the root directory of *WebARXaaS*.
3. Run `npm install` in your terminal in order to download all the dependencies specified in package.json.
4. Run `npm start` in your terminal. This will start up an instance of the application running locally on port 3000.
5. You can now access the website locally by navigating to http://localhost:3000/ with your web browser.

6.3.2 Deploying to production

This application is built using the node *create-react-app* package. In order to generate files ready to be deployed to production you must first build the application.

1. Download the WebARXaaS client
2. Run the command `npm install` inside the project directory, in order to ensure you got the necessary dependencies downloaded locally.

3. Run the command `npm build`, this command is an alias for `react-scripts build` and will generate production ready files into the */build* directory.
4. Copy the content of the */build* into the public directory of a web server. To do this you can use the *nginx* docker image by using the docker image in the root directory of the project.
5. Run the command `docker build --tag=webarxaas .` from the root directory of the application, to make docker start the building of the docker image shown below.
6. For starting the built docker image which was built on the previous step, run the command `docker run -p 80:8080 webarxaas`. This will start the docker container running the application, making the server start running on port 80.
7. Use your browser to navigate to the website at http://localhost:80 and check that the website is up and running

```
FROM nginx
COPY build /usr/share/nginx/html
COPY nginx.conf /etc/nginx/conf.d/default.conf
```

6.3.3 Configuration

By default the application connects to the URL defined inside web-aaas\src\App.js. The url should be changed if your organization is running your own ARXaaS service. It is also possible to define the service url manually on the website, but this is mainly intended for testing purposes as the entered URL currently does not get saved.

```
const [endpoint, setEndpoint] = useState('http://35.228.21.181:8080')
```

6.3.4 Analyzing

Steps on how to analyze a dataset against re-identification risk.

1. Import the dataset by clicking on the “browse” button

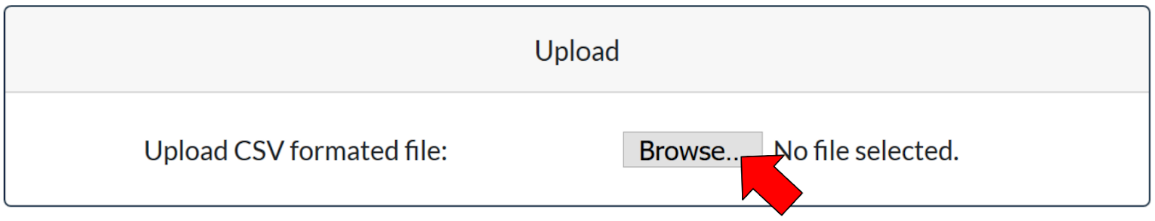


Figure 72 - Importing of dataset to analyze

2. Select the csv file containing a dataset to be analyzed.

3. After selecting a dataset, an auto generated section containing the dataset headers will appear below.

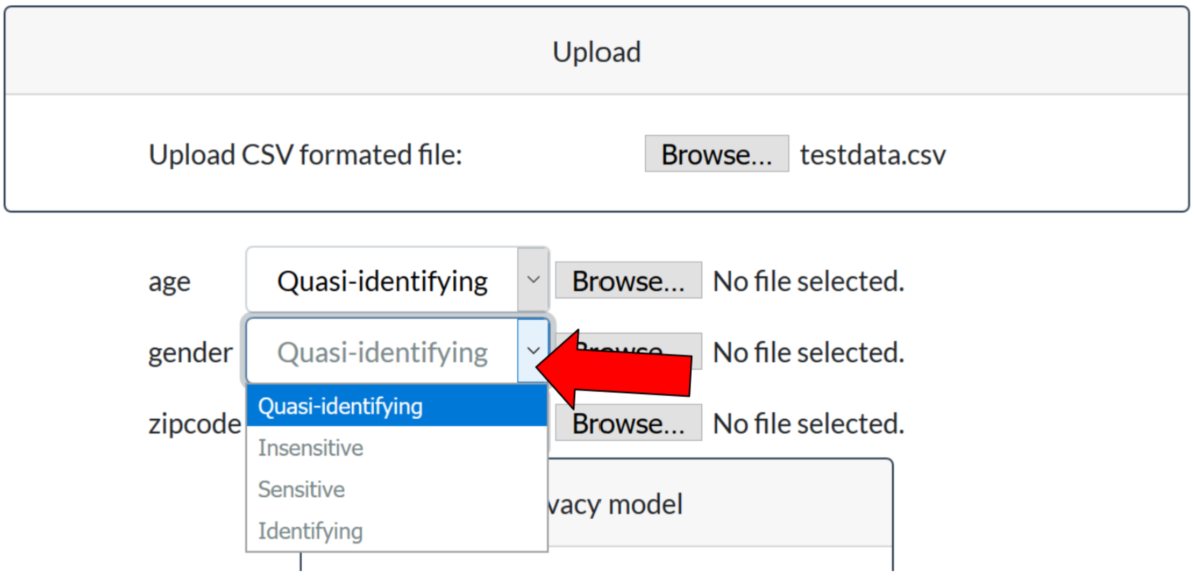


Figure 73 - Setting attribute type of dataset to analyze

a. Select the attribute type of each dataset headers by clicking on the drop down menu(where the red arrow is pointing).

NB When analyzing a dataset, there is no need to import hierarchies.

4. After selecting the attribute types for the dataset headers. Click the “analyze” button.

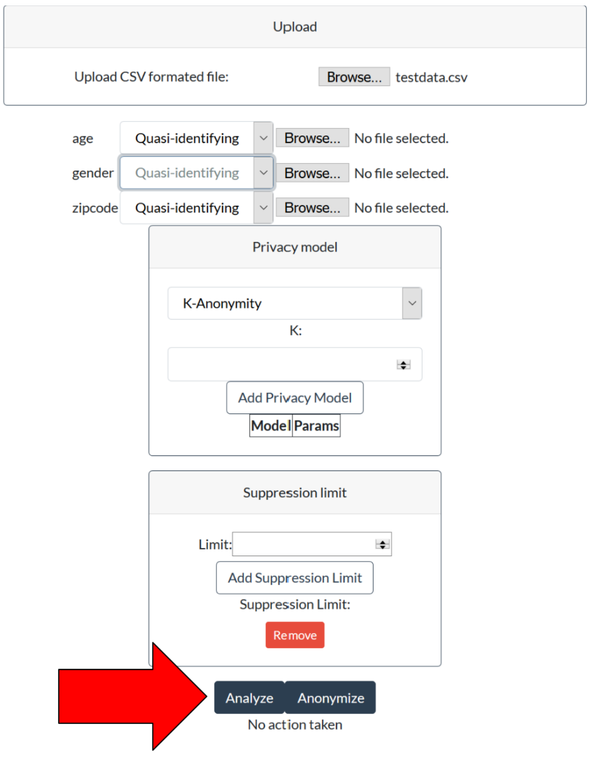
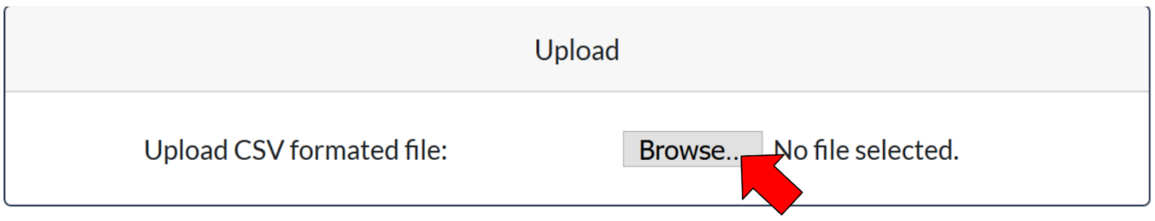


Figure 74 - Analyze the dataset

6.3.5 Anonymizing

Steps on how to anonymize a dataset against re-identification risk.

1. Import the dataset by clicking on the “browse” button(where the red arrow is pointing).

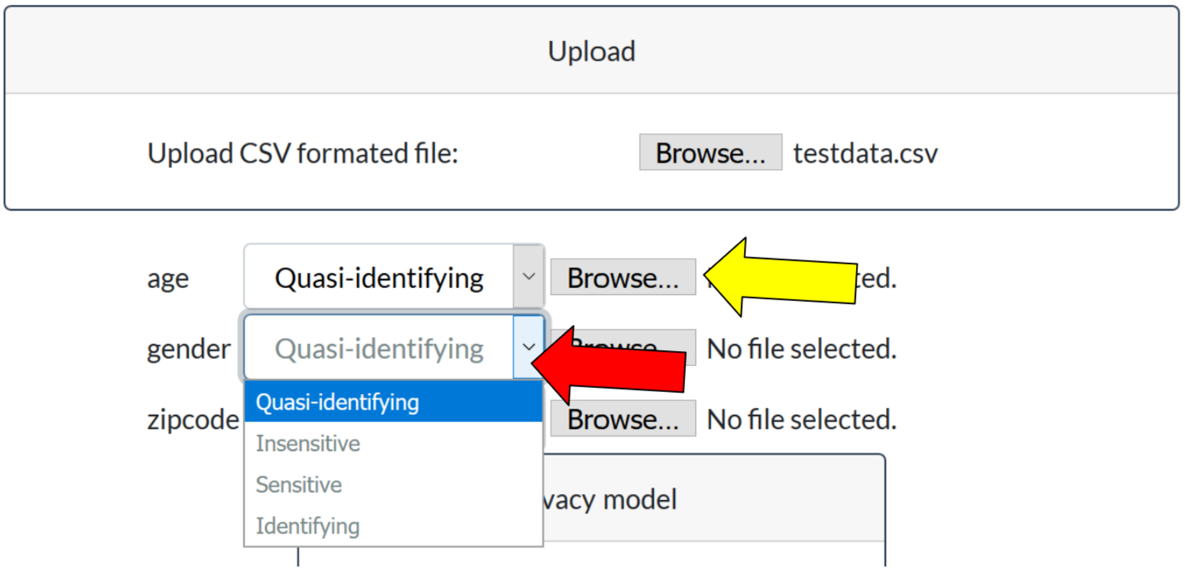


Upload

Upload CSV formated file: Browse... No file selected.

Figure 75 - Importing of dataset to anonymize

2. Select the csv file containing a dataset to be anonymized.
3. After selecting a dataset, an auto generated section containing the dataset headers will appear below.



Upload

Upload CSV formated file: Browse... testdata.csv

age Quasi-identifying Browse... ed.

gender Quasi-identifying No file selected.

