Redesign and Optimization of the GIS Triage Process for Construction Permit Evaluations

Bachelor Thesis

Degree programme:	BSc Computer Science
Author:	Timo David Schlatter
Thesis advisor:	Marcel Pfahrer
Project partner:	Dominik Bauer
Expert:	Joachim Wolfgang Kaltz
Date:	15.06.2023

Bern University of Applied Sciences

School of Engineering and Computer Science
Computer Science

Abstract

The Bau- und Umweltschutzdirektion BL operates a system, called "eBaugesuche", to handle construction permits electronically. An in-house development department maintains and extends this system.
A key component of the system is the "GIS triage" process. In order to identify the specific departments and institutions to be involved in the assessment of the construction permit, this process evaluates geographical information at the location of the construction project.

The goal of the bachelor thesis is to redesign and optimize the GIS triage process and the corresponding system components. The current system's organic growth has led to significant drawbacks, including poor performance, high operating and maintenance costs, and limited extensibility. In order to address these issues, the redesign aims to integrate software components seamlessly into the existing "eBaugesuche" system, ensuring flexibility while resolving the aforementioned problems.

The project followed the SCRUM methodology, incorporating some of its principles. It commenced with a concept phase aimed at assessing various options related to architecture, communication and performance. Subsequently, the project progressed through five sprints, each dedicated to implement different aspects of the software.

The outcome of the project is a standalone .NET web service that offers seamless integration through HTTP requests. An HTTP post request with the construction application's location triggers GIS queries on the geo data warehouse. Crucial data, like proximity to monuments, is extracted and used in business rules evaluation. Based on this assessment, a specialized department may be selected to review the selection. Results are returned to the requester in JSON format.

Notably, the new service is much faster compared to its predecessor, primarily attributed to its extensive utilization of parallelization techniques. Furthermore, the enhanced service greatly facilitates maintenance and expansion efforts. Leveraging the open source “Microsoft Rules Engine”, the implementation of triage rules is now simpler, especially when compared to the previous reliance on a database trigger.

The project achieved its objective of delivering an improved GIS triage service. Further improvements include integrating it into the eBaugesuche system and implementing a user-friendly web interface for rule management, empowering business users in maintaining the rules independently.

Contents

1 Introduction

1.1 Task statement

The Bau- und Umweltschutzdirektion (BUD) BL operates a system, called "eBaugesuche", to handle construction permits electronically. An in-house development department maintains and extends this system.

A key component of the system is the "GIS triage" process. In order to identify the specific departments and institutions to be involved in the assessment of the construction permit, this process applies a series of queries in a cantonal geographical information system based on the geographical location of the construction project applied for [1].

1.2 Problem

The current system has serious disadvantages due to its organic growth. In particular, it has poor performance, is costly to operate and maintain and difficult to extend [1].

1.3 Goal

The goal of the bachelor thesis is to redesign and optimize the GIS triage process and the corresponding system components. The redesign is intended to solve the problems mentioned and the software components to be realised for this purpose are to be flexibly integrated into the overall "eBaugesuche" system [1].

1.4 Technology stack

Technologically, the project is based on C# and .Net for the backend, and on Angular und Devexpress for the frontend. For code and project management, deployment, and integration, the existing CI/CD environment GIT-Lab will be used [1].

2 Current State of the Triage Process

2.1 Crucial Parts of the Process

Important parts of the process need to be explained before looking at the specific whereabouts of the process.

2.1.1 The Geo Data Warehouse

A data warehouse is a comprehensive data management system that contains large amounts of data from multiple sources. It enables various business intelligence operations, in particular the execution of queries and the analysis of results [2]. In this context, a Geo Data Warehouse is specialised in handling geographical data. The GDWH (Geo Data Warehouse) is maintained by the GIS department and is used in this project.

2.1.2 GIS Queries

The core functionality of the application lies in the 'GIS queries', which are essential for retrieving specific information related to categories of interest for new buildings. For example, these queries can be used to quickly assess the risk of flooding. The term "GIS queries" refers to all SQL select queries that use geographic data. These queries are executed against the GDWH. The geographic data can be in the form of a geographic point or a GeoJSON parameter. These queries typically retrieve one or more values associated with a particular category and return the results. Examples of categories are 'heritage', 'heat network' or 'mobile phone antennas'. If there is no data available for a particular location in a particular category, the value returned will be null.

Here is the example query code for "10_Denkmalschutz_Bauinventar":

```
SELECT DISTINCT
    SITE_ID, PERIMETER, trunc(ST_DISTANCE(s.geom,ST_POINTFROMTEXT('POINT (xxx
yyy)',2056))) as distance
FROM
    LU_MOBIL.v_STRAHBER_POLY s, LU_MOBIL.v_MOBILFUNK_PKT m, av_ls.v_grundstueck
g1
WHERE
    m.mobilfunk_pkt_id = s.strahber_poly_id
    AND ST_WITHIN(ST_POINTFROMTEXT('POINT (xxx yyy)',2056), g1.geom)
    AND (ST_TOUCHES(s.geom, g1.geom)
        OR ST_INTERSECTS(s.geom, g1.geom)
        OR ST_WITHIN(s.geom, g1.geom))
ORDER BY
    distance;
```

2.1.3 The Database After-Insert Trigger

The results of the GIS queries provide valuable insights for new buildings. However, not all of these values are of equal importance, as some are simply nice-to-knows. To address this, the current process includes an after-insert trigger on the "GIS_Result" database table. A database trigger is fired automatically after an insert event [3]. Therefore, each time a query is run on the GDWH, the resulting data is stored in this table and the trigger is then activated.

The trigger contains several conditional statements. If one of these conditions evaluates to true, it triggers an additional insert operation into the "Fachstellenbericht" table. In the context of this project, this action is referred to as 'activation'. The new entry in the table indicates that a department of the canton should review the building application based on the query result.

2.2 How it Works

The existing GIS triage process operates within the eBau system, which is a Microsoft Access application. The following diagram illustrates the steps following the submission of a construction application by a citizen. Then, the application is submitted to the eBau system. The diagram starts with the BUD-BIT clerk having the eBau application open.

1. The clerk's first task is to manually add one or more geographical points to the construction application's building plans. These points play a crucial role in the triage process.
2. The clerk initiates the triage process by manually clicking a button. This step cannot be automated upon the arrival of a new application since it requires the intervention of a clerk for the first step.
3. The triage process begins and runs in a loop until all queries have been processed.
    1. A GIS query is run against the GDWH database.
    2. The result of the query is then inserted into the BBPlus.GIS_Result table.
    3. A database after-insert trigger is automatically triggered, checking the GIS results.
    4. If any of the checks return true, a new entry is added to the BBPlus.Fachstellenbericht table, activating a specific department. If the checks return false, no further action is taken.
4. The eBau frontend retrieves data from the BBPlus database and displays the GIS results along with the activated departments.

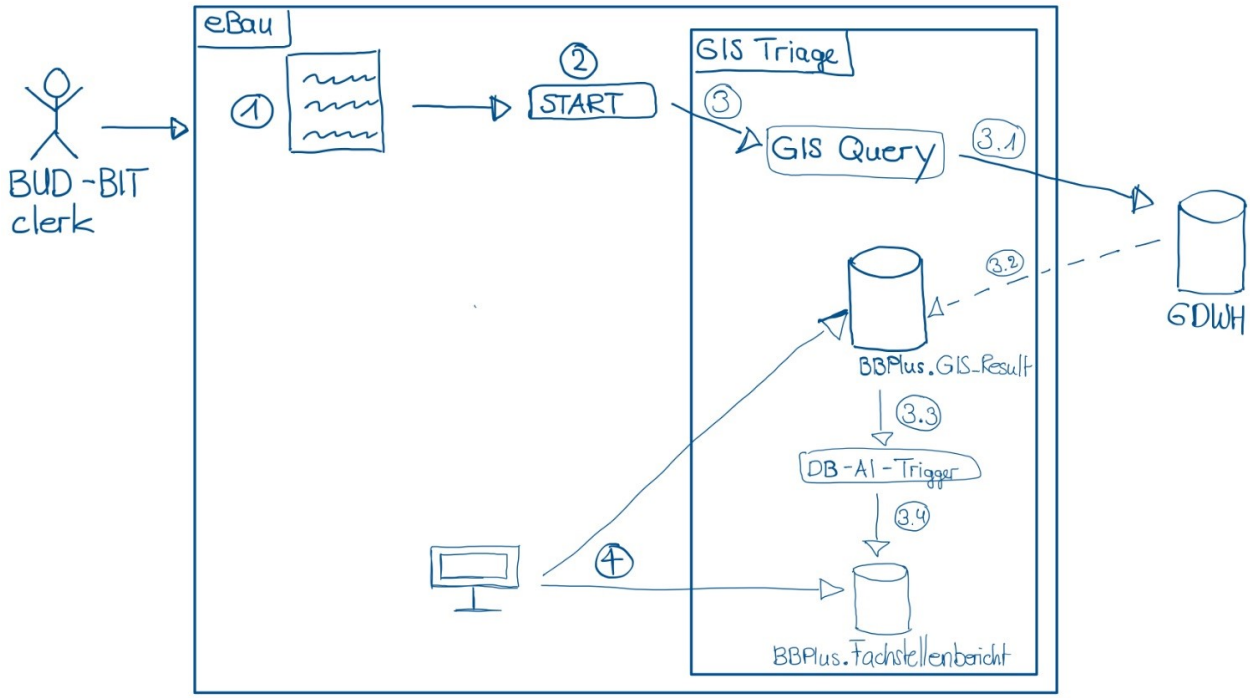

Illustration 1: GIS Triage Overview

2.3 Implementation Details of the GIS Triage Procedure

The Illustration 2 displays a sequence diagram which shows the high-level process of the triage procedure. The main cause of the diagram is to clarify where the intersections between the GIS triage and the eBau system are. For that, it consists of the "eBau Component" – representing the complete eBau system without the GIS triage, the "GISTriage" – representing the GIS triage - and the "BBPlus" – representing the used database.

The interfaces are highlighted with a yellow background. On step two eBau starts the GISTriage by invoking a procedure call with a parameter. And on step six eBau gets the data of BBPlus.Fachstellenbericht and BBPlus.GIS_Result and updates the application accordingly.

1. The system processes a construction application.
2. The component starts the GIS triage with a point coordinate or GeoJSON as parameter.
3. Get all GIS queries from the BBPlus.GIS_Query table.
4. The database returns all queries.
5. The GIS triage process is explained in Illustration 3 so as not to overload this diagram.
6. The eBau component selects all GIS results and fachstellenberichte.
7. The eBau component then updates its frontend. It shows if a specific BL department needs to have a look at the application.

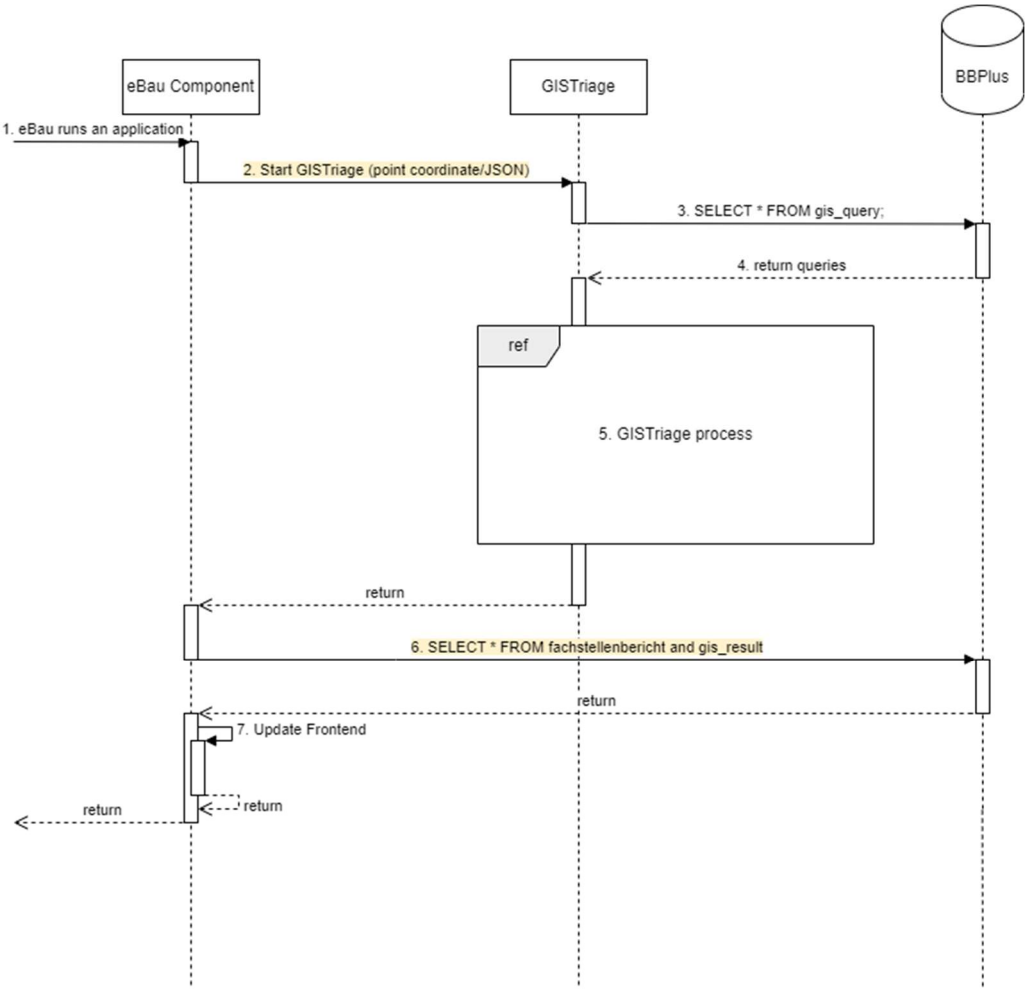

Illustration 2: Sequence Diagram: Implementation Details GIS Triage Procedure

The Illustration 3 displays the sequence diagram showing the detailed GIS triage process. The process works with three components: The “GISTriage”, which is the GIS Triage component, the database “BBPlus” and “GDWH”. The process starts by looping over all queries.

