Department of Computer Science
University of Pretoria

Architectural Specification for GND

April Four

August 2024

Version	Version History	Author
1.0	Initial release	AprilFour
1.1	Demo 3 Revision	AprilFour

Table 1: Version History

GND

Table of Contents

1 Introduction

This document serves to outline the Architectural Specifications of the GDPR Non-Compliance Detector (GND). In this document you will find the design strategy that the team, AprilFour, have decided upon to implement the project, the quality requirements identified, architectural and design patterns used and the technologies that are used to build the system.

2 Design Strategy

The team, AprilFour, has decided to use the Decomposition design strategy to implement the GND system. The decomposition design strategy is based upon breaking down a system into smaller subsystems that can each run independently and then together as a fully fledged system. This strategy goes hand in hand with Pipe and Filter design pattern that we use in the system's document parser and GDPR detection engine and the structure of Angular applications. Both promote loose coupling which will lead to a design that is modular and efficient.

3 Quality Requirements

The following quality requirements are those deemed most important in the context of the purpose and function of the GDPR Non-Compliance Detector (GND). Due to the nature of the system and agile methodology employed, more quality requirements may be added at a later stage.

3.1 Security

Due to sensitive data accessed and processed, security is the most important quality requirement for the GND application. Since the system will have to access and process sensitive data it is important that it handles any data it uses securely. Documents should be sanitised before processing to remove any identifying features such as metadata, geolocation data and data embedded within. After processing, the document should not be stored and immediately deleted. Any information collected during processing that may be needed for training purposes should also be sanitised and stored anonymously.

3.2 Compliance

Due to the nature of the system and its purpose of detecting potential GDPR violations, it is imperative that it comply with the laws and regulations laid out by the GDPR act. Any data that the system needs, should be sanitised, anonymised, and promptly deleted after processing. Failure to adhere to GDPR regulations may lead to legal action taken against the owners of the product.

3.3 Performance

The client (Africa Talent by Deloitte) requires that the application be able to run in the background and be able to access any documents received via email. The system will have to access the user's attachments folder and automatically process incoming documents for possible GDPR violations. The system should be lightweight and consume as little resources as possible. Due to the need to regularly access a user's inbox and inform the user of possible GDPR violations in a document they may have just received, the system must have fast response times, be able to handle multiple documents at the same time, different document types and process large files as well.

3.4 Reliability

Due to the system needing to run in the background and monitor a user's inbox or be able to process documents uploaded manually, it is important the system be designed and implemented in a robust manner. A failure of the system may result in documents not being scanned for potential GDPR violations could result in legal and financial issues.

3.5 Usability

The primary users of the software will be people that may not have technical knowledge of the system and how it works. The program should therefore be easy to understand and navigate ensuring that the intended users can utilise the application with minimal knowledge of its implementation and any additional training.

3.6 Scalability

The application should have the capability to process multiple documents at the same time. This could be via manual file inputs or retrieving multiple email attachments. It should also be able to accommodate new filters that will be used to detect violations in future. Therefore, the system should be designed with these factors mind and multiple file processing should not affect performance of the app.

