Architecture Document
DataForge

Guido Baels (s2676478) - Chris Bleeker (s2462648)
Asen Pantov (s2853418) - Joris Vlaar (s2607867)
Pascal Vriend (s2739046) - Bas van de Weerd (s2372975)

DataForge Group 4

Table of contents

1 - Introduction

The following document will detail the architecture for a new all-encompassing data analysis tool for professionals and novices called DataForge. The vision of DataForge is that anybody can use the application for data analysis and visualisations with any data source, while it also provides a platform for collaboration between users. First, the scope of DataForge will be presented. The scope shows how we envision DataForge should look like as a final product as well as two top level views of the system. The next section explores the related systems to give an idea of what the market currently looks like and how DataForge is able to combine these different areas of the market into one application that surpasses all of the current options. This is followed by a section on trends and developments that are relevant for DataForge. This section also explains which of these trends are suitable to be implemented within DataForge and provides reasoning why some trends will not be implemented.

Following this section, there will be a section on stakeholders where it will be explained what stakeholders are involved in DataForge and how they contribute to the system. The concerns of these stakeholders will be discussed to form an idea on the functionalities that need to be present in DataForge. Furthermore, the way stakeholders will communicate with and amongst each other is also discussed in this section. After that, there will be a section about the business goals. This section will highlight what the interests of the company are and what DataForge strives for. Then, there will be a section about the quality attributes that are of most value to DataForge and its stakeholders. These quality attributes are then also explained in more detail and prioritised from high to low, so that we can differentiate and make choices in the design to improve these important attributes, while others might suffer from that. There is an extra subsection for this that highlights the trade-offs between quality attributes and explains some of the choices we made and what the consequences of these choices are on some of those quality attributes.

This leads to the next section, which is about the requirements for DataForge. Here, the functional and nonfunctional requirements for the system have been derived from the stakeholder’s needs and the quality attributes that are the most important. Some of the requirements will benefit multiple stakeholders and encompass more than one quality attribute. After that, there is a section that uses the quality attributes and requirements to make scenarios that are relevant to the system. These scenarios help describe situations that can happen during the lifespan of DataForge. For each scenario, tactics and patterns are listed that could help solve the situation. There is a short explanation on how (some of) these patterns and tactics will be used to ensure that the system works as intended.

Then, moving on to the next section, there will be a detailed description of the views of our system. These include the data, component, functional, deployment and development view. These views should address all of the stakeholder concerns. The data view is there to help visualise and give more insight into where data comes from and how it moves through the system. Then the component view regards the function and interaction of the DataForge system's components (at the highest level and their subsystems). It also goes deeper into how the modularity in the system works. Furthermore, the functional view should address most of both the compliance teams, experienced data scientists' and newcomers' concerns. Then, the deployment view describes the workings of our microservice architecture. So in short, how the analysis backend and the app backend work with each other and how this in turn works with the users and the output/input. Finally, the development view should address the concerns of the developers and how different teams work together. Also it will explain how the modularity and incremental design are addressed by the systems architecture. Then the final section is about the specifications of the system. These include formal, requirement, interface, functional, technology and data specifications.

2 - Scope

DataForge will allow its users to process data from any combination of their data sources. With automatically adjusted user interfaces, DataForge will bridge the gap between expert and novice data analysts and will provide a top of the line user friendly experience to be used by everyone. Dataforge aims to be accessible for as many users' systems as possible. To achieve this goal, DataForge will be available as a web interface requiring low processing power on the users' end. Through this web interface users can import all forms of data. Once data is uploaded to the DataForge system, users can perform all types of actions on the data. Users will be able to analyze data and generate new data insights and represent this information most fittingly and appealingly. Actions include but are not limited to: cleaning data, analyzing data and creating data visualisations in the form of interactive dashboards. In addition newly created representations of source data generated will also be able to be exported in any of the supported input data formats. DataForge is not meant to serve as a primary method for the storage of data. DataForge does not aim to replace a company's pre-existing database or serve as a cloud storage provider. Additionally, DataForge will never modify source data stored on external databases. This ensures data security when used by novice data analysts. Finally, users will be able to work together in a collaborative environment. This collaborative environment is up to modern standards providing features for project communication and a ticket system. These core functionalities are all presented in the use case diagram in figure 2.1 which also presents a top level functional view of the system.

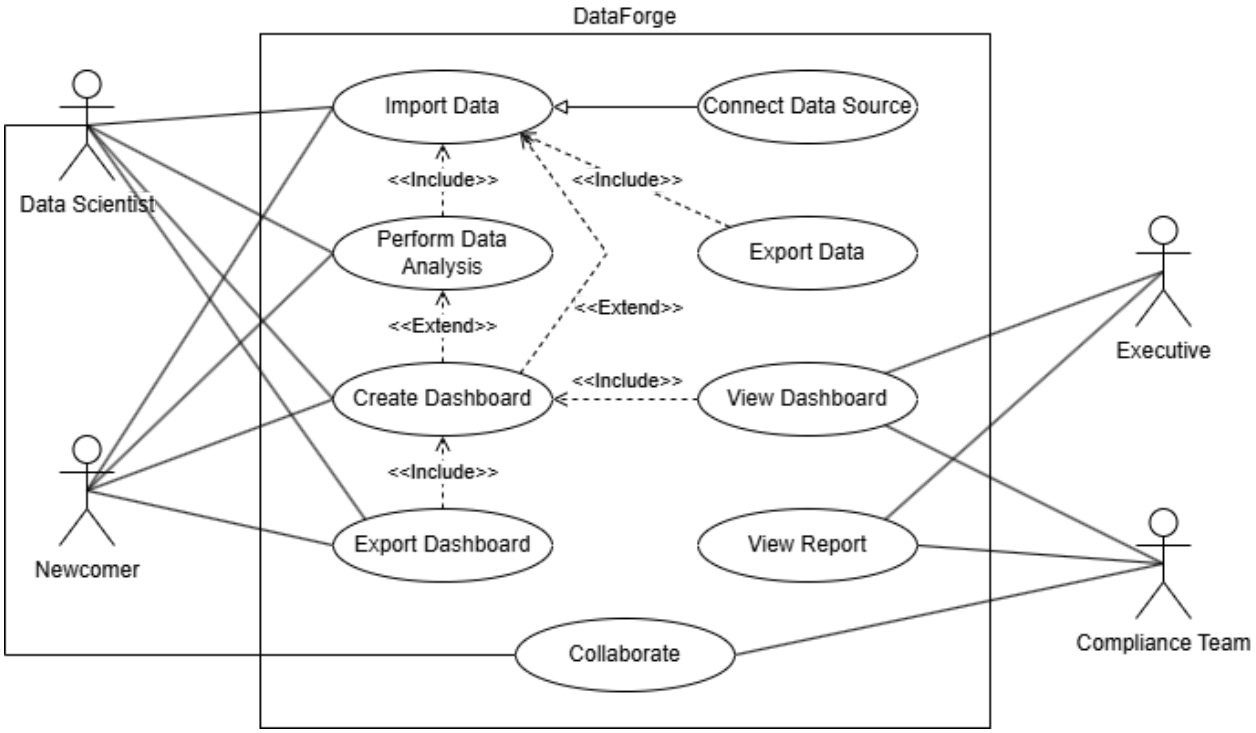

Figure 2.1: Core Use Case Diagram

Figure 2.2 shows a top level overview of the system's components. The user interacts with the system through the web interface and selects their own data input, and imports it into the system. The application then stores this data temporarily in the analysis backend. Analysis of the data are all performed on the DataForge Analysis Backend. This way DataForge provides all users with the required processing power for analyzing the data further increasing its accessibility. DataForge services will be scalable to provide its services to as many users as possible and leave room for future upscaling.

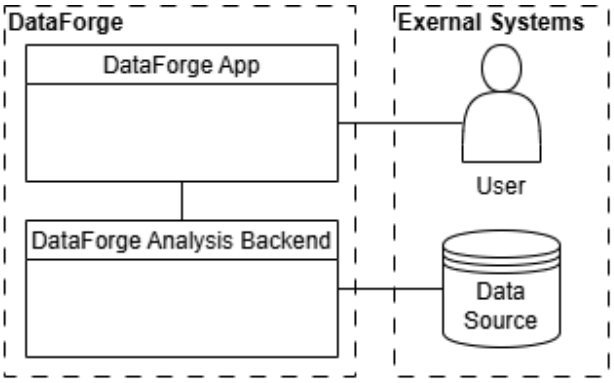

Figure 2.2: Overview of the main system components

3 - Related Systems

This chapter will highlight some of the important related systems to DataForge and will analyse their significance to the DataForge system. Starting with PostgreSQL [14], PostgreSQL is an open-source database system that helps with the visualisation and processing of object-relational data. PostgreSQL is very similar and thus closely related to DataForge, in the sense that it has part of the functionality that is envisioned for DataForge. Namely the processing and a little visualisation of tabular data.

Other highly relevant related systems are Apache Kafka and Apache Flink [15]. Together these can be used for the streaming and processing of data from IoT devices and API's. Specifically live data. Here, Kafka is used for the streaming of data from one platform to another, while Flink is used for processing data, which can be in the form of analytics, or computations after which the results can be saved in a file. Again, these systems relate to DataForge in the sense that it has a similar functionality where it processes data, which means DataForge could use similar techniques in the future.

Jupyter Notebook [16] is another example of a system that can be related to DataForge. This is because part of the functionality of DataForge is to visualise processed data in a manner that is good for the user. This can easily be done in a Jupyter Notebook when it is combined with Matplotlib for example.

A good example of IoT devices that users might want to process data from are Arduino [17] devices. These are some of the most popular devices used in IoT and can range from input sensors and output devices to (micro)controllers. DataForge needs to be able to receive data from these devices and process it accordingly.

An example of a related system that has already tried to combine some of the aspects mentioned above is Tableau [18]. Tableau is a platform that mainly focuses on the visualisation of analytical data, but it also allows for data sharing, processing and working in a collaborative manner with businesses as their target user. It is also able to use a wide range of data sources, such as Excel files, SQL databases. Tableau is a useful system for DataForge because it embodies part of the concept of the application. DataForge would be a better version of Tableau, where even more data sources are available, such as live data from IoT devices, with more personalisation options.

Similar to Tableau there is SAP analytics [19], which is even more focussed on businesses and user satisfaction. SAP analytics is also used for analytics, which can then be used for predictions in the future. This is useful for business owners that want to get more insight into their data. SAP is also easy to connect to smartphones which can be useful for businesses if they want to share their data with others during meetings. DataForge can learn from this example in the hopes of increasing the collaboration options for its customers.

Even though there already exist quite some systems that are similar to what the DataForge application has to offer, DataForge is still useful. This is because DataForge aims to combine all these aspects and put them in one single place.

4 - Trends and Developments

In this chapter, some of the recent developments related to the industry of the application will be touched upon, by briefly explaining what the development is, the importance it holds and whether it is useful to take into account for our system.

First of all, a very impactful development that has exponentially grown [13] in recent years is artificial intelligence. AI is mainly relevant to the functionality of DataForge because it influences how the application has to work. The rise of artificial intelligence is of high importance to the system because AI models need to be trained on (a lot of) data. This means that there is an increase in the demand for data management and processing tools. Next to that, AI could also be used for data management itself. In DataForge it can be used to recognise input data and provide visualisation accordingly. This could greatly improve the usability and performance of the application. Specifically, for the data scientist users, it could help with the automation of data cleaning and feature extraction, which saves the user a lot of time.

Another trend that is a bit less recent, but still growing, is the IoT. The demand for IoT devices in households, as well as companies is still growing. People want the devices in their environment to be able to communicate with each other and work together to provide smart solutions to make their lives or work better. To get insight into IoT devices, especially sensors, the data that these devices provide needs to be processed. This is why the popularity of IOT devices is relevant to DataForge. It needs to be taken into account in the design and functionality of the system that it needs to be able to receive and process data of such IOT devices, because that is what the customer wants.