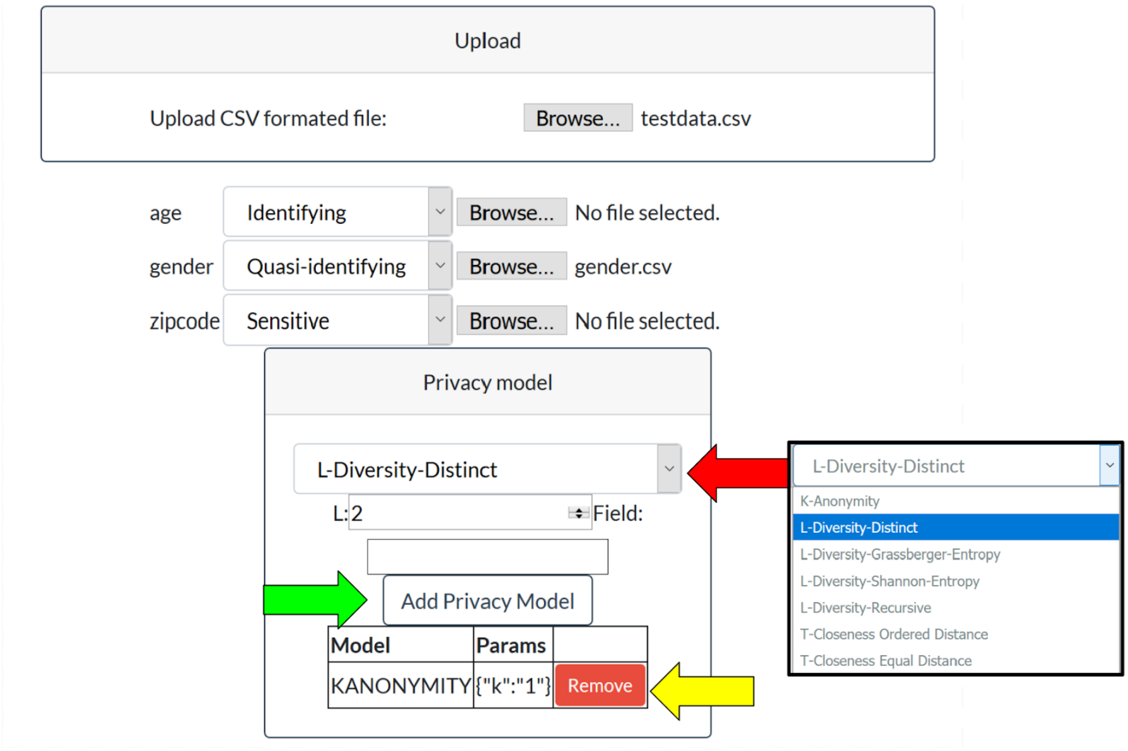
zipcode Browse... No file selected.

Figure 76 - Setting attribute type and importing transformation model

   a. Import transformation model csv file by clicking on where the yellow arrow is pointing.
   b. Select the attribute type of each dataset headers by clicking on where the red arrow is pointing.

NB When anonymizing a dataset, a quasi-identifying attribute type needs a hierarchy/transformation model.

4. Select a privacy model to anonymize the dataset


Upload

Upload CSV formated file: Browse... testdata.csv

age	Identifying	Browse...	No file selected.
gender	Quasi-identifying	Browse...	gender.csv
zipcode	Sensitive	Browse...	No file selected.

Privacy model

L-Diversity-Distinct

L: 2 Field:

Add Privacy Model

Model	Params	
KANONYMITY	{"k":"1"}	Remove

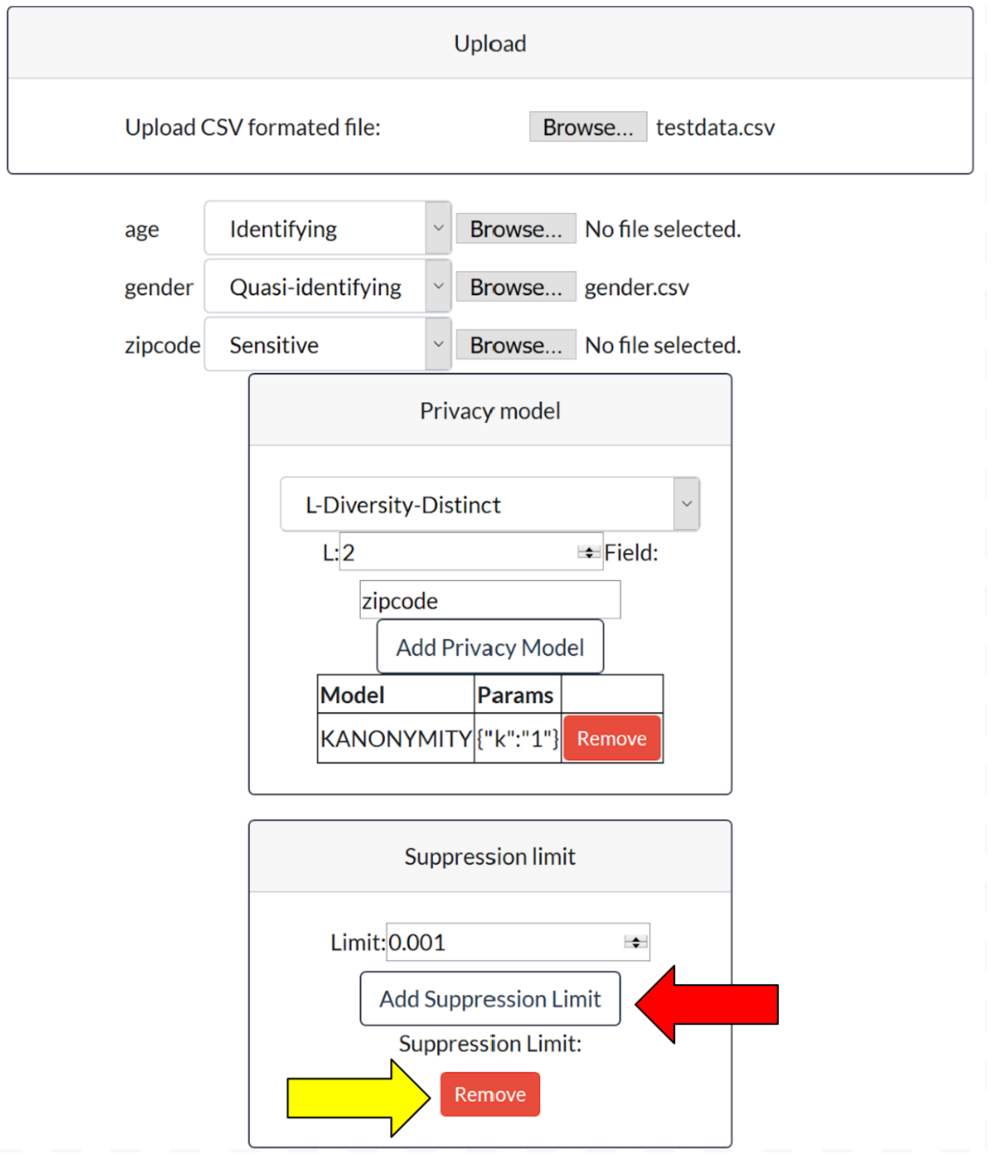
L-Diversity-Distinct
K-Anonymity
L-Diversity-Distinct
L-Diversity-Grassberger-Entropy
L-Diversity-Shannon-Entropy
L-Diversity-Recursive
T-Closeness Ordered Distance
T-Closeness Equal Distance

Figure 77 - Setting the privacy model to anonymize the dataset

a. By clicking where the red arrow is pointing a drop down menu will show the available privacy models.
b. Select a privacy model to anonymize the dataset.
c. After selecting a privacy model, type the desired privacy model configuration in the input fields.
    - Read more about the Privacy model: here

d. click the “add privacy model” button(where the green arrow is pointing) to apply the setting.
e. Added privacy model can be removed by clicking the “remove” button(where the yellow arrow is pointing).

5. Optionally a suppression limit can be added to anonymize the dataset.



Upload

Upload CSV formated file: Browse... testdata.csv

age	Identifying	Browse...	No file selected.
gender	Quasi-identifying	Browse...	gender.csv
zipcode	Sensitive	Browse...	No file selected.

Privacy model

L-Diversity-Distinct

L: 2 Field: zipcode

Add Privacy Model

Model	Params	
KANONYMITY	{"k":"1"}	Remove

Suppression limit

Limit: 0.001

Add Suppression Limit

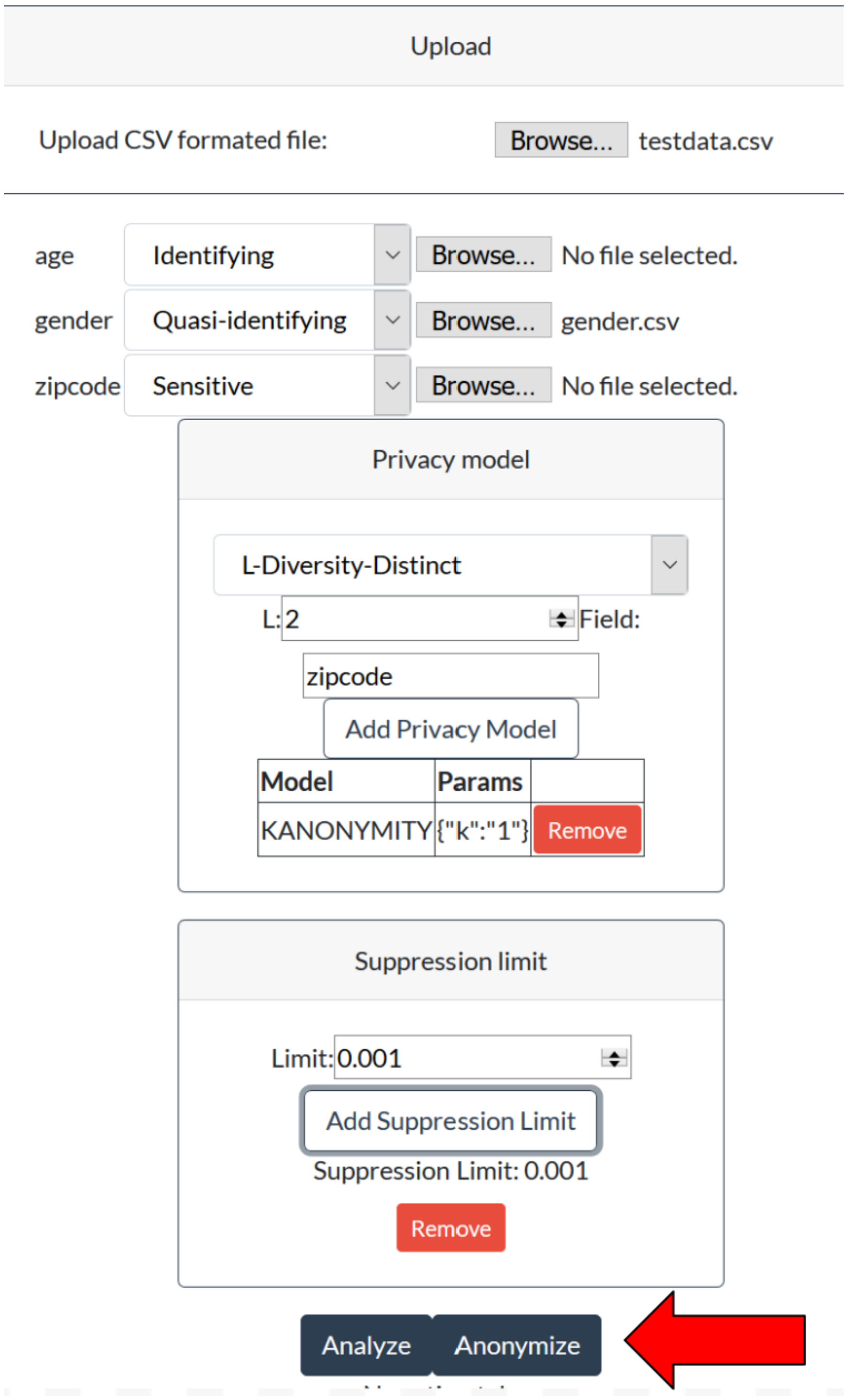
Suppression Limit:

Remove

Figure 78 - Setting the suppression limit for the dataset

a. Specify a suppression limit to be used in the limit input field, and click on the "add suppression limit" button(where the red arrow is pointing) to apply the setting.
b. A suppression limit can be removed by clicking on the "remove" button(where the yellow arrow is pointing).

