Remarks

108: odnośnik

109: To jest bardzo skrótowe.

Silesian
University
of Technology

SILESIAN UNIVERSITY OF TECHNOLOGY

FACULTY OF AUTOMATIC CONTROL, ELECTRONICS AND COMPUTER SCIENCE

PROGRAMME: INFORMATICS

Final Project

Data anonymisation web platform

author: Szymon Pluta

supervisor: Krzysztof Simiński, PhD DSc

Gliwice, January 2022

Contents

Abstract

The information has become a stimulus for innovation in today's data-driven world. The volumes of produced digital data are at an unstoppable growth. The world constantly evolves as organizations derive new insights from the collected knowledge. With information becoming a valued resource, our privacy concerns substantially increase. Refined data protection methodologies must be implemented to comply with the newly adopted strict regulations. Anonymisation is the state-of-the-art method for privacy-preserving identity protection. The objective of this thesis is to engineer a cutting-edge anonymisation platform that will provide diversified data masking capabilities, including suppression, generalisation, perturbation, randomisation, tokenisation, pattern masking, hashing, shortening, and artificial data substitution. The broad variety of encompassed techniques renders the web platform a versatile anonymisation tool suiting the needs of businesses and researchers.

Keywords: data anonymisation, privacy and data protection, GDPR compliance, platform as a web service

Chapter 1

Introduction

Technological advancements being observed in the past years fundamentally changed the relevance of data in today's digitalized world. Information became an innovation stimulus in the area of research and development. The quantity of data that organizations produce, process, store, and share is at a continuous growth. An enormous amount of 1.8 zettabytes ($1.8 \cdot 10^{21}$ bytes) of new data was produced only in 2011, and every two consecutive years this number is doubling [80]. After decades of observed technological advancement and innovation, the global internet traffic finally entered the zettabyte era, as it had reached a magnitude of one zettabyte in 2016, and in the calendar month being as early as September [16].

The vast quantities of processed information allowed for brand new research fields data science or big data analytics to form, which are used by organizations to derive new insights in a previously impossible way. Organizations collect and process the data to enhance the services they provide to the customers through statistical analysis or newly developed computer science processes including data mining and machine learning. The utility of delivered services is increased at a lower cost and improved efficiency through the insights extracted from the collected information about how the services are consumed [8]. Any modern organisation now processes digital information – the government, academia, professional industries, Internet of Things, or businesses.

For some individuals, it appears obvious that the information flows everywhere, but most people do not give it a second thought, and the everywhere-processed

information is becoming a more and more valued asset. In fact, the data is now a resource like any other – and resources are to be exploited.

The data breaches that we constantly observe [73] are typically huge in their nature as they concern the world's largest organisations. Even hundreds of millions of users' data can be compromised in one data breach – and consider the data breaches that we are not aware of. The origin usually involves a hacker gaining access, employees being tricked to unconsciously disclose it, or the software having critical vulnerabilities[1].

Nevertheless, the loudly commented data breaches are effective at raising the concerns of information privacy in society. The authorities react slowly when regulating the matter of privacy but eventually they regulate it. The regulation that was recently implemented was the General Data Protection Regulation and is already well recognised. It applies to organisations processing the data of European citizens and residents [86]. The organisation itself does not necessarily have to be European to be obliged to comply with this regulation, hence making this a global matter.

Software development craftsmanship is not a regulated profession yet – but soon enough it might just be [51]. We certainly are not yet being told how to write our modules and classes, however, we are now being told how to store the information and how to protect it with techniques like anonymisation. Perhaps the implementation of regulations protecting digital information is the introductory step towards the direction of formalizing our profession.

The stricter regulations that we must now be compliant with safeguard data privacy by enforcing the usage of refined data protection methods to preserve privacy. Although anticipated, the relevance of data protection has suddenly and significantly increased. The needs for improved privacy-preserving methods arise with the increasing privacy requirements.

Data anonymisation is the preferred way for achieving privacy in information as it is the most powerful data protection measure to render the data de-identified. At the same time, anonymisation is the cutting-edge method whose relevancy is only to be increased with the increasing privacy concerns. However, as every data

[1]Which was the case during the Facebook breach of 2018 that affected at least 50 million of users and involved exploiting the "View as" browsers functionality to steal the accounts.

and every context surrounding the data is unique, there is – unfortunately – no general method that needs to be performed to render the data anonymous. At least not unless we prune all the data.

Instead, an extensive context assessment needs to be conducted to establish the data masking techniques that need to be joined to achieve the anonymisation of the previously identifiable information. The context that must be evaluated includes at least answering the questions of *"What is the data?"*, *"Who will use the data?"*, and *"How will the data be used?"*. Only upon understanding this context the data controllers can start their preparation for anonymisation.

The fundamental objective of this thesis is to design, implement and document an innovative solution that will meet the versatile needs of data controllers. The software should be enriched with a variety of data masking techniques to offer generic context-flexible anonymisation capabilities. The controllers should be able to customise the anonymisation in a way optimally fitting the exact business needs that they are aware of. For this reason, the diversified perturbative and non-perturbative data masking capabilities are engineered – the techniques include suppression, generalisation, perturbation, randomisation, tokenisation, pattern masking, hashing, shortening, and artificial data substitution.

These functionalities should be delivered as a modular and containerized web platform. The scope of the thesis also partially involves researching this modern civilization problem to understand the requirements and possibilities.

This chapter lays the motivation for this problem. The second chapter goes into the details of analysing the root cause for the problem to exist. It examines the privacy significance and the regulations covering it. Moreover, the chapter investigates the details of anonymisation by analysing the available data masking techniques, the existing software, and finally establishes the foundation for the web platform. The third chapter makes assumptions about the system and its requirements. The chosen tools and development methodology are explained. The fourth allows the reader to understand how to interact with the platform on a high level. The fifth chapter dives into the vast details of the system architecture and internals. Chapter sixth concerns the quality assurance for the platform. The final chapter summarizes the achieved results and future expansions.

Chapter 2

Problem analysis

An extensive analysis of data anonymisation subject is required to be conducted to comprehend the growing data protection needs of the society and to engineer a solution that will properly suit those needs.

2.1 Data explosion

2.1.1 Technology advancement

The continuous and rapid exponential growth of data being collected globally is further excited by improvements to the overall population's accessibility to digital technology [10]. Cisco Systems estimates within its annual report [15] that 66 percent of the world population will have access to the web by 2023, compared to 51 percent in 2018, whereas the number of devices that are connected to the web will reach a staggering value of three times as many as the entire population size – demonstrating a total of 60 percent expansion when compared to 2018. Even the area of mobile connection, which was established long ago, is still sustaining growth – by 2023, mobile connectivity will be a privilege for 70 percent of the world's population, compared to 66 percent in 2018. The global average mobile network speeds will be tripled through a rapid increase from 13.2 Mbps in 2018 to 43.9 Mbps in 2023.

2.1.2 Data analytics

The raw representation form of the data is not interpretable until it is put under a context and processed into practical information. Acquiring relevant insights and conclusions from the information can be achieved through a wide term of analytics, which encompasses the actions needed to be performed to produce new information, including analysis, filtering, processing, grouping, and contextualizing the information. Newly discovered knowledge is inferred from the produced information. Apart from the processes, analytics also includes the technologies, methodologies, and algorithms to use and could be divided into descriptive analytics, diagnostic analytics, prescriptive analytics, and predictive analytics [10].

2.1.3 Big data

Big data analytics deals with the difficulties of managing the observed exponentially increasing collected volumes of data. Its purpose is not only to handle the processing and analysis of the data through specialized software tools and frameworks but also to handle the means on how this enormous amount of data is collected and stored in the first place. It is in its nature that big data is all about massive volumes of information that require specialized hardware infrastructure to store it [10].

Services of enterprise organizations are running on all the collected data which can take various forms such as database entries, metrics, logs, or outgoing messages. New data streaming technologies working at a large scale needed to be engineered to handle the continuous flow of data between systems and databases. An example of such technology includes Apache Kafka which generates even more than a trillion of messages per day for individual large enterprise organizations taking advantage of it [36, 58].

This only proves that big data deals not only with massive volumes – it has also to deal with the high velocities of data generation, which is yet another characteristic of data [10]. According to DOMO report published back in 2020, 90% of the world's data was generated just in the preceding two years, and on average every person in the world created 1.7 megabytes of data per second – which yields 2.5 quintillions ($2.5 \cdot 10^{18}$) bytes of new data each day [28].

The momentum of the immense big data interest growth among organizations is not fading away yet, as more and more new businesses and researchers are drawn to this subject. The benefits of big data especially concern scientific organizations and large enterprises of which the financial domain and IT industry are the common consumers [31]. Organizations find interest in information analytics for remarkably diversified reasons. It is recognized as a field that will entirely alter all parts of civilization such as businesses or society as a whole [48].

Applications

Various types of organizations collect data to take advantage of the insights derived out of the data. The big data analytics applications impact can be observed already today in a broad spectrum of domains.

Leading technology companies, such as Google and Facebook, to name a few, sell anonymised collected user data access to their partner advertisers [48]. This is legally possible as the information that was anonymised, i.e., de-identified in a way that it is no longer bound to an individual, may flow from one system to a system.

The Large Hadron Collider (LHC) located in CERN, being the largest physical experiment, annually produces approximately 30 petabytes of data. LHC takes advantage of light sensors that monitor the collisions of hundreds of millions of particles accelerated nearly to the speed of light. The collisions create an enormous amount of data to be processed by computer algorithms in the hope of discovering new particles, e.g., a Nobel prize-awarding discovery of the Higgs boson had taken place in 2012 [1].

Enterprise stores such as Amazon or SAP Commerce Cloud collect information regarding the way how the visitors browse and interact with these stores. Collected information may involve behavioural data related to customer engagement, such as the pages we visit, event clicks, or the way we scroll the page. The insights derived from the collected information enable making future improvements of these services – for example by improving the digital marketing or performance improvements based on the metrics [43]. The customer experience is also improved as based on the collected data the advertisements or item recommendations can be tailored to the

specific user's preferences. The recommendation engine may also attempt to match your profile data to people of similar profiles to provide better recommendations. Services attempt to analyse the behavioural patterns such as time of day we browse the store or what circumstances caused our last visit to finish. Even the details such as the exact neighbourhood location we live in, combined with its estimated wealth, organizations may attempt to guess our potential income level [48]. These data analytics are performed to improve the possibility of customers buying yet another item.

2.2 Data privacy

Privacy endangerment is inherent to big data and it is its major drawback. The personal data we continuously give away to third parties is the big data fuel.

As technology evolves, concerns relating to the privacy of our personal information should also grow – and for a relevant reason. We tend not to give a second thought to whom the data is shared, how it may be used, and in what kind of circumstances. We don't wonder how our data may be exploited to alter our thinking, decisions, or even ideas – whether in an ethical manner or not.

Although the use of our personal information existed ever since the very first census was created, data privacy is a relatively new concept, as it did not exist prior to the global adoption of the internet. Granted that our data was used by the researchers even before the digitalized era of the internet, the motives for that usage were not commercial [73]. Nowadays, the data has most certainly become an asset – a resource like any others, and a rather precious one.

It is argued that data privacy should be centered only around data usage that has the potential to be a privacy breach. On the other hand, it is argued that merely a collection of the data is already privacy harm. The information that was collected is endangered by many threats, including data misuse, data breach, data leak, or even authorities access without legal obligations. Anonymisation is the best method to mitigate conflicts raised by big data with respect to data privacy and data protection [77].

Luckily the law had finally caught up to the circumstances of the increasing data usage and the associated risks. New European regulations were adopted in

2018 in the form of the General Data Protection Regulation (GDPR) to protect our data privacy in a refined fashion. The data subjects, i.e., the individuals represented by the data [8], now have better control over their personal information – we are now entitled to know what information concerning us is being processed and for what purpose. We are also entitled to withdraw at any time the consent for the processing of our data. In case of violations, we have the right to complain to authorities and seek justice against both the data controllers, i.e., the entities that determine the intention and means of processing the personal information, and the data processors, i.e., the entities that process the personal information on behalf of the data controllers [81].

Overall awareness of the data privacy significance had improved in the society, and the means to achieve data privacy through data protection had also improved as organizations needed to adapt to the new situation by applying enhanced measures to their protection of data – anonymisation and pseudonymisation being the notable examples of such measures.

2.2.1 Authorization to share data

Majority of data privacy regulations are based on a consent of an individual, i.e., it is lawful to process and use the information for secondary purposes only if an individual explicitly acknowledge their consent for that [84]. This may appear easier said than done due to the unobvious difficulties data controllers face when trying to obtain such a broad authorization consent that will take into account all possible secondary purpose usages.

Consider a patient entering medical facility for an ordinary appointment. The patient would likely find it unusual, disturbing or even shocking if upon his entrance to the facility he was to receive an overwhelming form that included dozens of independent consent authorization requests. The consents could give the impression of being seemingly unrelated to his visit in the first place, e.g., a constent to share the data with researchers of an university located on another continent. In the end that could destroy the data subject's trust – in this case patient's trust.

This theoretical scenario may not easily be implemented in the real world counterpart, as it could be even impossible to know or predict all possible secondary

purpose usages in the first place, and consent-based authorization is all about knowing the usages.

Consider a newly discovered purpose to process personal information of an already existing database. Getting consent after the data had already been collected, i.e., backward in time, would be impossible to accomplish as the data controller would need to contact potentially hundreds of thousands of people for their explicit consent. New purposes can be discovered years after the data collection.

Having that in mind, no consent is required when processing the data that is already anonymised. Data that was stripped from personal identifying or identifiable information data can be used in any way and can be shared with third parties without previously agreed consent. Data controllers now face a realistic to solve the problem of information anonymisation rather than an unrealistic problem of consents collection [29].

It is worth mentioning that there exist cases which are defined under Article 6 of General Data Protection Regulation (GDPR) [22] when consent is not required to process the data, e.g., if the processing is required to defend the data subject's interests or in order not to break the compliance with legal obligations as a data controller. GDPR is a new European law that replaces the preceding Data Protection Directive regulation adopted by European Community in 1995 [84].

2.2.2 General Data Protection Regulation

One of the primary objectives of GDPR is personal data privacy protection which is a fundamental right and freedom of people as defined under the Recital 1 of GDPR [18] and the Charter of Fundamental Rights of the European Union [17]. Newly discovered challenges for the protection of personal data arising from the ongoing globalization and quick development of digital technologies. This in turn vastly had increased both the scope of the gathering of the data and the sharing of thereof. General Data Protection Regulation (GDPR) is a data protection law that came into force on May 25, 2018, to addresses these data privacy-related issues in a strict manner [20].

Compliance with GDPR law is critical for organizations in given the significant administrative fines they face. Violations of the data processor and data controller

obligations defined in GPDR are subject to costly penalties that are imposed by European authorities. Non-compliance with technical rules implies a penalty of 2 percent of the total annual turnover of the previous year, or €10 million, whichever one is higher, whereas non-compliance with basic data protection principles imply an even higher penalty of 4 percent of the total annual turnover of the previous year, or €20 million, whichever one is higher [21][54].

Implementation of this law had immediately increased the significance of data anonymisation as an information sanitization process in today's world [59]. Anonymisation is a specific form of data masking that suddenly became more relevant in today's world for the reason that the strict regulations, and therefore administrative fines, defined in GDPR do not apply to the anonymised information. Data protection principles covered throughout GDPR concern only the processing of information that is already identified to a natural person, or that is identifiable to a natural person, i.e., an individual is yet to be identified. Given the fact that anonymised information is by definition not relating to a person, hence it can be completely exempted from falling under GDPR requirements, which apply only to personal data, as stated under Recital 26 [19]:

> The principles of data protection should apply to any information concerning an identified or identifiable natural person. [...] The principles of data protection should therefore not apply to anonymous information, namely information which does not relate to an identified or identifiable natural person or to personal data rendered anonymous in such a manner that the data subject is not or no longer identifiable. This Regulation does not therefore concern the processing of such anonymous information, including for statistical or research purposes.

GDPR distinguishes personal data, anonymised data, and pseudonymised data as distinct variations of data. The information that had gone merely through the pseudonymisation process would still fall under the regulations of GDPR, due to the existing relevant possibility of future re-identification of the data subject, whereas in the case of the anonymised data, such re-identification is by definition either impossible or extremely impractical, and the anonymisation is irreversible by definition. Anonymised data is completely exempted from being governed by

GDPR. Nevertheless, pseudonymisation is still one of many possibilities for the data controllers and data processors to be GDPR compliant [84]. The point of anonymisation in the context of GDPR is to be completely exempted from being governed by this regulation.

2.3 Data anonymisation

The data released by organizations exclude identity-related information such as names, addresses, telephone numbers, or credit card numbers. Personal data is stripped from disseminated data sets through anonymisation to protect the anonymity of the individuals, i.e., data subjects [74].

2.3.1 Background

Disseminating the medical data volumes is crucial for the evolution of the world's healthcare services. Consider a collection of medical data concerning patients' clinical information. Medical researchers and doctors take an advantage of the collected data sets to improve their comprehension of diseases and explore new possibilities to treat these diseases, and hence both the overall capability to treat the diseases and the general efficiency of health services are improved. At last, it is the patients who benefit from the research conducted on their data since the services they are offered continuously improve. Nevertheless, it is known that medical data is exceptionally sensitive by its nature due to the details that include e.g., patient data, laboratory tests results, diagnosis details, prescribed medications, and history of diseases [40].

Having understood how sensitive by its essence is the patient information and the vital needs to share this data, this is where data anonymisation plays an indeed crucial role. It would be impossible to disseminate patient information without prior anonymisation of thereof.

2.3.2 Definition

Anonymisation is a statistical disclosure control method of particular importance. The ultimate anonymisation goal is to de-identify the data by pruning the personal information in such a way that the relationship between the data subject and corresponding records is blurred.

The data anonymisation is considered to be an effective one only if both of the following criteria hold true [42]:

- performed anonymisation operations are irreversible,
- data subject re-identification is impossible or impractical.

2.3.3 Data classification

The prerequisite for anonymisation is to understand the context of what does the data represent and by whom – in addition to how – will it be processed [29].

The commonly adopted approach when attempting to anonymise the data is to first classify the data attributes as direct identifiers (i.e., personal identifying confidential attributes unique to an individual, such as address or tax identification number), indirect identifiers (i.e., so-called quasi-identifiers that when combined can discover the individual's identity – such as sex or date of birth) and non-identifiers [27, 29].

It is context-dependent to determine which attributes are quasi-identifiers – and finally which techniques need to be combined to achieve anonymisation. Virtually, even the most unexpected attributes could be quasi-identifiers. Ignoring such fields when preparing for anonymisation may eventually lead to future re-identification of the data subjects, as successfully demonstrated, e.g., when breaking the anonymity of the Netflix Prize dataset back in 2006 [7]. Since then significant advancements in the field of de-identification were yielded – and in various domains [9].

On the other hand, too much anonymisation of quasi-identifiers strips the data out of utility.

2.3.4 Utility vs privacy trade-off

The organizations need to compromise between the utility and privacy when releasing the data. As anonymisation level increases when more and more restrictive techniques are applied, the utility of the same data decreases.

Consider the two theoretical boundary cases of either releasing data in its original form which maximizes the utility at the cost of discarding the protection of the data subjects' privacy altogether, and the second case of not releasing any data at all which completely preserves the privacy [88].

Clearly the data must not be released in its original state – strict GDPR that imposes large administrative fines [21] exists to mitigate such violations. On the other hand, data that completely suppresses all the information has no value. The data controllers therefore need to strike a balance between the two cases. An appropriate anonymisation strategy maximizing the utility without endangering the privacy must be adopted.

For this reason the data controllers must select appropriate data masking techniques.

2.4 Data masking techniques

The analysis of the domain concludes that the designed system needs to be enriched with diverse masking techniques to meet the fundamental requirement of offering a generic context flexible anonymisation tool. This is achievable on the account of the breadth of the available data masking techniques being genuinely remarkable. The methods that are supported by the anonymisation platform include suppression, generalisation, perturbation, pattern masking, hashing, attribute randomisation, record randomisation, substitution, tokenisation, shortening, or random number generation. All of the available methods are subject to customisations.

An individual method can be classified either as a non-perturbative method if it reduces the data detail by removing some of the information (e.g., suppression, generalisation, pattern masking or shortening) or as a perturbative method if it alters the data by creating some level of uncertainty (e.g. by adding noise through

perturbation or randomisation) [40]. Perturbative techniques are the best for maintaining a high analytical value of the data as they preserve the data truthfulness, while non-perturbative methods are the best for protecting privacy as they prune the information.