Furthermore, cybersecurity has been an important development for years and with the increase in the amount of data that has to be processed it is still an important development today. We need to adhere to the GDPR at least and thus we need to take cybersecurity, data protection, and privacy into account. Privacy and security is important, because the user needs to feel safe and be confident that their data is protected. Also, data should be processed in an ethical and secure way. This field mainly applies to the design of the application. To realise this, the system uses the security by design principle. This means the system is designed with security as a high priority. In practice the system is not made any more complex than necessary, because this makes it easier to secure the system. Next to that, the system will be secure by default and be tested regularly. In a system with databases and servers such as DataForge this entails that there is secure storage, encryption and authentication. Also, it is more efficient to go for security practices that are known to work instead of using new developments in encryption for example, because that would be costly for the performance of the application.

Next to that, a trend in data processing is edge computing. Edge computing is a partial solution for the exponential increase in data that needs to be processed because it is a practice where data is processed close to the source, instead of having to go to many different channels. This saves on processing time and thus is more efficient. However, unfortunately for DataForge, this concept is not of use, because it is meant to be one big platform and is kind of the opposite of edge computing. This should not be a problem, because even though edge computing is a big development, a central platform where data can be processed is still of value for a lot of users.

Finally, a trend that is still mostly in the development phase is quantum computing [20]. The basic idea of a quantum computer is that it, unlike a traditional computer, uses qubits instead of bits. This allows it to perform more calculations at the same time than a normal computer, which can be very useful in some specific situations. Although some breakthroughs have been made in this field and it has proven to be efficient in, for example, artificial intelligence, the development is not (yet) of use to DataForge. The development is still very much in its experimental phase and there is no indication that quantum computing as of now can speed up the process of data processing for all types of data. Therefore the development will be ignored during the design of the application.

5 - Stakeholders

Identifying the stakeholders is an important step in designing a successful software architecture for DataForge. Through this process, we also get an initial feel of the basic requirements indirectly by mentioning what is the relationship of the different stakeholders with the system. The next subsection includes the most important DataForge stakeholders who were classified by their priority level, according to their importance and influence within the system.

- **Data Scientists** <high>:
    - People with expertise in statistics, computer programming, and machine learning who develop models and processes for retrieving valuable information from large data sets [1].
    - They would make use of the platform in advanced data modelling, machine learning model integration, deep analysis, and prediction of insights.
- **Newcomers** <high>:
    - People who have just started learning about data processing and analysis concepts
    - Get help from instructions and user-friendly interfaces to bring out their data processing ideas and to sharpen their skills.
- **Compliance teams** <high>:
    - Teams who are accountable for ensuring regulatory adherence within their organisation.
    - They need compliance monitoring, reporting features, and the ability to track data usage and modifications for the purpose of regulatory compliance.
- **Business owners** (c-level executives) <high>:
    - Senior executives overseeing strategic decision-making in the organisation
    - Access high-level insights, dashboards and compliance summaries to inform. strategic decisions.
- **Developers** <medium>:
    - Engineers who invent, construct, and manage the platform
    - Create new features, solve bugs, and boost system performance.
- **Support and Maintenance Teams** <medium>:
    - Teams providing technical support and platform maintenance.
    - Handling troubleshooting, performing updates, and assisting users with technical issues related to DataForge.
- **Government** <low>:
    - Government entities concerned with regulation, compliance, and public data management.
    - Review our compliance features and data handling practices to make sure they are in line with the respective laws and regulations
    - The government could also be a potential customer since they have to process a lot of data too.

- **Competitors** <low>:
    - Other companies or entities offering data processing platforms or similar solutions.
    - Analyse DataForge's features and innovations for their internal reasons and potentially collaborate on industry standards.
    - Performance of competitors might influence the availability of clients for DataForge.

To further visualise and elaborate on the stakeholders selection and prioritisation, a power-interest grid has been constructed (figure 5.1). It is utilised to show the level of power/influence and the level of stakeholder interest for each stakeholder. In categorizing stakeholders in the power-interest matrix, we consider two dimensions with one being how far they can influence the direction of the project, while the second is the level to which they are interested or have a stake in the outcomes of the project.

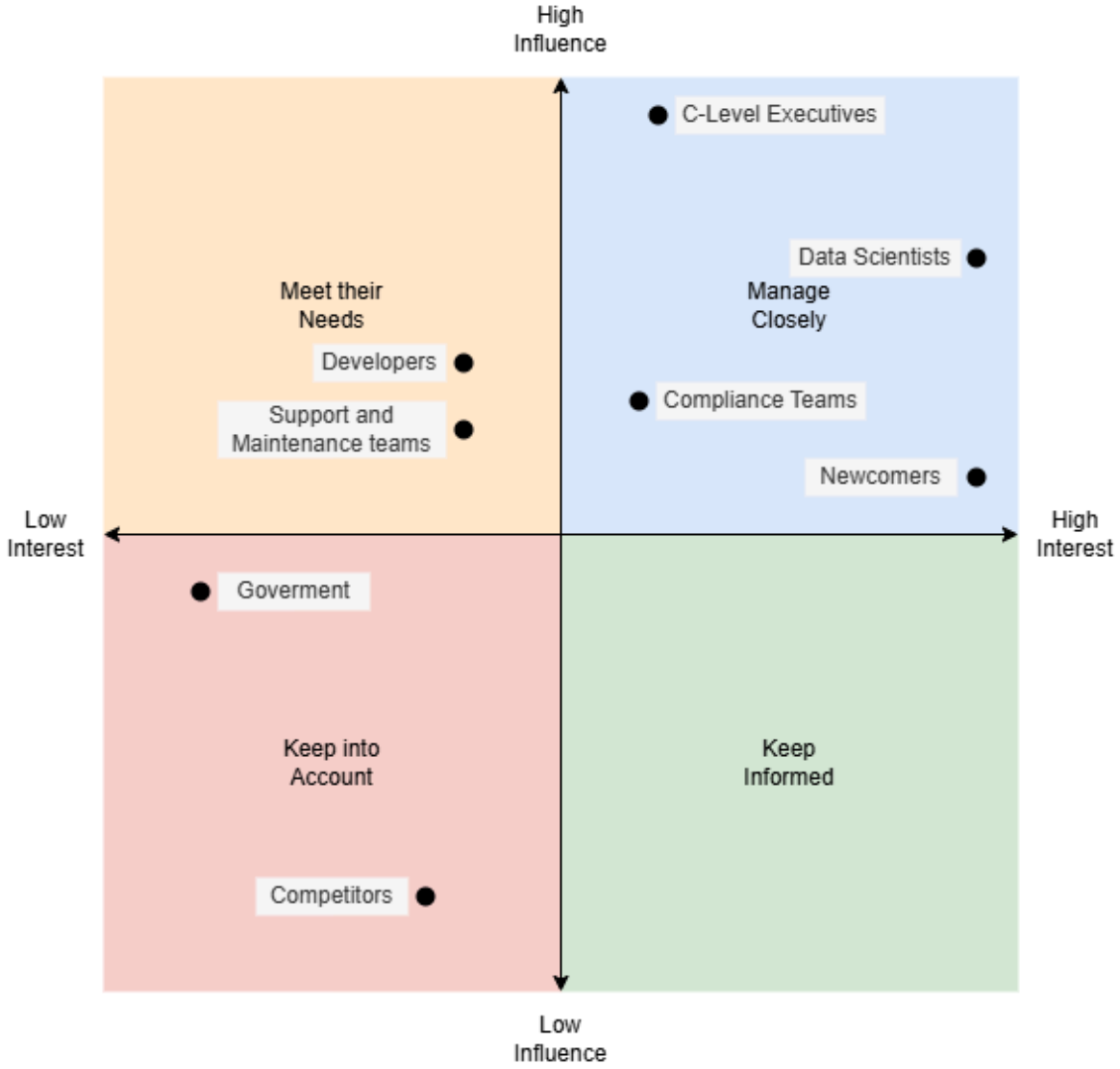

Figure 5.1: Stakeholder power-interest matrix

The top-right quadrant consists of the high-power, high-interest stakeholders. Starting with the data scientists, they have significant interest in the platform through their advanced analytical work. New entrants are less technologically savvy but remain a core user class. These two users can influence the system by means of user feedback. Compliance teams are also an expert user which strongly affects DataForge's design and operational choices. Their interest is high since any missteps could jeopardise the platform's viability. The C-level executives control resource allocation, strategic vision, and long-term direction; therefore, their power and interest are as high as they can be since the platform directly influences business outcomes and competitive positioning.

The top-left quadrant consists of the medium-power, medium-interest stakeholders. Those are namely the developers and support and maintenance teams. They are key in constructing, maintaining, and refining this platform. They can affect technical decisions and system stability, but often are focused on implementation details rather than strategic interests of a higher level. Their interest is stable-invested enough to create a working, efficient product-but not concerned with market strategy or long-term business trajectories.

The bottom-left quadrant includes the government entities and competitors. This quadrant is characterised by low power and low interest. It is important to mention that while an argument could be made why the government should be a high power stakeholder since they are capable of establishing regulatory frameworks, they assume a comparatively more passive role regarding the daily decision-making processes of the platform. They oversee compliance with laws from an objective position without direct interference over aspects, hence lesser direct contact with technical aspects of the platform. Competitors review DataForge's products from a distance and make their decisions about them but have no capability to intervene with design and development processes directly. They can only make judgments based on their positioning in the market but with no interference over the internal decision-making process or the features of the platform.

When thinking about the stakeholders, the stakeholder communication matrix is another effective technique that should not be overlooked. It is a tabular template that ensures that the right information gets to the right people at the right time, supports transparency, collaboration, and good decision making by focusing on what objectives relate to each stakeholder, what is communicated, when-or during which event-it shall be best to have this engagement, through which channel the information shall pass, and who shall deliver this news. This ensures that the wide range of users within DataForge-from data scientists, compliance teams, and executive management down to even the inexperienced user receives only the communications relevant to their unique interests and aligned with the purpose of the engagement. The relevant matrices can be observed below.

-	Data Scientists
Why (Objective)	Inform them about new/updated analytical features. Think with them on advanced ML workflows. Decide how best to integrate model
About What (Topics)	Advanced modeling and ML integration, Data cleaning, transformations, visualizations
When (Frequency)	During iterative dev cycles and when adding complex analysis capabilities
With What (Format)	Email updates and User testing sessions
By Whom (Role)	Development team

Figure 5.2: Data Scientists Stakeholder Communication Matrix

-	Newcomers
Why (Objective)	Inform them about onboarding and new user-friendly updates. Think about improving usability. Decide on essential features to aid learning
About What (Topics)	Onboarding instructions, simplified UI/UX, easy data pipelines and documentation
When (Frequency)	Regular user testing, after major UI/UX releases, during onboarding phases
With What (Format)	Email updates and tutorials
By Whom (Role)	Development Team

Figure 5.3: Newcomers Stakeholder Communication Matrix

-	Compliance Teams
Why (Objective)	Inform them on regulatory adherence. Think about upcoming compliance priorities. Decide on adjustments needed for new regulations
About What (Topics)	Compliance monitoring/reporting, security controls, audit trails, regulatory obligations
When (Frequency)	Periodic reviews (quarterly or as needed) or whenever new compliance rules emerge
With What (Format)	Email updates and formal compliance reports
By Whom (Role)	Development Team

Figure 5.4: Compliance Teams Stakeholder Communication Matrix

5.1 - Concerns

Below we list some concerns per stakeholder all phrased as questions to help create an idea of what the system should look like to satisfy all of the stakeholders. We also connect these to a quality attribute(s).