4 Architectural Diagram

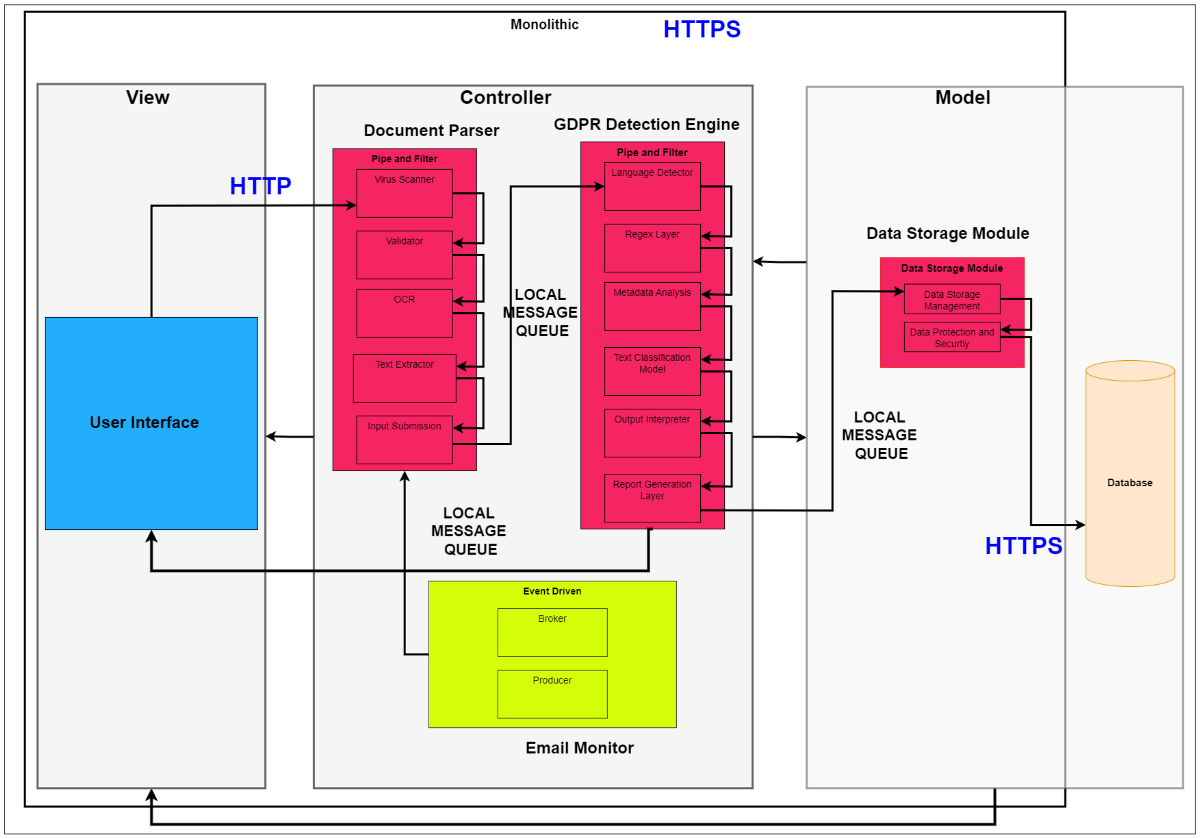

Figure 1: Architectural Diagram

5 Deployment Diagram

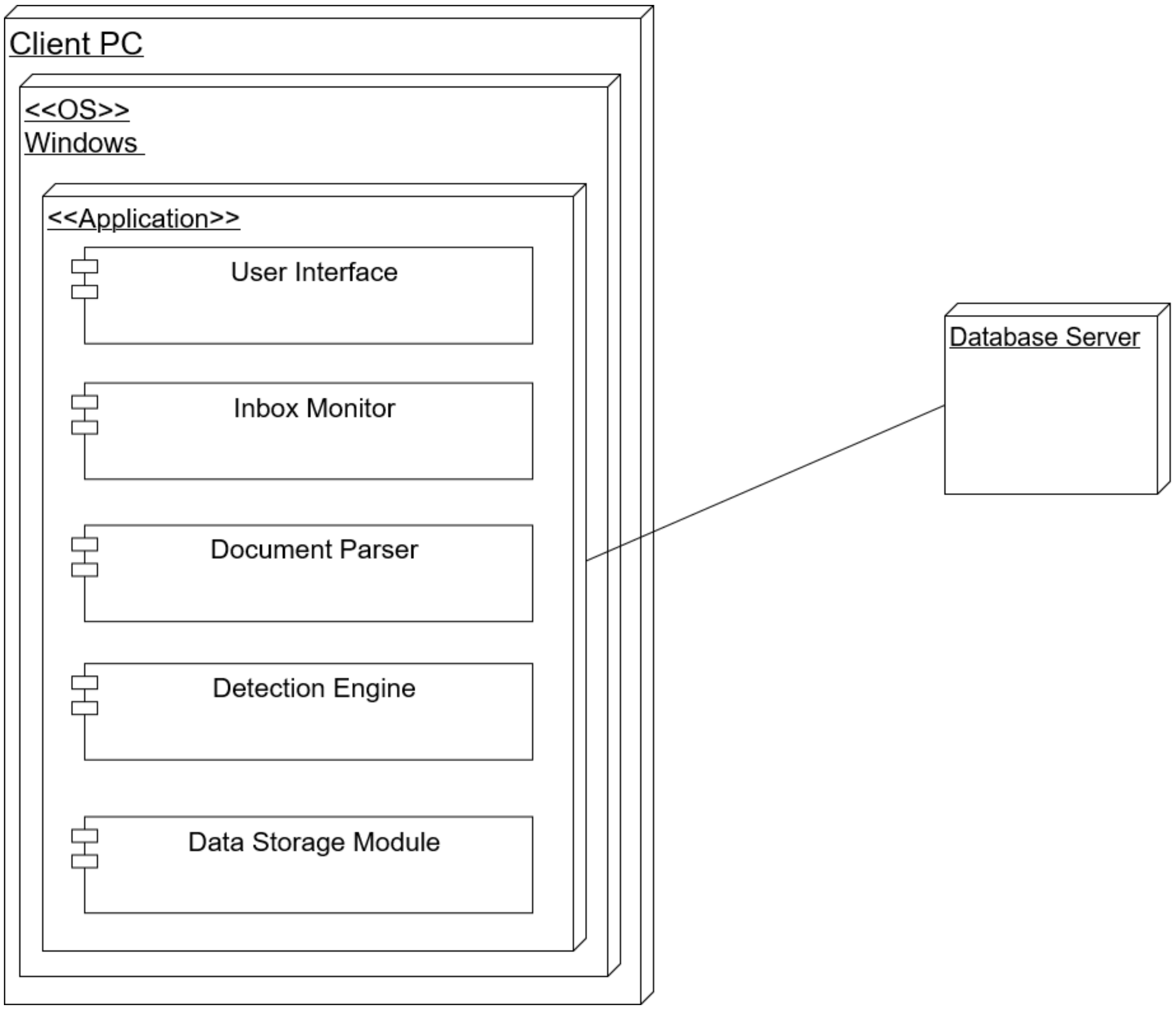

Figure 2: Deployment Diagram

6 Architectural Patterns

The following describes the architectural patterns that will be used in the design of the GND to achieve the required functionality of the system and how they can be used to enforce the quality requirements previously described.

6.1 Monolithic Pattern

As specified by the client, we are required to make use of a single system architecture. The Monolithic architecture aligns with this constraint, ensuring that all subsystems are encapsulated within one major system. The use of Monolithic architecture also aids in meeting system requirements, such as tight coupling, which can improve efficiency—a key concern—and the deployment of a single executable.

6.1.1 Quality Requirements Addressed

- **Security** - The single entity characteristic of the Monolithic pattern it is easy to enforce consistent security measures across all subsystems and the single point of entry makes it difficult for bad actors to access the system.
- **Performance** - Due to the tight coupling enforced by the Monolithic pattern, interaction between components is fast, efficient and overhead is reduced.

6.2 Model-View-Controller Pattern

MVC facilitates user interaction with the program, as the user will be able to specify input which will be processed by the system (Controller) through an interface (Model) and be able to view the resulting analysis (View) of the process carried out on the input. The use of MVC allows us to separate concerns and facilitates concurrent design, development, and testing of the frontend and backend features.

6.2.1 Quality Requirements Addressed

- **Performance** - By separating concerns of the different components, we can easily delegate the different types of interactions that the system will experience leading to modular, robust and efficient code.
- **Usability** - By separating concerns of the different components, we can easily delegate the different types of interactions that the system will experience leading to modular, robust and efficient code.
- **Reliability** - The different responsibilities of the different components allow easy error handling, debugging and different thresholds of fault tolerance.

6.3 Event Driven Pattern

The Event Driven pattern will be utilised primarily for the implementation of email inbox monitoring. Once a new email with an attachment is received, the program (which monitors the attachments folder of the email client) will access, process, and detail a report on whether the document is GDPR compliant and possible violations if it is not. The pattern can also be used to address certain events that may happen within the system itself.

6.3.1 Quality Requirements Addressed

- **Security** - When a document is uploaded manually or automatically accessed from an email inbox, event handlers can be used to sanitise, encrypt and anonymise the document before processing. Data can automatically be deleted after processing as well.
- **Reliability** - Events can be logged by the system to ensure that all possible violation checks are performed by the system and provide strong error handling assistance should a sub-system break down.