1. Executes the query with a point coordinate / GeoJSON on the GDWH.
2. The query returns a key-value pair as result. It may return no result.
3. If a result exists, the GISTriage inserts it into BBPlus.GIS_Result.
4. The insertion starts the after-insert trigger.
5. If a result meets the trigger condition, the trigger inserts a new entry into BBPlus.Fachstellenbericht

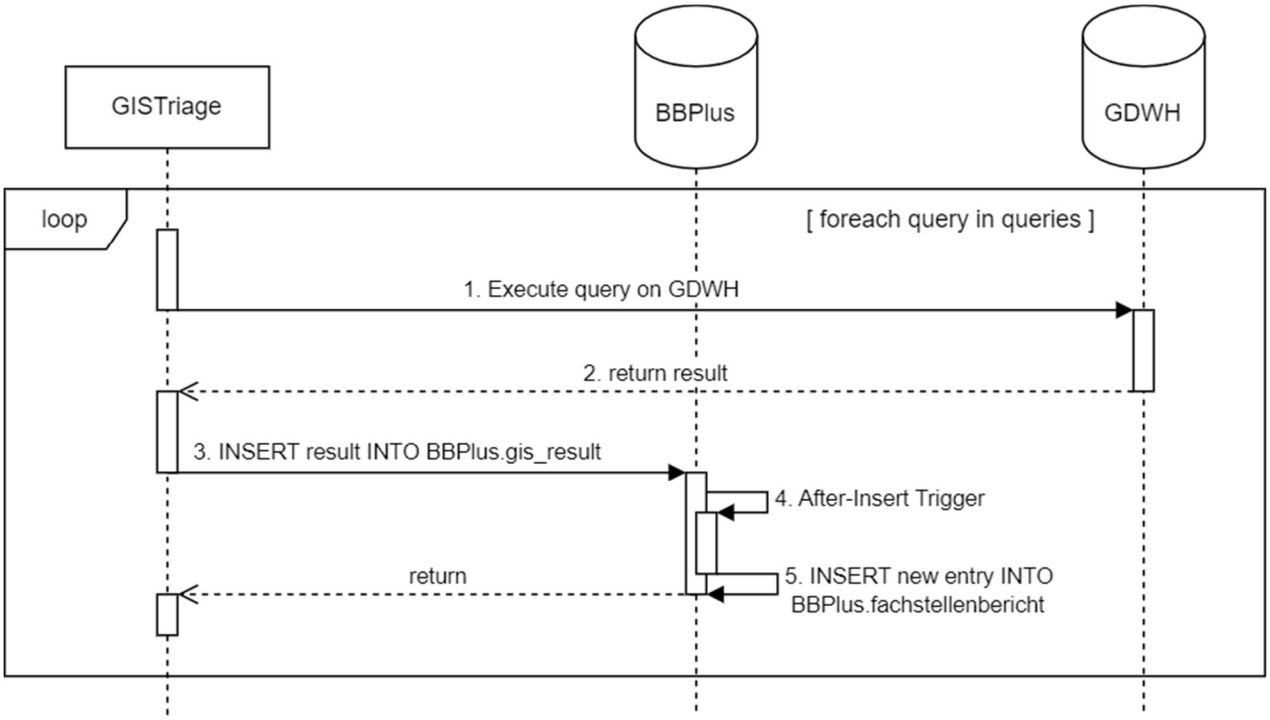

Illustration 3: Sequence Diagram showing the detailed GIS Triage

2.4 Addressing the Issues

The existing triage system has several significant drawbacks, in particular performance issues and a lack of maintainability and extensibility.

2.4.1 Performance Issues

The current trigger works as a sequential process, executing one query at a time and adding the results to the table "GIS_Result". This action in turn starts the after-insert trigger, potentially creating a new entry into the "Fachstellenbericht" table. Unfortunately, this sequential approach lacks efficiency, resulting in significant waiting times, particularly when an application contains a n enormous amount of geographical data. To improve performance, it would be beneficial to parallelize the process and leverage multiple CPU cores.

2.4.2 The Trigger

Currently, the trigger is designed to be executed after every single result. This results in many unnecessary conditionals being checked for each result. That redundancy not only affects performance but also leads to an overloaded trigger that includes numerous checks in the SQL query language. As a consequence, the trigger becomes challenging to read and comprehend, making it difficult to maintain an overview of its functionality. Moreover, the checks sometimes have different structures, which can further contribute to confusion.

The provided image shows a small part of the trigger:

```
-- BKSD: Augusta-Raurica: innerhalb Perimeter , Versand direkt ausgelöst
if :NEW.QUERY_NAME = 'Augustaraurica' then
   select count(*) into nID FROM FACHSTELLENBERICHT where BG_ID = :NEW.BG_ID and FS_ID=388 and VERSION=1;
   if nID = 0 then
       INSERT INTO FACHSTELLENBERICHT (BG_ID,FS_ID,VERSION,STATUS_ID,VERSAND_BEMERKUNG,VERSANDDATUM,FAELLIGDATUM,Nachtrag,BERICHT,PLANVERSAND)
                           VALUES(:NEW.BG_ID, 388,1,1803,'autom.Triage: '||:NEW.WERT,trunc(dGesch),trunc(dGesch)+10,nNT,-1,-1);
       insert into fb_unterlagen select FB_ID, nUNTID from fachstellenbericht where BG_ID = :NEW.BG_ID and FS_ID = 388 and STATUS_ID = 1803;
   end if;
end if;

-- SIT: Störfallbetriebe: innerhalb 120m (Wohnhäuser, Geschäftshäuser, Öffentliche Gebäude)
if nTyp=220 or :NEW.QUERY_NAME = 'StFV' or :NEW.QUERY_NAME = 'ZONE' then
    select count(*) into nID FROM FACHSTELLENBERICHT where BG_ID = :NEW.BG_ID and FS_ID=328 and VERSION=1;
    if nID = 0 and nTyp in (210,212,238) and :NEW.QUERY_NAME = 'StFV' and :NEW.QUERY_FELD = 'Distanz'  and :NEW.WERT < 120 then
       INSERT INTO FACHSTELLENBERICHT (BG_ID,FS_ID,VERSION,STATUS_ID,VERSAND_BEMERKUNG,VERSANDDATUM,FAELLIGDATUM,Nachtrag,BERICHT,PLANVERSAND)
                           VALUES(:NEW.BG_ID, 328,1,1803,'autom.Triage, innerh. Perimeter: '||:NEW.WERT||'m',trunc(dGesch),trunc(dGesch)+10,nNT,-1,-1);
       insert into fb_unterlagen select FB_ID, nUNTID from fachstellenbericht where BG_ID = :NEW.BG_ID and FS_ID = 328 and STATUS_ID = 1803;
    -- alle Fabrikanlagen
    elsif nID = 0 and nTyp=220 then
       INSERT INTO FACHSTELLENBERICHT (BG_ID,FS_ID,VERSION,STATUS_ID,VERSAND_BEMERKUNG,VERSANDDATUM,FAELLIGDATUM,Nachtrag,BERICHT,PLANVERSAND)
                           VALUES(:NEW.BG_ID, 328,1,1803,'autom.Triage, Fabrikanlagen',trunc(dGesch),trunc(dGesch)+10,nNT,-1,-1);
       insert into fb_unterlagen select FB_ID, nUNTID from fachstellenbericht where BG_ID = :NEW.BG_ID and FS_ID = 328 and STATUS_ID = 1803;
    -- alle Um/Anbauten in Zonen G/I
    elsif nID = 0  and nTyp = 216 and :NEW.QUERY_NAME = 'ZONE' and :NEW.QUERY_FELD = 'Zone Kantonal'
       and (:NEW.WERT like 'Industrie%' or :NEW.WERT like 'J%' or (:NEW.WERT like 'G%' and :NEW.WERT not like 'Grün%' and :NEW.WERT <> 'G Gärtnerei')) then
       INSERT INTO FACHSTELLENBERICHT (BG_ID,FS_ID,VERSION,STATUS_ID,VERSAND_BEMERKUNG,VERSANDDATUM,FAELLIGDATUM,Nachtrag,BERICHT,PLANVERSAND)
                           VALUES(:NEW.BG_ID, 328,1,1803,'autom.Triage, Um-/Anbauten Zone: '||:NEW.WERT,trunc(dGesch),trunc(dGesch)+10,nNT,-1,-1);
       insert into fb_unterlagen select FB_ID, nUNTID from fachstellenbericht where BG_ID = :NEW.BG_ID and FS_ID = 328 and STATUS_ID = 1803;
    end if;
end if;

-- ARP Kantonsplanung (ARP KP 424): prov.Gewaesserraum GWR
```

Illustration 4: Part of the After Insert Database Trigger

3 Requirements Analysis

3.1 Functional Requirements

The Table 1 shows all functional requirements and their priorities.

Nr.	Requirement	Priority
1	The service must support all current features of the triage process.	Must-have
2	The queries must be maintainable without changing the running system.	Must-have
3	The trigger must be maintainable without changing the running system.	Must-have
4	The trigger message must be maintainable without changing the running system.	Must-have
5	The GIS Query Checker application needs to be updated to work with the new service.	Nice-to-have
6	The queries and trigger should be maintainable by business users.	Nice-to-have

Table 1: Functional Requirements

3.2 Non-functional Requirements

The Table 2 shows all non-functional requirements and their priorities.

Nr.	Requirement	Priority
1	The application is designed as a web service.	Must-have
2	The application is clearly separated of eBau.	Must-have
3	The application must be easy to maintain.	Must-have
4	The application must be easy to extend.	Must-have
5	The application should be faster than the current implementation.	Must-have

Table 2: Non-functional Requirements

3.3 User Stories

The Table 3 shows user stories.

Nr.	Description	Implementation
1	As a BUD-IT developer, I want to test the new web service with a test request, so that I know that the request and response format is correct.	Sprint 1
2	As a BUD BIT clerk, I want to have an automatic execution of the gis queries against the gdwh, so that each construction application has their gis results for the end user to see.	Sprint 2
3	As a BUD-IT developer, I want to change the SQL queries without changing the source code, so that I can modify them without a new release.	Sprint 3
4	As a BUD-IT developer, I want to be able to change the rules outside of the application. It should also be easy to maintain.	Sprint 4
5	As a BUD-IT developer or clerk, I want to be able to test the queries and activation remarks by using the test tool GIS query checker.	Not implemented
6	As a BUD-IT developer, I want to be able receive the translated query result fields through a GET request.	Sprint 5

Table 3: User Stories

3.4 Business View

The diagram below illustrates the key business aspects which the new application should address. Notably, the GIS Triage Service and eBau are clearly separate components that interact through a request-response mechanism. To ensure flexibility, the queries should be configurable by the BUD-IT without requiring application redeployment. However, due to their SQL-based nature, it might not be feasible for business users from the BUD-BIT department to configure these queries directly. Additionally, the replacement of the trigger should also be configurable without redeployment. In this scenario, the BUD-BIT department could potentially handle the configuration.

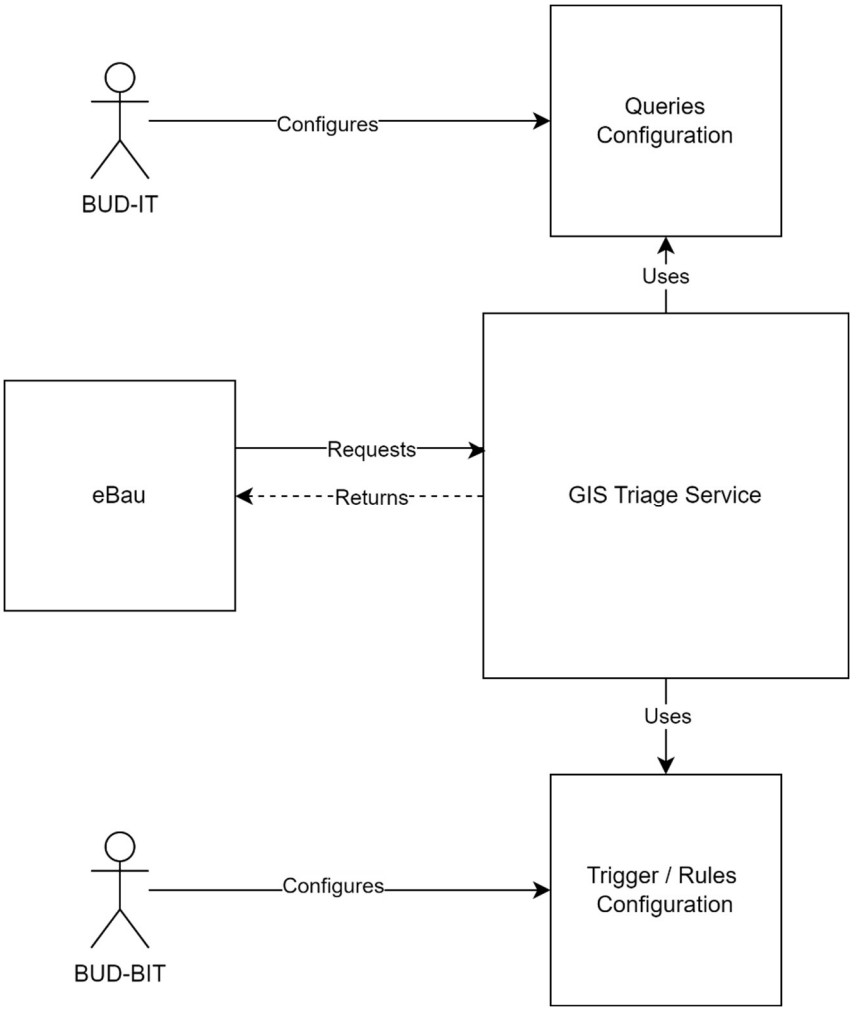

Illustration 5: Business Components View

4 Comparative Analysis: Future Triage Process

4.1 Communication

Two possibilities are feasible for the communication between the GIS Triage service and eBau:

1. Using a message service
2. Remote procedure call

Both of them have different pros and cons.

4.1.1 Using a Message Service
In messaging communication, the parties do not communicate directly with each other. Instead, they use a message broker for all communication [4]. The Illustration 6 shows an example of the communication between two clients using a message service.

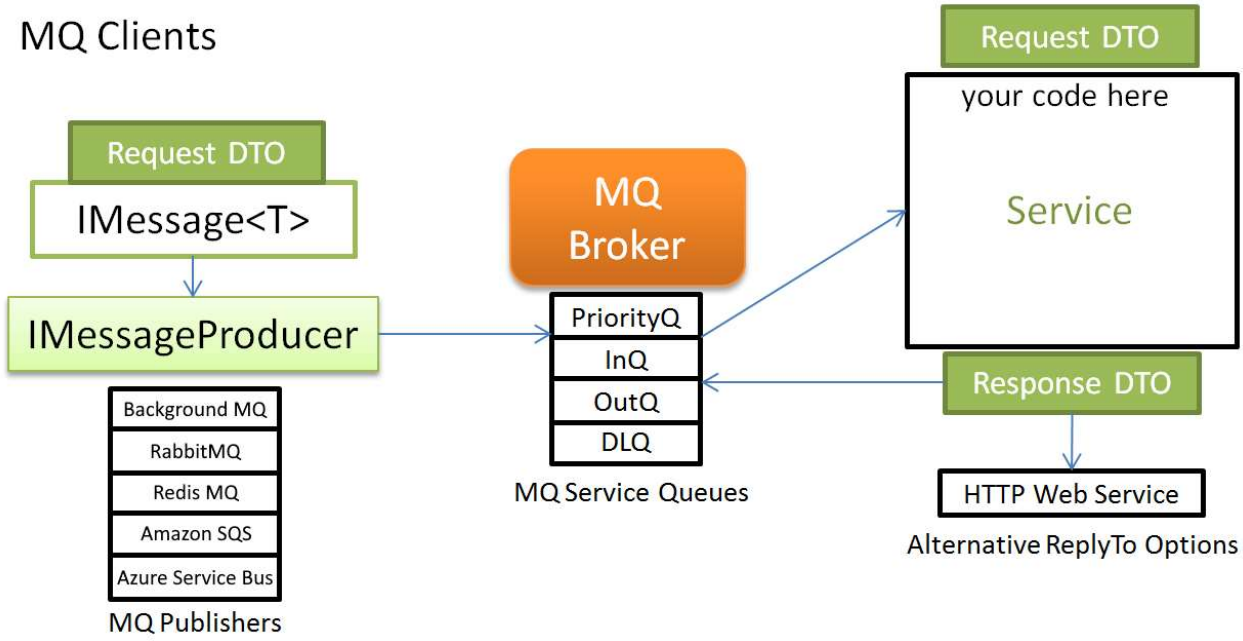

Illustration 6: Message Service Architecture
Source: [5]

Loose coupling is a major advantage of this method. Clients only have a connection to the broker. They do not need any additional information about the receiver except the address. In addition, the asynchronous communication is another advantage: a thread is not blocked while it is waiting for an answer. It is immediately free.
However, such a service always adds more complexity to the overall system. It needs to be managed and maintained.

The messaging service is not an ideal solution for this project. The business process for handling construction applications is sequential. There is no need to run multiple iterations of GIS triages simultaneously. Therefore, it does not take advantage of a major benefit of the messaging service. Furthermore, BUD-IT wants to reduce the overall complexity. By using this approach, it would be increasing it with a new service to manage.