It is not possible to predetermine – in a general fashion and without an extra context – the data masking techniques that need to be combined to achieve actual anonymisation of the previously identifiable information. Instead, a context consisting of:

- exact representation form of the data being processed *(What is the data?)*
- data processors who use the data *(Who will use the data?)*
- processing purpose, e.g., research objective *(How will the data be used?)*

is always required when considering an effective way to anonymise data under that context [29].

2.4.1 Suppression

Suppression is the strongest anonymisation technique which completely removes all the values associated with the given attribute, hence rendering the complete protection of the data and therefore enforcing the best data subjects privacy. It is guaranteed that no further implication attacks can be performed against this data [74]. Nevertheless, suppression being the strongest privacy-preserving method implies the highest (i.e., worst) data utility loss as an inseparable consequence – the ultimate cost of complete anonymisation that is to be paid.

Consider the raw data of the three attributes presented in Tab. 2.1 to be put under the process of suppression.

Exemplary results of performed suppression are depicted in Tab. 2.2. All of the three attributes values were suppressed to the value of the suppression token supplied by the data controller.

Phone number suppression deserves a brief discussion. Consider a context in which an attacker successfully obtained the information of exactly one individual data subject of his interest. Consider that the attacker decided to call the anonymised phone number. This number could be suppressed in a way to specifically

Table 2.1: Suppression – input data.

Sex	PIN codes	Phone number
F	3248	1212 – 345345
F	8090	4000 – 303030
M	1337	5191 – 915100

Table 2.2: Suppression – masked data.

Sex	PIN codes	Phone number
F/M	####	3000 – 123123
F/M	####	3000 – 123123
F/M	####	3000 – 123123

protect against this attack and the attacker could call someone intentionally most definitely did not, i.e., data protection officer or authorities. In this scenario, it could be trivial to trace the call and find the potential attacker. As demonstrated, suppression could be used in unobvious ways.

Suppression is typically applied on the column-level of an attribute, however, it is also applicable to the individual records. When compared to plain generalisation, suppression yields a higher information loss and can be thought of as a particular case of generalisation [40].

The designed software offers column level suppression with a user-specified value (i.e., suppression token). The values are transformed to this value using a simple algorithm shown in Fig. 2.1.

```
1 procedure suppression(values, token)
2   for (value in values)
3     value := token
4   return values
```

Figure 2.1: Pseudocode – suppression.

Table 2.3: Generalisation – input data.

Age	Salary	Location
27	36 000	Poland
52	54 000	Canada
30	180 000	Poland
68	128 000	Switzerland

Table 2.4: Generalised – masked data.

Age	Salary	Location
26 – 30	1 – 60000	Europe
51 – 55	1 – 60000	North America
26 – 30	120001 – 180000	Europe
66 – 70	120001 – 180000	Europe

2.4.2 Generalisation

Generalisation is the second most common non-perturbative anonymisation technique [27]. This technique processes the raw data by aggregating and substituting the initial values of a given attribute to the values that are more general [6]. The process of generalisation constitutes a conversion of any value to a more general scope.

The proposed software solution includes two generalisation strategies, namely:

- based on the size of distributions,
- based on the number of distributions.

Consider the raw data of the three attributes presented in Tab. 2.3 to be put under the process of generalisation.

Results shown in Tab. 2.4 include exemplary possible values after being processed by the generalisation method. As depicted in the table, all three attributes have been grouped into broader value ranges, hence undergoing the generalisation, yet every column was generalised with a different generalisation strategy.

The age attribute was generalised with a strategy based on the size of distribution. In this case, the size of distribution was defined as 5, as every interval

aggregates five distinct increasing integer values.

On the other hand, salary data was generalised with a strategy based on the number of distributions – the values were aggregated to three even distributions.

Finally, the generalisation does not necessarily concern only numerical values as observed in the generalisation of location attribute. To better understand this type of generalisation, consider that the age attribute could be generalised into the values of e.g., *young adult, adult, senior*. This type of generalisation is not supported by the developed software, however, an illustration of the existence of such a method is necessary. The fundamental assumption of the developed software is to be generic, i.e., data context-agnostic, and this generalisation method is too specific in its essence to be implemented generically.

Both supported generalisation strategies were further enhanced with customizations. The data processor is allowed to predefine minimum and maximum boundary values for the generalised intervals. The computed starting value for the distribution concerning the lowest interval is either minimum raw value or user predefined minimum value, whichever is lower. The maximal value for the highest interval is computed in an analogical way.

The generalisation strategies can be summarized with the pseudocodes shown in Fig. 2.2 and Fig. 2.3.

Generalisation may be applied on a global level or local level [74]. In terms of database-related vocabulary, as this paper primarily concerns an anonymisation of databases, we say generalisation on a column or record level, respectively.

2.4.3 Perturbation

Perturbation is a data masking technique that encompasses the process of swapping the original values with artificial ones in a way that the statistical factors of the data are similar to the initial data. Perturbation is typically achieved by adding noise to the information so that the values are slightly different [35]. Consider weight and height data shown in Tab. 2.5a. Tab. 2.5b shows the data after it was perturbed using the two different perturbation methods that are supported by the engineered software. The supported methods include adding the noise based on the strategies of:

```
procedure generalise(values, configuration)
  computedMin := min(values.min(), configuration.userMin())
  computedMax := max(values.max(), configuration.userMax())
  distributionSize := configuration.distributionSize()

  // Phase 1: Generate empty intervals.
  intervals := <Interval, Values>
  i := computedMin
  while (i < computedMax)
    interval := asEmptyInterval(i, i + distributionSize)
    insert interval into intervals
    i += distributionSize

  // Phase 2: Populate intervals.
  for (value in values)
    for (interval in intervals)
      if (value is within interval)
        interval := get interval from intervals
        add value to interval
  return intervals
```

Figure 2.2: Pseudocode – generalisation based on distribution size.

- fixed value,
- percentage value.

Height attribute values could have been perturbed using the former (i.e., fixed noise of ± 3), whereas weight attribute values could have been perturbed using the latter (i.e., percentage value of $\pm 5\%$). In either case, the linkage between individual records is blurred with this technique.

Similarly to generalisation, both perturbation techniques allow a user to explicitly specify the minimum and maximum accepted boundary values. Perturbation

Table 2.5: Perturbation.

(a) Input data.

Height	Weight
166	58
170	66
194	91

(b) Masked data.

Height	Weight
169	57
168	67
193	91

```
procedure generalise(values, configuration)
  computedMin := min(values.min(), configuration.userMin())
  computedMax := max(values.max(), configuration.userMax())
  numberOfDistributions = configuration.numberOfDistributions()

  // Phase 1: Generate empty intervals.
  intervals := <Interval, Values>
  i := computedMin

  distributionSize := (computedMax - computedMin) / numberOfDistributions
  j := 0
  while (j < numberOfDistributions)
    interval := asEmptyInterval(i, i + distributionSize)
    insert interval into intervals
    i += distributionSize
    j += 1

  // Phase 2: Populate intervals.
  for (value in values)
    for (interval in intervals)
      if (value is within interval)
        interval := get interval from intervals
        add value to interval
  return intervals
```

Figure 2.3: Pseudocode – generalisation based on number of distributions.

technique works the best with continuous values – it is an effective method for de-identification of quasi-identifiers such as dates and numbers [75]. Scientifically advanced perturbation approaches are further divided into linear and non-linear perturbation models [70].

2.4.4 Pattern masking

Pattern masking is a data distortion technique designed to be used when only some parts of the data should be masked. This technique is typically used to mask e.g., codes of various forms, phone numbers, credit card numbers, and other structured data. Consider the data shown in Tab. 2.6 to be masked using the patterns presented in Tab. 2.7. Exemplary masked output is visible in Tab. 2.8. Analysis of pattern masking possibilities has yielded the need to create a versatile pattern configuration. This technique is therefore highly configurable, i.e., available

```
procedure perturbation(values, noise)
  for (value in values)
    random := pick random from [value - noise, value + noise] interval
    value := random
    value := fit value within boundaries
  return values
```

Figure 2.4: Pseudocode – perturbation based on fixed noise.

```
procedure perturbation(values, noise)
  for (value in values)
    coeff := pick random from [1 - noise, 1 + noise] interval
    value := value * coeff
    value := fit value within boundaries
  return values
```

Figure 2.5: Pseudocode – perturbation based on percentage noise.

Table 2.6: Pattern masking – input data.

PIN code	Software version	Product code
54850185	2.7.1	BAR/service/1
03013844	2.4.0-rc.3	FOO/service/7
76590209	1.0.1-alpha	QUX/utility/0

Table 2.7: Pattern masking – exemplary patterns.

PIN code	Software version	Product code
OOXXXXXO	OOXOX	UUUOOOOOOOOON

Table 2.8: Pattern masking – masked data.

PIN code	Software version	Product code
54#####5	2.0.0	RAN/service/0
03#####4	2.7.1	DOM/service/7
76#####9	0.1.9	IZE/utility/1

Table 2.9: Hashing – input data.

Server logs
185.184.2.198 — 200 POST: /api/v1/auth/refresh-token
185.184.2.198 — 200 POST: /api/v1/worksheets
255.7.141.233 — 200 POST: /api/v1/outcomes/generate

Table 2.10: Hashing – masked data.

Server logs
b27ffd54e5b05a538f333157363f18df0a2aaae5754dfd9ec9daad9cc4ccd7a2
477784538ed600c38f586079a7d5e99aac4af97d1cb322888de54edeb600b14d
2cd3e1912285c765f1746d5b68b1fdbbff6be9460e305acc18a1d9d777d89b5e

pattern tokens include: O – preserve the character, X - mask the character, U – mask with random uppercase letter, L – mask with random lowercase letter, N – mask with random digit, A – mask with random alphabetic character and C – mask with random alphanumeric character. Users can specify the masking character, e.g., #, and can also decide whether the result is truncated to the pattern length (as shown in the masking of software version in Tab. 2.8).

2.4.5 Hashing

Hashing is a deterministic masking technique [59] that transforms data to the fixed-length output. This technique is best used for unstructured data. Available implementations include SHA2 and SHA3. Consider the server logs depicted in Tab. 2.9. Exemplary hashed values are located in Tab. 2.10. Although a mere inspection of the generated hash value does not immediately trace back to the original value, an attack to de-anonymise the data can be easily performed when the attacker knows what the hashed data represents, e.g. a structured birth date [78]. Like in the case of all techniques, the decision to use this technique is context-dependent.

Table 2.11: Randomisation – column shuffle.

(a) Input data.

Identity	Virus
John	Influenza A
Marc	Pneumonia
Stephen	Bronchitis

(b) Masked data.

Identity	Virus
Stephen	Pneumonia
Marc	Influenza A
John	Bronchitis

2.4.6 Randomisation

Randomisation methods offer the best middle ground between maintaining the data utility and preserving privacy [71]. This method can be performed on a column or a row level.

Column shuffle

Column shuffle is an exceptionally effective method for breaking strong links of data belonging to individual data subjects. All of that is achieved while preserving the data utility [73], hence this technique is great for maintaining the analytical value of the data [71]. Consider the medical data from Tab. 2.11a An exemplary randomised data is shown in Tab. 2.11b. This technique can be used in two ways:

- shuffle without repetitions which preserves the data distribution,
- shuffle with repetitions which distorts the data distribution.

Row shuffle

The data may also be needed to be shuffled on a record level – similarly, with or without preserving the distribution. Consider shuffling the characters of an RGB colour described in hexadecimal format. Furthermore, consider a concept of storing a set of abstract decisions (e.g., willingness to participate in the election voting) as a sequence of bits where each bit represents an individual boolean decision. Tab. 2.12a shows the data. Table 2.12b shows the data after the masking process. The colours were processed without preserving the original characters distribution.

Table 2.12: Randomisation – row shuffle.

(a) Input data.

Hex triplet	Set of decisions
FF00FF	1101
54E7CD	1010
E5E5E5	0000

(b) Masked data.

Hex triplet	Set of decisions
0000F0	0111
E57DC4	1100
5E5555	0000

Table 2.13: Substitution.

(a) Input data.

Name	Surname
Jan	Gold
Bob	Ng
Bob	Xi
Maria	Robin

(b) Masked data.

Name	Surname
Lucius	Lucci
Decimus	Rector
Decimus	Lucci
Amanda	Rector

2.4.7 Artificial data

Replacing the original data with an artificial data can be achieved through substitution, tokenisation or random number generation.

Substitution

Substitution is of particular relevance when the masked data needs to be realistic [65]. Consider the data from Tab. 2.13a undergoing the substitution process. Consider that the data controller provides the following data for the name parameter: *Lucius*, *Decimus*, *Amanda*, and the following data for the surname parameter: *Lucci*, *Rector*. The table 2.13b presents an exemplary possible masked result. The name attribute is masked without memorylessness, and the masking occurs in a circular manner. The method is designed to be used with or without memorylessness in mind. Duplicate entries of the same unmasked data values may always be altered to the same masked data values. This technique is also a great extension point for data controller customisations, as the external data set needs to be provided, therefore the data controller can have full control over the data.

Table 2.14: Tokenisation.

(a) Input data.

Survey response
Agree
Not sure
Agree
Strongly disagree

(b) Masked data.

Survey response
1
2
1
3

Table 2.15: Random number.

(a) Input data.

Customer satisfaction
2
1
3

(b) Masked data.

Customer satisfaction
5
5
4

Tokenisation

In comparison to the substitution technique, this operation is relevant when the masked data does not need to be realistic [65], i.e., it could be useful to perform the analytics on the frequencies of data duplicates. Consider the data from a survey question shown in Tab. 2.14a. The tokenised data is shown in Tab. 2.14b. The *agree* value was tokenised to the same token value hence conserving the frequency characteristics of the data.

Random number

The fundamental requirement of anonymisation platform is to be a versatile software to anonymise the data, hence even a simple random number technique needs to be supported to possibly cover a broader range of the business needs. Consider the data of customer satisfaction described as a value ranging from 1 to 5 shown in Tab. 2.15a. The values from Tab. 2.15b show the new masked values.

Table 2.16: Shortening.

(a) Input data.

Surname
Kowalski
Kowalewski
Nowak

(b) Masked data.

Surname
Kowal.
Kowal.
Nowak

2.4.8 Shortening

Consider the surnames from Tab. 2.16a to be shortened to a length of 5 characters. The shortened data is visible in Tab. 2.16b. It is possible to specify whether or not each data record should be terminated with a dot.

2.5 Existing solutions

All of the tools have a shared goal of being GDPR compliant in mind. As this section shows, the existing solutions are greatly similar to the engineered software and often offer *less* versatility than the designed anonymisation web platform.

Of course, as data privacy concerns are increasing in society, the need for data protection also increase. The relevance of anonymisation will only increase in the future. The market is just not steady yet.

2.5.1 Open-source software

Anonimatron

Anonimatron [69] is a Java tool designed for anonymisation of databases specifically for development purposes. The idea behind the tool is that the software developers may not be authorized to access the production data, yet may be required to reproduce, e.g., data-related performance problems. Internally, anonimatron uses the concept of synonyms.

Anonymizer

Similarly to Anonimatron – Anonymizer [25] is a tool that anonymises the databases specifically for development. This tool was written in Ruby.

ARX

ARX [63] is a Java based desktop solution offering a significant variety of advanced data masking techniques [64]. The tool offers the functionality of analysing the usefulness of the generated outcomes.

Amnesia

Amnesia [60] is a software system that allows uploading the data, configuring the anonymisation parameters, and downloading anonymised outcomes. However, it supports merely the text files.

2.5.2 Enterprise-class software

SAP Data Services

SAP Data Services is a data management software that internally uses Data Mask component of the Transformation module to provide the masking functionalities [72]. Data Mask component offers highly configurable functionalities like perturbation, generalisation, and pattern masking.

Oracle Data Masking and Subsetting

This is yet another enterprise-class data masking software, however, it is solely focused on masking. This software is optimised for Oracle database, however other databases are also partially supported. The available operations include shuffling, randomisation, pattern masking [23].

Microsoft Azure ecosystem

The anonymisation techniques that are used throughout Azure Synapse Analytics, Azure SQL Managed Instance, or Azure SQL Database include pattern

masking, random number generation, and text substitution [2].

TurboCat.io

This tool primarily oscillates between the encryption method to protect sensitive data [47].

2.6 Anonymisation as a platform

The people who were software engineers back in the 1990s remember how the web had changed everything. The market market was rapidly dominated and the way business systems are now engineered [49].

The companies started to massively migrate their services to a cloud with the quite recent rise of containerization and microservice architecture. Trending software concepts such as serverless or event-storming and tools such as Kubernetes — all of them are oscillating around the web, which is the only clear and technologically available choice to build a new system. All this technological evolvement is enabled by the open-source which is also continuously growing together with the web-based software systems, in a feedback loop manner [12]. The start of the decline in the web's growth is yet to be discovered and likely will not emerge in the foreseeable future.

With that being said, the designed software system is categorized and delivered as Platform as a Service (PaaS) and runs in a cloud. The fundamental and the most significant objective of the system is to design an innovative platform that will offer anonymisation capabilities through a broad variety of highly configurable anonymisation techniques in such a way that the data controllers can fit their unique business needs easily.

Chapter 3

Requirements and tools

3.1 Requirements engineering

3.1.1 Functional requirements – domain

Data masking

There should exist data transforming services that will offer the functionalities of the following techniques:

- suppression,
- generalisation,
- perturbation,
- randomisation,
- tokenisation,
- random number replacement,
- hashing,
- column shuffle,
- row shuffle,

- shortening,
- pattern masking.

There should be a validation standard established and enforced that will determine if a particular anonymisation technique is compatible with another technique. The techniques should be configurable.

Starting template generation

The user should be able to start the generation of a new template based on the database dump file he provides. The acceptable dump files should include a compressed archive or plain script file. The user must successfully upload the dump before starting the template generation.

Template generation progress

The template generation should involve the sequence of three steps: persisting the dump file, restoring a new database from the dump, extracting metadata about the restored database. The user should be able to view the template generation progress.

This should be presented as a progress panel containing a visualization of three steps. Completion of a step either marks it as a success or an error. Currently processing step should be shown. Errors informing about e.g. invalid databases must be reported to the user. The generation progress should internally update status of the template and should be performed as a sequence of events.

Restore database

The system should be able to restore a new database from a dump file of either a compressed archive or script file.

Mirror database

The system should be able to create a mirror of an existing user database. The mirror should be used for writing the anonymisation script into it and then dumping it into the final outcome result.

Extract metadata

The system should extract the metadata from the uploaded database when creating a new template. The metadata must include information concerning the structure of the database, e.g., the tables, columns, and primary keys the database contains. Extracted metadata should be persisted in the server database as JSON type.

Inspecting metadata

The user should be able to inspect the extracted metadata.

Download metadata

The user should be able to download the extracted metadata.

Viewing templates

The user should be able to view his templates.

Producing worksheet

The user should be able to produce a new worksheet from the templates he created earlier on. The worksheet should be used for building up the anonymisation setup. The worksheet must not be accessible to other users.

Viewing worksheets

There should be a table giving an overview of the created worksheets. Only worksheets belonging to the user should be shown. The table should embed buttons to go to the summary view or outcome generation view.

Viewing summary

There should be a summary view which is an overview for the given worksheet. It should contain four distinct sections of information of the worksheet: template section, tables section, operations section, and outcomes section.

Viewing tables

The tables included in the template should be enlisted in a worksheet summary.

Adding column operations

Users must have the possibility of adding new anonymisation operations to the anonymisation concerned columns.

Viewing column operations

The so far accumulated column operations should be enlisted for each individual database table. Different operations should use different colors to provide better visualization. Apart from accumulated operations, the view should also show information such as column name, type, nullability, or whether the column is a primary or foreign key.

PK and FK detection

Primary and foreign keys should be disallowed to undergo anonymisation. An appropriate information should exist to indicate to the user that a given column is a primary or foreign key.

Starting outcome generation

The user should be able to start the generation of an outcome based on the worksheet he prepared. It should be possible to select a compressed archive or script file as a target output for the outcome. Starting the outcome generation must happen in an asynchronous non-blocking way.