6. Click the anonymize button to start anonymizing the dataset against re-identification risk.


Upload

Upload CSV formated file: Browse... testdata.csv

age	Identifying	Browse...	No file selected.
gender	Quasi-identifying	Browse...	gender.csv
zipcode	Sensitive	Browse...	No file selected.

Privacy model

L-Diversity-Distinct

L: 2 Field:

zipcode

Add Privacy Model

Model	Params	
KANONYMITY	{"k":"1"}	Remove

Suppression limit

Limit: 0.001

Add Suppression Limit

Suppression Limit: 0.001

Remove

Figure 79 - Anonymizing the dataset

6.3.6 Privacy model input field description

Descriptions of the different input fields for each privacy model.

6.3.6.1 K-Anonymity

Anonymizing with K-anonymity.

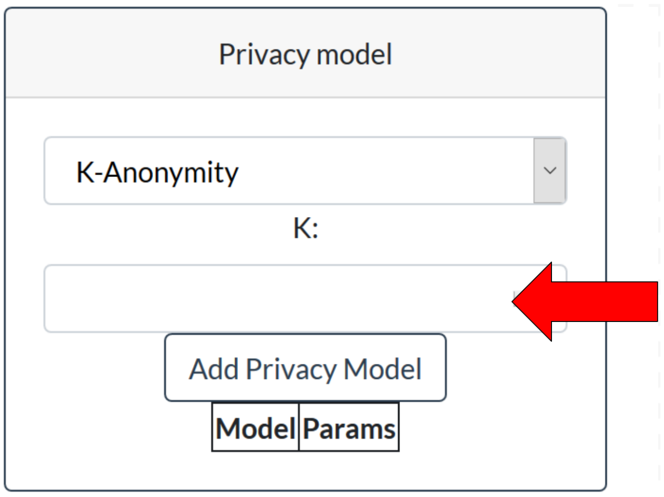


Figure 80 - K- Anonymity input field

- K = K-anonymity generalization value when anonymizing the dataset.
  - K must have a value of 2 or higher to take effect.

Read more about K-Anonymity on chapter 2.5.1 K-Anonymity.

6.3.6.2 L-Diversity

Anonymizing with L-Diversity.

Read more about L-Diversity on chapter 2.5.2 L-Diversity.

6.3.6.2.1 Using non-recursive variants of L-diversity

When using Distinct, Grass-Berger-entropy, and Shannon-entropy only two input fields are available.

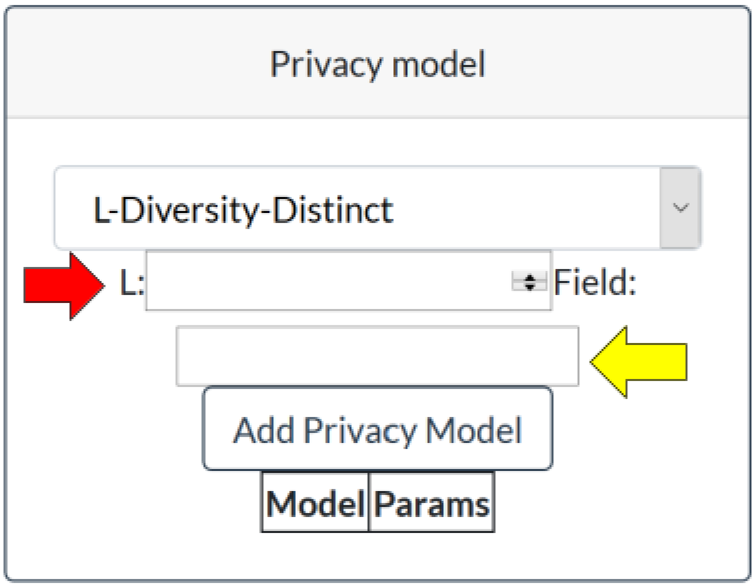


Figure 81 - Input fields for Distinct, Grass-Berger- and Shannon-Entropy

- L = Value when anonymizing the dataset, based on a column or dataset field that has a sensitive attribute.
    - L must have a value of 2 or higher to take effect. (Red arrow)

- Field = Column or dataset field that has a sensitive attribute type. (Yellow arrow)

6.3.6.2.2 Using Recursive variant of L-diversity

When using the Recursive variant of L-diversity a three input fields are available.

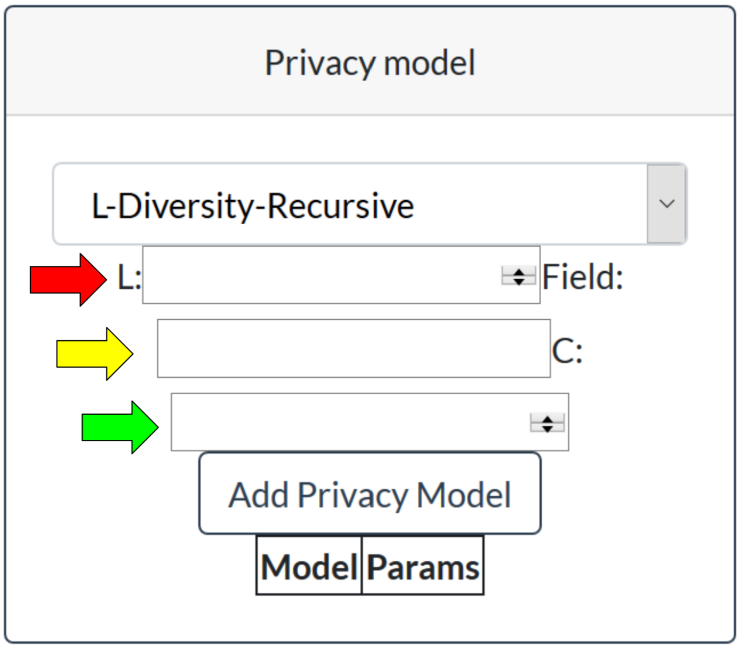


Figure 82 - Input fields for Recursive L-Diversity

- L = Value of L to anonymize the dataset based on a column or dataset field that has a sensitive attribute.
    - L must have a value of 2 or higher to take effect. (Red arrow)
- Field = Column or dataset field that has a sensitive attribute type. (Yellow arrow)
- C = Value of C to anonymize the dataset based on a column or dataset field that has a sensitive attribute.
    - C must have a value of 0.00001 or higher to take effect. (Green arrow)

6.3.6.3 T-Closeness

Anonymizing with T-closeness. The Ordered and Equal Distance variant of T-closeness takes two input fields

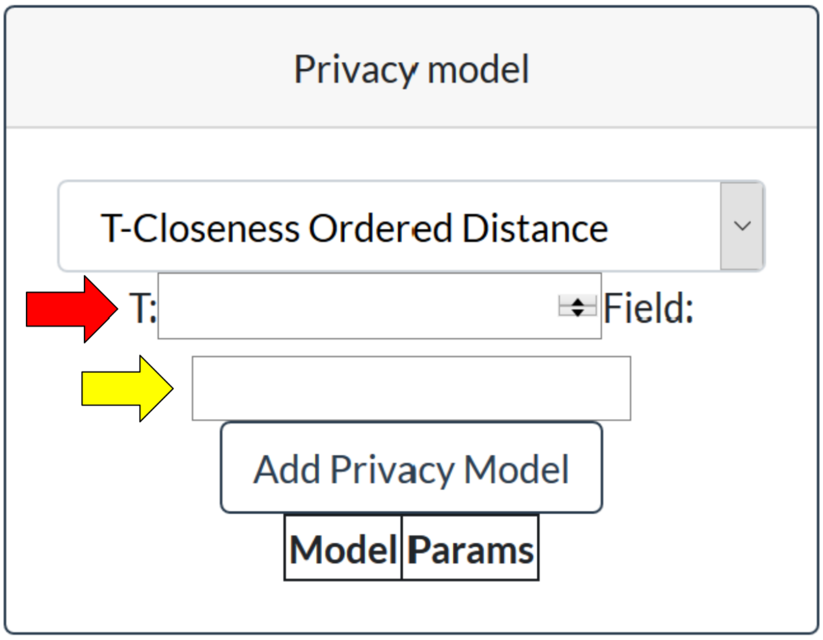


Figure 83 - Input fields for Order and Equal Distance T-Closeness

- T = Value of T to anonymize the dataset based on a column or dataset field that has a sensitive attribute.
    - T must have a value between 0.000001 to 1.0
- Field = Column or dataset field that has a sensitive attribute type.

Read more about T-Closeness on chapter [2.5.3 T-Closeness](#).

7 Appendix

7.1 Terminology

Abstract base class - are classes that contain one or more abstract methods. An abstract method is a method that is declared, but contains no implementation. Abstract classes may not be instantiated, and require subclasses to provide implementations for the abstract methods.
Agile - Agile teams have a specific approach to software development, especially in regards to self-organizing and cross-functional teams and their customer(s)/end user(s).
Attribute - column of values in a dataset representing a set of attributes.

Attribute Type - the disclosure risks from which a dataset is to be protected.

ARX - is a comprehensive *open source* software for anonymizing sensitive personal data.
Certificate Authority (CA) - A Certificate Authority is an entity that issues digital certificates.

Code Climate - engineering process insights and automated code review for GitHub and GitHub Enterprise help you ship better software, faster.

Container orchestration platform - cloud orchestration involves the end-to-end automation and coordination of multiple processes to deliver a desired service to its clients. Typically used to provision, deploy or start servers; acquire and assign storage capacity; manage networking; create VMs; and gain access to specific software on cloud services.

Continuous Integration (CI) - the practice of merging all developer working copies to a shared mainline several times a day.

Continuous Delivery (CD) - Continuous delivery is a series of practices designed to ensure that code can be rapidly and safely deployed to production by delivering every change to a production-like environment and ensuring business applications and services function as expected through rigorous automated testing.
Control-flow graph (CFG) - In computer science, CFG is a graphical display of how many paths that can possibly be traversed during a program's runtime.

CRUD application - The acronym CRUD refers to all of the major functions that are implemented in relational database applications. Create, read, update, and delete.

Data-driven - The progress of a data-driven activity is influenced by data, rather than intuition or personal experience.

Data Package - Data Package is a simple container format used to describe and package a collection of data.

Data pipeline - A data pipeline automates the moving and transformation of data. In the case of this project, it is a controlled sequence of actions, that is performed on data, triggered by a specific event.

Domain Driven Design - is an approach to developing software for complex needs by deeply connecting the implementation to an evolving model of the core business concepts. It aims to ease the creation of complex applications by connecting the related pieces of the software into an ever-evolving model.

Digital Transformation - the integration of digital technology into all areas of a business resulting in fundamental changes to how businesses operate and how they deliver value to customers.
Equivalence class - Records in a dataset that have the same values on the *quasi-identifiers*.
Factory method - factory method pattern is a creational pattern that uses factory methods to deal with the problem of creating objects without having to specify the exact class of the object that will be created.