Data Scientists:

- Does the system have all the functionality I need?
    - Functionality
- Can my source data be imported into the system with minimal effort
    - Usability
- Can the system process data sufficiently fast?
    - Performance
- Can I integrate the system with existing tools or workflows?
    - Interoperability
- Can the system handle large amounts of source data?
    - Performance, Scalability
- How long does it take to become an expert at using the system?
    - Learnability
- Can data be visualised according to my needs?
    - Usability

Newcomers:

- Does the system have all the functionality I need?
    - Functionality
- Is the system intuitive and easy to learn?
    - Learnability
- Can simple tasks be performed with limited knowledge?
    - Usability, Learnability
- Is the system fun to use?
    - Usability

Compliance teams:

- How sure can we be that there are no unintended bugs or flaws in the system threatening data integrity?
    - Testability
- Can every processed data point be traced back to its source?
- Does the system comply with rules and regulations?
    - Auditability
- How is the data protected when intending to modify it?
- Can bugs in one part of the system affect other parts of the system?
    - Resilience
- How secure is the system against targeted attacks?
- How secure is the infrastructure hosting the system?

    - Security

Business owners (c-level executives):

- The system should provide a dashboard for monitoring data, how will the dashboards be implemented?
    - Functionality
- What mechanisms are in place to ensure the data displayed in the dashboards is accurate and free of errors?
- What role-based access controls will ensure that sensitive data is visible only to authorised personnel?
    - Security
- Can the system track and log who accesses dashboard data?
    - Functionality, Security
- How will the system ensure that dashboards load quickly, even with large datasets or concurrent users?
    - Performance
- The data shown in the dashboards should be updated, how will the data be integrated with the dashboards?
    - Usability

Developers:

- How complex are the required features and are the resources (such as time, tools, expertise, etc) sufficient for deployment/implementation?
    - Deployability
- How easy is it to debug, troubleshoot and update the system?
- What technologies are used for version control and dependency management?
- Are there clear guidelines for development, such as rules for documentation or code structure?
    - maintainability
- How will the system support growing demands such as uploading big data sets and more (simultaneous) users of the platform?
- How will performance profiling and optimisation be performed?
    - performance/scalability
- How does the system apply automated testing and continuous integration?
- How easily can tests, such as unit, integration and system testing be implemented?
    - Testability
- How secure is the development environment against attacks or accidental breaches?
- What regulations are there for secure coding practices?
    - Security
- How well does the system integrate with other software frameworks/libraries and APIs?
    - interoperability

Support and Maintenance team:

- The platform needs to be maintained so that it will keep on working correctly also in the future. How will we realise this?
    - Maintainability, Functionality, reliability and availability
- How do we keep the user satisfied?
    - Maintainability, Extensibility, Adaptability
- How can we get feedback from the user to improve our system?
    - Testability, Usability

Government:

- The privacy and rights of the users need to be protected. How will this be done?
    - Security, Auditability

5.2 - Communication

Effective, transparent, and focused communication means that DataForge will keep the technical staff, executives, regulatory bodies, and end users who are concerned continuously informed, involved, and up-to-date with regard to the architectural framework and dynamic implementation path. Communication strategies shall be encapsulated by DataForge within its systematic approach to incremental and agile software development. This integration will promote iterative feedback mechanisms enabling the development team to quickly address the emergent requirements, tinker with the architecture, and integrate the stakeholders' perspectives without significant delays.

With DataForge, communication with the stakeholders is done transparently and clearly, enabling feedback, and aligning interests among parties. Transparency will involve routine updating of stakeholders on the attainment of milestones, architectural updates, and overall system roadmap. That is achieved through regular reports of minutes of work undertaken, upcoming features, regulatory measures taken, and integration. Clarity means that such technical and architectural information is available in a digestible, role-appropriate manner: detailed diagrams and interface specifications for the developers, tutorials for the data scientists and newcomers, and thoroughly documented adherence to standards with traceable audit trails for the regulators. Alignment assures that the teams have a common view of scope, timelines, and standards for projects. This enables market growth, regulatory compliance, and satisfies the needs of both beginning and advanced users. Analysis and review are constantly done by DataForge to ensure that the architectural plans and their implementation are aligned with these.

With all of this in mind, it is important to go into the specifics of the communication channels and tools one by one:

Slack
Slack is intended to be used as the main internal communication channel to achieve fast, real-time communication among developers, support and maintenance, and managerial staff. Special interest channels dedicated to discussing architectural concerns, integrations in question, and performance appraisals will have stakeholders quickly resolve questions, share code snippets, or raise concerns. Notably, it is already utilised as confirmed by a company representative.

Jira
This is a centralised task and issue management system that is going to keep track of development tasks, bugs, feature requests, and enhancements of the architecture. Importantly, the team of DataForge is already getting used to using this system so it should be just further reinstated. Developers and support staff shall use Jira for creating, prioritising, and assigning tickets concerning architectural components. These tasks then can link up to confluence pages or other documentation, therefore giving an integrated view of both technical details and the status of their implementation.

E-mail and external dashboards
Formal communication to external stakeholders, such as compliance teams and government organisations, will be provided through email and secure read-only dashboards. Summary reports, run on a regular cadence and emailed out, will include new features deployed, architecture changes, how to validate compliance, and performance metrics of said deployment. Role-based access to dashboards or portal views enables stakeholders to understand the system's health, compliance with regulations, and attainment of strategic milestones without requiring deep technical knowledge.

Documentation Repositories
A properly maintained and accessible repository for architectural documentation, design decisions, integration guides, and best practices will serve as the recorded knowledge base on the project. This may be implemented as a Git-based wiki or any other version control-based documentation base. These types of repositories provide consistency among developers, support, compliance, and end users as to system components and interfaces, as well as approved system or coding standards. Continuous updating of these repositories will indicate the constant development of the system, providing a means for stakeholders to get up-to-date information with ease.

In-app channels and questionnaires for feedback
Direct input channels will be available through the app for newcomers and data scientists collaborating with DataForge. Embedded support menus, contextual help pages, and speedy feedback forms should further encourage users to report problems, request new visualisations, or make suggestions toward improvements in data processing workflows. Periodic surveys launched in the aftermath of major releases or feature rollouts will help understand user satisfaction, recognise training gaps, and inform UI/UX improvements.

6 - Business Goals

The following list of business goals presented covers the interests of the company extensively. They are numbered in order to be more easily referenced when elaborated upon below.

1. Stay on the course of sustained growth, which implies long-term viability in the data processing sector.
2. Secure profitability and meet the financial goals that stakeholders and employees have set.
3. Create a platform that is both quality, reliable and secure to the users, thus the users will trust it.
4. Be agile and quick to adapt to the technological changes and the market trends.
5. Improve the user experience (UX) through the creation of Interfaces that are adaptable to clients' needs.
6. Utilise a strong data security plan and a data compliance program that are of high standards
7. Ensure high performance and reliability

The first business goal can be assigned to the category of organisational growth and continuity. The stakeholders related to it are the C-level executives (responsible for strategic planning), developers (improvements to the platform contribute to growth) and the support and maintenance teams (help retain existing users and attract new ones). The goal emphasises DataForge staying competitive and expanding its market share in the data processing industry.

The second business goal can be attributed to the category of financial objectives. The related stakeholders are once again the executives, developers, and support and maintenance teams. The goal is a somewhat straightforward and standard financial objective whose completion enables the company to continue to operate, invest in new developments and provide financial rewards to its employees.

The third business goal overlaps with the categories of product quality and reputation as well as responsibility to employees, society, the government and shareholders. The secure aspect of the objective forms the relationship with the responsibility category. The stakeholders in this case are the data scientists, newcomers, compliance teams, government and also the competitors. All of the different users naturally have a relation to the goal, but the more interesting relation is with the government and competitors. The government is interested in the platform's compliance with laws whereas, for the competitors, a high-quality platform could set the benchmark in the industry, directly affecting them.

The fourth business goal is from the category of adapting to environmental changes over time. The related stakeholders are the developers, support and maintenance teams, and data scientists. The developers are affected because that would require them to implement new features and technologies promptly, similarly for the support and maintenance teams who must adapt to those changes efficiently. The data scientists are the segment of users benefiting most from cutting-edge tools and functionalities. This agility should allow DataForge to remain at the forefront of technological advancements in the industry.

Business goal 5 is another one from the category of product quality and reputation. The relevant stakeholders are data scientists, newcomers, compliance teams, and support and maintenance teams. An adaptable and user-friendly interface benefits the business by enhancing satisfaction across all types of users, from novices to experts.

The sixth aforementioned goal can be mainly attributed to the category of responsibility to employees, society, the government and shareholders. The compliance teams, government data scientists and newcomers are the stakeholders really related to this objective. The government is responsible for making certain that the platform adheres to regulations and protects the public interest, while the different users require assurance that their data is secure and privacy is maintained.

The last business goal can be assigned to the product quality and reputation category. The main related stakeholders are the developers. They are the ones responsible for creating an efficient platform that meets performance benchmarks. Handling demanding tasks without failures is essential and this goal embodies that need.

7 - Quality Attributes

Below we list the quality attributes that we found to be most important or relevant to our system, with a small explanation of what it is about and how we plan to achieve them. By addressing these quality attributes, the business goals mentioned in the previous section can be realised.

Functionality <high>

- The system should perform all its intended functions, such as ingesting data, performing analysis, generating reports and supporting collaboration.
- The functionality should conform both to the technical and business needs of its users.

  Implementation plan/tactics

  Clearly define and document system requirements and use cases. use a modular Architecture design and validate features with stakeholder feedback.

Usability <high>

- The system must be intuitive and user-friendly, such that the learning curve is small for all user groups.
- It should provide clear interfaces and workflows tailored to different user roles.

  Implementation plan/tactics

  User-friendly interfaces and role-based workflows will be designed through usability testing. Clear guides, tutorials, and an intuitive design can minimise the learning curve.

Scalability <high>

- The system must handle growing workloads efficiently. It should support an increase in the number of users as well as the use of big data sets without compromising performance.

  Implementation plan/tactics

  Distributed processing and load balancing will enable efficient handling of larger datasets and more users. Regular performance tests will identify scalability issues.

Performance <high>

- The system should have minimal latency in multiple cases: ingesting data, processing, visualisation and collaboration. This is important for user satisfaction, especially in cases where real-time analysis is required.

  Implementation plan/tactics

  Optimised data pipelines, caching, and high performance hardware will make sure there are minimal delays. Monitoring tools can be used to track and improve processing, visualisation, and collaboration speeds. RabbitMQ will make job queuing in microservices possible

Testability <high>

- The platform should be thoroughly testable, such that all features are tested both individually and as a whole. Additionally, a method of efficiently testing the system with big data should be established.

  Implementation plan/tactics

  Automated tests will cover all features, supported by continuous integration/continuous development (CI/CD) pipelines. Large data sets can be used to test performance for certain high demanding scenarios.

Security <high>

- Security is one of the most important requirements for this platform. Since companies might be storing sensitive information in there to analyse, we need to make sure that the system is safe and secure, so that no unauthorised users can access the data.
- Next to that, we have to adhere to GDPR and also privacy and data protection laws.

  Implementation plan/tactics

  Encryption, role-based access, and regular security audits will protect user data. Compliance with GDPR and other laws will be a priority, ensuring privacy and safety. Code quality will be analyzed with Sigrid to ensure proper and secure code is used.

Auditability <high>

- The system must maintain a detailed log of actions, including data access, modifications, and user activities. This ensures compliance with regulations and can give accountability within teams.

  Implementation plan/tactics

  A centralised, secure logging system will record all activities. Audit reports and alerts for suspicious behavior will improve accountability and compliance.

Resilience and recoverability <high>

- The platform must handle unexpected failures or bugs gracefully, ensuring data integrity and no loss of functionality.
- In case of failures, it should recover quickly to restore normal functionality.