On a high level, the outcome generation should include the complex sequential steps of:

- creating a writeable mirror database of the read-only template database,
- creating a new empty anonymisation script,
- populating the anonymisation script based on the user-defined anonymisation setup,

- executing the anonymisation script against the mirror database,
- dumping the anonymised database to the compressed archive or script file.

There should be an outcome database entity created with a status field. Each consecutive step should update this status. The total time it took to process an outcome should be measured.

Browsing outcomes

The user has to have the possibility to browse the outcomes he created in a form of a dedicated table. Each outcome should include the status, anonymisation script download button, outcome dump download button. processing time, template name, etc.

Anonymisation script download

There should be a way to download an anonymisation script.

Anonymised dump download

There should be a way to download an anonymised dump result.

Account creation

To create a new account, an anonymous user needs to provide valid information including e-mail address, password and confirmation, name, surname, and optionally the purpose for anonymisation. Creating an account with an e-mail that is already in use must be prohibited. A new account must be verified to access the broad functionalities.

Account verification

Upon successful account creation, the user must receive an e-mail with a link to verify the account. User role should change from *unverified* to *verified* due to confirmation.

Admin verification

Admin has the means to verify the specified user on his behalf from the users panel.

Verification expiration

The verification link that is sent to the user after creating a new account should contain a token that expires after a set amount of time. The system administrator has to have a possibility of runtime configuration of the expiration time.

User profile

The user should be able to view and edit his profile details. The user must not be able to change his e-mail address.

Users panel

There should be a users panel for the admin to present an overview and manage the users' accounts.

Block user

Admin should have the means to block the user from the users panel, effectively disabling the possibility of logging in.

Unblock user

Admin will have the means to unblock the blocked user from the users panel, restoring the login capabilities.

Expiring accounts

Users that do not login to the platform for a configured amount of time should be deleted due to the account expiration. The user should receive a notification e-mail prompting him to login to prevent the removal action.

Mark account for removal

The user must be able to initiate the process involving the removal of his account along with all the associated data, hence ensuring GDPR compliance.

The process is started by an explicit user's requests to do so. The system should warn the user with a detailed message and a prompt to type in both the confirmation and the password. If successful, the account is marked for future removal – the account is removed by a dedicated cron-based data pruning service.

Forcing account removal

Admin must have the possibility to force the account to be removed in case of urgent needs e.g. due to reasons concerning system security or performance.

Revert marking account for removal

A scenario in which a user changes his mind to delete his account needs to be supported. The user can revoke his request to remove the account by accessing the link that was earlier sent to his e-mail address.

The revert action should be possible only if all three conditions are met:

- the link has not yet expired,
- the admin did not explicitly force the account removal,
- the account has not already been removed by the pruning service.

Revert account removal expiration

The undo removal link sent to the user after requesting his account removal contains a token that expires after a set amount of time. The time of exactly how long is the link valid should be determined by the runtime configurable by the system administrator.

Forgot password

There should be a way to reset the password that was forgotten. The link should be sent to the user's e-mail address and should be valid only for a configured duration.

Tasks panel

There should be a tasks panel for the admin to retrieve and monitor the tasks. The panel should permit a non-blocking on-demand execution of the selected individual tasks.

3.1.2 Functional requirements – infrastructure

Login

The user should be able to login to the system and access his account's resources.

User context

The client should know the details of the user who is logged in. There should exist a user context functionality. Upon successful login the information about the principal user should be fetched and stored on the client's side in the user context.

Access token

A successful authentication should generate an access token that encompasses all the information necessary for user authentication and authorization. The token should be valid for a brief duration, e.g., 5 minutes – which should be a matter of runtime configuration conducted by the system administrator. The token should be implemented using JSON Web Token (JWT) standard and should be stored on the client's side.

Refresh Token

The refresh token which is generated together with the access token during authentication should be used to periodically generate a new pair of the access and refresh tokens. There should exist a REST endpoint that will accept the currently valid refresh token and will output new access and refresh token - effectively prolonging the user's session. This token should be valid for a very long duration, e.g., 6 months – which should be a matter of runtime configuration.

Logout

Discarding the access and refresh tokens permits user logout.

Remember me

The user should not have to login to his account when starting a new visit unless

- the refresh token has expired after e.g., 6 months of inactivity,
- the user has cleared browser's cache,
- the user is logging from a new device.

Restarting the anonymisation platform should not logout the user.

Seamless tokens regeneration

Axios HTTP client should be configured in such a manner that it will detect a state of the expired access token and will seamlessly generate new tokens. When the access token expires, the client should invoke refresh token service to generate new token pairings in such a manner that the user does not even notice this implementation-related behaviour. It is required that no unnecessary page refreshes, errors, logouts, or prompts to login show up to the user.

Role-based authorization

All non-whitelisted REST resources should be protected with a declarative role-based authorization.

Resources whitelist

The system administrator should be able to configure the REST resources that are whitelisted (i.e., unprotected). These endpoints should not require to be authenticated (i.e., logged in) to access them.

Redirection

The platform should redirect the user to the login page when accessing routes that are non existing or forbidden to access for them.

Password encryption

User password should be stored in the database in an ecrypted form using e.g., bcrypt hashing.

Navigation menu

The client application should have a navigation menu. The panel should be located on the left side of the screen. Each navigation item should consist of a large icon and a text label. The entire panel should be hideable – the entire page content should seamlessly adapt to this. Optionally, smooth transition animation should be configurable during TypeScript compilation time.

The items should be easily defined, ordered, and configured by adopting a standard. Visibility of a particular item must be role-based, e.g. only admin can see *users* item.

Timestamp data collection

The system should collect timestamp data about various resources such as users or templates. Some of the data should be client renderable, and some of the data should be merely collected.

Exemplary dates concern:

- account registration, block, request for removal, removal,
- template generation.

Requests data collection

All HTTP requests to the anonymisation server should be logged in the system. The logged information should include HTTP method, HTTP response status, and the request URI.

Request validation – client

The system should disallow sending HTTP requests that fail to meet the validation criteria set with Yup library.

In-depth validation – client

Validation rules set by Yup library for the client forms should be enforced by React Hook Forms.

The client should have the possibility to specify required and optional fields and use custom validation rules such as e-mail or file presence validation.

Display validation errors

The client displays validation errors below concerning HTML inputs.

Error notifications

The frontend view uses toast pop-ups to display client and server HTTP errors [34] – for example, 400 Bad Request. The notifications along with the associated message are displayed in the upper-right corner. The pop-ups are red to underline its purpose and the purpose should be further emphasized - by adding more red color – when displaying the errors of extreme importance e.g. database restore failure.

Success notifications

Client-sent HTTP requests that finished with a successful response [34] – for example, 201 Accepted - should be displayed in a green coloured pop-up along with the associated message.

Request validation – server

The system must validate the HTTP requests on the server to protect the system from passing malformed requests sent by malicious external HTTP clients. The validations should be declaratively used on the Data Transfer Object (DTO) classes.

Exemplary validation constraints could include checking for values that meet any of the following criteria:

- not null,
- not blank,
- not empty,
- for integers – of a value in an allowed range,
- for strings – of size in an allowed range,
- non-standard such as e-mail.

Maximum file size

The system administrator should be in control of the configuration of the maximum accepted sizes of the files sent to the system (i.e. database dumps). The file sizes that are too large must not be accepted.

In-depth validation – server

The server should perform numerous other advanced validations specific to individual scenarios.

Exemplary infrastructure related cases should check if:

- the system is running in cloud environment,
- an entity non compliant with the database schema is trying to be persisted,
- database connection is established.

Exemplary authorization related cases should check if:

- user is trying to access another user's resource,
- user is unauthorized to access the resource due to lack of privileges,
- long lasting refresh token has expired.

Exemplary domain related cases should check if:

- anonymisation operation is applicable the particular column,
- column is a primary or foreign key,
- a given task is manually executable.

Client tables settings

The tables presented used extensively by the client application should offer the functionalities of:

- pagination to a set page size,
- filtering by attribute,
- sorting by attribute,
- hiding and showing of columns.

File selector

There should be a custom file selector input that will allow uploading files i.e., database dumps. The component should show the upload progress bar and success or error message upon completion.

Uniform design

The user must not be surprised with a particular page looking in an unexpected way. There should be established a uniform design for the client pages. The design should uniformise, e.g., colors, borders, paddings, margins, shadows, or components.

Theme

The client application should allow an easy way to change the theme colors of the entire application by changing it in one configuration.

Skeletons and spinners

The client should use skeletons and spinners to provide a seamless user experience.

Mail notifications

The system has the capabilities of sending e-mails notifying about various actions such as expiring accounts. This functionality should be runtime configurable by the system administrator.

Database

The system needs to use a database. The system should not be coupled to a particular database provider.

Database preloader

There should be a functionality to preload the system with prepared data. A new system instance must include at least one administrator account.

Dynamic database connection

There should exist a factory for data sources that will establish a connection to the dynamically created databases restored from dump files uploaded by the users.

Prune connection pool

There should be a functionality to drop all existing database connections, effectively allowing the creation of the database mirrors dynamically.

Containerization

Software components of the system should be containerized. All services should be running as containers when running the system in the production environment.

Multiple environments

The designed system has to be runnable in multiple environments, i.e.:

- production,
- development,
- development with the *server* profile enabled,
- locally on the host with a database container,
- fully locally on the host.

Environment based configuration

Different environments, i.e., production, develop, and local should use different configurations of the environment and anonymisation server.

Storage configuration

The location of the files managed by the system, i.e., uploaded templates, anonymisation scripts and anonymised dumps (scrip based or compressed archive-based), should be configurable.

Running processes from Java

There should be the possibility of running external processes such as *psql* directly from the Java code.

Asynchronous events

There should be a configuration for handling asynchronous events. The core functionalities of *uploading* and *processing* server modules should be initiated with an asynchronous non-blocking event.

Scheduling

There should be a functionality for defining schedulable task services. The services should rely on the cron expression-based triggers. Individual tasks should be runtime configurable in terms of their:

- cron expression,
- whether the task is enabled for automated scheduling,
- whether the task is enabled for manual execution,
- displayed description in the panel.

Secrets

There should be a functionality prepared to load the secrets (i.e., password for the database, JWT secret key, mail service password) from an external file non-published to the public git repository. Continuous Integration related secrets should be configured directly in GitHub.

SonarQube

Both the client and the server should be scanned by the SonarQube static analysis tool. This software should be integrated into the Quality CI workflow.

CodeQL

The application should be scanned by CodeQL for security issues as a part of the Security CI workflow.

Swagger

The REST API of the anonymisation server is to be documented with Swagger UI.

File storage

There should be a way to store the files, i.e., user input dumps, anonymisation scripts, anonymised output dumps (archives, scripts). The files should be stored in grouped directories whose paths are runtime configurable. The file names should be randomly picked e.g. as a universally unique identifier (UUID).

Prune user data

Service responsible for deleting accounts that were marked for removal should be executed in a periodic manner. Such a service should prune the user data. The frequency of how often is the service executed is described as a cron expression. It should be a system administrator runtime configurable property.

Prune system data

There should be a script pruning the Docker data from the host system.

3.1.3 Non-functional requirements

Modular

The anonymisation server should be divided into modular architecture where each module has its own responsibiltiy. Modules have to be further divided into layers or packages. The system implements client-server architecture.

Secure

Achieved through, e.g., vulnarability scanning, token-based authentication enhanced with refresh token functionality, client and server validation, containerization, role-based endpoint protection.

Configurable

The platform should be highly configurable by the system administrator. The software can be separately configured on the server, client and containers side.

Confidential

Data sent by the users should not be accessible by the admins.

Extensible

The system should be engineered in a way to be easily extensible with new features such as supporting new database types or new data masking techniques.

Adaptable

The system should be able to easily adapt to changing requirements by the means of clean code practices such as modularity and testability.

Performant

Achieved through, e.g., using lazy loading of the relations and individual optimisations.

Responsive

Achieved through, e.g., performant client routing.

Engaging

Achieved by displaying skeletons, spinners, and notifications that catch the user's attention.

Usable

The system features should be easily understandable – the functionalities must not be confusing to be used.

Testable

The system code should be written in a testable manner, e.g. by ensuring high quality of the code and separation of concerns.

Reliable

Achieved through continuous integration testing and using well-known secure libraries.

User experience

Achieved through client functionalities such as intuitiveness, responsive design, and uniform styling of the system.

Measurable

The system should collect metrics about e.g. processing time of outcomes.

Modern

The system must not use obsolete technologies or software engineering design techniques.

3.2 System engineering

3.2.1 Use cases

The use cases were split into two distinct diagrams which group the related use cases concerning:

- management – shown in Fig. 3.1,
- anonymisation – shown in Fig. 3.2.

3.2.2 REST API

The communication with the anonymisation server is driven by the RESTful APIs. An overview of the available endpoints is listed in Tab. 3.1. The particular endpoint's purpose can be inferred from the surrounding context i.e., requirements, use cases, controller name, and the endpoint URI.

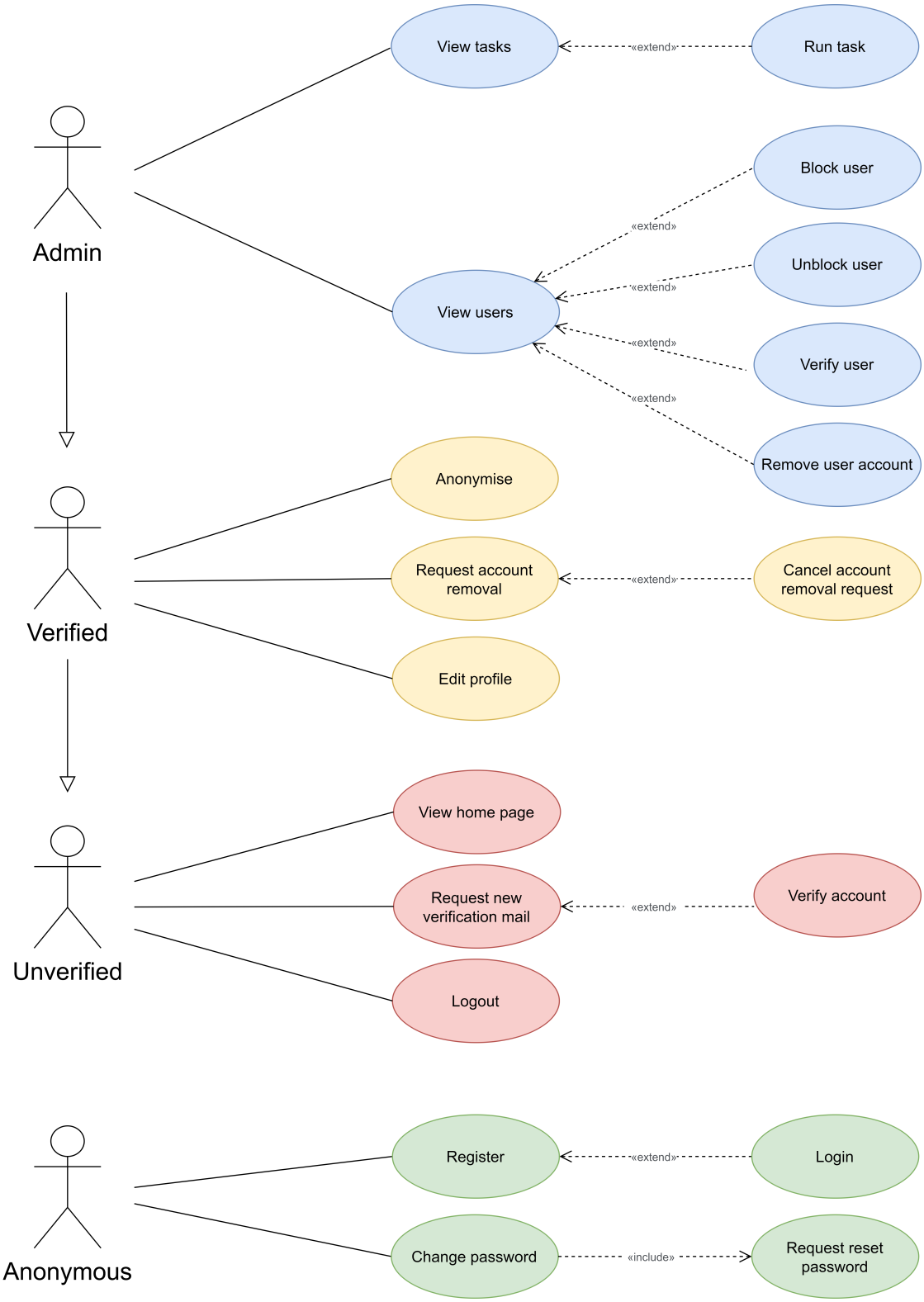

Figure 3.1: Use cases – core domain

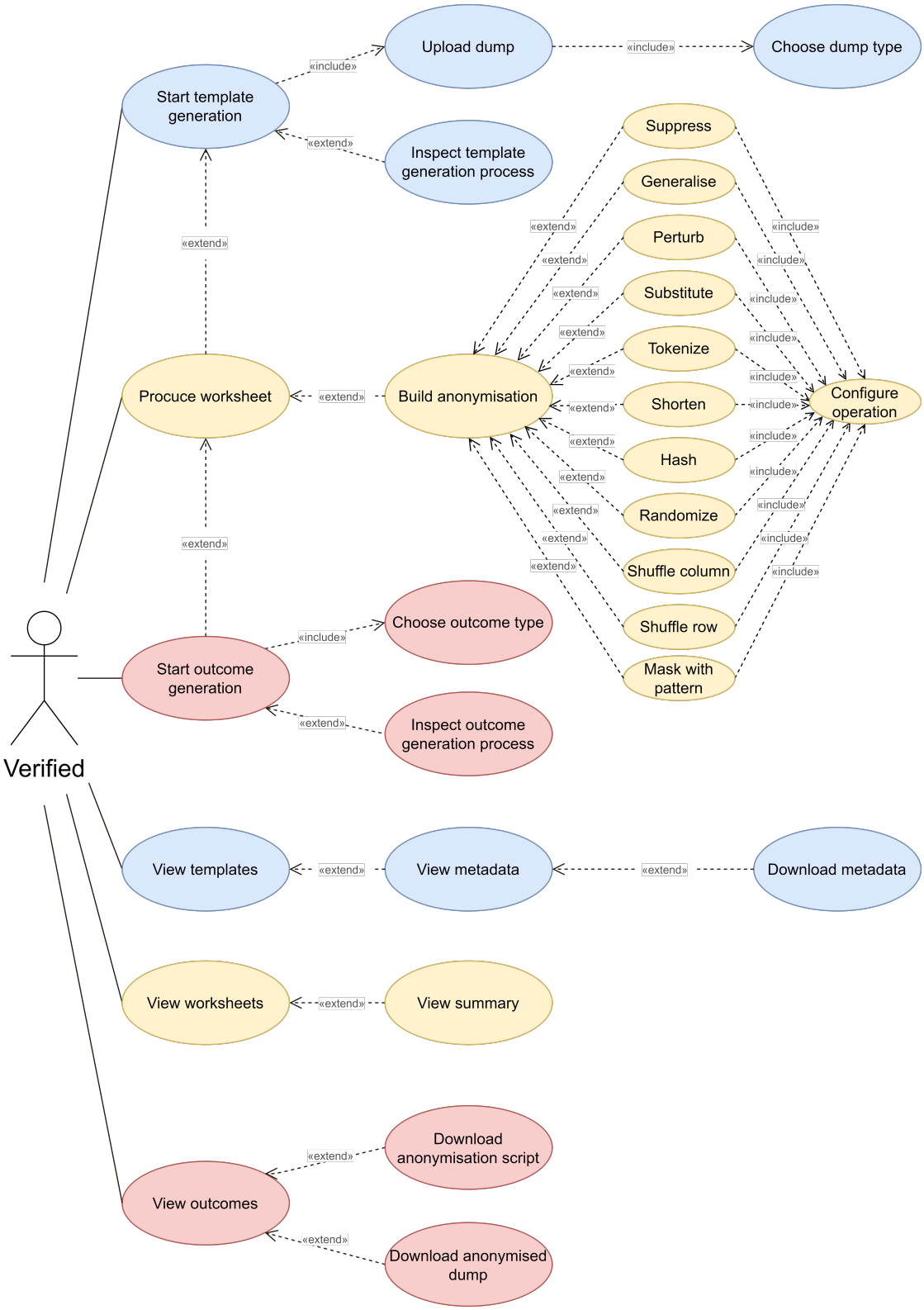

Figure 3.2: Use cases – anonymisation domain

Table 3.1: REST resources – summary.