4.1.2 Remote Procedure Call
The remote procedure call (RPC) communication pattern uses request and reply messages between a client/server. The procedure works as follows: A client sends a request message to the server. The server then processes the request. After processing, the server sends a response message back to the client. While the server computes the task, the client blocks the execution thread waiting for the reply [6].

The Illustration 7 shows a remote procedure call between a client and a server.

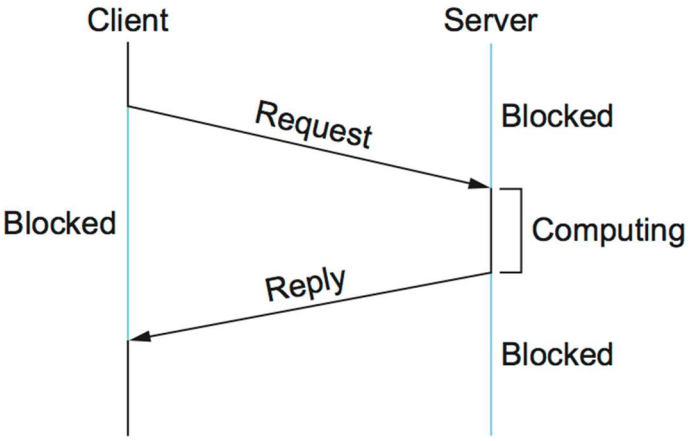

Illustration 7: RPC example
Source: [6]

This approach is straightforward to implement. It does not require any additional service to work. Furthermore, It is possibly faster than the messaging approach because the request is sent directly to the server.
However, the requesting thread is blocked until the server has finished processing it. This could lead to a memory problem if there are many threads blocking at the same time [7].

Considering the advantages and disadvantages, remote procedure calls are a good solution. It is fast, easy to implement and the disadvantages are not a problem in this case.

4.1.3 Summary
The Table 4 compares the two solutions against each other.

	Messaging service	Remote procedure call
Loose coupling	✓	✓
Async	✓	✗
Additional service needed	Yes	No
Fast	Fast	Very fast
Preferable solution	✗	✓

Table 4: Compares message service against remote procedure call

Considering that there is no need for asynchronous communication and no one wants to maintain an additional communication service, the remote procedure call is a better solution than the messaging service.

4.2 Rules Configuration

A rules configuration should replace the after-insert trigger. The use of this configuration is to determine which department should be activated. For this, it must check many conditions. If one of them should be true, a new activation remark – which is the replacement of the entry to BBPlus.Fachstellenbericht - is created.

4.2.1 Requirements

The requirements are as follows:

- The rules must be configurable. There should not be the need to recompile the application.
- If possible, business users should be able to understand and update the rules.
- Where possible, a user should be able to update the rules via a web interface.

4.2.2 Possibilities

There are two options that comply with the requirements:

- Scripting Engine
- Rules Engine

A scripting engine is a script code interpreter. It can execute script code at runtime [8].
A rules engine manages business rules. The rules are often "if-then" patterns, e.g. "If A then B else C" [9].

Both of them have advantages and disadvantages. But since the trigger implementations are mostly business rules, the rules engine appears more appropriate for this use case as the scripting engine allows for much more not needed functionality.

4.2.3 Comparing Different Types of Rules Engines

The Table 5 shows the comparison of different rule engines.

	Microsoft Rules Engine	NRules
Open Source	✓	✓
Licence	MIT	MIT
Latest Commit (23.03.2023)	3 days ago	4 months ago
Language	C#	C#
Community size	Small	Very small
Web interface	Through third party project	✗
URL	https://microsoft.github.io/RulesEngine/	https://github.com/NRules/NRules
Preferable Solution	✓	✗

Table 5: Comparison rule engines
Sources: [10] [11] [12]

The one rule engine that stands out the most is the Microsoft Rules Engine. It is open source, well-documented and maintained and allows the rules to be stored in a file, database or cloud. There is also an additional project that has created a web interface for it. Finally, it has a bigger community than its competitor, NRules.

4.3 Queries Configuration

The queries configuration should allow to maintain and extend the GIS queries without recompiling.

4.3.1 Requirements

The requirements are as follows:

- The queries must be stored outside of the application. There should not be the need to recompile the application.

4.3.2 Storage Possibilities

There are two storage options that comply with the requirements:

- Database
- SQL-Files

The current triage is based on a database approach. The queries are all stored in the BBPlus database. Contrary, the file based approach would use one or more SQL files for the queries. The files would then be read into the application where they are further utilized.

4.3.3 Summary

The Table 6 compares the two solutions against each other.

	Database	SQL-Files
Easy to maintain	(✓)	✓
Version control	Hard	Easy
Preferable solution	✗	✓

Table 6: Queries Configuration Comparison

In both methods the queries could be updated through an editor. But with the file-based approach, the queries could be saved immediately. Contrary, with the database approach, the queries must then be inserted into the database using an SQL statement. In addition, the queries only need to be read into the application. There is no need for complex database operations. Finally, as the file-based approach allows the queries to be easily added to version control, it is the better choice for the project.

4.4 Choosing the API Framework

The main purpose of the API is to start the triage process. This is done using a remote procedure call: The client initiates the process with a request. While the server processes the request, the client waits for the response. When the process is complete, the server returns a response to the client.
There are several frameworks and architectures that support this kind of behaviour: REST, JSON-RPC, GraphQL or gRPC. As REST and GraphQL focus on resources and not the execution of actions, they are not considered further in this case.

4.4.1 gRPC

gRPC is a framework developed by Google and stands for Google Remote Procedure Calls. It is an architectural service framework that uses buffers for high efficiency.
It's main advantages are high scalability and efficiency. While the main disadvantages are the limited browser support and that the messages are not human readable because they are sent in binary format [13] [14].

4.4.2 JSON-RPC

JSON-RPC is a method of invoking actions on distributed systems by sending messages in JSON format. It is similar to XML-RPC, which it has replaced. Its main advantage is its simplicity. It can also be used with any transport protocol as it is protocol independent [15].

4.4.3 Summary

The API is quite simple with only a few endpoints. The choice of frameworks and protocols listed would be JSON-RPC because of its simplicity.

However, since simplicity really is the goal, none of these protocols will be used. Even JSON-RPC adds another layer on top of the API. Instead, the remote procedure call will be sent via HTTP Post and the web API will be described via Swagger.

5 System Design

5.1 Architecture

5.1.1 High-Level View

The Illustration 8 shows the architecture of the new GIS service. It uses the following components:

- GDWH: The GIS queries are executed on the GDWH.
- GIS API: The service provides an API to start the triage process.
- Controller: The controller handles all incoming request.
- Queries Configuration: The queries are stored in JSON files. They can be maintained using a text editor.
- Microsoft Rules Engine: This engine is used to maintain the rules.
- Rules Configuration: The rules and workflows are stored in JSON files.
- The Rules Engine Editor is a web interface for the Microsoft Rules Engine. It allows to maintain the rules. However, this was not implemented.

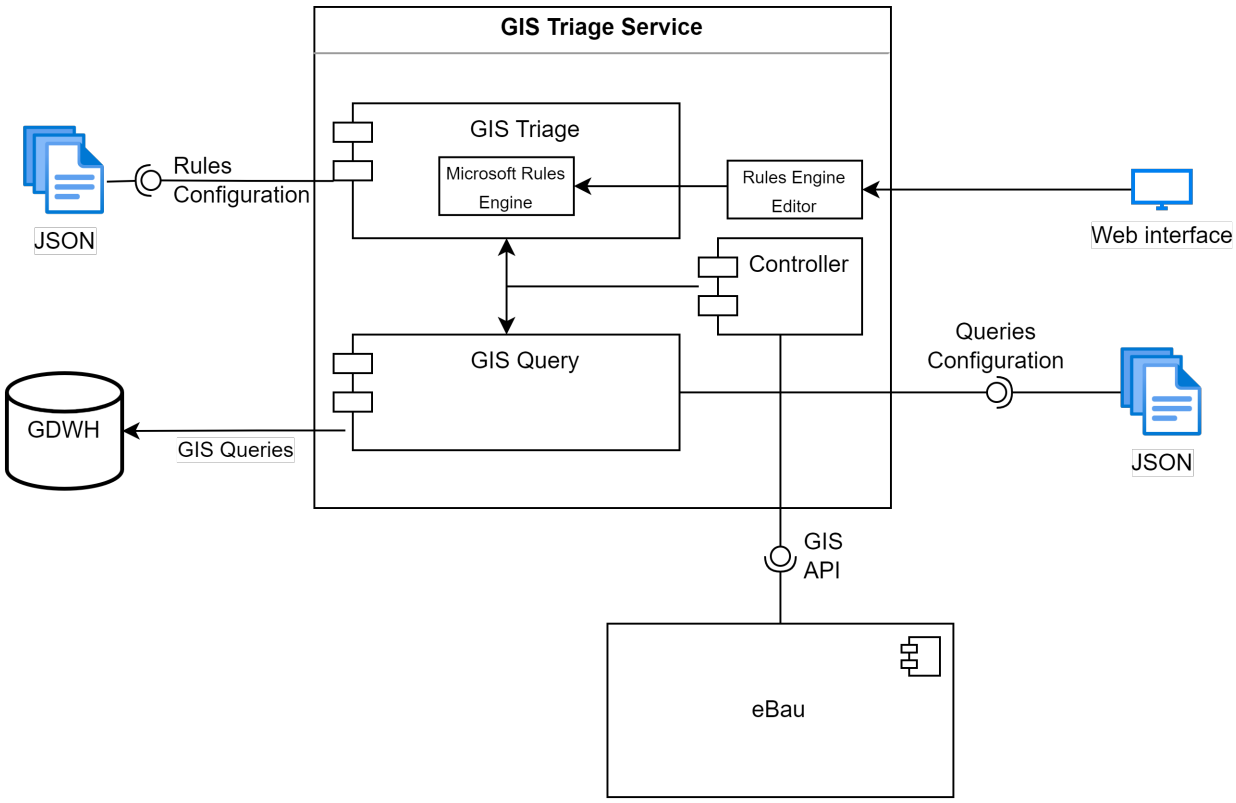

Illustration 8: High-Level View of the Architecture

5.1.2 Class Diagram

The Illustration 9 shows an overview of the classes.
The service receives requests from external clients, such as eBau, which are then managed by the TriageController. This controller handles the request and response flow, distributing the workload to the services. The QueriesService is responsible for executing queries in parallel, while the TriageService handles triage-related tasks. The repositories, utilizing the repository pattern, ensure controlled data access for these services.

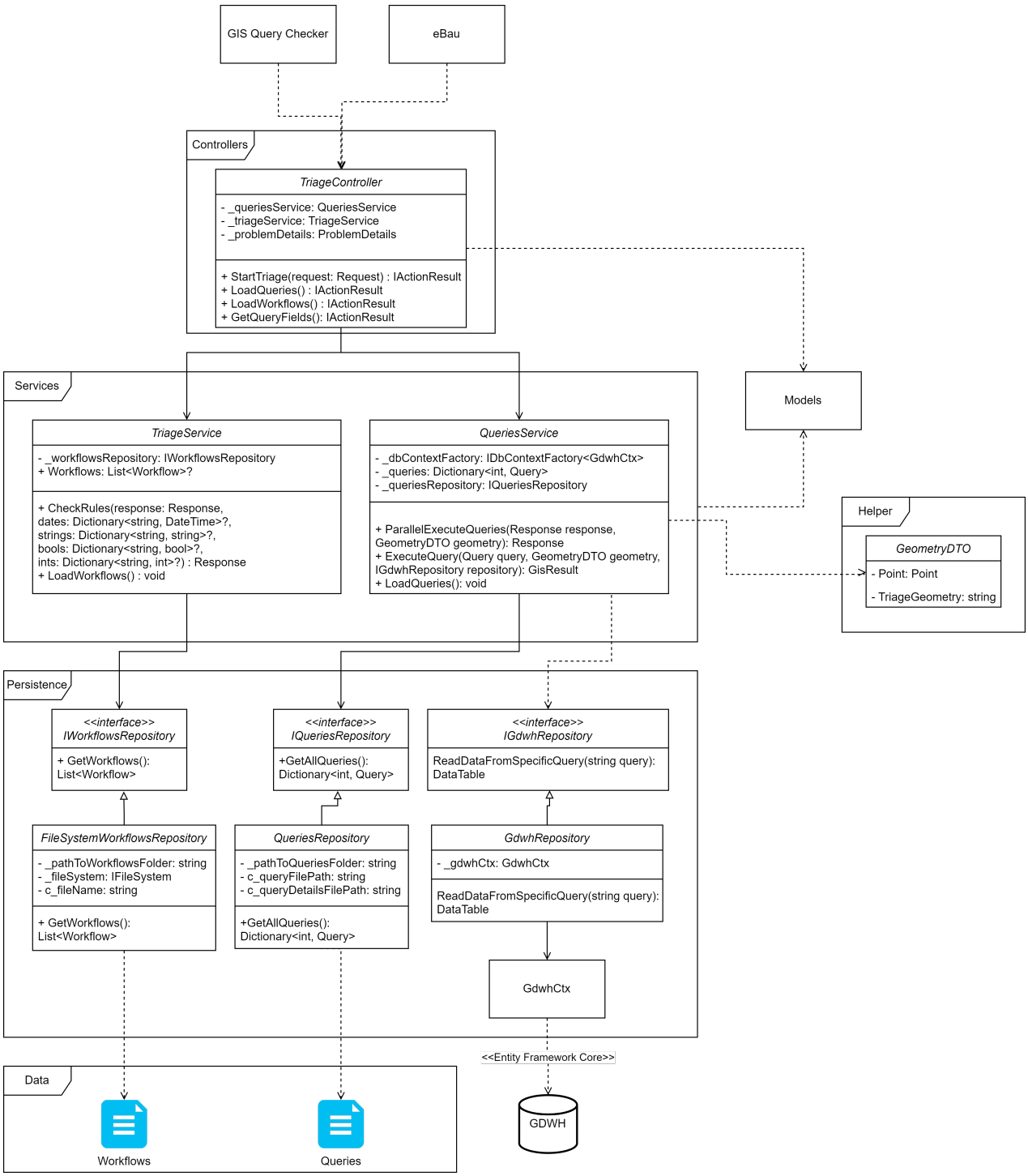

Illustration 9: Class Diagram

The Illustration 10 shows the class diagram of the models.