  Implementation plan/tactics

  The system will include failover mechanisms, regular backups, and data replication for data such as passwords and auditing reports. Testing recovery plans will ensure quick restoration after unexpected failures.

Maintainability <high>

- The system should be easy to maintain, such that developers can easily troubleshoot, fix bugs and implement new features with minimal effort.

  Implementation plan/tactics

  We will use clean code, detailed documentation, and modern debugging tools to make the system easy to update. Regular refactoring will keep the codebase efficient and manageable. The agile and incremental development process makes it easier to implement new features. This together with development tools such as Git and Jira will improve maintainability.

Interoperability <medium>

- The platform must integrate seamlessly with existing tools, databases and APIs.
- It should be able to efficiently transfer data between various environments.

  Implementation plan/tactics

  The platform will use standardised APIs for seamless integration with third-party tools and databases. Compatibility will be ensured through regular testing and robust documentation.

Adaptability <medium>

- The system must adapt to every level of user expertise, accommodating both novice and advanced users.
- It must address a wide range of data challenges, from basic processing to complex analytics.

  Implementation plan/tactics
  Configurable settings will support both novice and advanced users. Modular design and user feedback will inform developers of diverse needs and data challenges.

Reliability and Availability <medium>

- It would be preferable if the platform has little downtime and if there is it should be scheduled. This is because downtime can result in a loss of data.
  Implementation plan/tactics
  Failover systems and rolling updates will reduce downtime. Maintenance will be planned during non-peak hours, and tools will monitor uptime to ensure availability.

Portability <medium>

- The program must be operable across multiple operating systems and data must be stored on multiple environments, such as the cloud or servers owned by the users. This makes sure that users can choose a deployment model that fits their needs.
  Implementation plan/tactics
  A web application will be available for all platforms, ensuring the application works for all users systems. API's will allow flexible data migration.

Extensibility <low>

- The system should provide a way to easily add custom features and possible extensions in the feature.
  Implementation plan/tactics
  A plugin-based architecture and detailed guidelines will make adding features easy.

7.1 - Trade-offs

In setting relative priorities amongst the many attributes, it needs to be remembered that very often improving one quality of a system has adverse effects on another one. Regarding DataForge, decisions about which attributes to consider as "high" priority and which should be lower come from a careful consideration of the platform's core purpose-that is, powerful, secure, and efficient data analysis-balanced by pragmatic considerations of time, resources, and long-term strategic importance.

As shown in the list above, the high-priority attributes are functionality, usability, scalability, performance, testability, security, auditability, maintainability, resilience & recoverability. It is clearly elaborated above why they are crucial, however, intense emphasis on these attributes can conflict with other ones. For example, strong focus on Security and Auditability is typically synonymous with additional complexity that can render a system more challenging to operate or extend. Ambitious Performance and Scalability goals drive the adoption of specialised architectures or technologies that can lower portability or raise development costs. While these kinds of trade-offs may be inevitable to meet key user needs and ensure regulatory compliance, they do require management with care. Medium priority attributes, such as Interoperability and Adaptability, reflect the aspiration of the project to easily integrate into various environments and support various kinds of data-related activities. These attributes are important but may be enacted secondarily, after stabilisation of the core product, which may correspond to later stages of development. For example, full optimisation of interoperability might be an obstacle to early-stage development if it would involve a lot of custom integrations or strict compliance to an interface standard. Adaptability requires modular designs and dynamic interfaces that increase the complexity upfront.

More specifically, some of the most notable trade-offs for DataForge are firstly that we had to sacrifice part of our security to ensure the performance of the application is high enough to satisfy the customers. For example, as will be explained in more detail later, DataForge does not encrypt all data, because that would be a disaster for the performance of the system. However, because of this there is a risk that when there is a data leak, all the data can be read by the person that managed to get into the system. So, to compensate for this, DataForge would have to increase their security in other parts, to make sure the risk of a data leak is minimised.

Next to that, there is a big tradeoff between modularity and maintenance. Modularity is an important part of dataforge, because it increases the resilience and reliability of the system, while it can also improve performance. However, the sacrifice that needs to be made for this is that the difficulty of maintaining these subsystems increases. This would mean that the DataForge maintenance team has to contain more people, which costs more money, but that can be justified by the gains of having a system that performs well.

Furthermore, the decision was made to process all data on the servers of dataforge. This choice was made to improve the usability, because this would mean that the user does not need a lot of storage space themselves. The trade-off that comes with this is that DataForge needs to make sure that the servers are working at all times, because otherwise the users would not be able to make use of the processing functionality of the application. This decreases the reliability and availability of the application. This trade-off was made, because we thought that the customers would find it significantly more important that the data processing could be done on our servers instead of their own. Especially for newcomers and individuals.

Finally, another trade-off that was made is that we chose not to save any data permanently on the server, only for a short period of time when the data is being processed. This was done to cut down on the storage space that was needed. The trade-off for this is that the system has less usability and functionality, because if the data was stored for a longer period of time, the user would not need to have their own database.

Those are some of the important trade-offs considered when devising the list of quality attributes and their prioritisation. High-priority attributes indicate urgent needs that impress user expectations, regulatory requirements, and competitive advantage. Medium and low priority attributes, though important, are deferred in maintaining a balance between near-term deliverables and long-term architectural strategies.

8 - Requirements

Below we list the requirements that we found to be the most important or relevant to our system, with a small explanation of what they are about. Each requirement also has a small list of more specific sub-requirements.

Functional Requirements

- The platform must be able to process structured, semi-structured and unstructured data. <high>
    - QA's: **Adaptability**
    - Stakeholders: **Data scientists, Newcomers**
    - Sub-requirements:
        - The platform must be able to process structured data in common formats such as JSON and XML.
        - The platform must be able to perform queries on unstructured data.
        - The platform must be able to combine queries with structured data.
- The platform must be able to provide a flexible visualisation interface for displaying data. <high>
    - QA's: **Usability, Functionality**
    - Stakeholders: **Data scientists, Newcomers, Business owners**
    - Sub-requirements:
        - The user must be able to change how data is visualised.
        - The platform must be able to forward data to visualisations from all sources.
- The user should be able to define their own processing standards to ensure consistency, compliance and adherence to best practices across teams. <medium>
    - QA's: **Functionality, Adaptability**
    - Stakeholders: **Data scientists**
- The user should be able to identify themselves on the platform. <medium>
    - QA's: **Functionality, Security, Auditability**
    - Stakeholders: **Data Scientists, Compliance teams, Business owners**
    - Sub-requirements:
        - The user must be able to log in.
        - The platform must be able to integrate into existing SSO.
        - The support and maintenance teams must be able to create new users and manage existing users.

- The platform should be able to automatically adapt to the user's needs and expertise level. <medium>
    - QA's: **Usability**
    - Stakeholders: **Data scientists, Newcomers, Compliance teams**
    - Sub-requirements:
        - The platform should be able to automatically change the visualisation based on the data type.
        - The platform should be able to provide different views for experienced data scientists and newcomers.
- The user should be able to customise the platform to meet their needs. <medium>
    - QA's: **Usability**
    - Stakeholders: **Data scientists, Newcomers, Compliance teams**
    - Sub-requirements:
        - The user should be able to override the settings that are automatically inferred.
- The platform must be able to generate compliance and performance reports. <medium>
    - QA's: **Maintainability, Auditability, Security**
    - Stakeholders: **Support and maintenance teams, Business owners, Compliance teams**
    - Sub-requirements:
        - Business owners should be able to see which user made what changes.
        - Compliance teams should be able to access a full compliance report.
        - Support and maintenance teams should be able to access internal data from the platform in order to evaluate the status of the system.
- The user must be able to chat with other users on the platform within the same team. <medium>
    - QA's: **Functionality**
    - Stakeholders: **Data scientists, Business owners**
    - Sub-requirements:
        - The user should be able to create public chat rooms.
        - The user should be able to send private messages to other users.
- The user should be able to create tickets and assign tasks within teams. <medium>
    - QA's: **Functionality**
    - Stakeholders: **Data scientists, Business owners**
    - Sub-requirements:
        - The user should be able to create tickets.
        - The user should be able to assign tickets to team members.
        - The user should be able to close tickets that are assigned to them or that they created.

Non-functional requirements

- The platform must be able to process data from any data source. <high>
    - QA's: **Interoperability, Extensibility**
    - Stakeholders: **Data scientists, Developers**
    - Sub-requirements:
        - The platform should provide an interface to integrate into existing tools and API's.
- The platform should be able to efficiently process large quantities of data in batches. <medium>
    - QA's: **Performance, Scalability**
    - Stakeholders: **Data scientists**
    - Sub-requirements:
        - The platform should be able to concurrently process multiple data streams in parallel.
        - The platform should be able to batch large quantities of data together in order to process them all at once.
- The user should be able to access the data quickly and efficiently. <medium>
    - QA's: **Performance**
    - Stakeholders: **Data scientists, Newcomers, Business owners**
- The platform should be able to integrate with existing tools, such as data processing libraries, analytics software, and machine learning models. <medium>
    - QA's: **Extensibility, Interoperability**
    - Stakeholders: **Data scientists, Newcomers, Developers**
- The platform should be able to handle errors gracefully without crashing. <medium>
    - QA's: **Resilience, Recoverability, Reliability**
    - Stakeholders: **Support and maintenance teams, Data scientists**
- The platform should be able to provide full transparency by tracing every action taken. <medium>
    - QA's: **Security, Auditability**
    - Stakeholders: **Business owners, Compliance teams, Government**
    - Sub-requirements:
        - The platform should keep track of every action taken.
        - The compliance teams should be able to get a report of every action taken.
        - The government should be able to verify whether the actions taken conform to the relevant regulations.
- The platform should be able to be updated without downtime. <low>
    - QA's: **Maintainability**, **Reliability**
    - Stakeholders: **Support and maintenance teams, Developers**

9 - Scenarios and Tactics/Patterns

Maintainability:

Scenario:

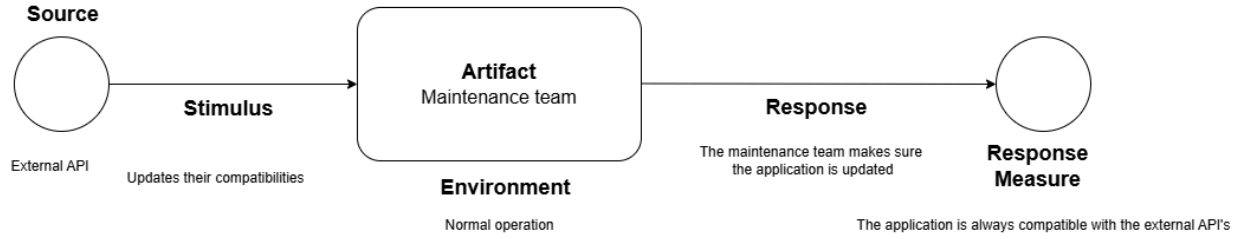

Figure 9.1

Patterns	Tactics
Modular Design	Separation of concerns Documentation standards

The best way to keep a system maintainable is to make sure it has a modular design. When making a separation of concerns, each part of the system can be updated separately from the others, which makes it much easier to maintain the system. Next to that, our developers need to adhere to documentation standards, so that future developers have no trouble understanding and updating the code.