Controller	Type	Endpoint
Auth	POST	/api/v1/auth/login
	POST	/api/v1/auth/refresh-token
User	GET	/api/v1/users
	POST	/api/v1/users/register
	PUT	/api/v1/users/id/block
	PUT	/api/v1/users/id/unblock
	PUT	/api/v1/users/id/force-removal
Verify	POST	/api/v1/users/verify-mail
	POST	/api/v1/users/verify-mail/send-again
	POST	/api/v1/users/{id}/confirm-mail-verification
ResetPassword	POST	/api/v1/reset-password/request-reset
	POST	/api/v1/reset-password/change-password
	GET	/api/v1/reset-password/show-change-password-form
Task	GET	/api/v1/tasks
	POST	/api/v1/tasks/{task}/execute
Template	POST	/api/v1/templates
	GET	/api/v1/templates/me
	GET	/api/v1/templates/{id}/status
	GET	/api/v1/templates/{id}/metadata
	GET	/api/v1/templates/{id}/dump/download
Worksheet	POST	/api/v1/worksheets
	GET	/api/v1/worksheets/me
	GET	/api/v1/worksheets/me/{id}/summary
TableOperation	GET	/api/v1/worksheet/{id}/table-operations/{table}
ColumnOperation	PUT	/api/v1/worksheet/{id}/column-operations/add-suppression.
	PUT	/api/v1/worksheet/{id}/column-operations/add-generalisati.
	PUT	/api/v1/worksheet/{id}/column-operations/add-perturbation
	PUT	/api/v1/worksheet/{id}/column-operations/add-column-shuff.
	PUT	/api/v1/worksheet/{id}/column-operations/add-hashing
	PUT	/api/v1/worksheet/{id}/column-operations/add-pattern-mas.
	PUT	/api/v1/worksheet/{id}/column-operations/add-random-num
	PUT	/api/v1/worksheet/{id}/column-operations/add-row-shuffle
	PUT	/api/v1/worksheet/{id}/column-operations/add-shortening
	PUT	/api/v1/worksheet/{id}/column-operations/add-substitution
	PUT	/api/v1/worksheet/{id}/column-operations/add-tokenization
Outcome	POST	/api/v1/outcomes/generate
	GET	/api/v1/outcomes/me
	GET	/api/v1/outcomes/{id}/status
	GET	/api/v1/outcomes/{id}/anonymisation-script/download
	GET	/api/v1/outcomes/{id}/dump/download

3.3 Software tools and technologies

The following listings are presented to give an overview of the designed system in terms of tools and technologies involved to engineer it without diving into the implementation details. This is a brief summary that merely presents the technological landscape of the designed system – should a particular technology receive a greater detail of explanation, e.g., exemplary usage, it will be described in another dedicated unit.

3.3.1 Tools

This section encompasses a summary of tools needed to develop, build, and maintain the system.

Docker

Docker is a virtualization software used to develop, run and manage applications in the form of containers taking advantage of Linux kernel features namespaces and cgroups. It consists of three components: a runtime, a daemon engine, and an orchestrator.

To run the production environment of anonymisation platform, only prior installation of Docker on the host system is required, because containerization technologies like Docker enable a separation of the software from the infrastructure the software is running on [26, 61].

Apache Maven

Apache Maven is an open-source build tool that automates the build process of foremostly Java based projects. Primary Maven objectives include ensuring an easy build process and an uniformly established build system which is achieved through Maven's build lifecycle (i.e., predefined build steps, e.g., *validate*, *compile*, *test* or *install*) [37].

Anonymisation platform uses this tool to build the backend server. Maven is put into service in one of the two ways – depending on the selected environment – namely:

- internally from the Docker container where Maven can be considered as an abstraction that the system administrator does not have be aware of,
- on the host system itself.

Maven is also used to run the tests. Gradle could be a great alternative to Maven, the choice being preference based in usual scenarios.

Node.js

Node.js is a Javascript runtime engine encapsulating V8 engine developed by Google which implements ECMAScript [45]. While node.js may function as a dedicated backend server, in the case of anonymisation platform it is used merely to build the client application with yarn.

yarn

Yet another resource negotiator (yarn for short) is a package manager developed by Facebook, Google and others [57], whose primary purpose is – similarly to *npm* – installation and management of dependencies used by Node.js runtime environment based projects. Yarn also focuses on ensuring high level of security, performance and reliability [33].

While the purpose of Maven is to build and run the backend server, yarn's purpose is to build and run the frontend client application. Consequently, yarn is used in the same way as Maven – either internally in Docker or on the host system – depending on the chosen environment.

nginx

Nginx is an open-source that works as a reverse proxy, web server and HTTP load balancer with the first one being the primary function for anonymisation platform. Reverse proxies improve resiliency, security, scalability and performance of the software systems. This is particularly relevant when designing systems that scale horizontally [24].

git

It's been a while since most developers had already started using git for development, which is the leading software for version control, however not everyone knows it was originally invented by Linus Torvalds specifically for Linux kernel development needs [82]. It is clearly necessary to use git during software development due to the essential needs to publish, stash or revert the code changes [14].

GitHub Actions

GitHub Actions is a continuous integration (CI) and continuous delivery (CD) tool that enable creating workflows which automate development processes such as building, testing and deployment of the system.

Developed software uses three distinct workflows to ensure an automation of testing, quality assurance and security resilience.

SonarQube

SonarQube is a large scale open-source tool to conduct the code inspection through static analysis methods. This tool platform can reliably detect vulnerabilities, security hotspots, code smells and bugs.

In the context of designed anonymisation platform, this software is integrated together with Github Actions to ensure quality and security of developed software. Two distinct instances of SonarQube are configured, independently for the client and the server.

ESLint

ESLint is yet another tool for static code analysis for JavaScript applications.

It had been necessary to enhance this tool with TypeScript extension in order to successfully integrate it with client application and Github Actions workflow concerning quality.

Prettier

For an unified code formatting standard to be established for the client's application, prettier tool which is a modern code formatter was configured within JetBrains Webstorm IDE.

3.3.2 Server

This section encompasses a summary of technologies used for the implementation of the modular and REST-driven anonymisation server.

Java

Anonymisation server uses the newest long-term support JDK 17 version released in September 2021, although the server is backward compatible with JDK 11.

Java is still globally the most widely adopted development language [13], even after the 25 years that had passed after its first release – it is a truly mature language, accompanied by all types of frameworks and libraries that one can imagine.

Spring Framework

Spring projects can be viewed together as a modular toolkit that addresses the concerns of modern software development [85].

Spring driven software system may include the following Spring projects [32]:

- Spring Framework - the core supporting e.g., dependency injection and web applications.
- Spring Boot – to achieve an easier setup of the Spring driven system.
- Spring Data – to retrieve information from databases in a consistent, database-agnostic way.
- Spring Security – to secure the application with authentication and authorization.

Java-based systems typically employ Spring technologies, and so does the anonymisation server, which extensively uses all of the listed above projects.

Hibernate

Hibernate – a JPA implementation – which is a well-tested and high performant [53] object-relational mapper was a natural choice for the engineered system given the fact that Spring has built-in support for this library. It is used to map the database entities to the corresponding database tables [67].

Hibernate functionality was additionally enhanced with Hibernate Types library [52] to provide the support for JSON column type – which is not supported by the plain Hibernate but is greatly supported by the underlying PostgreSQL database [41].

PostgreSQL – Server

The choice of an appropriate database was of particular importance due to the domain of the designed system.

PostgreSQL was the fourth most popular database management system worldwide back in 2017 as shown in the annual Stack Overflow Developer Survey report [55]. Each consequtive report demonstrated further growth of PostgreSQL. As PostgreSQL is at the continuous growth, MySQL (i.e., the most popular database) is at the continuous decline. The latest report published in 2021 [56] placed PostgreSQL in second place and it is believed that it will at last beat MySQL this year.

For this reason, PostgreSQL was chosen as the database for the server system. This database was also chosen as the supported database type for the data dumps uploaded by data controllers for the purpose of anonymisation.

PostgreSQL – Client

PostgreSQL client is installed in the containerized anonymisation server to execute *psql* processes throughout the application's runtime, hence providing the communication capabilities with the potentially remote PostgreSQL server.

The client is used by the ZT Processor Executor library – directly from the Java code.

ZT Processor Executor

ZT Process Executor provides capabilities to run external processes directly from the Java code.

This library found many domain-critical applications related to PostgreSQL resources management. The library executes PostgreSQL client processes including:

- creation of a new database,
- replication of an existing,
- database dump restoration,
- anonymisation script execution,
- database dump to script or to archive.

JUnit 5

JUnit 5 is the new and refined version of the worldwide recognized Java testing framework. It is used to drive the unit and integration tests [83].

Testcontainers

Testcontainers is a cutting-edge Java library to support the instantiation of lightweight Docker containers for the tests needs. Effectively – the tests run in a containerized environment just like the real application hence blurring the technical differences between real and test environments [5].

This state-of-the-art library is used by the integration tests to bootstrap the PostgreSQL database for the tests.

REST Assured

REST Assured is an integration tests library and an HTTP client which allows sending true HTTP requests to the test server (bootstrapped by Testcontainers library). This library also offers the means to validate received HTTP responses [3].

The library is used by the integration tests.

Swagger UI

One of many Swagger modules is Swagger UI which deals with the design, description, and documentation of REST APIs. The library allows visualization and interaction with the API [76, 66].

This library is relevant due to the complexity of the designed system, as it provides the API documentation for the client.

JJWT

JSON Web Tokens are required to transport the implemented access and refresh tokens necessary for authentication and authorization purposes. This popular library deals with the creation and validation of the tokens [4].

Bouncy Castle

One of many cryptographic APIs offered by Bouncy Castle library is SHA 3 which internally implements Keccak-f[1600] algorithm [11].

This library is used by the hashing data masking technique.

Apache Commons Lang

The sole purpose of this library is a generation of random alphabetic and alphanumeric characters for the pattern masking anonymisation technique [38].

3.3.3 Client

This section encompasses a summary of technologies used for the implementation of React-based client.

TypeScript

TypeScript is a transcompiled superset of JavaScript that the developers want to work with the most unless already doing so [56]. The language was designed by Microsoft to extend JavaScript with types, interfaces, and generics.

The language is powering the frontend client application and React.

React.js

React.js was the most popular web framework as of 2021 [56] as it has finally surpassed the obsolete jQuery library. This is an open-source library developed by Facebook to build user interfaces for the web. As a high-level overview – React centers around the concept of virtual DOM [87].

React ecosystem

Independent authors create many external libraries to drive React with extended functionalities. React ecosystem is rich in various libraries which typically include *React* word in their names.

Anonymisation client code extensively uses the following popular libraries:

- React Router – to handle the URL related concepts including routing, navigation and query parameters.
- React Query – as an abstraction and improvement layer for fetching the data.
- React Hook Form – to provide user input validation in the forms based on Yup defined rules.

Yup

Yup is a validation schema builder. In the case of engineered software, the schemas created by Yup are consumed by React Hook Form library to validate the forms.

Material UI

Material UI is yet another React library and a large one. This library is responsible for the appearance of the application.

Axios

Axios is an asynchronous HTTP client used for sending requests to the anonymisation server.

3.4 Development methodology

The development was driven by git like in the case of any software project that is of non-trivial size. A total number of 6XX **todo** commits[1] were pushed to the master branch. Conventional Commits specification was followed in a simplified way. The branches were deleted following the merge.

There were three possible issue types:

- **Feature** – describing a new functionality.
- **Improvement** – describing an upgrade to the existing functionality.
- **Epic** – describing a big user story that aggregates other issues. [44]

Note that an improvement to the functionality that is still under development is considered a *feature* and not an *improvement*.

Furthermore, the labels with the increasing values of XS, S, M, L, and XL were being added to all the tasks upon creation to estimate the time-to-finish complexities. The distribution of the estimations is shown in Fig. 3.3. **todo updated pie chart**

The non-epic tasks were marked either as resolved or as canceled when closing the tasks. The vast majority of tasks were successfully resolved as shown in Fig. 3.4. Exemplary reasons to close a given task include changing requirements or task duplication.

[1]The data as of January 21st, 2022.

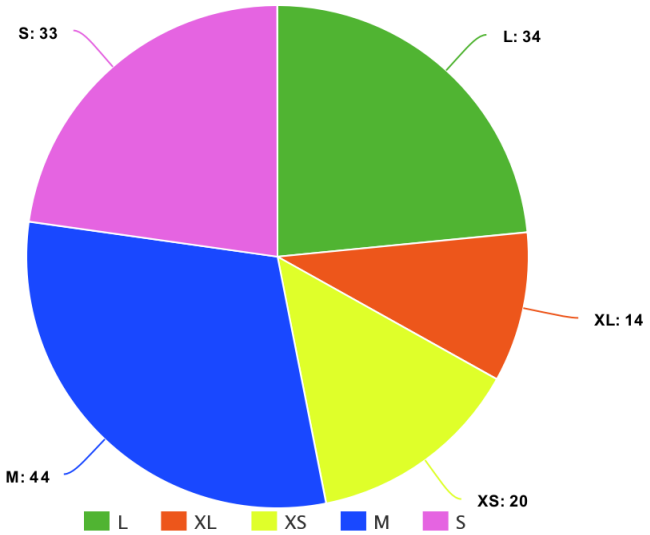

Figure 3.3: GitHub – distribution of tasks estimations.

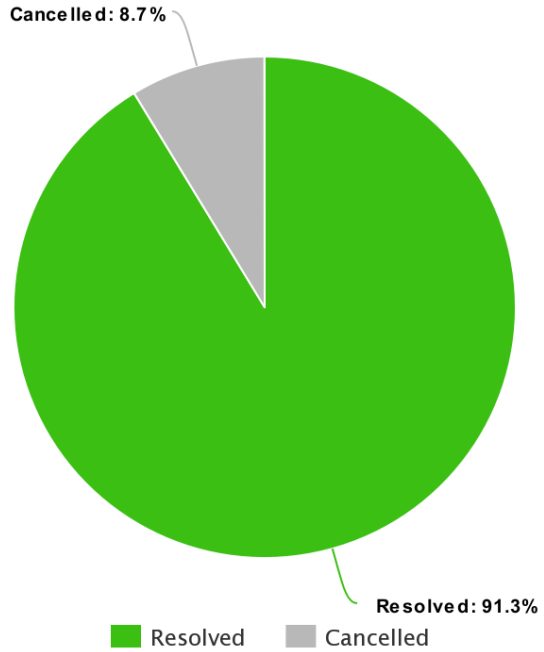

Figure 3.4: GitHub – distribution of resolved tasks

Chapter 4

External specification

4.1 Domain specific glossary

The application defines several business names related to the anonymisation domain to concisely introduce complex technical problems. These names include:

- **Template** – the user restores the database from the dump he provides to create a read-only replicable template that is used to produce worksheets.
- **Metadata** – represents the information about the template database, i.e., information concerning columns, tables, primary keys, foreign keys, number of tables and records, etc.
- **Worksheet** – a concept that represents the user's working area for configuring the anonymisation to match the exact business requirements of his interest. The worksheet is produced from the template and is used to generate the outcomes.
- **Summary** – this is the worksheet's representation – a summary layer of the configured anonymisation that presents in one place all the information concerning the template, worksheet, operations, and outcomes.
- **Operation** – an individual anonymisation technique applied to the given attribute.

- **Outcome** – the outcomes are generated from the worksheet after processing the anonymisation request and encapsulate the anonymisation script and anonymised dumps.

4.2 Installation

One of the primary reasons why the containerization concept was so remarkably well-received [56] and exhibited by the massive global adoptions of the Docker technology is the ease it offers regarding software installation. Decoupling the software from the hardware it runs on allows a portable way to install the software in a simple manner. Instead of making it a responsibility for the end-users to manually install the dependent software (e.g., database server, JDK, Node.js, or yarn), all the dependencies come encapsulated within the image, effectively transferring the difficulties of managing the dependencies to the application developers.

4.2.1 Environments — overview

The software solution is designed to be runnable in multiple deployment environments hence reproducing the deployment characteristics of a simple modern business-class software system.

As depicted in Fig. 4.1 the available environments include:

- production,
- development.

The environments are further divided into three available setup options, i.e.:

- cloud setup,
- semi-cloud setup,
- local setup.

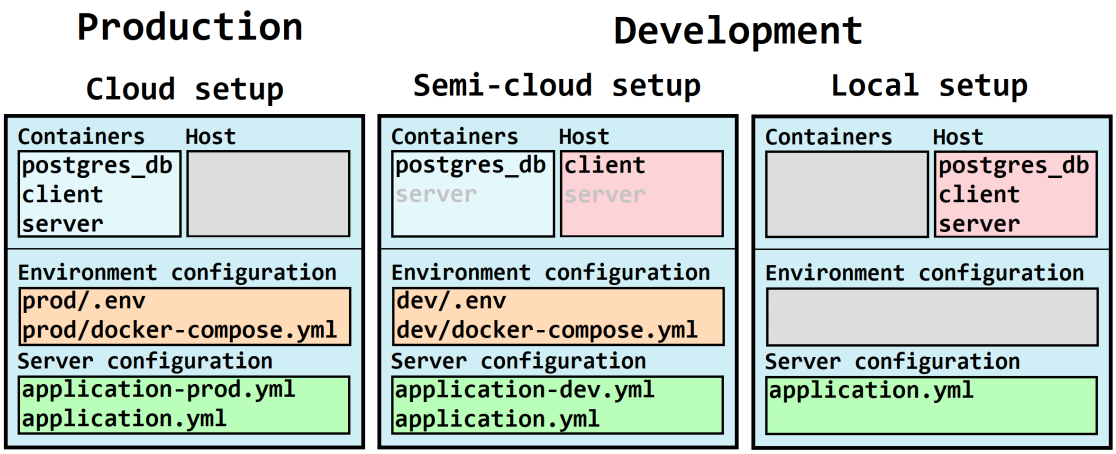

Figure 4.1: Environments

The shared prerequisite for the cloud and semi-cloud setup is to have Docker up and running on the host system. Furthermore, the development environments Local setup does require the installation of additional dependencies.

4.2.2 Cloud setup

The cloud-ready production environment is the easiest one to install.

Prerequisite

Docker up and running.

Installation procedure

Start up everything at once:

```
cd docker/prod
docker compose up
```

If successful the application should be accessible at `http://localhost:3000`.
The setup is customized from:

- `prod/.env`,
- `application.yml`,

- `application-prod.yml`.

4.2.3 Semi-cloud setup

This is the preferred setup for development. PostgreSQL server is abstracted away from the host thanks to this setup. Optionally, the anonymisation server can also run in the container.

The semi-cloud setup requires the installation of additional software.

Prerequisite

JDK 17, Node.js, and yarn need to be installed on the host system — apart from the Docker. Note that the Apache Maven which is the build tool for the anonymisation server does not need to be installed neither in the host nor in the container because Maven wrapper is provided and used.

The semi-cloud setup requires an additional installation of the PostgreSQL client in the host system if the setup is run without the server profile. The minimum supported version is version 13.5 released on November 11, 2021.

The setup is customized from:

- `dev/.env`,
- `application.yml`,
- `application-dev.yml`.

Verify if the required software is installed:

```
java --version
psql -V # if running without server profile
node -v
yarn -v
```

Installation procedure

Start up the database server:

```
cd docker/dev
docker compose up
```

Start the anonymisation server:

```
cd backend
./mvnw spring-boot:run
```

Alternatively, the server could be started up in the container together with the database server by specifying the server profile:

```
cd docker/dev
docker compose --profile server up
```

Start the client:

```
cd frontend
yarn install
yarn start
```

4.2.4 Local setup

While the preferred setup for development is the semi-cloud, the local on-premise setup is the preferred option if the Docker is not installed.

The setup is customized from `application.yml`.

Prerequisites

Apart from Docker, local setup requires everything that the semi-cloud setup requires, and an additional installation of the PostgreSQL *server*.

Verify if the required software is installed:

```
java --version
postgres -V
psql -V
node -v
yarn -v
```

Installation procedure

```
cd backend
./mvnw spring-boot:run
```

Start the anonymisation server:

```
cd backend
./mvnw spring-boot:run
```

Start the client:

```
cd frontend
yarn install
yarn start
```

If all steps are successful the application is fully accessible.

Note that the services are decoupled from each other, e.g. the client service (frontend) does not necessarily need to be started to access the anonymisation server capabilities, i.e., the server would still be accessible through the REST API.