GitHub - a git repository hosting platform. Commonly used for *open source* software.
HTTP - Hypertext Transfer Protocol
HTTPS - Hypertext Transfer Protocol Secure

Jacoco - is a free Java code coverage library distributed under the Eclipse Public License.

Java - is a high-power, stable and highly trusted programing language. Commonly used in backend application with a requirement for stability.

Javascript - an object-oriented computer programming language commonly used to create interactive effects within web browsers.

JSON - is an open-standard file format that uses human-readable text to transmit data objects consisting of attribute–value pairs and array data types.

Jupyter Notebook - Project Jupyter exists to develop open source software, open-standards, and services for interactive computing across dozens of programming languages.

Kubernetes - is an *open source* container orchestration system for automating application deployment, scaling, and management.
K-anonymity, L-diversity, T-closeness - *Privacy Models* for protecting privacy. Explanations sourced in the Reference List.
Masking - The process of removing a variable or replacing it with pseudonymous or encrypted information.

MIT Licence - The MIT License is a permissive free software license originating at the Massachusetts Institute of Technology.

Mocking - (testing) Mocking is make a replica or imitation of something. In testing of programs mocking is creating objects that simulate the behavior of real objects. The goal is to use mocks to help test parts of a program in isolation.

NAIS - is an application platform built to increase development speed by providing our developers at NAV with the best possible tools to develop and run their applications.

open source - is a term denoting that a product includes permission to use its source code, design documents, or content.

Pandas - is an *open source*, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.

Pandas Dataframe - A one-dimensional labeled array capable of holding any data type with axis labels or index.
Personal Data - Personal data is any information which is related to an identified or identifiable natural person.
PII - PII or Personally Identifiable Information is any data that can be used to clearly identify an individual.

Privacy Model - Privacy Models are specific algorithms applied to a dataset to protect the dataset from an identification risk vector.

Product Backlog - The list of items, ordered by priority (by the *Product Owner),* that need to be done within the project.

Product Owner - The individual that is responsible for maintaining the *Product Backlog* and maximizing the value of the solution.

Project Stakeholders - Individual, group or corporation that may be affected by the outcome of this project. In this case NAV IT, OsloMET, and the development team.

Python - Python is a powerful, high-level language. Used in everything from web apps to data-science/machine learning.

Pull Request - Pull requests let you tell others about changes you've pushed to a GitHub repository. Once a pull request is sent, involved parties can review the changes, discuss potential modifications, and even push follow-up commits if necessary.

Re-identification risk - The risk of an individual being identified from a row of data in a dataset.

Test coverage - Metric on the percentage of code covered by tests.

Tabular dataset - a data consisting of or presented in columns or tables.

Transformation Model - Transformation Models are strategies that describes how a specific column in the dataset should lose data as the anonymization tries to achieve the required anonymization level specified by the *Privacy Model*.

Travis CI - is a hosted, distributed continuous integration service used to build and test software projects hosted at GitHub.

Scrum - is an *agile* framework for managing work, with an emphasis on software development. It is designed for teams of three to nine members, who break their work into actions that can be completed within timeboxed iterations, called "sprints", no longer than one month and most commonly two weeks, then track progress and re-plan in 15-minute *stand-up meetings*, called daily scrums.

Scrum Master - The Scrum Master is responsible for promoting and supporting Scrum.

Spring - is an *open source*, high-power, Java application framework for business applications.

Stand-up meetings - Attendees of a stand-up meeting (or simply "stand-up"), participate in a meeting while standing. The intention is to keep the meetings short because of the discomfort of standing.

Unique identifier - Unique identifiers are pieces of information that provide sufficient data as to independently identifying a person.

Quasi identifier - Quasi identifiers are pieces of information that can be combined with other quasi-identifiers to create a *unique identifier*.

7.2 Product Specification

Product Specification

Anonymization as a Service

08.02.2019

Group 8

Members	Student Number	Email
Jeremiah Augie Salita Uy	s181369	jeremiah.uy@outlook.com
Lord André Groseth	s181365	andregroseth@outlook.com
Sondre Halvorsen	s305349	sondre.hal@gmail.com
Julian Sagen	s315584	julian.sagen@gmail.com
Viktor Vartdal Johansen	s315615	viktor.v.johansen@hotmail.com

Table of Content

1. Presentation
- Client
  - About the Client
- Person of Contact
- Client Stakeholders
- Supervisor from OsloMET

2. Project Background

3. Preface

4. Short System Description
  - Deliverables
  - System Diagram

5. Requirements
- 5.1 Functional requirements
- 5.2 Non-Functional Requirements
  - 5.2.1 Software Requirements
  - 5.2.2 Design Decisions
- 5.3 Stretch requirements

6. Actors and User Stories
- 6.1 Actors
  - 6.1.2 Actor Characteristics
    - Data scientist
    - Data engineers
- 6.2 User Stories

7. System Restriction
- 7.1 Security
  - End-to-end encryption
- 7.2 Data Storage/Cache
- 7.3 Open Source
- 7.4 Accessible API

8. Additional Requirements for System Construction
- 8.1 Process Requirements
  - 8.1.1 Continuous Integration/Continuous Delivery (CI/CD)
  - 8.1.2 System Development Framework
- 8.2 Technical Requirements
  - 8.2.1 System Packaging

9. Additional Requirements for Documentation
9.1 System Documentation
10. Dictionary

1. Presentation

Client

Arbeids- og velferdsdirektoratet
Sannergata 2
0557 Oslo, Norway

About the Client

The Norwegian Labour and Welfare Administration (NAV) is the backbone of the Norwegian welfare state, administering a third of the national budget and servicing almost 2.8 million people through a range of schemes such as unemployment benefit, work assessment allowance, sickness benefit, pension, child benefit and cash-for-care benefit.

In addition, NAV manages and retains stewardship of several important data sources containing information on its users and the services it provides. NAV IT is currently in the midst of a digital transformation, undergoing significant changes in team organization, work processes and harnessing new technologies.

Its use of data in the development of new and improved services is often contingent upon strict data privacy practices and its ability to innovate in a privacy-preserving manner.

Our client for this assignment is NAV IT Data og Innsikt. Data og Innsikt is a department within NAV IT. The department delivers the development of systems and operations of data storage, data processing, data access and analytics.

Person of Contact

Name	Role	Email

Gøran Berntsen	Tech Lead - Åpne Data	Goran.Berntsen@nav.no

Client Stakeholders

Robindra Prabhu	Data Scientist - AI Lab	Robindra.Prabhu@nav.no
Paul Bencze	Tech Lead - AI Lab	Paul.Bencze@nav.no

Supervisor from OsloMET

Name	**Role**	**Email**
Eva Hadler Vihovde	Associate Professor	evahadler.vihovde@oslomet.no

2. Project Background

AI Lab is a department in NAV IT Data og Innsikt, functioning as NAV ITs internal knowledge hub in machine learning and data science. One of the areas the team currently is handling is data anonymization. This area presents problems in both the legal and ethical domain, because of its close connection to personal data.

Data anonymization is a large field with many projects worldwide. There are well established models and algorithms for both anonymization and analytics of data. AI Lab is currently utilizing both internal and external tools for data anonymization. One such tool is ARX, which is widely regarded as top-class anonymization software. ARX is an open source (Apache License, Version 2.0) project distributed as a GUI application and as a Java JAR library. It is prominently used by large organizations to anonymize health and patient data.

ARX contains a large and powerful amount of functionality for data anonymization, but the interaction with said functionality is limited to either interaction with the GUI application or programmatically with the Java API provided by the JAR. Neither option is well suited to modern data science applications.

Data science today is typically conducted within programming languages like Python and R. The data scientist develops scripts, notebooks and larger programs for extracting and analysing data. Early stage tasks typically include data cleaning and data transformation. Anonymisation may be viewed as such a data transformation task, but ARX currently does not integrate seamlessly into the typical Python/R-based data science workflow.

Moreover, while ARX provides functionality and flexibility for a skilled user, the front-end arguably requires knowledge of technical concepts and methods in anonymisation above and beyond that of the typical developer and which requires a non-trivial investment of time to learn and understand.

The AI-lab has therefore requested the group to:

- Provide access to ARX functionality from modern data science toolsets and workflows
- Provide an extendable framework for making state-of-the-art anonymisation methods accessible to a wider audience in NAV IT by lowering the barriers to use.

3. Preface

Sommerville (2010). “The software requirements document [...] is an official statement of what the system developers should implement.” “[...] the requirements document has to be a compromise between communicating the requirements to customers, defining the requirements in precise detail for software developers and testers, and including information about possible system evolution.” *Software Engineering* (91-92)

This is a technical document meant for the product stakeholders, with the purpose of providing clarification and guidance to the project. It contains both the technical and non-technical requirements and they are written in close cooperation with our client.

The terms *data* and *dataset* are used continuously throughout the documentation. Unless anything else is specified, the term refers to tabular data/datasets. The problem space is data anonymization, as such, when talking about data/datasets we are generally referring to population data.

The team employs an Agile work process. The product specification also serves as our basis for the backlog of user stories to be prioritised and implemented during the project lifetime.

4. Short System Description

The system will provide access to anonymization tools for data scientists at NAV IT. A data scientist should be able to anonymize tabular dataset based on user-specific configurations. Configurability includes privacy models, column attribute types and transformation models that determine how much data will be lost in the resulting anonymized dataset.

A common use case would be in a workflow where the data scientist is manipulating a dataset, and requires dynamic analysis of the data’s anonymity metrics. Another use case could involve integrating the system in a data pipeline to provide data analytics and anonymization capabilities.

Deliverables

- **Python Package - working title: PyAaaS**

Python package wrapper providing abstracted access to the backend service.