Resilience and Recoverability:

Scenario:

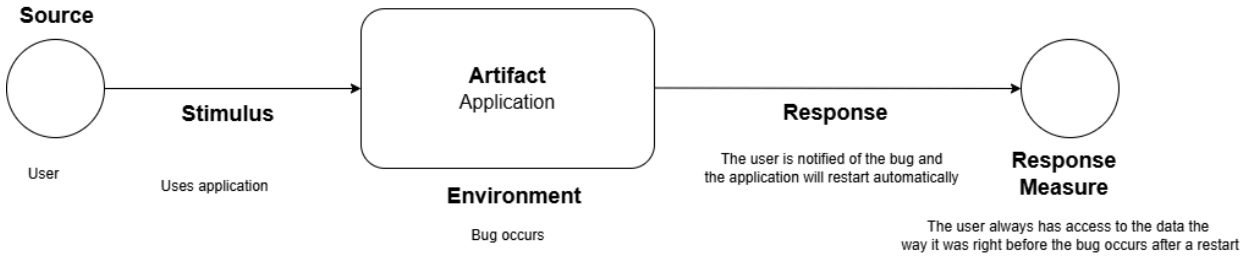

Figure 9.2

Patterns	Tactics
Backup and restore Automatically retry failed operations Modularity	Redundancy Failover mechanisms

To ensure resilience and recoverability for DataForge, since DataForge works with servers, it is important that there are backup servers in case the regular servers fail in some way. This way the backup servers can temporarily take over the workload, while the regular servers are restarting and the users are not inconvenienced. Furthermore, critical data such as passwords can be stored in multiple locations, such that when a disk is corrupted, the data is not lost.

Next to that, the use of modularity in our system adds to the resilience and recoverability of the system, because the system is divided in different subsections. As a result of this, when a subsystem fails, the other systems are able to operate on their own, which means that not everything will shut down when there is a tiny error somewhere in the system.

Auditability:
Scenario:

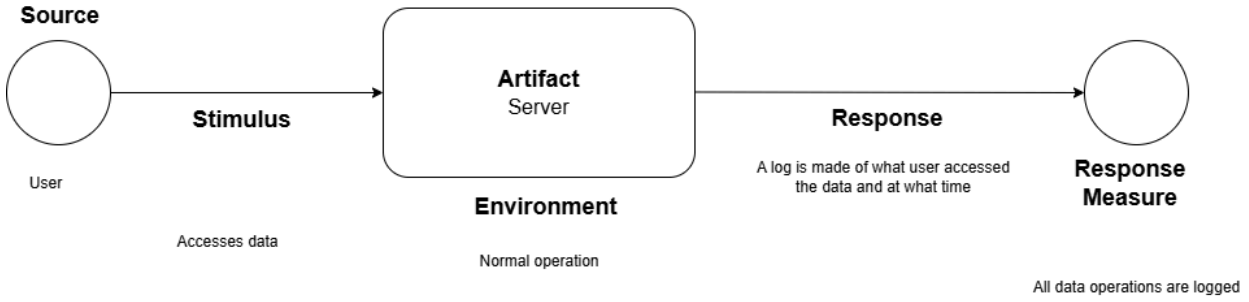

Figure 9.3

Patterns	Tactics
Event logging	Access Logging

Auditability can be ensured by event logging, where all important interactions of users with the system and the database are recorded. This means that when the user accesses or modifies data, a log is made of it.

Scalability:
Scenario:

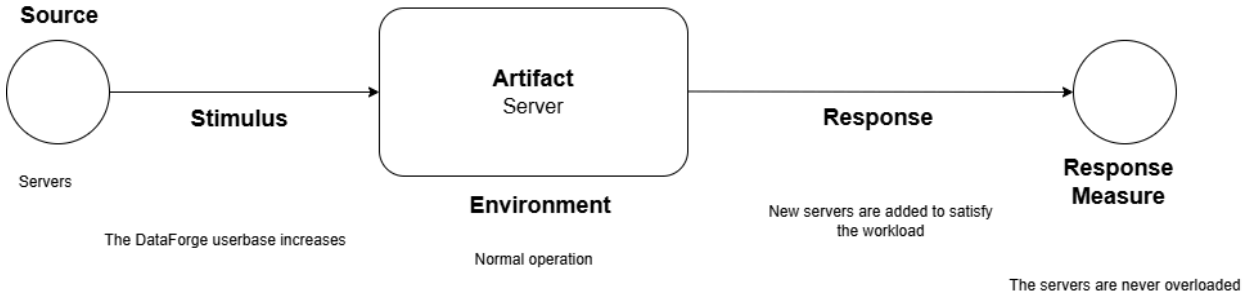

Figure 9.4

Patterns	Tactics
Load balancing Microservice Architecture Caching Pattern	Distributed processing Cloud services

The load balancing and microservice architecture which realises distributed processing seems like the best option for DataForge. This is because the microservice also separates concerts which is beneficial for security too.

Security:
Scenario:

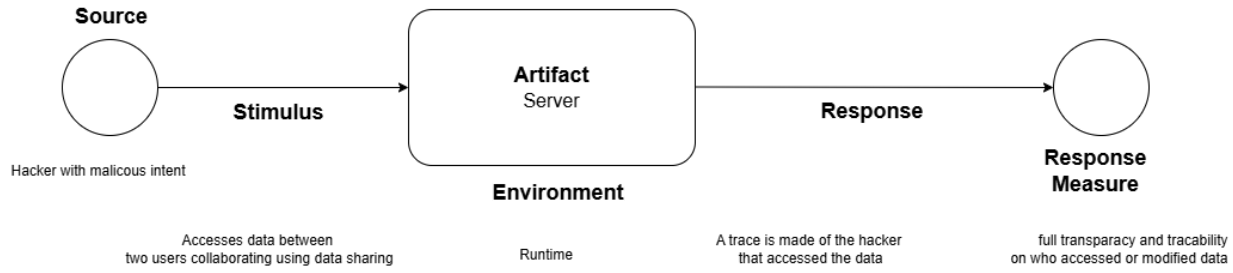

Figure 9.5

Patterns	Tactics
Security by design 2FA Role Based Access Control (RBAC)	Authentication and authorisation Encryption Auditing Input validation Firewalls

As for security, the most simple but also important tactic that needs to be implemented is authentication and authorisation. This way we can see who accesses the data and check if this is allowed. Also, this can help with the collaboration functionality. This is all part of security by design.

Performance:
Scenario:

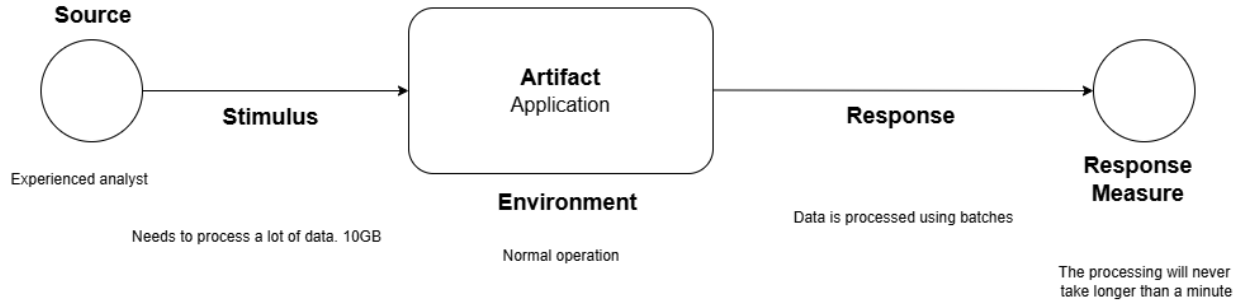

Figure 9.6

Patterns	Tactics
Microservices architecture Event-Driven Architecture Divide and conquer	Batch processing Load Balancing Caching Data compression Profiling and optimisation

What is most important for the performance of the system is that we make use of batch processing so that the operations on the data will be done as smoothly and fast as possible. This can be done with divide and conquer as a pattern, where we make sure we divide the data or assignments into smaller partitionings that can then be performed.

Usability:
Scenario:

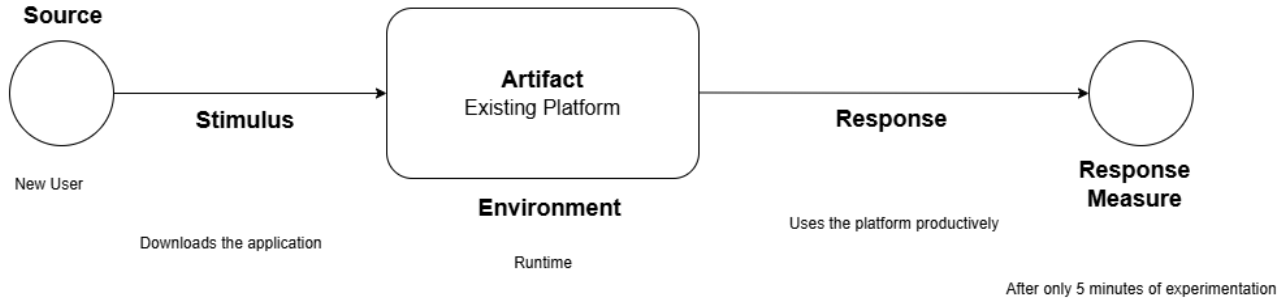

Figure 9.7

Patterns	Tactics
User testing Model view controller Intuitive UI design Feedback to user Personalisation	Beginner tutorial Help function option Allow user customisation Usability testing

We decided to go with User testing paired with intuitive UI design as our design patterns. What this means is that the system needs to be designed in an intuitive way and afterwards it should be checked that the users agree by doing some Usability testing. Next to that, some personalisation paired with feedback to the user would be nice, because we want each type of data to have its own visual representation.

Adaptability
Scenario:

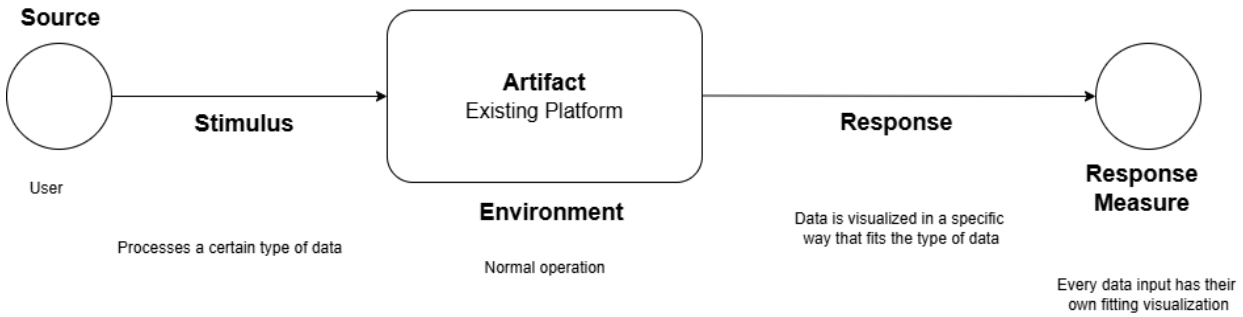

Figure 9.8

Patterns	Tactics
Plugin architecture Service-oriented architecture	Modular design Configuration over code Plugin based system Extensibility points

To achieve sufficient adaptability of the system, we decided to apply a modular design and add extension points such as plugins, APIs, etc. Together with providing configuration settings for the system, this will reduce the amount of code needed significantly. Patterns used to implement these tactics are a plugin architecture and a service-oriented architecture. By creating services the design becomes more modular and adding plugins becomes more feasible this way.