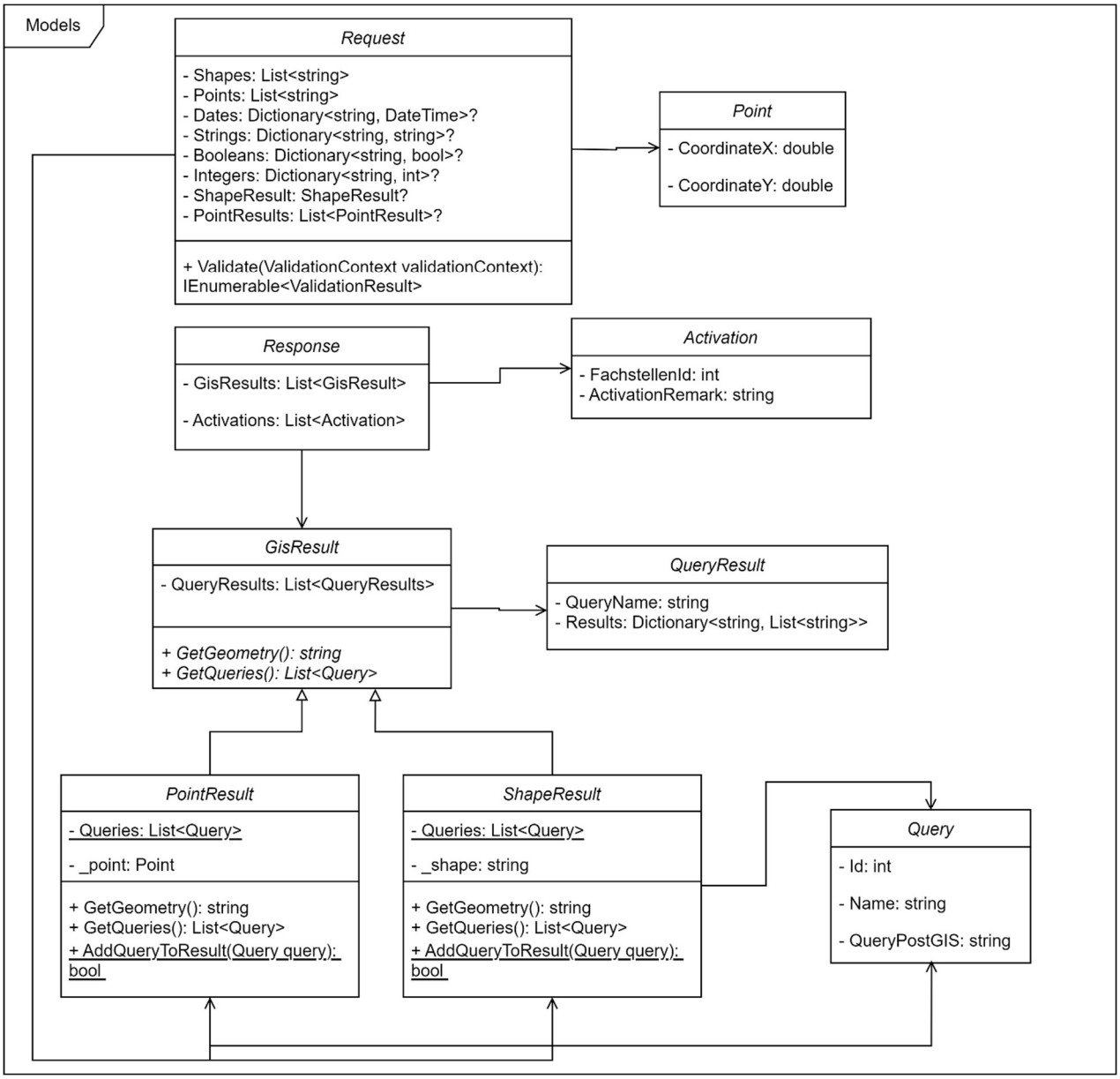

Illustration 10: Models Class Diagram

5.2 API

The documentation of the API is done using the OpenAPI specification. Starting the service in development mode automatically loads the documentation. Alternatively, the documentation can be viewed publicly using the link below.

Link: https://app.swaggerhub.com/apis/timodavid.schlatter/gis-triage-api/v1

Endpoint getQueryFields
The GDWH query results are directly provided in the response as they are. However, the column names used as keys are technical variable names, which are not very user-friendly when displayed in the eBau application for other users. To ensure a clear distinction between the presentation layer (eBau) and the data layer (triage service), the data is still returned as it is, but alongside it, the translated, user-friendly names can be obtained for each query. By utilizing the getQueryFields endpoint, it becomes possible to select the query fields with their translated names.

5.3 Process Workflow

The following chapter presents an in-depth exploration of the new process, providing a better understanding of its inner workings and key components. In order to provide a clear understanding of the process, a series of sequence diagrams have been developed as a visual aid to illustrate the sequential flow and dependencies of the various elements of the system. To aid comprehension and to avoid overwhelming the size of the diagram, it has been divided into three distinct parts, with a description for each part.

5.3.1 Triage Controller

The next illustration portrays the sequence diagram designed for the triage controller. To ensure its clarity, only the essential actions have been included and to provide a comprehensive understanding, the following list elaborates on the different components of the diagram in greater detail:

1. Start point: The diagram begins with a request being sent by either a user or a system. The specific format and structure of this request are explained in detail in the chapter titled “Request Format”.
2. LoadQueries: If the queries have not been loaded, this function is executed. Its purpose is to load the necessary queries required for processing the request effectively.
3. LoadWorkflows: Similarly, if the workflows have not been loaded, this function is executed. It aims to load the relevant workflows needed for the subsequent steps in the process.
4. ParallelExecuteQueries: This function is initiated, and it takes as input the response and the geometryDTO object. The geometryDTO serves as a Data Transfer Object, containing essential information such as points and shapes associated with the initial request. By invoking this function, the system begins parallel execution of the queries using the provided response and geometryDTO.
5. CheckRules: In order to finalize the process and obtain activation remarks, the CheckRules function is triggered. It takes the response object and additional parameters from the initial request as input. These additional parameters are utilized within the rules engine, enabling the system to apply relevant rules and generate activation remarks based on the provided response and parameters.

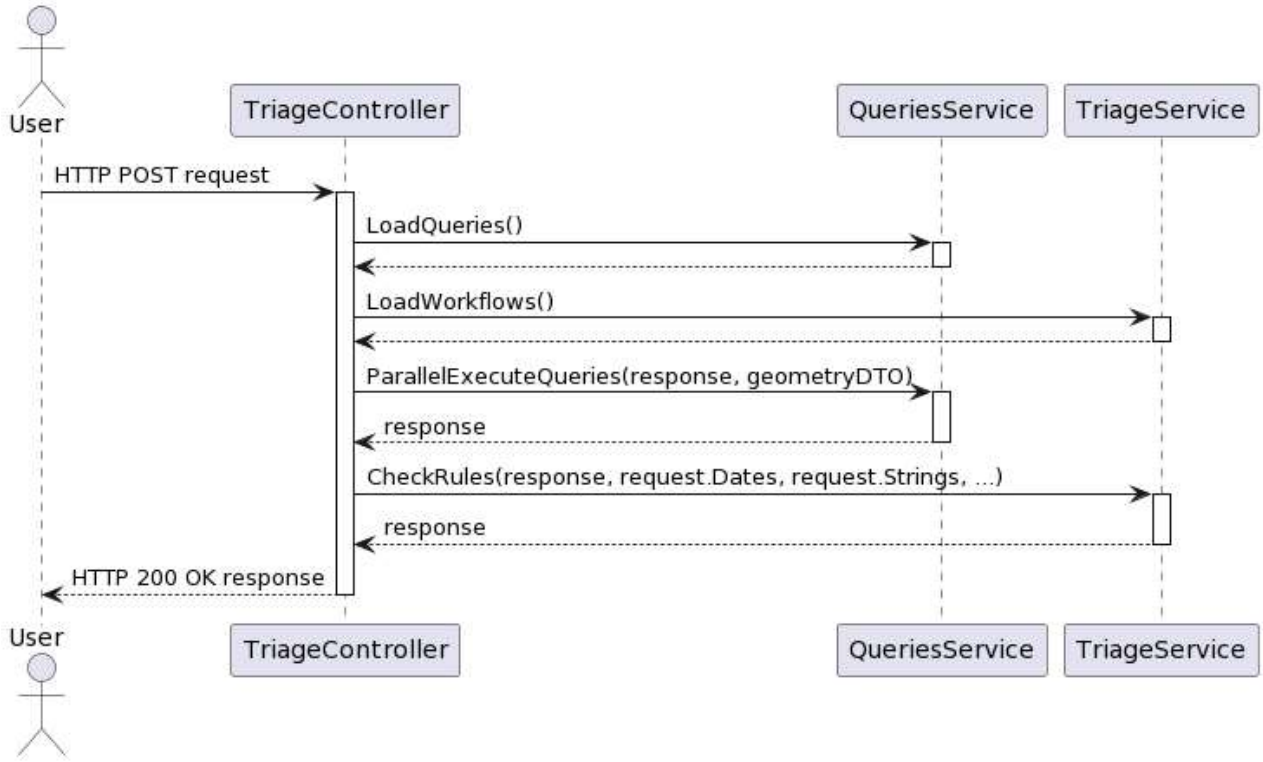

Illustration 11: TriageController Sequence Diagram

5.3.2 Queries Service

The primary responsibility of the queries service is to handle all actions related to the queries. Its key task is to execute the queries using the input coordinates of points and shapes in parallel, which significantly enhances performance compared to sequential execution.

Let's delve into the step-by-step breakdown of the queries service process shown in Illustration 12:

1. Loop over gisResult in response.GisResults: This step involves iterating over each GIS result within the response. A GIS result can represent either a point or a shape.
2. Loop using Parallel.ForEach: At this stage, the parallelization process is initiated. Within each GIS result, the query requests are parallelized, enabling simultaneous execution.
3. CreateDbContext: To ensure thread safety and compliance with the Entity Framework, each thread requires its own gdwhCtx (database context).
4. GdwhRepository: Additionally, for each thread executing the query requests, a GdwhRepository is created. This repository supports the execution of the query requests within the respective threads.
5. ExecuteQuery: This function carries out the execution of a single query on the database, returning a QueryResult object. The following sub-steps occur within this function:
    a. ReadDataFromSpecificQuery: This sub-step involves executing a query and obtaining a result table.
    b. ReadSqlResult: The result table obtained from the previous step is transformed into a dictionary format, which is then returned.
6. After adding all QueryResults to the response and upon the completion of each thread's execution, the response containing the aggregated QueryResults is returned back to the caller.

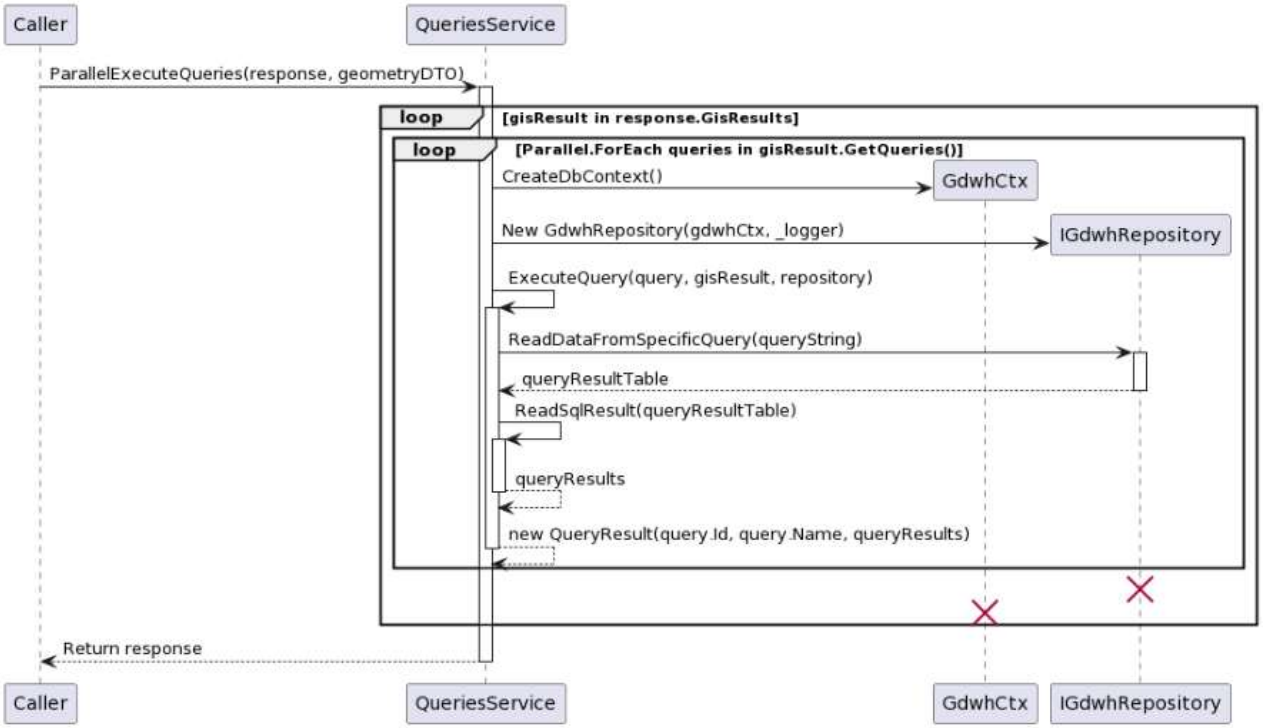

Illustration 12: Queries Service Sequence Diagram

5.3.3 Triage Service

The triage service plays an important role in utilizing the Microsoft Rules Engine, encompassing various operations related to it. The core focus lies in the rule evaluation process on the GIS results. A more detailed explanation and visual representation can be found in Illustration 13.

1. TransformQueryResultsToJObjects: This function is responsible for transforming the query results into JObjects. These JObjects are utilized within the rule expressions, enabling the utilization of JObject functions.
2. CreateRuleParameters: In this step, rule parameters are created. This enables the identification of variables by their respective names within the expressions. Without this step, each input would have an incremental name like "input1," which might lead to ambiguity.
3. ExecuteRules: This function executes the rules using the provided input and filters the activation remarks. The following sub-steps occur within this function:
    a. Create ReSettings: This sub-step allows the utilization of custom functions within the expressions, enhancing the flexibility and extensibility of the rule engine.New RulesEngine: The RulesEngine object is created at this stage, serving as the core component for rule evaluation.
    b. ExecuteAllRulesAsync: This sub-step executes all rules within the given workflow for all inputs, returning a RuleResultList.
    c. GetActivationRemarkFromRuleResultTree: If a rule was successful, this function retrieves the activation remark associated with that rule and adds it to a dictionary.
4. Add all activations, including the department ID and activation remark, to the response.
5. Finally, the response is returned back to the caller.

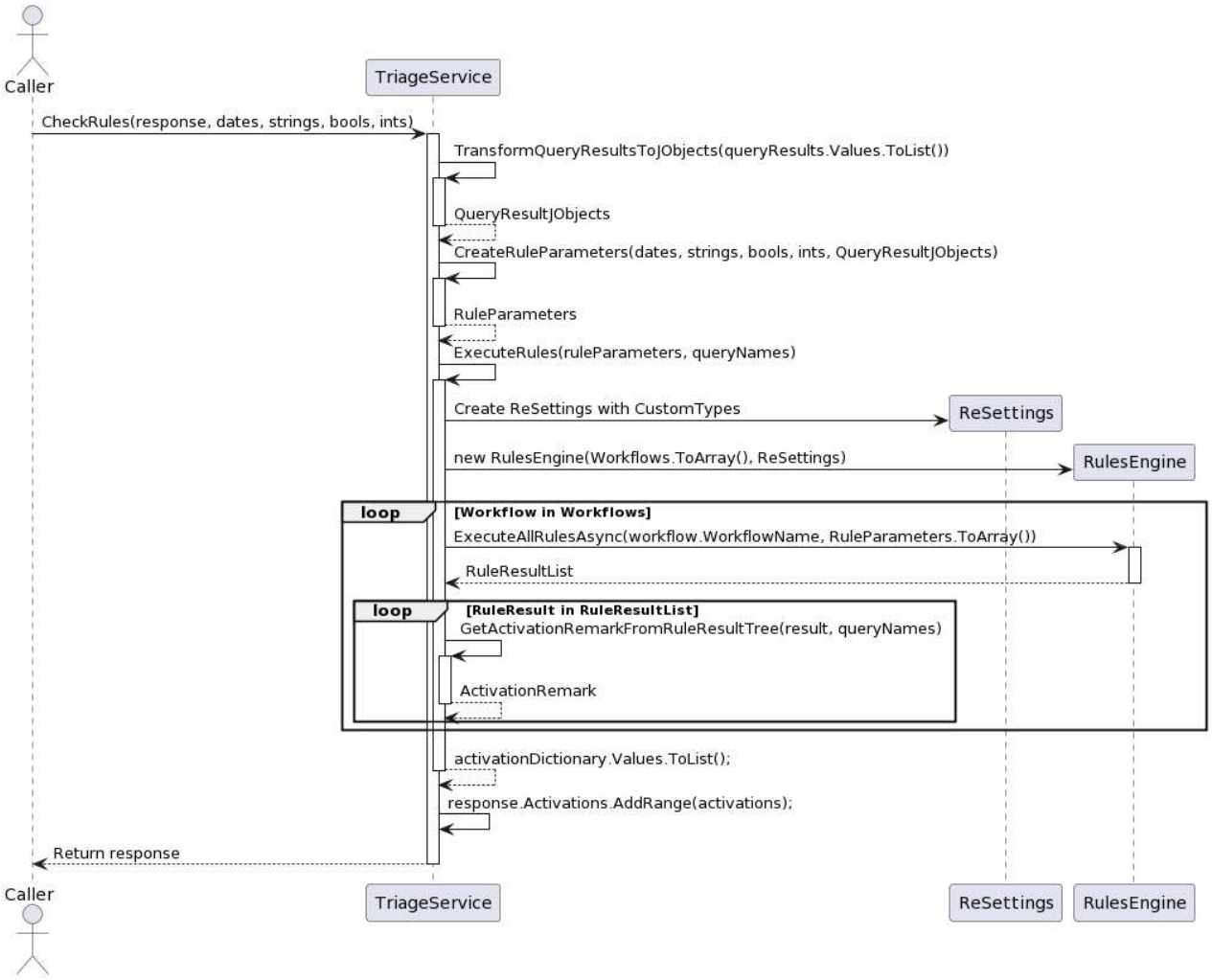

Illustration 13: Triage Service Sequence Diagram

5.3.4 Request Format

The table below shows each input of a request, its description and whether it is required.