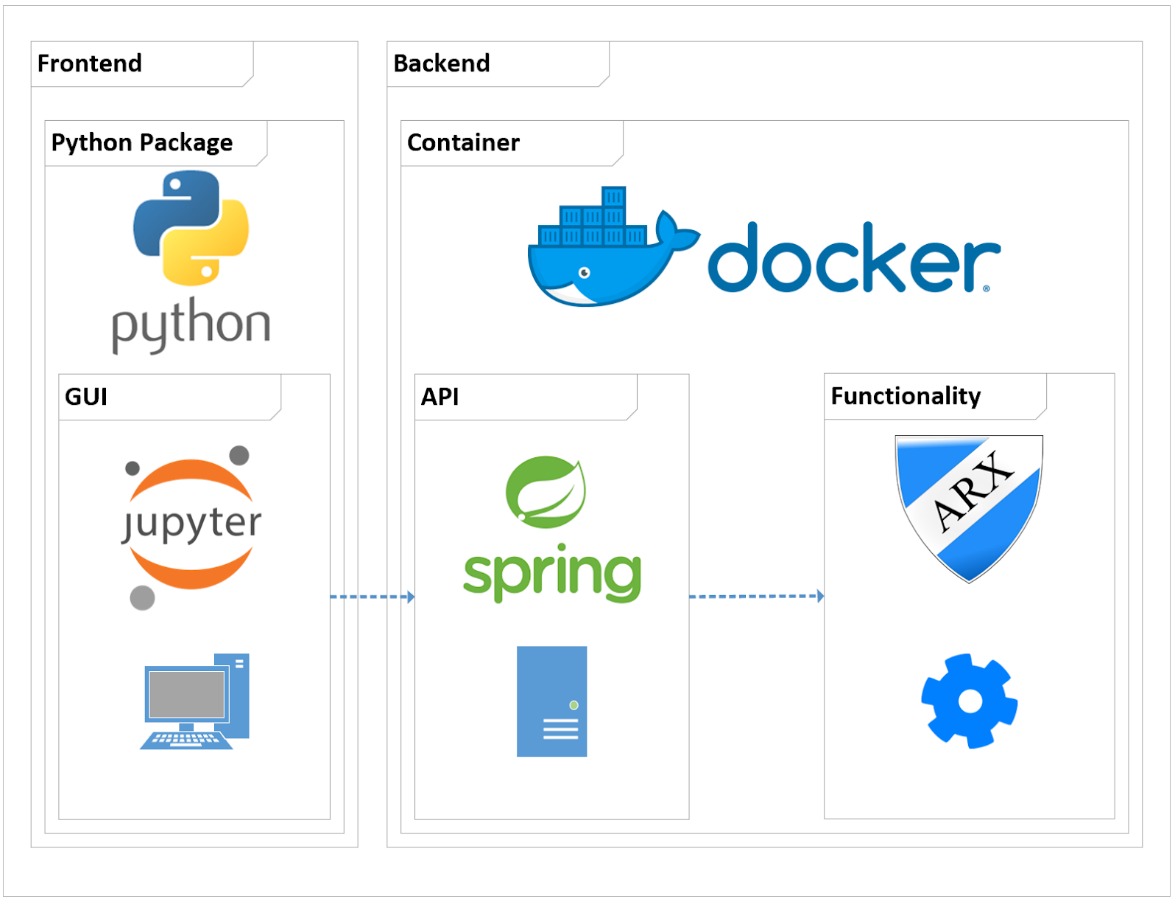
- **Web Service - working title: AaaS**

  Java Spring web service.

System Diagram

Preliminary draft of the system to be implemented

Jupyter notebook is a common user interface among data scientists, and will be a important platform for the system to support. In a Jupyter Notebook a data scientist that wishes to anonymize or analyze a dataset will import a Python package which wraps and abstracts the backend service. The backend service utilizes the ARX library and Spring framework to deliver the anonymization and analytics functionality as a web service. The service is packaged as a Docker container.



5. Requirements

The collection of system requirements defined in collaboration with the client(NAV IT - AI lab)

5.1 Functional requirements

- The system will provide the ability to complete data anonymization with the provided user configurations on tabular datasets.
- The system will provide the ability to analyze re-identification risks on tabular datasets.
- The system will provide the ability to configure the Privacy Models to use in the anonymization.
- The system will provide the ability to configure data Attribute Type to use in the anonymization.
- The system will provide the ability to configure the Transformation Models to use in the anonymization.
- The system will provide the ability to produce a visual presentation of data anonymity metrics.
- The system will provide the ability to compare data from before and after data anonymization.
- The system will be able to consume data in a variety of formats including (pandas.DataFrame, path to csv file, url to data resource, csv string, JSON).
- The system will be able to deliver the anonymized dataset in a variety of formats including (pandas.DataFrame, csv file, JSON).
- The system will be able to deliver metrics about the anonymization in a variety of formats including (pandas.DataFrame, csv file, JSON, Data Package).
- The system will provide the ability to produce data package metadata regarding the anonymization process that has been completed on the dataset and the relevant metrics.

5.2 Non-Functional Requirements

5.2.1 Software Requirements

- The client has requested that the team uses the ARX anonymizer library to implement anonymization functionality.
- The client has required that resulting anonymization process has to be more efficient than the previous and reduce the hours that are spent doing this manually.
- The client has required that the project is published as an open source project with an MIT licence.

- The client has required that the system backend will be packaged as a docker image so the service can be deployed to the NAIS/Kubernetes platforms.
- The client has required that the team develop a Python package to “wrap” the web service, it will provide easy integration and interaction between the web service and data scientist tools and processes.
- The client has requested that the Python package has to be designed for use in a Jupyter notebook.
- The client has required that the system will utilize end to end encryption for data in transit, to and from the web service backend.

5.2.2 Design Decisions

Design decisions made by the team in collaboration with the customer to achieve the stated goal of the system.

- English will be the main language used for both the documentation and programming to make it easier for the team to deliver on the open source requirement from the client.
- The team has decided to utilize Java as its runtime environment for the backend service. The ARX library that the client has requested to be used is packaged as a Java JAR file. Using Java was a logical choice.
- The team has decided to use a service architecture to decouple the different logical components of the project. A service architecture will also deliver on the clients wish to be able to scale the system dynamically according to use.
- The team has decided to utilize Spring as its backend framework to deliver a web service in accordance with the service architecture. Spring is the defacto standard for Java web applications and has great libraries for development of secure, scalable web applications.

5.3 Stretch requirements

Stretch requirements are wishes from the customer that the development team will try to achieve if there is time left after the main requirements.

- The system will be able to **auto generate a hierarchy** level based on the column attribute type.
- Provide specific Transformation Model hierarchies for **NAV specific use cases** (eg. Norwegian geographical hierarchies, Norwegian zip code hierarchies).

- Provide an alternative **web frontend** that provides a lower barrier to entry, and a more user friendly interaction.
- Grafana dashboard for surveillance of the anonymization service.

6. Actors and User Stories

Description of Actors, their characteristics and user stories that defines their interest.

6.1 Actors

An actor is an entity who interacts with the system from the outside. Primary actors are those who require assistance from the system. While secondary actors are those who the system needs assistance from.

Primary Actors	**Description**
Data Scientist	A data scientist working at NAV.

Secondary Actors	**Description**
Data Engineer	A data engineer working at NAV.

6.1.2 Actor Characteristics

Data scientist

Data scientist are super users with good programming and statistical knowledge. Data scientists are the primary user of the system.

Data engineers

Data engineers are system engineers specialised in data driven applications. They support the data scientists by building and deploying data pipelines and other data infrastructure.

6.2 User Stories

User stories are one of the primary development artifacts for Scrum project teams. A user story is a very high-level definition of a requirement, containing just enough information so that the developers can produce a reasonable estimate of the effort to implement it.

Actor	**Story**	**Priority**
Data Scientist	As a data-scientist, I would like to easily **visualize the anonymization metrics** for anonymized datasets Reasoning: Visualization is a powerful tool to get an understanding of complex data.	High
Data Scientist	As a data-scientist, I would like to **analyze the anonymization metrics (re-identification risks)** of my dataset Reasoning: Getting metrics of the anonymization level of a dataset is necessary to judge how safe the dataset is to use in production, and/or if the dataset should be anonymized further.	High
Data Scientist	As a data scientist, I would like to have a single source where to lookup the **documentation for the PyAaaS package** (AaaS Python wrapper package). Reasoning: To facilitate efficient use of the Python package it is critical to make available up-to-date documentation of both the package API and tutorials for common use cases.	Medium
Data Scientist	As a data scientist, I would like to be able to **configure the Privacy Models** to be used when anonymizing my dataset	High
Data Scientist	As a data scientist, I would like to be able to **configure the Transformation Models** to be used when anonymizing my dataset.	High
Data Scientist	As a data scientist, I would like to be able to use **K-Anonymization as a Privacy Model** for my dataset.	High
Data Scientist	As a data scientist, I would like to be able to **set a global transformation scheme for all record**s in a column/field.	High
Data Scientist	As a data scientist, I would like to be able to **set a local transformation scheme for a column/field**. Meaning a unique transformation scheme for each individual row or subset of rows in a column/field.	Low

Data Scientist	As a data scientist, I would like to be able to use a **Value Generalization hierarchy as a Transformation Model** for a column/field.	High
Data Scientist	As a data scientist, I would like to be able to use **random sampling as a Transformation Model** for a column/field	Medium
Data Scientist	As a data scientist, I would like to be able to use **attribute suppression as a Transformation Model** for a column/field	Medium
Data Scientist	As a data scientist, I would like to use **microaggregation as a Transformation Model**.	Medium
Data Scientist	As a data scientist, I would like to use **Top- and bottom-coding as a Transformation Model.**	Medium
Data Scientist	As a data scientist, I would like to use **Categorization as Transform Model for a column/field.**	Medium
Data Scientist	As a data scientist, I would like to **identify rows affected by lowest risk** in a dataset.	Low
Data Scientist	As a data scientist, I would like to determine the **Lowest prosecutor re-identification risk.**	Low
Data Scientist	As a data scientist, I would like to determine **highest prosecutor re-identification risk.**	High
Data Scientist	As a data scientist, I would like to **identify rows affected by highest risk.**	High
Data Scientist	As a data scientist, I would like to **determine average prosecutor re-identification risk.**	High
Data Scientist	As a data scientist, I would like to **determine fraction of unique records.**	High
Data Scientist	As a data scientist, I would like to be able to **use K-map as a Privacy Model** for my dataset.	Low
Data Scientist	As a data scientist, I would like to be able to use **Average risk as a Privacy Model** for my dataset.	Low
Data Scientist	As a data scientist, I would like to be able to use **Population uniqueness as a Privacy Model** for my dataset.	Low
Data Scientist	As a data scientist, I would like to be able to use **Sample uniqueness as a Privacy Model** for my dataset.	Low

Data Scientist	As a data scientist, I would like to be able to use **δ-Disclosure privacy as a Privacy Model** for my dataset.	Low
Data Scientist	As a data scientist, I would like to be able to use **β-Likeness privacy as a Privacy Model** for my dataset.	Low
Data Scientist	As a data scientist, I would like to be able to use **δ-Presence privacy as a Privacy Model** for my dataset.	Low
Data Scientist	As a data scientist, I would like to be able to use **Profitability privacy as a Privacy Model** for my dataset.	Low
Data Scientist	As a data scientist, I would like to be able to use **Differential privacy as a Privacy Model** for my dataset.	Medium
Data Scientist	As a data scientist, I would like to be able to use **ℓ-Diversity as a Privacy Model** for my dataset	High
Data Scientist	As a data scientist, I would like to be able to **set presets for anonymization** e.g loss percentage, min/max risk of prosecution	Low
Data Scientist	As a data scientist, I would like to be able to **verify whether an anonymized dataset** has been anonymized from an original dataset	Low

Actor	**Story**	**Priority**
Data Engineer	As a data engineer supporting the AaaS service in NAV, I would like to be able to **deploy the service to a docker container** environment.	high
Data Engineer	As a data engineer supporting the AaaS service in NAV, I would like **continuous information about the build status of the AaaS web service** source code	medium
Data Engineer	As a data engineer supporting the AaaS service in NAV, I would like a **single source/location for documentation,** for setup and deployment of the AaaS service.	medium
Data Engineer	As a data engineer supporting the AaaS service in NAV, I would like a single **source/location for the AaaS projects JavaDoc**	low
Data Engineer	As a data engineer supporting the AaaS service in NAV,I would like to have **logging available from the AaaS service.**	high

7. System Restriction

This section explains the system and client defined restriction. In this section the team defines the limitations of the system to be developed.

7.1 Security

End-to-end encryption

The client has requested that the system use end-to-end encryption between the backend service and consumers of the service (Python package, Third-party applications). The team has decided to use TLS/SSL provided by the Spring framework.