Testability
Scenario:

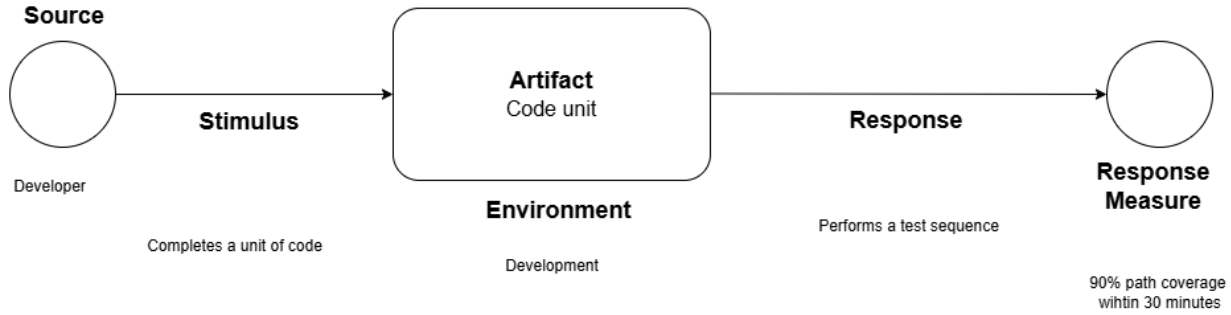

Figure 9.9

Patterns	Tactics
Continuous Integration/Continuous Deployment	Automated testing Dependency injection

We want the system to be testable as thoroughly as possible. To achieve this, we apply automated testing and dependency injection. By making software components dependent on each other, bad code will become more apparent and testing components together (automatically) will be more doable. Together this all comes down to the Continuous Integration/Continuous Deployment pattern, where code changes are automatically built, tested and prepared for deployment.

Interoperability
Scenario:

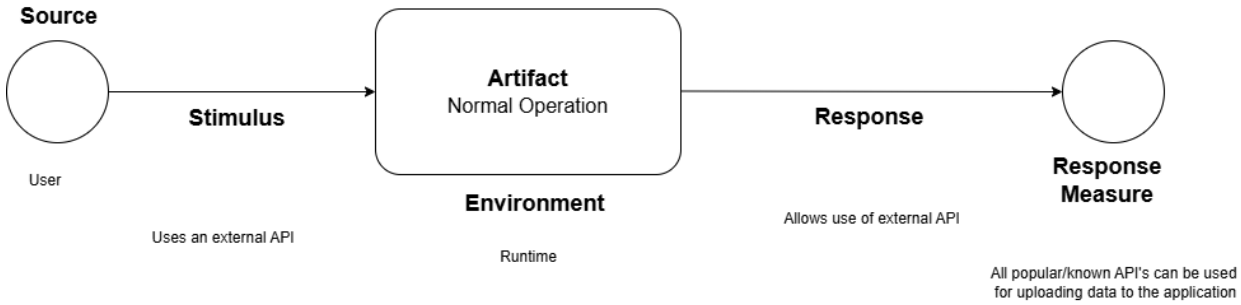

Figure 9.10

Patterns	Tactics
Use Open Standards Use of API's	Standardised protocols Data standards

Making use of open standards will improve interoperability. We will make use of existing open standards, such as HTML, TCP/ICP, Unicode, etc. This way, the software will be available across multiple platforms and will work on different operating systems. Furthermore, we will make sure that different systems and devices will work together in our product by using APIs. These APIs will act as a common interface that works on all systems.

10 - Views

To address the concerns discussed earlier we identified four different views of the system. Since data plays quite a central role in the purpose of Dataforge, this is the first view we will address. This view serves as the most general flow within our system, whilst other views will look at certain aspects of the system on a more detailed lower level. The component view is one of these views. Furthermore, the functional view should address most of both the experienced data scientists' and newcomers' concerns. Additionally, the deployment view will shed light on how different parts of the system are deployed in microservices and the separation of various components in the system. Finally, the development view should address the concerns of the developers and should address how the modularity and incremental design are addressed by the systems architecture.

10.1 - Data View

Figure 10.1 shows the flow of data within DataForge. The data in this diagram is separated into two types, analysis data and app data, as shown in the legend of the diagram. The app data flows contain any other data excluding the data on which the analysis is performed. The analysis data contains all types of data including the input data. The purpose of this separation is to show how the imported data is handled within the DataForge system.

The flow of the data starts with the input data selected through the web interface (Frontend Server). What input data source to use can be selected by the user. This request to ingest the data is then sent to the data Ingestion system. The Data Ingestion System is responsible for handling input in our system. As can be seen in data flow diagram 1, the source of the input data is categorised into three different categories:

- **Databases:** data from external sources.
- **Data files:** data in any file format such as: CSV, .xlsx, xml, etc.
- **Live Data:** live data flow from for example obtained from a sensor.

This distinction is made since the system will act differently depending on the type of imported data. The most straightforward data type being the data files that can be uploaded from the users system to the analysis system.

Next is the data from databases, here the data should be requested from an external database and permission should be granted from that database for the data to be imported. Dataforge will only store the addresses of the external data sources to maintain the security provided by the external databases.

The live data behaves the most unique out of three, since its use can differ from the intention of the user. The user can either choose to actively update the data for continuous analysis, or to save or record the data over a period in time.
The Data Ingestion system then converts the data to a unified data format handled within DataForge. These three different types of data provide the framework to be used for the modularisation of the data ingestion. The data conversion for these different input types is modularised such that support for more files can be added independently and allows for adding support of new data formats in the future.

After conversion, the data is stored in the analysis backend on the intermediary database. For large datasets, the system also supports batch processing. This is handled in the Analysis system by its microservices architecture. More detail on the microservice architecture is given in the deployment view.

After importing the data, the user can perform analysis on the data. The analyzed data is then stored on the output/view database. Imported analysis data can only be requested by the frontend from this database. The main advantage of using a web interface for the dataforge system is that data analysis can be synchronised by using the same database in the collaboration system.

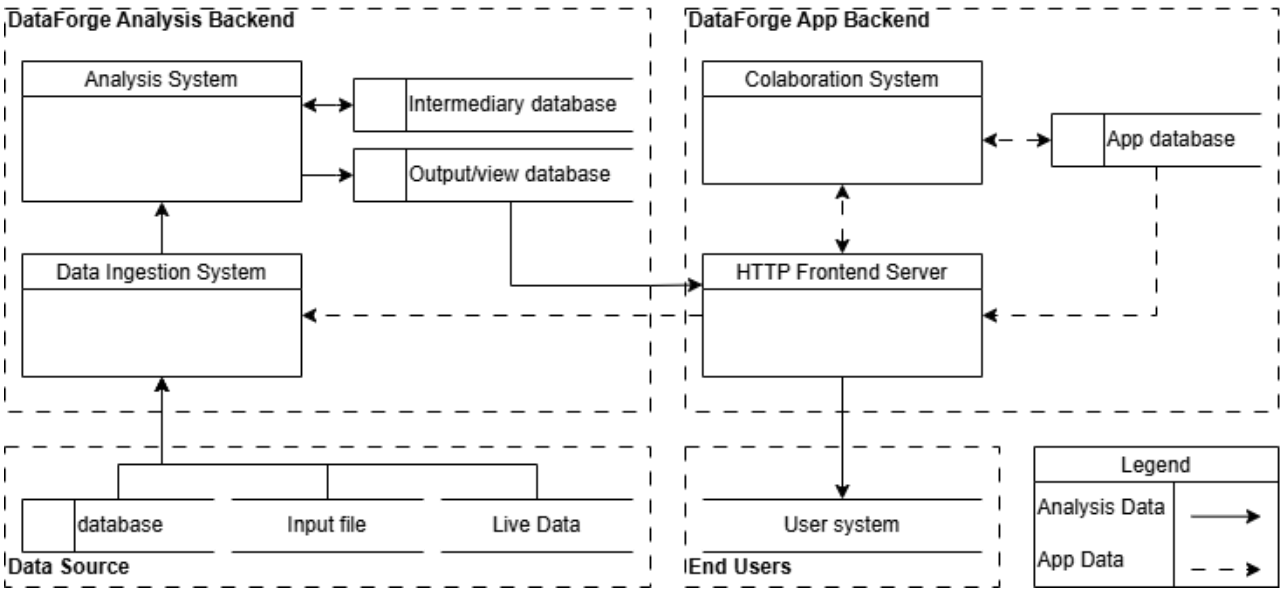

Figure 10.1: Dataflow Diagram DataForge

10.2 - Component view

This view regards the function and interaction of the DataForge system's components (at the highest level and their subsystems). Its main concern is the system's modularity. This decomposition is highly relevant since DataForge supports many data formats and many data analysis methods. This viewpoint expresses that these are two different components that should be independent of each other as much as possible. We summarise all of DataForge's components in a UML component diagram.

At the core of the dataforge system is the *App* component, this component controls all major functionality of the application, and communicates to the client application of the user. Since all other components are controlled by the app, the logging system does not need to interface with any other component than this and the *App storage* to store results. The actual data analysis happens between the *Data conversion*, *Analysis*, and *View* systems. Data to be processed always goes through the conversion system which transforms all inputted data to a single format expected by the *Analysis System*. There is a different port for each type of data that is supported (see: data view section). As well as an output for data files to support analysis activities such as data cleaning. Once data is converted to the expected format, the analysis component takes over, which and how the analysis is performed is informed by the *App* and optionally the processing standards stored in *App Storage*. The results of the Analysis are then given to the *View System* which can be used to create/generate reports and make dashboards, which can be stored in *App Storage*. Thus the three largest elements of DataForge - supporting many different data types, many forms of data analysis, and viewing the results of the analysis - together with all other elements of the system have been modularised to support many of our most important quality attributes and concerns.

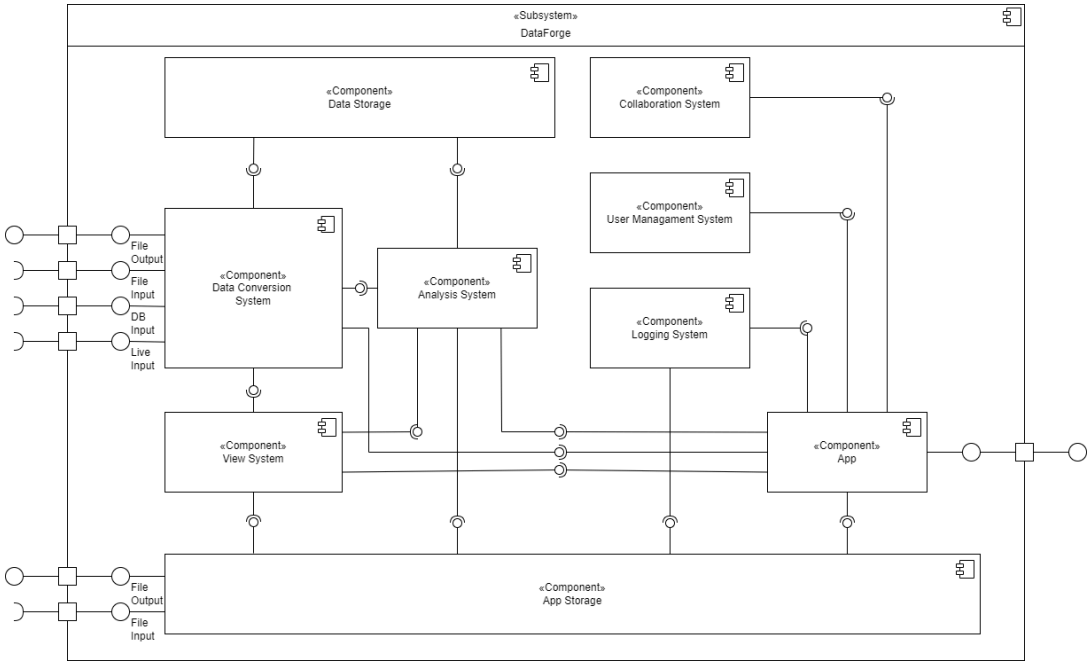

Figure 10.2: Component Diagram DataForge

10.3 - Functional View

The functional view describes how DataForge may be used. It addresses therefore the concerns held by the end-users. The core functional elements of DataForge have been divided into four categories as can be seen in the functional decomposition diagram (Figure 10.3). Recalling the components View it is clear these functional elements are stand-alone with clearly bounded Interaction. This view addresses how combinations of these components interact to provide all intended functionality.