4.3 Activation

The database of a new instance of the anonymisation platform is preloaded with at least one admin account. The credentials of this initial account can be configured from the properties and must be overridden in production:

```
core:
  preloader.admin:
    login: admin@admin.com
    password: admin
```

4.4 Types of users

Four types of users can interact with the system:

- admin user,
- verified user,

- unverified user,
- anonymous user.

It is very important to note that while the admin user can manage other users' accounts, he is not authorized to access the users' confidential data such as anonymisation scripts. The admin user who may be a non-technical business person should not be confused with the system administrator.

The relation between the users and their use-cases are summarized in Fig. 3.1 and 3.2.

4.5 User manual

It is easy to say that anonymisation platform was designed with the user experience in mind. The application presents complex technical concepts with graspable business abstractions. The end-user must not be surprised with the functionalities he interacts with – intuitiveness is a crucial component for building user engagement. This had allowed limiting the user manual's complexity.

This section presents a brief overview of how different users interact differently with the platform.

4.5.1 Anonymous user

An anonymous user can register, login, or reset his password.

4.5.2 Unverified user

The unverified user should activate his account by verifying the e-mail address to access the platform interface.

4.5.3 Verified user

A sequential scenario for anonymising the database uploaded by the verified user involves generating the template by uploading the database dump, producing the worksheet from it, building up the anonymisation request by adding individual

anonymisation operations to the selected attributes, and finally generating the outcome.

To offer maximum flexibility for the users trying to anonymise their data, the user can produce multiple worksheets from one template and can also generate multiple outcomes from one worksheet. This allows the user to produce several de-identified dumps and compare them for the best-expected results that meet the established business needs.

The summary interface exists to concisely represent the worksheet in one place reducing the need to transition between the platform's views.

It is possible for the user to always return to the previously generated templates, produced worksheets or generated outcomes. Additionally, the user may want to inspect and download the template metadata or request to remove his account along with its data.

4.5.4 Admin user

Admin user must not be able to access the user's sent database data. This user role can merely manage the users themselves, i.e., block, unblock, verify or delete the accounts. The user management is accessible through the users panel.

Another admin-related functionality is inspecting and on-demand running of the defined tasks (i.e., schedulers). This is accessible through the tasks panel.

4.6 System administration

System administrator and admin user must not be confused. The former is a technical user responsible for technical software processes such as configuration and deploy, whereas the latter is a non-technical business user responsible for managing the platform users.

Anonymisation server is highly configurable from `application.yml`, `application-<env>.yml`, and `<env>/.env` files. The idea is that concrete environments may need to behave differently, therefore the possibility to override only the selected local properties must be supported and is achievable through the creation of extra overriding files. A property from `application.yml` may be overridden

```
POSTGRES_TAG=13.1
POSTGRES_HOST_PORT=5007
POSTGRES_CONTAINER_PORT=5432
POSTGRES_PORTS_MAPPING=${POSTGRES_HOST_PORT}:${POSTGRES_CONTAINER_PORT}

POSTGRES_DB=anonymisation_db
POSTGRES_USER=postgres
POSTGRES_PASSWORD=postgres
POSTGRES_DATA_PATH=/var/lib/postgresql/data

SERVER_PORTS_MAPPING=8080:8080
CLIENT_PORTS_MAPPING=3000:3000
```

Figure 4.2: Environment file – development.

from the `application-<env>.yml`.

The system administrator must be provided with a possibility to configure the platform in an externalized way. The configuration changes merely require to re-deploy the platform – it is not required to re-build it.

4.6.1 Environments

Consider the docker compose visible in Fig. 4.3 which is further configured by the environment file visible in Fig. 4.2. Note that the environment files are typically not uploaded to the repository. The system administrator must configure the docker compose files from the environment files. Most of the configuration is related to the database connection.

The anonymisation server can override the individual properties in `application-<env>.yml`. The configuration visible in Fig. 4.4 shows an example of how the production environment may want to override certain properties.

```yaml
version: '3.9'

services:
  anonymisation_postgres_db:
    image: postgres:${POSTGRES_TAG}
    ports:
      - '${POSTGRES_PORTS_MAPPING}'
    environment:
      POSTGRES_DB: ${POSTGRES_DB}
      POSTGRES_USER: ${POSTGRES_USER}
      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
    volumes:
      - postgres_data:${POSTGRES_DATA_PATH}
    tty: true

  anonymisation_server:
    ports:
      - '${SERVER_PORTS_MAPPING}'
    build:
      context: ../../backend
    depends_on:
      - anonymisation_postgres_db
    environment:
      SPRING_PROFILES_ACTIVE: prod
      POSTGRES_IP_ADDRESS: anonymisation_postgres_db
      SPRING_DATASOURCE_URL: jdbc:postgresql://anonymisation_postgres_db
      ↪  :${POSTGRES_CONTAINER_PORT}/${POSTGRES_DB}
    tty: true

  anonymisation_client:
    ports:
      - '${CLIENT_PORTS_MAPPING}'
    build:
      context: ../../frontend
    depends_on:
      - anonymisation_server

volumes:
  postgres_data:
```

Figure 4.3: Docker compose – production.

```
# Miscellaneous
server.environment.cloud: true

# Uploading module configuration
uploading.templates.path: /var/lib/anonymisation/templates

# Processing module configuration
processing:
  anonymisations:
    scripts.path: /var/lib/anonymisation_platform/anonymisations/scripts
  dumps:
    scripts.path: /var/lib/anonymisation_platform/dumps/scripts
    archives.path: /var/lib/anonymisation_platform/dumps/archives
```

Figure 4.4: Anonymisation server configuration – production.

Prune containers data

The system administrator may want to execute `docker/prune.sh` script to prune all Docker-related data. The prepared script may be useful for development and must not be used for production.

To run the script after the prior assessment of necessity to do:

```
cd docker
sh prune.sh
```

The script's internals:

```
#!/bin/sh
docker system prune -f
docker container stop $(docker container ls -aq)
docker container rm $(docker container ls -aq)
docker rmi $(docker images -aq)
docker volume prune -f
```

4.6.2 Database

The primary database configuration can get complex.

Schema

By default, the database schema is dropped and re-created when the server restarts. This is configured through `spring.jpa.hibernate.ddl-auto` and must be overridden in production. Typically, production environments need to execute the schema-altering SQL scripts manually and tools like Flyway [46] are designed to support the versioning of the database.

```
spring:
  jpa.hibernate:
    ddl-auto: create
    show-sql: true
    generate-ddl: false
```

When `spring.jpa.hibernate.show-sql` property is enabled, all SQL queries are logged and enabling the `spring.jpa.hibernate.generate-ddl` will generate the log of the initial database schema. The script could be used by Flyway as the first initial version of the database.

Connection

The database url, username and password are all configurable and individual deployment setups override this property.

Local setup:

```
spring:
  datasource:
    url: jdbc:postgresql://localhost:5007/anonymisation_db
    username: postgres
    password: postgres
```

For the containerized setups, this property is overridden in docker compose:

```
services:
  anonymisation_server:
      SPRING_DATASOURCE_URL:  jdbc:postgresql://anonymisation_postgres_db
      ↪  :${POSTGRES_CONTAINER_PORT}/${POSTGRES_DB}
```

4.6.3 Files

Files are the fuel for anonymisation platform and hence the need for customizations.

Maximum accepted file size of the user uploaded dumps:

```
spring.servlet.multipart:
  max-file-size: 100MB
  max-request-size: 100MB
```

The location configuration of the files managed throughout the anonymisation platform: processing:

```
  anonymisations.scripts.path: stored_files/anonymisations/scripts
  dumps:
    scripts.path: stored_files/dumps/scripts
    archives.path: stored_files/dumps/archives
```

Note that those are relative paths. Containerized environments override these properties to the full paths as depicted in Fig. 4.4.

4.6.4 Mail service

Mail service is also configurable:

```
spring:
  mail:
    host: smtp.gmail.com
    port: 587
    username: data.anonymisation
    # password: <Specified in secrets.properties>
    properties:
      smtp:
        auth: true
        starttls.enable: true
```

In this case, the password for the mail service is provided in separate `secrets.properties` file that is not the part of the git repository.

4.6.5 Unprotected resources

By default, all REST resources require authentication and may require authorization. REST resources that do not require the user to authenticate must be configured:

```
core:
  api:
    unprotectedResources: /api/v1/auth/**,\
                          /api/v1/users/register/**,\
                          /api/v1/users/verify-mail/**,\
                          /api/v1/reset-password/**,\
                          /api/v1/me/restore-account/**
```

4.6.6 Authorization

The access and refresh tokens are used for authorization and are generated after the user authenticates himself by logging in. The configuration is necessary to control the behaviour of this:

```
core:
  jwt:
    algorithm: HS512
    secretKey: /EOAqvJOzlPxRdTJO5iblCYSCGMXsVaCU47BpvxvO19L87
    ↪  /pVpyoab9sy2rDzHS5vNwHzu8rX/E8FJGqx5oCkA==
    accessToken.expireTimeInSeconds: 1800
    refreshToken.expireTimeInSeconds: 2419200
```

Note that the access token expires quickly while the refresh token is to stay for a long time. The secret key needs to be both valid for the selected hashing algorithm and also base64 encoded. Currently, the only supported algorithm is `HS512`.

4.6.7 User account

The settings for the expiring time duration of the user account related services:

```
core:
  resetPasswordToken.expireTimeInSeconds: 3600
  verifyMailToken.expireTimeInSeconds: 604800
  undoRemoveAccountToken.expireTimeInSeconds: 604800
```

4.6.8 Scheduler

The existing schedulers which are triggered based on the cron expression must be highly customizable, as there are quite a few of them, namely the schedulers are responsible for:

- notifying the users about approaching expiration of their account expiration due to inactivity,
- removing the inactive accounts after the configured duration of being inactive,
- removing the unverified accounts after the configured duration of being unverified,
- pruning the user data who wish to delete their account.

For brevity, the configuration of only one scheduler is shown:

```
scheduler:
  notifyExpiringAccounts:
    scheduled: true
    executable: true
    cron: '0 0 0 * * *'
    notifyAfterTimeInSeconds: 300
    description: Send mail notifications to expiring accounts
```

The expression from the example is triggered daily and an easier form of `@daily` could also be used [62]. There is a functionality that is able to compute the exact date of the next scheduled execution and shows it in the tasks panel.

The available paramaters include:

- `scheduled` – controls if the given scheduler is periodically launched.
- `executable` – controls if the given scheduler is manually executable from the tasks panel.
- `cron` – the trigger expression.
- `description` – the description for the tasks panel to identify the scheduler.
- `notifyAfterTimeInseconds` – this particular parameter is specific to the scheduler and specifies a way of how long the user needs to be inactive to receive a notification e-mail with a prompt to login.

4.7 Security issues

4.7.1 User account

The system requires the user to verify his e-mail address to access the platform capabilities. The application allows the blocking of potentially malicious users. To ensure the security of users' data through data protection measures, the users can request to remove their accounts.

4.7.2 Continuous integration

Multiple advanced continuous integrity workflows were established to follow the state-of-the-art software development practices of ensuring quality and security. The plans verify the state of quality, security, and tests. The workflows are triggered daily at a set time, upon creating a pull request or upon committing to the master branch.

4.7.3 Containers isolation

The platform runs as a secure network of isolated Docker containers hence minimising the possible system vulnerabilities that could be taken advantage of.

4.7.4 Authentication and authorization

Platform endpoints are gated behind the authorization logic, i.e., to access the platform resources the user first needs to authenticate himself by logging in to his account, and then his account is also required to have the appropriate role authority to access the particular resources such as the templates or worksheets.

The authentication and the authorization is implemented using JWT access tokens. To increase the safety of the system, the functionality is further enhanced by the functionality of refresh tokens.

4.7.5 Validation

The application pays special attention to the validation which is performed on the client's side and the server's side in a multi-level fashion. Taking the proper validation measures prevents the potentially devastating effects of putting the application in a breaking, unsupported state.

4.8 Working scenarios

Zajmę się tym podrozdziałem po skończeniu pozostałych rozdziałów.

Chapter 5

Internal specification

This chapter summarizes the most significant internal implementation aspects of the engineered anonymisation platform. The concepts and architecture of the system are supported by diagrams that are rich in details. A total number of more than 400 source code files[1] somewhat describe the system's complexity hence it could not be a goal for this chapter to present all the – often repetitive and not engaging – technical details. Instead, the chapter attempts to focus on the quality of explanation and examples rather than repetitiveness.

5.1 Architecture analysis

Anonymisation platform implements a client-server architecture that was modernized through modularization and containerization. The high-level overview of the system architecture is visible in Fig. 5.1.

5.1.1 Architecture – server

The server had been designed in a modular manner since the very beginning of its implementation – with the eventual distributed architecture preserved in mind. The server aims to separate the domain and infrastructure following the principles of clean architecture [49].

[1]The data as of January 21st, 2022.

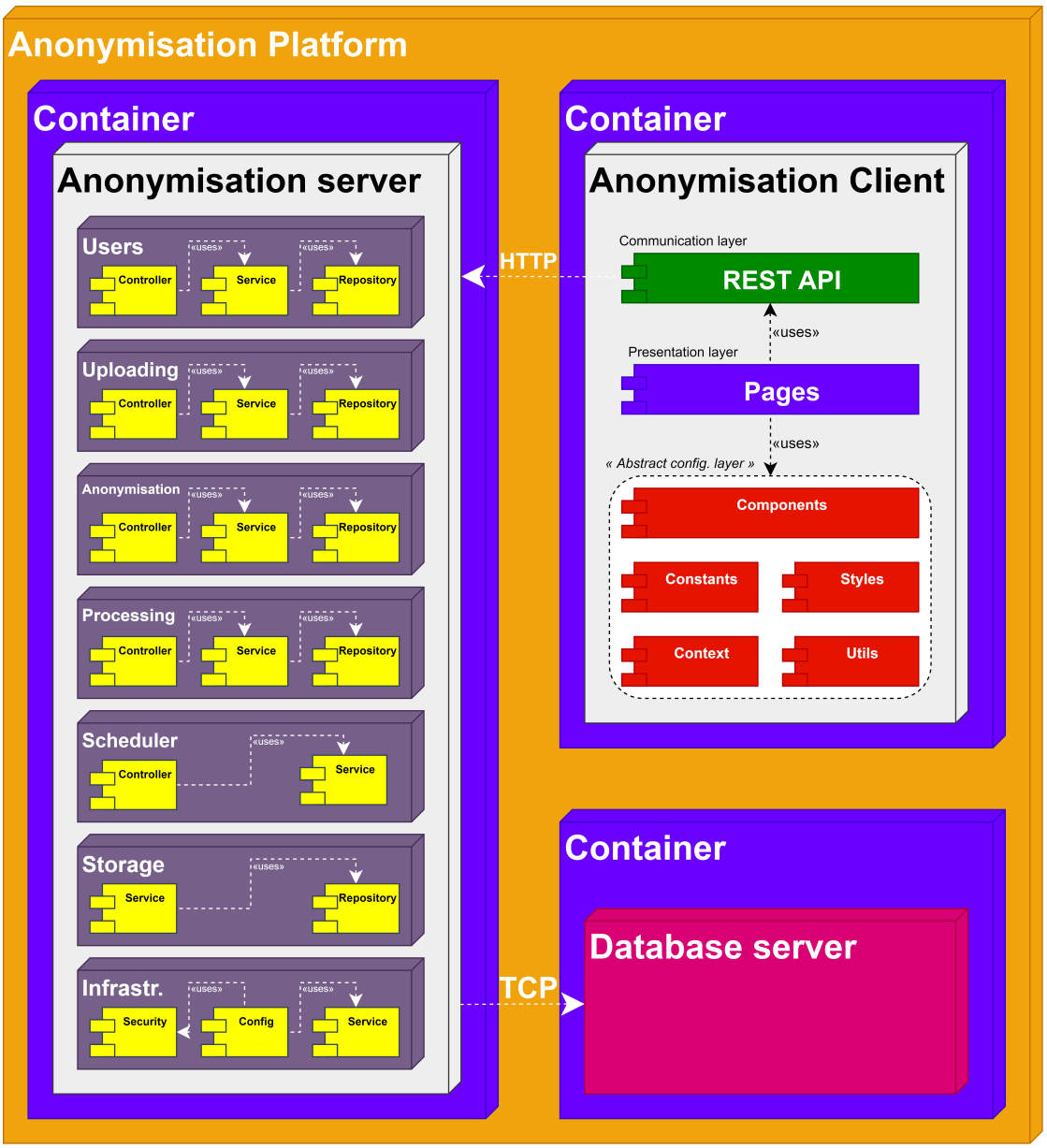

Figure 5.1: Architecture – anonymisation platform overview

Microservices

The developers who design and develop a new software system and start right away with adopting the microservice-oriented distributed architecture may endanger their system with the risk of becoming the so-called *distributed monolith* – rather than the intended true distributed microservice system. The system is classified as a distributed monolith if the seemingly independent services cannot coexist independently – therefore enforcing the deployment of all the services to make anything at all work.

Instead, to mitigate this risk, it is thought it would be better to start with the adoption of *modular monolith* architecture first [79]. As the services in this case are already divided into separate business concerns, future transition to distributed architecture is a natural consequence. The platform is designed in this modular fashion.

Modularity is always the prerequisite for reusability, scalability, high availability, testability, or decent maintainability.

Modularity

There are seven modules in the server – which are to a high extent decoupled from each other hence allowing the *separation of concerns* which is one of the most relevant development technique [50].

The available modules are:

- **Uploading** – responsible for template management, i.e., persisting the user provided dump, restoring the database from it and extracting the metadata.
- **Anonymisation** – responsible for worksheet management, i.e., building up the anonymisation request through adding the anonymisation operations to the individual columns of the worksheet produced by the template.
- **Processing** – responsible for outcome generation, i.e., the processing of the anonymisation request built by the user. The outcome generation includes creating a mirror database, creating and populating anonymisation script, executing the anonymisation script on the mirror database, and finally dumping and persisting the anonymised database as the result outcome.

- **Users** – responsible for the functionalities related to the users management including technical details like user's authentication.
- **Scheduler** – responsible for the scheduling and execution of the administration tasks.
- **Storage** – responsible for storing the variety of available types of files created throughout the platform's use-cases.
- **Infrastructure** – responsible for providing other domain-agnostic services such as the server configurations, security, mail service, processor executor, and a number of other database-related functionalities.

The modularization eases the enforcement of the single responsibility principle of SOLID [50] – after all the only reason for, e.g., the uploading module to change is to alter the template generation behaviour.

Horizontal architecture

The individual modules follow the approach of the horizontal architecture design, i.e., the modules are implemented in a package-by-layer way. It is commonly agreed to be the simplest design in the enterprise-class software development [39].

The packages that are typically adopted as the layers are the three:

- **Controller**[2] – define the REST API endpoints[3] that work as the application's entry points for the external clients.
- **Service**[4] – define the domain functionalities of the application. In a non-trivial application, which this engineered solution is, activating one controller may activate dozens of services.
- **Repository** – the layer responsible for data access, i.e., persistence.

[2]The other synonyms recognised in the industry are *ui*, *presentation*, and *resource*.

[3]Although starting to become obsolete, controllers often perform the so-called server-side rendering which is a fancy name for returning the view that should be displayed to the clients.

[4]Commonly referred to as the domain.

Other packages may include, e.g., the domain objects or mappers for data transfer objects.

Given the fact that the names of the packages are fixed, the code is decoupled from the underlying technical purpose. An individual layer is dependent only on the adjacent layers – for example controller layer must not be dependent on the persistence layer, meaning that the persistence layer is not aware of the controllers existence, and vice versa.

Vertical architecture

Although a particular module has the characteristics of a layered horizontal architecture, the server being divided into seven distinct modules also exhibits the characteristics of a vertical architecture [49] where the related domain concepts are encapsulated within one package, i.e., package-by-feature. Rather than using the fixed technical package names, the top-level modules are now able to explain their business domain purpose simply by their names, e.g., it can be expected from the anonymisation module to be responsible for thereof.

Horizontal-vertical duality

The engineered software exhibits the horizontal-vertical duality[5] of its architecture concerning the packages organisation. Depending on the abstraction level, the system benefits from both of the horizontal and vertical ways to organise packages – and hence responsibilities.

On a high level, we have the business domain modules such as uploading, anonymisation or processing. On a lower technical level we have the controllers, services and repositories. Therefore this design combines the advantages of both architectural approaches depending on the level of abstraction.