7.2 Data Storage/Cache

The developed system cannot store or implement caching due to the sensitive nature of the data used.

7.3 Open Source

In accordance with the clients request the project with all source code and documentation will be released under an open source licence. Currently the team working under the MIT Licence.

7.4 Accessible API

Our API must follow RESTful guidelines and strive to provide endpoints that allow for seamless interaction with the ARX library.

8. Additional Requirements for System Construction

Additional non-functional requirements defined by the development team in cooperation with the client. These requirements are meant to improve the quality and maintainability of the end product.

8.1 Process Requirements

8.1.1 Continuous Integration/Continuous Delivery (CI/CD)

CI platform

The system uses Travis CI and Code Climate as Continuous Integration tools. Travis ensures that the codes are tested, while code climate checks the code quality and test coverage before being pushed to the repository. Along with Travis and Code Climate, the team uses GitHub with merge rules to ensure that the master build stays stable.

Each new iteration of the master build is packaged into a jar file, which would be packaged again into a Docker image. This way we will have different stable builds that can be deployed easily as a Docker container.

Version Control System (VCS)

Travis CI along with GitHub Merge rules maintains the stability of each version before a release.

Github Merge rules does not allow directly pushing to the master build, along with not allowing to push to the repository unless all test class passes. Each time there is a new implementation a new branch needs to be made. This new branch is then tested before being pushed to the repository, which is then finally merged to the master build.

Travis CI instantiates a docker container that runs the build being pushed along with all the test classes. If a build passes Travis will then allow the build to pushed.

Static Code Analysis

Code climate is used to ensure the maintainability and test coverage of the codes written. Travis generates a test report using Jacoco, which is then forwarded to Code Climate by using a unique ID. Code climate reads through the report and generates a grade for test coverage of our system. Code climate is directly link to the systems GitHub repository, granting access to check the quality of the codes written. Based on the quality of the code a grade will be generated for the maintainability of our system.

8.1.2 System Development Framework

The team is developing the system under the SCRUM framework.

8.2 Technical Requirements

8.2.1 System Packaging

Backend Service

- Docker Image

Python Wrapper Package

- Python package wheel and source distribution

9. Additional Requirements for Documentation

Additional non-functional requirements defined by the team in cooperation with the client. These requirements are meant to improve the quality of the documentation.

9.1 System Documentation

The system backend service and python package will be delivered with documentation for the corresponding to the intended usage.

Backend Service Documentation

- Setup and deployment tutorial
- System JavaDoc

Python Wrapper Package Documentation

- Installation
- Common usage tutorial
- Examples (notebooks)
- API docs

10. Dictionary

Attribute Type - the disclosure risks from which a dataset is to be protected.

ARX - is a comprehensive open source software for anonymizing sensitive personal data.

Code Climate - engineering process insights and automated code review for GitHub and GitHub Enterprise help you ship better software, faster.

Continuous Integration - the practice of merging all developer working copies to a shared mainline several times a day.

Data Package - Data Package is a simple container format used to describe and package a collection of data. link: https://frictionlessdata.io/data-packages/

Data pipeline - a data pipeline is a set of actions that extract data (or directly analytics and visualization) from various sources.

Docker - the fastest growing cloud-enabling technology and driving a new era of computing and application architecture with their lightweight approach to bundle applications and dependencies into isolated, yet highly portable application packages.

GitHub - a git repository hosting platform. Commonly used for open-source software.

Jacoco - is a free Java code coverage library distributed under the Eclipse Public License.

Java - is a high-power, stable and highly trusted programing language. Commonly used in backend application with a requirement for stability.

Javascript - an object-oriented computer programming language commonly used to create interactive effects within web browsers.

JSON - is an open-standard file format that uses human-readable text to transmit data objects consisting of attribute–value pairs and array data types.

Jupyter Notebook - Project Jupyter exists to develop open-source software, open-standards, and services for interactive computing across dozens of programming languages.

Kubernetes - is an open-source container orchestration system for automating application deployment, scaling, and management.

MIT Licence - The MIT License is a permissive free software license originating at the Massachusetts Institute of Technology (MIT). link: https://opensource.org/licenses/MIT

NAIS - is an application platform built to increase development speed by providing our developers at NAV with the best possible tools to develop and run their applications.

Pandas - is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.

Pandas Dataframe - a one-dimensional labeled array capable of holding any data type with axis labels or index.

Privacy Model - Privacy models are specific algorithms applied to a dataset to protect the dataset from an identification risk vector.

Product Stakeholders - NAV IT, OsloMET, the development team

Python - Python is a powerful, high-level language. Used in everything from web apps to data-science/machine learning.

Spring - is an open-source, high-power, Java application framework for business applications.

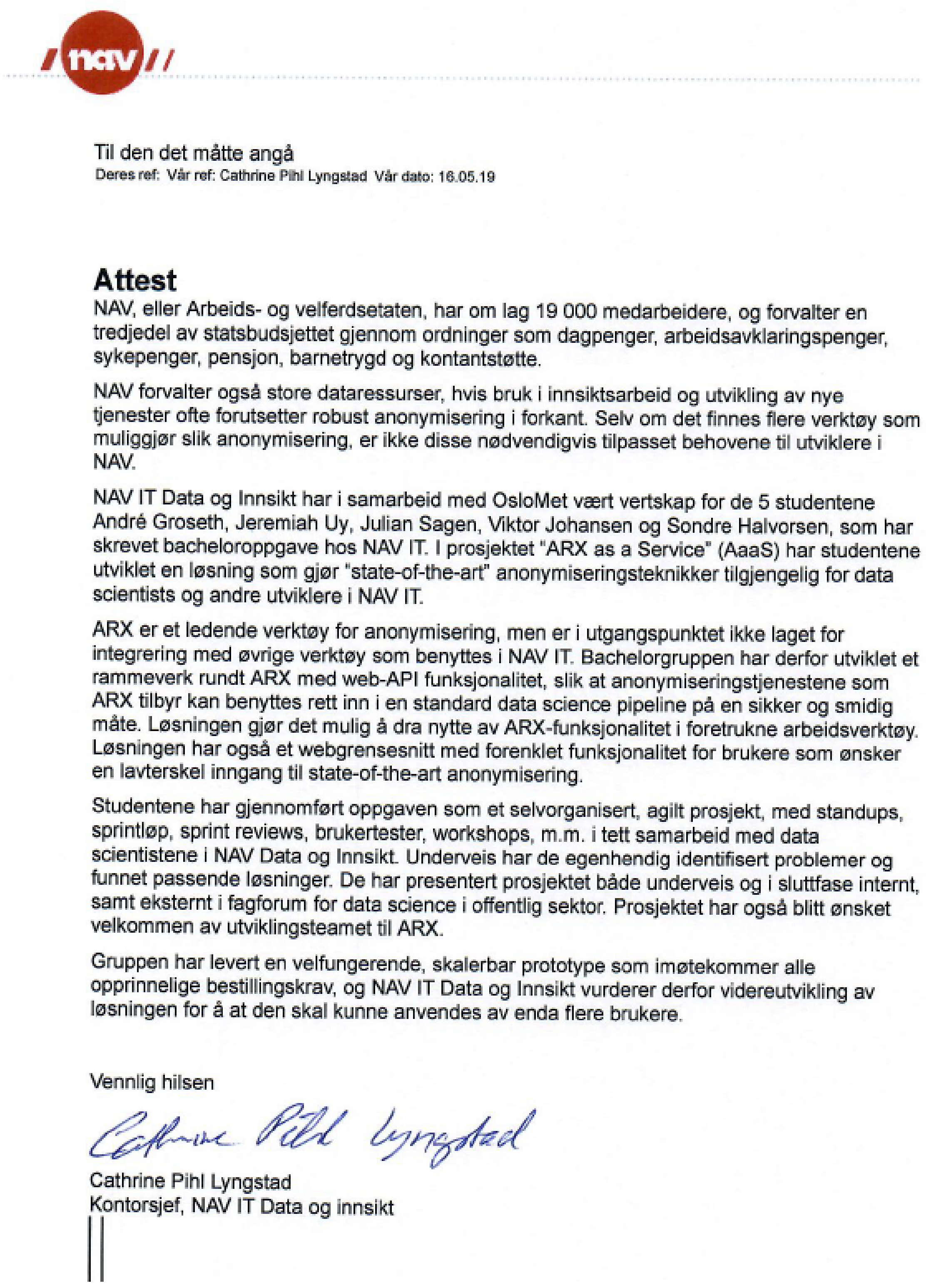
Scrum - is an agile framework for managing knowledge work, with an emphasis on software development. It is designed for teams of three to nine members, who break their work into actions that can be completed within timeboxed iterations, called "sprints", no longer than one month and most commonly two weeks, then track progress and re-plan in 15-minute stand-up meetings, called daily scrums.

Tabular dataset - a data consisting of or presented in columns or tables.

Transformation Model - Transformation Models are strategies that describes how a specific column in the dataset should lose data as the anonymization tries to achieve the required anonymization level specified by the Privacy Model.

Travis CI - is a hosted, distributed continuous integration service used to build and test software projects hosted at GitHub.

7.3 Testimonial from NAV IT (Norwegian)



Til den det måtte angå

Deres ref: Vår ref: Cathrine Pihl Lyngstad Vår dato: 16.05.19

Attest

NAV, eller Arbeids- og velferdsetaten, har om lag 19 000 medarbeidere, og forvalter en tredjedel av statsbudsjettet gjennom ordninger som dagpenger, arbeidsavklaringspenger, sykepenger, pensjon, barnetrygd og kontantstøtte.

NAV forvalter også store dataressurser, hvis bruk i innsiktsarbeid og utvikling av nye tjenester ofte forutsetter robust anonymisering i forkant. Selv om det finnes flere verktøy som muliggjør slik anonymisering, er ikke disse nødvendigvis tilpasset behovene til utviklere i NAV.

NAV IT Data og Innsikt har i samarbeid med OsloMet vært vertskap for de 5 studentene André Groseth, Jeremiah Uy, Julian Sagen, Viktor Johansen og Sondre Halvorsen, som har skrevet bacheloroppgave hos NAV IT. I prosjektet "ARX as a Service" (AaaS) har studentene utviklet en løsning som gjør "state-of-the-art" anonymiseringsteknikker tilgjengelig for data scientists og andre utviklere i NAV IT.

ARX er et ledende verktøy for anonymisering, men er i utgangspunktet ikke laget for integrering med øvrige verktøy som benyttes i NAV IT. Bachelorgruppen har derfor utviklet et rammeverk rundt ARX med web-API funksjonalitet, slik at anonymiseringstjenestene som ARX tilbyr kan benyttes rett inn i en standard data science pipeline på en sikker og smidig måte. Løsningen gjør det mulig å dra nytte av ARX-funksjonalitet i foretrukne arbeidsverktøy. Løsningen har også et webgrensesnitt med forenklet funksjonalitet for brukere som ønsker en lavterskel inngang til state-of-the-art anonymisering.

Studentene har gjennomført oppgaven som et selvorganisert, agilt prosjekt, med standups, sprintløp, sprint reviews, brukertester, workshops, m.m. i tett samarbeid med data scientistene i NAV Data og Innsikt. Underveis har de egenhendig identifisert problemer og funnet passende løsninger. De har presentert prosjektet både underveis og i sluttfase internt, samt eksternt i fagforum for data science i offentlig sektor. Prosjektet har også blitt ønsket velkommen av utviklingsteamet til ARX.