Converting data is a prerequisite for using any other functionality of DataForge, since DataForge has to handle so many different types of data it would be untenable to handle those differently. Thus this functionality is required to have the user access the rest of the system with any data they might want. However, since the conversion system has an option for a file output, it provides functionality all on its own. Namely as a powerful all-in-one data conversion tool. Many tools for converting one file type to the other already exist DataForge supports any combination of file types and Database/live data to any of these file types all in one place. While this system is very complicated in the background to the user it must seem simple. They will have the option to (1) choose which type of data they wish to import/connect (file, DB, live), but not the file type, this should happen automatically. And (2) which type they wish to convert it to, either a new file type which should be done manually in this case of course or to continue with the data to either perform Analysis or immediately visualise, one of these should be set as default optionally to be changed by the user.

Normally the practice of data analysis would refer to the combination of what we have called *perform analysis* and *view results*, but in data forge these have been split, from the components view to aid in modularity, but from the functional view to allow the user more freedom in how to use the platform. For example, a newcomer might wish to neatly present data collected by their smartwatch such as their step count, this is a very simple use case wherein the user would not want to do any activities listed under perform analysis. Thus they should not have to click through many pages to get where they want to be and instead go straight to the choice of creating a single diagram or dashboard in either case they should be suggested which type of diagram would work best for their data (in case of step counts a bar chart with a single box per day is the obvious choice). Even when doing more data analysis activities than just visualising not all might be required all of the time. A data scientist working with a structured relational database might not need to do any data cleaning for example DataForge should recognise this and hide (but not remove!) this option to provide a seamless experience.

Finally, there is the collaboration functionality, while this is very important, it stands in support of the other three areas of functionality. While a member of a compliance team might only ever interact with this pact of the functionality such as: reviewing actions taken by data scientists, interacting with the ticketing system, or updating the processing standards.

For data scientists it is important that collaboration is as unobtrusive as possible. In general, a data scientist interaction will be short and sporadic, for example, quickly sending a chat message or accepting/completing a ticket. the parts of the collaboration system to be used by data scientists should be available from all the other parts for example from a side panel or a widget for the chat, this keeps the data scientist focussed on what is important, but more importantly, avoids the risk of opening them up to long wait times when going back and forth between parts of the system.

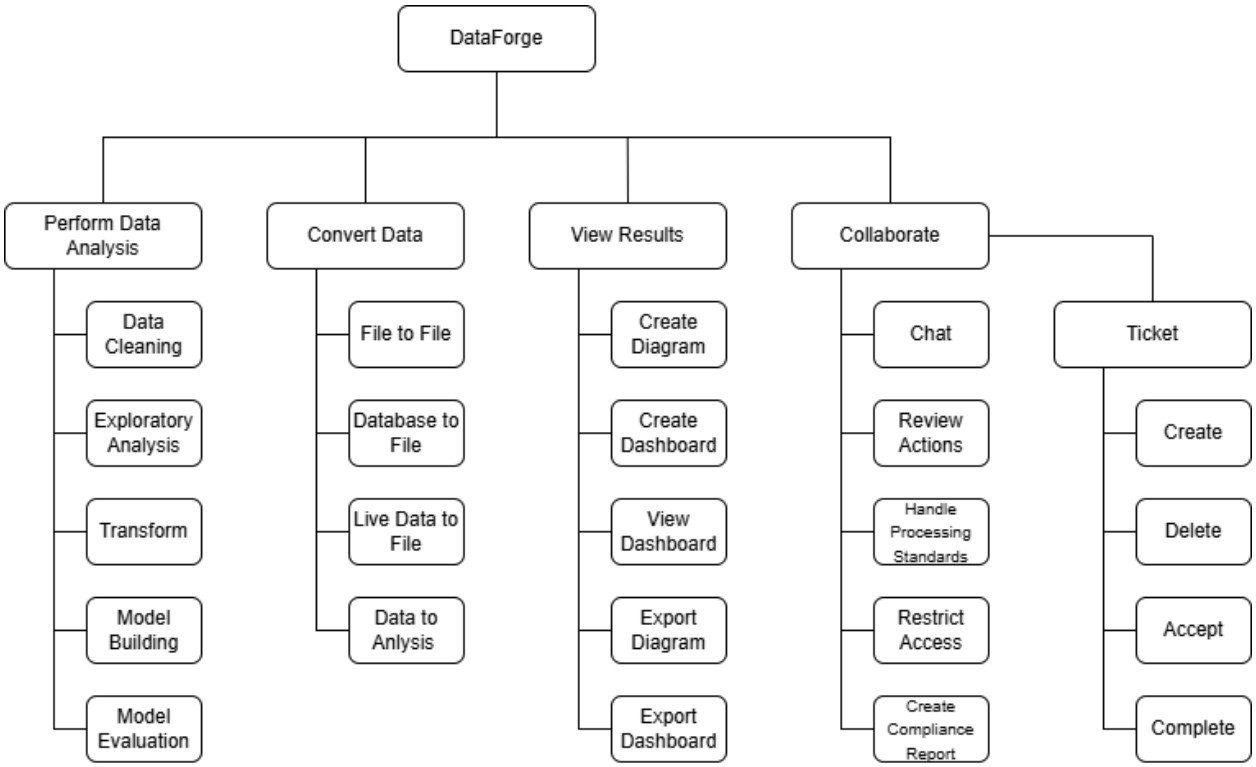

Figure 10.3: Functional Decomposition DataForge

10.4 - Deployment View

As scalability, resilience and performance are important attributes for DataForge, the deployment architecture that best fits these requirements is the use of microservices, as can be seen in Figure 10.4. While using microservices may decrease the maintainability of the system, as developers and maintenance teams have to keep dozens of interconnected systems in mind, it will allow for high scalability, modularity and performance. As described in the trade-offs section, high priority is put on the latter quality attributes. The full set of microservices used for the backend can be split into two distinct clusters: the app backend and the analysis backend.

The analysis backend is responsible for ingesting and converting the data, and processing it to the views the user requests. All kinds of data sources are connected to the data ingestion server, which processes, converts and distributes the data to the other systems. Processing happens on the various analysis nodes. These nodes all have a connection to an intermediary database and an output database, which will contain our final data views. The nodes receive data from the ingestion server and process it according to jobs received from the analysis coordination server. This server manages splitting up larger jobs into smaller atomic parts, and maintains a queue of small jobs to instruct the analysis nodes. Architecting the backend like this allows for large data throughput and easy scalability. The number of nodes can be increased for more processing power, and ingestion and coordination servers can be doubled up if capacity is exceeded. All of the servers of the analysis backend should be deployed in close proximity to each other in order to reduce latency and increase data throughput.

The app backend facilitates communication between the end user and the data analysis system. As with the other systems, the app backend is split into multiple services allowing for high scalability and modularity. The main point of contact is the HTTP frontend server, which serves the app to the internet and allows end users to use the platform. It is responsible for instructing the analysis systems, setting appropriate inputs on the ingestion server, forwarding user jobs to the coordination server and serving the resulting views from the database. Access control is implemented using the account server, which manages user accounts, authentication and authorisation. Collaboration is facilitated with help of the collaboration server, which is responsible for synchronizing collaboration sessions, working closely with the account server to manage teams and companies, and sharing data. Finally, all the actions taken on the app are sent to the logging server, which is responsible for keeping track of everything that happens, in order to generate compliance reports. All of the systems in the app backend have access to the app database, which is a general-purpose database to store user preferences, accounts, views and logs. It is deliberately kept separate from the other databases to keep a separation between app data and processed data. This database is encrypted in order to keep user data secure in the event of a potential attack, as this is the first line of defense. The app backend should be deployed in the cloud and close to the user, possibly using modern technologies such as edge computing in order to improve the user experience by increasing responsiveness and reliability.

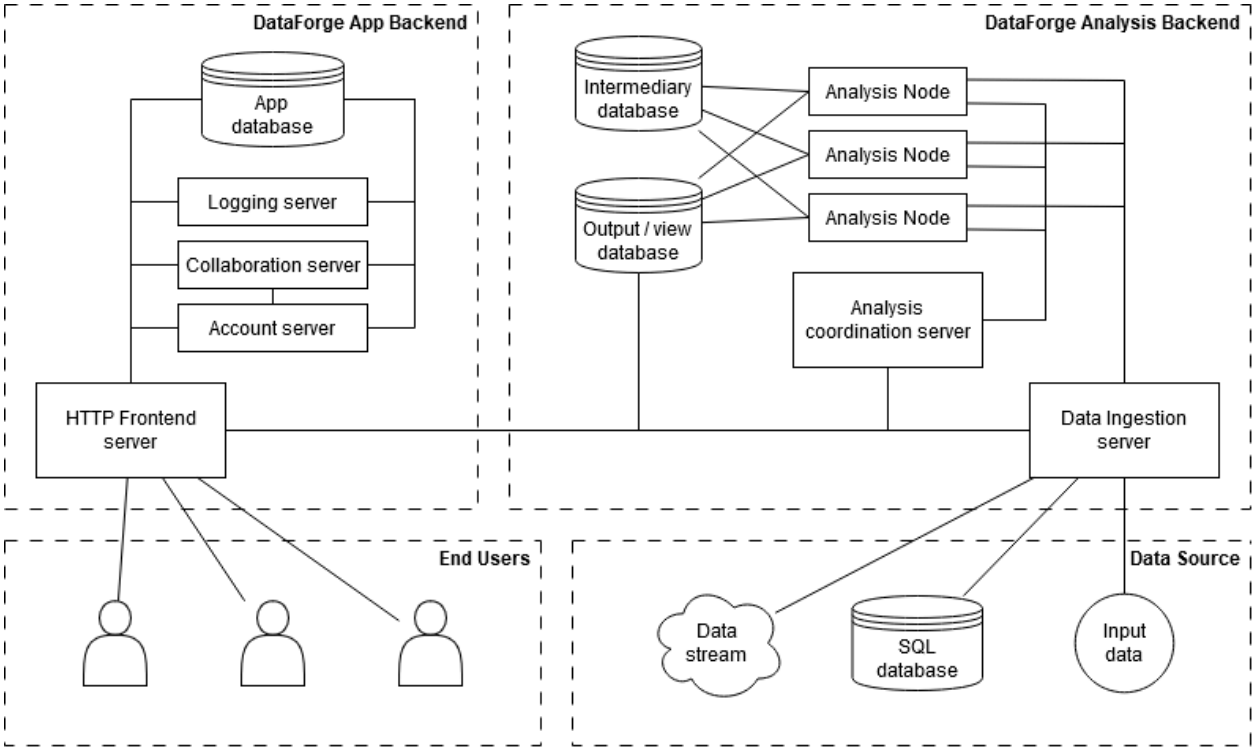

Figure 10.4: Deployment diagram for DataForge

10.5 - Development View

As per the work of Kruchten (1995), "*the development view serves as the basis for requirement allocation, for allocation of work to teams (or even for team organisation), for cost evaluation and planning, for monitoring the progress of the project, for reasoning about software reuse, portability and security*". To put simply what the extensive description refers to, it describes how the software is structured in the development environment, showing its various modules and subsystems and the interrelation among them. The purpose of this view is to provide a maintainable, scalable, and well-organised codebase, as according to the Agile development principles.

DataForge has such a structure that further promotes modularity by the separation of different functional issues. The system is composed of a number of components, each of which focuses on different aspects of its overall functionality. The reason for separating these concerns is that we can have separate teams working on each of those components. If there is a problem with one of these components, the corresponding team will work to fix it, without having to disturb the work of other teams, since those components are still intact.

In terms of the development process and collaboration, DataForge follows the Agile development methodology with CI (Continuous Integration) and automatic deployment, ensuring quality code and simplicity in development. There are a lot of use cases and data integration options to be implemented in the system. These functionalities can be broken down into multiple development stages. Each stage delivers a partial implementation of the final system, adding features to the result of the previous stage. The main benefit of this process is that this allows for early user involvement. As DataForge aims to be a true competitor in the market, it is important to involve users as early as possible during the development process. Customer satisfaction is crucial for the success of the development of the system. Especially, because DataForge tries to enter a market that is already highly competitive. Another reason these processes have been chosen, because they have been proven to work effectively in the past and are a cost efficient way of working. Next to that, most developers have worked with Agile before and thus will be comfortable to keep working in this manner.