Implementation context

To meet the complex and versatile needs of anonymisation platform, the server needs to implement a wide range of services. Wherever possible, the existing well-

[5]The paper introduces this name as the inspiration of wave-particle duality from quantum mechanics.

tested and secure libraries are used to avoid re-inventing the wheel. Note that the actual implementations of interfaces do not follow the convention of putting *-Impl* suffix to the class name – this is considered a poor coding practice [49].

5.1.2 Architecture – client

The author believes that – in terms of architecture – complex software systems mostly evolve on the server-side. The client architecture is less likely to be changing. As the system grows, the complexity of the client architecture stays on the relatively unchanged level.

The client code was divided into:

- **Api** – communication layer with the server – the collection of available RESTful endpoints ready to be requested through axios HTTP client,
- **Pages** – presentation layer – renders the view pages,
- **Components** – re-usable code for the pages such as dialogs or inputs,
- **Constants** – client's editable configuration,
- **Styling** – drives the client's appearence,
- **Context** – encapsulates authentication context, i.e., currently logged in user,
- **Utils** – re-usable functions.

The client architecture is visualised in details as the part of the diagram in Fig. 5.1 that shows the entire platform's architecture overview.

5.1.3 Architecture – containerization

The platform consists of three communicating containers, i.e., anonymisation server, anonymisation client and PostgreSQL database.

The containers are decoupled from each other and do not necessarily require to be deployed on the same host machine and enable an independent maintenance of thereof, i.e., the client can be re-deployed without re-deploying the server. The same applies to the database re-deployment.

5.2 Server in details

The modules are supported by the detailed diagrams whenever appropriate.

5.2.1 Platform development concepts

A brief introduction to the concepts behind the platform's code – that is mostly based on the Spring Framework – needs to be introduced as the knowledge basis for the extensive code-base. Presented code snippets strip the code from the irrelevant details such as the access modifiers.

Controller layer and dependency injection

Consider the simplified code of `UserController` of the users module:

```
1  @RestController
2  @RequestMapping("/api/v1/users")
3  class UserController {
4    UserService userService;
5    AuthService authService;
6
7    UserController(UserService userService, AuthService authService) {
8      this.userService = userService;
9      this.authService = authService;
10   }
11 }
```

The code defines a Spring component in the form of a REST controller with the base URI of `/api/v1/users`. This component is composed of `UserService` and `AuthService` that also happen to be Spring components.

Spring components are internally singletons [30] created at the application's start-up time – all Spring dependencies are created in this manner and are maintained within the Spring container. The constructor is only called internally by the Spring and never called explicitly by the developer. This allows the framework to provide the core functionality of dependency injection [85] – the dependencies of `UserController` are automatically wired into it – which is the primary reason why Spring became so popular as this speeds up the development process.

Note that the REST API is being versioned [87].

Consider the concise REST endpoint responsible for user registration:

```
@PostMapping("/register")
  ResponseEntity<ApiResponse> register(@Valid @RequestBody
  ↪  RegisterUserRequest dto) {
  return ResponseEntity.ok(authService.register(dto));
}
```

The declarative specification of `@PostMapping("/register")` explains that this will be the HTTP POST endpoint. When combined with the class level base URI declaration, the resource is computed to be available at `/api/v1/users/register`. This endpoint will be executed only if the validation of the input `RegisterUserRequest` meets the validation criteria. The data transfer object (DTO) is in fact the request body sent by the client. Upon successful validation, the controller delegates its work to the `AuthService` which belongs to a different abstraction layer. Finally, the endpoint returns the response 200 OK [34] with the response body of whatever was returned by the service. The response body is serialised to a JSON representation.

Request validation

Let's examine the declarative approach to the validation of DTOs based on the `RegisterUserRequest`:

```
class RegisterUserRequest {
  @Email
  @NotBlank
  @Size(max = 40)
  String email;

  @NotBlank
  @Size(min = 4, max = 40)
  String password;
}
```

The server requires the client to provide a valid e-mail, among others, that is not blank and of size up to 40 characters. Failure to meet the criteria returns an error of 400 Bad Request to the client. Let's consider yet another endpoint of the same controller, responsible for platform users retrieval hence specifying `@GetMapping`:

```
@GetMapping
@PreAuthorize("hasAuthority('ADMIN')")
```

```
ResponseEntity<List<FullUserResponse>> getAll() {
  return ResponseEntity.ok(userService.getAll()
                                      .stream()
                                      .map(FullUserResponse::from)
                                      .collect(Collectors.toList()));
}
```

The declarative specification of `@PreAuthorize("hasAuthority('ADMIN')")` effectively authorizes only the admin users to access this endpoint. Unauthorized users will get the error of 403 Forbidden.

Data transfer object mapping

Note that the result list obtained from the user service is mapped to the list of `FullUserResponse` which is the DTO returned to the client. This mapping occurs in the controller layer and not in the service layer as this is the concern of presentation and not domain[6]. The relevant concept to understand here is that the underlying database entities[7] are not necessarily the same as returned data transfer objects[8] returned to the client. Consider the internals of the returned data transfer object and its mapping logic:

```
/** Password and relations are not returned to the client. */
class FullUserResponse {
  String id;
  String email;

  /** User is a complex database object. */
  static FullUserResponse from(User user) {
    var dto = new FullUserResponse();
    dto.setId(user.getId());
    dto.setEmail(user.getEmail());
    return dto;
  }
}
```

The method `from` maps the complex database entity object to a simple data transfer object.

[6]The domain layer is not aware of the presentation layer

[7]Possibly very complex database objects.

[8]Usually simple objects encapsulating only the necessary information for the particular endpoint.

Database entity

Consider the drastically simplified form of the `User` database entity.

```
@Entity
@Table(name = "users")
class User {
  String email;
  @Id String id = UUID.randomUUID().toString();
  @OneToMany(mappedBy = "user") List<Template> templates;
  @OneToMany(mappedBy = "user") List<Worksheet> worksheets;
  @Enumerated(EnumType.STRING) Role role;
}
```

The user object is linked with a one-to-many relationship with templates and worksheets and is saved in the `users` table using the capabilities of Spring Data repositories.

Repository

Spring's declarative approach is demonstrated in the repository layer as well. The developer defines the following simplified interface:

```
@Repository
interface UserRepository extends PagingAndSortingRepository<User, String> {
  Optional<User> findByEmail(String email);
  boolean existsByEmail(String email);
}
```

The implementation for this interface and other Spring repositories are automatically created during application's start up.

The developer merely specifies interface methods such as `Optional<User> findByEmail(String email)`. This method needs to be named in a specific manner [85] as the actual SQL implementation (e.g. `SELECT * FROM users WHERE email = ?`) are created by the Spring solely based on the method name.

5.2.2 Users module

The internals of the users module are described in detail in Fig. 5.2.

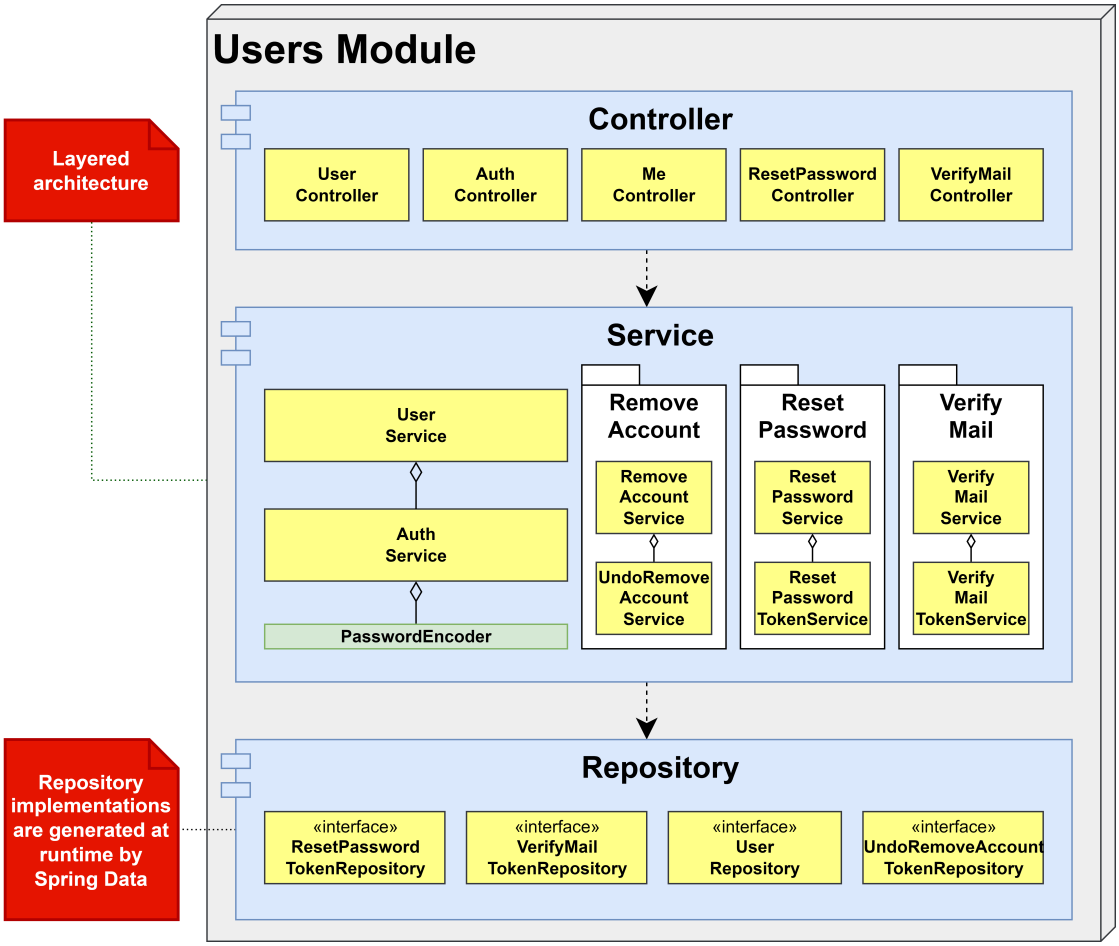

Figure 5.2: Architecture – users module

5.2.3 Uploading module

The internals of the uploading module are described in detail in Fig. 5.3. There are two possible restore modes for the dumps provided by the user:

```
public enum RestoreMode {
  SCRIPT, ARCHIVE
}
```

5.2.4 Anonymisation module

The internals of the anonymisation module are described in detail in Fig. 5.4. This module defines database entities for each individual anonymisation operation and links it with `ColumnOperations`. Example for the suppression method:

```
@Entity
@Table(name = "suppressions")
class Suppression {
  String suppressionToken;

  @OneToOne(mappedBy = "suppression")
  ColumnOperations columnOperations;
}
```

`ColumnOperations` is an aggregate of the configured operations for the given uniquely identifiable column.

```
@Entity
@Table(name = "column_operations")
class ColumnOperations {
  String tableName;
  String columnName;

  @ManyToOne(fetch = FetchType.LAZY) Worksheet worksheet;

  @OneToOne Suppression suppression;
  @OneToOne ColumnShuffle columnShuffle;
  @OneToOne RowShuffle rowShuffle;
  @OneToOne PatternMasking patternMasking;
  @OneToOne Shortening shortening;
  @OneToOne Generalisation generalisation;
  @OneToOne Perturbation perturbation;
```

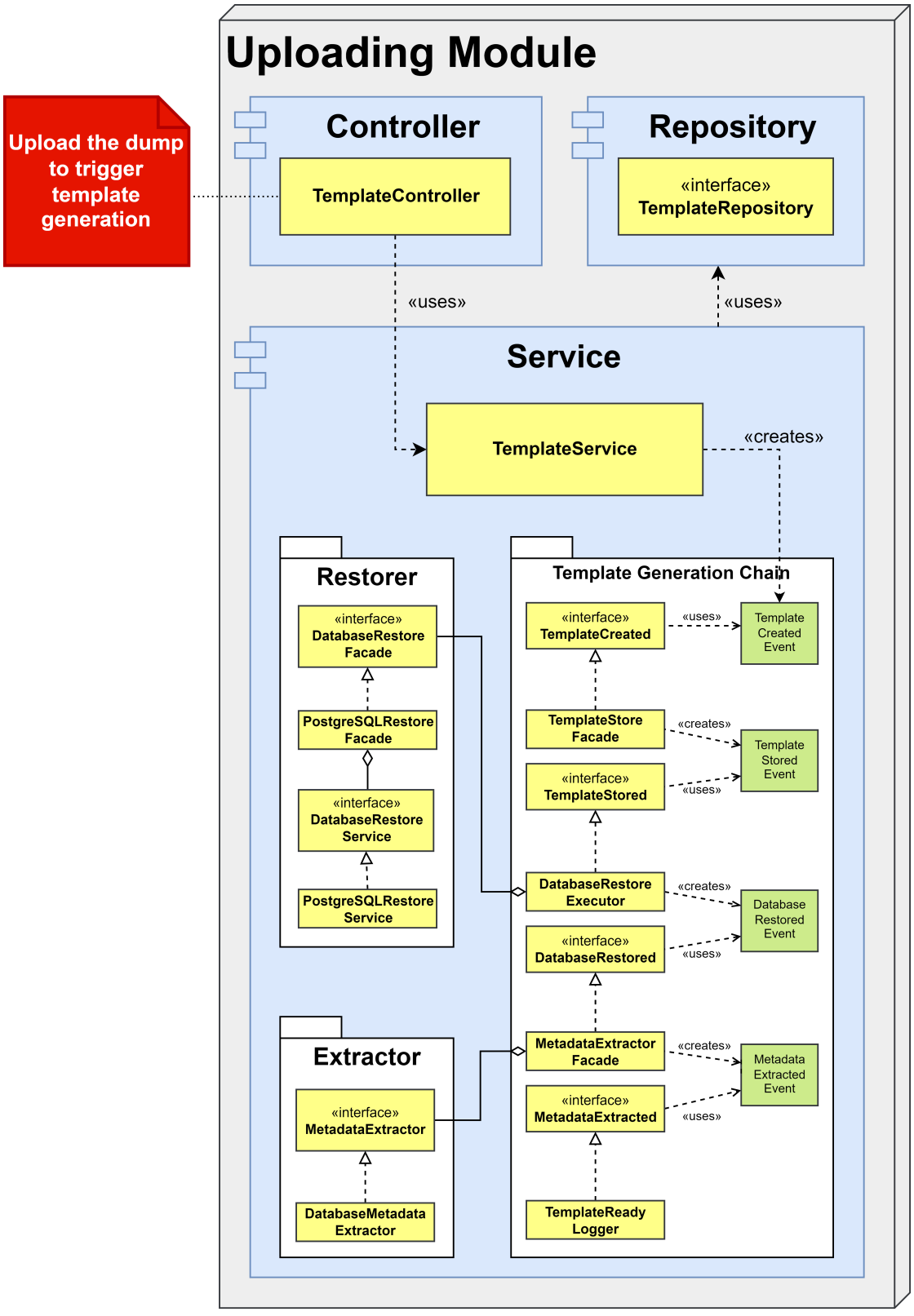

Figure 5.3: Architecture – uploading module

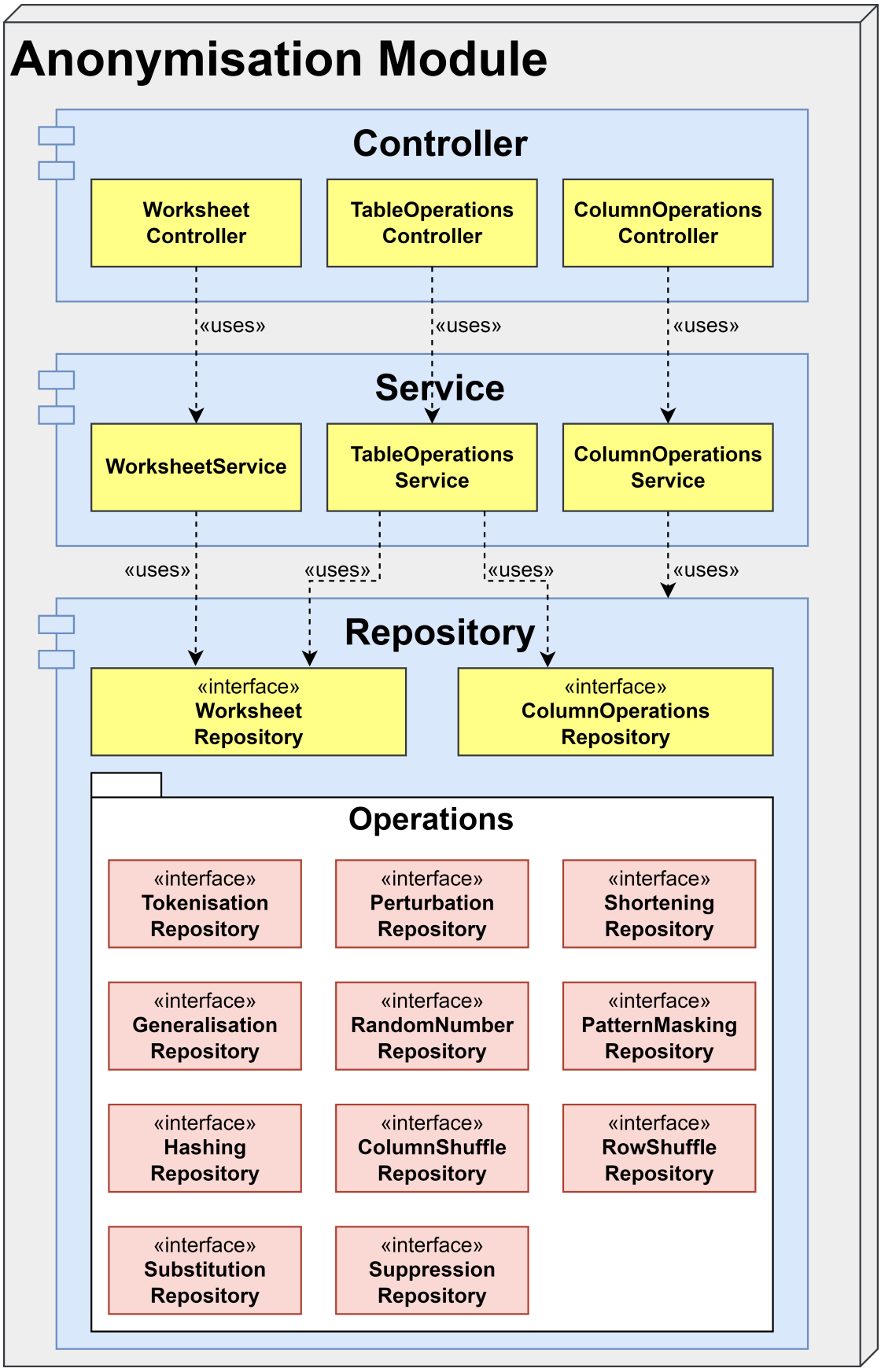

Figure 5.4: Architecture – anonymisation module

```
    @OneToOne RandomNumber randomNumber;
    @OneToOne Tokenization tokenization;
    @OneToOne Hashing hashing;
    @OneToOne Substitution substitution;
  }
```

5.2.5 Processing module

The internals of the processing module are described in detail in Fig. 5.5. To provide an even greater detail of this module, an additional diagram is provided in Fig. 5.6. The implementation shown takes the advantage of a combination of design patterns including template method, facade and factory [30].

5.2.6 Scheduler module

The scheduler module defines a set of cron job related task services as shown in Fig. 5.7.