Gruppen har levert en velfungerende, skalerbar prototype som imøtekommer alle opprinnelige bestillingskrav, og NAV IT Data og Innsikt vurderer derfor videreutvikling av løsningen for å at den skal kunne anvendes av enda flere brukere.

Vennlig hilsen

Cathrine Pihl Lyngstad

Cathrine Pihl Lyngstad
Kontorsjef, NAV IT Data og innsikt

Figure 84 - Testimony from NAV IT

7.4 Analyzation HTTP JSON request body

```
$ curl 'http://localhost:8080/api/analyze' -i -X POST \
    -H 'Content-Type: application/json;charset=UTF-8' \
    -d '{
  "data" : [ [ "age", "gender", "zipcode" ], [ "34", "male", "81667" ], [ "35", "female",
"81668" ], [ "36", "male", "81669" ], [ "37", "female", "81670" ], [ "38", "male",
"81671" ], [ "39", "female", "81672" ], [ "40", "male", "81673" ], [ "41", "female",
"81674" ], [ "42", "male", "81675" ], [ "43", "female", "81676" ], [ "44", "male",
"81677" ] ],
  "attributes" : [ {
    "field" : "age",
    "attributeTypeModel" : "IDENTIFYING",
    "hierarchy" : null
  }, {
    "field" : "gender",
    "attributeTypeModel" : "SENSITIVE",
    "hierarchy" : null
  }, {
    "field" : "zipcode",
    "attributeTypeModel" : "QUASIIDENTIFYING",
    "hierarchy" : null
  } ],
  "privacyModels" : null,
  "suppressionLimit" : null
}'
```

7.5 Analyzation HTTP JSON response body

```
HTTP/1.1 200 OK
Content-Length: 4123
Content-Type: application/json;charset=UTF-8

{
  "reIdentificationRisk" : {
    "measures" : {
      "estimated_journalist_risk" : 1.0,
      "records_affected_by_highest_prosecutor_risk" : 1.0,
      "sample_uniques" : 1.0,
      "lowest_risk" : 1.0,
      "estimated_prosecutor_risk" : 1.0,
      "highest_journalist_risk" : 1.0,
      "records_affected_by_lowest_risk" : 1.0,
      "average_prosecutor_risk" : 1.0,
      "estimated_marketer_risk" : 1.0,
      "highest_prosecutor_risk" : 1.0,
      "records_affected_by_highest_journalist_risk" : 1.0,
      "population_uniques" : 1.0
    },
    "attackerSuccessRate" : {
      "successRates" : {
        "Prosecutor_attacker_success_rate" : 1.0,
        "Marketer_attacker_success_rate" : 1.0,
        "Journalist_attacker_success_rate" : 1.0
      }
    },
    "quasiIdentifiers" : [ "zipcode" ],
    "populationModel" : "ZAYATZ"
  },
  "distributionOfRisk" : {
    "riskIntervalList" : [ {
      "interval" : "]50,100]",
      "recordsWithRiskWithinInteval" : 1.0,
      "recordsWithMaxmalRiskWithinInterval" : 1.0
    }, {
      "interval" : "]33.4,50]",
      "recordsWithRiskWithinInteval" : 0.0,
      "recordsWithMaxmalRiskWithinInterval" : 0.0
    }, {
      "interval" : "]25,33.4]",
      "recordsWithRiskWithinInteval" : 0.0,
      "recordsWithMaxmalRiskWithinInterval" : 0.0
    }, {
      "interval" : "]20,25]",
      "recordsWithRiskWithinInteval" : 0.0,
      "recordsWithMaxmalRiskWithinInterval" : 0.0
    }, {
      "interval" : "]16.7,20]",
      "recordsWithRiskWithinInteval" : 0.0,
      "recordsWithMaxmalRiskWithinInterval" : 0.0
    }, {
      "interval" : "]14.3,16.7]",
      "recordsWithRiskWithinInteval" : 0.0,
      "recordsWithMaxmalRiskWithinInterval" : 0.0
    }, {
      "interval" : "]12.5,14.3]",
      "recordsWithRiskWithinInteval" : 0.0,
      "recordsWithMaxmalRiskWithinInterval" : 0.0
    }, {
      "interval" : "]10,12.5]",
      "recordsWithRiskWithinInteval" : 0.0,
      "recordsWithMaxmalRiskWithinInterval" : 0.0
    }, {
      "interval" : "]9,10]",
      "recordsWithRiskWithinInteval" : 0.0,
      "recordsWithMaxmalRiskWithinInterval" : 0.0
    }, {
      "interval" : "]8,9]",
      "recordsWithRiskWithinInteval" : 0.0,
      "recordsWithMaxmalRiskWithinInterval" : 0.0
    }, {
      "interval" : "]7,8]",
      "recordsWithRiskWithinInteval" : 0.0,
      "recordsWithMaxmalRiskWithinInterval" : 0.0
    }, {
      "interval" : "]6,7]",
      "recordsWithRiskWithinInteval" : 0.0,
      "recordsWithMaxmalRiskWithinInterval" : 0.0
    }, {
      "interval" : "]5,6]",
      "recordsWithRiskWithinInteval" : 0.0,
      "recordsWithMaxmalRiskWithinInterval" : 0.0
    }, {
      "interval" : "]4,5]",
      "recordsWithRiskWithinInteval" : 0.0,
      "recordsWithMaxmalRiskWithinInterval" : 0.0
    }, {
      "interval" : "]3,4]",
      "recordsWithRiskWithinInteval" : 0.0,
      "recordsWithMaxmalRiskWithinInterval" : 0.0
    }, {
      "interval" : "]2,3]",
      "recordsWithRiskWithinInteval" : 0.0,
      "recordsWithMaxmalRiskWithinInterval" : 0.0
    }, {
      "interval" : "]1,2]",
      "recordsWithRiskWithinInteval" : 0.0,
      "recordsWithMaxmalRiskWithinInterval" : 0.0
    }, {
      "interval" : "]0.1,1]",
      "recordsWithRiskWithinInteval" : 0.0,
      "recordsWithMaxmalRiskWithinInterval" : 0.0
    }, {
      "interval" : "]0.01,0.1]",
      "recordsWithRiskWithinInteval" : 0.0,
      "recordsWithMaxmalRiskWithinInterval" : 0.0
    }, {
      "interval" : "]0.001,0.01]",
      "recordsWithRiskWithinInteval" : 0.0,
      "recordsWithMaxmalRiskWithinInterval" : 0.0
    }, {
      "interval" : "]0.0001,0.001]",
      "recordsWithRiskWithinInteval" : 0.0,
      "recordsWithMaxmalRiskWithinInterval" : 0.0
    }, {
      "interval" : "]1e-5,0.0001]",
      "recordsWithRiskWithinInteval" : 0.0,
      "recordsWithMaxmalRiskWithinInterval" : 0.0
    }, {
      "interval" : "]1e-6,1e-5]",
      "recordsWithRiskWithinInteval" : 0.0,
      "recordsWithMaxmalRiskWithinInterval" : 0.0
    }, {
      "interval" : "]0,1e-6]",
      "recordsWithRiskWithinInteval" : 0.0,
      "recordsWithMaxmalRiskWithinInterval" : 0.0
    } ]
  }
}
```

7.6 Anonymization HTTP JSON request body

```
$ curl 'http://localhost:8080/api/anonymize' -i -X POST \
    -H 'Content-Type: application/json;charset=UTF-8' \
    -d '{
  "data" : [ [ "age", "gender", "zipcode" ], [ "34", "male", "81667" ], [ "35", "female",
"81668" ], [ "36", "male", "81669" ], [ "37", "female", "81670" ], [ "38", "male",
"81671" ], [ "39", "female", "81672" ], [ "40", "male", "81673" ], [ "41", "female",
"81674" ], [ "42", "male", "81675" ], [ "43", "female", "81676" ], [ "44", "male",
"81677" ] ],
  "attributes" : [ {
    "field" : "age",
    "attributeTypeModel" : "IDENTIFYING",
    "hierarchy" : null
  }, {
    "field" : "gender",
    "attributeTypeModel" : "SENSITIVE",
    "hierarchy" : null
  }, {
    "field" : "zipcode",
    "attributeTypeModel" : "QUASIIDENTIFYING",
    "hierarchy" : [ [ "81667", "8166*", "816**", "81***", "8****", "*****" ], [ "81668",
"8166*", "816**", "81***", "8****", "*****" ], [ "81669", "8166*", "816**", "81***",
"8****", "*****" ], [ "81670", "8167*", "816**", "81***", "8****", "*****" ], [ "81671",
"8167*", "816**", "81***", "8****", "*****" ], [ "81672", "8167*", "816**", "81***",
"8****", "*****" ], [ "81673", "8167*", "816**", "81***", "8****", "*****" ], [ "81674",
"8167*", "816**", "81***", "8****", "*****" ], [ "81675", "8167*", "816**", "81***",
"8****", "*****" ], [ "81676", "8167*", "816**", "81***", "8****", "*****" ], [ "81677",
"8167*", "816**", "81***", "8****", "*****" ] ]
  } ],
  "privacyModels" : [ {
    "privacyModel" : "KANONYMITY",
    "params" : {
      "k" : "5"
    }
  }, {
    "privacyModel" : "LDIVERSITY_DISTINCT",
    "params" : {
      "column_name" : "gender",
      "l" : "2"
    }
  } ],
  "suppressionLimit" : 0.02
}'
```