Key	Value	Required
Shapes	Array of strings. Each string should be a valid geoJSON object. There is no validation, however.	True
Points	Array of strings. Each string should be a geoJSON point: "POINT(xxx yyy)". The service validates that the part (xxx yyy) exists.	True
Dates	Dictionary<string, string>. The value must be parsable into a DateTime object. This is an additional parameter that can be used in the expression of the rules engine.	False
Strings	Dictionary<string, string>. This is an additional parameter that can be used in the expression of the rules engine.	False
Booleans	Dictionary<string, bool>. This is an additional parameter that can be used in the expression of the rules engine.	False
Integers	Dictionary<string, int>. This is an additional parameter that can be used in the expression of the rules engine.	False

Table 7: Description of each Request Input

Illustration 14 shows the input format scheme.

```
{
  "shapes": [
  ],
  "points": [
  ],
  "dates": {
    "key": "dateTimeValue"
  },
  "strings": {
    "key": "stringValue"
  },
  "booleans": {
    "key": true
  },
  "integers": {
    "key": 210
  }
}
```

Illustration 14: Input Format Scheme

The Illustration 15 shows an input example.


```
{
  "shapes": [
    "POLYGON((2627973.6339951 1254304.548002,2627980.2419907 1254307.8530007,2627988.32900
  ],
  "points": [
    "POINT(2627981.111998 1254296.9829995)"
  ],
  "dates": {
    "baugesuch_geschaeftsdatum": "2018-06-11T00:00:00"
  },
  "strings": {
    "baugesuch_prj_ort1": "Lausen"
  },
  "booleans": {
    "baugesuch_neuauflage": true
  },
  "integers": {
    "baugesuch_gebauedetyp_id": 210
  }
}
```


Illustration 15: Input Format Example

5.3.5 Response Format

In Illustration 16, the output format scheme is presented, consisting of two primary components: GIS results and activations. The GIS results encompass the outcomes obtained from executing queries on the GDWH. Each GIS result object represents a point or shape derived from the input data.

On the other hand, the activations are represented as an array of JSON objects. These objects provide information pertaining to the activated department, including its ID and remarks that offer insights into the reasons behind the department's activation.

```
{
  "gisResults": [
    {
      "queryResults": [
        {
          "queryId": {query id}
          "queryName": "{query name}",
          "results": {
            "{key1}": [ "{value1}", "{value2}" ]
          }
        }
      ]
    }
  ],
  "activations": [
    {
      "fachstellenId": {id},
      "activationRemark": "{remark}"
    }
}
```

Illustration 16: Output Format Scheme

The illustration below shows an output format example.

```
{
  "gisResults": [
    {
      "queryResults": [{
          "queryId": 72,
          "queryName": "Wildruhegebiet",
          "results": { "wrg": ["Wildruhegebiet"] }
        }]
    }
  ],
  "activations": [
    {
      "fachstellenId": 437,
      "activationRemark": "Wildruhegebiet"
    }
}
```

Illustration 17: Output Format Example

5.4 Techniques for Improved Performance

Two main ways to address the performance issues are implemented. First, parallelizing execution rather than using sequential operations is critical. Secondly, the minimisation of data access helps to avoid wasting time on unnecessary loading of data.

5.4.1 Parallelism

Parallelism refers to technologies that make programs run faster by performing multiple operations simultaneously. This requires hardware with multiple processing units [16].

The Illustration 18 shows how parallelism is built into the process. The process splits the number of queries into multiple threads, which are then processed simultaneously. The threads merge after all queries have been executed and the result is stored. The results are then executed within the rules engine.

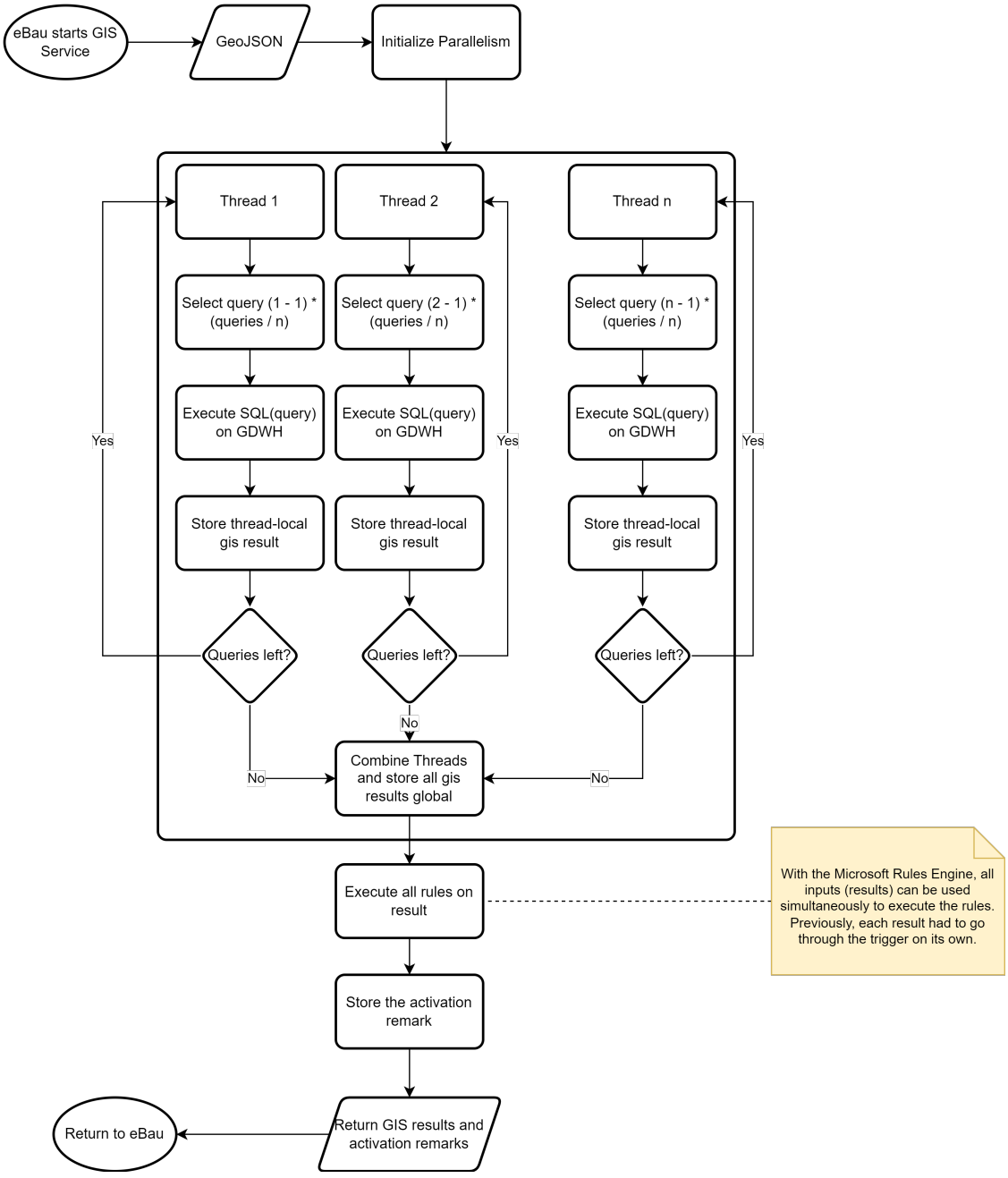

Illustration 18: Parallelism Process as Flow Chart

5.4.2 Reduce External Data Access

Accessing external data is usually the most time-consuming part of an application. Therefore, the goal is to retrieve only the data that is relevant to the application and not to repeatedly request the same data [17].

The service has the potential to optimise data access to SQL queries and rules / delivery notes. Instead of accessing the GIS queries and rules data on each run, they are loaded into the application on start up. As the queries and rules are not changed often, an efficient way is to update them only if something has changed. Currently, a reload is only manually possible.

5.5 Queries Configuration

Besides file-based configuration, the other major requirement is that the query should be maintainable in its own file. For that, the service uses two files per query. The first contains the query itself. The second is a JSON file containing detailed information like the query id or name for the query.

In terms of performance, the question arose as to whether it matters whether multiple small files are read or one large file. In general, one large file leads to fewer operations in terms of opening and closing a file. However, as there are only around 100 – 200 files this does not matter as much.

5.5.1 Structure

The queries are found in the GISQueries folder within the “Data” folder of the eBauGISTriageApi project. Each query has its own folder containing two files: "Query.sql” – containing only the query - and “QueryDetails.json” – containing the query details.

5.5.2 Naming Convention

The Table 8 shows the naming convention for the queries configuration.

Folder/File	Naming convention	Priority
Root folder	GISQueries	Must
Query folder	{id_name}	Should
Query file	Query.sql	Must
Query details file	QueryDetails.json	Must
Query.sql	Valid SQL syntax	Must
QueryDetails.json	Valid JSON with the attributes id, name and queryPostGis	Must

Table 8: Queries Naming Convention

5.6 Rules Configuration

The Microsoft Rules Engine is a freely available software that serves as a replacement for the database trigger utilized in the earlier GIS triage. By enabling to store rules separately from the core logic of the system, it guarantees that modifications to the rules won't impact the essential functioning of the system.

The rules engine provides a direct solution to the issues caused by the trigger. Firstly, unlike the trigger that executes checks for each individual rule, the rules engine executes once with the entire set of results.
Additionally, the rules in the engine follow a consistent template structure, ensuring a standard while allowing flexibility through the use of local or global parameters.
Moreover, the engine enables enhanced extensibility by enabling the execution of code and the ability to check different results within a single rule. This capability was not previously possible and opens the door to more complex and advanced rules in the future.

5.6.1 How Does it Work

The engine functions through workflows and rules. A workflow specifies a set of tasks to be performed and serves as a container for rules. A workflow can include multiple rules arranged in an array. Each rule can consist of an expression, which represents the condition to be validated, and a success/failure event, determining the outcome when the expression evaluates to true/false.
Alternatively, a rule can contain sub-rules. When using sub-rules, an operator is defined to indicate whether all sub-rules must succeed (using the AND operator) or if one success is sufficient (using the OR operator).
This approach enables a flexible means of defining business rules.

5.6.2 What is Supported

It is quite powerful; however, not all functionalities have been incorporated into the application for immediate use. The table below provides an overview of the current supported features when utilizing the engine.

Function	Description
SuccessEvent	This is a static string intended for usage in the activation remark.
Actions.OnSuccess.OutputExpression	This feature enables the output of a C# variable value within the activation remark. It offers the capability to seamlessly combine with a static string, expanding its functionality.
Rule nesting	The rules are processed recursively, allowing for extensive depth in rule hierarchy.

Table 9: Supported Rules Engine Functionalities

5.6.3 Custom functions / ReSettings

Every .NET LambdaExpression is permitted in the expression section of the rule. Nevertheless, in certain cases, additional custom code may be necessary for more complex solutions. This can be accomplished by appending new static functions to the class Helper.Functions.Utils.

An example can be seen in Illustration 19.

```
public static class Utils
{
    /// <summary> Checks if a result is not null.
    0 references
    public static bool ResultExists(JToken value)
    {
        return value != null;
    }
}
```

Illustration 19: Utils Class

Illustration 20 demonstrates the utilization of the function by referring to the class name and function name.

```
"WorkflowName": "305_LZE-NL",
"Rules": [
  {
    "RuleName": "LZE-NL-Landwirtschaftszone",
    "SuccessEvent": "ZPL",
    "RuleExpressionType": "LambdaExpression",
    "Expression": "Utils.ResultExists(lze_nl_landwirtschaftszone.Results.wert)"
  },
```

Illustration 20: Usage of Custom Function

5.6.4 Ignoring the ‘Unknown Identifier’ Exception in Specific Cases

The rules engine has the capability to utilize all GIS results simultaneously, rather than sequentially, enabling the inclusion of more complex rules. However, this also gives rise to an exception known as the 'unknown identifier' exception. This exception occurs when an identifier, specifically a unique identifier for a query result, is missing as input to the engine within the expression of a rule. Since the query results vary based on the location input and many queries do not return a result, this exception is thrown for nearly every service request.

To address this issue, a solution has been implemented that involves checking each rule result for the presence of the phrase "unknown identifier" in the exception message. If this phrase is found, the identifier is verified as a valid query name. If it is a valid query name, the exception is ignored. However, if it is not a valid query name, an exception is thrown as the identifier does not exist as input.

A part of the implementation of this solution is demonstrated in Illustration 21.

```
private bool IsUnknownIdentifierEqualToQueryName(RuleResultTree ruleResultTree, HashSet<string> queryNames)
{
    if (ruleResultTree.ExceptionMessage.Contains("Unknown identifier"))
    {
        string unknownIdentifier = ruleResultTree.ExceptionMessage
            .Substring(ruleResultTree.ExceptionMessage.IndexOf("Unknown identifier"));

        string identifier = unknownIdentifier
            .Substring(unknownIdentifier.IndexOf("'") + 1, unknownIdentifier.LastIndexOf("'") - unknownIdentifier.IndexOf("'") - 1);

        if (queryNames.Contains(identifier))
        {
            return true;
        }
    }

    return false;
}
```

Illustration 21: Function TriageService.IsUnknownIdentifierEqualToQueryName()

5.6.5 Implementation details

Table 10 shows some implementation details of the rules engine, such as naming conventions.