5.2.7 Infrastructure module

This module is different from the majority of other modules because it does not have controller or repository layer. Infrastructure module is responsible for the configuration of security, swagger, web, asynchronous events, scheduling. The configurations are declarative and simple, e.g.:

```
@Configuration
@EnableAsync
class AsyncEventsConfiguration {}
```

Services

Provided services include, e.g., a mail service shown in Fig. 5.8. Another exemplary service is a factory for processor executor which is responsible for executing commands from the Java code. The following is a simplified form of it:

```
class ProcessExecutorFactory {
  static ProcessExecutor newProcess(String... cmd) {
    return new ProcessExecutor(cmd)
      .exitValue(0)
```

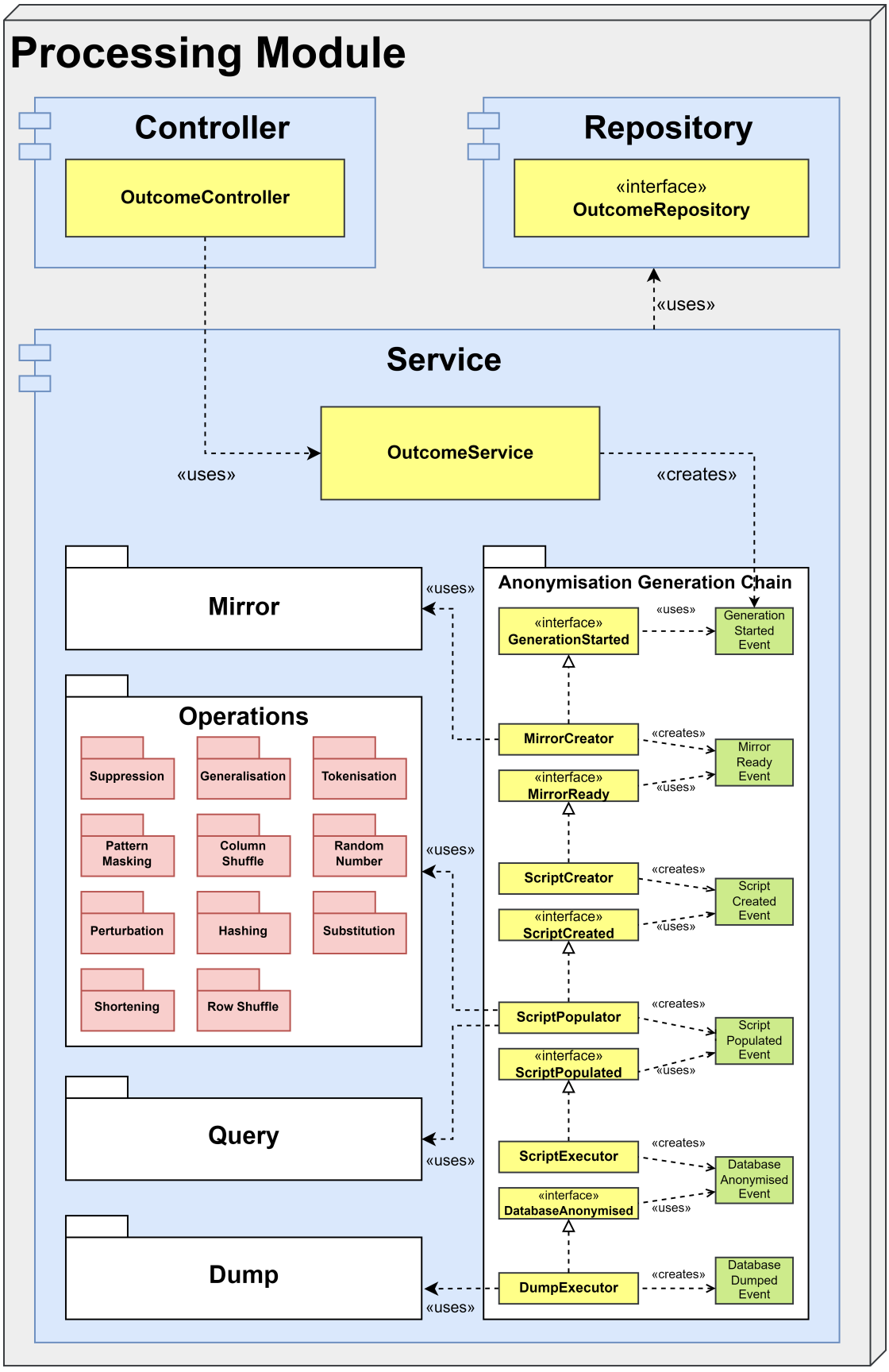

Figure 5.5: Architecture – processing module

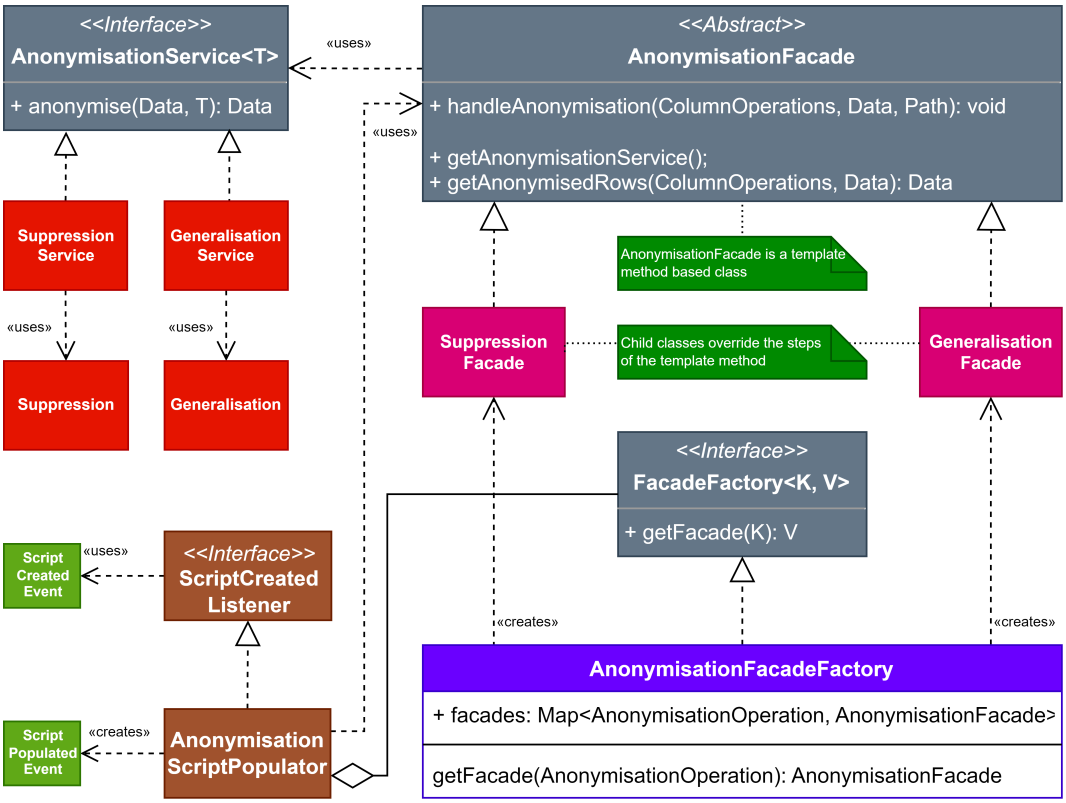

Figure 5.6: Processing module – anonymisation details

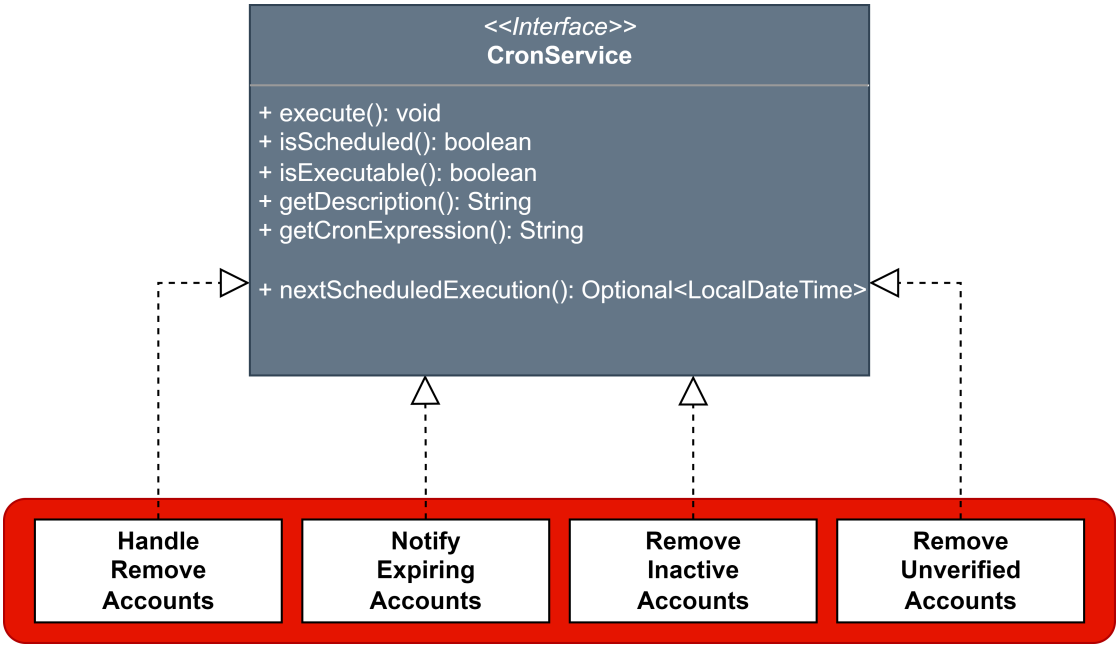

Figure 5.7: Scheduler module internals

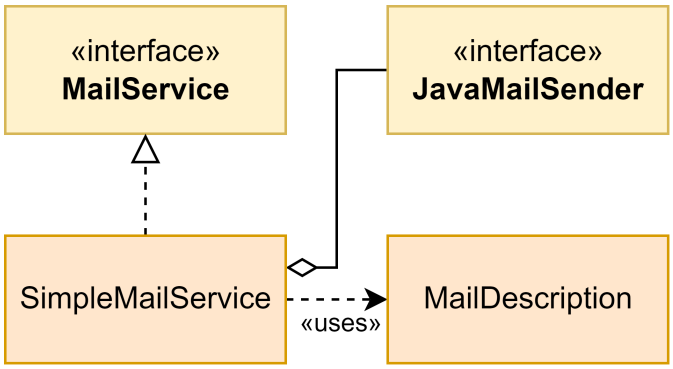

Figure 5.8: Mail service diagram

```
        .readOutput(false)
        .destroyOnExit();
    }
  }
```

An exemplary usage of the processor executor for the dynamic database creation:

```
if (details.isRunningOnCloud()) {
  ProcessExecutorFactory.newProcess(
    "createdb",
    "-h", details.getIpAddress(),
    "-U", "postgres", "--no-password",
    "-T", databaseName,
    newDatabaseName
  ).execute();
}
```

5.2.8 Storage

The storage module is responsible for managing the files. There is an implementation for storing the files on a local host. The implementation heavily relies on the template method design pattern [30] as shown in Fig. 5.9. The storage is configured with the docker volumes capabilities to decouple the files from the lifetime of the containers. Other design patterns involved in the platform are facades to provide simpler interface to complex objects [30] or builder for querying the database.

5.3 Client in details

The client internals are shown as the part of the diagram depicted in Fig. 5.1.

5.3.1 Presentation layer

An exemplary greatly simplified code for the rendering of TypeScript and React driven pages is shown below – the code renders users panel:

```
const Users = () => {
  const { data, isLoading, isRefetching } = useQuery('users', getUsers);
```

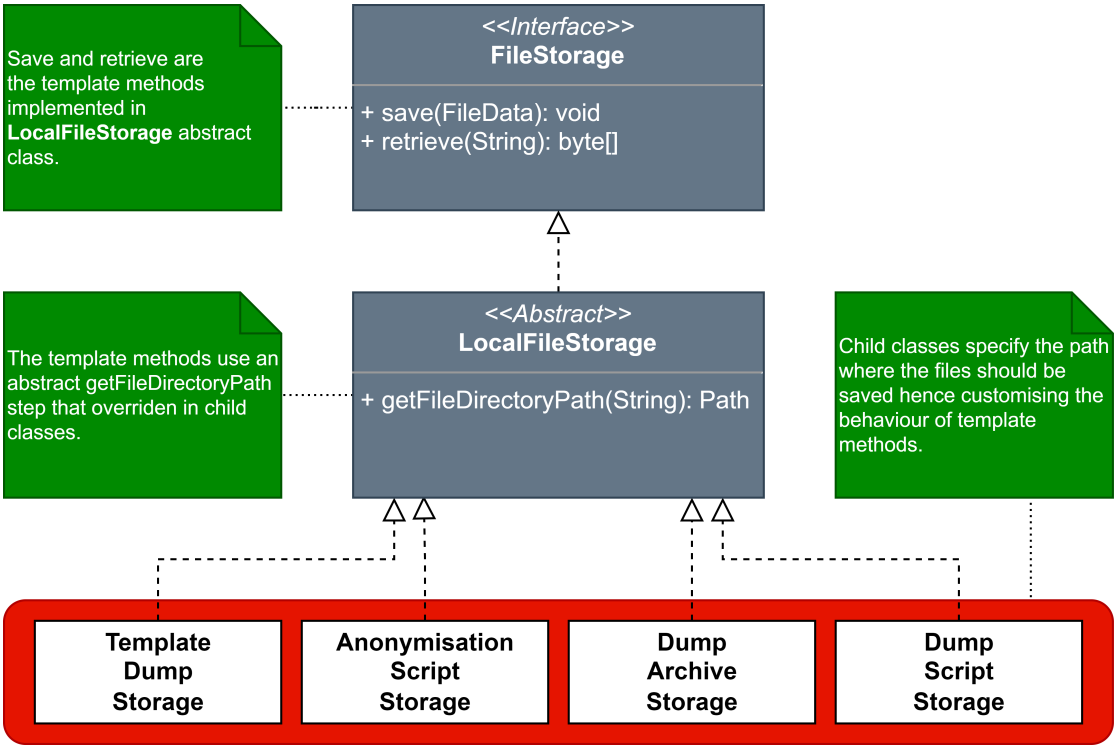

Figure 5.9: Storage service – template method example

```
    return <DataGrid autoHeight columns={columns} rows={users}
    ↪  loading={isLoading || isRefetching} />
}
```

5.3.2 Communication layer

The APIs are easy to define. To create the method that will have the capabilities to asynchronously call the server using REST API is simple:

```
export const getUsers = () => {
  return axios.get<FullUserResponse[]>(`/api/v1/users`);
};
```

Type safety

The TypeScript offers the type safety possibilities through interfaces:

```
export interface FullUserResponse {
  id: string;
  email: string;
  role: Role;
  // ..
}
```

5.3.3 Configuration layer

We are a lot of configuration details in the client. The application must be re-deployed for the changes to the TypeScript files to take effect.

Routing

The application routes can be easily configured in a fine-grained fashion:

```
export const APP_ROUTES: RouteDescription[] = [
  {
    path: ROUTES.LOGIN,    // This path will...
    element: Login,        // ...render this.
    authenticated: false, // Requires logging in?
    menu: false,           // Should the menu be visible?
  },
  {
    path: ROUTES.MY_WORKSHEETS,
```

```
10      element: MY_WORKSHEETS,
11      authenticated: true,
12      menu: true,
13    }
14  ];
```

Menu

The menu is configurable in a fashion similar to the routes configuration. Developer can easily control which menu items should be displayable to which user roles. For example, only the admin user would be able to see the users item in the menu:

```
1  export const NAVIGATION_ITEMS: NavigationItemDescription[] = [
2  {
3    path: ROUTES.USERS,
4    name: 'Users',
5    roles: [Role.ADMIN],
6    icon: Groups,
7  },
8  {
9    path: ROUTES.MY_WORKSHEETS,
10   name: 'Worksheets',
11   roles: [Role.ADMIN, Role.VERIFIED_USER],
12   icon: TableView,
13 }];
```

There are other configuration, e.g., the colours theme can be changed from just one place, or the minimum durations for the skeletons and spinners to show up for building up the user engagement.

Chapter 6

Verification and validation

The researchers must remain unbiased to innovate with the research of a reliable value [68] – and likewise, the assessment of the designed software quality must be free of bias to be genuinely objective. This chapter evaluates the methods applicable for ensuring the high quality of the system.

6.1 Testing

The author believes there exists room for improvement as far as the unit tests and integration tests are concerned. To compensate for the problem of a small quantity of the created tests – mostly caused by the volume of the designed system – other cutting-edge software practices must be introduced and demonstrated.

6.1.1 Server

Testcontainers library

The server takes the advantage of the Testcontainers library for containerizing the integration tests. Effectively, the tests can be run in isolation from each other. This is the start-of-the-art library for running the integration tests (or even E2E tests as it is also supported).

Thanks to the Docker capabilities, it is easy to bootstrap the PostgreSQL database for tests – which is the same type of database as the production database.

Running the same type of database for both tests and production yields maximum reliability. A common poor practice among developers is to bootstrap tests with in-memory databases like H2. This is a poor practice for integration tests. The tests have diminished value if they are running on, e.g., H2 database, whereas the production is running on, e.g., PostgreSQL database. This may in the end produce false negatives and false positives. Such poor practices are avoided.

The containers can be instantiated based on a test suite level, test case level, or other. The configuration is simple:

```
@Testcontainers
@ActiveProfiles("test")
@SpringBootTest
public class PostgreSQLSpecification {

  @Container
  public static PostgreSQLContainer<?> container = new
  ↪  PostgreSQLContainer<>("postgres")
      .withUsername("postgres")
      .withPassword("postgres")
      .withDatabaseName("anonymisation_db");
}
```

The test classes need to extend this specification.

REST Assured

Another poor practice in the industry is to use `MockMvc` or a similar tool for integration tests. The reason why this is poor practice is that the integration tests must resemble the production environment, and mocked HTTP requests are what they are – a mock.

For this reason, a library that offers to send real HTTP requests in the tests is configured. The library also allows functionalities for validating the HTTP responses.

User in tests

`WebPostgreSQLSpecification` is created as the class that should be extended by the tests. This created specification allows the possibility to create a real admin user, verified user, unverified user or anonymous user in the test.

This class is shown in its full size for reference:

```
@ActiveProfiles("test")
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
public class WebPostgreSQLSpecification extends PostgreSQLSpecification {
  @LocalServerPort
  private int serverPort;

  public RequestSpecification adminClient() {
    return authenticatedClient(new AdminDetails());
  }

  public RequestSpecification verifiedUserClient() {
    return authenticatedClient(new VerifiedUserDetails());
  }

  public RequestSpecification unverifiedUserClient() {
    return authenticatedClient(new UnverifiedUserDetails());
  }

  public RequestSpecification authenticatedClient(AccountDetails details) {
    String accessToken = authenticate(details);
    return unauthenticatedClient()
        .header(HttpHeaders.AUTHORIZATION, "Bearer " + accessToken);
  }

  public String authenticate(AccountDetails details) {
    return unauthenticatedClient()
        .port(serverPort)
        .contentType(MediaType.APPLICATION_JSON.toString()).body(details)
        .when()
        .post("/auth/login")
        .then()
        .statusCode(200)
        .extract()
        .header("access_token");
  }

  public RequestSpecification unauthenticatedClient() {
    return given()
        .basePath("/api/v1")
        .port(serverPort)
        .accept(MediaType.APPLICATION_JSON.toString())
        .contentType(MediaType.APPLICATION_JSON.toString());
  }
}
```

Notice how the JWT access token is generated by forcing the user to login

before sending the HTTP request under test. Given that there is no stubbing of any authorization related-behaviour, the tests are rendered reliable.

Example

An exemplary integration tests taking the advantage of both of Testcontainers and REST Assured are shown:

```
public class GetUsersIntegrationTest extends WebPostgreSQLSpecification {

  @Test
  public void getAllAsAdmin() {
    adminClient()
        .when()
        .get("/users")
        .then()
        .statusCode(200)
        .body("size()", equalTo(3));
  }
  @Test
  public void getAllAsVerifiedUser() {
    verifiedUserClient()
        .when()
        .get("/users")
        .then()
        .statusCode(403);

  @Test
  public void getAllAsUnverifiedUser() {
    unverifiedUserClient()
        .when()
        .get("/users")
        .then()
        .statusCode(403);
  }
  @Test
  public void getAllAsUnauthenticated() {
    unauthenticatedClient()
        .when()
        .get("/users")
        .then()
        .statusCode(403);
  }
}
```

The database was preloaded with 3 users. By comparing the response status

Table 6.1: Client – code analysis tests.

Custom command	Underlying command
`yarn lint`	`eslint ./src --ext .js,.jsx,.ts,.tsx`
`yarn fix`	`eslint --fix ./src --ext .js,.jsx,.ts,.tsx`
`yarn format`	`prettier --write ./src/**`

codes, notice how it is possible to retrieve the users when trying to do as the admin. Different roles may not access this REST API endpoint[1] – after all the business logic is that only the admin has access to the users through the users' panel.