7.7 Anonymization HTTP JSON response body

```
HTTP/1.1 200 OK
Content-Length: 7296
Content-Type: application/json;charset=UTF-8

{
  "anonymizeResult" : {
        "data" : [ [ "age", "gender", "zipcode" ], [ "*", "male", "816**" ], [ "*",
"female", "816**" ], [ "*", "male", "816**" ], [ "*", "female", "816**" ], [ "*", "male",
"816**" ], [ "*", "female", "816**" ], [ "*", "male", "816**" ], [ "*", "female", "816**"
], [ "*", "male", "816**" ], [ "*", "female", "816**" ], [ "*", "male", "816**" ] ],
        "anonymizationStatus" : "ANONYMOUS",
        "metrics" : {
        "attributeGeneralization" : [ {
        "name" : "zipcode",
        "type" : "QUASI_IDENTIFYING_ATTRIBUTE",
        "generalizationLevel" : 2
        } ],
        "processTimeMillisecounds" : 5,
        "privacyModels" : [ {
        "monotonicWithGeneralization" : true,
        "attribute" : "gender",
        "l" : 2.0,
        "localRecodingSupported" : true,
        "minimalClassSize" : 2,
        "requirements" : 4,
        "riskThresholdJournalist" : 0.5,
        "riskThresholdMarketer" : 0.5,
        "riskThresholdProsecutor" : 0.5,
        "minimalClassSizeAvailable" : true,
        "populationModel" : null,
        "dataSubset" : null,
        "subset" : null,
        "monotonicWithSuppression" : true,
        "sampleBased" : false,
        "subsetAvailable" : false
        }, {
        "monotonicWithGeneralization" : true,
        "k" : 5,
        "minimalClassSize" : 5,
        "requirements" : 1,
        "riskThresholdJournalist" : 0.2,
        "riskThresholdMarketer" : 0.2,
        "riskThresholdProsecutor" : 0.2,
        "localRecodingSupported" : true,
        "minimalClassSizeAvailable" : true,
        "populationModel" : null,
        "dataSubset" : null,
        "subset" : null,
        "monotonicWithSuppression" : true,
        "sampleBased" : false,
        "subsetAvailable" : false
        } ]
        },
        "attributes" : [ {
        "field" : "age",
        "attributeTypeModel" : "IDENTIFYING",
        "hierarchy" : null
        }, {
        "field" : "gender",
        "attributeTypeModel" : "SENSITIVE",
        "hierarchy" : null
        }, {
        "field" : "zipcode",
        "attributeTypeModel" : "QUASIIDENTIFYING",
        "hierarchy" : [ [ "81667", "8166*", "816**", "81***", "8****", "*****" ], [
"81668", "8166*", "816**", "81***", "8****", "*****" ], [ "81669", "8166*", "816**",
"81***", "8****", "*****" ], [ "81670", "8167*", "816**", "81***", "8****", "*****" ], [
"81671", "8167*", "816**", "81***", "8****", "*****" ], [ "81672", "8167*", "816**",
"81***", "8****", "*****" ], [ "81673", "8167*", "816**", "81***", "8****", "*****" ], [
"81674", "8167*", "816**", "81***", "8****", "*****" ], [ "81675", "8167*", "816**",
"81***", "8****", "*****" ], [ "81676", "8167*", "816**", "81***", "8****", "*****" ], [
"81677", "8167*", "816**", "81***", "8****", "*****" ] ]
        } ]
  },
  "riskProfile" : {
        "reIdentificationRisk" : {
        "measures" : {
        "estimated_journalist_risk" : 0.09090909090909091,
        "records_affected_by_highest_prosecutor_risk" : 1.0,
        "sample_uniques" : 0.0,
        "lowest_risk" : 0.09090909090909091,
        "estimated_prosecutor_risk" : 0.09090909090909091,
        "highest_journalist_risk" : 0.09090909090909091,
        "records_affected_by_lowest_risk" : 1.0,
        "average_prosecutor_risk" : 0.09090909090909091,
        "estimated_marketer_risk" : 0.09090909090909091,
        "highest_prosecutor_risk" : 0.09090909090909091,
        "records_affected_by_highest_journalist_risk" : 1.0,
        "population_uniques" : 0.0
        },
        "attackerSuccessRate" : {
        "successRates" : {
        "Prosecutor_attacker_success_rate" : 0.09090909090909091,
        "Marketer_attacker_success_rate" : 0.09090909090909091,
        "Journalist_attacker_success_rate" : 0.09090909090909091
        }
        },
        "quasiIdentifiers" : [ "zipcode" ],
        "populationModel" : "DANKAR"
        },
        "distributionOfRisk" : {
        "riskIntervalList" : [ {
```

```
        "interval" : "]50,100]",
        "recordsWithRiskWithinInteval" : 0.0,
        "recordsWithMaxmalRiskWithinInterval" : 1.0
        }, {
        "interval" : "]33.4,50]",
        "recordsWithRiskWithinInteval" : 0.0,
        "recordsWithMaxmalRiskWithinInterval" : 1.0
        }, {
        "interval" : "]25,33.4]",
        "recordsWithRiskWithinInteval" : 0.0,
        "recordsWithMaxmalRiskWithinInterval" : 1.0
        }, {
        "interval" : "]20,25]",
        "recordsWithRiskWithinInteval" : 0.0,
        "recordsWithMaxmalRiskWithinInterval" : 1.0
        }, {
        "interval" : "]16.7,20]",
        "recordsWithRiskWithinInteval" : 0.0,
        "recordsWithMaxmalRiskWithinInterval" : 1.0
        }, {
        "interval" : "]14.3,16.7]",
        "recordsWithRiskWithinInteval" : 0.0,
        "recordsWithMaxmalRiskWithinInterval" : 1.0
        }, {
        "interval" : "]12.5,14.3]",
        "recordsWithRiskWithinInteval" : 0.0,
        "recordsWithMaxmalRiskWithinInterval" : 1.0
        }, {
        "interval" : "]10,12.5]",
        "recordsWithRiskWithinInteval" : 0.0,
        "recordsWithMaxmalRiskWithinInterval" : 1.0
        }, {
        "interval" : "]9,10]",
        "recordsWithRiskWithinInteval" : 1.0,
        "recordsWithMaxmalRiskWithinInterval" : 1.0
        }, {
        "interval" : "]8,9]",
        "recordsWithRiskWithinInteval" : 0.0,
        "recordsWithMaxmalRiskWithinInterval" : 0.0
        }, {
        "interval" : "]7,8]",
        "recordsWithRiskWithinInteval" : 0.0,
        "recordsWithMaxmalRiskWithinInterval" : 0.0
        }, {
        "interval" : "]6,7]",
        "recordsWithRiskWithinInteval" : 0.0,
        "recordsWithMaxmalRiskWithinInterval" : 0.0
        }, {
        "interval" : "]5,6]",
        "recordsWithRiskWithinInteval" : 0.0,
        "recordsWithMaxmalRiskWithinInterval" : 0.0
        }, {
        "interval" : "]4,5]",
        "recordsWithRiskWithinInteval" : 0.0,
        "recordsWithMaxmalRiskWithinInterval" : 0.0
        }, {
        "interval" : "]3,4]",
        "recordsWithRiskWithinInteval" : 0.0,
        "recordsWithMaxmalRiskWithinInterval" : 0.0
        }, {
        "interval" : "]2,3]",
        "recordsWithRiskWithinInteval" : 0.0,
        "recordsWithMaxmalRiskWithinInterval" : 0.0
        }, {
        "interval" : "]1,2]",
        "recordsWithRiskWithinInteval" : 0.0,
        "recordsWithMaxmalRiskWithinInterval" : 0.0
        }, {
        "interval" : "]0.1,1]",
        "recordsWithRiskWithinInteval" : 0.0,
        "recordsWithMaxmalRiskWithinInterval" : 0.0
        }, {
        "interval" : "]0.01,0.1]",
        "recordsWithRiskWithinInteval" : 0.0,
        "recordsWithMaxmalRiskWithinInterval" : 0.0
        }, {
        "interval" : "]0.001,0.01]",
        "recordsWithRiskWithinInteval" : 0.0,
        "recordsWithMaxmalRiskWithinInterval" : 0.0
        }, {
        "interval" : "]0.0001,0.001]",
        "recordsWithRiskWithinInteval" : 0.0,
        "recordsWithMaxmalRiskWithinInterval" : 0.0
        }, {
        "interval" : "]1e-5,0.0001]",
        "recordsWithRiskWithinInteval" : 0.0,
        "recordsWithMaxmalRiskWithinInterval" : 0.0
        }, {
        "interval" : "]1e-6,1e-5]",
        "recordsWithRiskWithinInteval" : 0.0,
        "recordsWithMaxmalRiskWithinInterval" : 0.0
        }, {
        "interval" : "]0,1e-6]",
        "recordsWithRiskWithinInteval" : 0.0,
        "recordsWithMaxmalRiskWithinInterval" : 0.0
        } ]
        }
  }
}
```

7.8 Redaction based hierarchy HTTP JSON request body

```
POST /api/hierarchy HTTP/1.1
Content-Length: 260
```

```
Content-Type: application/json;charset=UTF-8
Host: localhost:8080

{
  "column" : [ "0", "1", "2", "3", "4", "5", "6", "7", "8", "9" ],
  "builder" : {
       "type" : "redactionBased",
       "paddingCharacter" : " ",
       "redactionCharacter" : "*",
       "paddingOrder" : "RIGHT_TO_LEFT",
       "redactionOrder" : "RIGHT_TO_LEFT"
  }
}
```

7.9 Redaction based hierarchy HTTP JSON response body

```
HTTP/1.1 200 OK
Content-Length: 162
Content-Type: application/json;charset=UTF-8

{
  "hierarchy" : [ [ "0", "*" ], [ "1", "*" ], [ "2", "*" ], [ "3", "*" ], [ "4",
"*" ], [ "5", "*" ], [ "6", "*" ], [ "7", "*" ], [ "8", "*" ], [ "9", "*" ] ]
}
```

7.10 Interval based hierarchy HTTP JSON request body

```
POST /api/hierarchy HTTP/1.1
Content-Length: 832
Content-Type: application/json;charset=UTF-8
Host: localhost:8080

{
  "column" : [ "0", "1", "2", "3", "4", "5", "6", "7", "8", "9" ],
  "builder" : {
       "type" : "intervalBased",
       "intervals" : [ {
       "from" : 0,
       "to" : 2,
       "label" : "young"
       }, {
       "from" : 2,
       "to" : 4,
       "label" : "adult"
```

```
        }, {
        "from" : 4,
        "to" : 8,
        "label" : "old"
        }, {
        "from" : 8,
        "to" : 9223372036854775807,
        "label" : "very-old"
        } ],
        "levels" : [ {
        "level" : 0,
        "groups" : [ {
        "grouping" : 2,
        "label" : null
        } ]
        } ],
        "lowerRange" : {
        "snapFrom" : 0,
        "bottomTopCodingFrom" : 0,
        "minMaxValue" : -2305843009213693952
        },
        "upperRange" : {
        "snapFrom" : 81,
        "bottomTopCodingFrom" : 100,
        "minMaxValue" : 2305843009213693951
        },
        "dataType" : "LONG"
   }
}
```

7.11 Interval based hierarchy HTTP JSON response body

```
HTTP/1.1 200 OK
Content-Length: 352
Content-Type: application/json;charset=UTF-8

{
  "hierarchy" : [ [ "0", "young", "[0, 4[", "*" ], [ "1", "young", "[0, 4[", "*"
], [ "2", "adult", "[0, 4[", "*" ], [ "3", "adult", "[0, 4[", "*" ], [ "4",
"old", "[4, 8[", "*" ], [ "5", "old", "[4, 8[", "*" ], [ "6", "old", "[4, 8[",
"*" ], [ "7", "old", "[4, 8[", "*" ], [ "8", "very-old", "[8, 12[", "*" ], [
"9", "very-old", "[8, 12[", "*" ] ]
}
```

7.12 Order based hierarchy HTTP JSON request body

```
POST /api/hierarchy HTTP/1.1
Content-Length: 371
Content-Type: application/json;charset=UTF-8
Host: localhost:8080

{
  "column" : [ "Oslo", "Bergen", "Stockholm", "London", "Paris" ],
  "builder" : {
       "type" : "orderBased",
       "levels" : [ {
       "level" : 0,
       "groups" : [ {
       "grouping" : 3,
       "label" : "nordic-city"
       } ]
       }, {
       "level" : 0,
       "groups" : [ {
       "grouping" : 2,
       "label" : "mid-european-city"
       } ]
       } ]
  }
}
```

7.13 Order based hierarchy HTTP JSON response body

```
HTTP/1.1 200 OK
Content-Length: 204
Content-Type: application/json;charset=UTF-8

{
  "hierarchy" : [ [ "Oslo", "nordic-city", "*" ], [ "Bergen", "nordic-city", "*"
], [ "Stockholm", "nordic-city", "*" ], [ "London", "mid-european-city", "*" ],
[ "Paris", "mid-european-city", "*" ] ]
}
```