What	Description
Workflow folder	Workflows are stored in the project. Path: **eBauGISTriageApi/Data/Workflows/{...}_Workflow.json**
Workflow file naming	The application will only consider files that end in “Workflow.json”. All others will be ignored. Convention**: {Department ID}_{Department abbrevation}_Workflow.json** Example: **305_LZE-NL_Workflow.json**
Workflow naming	For it to function properly, the department ID must both exist and be separated by an underscore '_'. Failure to meet these requirements will prevent the ID from being effectively used for the activation result. Convention: **{Department ID}_{Department abbrevation}** Example: **305_LZE-NL**
One workflow per department	A workflow is generated for each department, facilitating a comprehensive overview of all the rules and enabling a distinct separation.
Rule naming	The rule name should be exactly the same as the query name defined as "name" in QueryDetails.json. However, some invalid characters (e.g. space ' ', comma ',') must be replaced by an underscore '_'. This then represents the identifier, making it easier to know what the identifier looks like. Convention: **{QueryDetails.json name}** Example: **LZE-NL-ZPS_Landwirtschaftszone**
Identifier naming	The identifier in the expression part represents a query result object. It is derived from the query name in lower case and invalid characters (e.g. space ' ', period '.', backslash '-') are replaced by an underscore '_'. Convention: **{QueryDetails.json name}** Example: **lze_nl_zps_landwirtschaftszone**
Query result	As mentioned above, the identifier represents a query result. The query result consists of a string 'QueryName' and a dictionary 'Results'. The keys of the results are the column names "as is" of the results of the select statements on the GDWH. Example of access to QueryName: **lze_nl_zps_landwirtschaftszone.QueryName == \"LZE-NL-ZPS_Landwirtschaftszone\"** Example of access to Results: **lze_nl_zps_landwirtschaftszone.Results.wert**

Table 10: Rules Engine Implementation Details

6 Quality Assurance

6.1 Testing - Best Practices

This project uses unit and integration tests. They are split into two projects: “eBauGISTriageUnitTests” and “eBauGISTriageIntegrationTests”. This is to ensure the control over which tests are run and that infrastructure components are not accidentally included in unit tests [18].

The tests follow some of Microsoft's best practices [19]. First, “naming your tests”: Method names should follow this convention:
“MethodNameToTest_ScenarioToTestOn_ExpectedBehaviorOnTheGivenResult” [20]. Second, “Arranging your tests”: Each test should use the three AAA's pattern (Arrange, Act, Assert) [19].

6.2 Integration Tests

In this project, quality is assured through unit and integration testing. While the unit tests are only documented within the code, the integration tests have a high level description of the tests for each user story.

The Table 11 describes the tests of agile story 1.

Agile Story 1				
Story Name				
As a BUD-IT developer, I want to test the new web service with a test request, so that I know that the request and response format is correct.				
Sprint Number	1			
Test Nr.	**Variations**	**Expected Result**	**Comments**	**Done**
1	Request uses not supported content-type header.	HTTP code 415		Yes
2	Request sends no body or false formatted JSON in body.	HTTP code 400		Yes
3	Request uses not supported accept header.	HTTP code 406		Yes
4	Request is correct.	HTTP code 200 and correct response		Yes
Any other comments / Questions				
-				

Table 11: Agile Story 1
Template source: [21]

The Table 12 describes the tests of the agile story 2.

Agile Story 2				
Story Name				
As a BUD BIT clerk, I want to have an automatic execution of the gis queries against the gdwh, so that each construction application has their gis results for the end user to see.				
Sprint Number	2			
Test Nr.	**Variations**	**Expected Result**	**Comments**	**Done**
1	Send multiple requests with different input and test for the correct response.	The correct response.		Yes
Any other comments / Questions				
-				

Table 12: Agile Story 2
Template source: [21]

The Table 13 describes the tests of the agile story 3.

Agile Story 3				
Story Name				
As a BUD-IT developer, I want to change the SQL queries without changing the source code, so that I can modify them without a new release.				
Sprint Number	3			
Test Nr.	**Variations**	**Expected Result**	**Comments**	**Done**
1	Send one request with no body to “api/Triage/loadQueries”.	HTTP code 204		Yes
Any other comments / Questions				
-				

Table 13: Agile Story 3
Template source: [21]

6.3 Error Handling

Effective error handling is a critical aspect of software development, ensuring that unexpected problems are handled gracefully, users are provided with meaningful feedback, and system stability is maintained. The GIS triage service uses the try-catch-finally mechanism to ensure this.

Errors are usually dealt with where they occur. In this case, if parts of the request fail and throw an exception, the entire request must fail and the user must be given appropriate feedback. The reason for this is simple: All details are needed for a construction application. This leads to the case where errors are actually handled where they occur, but in most cases this means logging all the information and throwing a new CustomException. This exception is then handled by the controller class. The controller creates a new ProblemDetails object with all the details and sends it back to the user. The user will then see the correct HTTP status code and an appropriate response.

Illustration 22 shows the CustomException class. It is a subclass of Exception and adds the AdditionalInfo property. The info is used to append more information about the exception that has occurred in a way that a business user can understand.

```
public class CustomException : Exception
{
    5 references
    public CustomException(string message, string additionalInfo)
        : base(message)
    {
        AdditionalInfo = additionalInfo;
    }

    4 references | 2/2 passing
    public string AdditionalInfo { get; }
}
```

Illustration 22: CustomException Class

6.4 Logging

Logging plays a vital role in software engineering, enabling effective debugging and troubleshooting. It is a critical part of a production application because, when used correctly, it helps to find errors easily.

The service logs some basic information when a request is started. But the most important information is logged when an error occurs and when an exception is caught.

Illustration 23 shows what is logged when the “DirectoryNotFoundException” is thrown in the “QueriesRepository”.

```
catch (DirectoryNotFoundException ex)
{
    this._logger.LogError("Queries directory not found.");
    this._logger.LogError("Path to queries folder: {pathToFolder} ", this._pathToQueriesFolder);
    this._logger.LogError("Problem directory: {directory}", currDir);
    this._logger.LogError("Error message: {message}", ex.Message);
    throw new CustomException("DirectoryNotFoundException", "Queries directory not found.");
}
```

Illustration 23: GdwhRepository Exception

The logs are written to the “/budlogs/ebaugistriage” folder on the IIS server.

7 Implementation Details

7.1 MVC Pattern

The application follows the Model-View-Controller (MVC) pattern to ensure a clear separation of concerns. The MVC pattern separates the user interface (view), data (model), and application logic (controller) [22]. In this service, the controller is responsible for handling all incoming requests. It immediately validates each request and, if necessary, stops the processing of invalid requests by responding with the appropriate HTTP code. The view in this case refers to the API response in a specific format. However, unlike the traditional use of the MVC pattern, there is no rendering of the view; only the relevant data is returned. The model primarily consists of service classes that process the request, retrieve the required data, and return it to the controller.

Illustration 24 provides a quick overview of how the MVC design pattern is applied in this application.

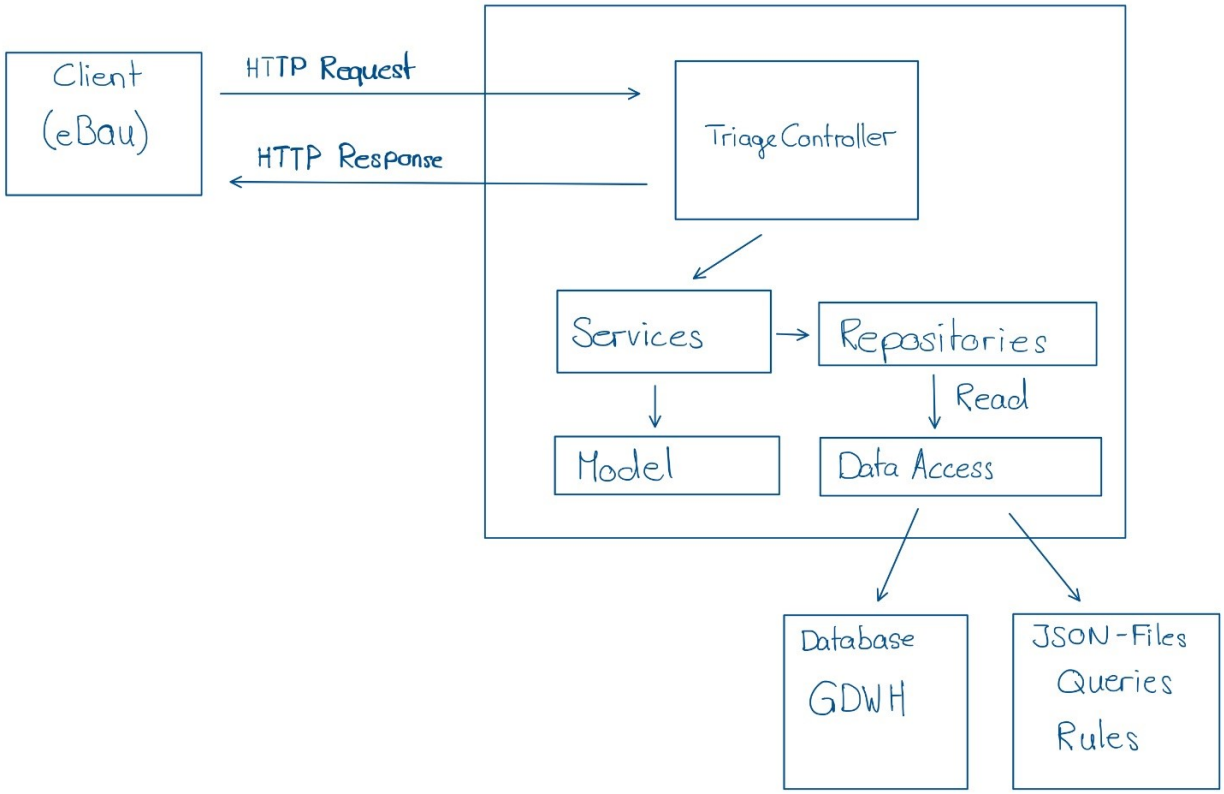

Illustration 24: MVC design pattern
Source: [23]

7.2 Repository Pattern

The repository pattern provides an additional layer of abstraction between the model and the data layer. Basically, any data request is handled by a set of methods provided by the repository. There is no need to worry about the database.
Using this pattern helps to achieve loose coupling and it is easier to test [24].

The diagram in Illustration 25 depicts the implementation of the pattern. Instead of directly accessing the database context, the QueriesService interacts with the IGdwhRepository, which in turn interacts with the GdwhCtx. Similarly, the QueriesService interacts with the IQueriesRepository for query handling. The IWorkflowsRepository follows the same principle. By utilizing interfaces rather than concrete implementations, the data layer becomes easily replaceable.

On the right-hand side, the diagram demonstrates how queries can be unit tested by utilizing mock repositories. This approach eliminates the necessity of establishing a real connection to the queries during testing.

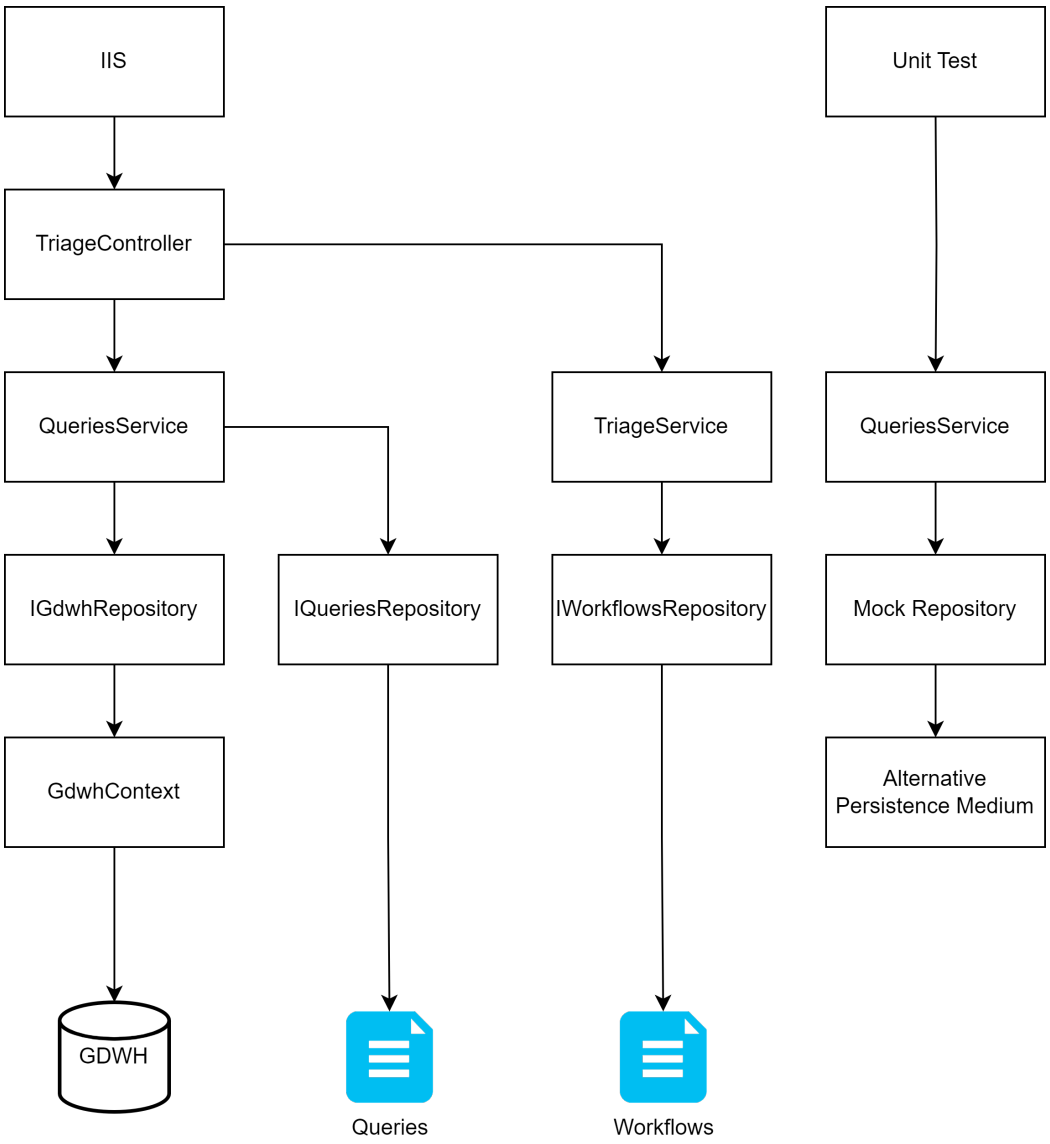

Illustration 25: Repository pattern

8 CI/CD

Continuous software development helps to build, test and deploy iterative changes to the code on an ongoing basis. This approach helps to aim for less or no human intervention from the development of new code to its deployment [25].

The CI/CD environment has three stages: test, pack and deploy.
First, the Test stage has only one job, Test. This job runs all unit and integration tests, and if one test fails, the whole job fails. However, as there were problems with the execution of some tests on the GitLab Runner, the test stage is currently not in use. Second, the "pack" stage contains two jobs: One pack job for each environment (test and prod). These jobs build, configure and store the files on a shared drive. Finally, the deploy stage contains four jobs: A deploy job for each environment and a service deploy job for each environment (service jobs are not used in this case). These jobs are run manually.

Most of the files and commands for the CI/CD environment are generic files created by the BUD-IT department. Only the new "test" stage including jobs was created new for this project.

9 User Documentation

9.1 Queries

9.1.1 Add Query

This is a brief description of how to add a new query.

1. In Visual Studio Code, open the GISQueries folder.
2. Copy an existing folder and its contents.
3. Rename the new folder to {id_name} to conform to the query folder naming convention.
4. Edit the SQL query in Query.sql and the details in QueryDetails.json.
5. After saving, send a post request to /api/Triage/loadQueries. The server will reload the queries.

9.1.2 Update Query

This is a brief description of how to update a query.

1. In Visual Studio Code, open the GISQueries folder.
2. Edit the SQL query in Query.sql and the details in QueryDetails.json in the query folder.
3. After saving, send a post request to /api/Triage/loadQueries. The server will reload the queries.

9.1.3 Delete Query

This is a brief description of how to delete a query.

1. Delete the Query folder and its contents.
2. Send a post request to /api/Triage/loadQueries. The server will reload the queries.

9.2 Rules

9.2.1 Add Department

This is a brief description of how to add a new department.

1. In Visual Studio Code, open the Workflows folder.
2. Copy an existing workflow JSON file.
3. Rename the file according to the naming convention.
4. Edit the workflow name to match the new department and naming convention.
5. Modify the existing rules as required.
6. When you have saved, send a post request to /api/Triage/loadWorkflows. The server will reload the workflows.