6.4 Pipe and Filter Pattern

The Pipe and Filter pattern is an ideal fit for the detection engine that will be used to detect possible GDPR violations within a document. The pattern allows for modular code where each data violation check is implemented as its own filter with a "pipe" and operates independent of each other. This promotes loose coupling within the engine and allows for more filters to be added later on without redesigning the entire system.

6.4.1 Quality Requirements Addressed

- **Scalability** - The pipe and filter pattern allows new checks/filters to be added to the detection engine without needing to change the design of the entire system.
- **Performance** - Each filter within the pipe can be run concurrently meaning that multiple documents can be processed at the same time, allowing near instant feedback for users on the status of documents.
- **Reliability** - Due to the loose coupling nature of the pipe and filter pattern debugging and error handling can be done independently and the failure of a single filter within the system will not affect the others.

7 Architectural Constraints

7.1 Client Defined Constraints

- The system must be file agnostic. This means that files of different types should be able to be handled and processed by the program without any input from the user.
- The system must be developed with a single system architecture.
- The deliverable must be a Windows executable and be integrated with Windows for automated scanning

7.2 Hardware and Operating System Constraints

- The program must be able to run on the Windows operating system. This is standard the operating system used across Deloitte computers.

8 Technologies Used

8.1 Frontend Development

8.1.1 Angular

Angular is a powerful Javascript framework that will use to build the user interface of the GND app. The clear separation of concerns and overall structure enforced by the MVC design approach will allow us seamlessly design the frontend of the application and display data in a clear and easy to understand manner.

Pros:

- Fast, responsive and efficient due to the use of Typescript.
- Ability to reuse components allows time efficient work.
- Large ecosystem backed by Google means robust and reliable functionality.

Cons:

- Steeper learning curve compared to alternatives such as React and Vue.
- Large amount of resources available may lead to increased complexity and issues when debugging.
- More resource intensive compared to alternatives

8.1.2 Electron

Electron is a framework that allows us to create the desktop application required by the client, Africa Talent by Deloitte, using web technologies such as HTML, CSS and Javascript. It will us to build an easily scalable application that may be used across many different systems.

Pros:

- Uses web technologies that the team is familiar with.
- Highly scalable.
- Efficient reuse of codebase that allows the creation of desktop and web based apps.

Cons:

- Executable generated may be quite large.
- Dependant on chromium.

8.1.3 Flowbite

Flowbite is a component library built on top of the Tailwind CSS framework. It provides prebuilt components that will the team to build the user interface efficiently and in line with the design and colours associated with the Deloitte platform.

Pros:

- Due to the use of Tailwind CSS "under the hood" all components are highly customizable to the developers needs.
- Utility first approach.
- Vast and active ecosystem regularly updating existing and creating new components.

Cons:

- Inline styling syntax reduces readability.
- No clear separation of concerns.
- Generates a large, inital CSS file due to the utility classes used in design

8.2 Backend Development

8.2.1 Python

Python is well equipped for data processing and analysis that will be required for processing large amounts of documents. It also provides a variety of libraries which can be used to develop reliable and performance driven modules such as a GDPR compliance detection engine, Document Parser and Data Storage Module. It provides solid cross-platform compatibility and a stable and consistent base for these modules as it can be used for integration of the system components.

Pros:

- Gentle learning curve.
- Offers a large and diverse set of libraries that may be used to assist developers.
- Cross-platform compatible.

Cons:

- Performance is much worse compared to alternatives such as C++ and Java.
- Poor dependency management may cause issues when collaborating with other developers.
- High memory usage.

8.3 Version Control

For version control, we will utilise GitHub. GitHub provides a highly efficient platform for managing and tracking changes to our codebase and provides a convenient way of handling each iteration of the system design. All group members are well versed in the use of GitHub and are also familiar with a GitHub development strategy which has proven effective for us and assists with the overall system development life cycle.

8.4 CI-CD

We are considering the use of GitHub for our CI-CD activities. GitHub will serve as the central repository for version control and code sharing, and GitHub Actions can be used for automating the build, testing and integrating processes.