Naturally, the library supports databases or queues such as Redis, MongoDB, Apache Cassandra, or Apache Kafka – which makes this a great choice as the future extension point.

6.1.2 Client

Custom yarn commands were defined for the client application. Their responsibility is finding the problems by the measures of static code analysis and formatting the code. The commands are shown in Tab. 6.1 with the first one being responsible for the static code analysis, the second one additionally attempts to fix the addressable problems, and the third and last one re-formats the code.

6.1.3 Manual tests

Manual tests are part of the development life-cycle and were conducted as the software was progressing forward. These tests mostly concern the client.

6.1.4 Portability tests

The containerization of the software makes the platform system-agnostic. A system that can run Docker can run the platform.

As far as the hardware is concerned, no problems were detected on various hardware implementations including Apple M1.

[1] Because `@PreAuthorize("hasAuthority('ADMIN')")` protects the users resource.

6.2 Code analysis

The platform is scanned by the SonarQube static analysis tool [odnośnik]. This is great software, even for the enterprise-class level software, for finding the problems in code, including:

- code smells,
- vulnerabilities,
- security hotspots.

Separate instances were configured for the client and the server.

6.3 Security

Apart from finding the security problems by SonarQube, an additional CodeQL tool was configured to scan the platform.

6.4 Continuous integration

All the described tools were integrated into the Continuous Integration (CI) workflows in GitHub. The workflows trigger daily, upon commit to the master branch, or upon pull request state update.

Three workflows exist with slightly different but co-related purposes:

- security,
- testing,
- quality.

Individual CI run includes eight different checks. An exemplary CI run is visible in Fig. 6.1. The results are also rendered to the main readme for an easy way to spot detected regressions, as visible in Fig. 6.2. The badges may seem to be duplicates – this is caused by the separation of the client checks and server checks.

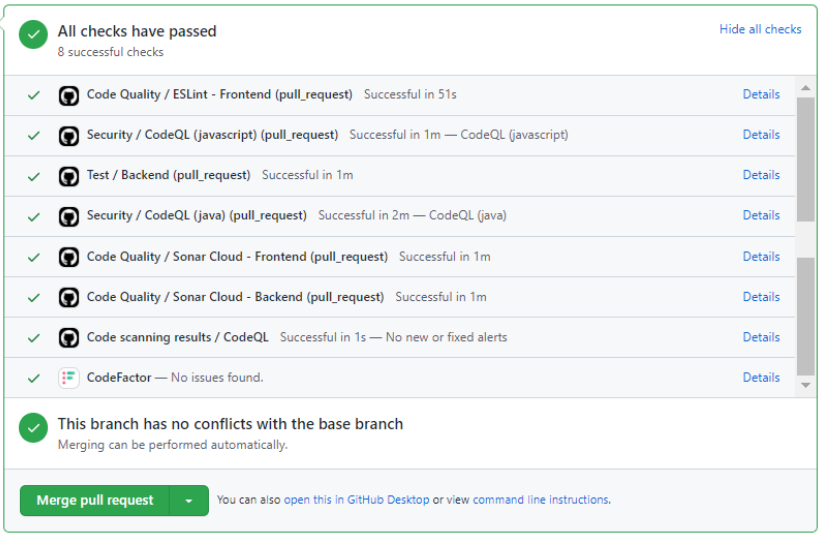

Figure 6.1: Continuous Integration – pull request checks

Figure 6.2: Continuous Integration – rendered badges

6.4.1 Detected problems

Various problems were detected as part of the development or testing. An example of a problem that was documented and fixed was described as an issue `#38` in the GitHub repository and was related to the concurrent file upload problems.

6.5 Validation

As defined in the functional requirements, the validation in the anonymisation platform is complex and multi-level. It works in the client's forms, the data transfer objects in the controller layer, the services in the business layer, and when persisting the objects to the database. [To jest bardzo skrótowe.]

Chapter 7

Conclusions

Wyniki są b. zadow. bo soft może konkurować z innymi.
Use special environment for inline code, eg **descriptor** or **descriptor_gaussian**.

Bibliography

[1] G. Aad et al. 'Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC'. In: *Physics Letter B* 716.1 (2012), pp. 1–29.

[2] David Trigano et al. *Microsoft Azure SQL Database – Dynamic data masking*. URL: `https://docs.microsoft.com/en-us/azure/azure-sql/database/dynamic-data-masking-overview` (visited on 21/01/2022).

[3] Johan Haleby et al. *REST Assured – Documentation*. URL: `https://rest-assured.io` (visited on 21/01/2022).

[4] Les Hazlewood et al. *jjwt – GitHub Repository*. URL: `https://github.com/jwtk/jjwt` (visited on 21/01/2022).

[5] Richard North et al. *Testcontainers*. URL: `https://www.testcontainers.org` (visited on 21/01/2022).

[6] Jens Albrecht, Marc Fiedler and Tim Kiefer. 'A Rule-Based Approach to Local Anonymization for Exclusivity Handling in Statistical Databases'. In: *Privacy in Statistical Databases: UNESCO Chair in Data Privacy International Conference*. Ed. by Josep Domingo-Ferrer and Mirjana Pejić-Bach. Springer Publishing, 2016, pp. 81–93. ISBN: 978-3319453804.

[7] Jens Albrecht, Marc Fiedler and Tim Kiefer. 'Robust De-anonymization of Large Sparse Datasets'. In: *Privacy in Statistical Databases: UNESCO Chair in Data Privacy International Conference*. Ed. by Josep Domingo-Ferrer and Mirjana Pejić-Bach. Springer Publishing, 2016, pp. 81–93. ISBN: 978-3319453804.

[8] Luk Arbuckle and Khaled El Emam. *Building an Anonymization Pipeline: Creating Safe Data*. O'Reilly Media, Inc, 2020. ISBN: 978-1492053439.

[9] Narayanan Arvind and Shmatikov Vitaly. *Robust de-anonymization of large sparse datasets: a decade later*. URL: https://www.cs.princeton.edu/~arvindn/publications/de-anonymization-retrospective.pdf (visited on 21/01/2022).

[10] Arshdeep Bahga and Vijay Madisetti. *Big Data Science and Analytics: A Hands-On Approach*. Vpt, 2016. ISBN: 978-0996025539.

[11] Legion of the Bouncy Castle Inc. *The Legion of the Bouncy Castle Java Cryptography APIs*. URL: https://www.bouncycastle.org/java.html (visited on 21/01/2022).

[12] Brendan Burns. *Designing Distributed Systems: Patterns and Paradigms for Scalable, Reliable Services*. O'Reilly Media, Inc., 2018. ISBN: 978-1491983645.

[13] Damien Carey. *Celebrating 25 Years of Java*. URL: https://blogs.oracle.com/oracleuniversity/post/celebrating-25-years-of-java (visited on 21/01/2022).

[14] Scott Chacon and Ben Straub. *Pro Git*. Apress, 2014. ISBN: 978-1484200773.

[15] Inc. Cisco Systems. *Cisco Annual Internet Report (2018–2023)*. Tech. rep. URL: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.pdf (visited on 21/01/2022).

[16] Inc. Cisco Systems. *The Zettabyte Era Officially Begins*. URL: https://blogs.cisco.com/sp/the-zettabyte-era-officially-begins-how-much-is-that (visited on 21/01/2022).

[17] European Comission. *Charter of Fundamental Rights of the European Union*. URL: https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:12012P/TXT&from=EN (visited on 21/01/2022).

[18] European Comission. *General Data Protection Regulation - Recital 1*. URL: https://gdpr-info.eu/recitals/no-1 (visited on 21/01/2022).

[19] European Comission. *General Data Protection Regulation - Recital 26*. URL: https://www.privacy-regulation.eu/en/recital-26-GDPR.htm (visited on 21/01/2022).

[20] European Comission. *General Data Protection Regulation - Recital 6*. URL: https://gdpr-info.eu/recitals/no-6 (visited on 21/01/2022).

[21] European Comission. *General Data Protection Regulation – Art. 83 – General conditions for imposing administrative fines*. URL: https://gdpr-info.eu/art-83-gdpr/ (visited on 21/01/2022).

[22] European Comission. *General Data Protection Regulation – Art. 83 – Lawfulness of processing*. URL: https://gdpr-info.eu/art-6-gdpr/ (visited on 21/01/2022).

[23] Oracle Corporation. *Oracle Data Masking and Subsetting*. URL: https://www.oracle.com/security/database-security/data-masking (visited on 21/01/2022).

[24] Derek DeJonghe. *NGINX Cookbook: Advanced Recipes for High-Performance Load Balancing*. O'Reilly Media, Inc., 2020. ISBN: 978-1492078487.

[25] Divante. *Anonymizer – Documentation*. URL: https://github.com/DivanteLtd/anonymizer (visited on 21/01/2022).

[26] Inc. Docker. *Docker overview*. URL: https://docs.docker.com/get-started/overview (visited on 21/01/2022).

[27] Josep Domingo-Ferrer and Jordi Soria-Comas. 'Anonymization in the Time of Big Data'. In: *Privacy in Statistical Databases: UNESCO Chair in Data Privacy International Conference*. Ed. by Josep Domingo-Ferrer and Mirjana Pejić-Bach. Springer Publishing, 2016, pp. 57–68. ISBN: 978-3319453804.

[28] Inc. Domo. *Data Never Sleeps 6.0*. Tech. rep. URL: https://www.domo.com/assets/downloads/18_domo_data-never-sleeps-6+verticals.pdf (visited on 21/01/2022).

[29] Khaled El Emam and Luk Arbuckle. *Anonymizing Health Data: Case Studies and Methods to Get You Started*. O'Reilly Media, Inc, 2020. ISBN: 978-1449363079.

[30] Gamma Erich et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994. ISBN: 978-0201633610.

[31] Can Eyupoglu et al. 'An Efficient Big Data Anonymization Algorithm Based on Chaos and Perturbation Techniques'. In: *Entropy* 20.5 (2018), p. 373.

[32] Facebook. *Spring – Projects*. URL: `https://spring.io/projects` (visited on 21/01/2022).

[33] Facebook. *Yarn – Introduction*. URL: `https://yarnpkg.com/getting-started` (visited on 21/01/2022).

[34] Roy T. Fielding and Julian F. Reschke. *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content*. URL: `https://datatracker.ietf.org/doc/html/rfc7231` (visited on 21/01/2022).

[35] Carl Folkesson. 'Anonymization of directory-structured sensitive data'. MA thesis. Linköping University, 2019.

[36] Apache Software Foundation. *Apache Kafka – uses*. URL: `https://kafka.apache.org/uses` (visited on 21/01/2022).

[37] Apache Software Foundation. *Maven – Introduction*. URL: `https://maven.apache.org/what-is-maven.html` (visited on 21/01/2022).

[38] The Apache Software Foundation. *Apache Commons Lang – Documentation*. URL: `https://commons.apache.org/proper/commons-lang` (visited on 21/01/2022).

[39] Martin Fowler. *Presentation Domain Data Layering*. URL: `https://martinfowler.com/bliki/PresentationDomainDataLayering.html` (visited on 21/01/2022).

[40] Aris Gkoulalas-Divanis and Grigorios Loukides. *Anonymization of Electronic Medical Records to Support Clinical Analysis*. Springer Publishing, 2012. ISBN: 978-1461456698.

[41] PostgreSQL Global Development Group. *PostgreSQL Documentation – JSON types*. URL: `https://www.postgresql.org/docs/14/datatype-json.html` (visited on 21/01/2022).

[42] Richie Koch. *Data anonymization and GDPR compliance: the case of Taxa 4×35*. URL: https://gdpr.eu/data-anonymization-taxa-4x35 (visited on 21/01/2022).

[43] Chris Lee. *Integrating Web Analytics with your SAP Commerce Cloud Storefront – An Overview*. URL: https://www.sap.com/cxworks/article/2589633935/integrating_web_analytics_with_your_sap_commerce_cloud_storefront_an_overview (visited on 21/01/2022).

[44] Patrick Li. *JIRA Agile Essentials*. Packt Publishing, 2015. ISBN: 978-1784394912.

[45] Google LLC. *V8 JavaScript Engine*. URL: https://v8.dev/docs (visited on 21/01/2022).

[46] Red Gate Software Ltd. *Flyway – Documentation*. URL: https://flywaydb.org/documentation (visited on 21/01/2022).

[47] TeskaLabs Ltd. *TurboCat.io*. URL: https://teskalabs.com/products/turbocat.io (visited on 21/01/2022).

[48] Bernard Marr. *Big Data in Practice: How 45 Successful Companies Used Big Data Analytics to Deliver Extraordinary Results*. John Wiley and Sons Ltd., 2016. ISBN: 978-1119231387.

[49] Robert C. Martin. *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Pearson, 2017. ISBN: 978-0134494166.

[50] Robert C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*. Pearson, 2008. ISBN: 978-0132350884.

[51] Robert C. Martin. *The Obligation of the Programmer*. 2014. URL: https://blog.cleancoder.com/uncle-bob/2014/11/15/WeRuleTheWorld.html (visited on 21/01/2022).

[52] Vlad Mihalcea. *Hibernate Types – GitHub repository*. URL: https://github.com/vladmihalcea/hibernate-types (visited on 21/01/2022).

[53] Vlad Mihalcea. *High-Performance Java Persistence*. Independently published, 2016. ISBN: 978-9730228236.

[54] Andrew Moore. *The GDPR and Managing Data Risk*. John Wiley and Sons, Ltd., 2018. ISBN: 978-1119487746.

[55] Prosus N.V. *Stack Overflow Developer Survey 2017*. Tech. rep. URL: https://insights.stackoverflow.com/survey/2017 (visited on 21/01/2022).

[56] Prosus N.V. *Stack Overflow Developer Survey 2021*. Tech. rep. URL: https://insights.stackoverflow.com/survey/2021 (visited on 21/01/2022).

[57] Christoph Nakazawa, Sebastian McKenzie and Jamie Kyle. *Yarn: A new package manager for JavaScript*. URL: https://engineering.fb.com/2016/10/11/web/yarn-a-new-package-manager-for-javascript (visited on 21/01/2022).

[58] Neha Narkhede, Gwen Shapira and Todd Palino. *Kafka: The Definitive Guide: Real-Time Data and Stream Processing at Scale*. O'Reilly Media, Inc., 2017. ISBN: 978-1491936160.

[59] Petter Ødegård. 'Data Anonymization for Research'. MA thesis. Norwegian University of Science and Technology, 2019.

[60] OpenAIRE. *Amnesia – Documentation*. URL: https://amnesia.openaire.eu/about-documentation.html (visited on 21/01/2022).

[61] Nigel Poulton. *Docker Deep Dive: Zero to Docker in a single book*. Independently published, 2016. ISBN: 978-1521822807.

[62] Arjen Poutsma. *New in Spring 5.3: Improved Cron Expressions*. URL: https://spring.io/blog/2020/11/10/new-in-spring-5-3-improved-cron-expressions (visited on 21/01/2022).

[63] Fabian Prasser et al. *ARX – Data Anonymization Tool*. URL: https://arx.deidentifier.org (visited on 21/01/2022).

[64] Fabian Prasser et al. 'Flexible data anonymization using ARX—Current status and challenges ahead'. In: *Software: Practice and Experience* 50.7 (2020), pp. 1277–1304.

[65] Balaji Raghunathan. *The Complete Book of Data Anonymization: From Planning to Implementation*. Auerbach Publications, 2013. ISBN: 978-1439877302.

[66] Ron Ratovsky. *Getting Started with Swagger [I] – What is Swagger?* URL: https://swagger.io/blog/api-development/getting-started-with-swagger-i-what-is-swagger (visited on 21/01/2022).

[67] Inc. Red Hat. *Hibernate Documentation – ORM*. URL: https://hibernate.org/orm (visited on 21/01/2022).

[68] D.B. Resnik. 'Objectivity of Research: Ethical Aspects'. In: *International Encyclopedia of the Social and Behavioral Sciences*. Ed. by Neil J. Smelser and Paul B. Baltes. Oxford: Pergamon, 2001, pp. 10789–10793. ISBN: 978-0-08-043076-8. DOI: https://doi.org/10.1016/B0-08-043076-7/00157-1.

[69] Rolfje. *Anonimatron – Documentation*. URL: https://realrolfje.github.io/anonimatron/documentation (visited on 21/01/2022).

[70] Rathindra Sarathy and Krish Muralidhar. 'Perturbation Methods for Protecting Numerical Data: Evolution and Evaluation'. In: ed. by Ranajit Chakraborty, C.R. Rao and Pranab Sen. Vol. 28. Handbook of Statistics. Elsevier, 2012, pp. 513–531. DOI: https://doi.org/10.1016/B978-0-44-451875-0.00019-1. URL: https://www.sciencedirect.com/science/article/pii/B9780444518750000191.

[71] Rathindra Sarathy and Krish Muralidhar. 'Protecting Numerical Confidential Data using Data Shuffling: A Demonstration of Effectiveness of Approach and Flexibility of Delivery'. In: *Federal Committee on Statistical Methodology Research Conference*. 2009.

[72] SAP SE. *SAP Data Services – Data Mask – Documentation*. URL: https://help.sap.com/viewer/e54136ab6a4a43e6a370265bf0a2d744/4.2.14/en-US/6073fc460d794b699e90ca8ce5bb095c.html (visited on 21/01/2022).

[73] Sanjay Sharma. *Data Privacy and GDPR Handbook*. John Wiley and Sons Ltd., 2019. ISBN: 978-1119594246.

[74] M. S. Simi, Nayaki K. Sankara and Elayidom M. Sudheep. 'An Extensive Study on Data Anonymization Algorithms Based on K-Anonymity'. In: *IOP Conference Series: Materials Science and Engineering* 225.1 (2017).

[75] Personal Data Protection Commission of Singapore. *Guide to Basic Data Anonymisation Techniques*. 2018. URL: https://www.pdpc.gov.sg/-/media/Files/PDPC/PDF-Files/Other-Guides/Guide-to-Anonymisation_v1-(250118).pd (visited on 21/01/2022).

[76] SmartBear Software. *REST API Documentation Tool – Swagger UI*. URL: https://swagger.io/tools/swagger-ui (visited on 21/01/2022).

[77] Jordi Soria-Comas and Josep Domingo-Ferrer. 'Big Data Privacy: Challenges to Privacy Principles and Model'. In: *Data Science and Engineering* 1.1 (2015), pp. 21–28.

[78] European Data Protection Supervisor. *Introduction to the hash function as a personal data pseudonymisation technique*. URL: https://edps.europa.eu/data-protection/our-work/publications/papers/introduction-hash-function-personal-data_en (visited on 21/01/2022).

[79] Łukasz Szydło. *Boiling Frogs: Modularity – The Final Frontier*. 2018. URL: https://www.youtube.com/watch?v=2oJrjyp7GHE (visited on 21/01/2022).

[80] Colin Tankard. *Big data security*. URL: https://www.sciencedirect.com/science/article/abs/pii/S1353485812700636 (visited on 21/01/2022).

[81] IT Governance Privacy Team. *EU General Data Protection Regulation (GDPR): An Implementation and Compliance Guide*. IT Governance Publishing Ltd., 2020. ISBN: 978-1787782495.

[82] Linus Torvalds. *Tech Talk: Linus Torvalds on git*. 2007. URL: https://www.youtube.com/watch?v=4XpnKHJAok8 (visited on 21/01/2022).

[83] Catalin Tudose. *JUnit in Action*. Manning, 2020.

[84] Paul Voigt and Axel von dem Bussche. *The EU General Data Protection Regulation (GDPR): A Practical Guide*. Springer Publishing, 2017. ISBN: 978-3319579580.

[85] Craig Walls. *Spring in Action*. Manning, 2018. ISBN: 978-1617294945.

[86] Ben Wolford. *Does the GDPR apply to companies outside of the EU?* URL: https://gdpr.eu/companies-outside-of-europe (visited on 21/01/2022).

[87] Frank Zametti. *Modern Full-Stack Development: Using TypeScript, React, Node.js, Webpack, and Docker*. Apress, 2020. ISBN: 978-1484257371.

[88] Sherali Zeadally and Mohamad Badra. *Privacy in a Digital, Networked World: Technologies, Implications and Solutions*. Springer Publishing, 2015. ISBN: 978-3319084718.

Appendices

Index of abbreviations and symbols

DNA deoxyribonucleic acid

List of additional files in electronic submission

Additional files uploaded to the system include:

- source code of the application,
- test data.

List of Figures

List of Tables