9.2.2 Add Rule

This is a brief description of how to add a new department.

1. In Visual Studio Code, open the Workflows folder.
2. Open the JSON file of the department.
3. Copy an existing rule and add it at before the closing tag ‘]’ of the rules.
4. Change the rule as required
5. When you have saved, send a post request to /api/Triage/loadWorkflows. The server will reload the workflows.

10 Project Management

10.1 Scrum

The project followed an agile approach, incorporating selected principles of the Scrum methodology. While it includes most of the Scrum artefacts and ceremonies, some adaptations have been made to meet the specific needs of the project [26]

10.1.1 Scrum Roles

The table below presents the project participants and their respective Scrum roles:

Name	First Name	Role
Schlatter	Timo David	Developer / Scrum Master
Bauer	Dominik	Product Owner

Table 14: Scrum roles
Source: [26]

10.1.2 Time Estimates

There are many different methods for estimating the time required for a user story. This project uses T-shirt sizes: A time range is defined for each size. Instead of estimating a specific amount of time, the user story is estimated as a size. This gives a certain range, but doesn't take too much time to estimate.

The Table 15 shows the different sizes and their time span.

T-Shirt size	Time span
XS	Up to 4 hours
S	Up to 8 hours
M	Up to 3 days
L	Up to 2 weeks

Table 15: T-shirt sizes

10.1.3 Definition of Done

The definition of done defines the minimum requirements for a user story to be called done. This is used to guarantee a solution with good standards.

This is the definition of done for a user story [27]:

- Assumptions of User Story met
- Feature is tested against acceptance criteria
- Unit tests written and passing
- Refactoring completed
- Documentation updated
- Feature ok-ed by Product Owner

This is the definition of done for a sprint [27]:

- All user stories are done
- Project deployed on the test environment identical to production platform

10.2 Roadmap

The Illustration 26 shows a high-level view of the project roadmap. The project consists of three phases. First, the concept phase consists of documenting the is-state, planning and designing the architecture of the optimised GIS-Triage process. Second, the agile phase consists of five two-week sprints to develop the new product. Third, in the closing phase the project finishes.

Illustration 26: Project Roadmap

Sprints

Table 13 shows the course of the sprints, the sprint goals and the duration. The "Appendix G – Initial Sprint Plan" displays the initial sprint plan.

The duration for each sprint was two weeks from Friday to Thursday and the closing sprint is to wrap up the project and complete all deliverables.

Sprint Nr.	Goal	Duration
Sprint 1	The web service is setup and works with test data.	24.03. – 09.04.2023
Sprint 2	The web service returns real GIS results.	10.04. – 28.04.2023
Sprint 3	Queries can be maintained through files	29.04. – 12.05.2023
Sprint 4	The web service returns real, production responses.	13.05. – 03.06.2023
Sprint 5	Translation for column names can be requested. The poster is finished and delivered.	03.06. – 05.06.2023
Closing	Bug fixing, small features, documentation, wrap up. The movie is finished and delivered. The book entry is finished and delivered. The presentation is finished and ready to present.	05.06. – 15.06.2023

Table 16: Course of Sprints

10.3 Concept Phase - Review

The review serves as an action to look at the objectives and their achievement. It is also a way of identifying mistakes and how to improve [28].
In the Appendix A – Concept phase are screenshots of the Gitlab board and milestone at the end of the concept phase.

10.3.1 Goals

The aim of this phase was:

- To create a concept proposal. Based on this proposal, the agile development of the product can be started.

The objective of this phase has been achieved. The concept addresses the key issues for the product: The requirements, the tools and the architecture. Based on this, development can begin.

10.3.2 Unfinished Tasks

The Table 17 shows the open tasks and how they are being handled:

Name	Status	Reason	Handling
Weiteres dokumentieren vom «is-state»	Doing	Task to unspecific	Product backlog
Kapitel «Introduction»	To-do	Not a high priority	Product backlog
Goal	To-Do	Not a high priority	Sprint 1
Übersichtsdiagramm	To-Do	Not a high priority	Product backlog

Table 17: Concept Phase: Unfinished Tasks

10.3.3 Upcoming Sprint

Upcoming tasks transferred to the next sprint:

- Goal

10.4 Sprint 1

10.4.1 Review
In the Appendix B – Sprint 1 are screenshots of the Gitlab board and milestone at the end of sprint 1.

Goals
The aim of this phase was:

- The web service is setup and works with test data.

The objective of this phase has been achieved. The web service handles requests and responses correctly with test data. It returns appropriate HTTP status codes and presents an open API specification as documentation.

Feedback to demo
The feedback from Dominik has been good on the whole. There are only minimal changes to be made to the format of the response.

Unfinished Tasks
The Table 18 shows the open tasks and how they are being handled.

Name	Status	Reason	Handling
Web service [M]	To test	Work took longer than anticipated	Sprint 2

Table 18: Sprint 1 - Unfinished Tasks

General Feedback
The Sprint took longer than expected in terms of days worked. This would not normally happen as Scrum Sprints have a clear end. But as only one person is working on the project, there is no need for a clear end. In this case it made more sense to work longer on the same sprint to complete the task. The reason why it took longer is because of the task "Web Service [M]". There were many things I had to learn about before I could implement it correctly. A complete overview of the time spent can be found in Appendix C – Time Tracking Report “Web Service”.

Technically, the sprint is not "done" in the DoD sense of sprints, as the project is not deployed to the test environment. Automatic deployment by gitlab is not yet implemented. This needs to be done in Sprint 2.

Upcoming Sprint
Based on the review, these are the upcoming tasks for the next sprint:

- Web service [?]
- Continuous integration

10.4.2 Retrospective
The DAKI method is used for the retrospective. It is an acronym and stands for:

- Drop: What do we need to remove?
- Add: What new things should we take on?
- Keep: What should we continue doing?
- Improve: What could be improved?

It helps to identify differences in perceived values across the team and is a great way to gain new insights [29].

DAKI
The Illustration 27 shows the result of the DAKI retrospective.

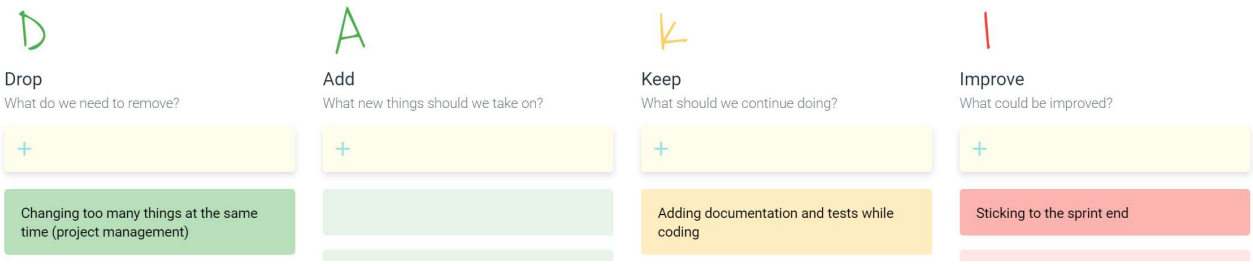

Illustration 27: Sprint 1 DAKI

Actions to Take On
The time estimate should be improved by thinking about knowledge that is not yet available. This should make it easier to stick to the end of the sprint.

10.5 Sprint 2

10.5.1 Review
In the Appendix D – Sprint 2 are screenshots of the Gitlab board and milestone at the end of sprint 2.

Goals
The aim of this phase was:

- The web service now works with the real gis queries. The gis queries can be maintained via a file/files as specified in the documentation.
- The Gitlab CI/CD works and the project can be deployed to the test environment.

The objective of this phase has been partially achieved. The web service returns real data from the GDWH database. However, the GIS queries cannot be maintained via files.
The Gitlab CI/CD is set up and working as expected.

Unfinished Tasks
The Table 19 shows the open tasks and how they are being handled:

Name	Status	Reason	Handling
Maintaining Queries [S]	To-do	The work on the task "Queries against GDWH [S]" took longer than expected.	Sprint 3
Security of API (Authentisierung)	To-do	The task is delayed as it has lower priority than others.	Product backlog

Table 19: Sprint 2 - Unfinished Tasks

General Feedback
As with Sprint 1, this Sprint was overrun. Again, it made more sense to complete the task rather than carry it over to the next sprint.
The reason for the delay was the task "Queries against GDWH [S]". The estimated time was one day. However, it took a total of five days to complete the task. The full time table can be found in Appendix E – Time Tracking Report “Queries against GDWH”. There were several issues with this task:

- Time estimation: Time estimation was poor. As this is very much based on experience, it is unlikely to get much better in future sprints.
- Tasks within the project plan: There are some tasks that need to be done at the end of the project for a minimum viable product. These are shown in chapter Roadmap. In order to work according to the plan, the time estimate for some tasks has been reduced so that they all fit in.
- Finding new subtasks on the way: During the work some new subtasks were found. They were related to the task but not part of the estimate. This led to an overhead.

The result of those errors are the unfinished tasks in Table 19.
The sprint is not finished in the sense of the DoD. Only one out of two user stories was completed and the task was not deployed to the test environment. The CI/CD was set up correctly. However, as the application is now using the GDWH, the tests fail on the server.

Upcoming Sprint
Based on the review, these are the upcoming tasks for the next sprint:

- Maintaining Queries [?]
- Fix the pipeline

10.5.2 Retrospective

The Illustration 28: Sprint 2 DAKI shows the result of the DAKI retrospective.

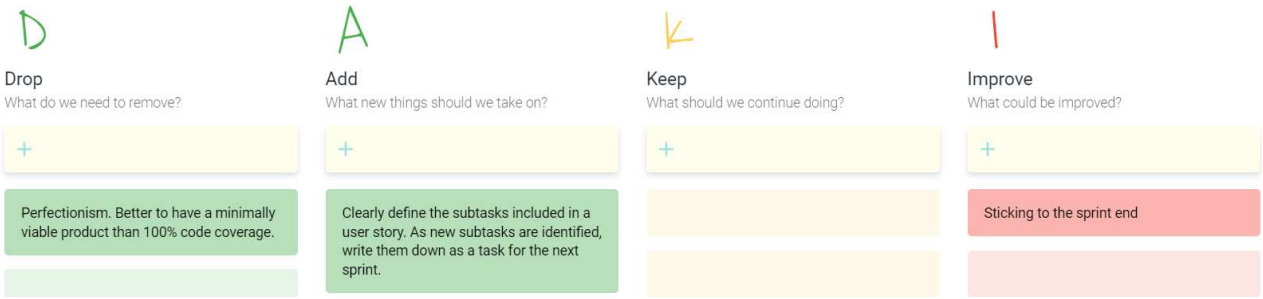

Illustration 28: Sprint 2 DAKI

Actions to Take On

The following actions were decided:

1. Good is better than perfect. Drop the perfectionism.
2. Clearly define subtasks and write upcoming subtasks to the product backlog.

These actions should help to ensure that the Sprint finishes as planned.

10.6 Sprint 3

10.6.1 Review
In the Appendix F – Sprint 3 are screenshots of the Gitlab board and milestone at the end of sprint 3.

Goals
The aim of this phase was:

- The GIS queries can be maintained via a file/files as specified in the documentation.
- The Gitlab CI/CD works with GDWH.

The objectives have been partially achieved. The GIS queries can be maintained through several files. For each query there is a JSON file with detailed information and a SQL file with the actual query. Continuous Integration / Continuous Deployment (CI/CD) still does not work with the Geo Data Warehouse (GDWH). One problem was that there was not enough time to complete all the tasks. Furthermore, this objective is not a high priority. The pipeline generally works. Only the integration and unit tests do not. Since the development work is behind schedule, it makes more sense to postpone this goal and work on the important tasks.

Unfinished Tasks
The Table 20 shows the unfinished tasks and how they are being handled.

Name	Status	Reason	Handling
Maintaining Queries [M]	Doing	The most important work for this task has already been done. It was not closed because it had subtasks defined.	Close
Unit-& Integration-Tests (Subtask Maintaining Queries)	To-do	Insufficient time	Sprint 4
Error-Handling und Logging (Subtask Maintaining Queries)	To-do	Insufficient time	Sprint 4
Optimierungen (Subtask Maintaining Queries)	To-do	Insufficient time	Product backlog

Table 20: Sprint 3 - Unfinished Tasks

General Feedback
The main objective was achieved. However, many tasks have not been completed. For example, the task "Maintain queries [M]" is not completed. The main problem in this case was the subtasks. They have something to do with it, but they can easily be done on their own.
The time estimation was better than before. But the estimation of all tasks together was more than the actual time in the sprint.
The task "Data that support multiple points of publishing [XS]" was more complex than estimated. Therefore it took more time and was the main issue.
Overall, the sprint is almost done in terms of the Definition of Done (DoD).

Upcoming Sprint
Based on the review, these are the upcoming tasks for the next sprint:

- Unit-& Integration-Tests
- Error-Handling und Logging

10.6.2 Retrospective
The Illustration 29 shows the result of the DAKI retrospective.

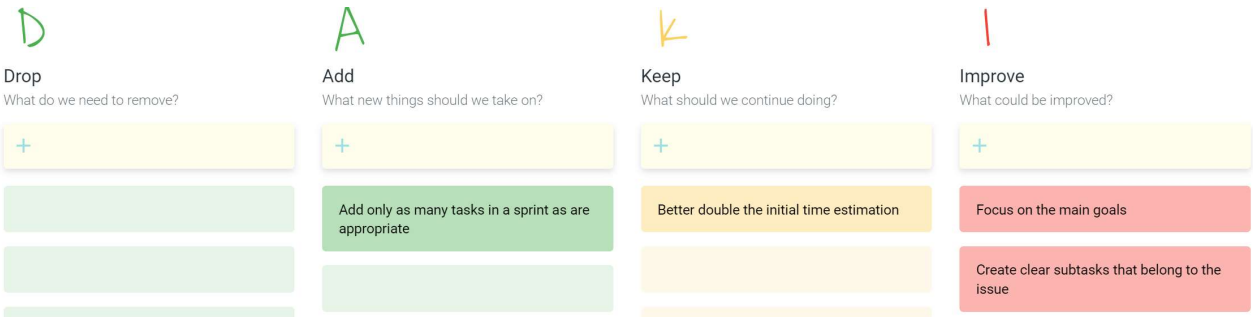

Illustration 29: Sprint 3 DAKI

Actions to Take On
The following actions were decided:

1. Add only as many tasks in a sprint as are appropriate.

10.7 Sprint 4

10.7.1 Review
In the Appendix H – Sprint 4 are screenshots of the Gitlab board and milestone at the end of sprint 3.

Goals
The aim of this phase was:

- The web service returns real, production responses.
- Error handling and testing are up to date and running.

Objective accomplished: The web service successfully delivers production responses with real data, allowing for its implementation into the eBau system. Moreover, notable improvements and updates have been made to the error handling and testing procedures.

General Feedback
The sprint objectives have been successfully accomplished, although the sprint duration had to be extended due to the significant time required for returning real, productive answers, which exceeded initial expectations by more than double. This highlights the challenge of accurately estimating tasks that are unfamiliar or new.

Furthermore, I have recognized the value of refactoring code incrementally, particularly for larger tasks, as it can be resource-intensive. By iteratively refactoring the code, certain sections remain fresh in memory, facilitating a smoother and more efficient refactoring process.