Development is done within 2-week sprints, which allows incremental releases and iterative improvement. Doing this, the customers continuously get new updates which will improve customer satisfaction. Also, when there are bugs in the system, these can be fixed relatively quickly. Regarding tests, these are automated with GitHub Actions or Jenkins in order to stabilise the code before production. Deployment will be via Docker and Kubernetes for cloud scalable infrastructure.

The development teams use GitHub or GitLab with feature-based branching implemented in an orderly fashion. Code review is obligatory for all pull requests with the aim of sustaining high-quality standards and preventing regressions. Swagger is used to generate API documentation automatically. Issue tracking and project planning have been done in JIRA to enable structured backlog management. Again, all these systems are well-known in the space of computer science, which means most developers will have worked with it before, which means they will not have to spend extra time on getting familiar with the systems.

The Development View presents a structured approach to the software architecture life cycle by guaranteeing system scalability, security, and maintainability using a multi-layered architecture in tandem with Agile processes and CI/CD automation. This ensures continuous improvement while preserving the integrity and high performance of the system. By doing so, this will allow the development team to successfully construct, test, and deploy new features in order to keep the system robust and responsive to dynamic user needs.

11 - Specifications

Below, a table of all specifications is displayed. These include formal, requirement, interface, functional, technology and data specifications.

Type	Description	Relation to Architecture	Dependencies
Formal	ISO 42010:2011 for software architecture documentation. This governs how components, concerns and viewpoints are documented [3]	Used in all sections for overall documentation structure	None
Protocol	HTTP 2.0 for communication between frontend and backend [4]	Using this protocol defines a single method of communication, improving the maintainability and interoperability.	None
Protocol	REST API for standard interaction. Allows for implementation of external software.	Used for realizing quality attributes: maintainability, interoperability, extensibility. Relates to requirements: Integrating existing tools	OAuth 2.0 JSON HTTP 2.0
Protocol	OAuth2 for authenticating using third-party services. [5]	Used for realizing quality attributes: security and interoperability.	JSON
Protocol	RabbitMQ for job queuing in microservices architecture. [6]	Used for the performance part of the AD: quality attributes and requirements.	JSON
Protocol	JSON for messaging and data format. Defines the way data is sent between software components. [7]	Quality attributes: Adaptability, interoperability, maintainability. (sub) Requirements: process multiple data types	None
Interface	User interface designed with HTML5, CSS3, and JavaScript. [8, 9, 10] with help from pre-existing frameworks like Django/React.	Impacts the user experience by usability of the UI and maintainability of the system by choice of languages	Visualisation
Interface	Dashboards for data monitoring ,visualisations and collaborative work	Impacts the choice for visualisation methods and enables auditing capabilities,	HTML/CSS/JS Generate reports Chat system

		aiding in usability and conforming to requirements.	
Functional	Generate and export reports in textual formats by integrating data on storage, processing and user actions.	By realising this, the auditability requirement is satisfied.	SQL Dashboard
Functional	A platform for real time chat capabilities and management system that for example is used for creating tickets	Satisfies chat, and ticketing/task assigning requirements	AES-256 SQL Communication tools
Technology	Programming languages: C++ for performance [12]; HTML5/CSS3/JS for UI.	Impacts the way certain components are coded, for example functional vs object oriented. Therefore this relates to maintainability aspects.	REST API
Technology	AES-256 encryption for stored account related data such as passwords. [11]	Relates to security requirements and aspects	None
Technology	Communication tools like Jira, Slack, git.	Influences the development process. Can have an impact on maintainability.	None
Technology	Code quality analysis tools such as Sigrid	Influences the development process, can have an impact on security and maintainability	C++ HTML5/CSS3/JS
Data spec.	Input stream (e.g., TCP) for continuous real-time data handling.	Used for realizing the quality attribute interoperability and extensibility.	JSON
Data spec.	SQL for interacting with databases as a data source.	Used for realizing the quality attribute interoperability.	None

In figure 11.1 we can see the dependency graph that has been derived from the table. It shows the interconnected nature of all of our dependencies, as most of the specifications used can be found in a single sub-tree. This further justifies our choice of technologies, as we use those best combined with other technologies, and we don't unnecessarily introduce new technologies where we already have dependencies on existing ones in order to keep maintainability high.

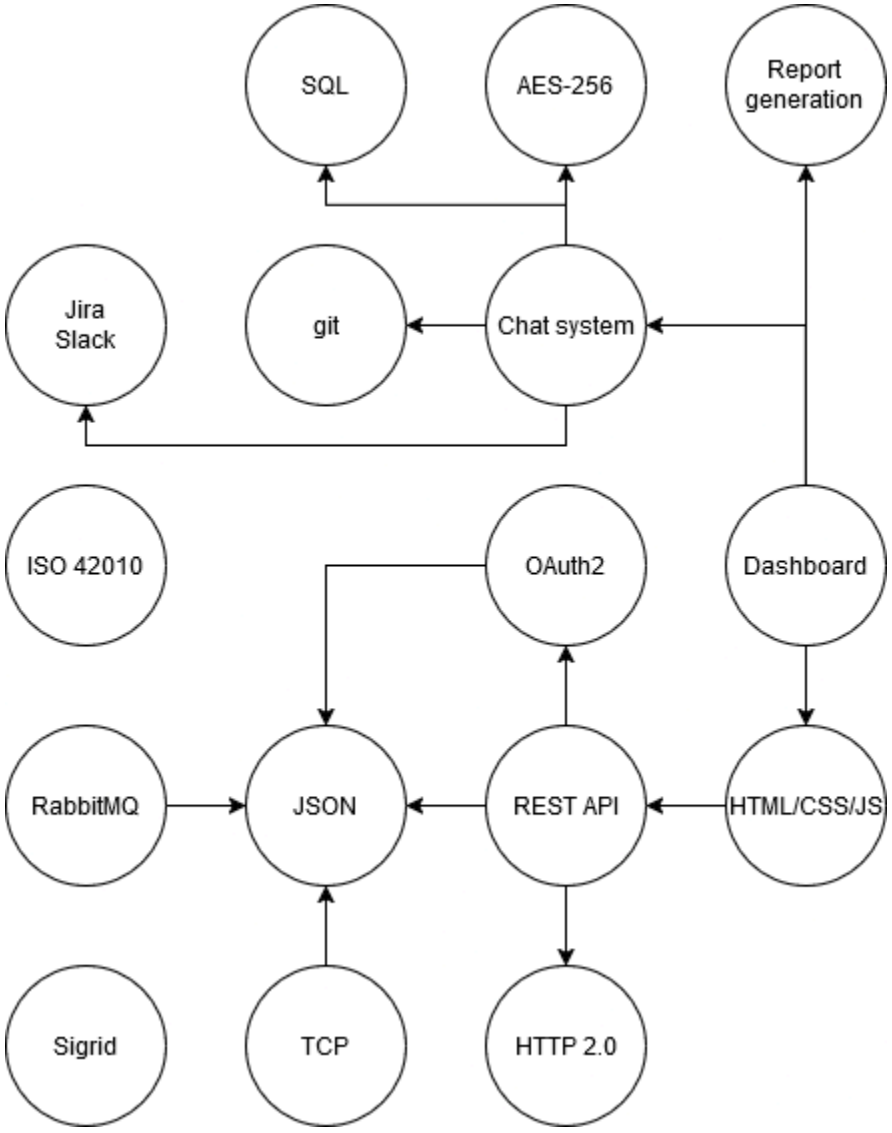

Figure 11.1: Dependency graph of specifications

References

1. Dictionary.com. (n.d.). Data scientist. In Dictionary.com. Retrieved November 19, 2024, from https://www.dictionary.com/browse/data-scientist
2. https://barc.com/data-management-trends-developments-2024/
3. ISO/IEC/IEEE Systems and software engineering -- Architecture description. (2011). *ISO/IEC/IEEE 42010:2011(E) (Revision of ISO/IEC 42010:2007 and IEEE Std 1471-2000)*, 1–46. doi:10.1109/IEEESTD.2011.6129467
4. Belshe, M., & Peon, R. (2015). *Hypertext Transfer Protocol Version 2 (HTTP/2)*. https://doi.org/10.17487/rfc7540
5. Hardt, D. (Ed.). (2012). *The OAuth 2.0 Authorization Framework*. https://doi.org/10.17487/rfc6749
6. RabbitMQ. (2019). *RabbitMQ*. Rabbitmq.com. https://www.rabbitmq.com/
7. Bray, T. (2017). *The JavaScript Object Notation (JSON) Data Interchange Format.* https://doi.org/10.17487/RFC8259
8. *HTML Standard*. (2019). Whatwg.org. https://html.spec.whatwg.org/multipage/
9. *All CSS specifications*. (n.d.). Www.w3.org. https://www.w3.org/Style/CSS/specs.en.html
10. *ECMAScript® 2022 Language Specification*. (2022). Ecma-International.org. https://262.ecma-international.org/
11. Evans, D., & Brown, K. (2001). FIPS 197 Federal Information Processing Standards Publication Advanced Encryption Standard (AES). *Advanced Encryption Standard (AES)*. https://doi.org/10.6028/NIST.FIPS.197-upd1
12. International Organization for Standardization. (2024). *Programming languages - C++* (ISO Standard No. 14882:2024). https://www.iso.org/standard/14882.html
13. The Science of Machine Learning & AI. (January 23, 2025). Exponential Growth. https://www.ml-science.com/exponential-growth#:~:text=The%20exponential%20growth%20of%20AI,doubles%20approximately%20every%20two%20years
14. PostgreSQL. (January 21, 2025). PostgreSQL. https://www.postgresql.org/
15. Kafka | Apache Flink. https://nightlies.apache.org/flink/flink-docs-master/docs/connectors/datastream/kafka/
16. Jupyter Notebook. https://jupyter.org/
17. Arduino. https://www.arduino.cc/
18. Tableau. https://www.tableau.com/why-tableau/what-is-tableau
19. SAP analytics. https://www.sap.com/products/technology-platform/cloud-analytics.html
20. Mario Piattini, Guido Peterssen, and Ricardo Pérez-Castillo. 2020. Quantum Computing: A New Software Engineering Golden Age. SIGSOFT Softw. Eng. Notes 45, 3 (July 2020), 12–14. https://doi.org/10.1145/3402127.3402131
21. Kruchten, P. (1995). Architectural blueprints—The “4+1” view model of software architecture. IEEE Software, 12(6), 42-50. https://doi.org/10.1109/52.469759

Appendix A - Contributions

Overall we experienced a fair contribution from all members in this project. All design decisions were discussed within the group. This is the list of chapters and its main contributors for working out these decisions in the final chapters.

Section	**Team members**
Introduction	Guido Baels
Scope	Bas van de Weerd, Joris Vlaar
Related System	Guido Baels
Trends and Developments	Guido Baels
Stakeholders	Asen Pantov
Concerns	All
Stakeholder Communication	Asen Pantov
Business Goals	Asen Pantov
Quality Attributes	Pascal Vriend, Guido Baels
Trade-offs	Guido Baels
Requirements	Chris Bleeker
Scenarios and Tactics/Patterns	Pascal Vriend, Guido Baels
Views	Bas van de Weerd, Joris Vlaar
Data View	Bas van de Weerd
Component View	Joris Vlaar
Functional View	Joris Vlaar
Deployment View	Chris Bleeker
Development View	Asen Pantov, Guido Baels
Specifications	Chris Bleeker, Pascal Vriend
Presentations	All