10.7.2 Retrospective
The Illustration 30 shows the result of the DAKI retrospective.

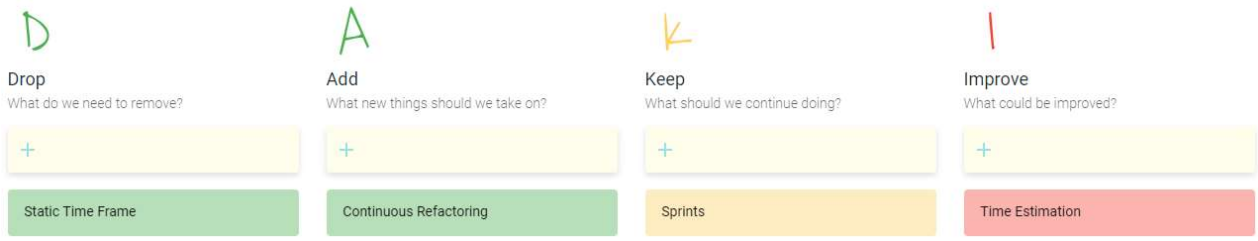

Illustration 30: Sprint 4 DAKI

Actions to Take On
Adhering to a fixed two-week time frame for sprints seems impractical. Due to the inaccuracy of my estimates, tasks often extend beyond the two-week range, necessitating additional time for their completion. Although I have been informally adjusting my approach in this manner, I am now formally adopting this practice and discontinuing the use of two-week sprints. Unlike traditional Scrum, where a static time frame allows stakeholders to observe working software at the end of each sprint, this is not applicable to my situation.

However, I remain committed to implementing sprints as they offer several advantages. They provide a structured framework for working towards specific goals and allow for reflection upon both the outcomes achieved and the process followed.

As highlighted in the previous review chapter, my approach for large tasks involves adopting continuous refactoring. Additionally, I aim to enhance my time estimation skills by multiplying my initial estimates by a factor of three.

10.8 Sprint 5

The duration of Sprint 5 was insufficient to adequately conduct a comprehensive review and retrospective.

Prioritizing Service Enhancements over GIS Query Checker Implementation
The GIS Query Checker was originally intended as an optional feature for the triage process and was meant to be integrated with the new GIS triage service. However, due to time constraints and technical difficulties, its implementation was not possible.

Although it was included in Sprint 5, the duration of Sprint 4 exceeded expectations, leading to a significantly reduced timeframe of just half a week for Sprint 5. Considering my previous experiences, I have realized that my time estimation skills are poor. Therefore, based on this understanding, I would need to triple my initial estimate of two days to arrive at a more "realistic" estimate.

Besides the time limitation, the GIS query checker operates differently from the triage service. While the checker allows users to select one or multiple queries to test the results, each query can have different location parameters. On the other hand, the service executes all queries for a given location parameter. Accommodating this difference would require extensive modifications to both the service and the checker, which is not feasible within the available time.

After careful consideration of possible implementation approaches, I have decided to decline this task and instead focus on enhancing the service and documentation.

11 Future Enhancements and Conclusion

11.1 Examining the Project's Overall Achievements

The project has achieved significant milestones and successfully met all the necessary requirements. First and foremost, all must-have requirements have been met, ensuring that the service fully supports the current features of the triage process. In addition, the queries and rules can be maintained without any changes to the running system, increasing the manageability of the system.

However, several planned nice-to-have features were not implemented. Firstly, adapting the "GIS Query Checker" to the new GIS triage service proved to be challenging. Due to time constraints and technical difficulties, this task could not be completed. The previous sprints took longer than expected, leaving only half a week for the sprint involving the Checker. Moreover, the Checker works differently to the service, requiring a significant amount of time for the necessary adjustments. As a result, it was not possible to implement it properly.
Secondly, the task of developing a web interface for the business rules was not completed due to lack of time. Despite its value, other priorities and time constraints prevented its implementation.
Finally, there were problems running unit and integration tests for continuous integration and continuous delivery (CI/CD). Although it was implemented in the YAML files, the tests involving the GDWH database failed when executed with the Gitlab runner. Interestingly, the tests functioned correctly on local environments. The problem may have been due to the runner's access restrictions for using the database. However, similar to previous challenges, time constraints prevented further investigation into the issue.
Furthermore, the project went beyond the must-have requirements and also met the nice-to-have non-functional requirements. The application was designed as a web service to ensure accessibility and ease of use. In particular, the application has improved performance compared to the previous implementation, delivering faster results.

As a result of these achievements, an improved triage process service was developed and integrated into the BUD IT department's CI/CD (Continuous Integration/Continuous Development) pipeline. The software has undergone quality assurance to ensure its reliability and compliance with established standards.

In summary, the project has been a resounding success. After some testing with real data, the triage process is ready for seamless integration and rapid deployment. This successful outcome provides the BUD IT department with the opportunity to benefit from an improved service that meets their needs and objectives.

11.2 Towards Optimizing the Triage Process Further

The project offers exciting opportunities for improvement and extension in a number of areas. One significant enhancement would be the implementation of a web interface to the rules engine. This interface would enable business users to maintain rules independently, reducing their reliance on IT for rules management.

In terms of performance, there has already been a notable improvement of 100%. However, there is still potential for further optimisation, particularly in construction applications involving large numbers of shapes and points. The current approach to parallelization focuses on parallelizing queries per shape/point, but has scalability limitations. For example, processing six times the number of points takes six times as long as processing a single point. Improving scalability and speed in such scenarios would be highly beneficial.

The capabilities of the rules engine have been greatly extended compared to the previous after-insert trigger implementation. It now allows the creation of rules that test multiple query results, providing better evaluation. In addition, the rules engine allows a greater number of parameters to be included in the query. This newfound flexibility allows the creation of complex rules that were not previously possible.

By addressing these areas of improvement, the project can achieve greater user empowerment, improved system performance, and enhanced functionality. These extensions will align the system more closely with the desired objectives.

11.3 Lessons Learned

Throughout my bachelor thesis, I had many learning experiences. In this chapter, I will discuss the main challenges I faced and the valuable lessons I learned from them.

One of the biggest hurdles I faced was working within the BUD-IT department. I would like to clarify that it was not a negative working environment, but rather an unfamiliar one. Working in such an environment was more challenging than working in a green field environment. In order to develop my software to match the functionalities of the existing GIS triage, I needed a full understanding of its current implementation. This proved difficult due to limited documentation and the unavailability of the original Microsoft Access source code to me. But, I was familiar with the "GIS Query Checker" from Project 2, which was used as a test tool for the triage. To overcome this obstacle, I did some manual testing with the Checker and got answers to my questions through open communication with Dominik, my industry partner.
Effective communication with all stakeholders is key to success. To facilitate this, I organised weekly scrum meetings where I would present my progress to Dominik and seek clarification on any questions I had. This became an excellent communication platform. While this approach worked well, there were times when I needed additional support or had urgent questions. Currently, I would wait until our scheduled meeting, which sometimes resulted in delays of almost a week. In hindsight, I believe it would have been more beneficial to address my questions immediately. I refrained from doing so, assuming that Dominik already had a heavy workload. In the future, I would definitely ask questions promptly, as this would greatly improve overall efficiency and therefore benefit everyone involved.

Using aspects of the Scrum methodology for project management proved effective. By defining clear tasks for each sprint, I was able to focus my efforts on achieving specific goals. This approach ensured that I always had a working product at the end of each sprint that I could demonstrate to Dominik for additional insight. Although this method worked well, there were aspects that I didn't fully appreciate. For example, the fixed length sprints did not fit my project schedule. None of the sprints was completed within the planned two-week timeframe, mainly due to inaccurate time estimates. In addition, strict adherence to the two-week timeframe was not critical as no stakeholder was eagerly awaiting my working product. To improve this aspect, I would reintroduce fixed sprint durations to provide a good time management structure, but allow flexibility to complete tasks without feeling overwhelmed.
The difference from now is that I would plan to work this way rather than feel guilty about it.

Overall, working on this project was an enjoyable experience that provided valuable lessons. In particular, tackling a new project in an unfamiliar environment was very educational. It provided me with practical experience in dealing with the challenges of such environments, which are common in the industry, as opposed to working in a green field environment.

12 List of Illustrations

13 Contents of the Table

14 Glossary

GIS
Geographic Information System 6

GeoJSON
"GeoJSON is an open standard geospatial data interchange format that represents simple geographic features and their nonspatial attributes. Based on JavaScript Object Notation (JSON), GeoJSON is a format for encoding a variety of geographic data structures." [30] 7

GDWH
Geo Data WareHouse 7

BIT
BIT stands for «Bauinspektorat», which is the department responsible for construction applications. 8

RPC
Remote Procedure Call 16

API
Application Programming Interface 19

REST
Representational State Transfer 19

CI/CD
Continuous Integration and Continuous Delivery/Deployment 40

15 Bibliography

[1] M. P. Timo Schlatter, “Bachelor Thesis - Task definition,” 2023.

[2] “Oracle,” [Online]. Available: https://www.oracle.com/ch-de/database/what-is-a-data-warehouse/. [Accessed 12 June 2023].

[3] «javatpoint,» [Online]. Available: https://www.javatpoint.com/mysql-after-insert-trigger. [Zugriff am 12 June 2023].

[4] Embitel, "Embitel," [Online]. Available: https://www.embitel.com/blog/ecommerce-blog/how-microservices-communicate-with-each-other. [Accessed 9 March 2023].

[5] Servicestack, "MQ Client Architecture," [Online]. Available: https://docs.servicestack.net/messaging#mq-client-architecture. [Accessed 9 March 2023].

[6] L. P. Bruce Davie, “Computer Networks: A Systems Approach,” 29 March 2021. [Online]. Available: https://book.systemsapproach.org/e2e/rpc.html. [Accessed 11 March 2023].

[7] D. Boike, «Particular Software,» 21 September 2021. [Online]. Available: https://particular.net/blog/rpc-vs-messaging-which-is-faster. [Zugriff am 11 March 2023].

[8] A. Pandey, “Quora,” [Online]. Available: https://www.quora.com/What-is-a-scripting-engine. [Accessed 19 March 2023].

[9] M. NOWAK, “Hyperon,” 5 September 2022. [Online]. Available: https://www.hyperon.io/blog/what-is-a-rules-engine. [Accessed 19 March 2023].

[10] «libhunt,» [Online]. Available: https://www.libhunt.com/compare-NRules-vs-RulesEngine. [Zugriff am 23 March 2023].

[11] farida-nullwala, “Github,” 11 December 2021. [Online]. Available: https://github.com/microsoft/RulesEngine/wiki. [Accessed 23 March 2023].

[12] S. Nikolayev, “Github,” 8 April 2021. [Online]. Available: https://github.com/NRules/NRules/wiki. [Accessed 23 March 2023].

[13] K. Abuhakmeh, “Blog.Jetbrains,” Jetbrains, 19 July 2021. [Online]. Available: https://blog.jetbrains.com/dotnet/2021/07/19/getting-started-with-asp-net-core-and-grpc/. [Accessed 26 March 2023].

[14] J. Newton-King, Microsoft, 9 September 2022. [Online]. Available: https://learn.microsoft.com/en-us/aspnet/core/grpc/comparison?view=aspnetcore-7.0. [Accessed 26 March 2023].

[15] “noname,” [Online]. Available: https://nonamesecurity.com/learn-what-is-json-rpc. [Accessed 26 March 2023].

[16] “Haskell,” [Online]. Available: https://wiki.haskell.org/Parallelism_vs._Concurrency. [Accessed 20 March 2023].

[17] J. Kanjilal, “Codeguru,” 25 August 2022. [Online]. Available: https://www.codeguru.com/dotnet/asp-net-performance/. [Accessed 22 March 2023].

[18] J. v. d. Til, “learn.microsoft,” Microsoft, 17 March 2023. [Online]. Available: https://learn.microsoft.com/en-us/aspnet/core/test/integration-tests?view=aspnetcore-6.0. [Accessed 9 April 2023].

[19] “Microsoft,” 11 April 2022. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/core/testing/unit-testing-best-practices. [Accessed 15 June 2023].

[20] “learn.microsoft,” Microsoft, 11 April 2022. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/core/testing/unit-testing-best-practices. [Accessed 9 April 2023].

[21] Narayanamurthi, «agileseeds,» 2 June 2017. [Online]. Available: https://agileseeds.com/agile-test-plan-template-samples-best-practices/. [Zugriff am 3 April 2023].

[22] “dotnet.microsoft,” Microsoft, [Online]. Available: https://dotnet.microsoft.com/en-us/apps/aspnet/mvc. [Accessed 9 April 2023].

[23] “Github,” [Online]. Available: https://github.com/Pufke/.NET-Core-3.1-MVC-REST-API. [Accessed 13 June 2023].

[24] Ardalis, “deviq,” 21 July 2023. [Online]. Available: https://deviq.com/design-patterns/repository-pattern. [Accessed 28 April 2023].

[25] M. Amirault, “docs.gitlab.com,” Gitlab, 24 February 2023. [Online]. Available: https://docs.gitlab.com/ee/ci/introduction/index.html. [Accessed 14 April 2023].

[26] T. Schlatter, «GIS Query Checker,» 2023.

[27] “Apiumhub,” [Online]. Available: https://apiumhub.com/tech-blog-barcelona/definition-of-done-examples-software-projects/. [Accessed 26 March 2023].

[28] “Aha!,” [Online]. Available: https://www.aha.io/roadmapping/guide/agile/sprint-review. [Accessed 25 March 2023].

[29] “teamretro.com,” [Online]. Available: https://www.teamretro.com/retrospectives/daki-retrospective. [Accessed 10 April 2023].

[30] ArcGIS Enterprise, "ArcGIS Enterprise," [Online]. Available: https://enterprise.arcgis.com/en/portal/latest/use/geojson.htm. [Accessed 4 March 2023].

[31] R. D. Hernandez, “freecodecamp,” freeCodeCamp, 19 April 2021. [Online]. Available: https://www.freecodecamp.org/news/the-model-view-controller-pattern-mvc-architecture-and-frameworks-explained/. [Accessed 9 April 2023].

16 Appendix

16.1 Appendix A – Concept phase

The Illustration 31 shows detailed information about the concept phase: The goal, duration and the tracked time.

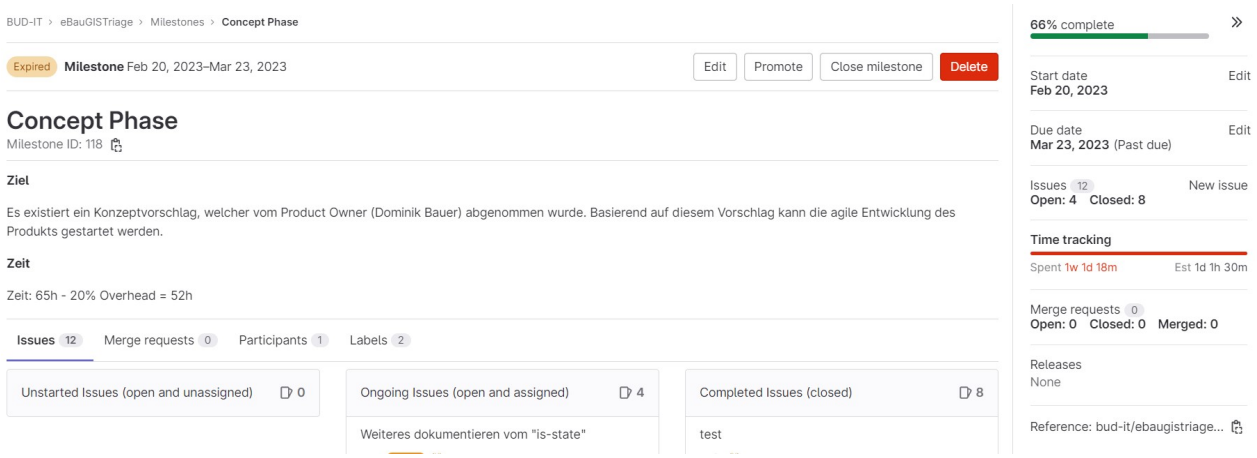

Illustration 31: Concept Phase – Gitlab Milestone

The Illustration 32 shows the Gitlab board at the end of the concept phase.

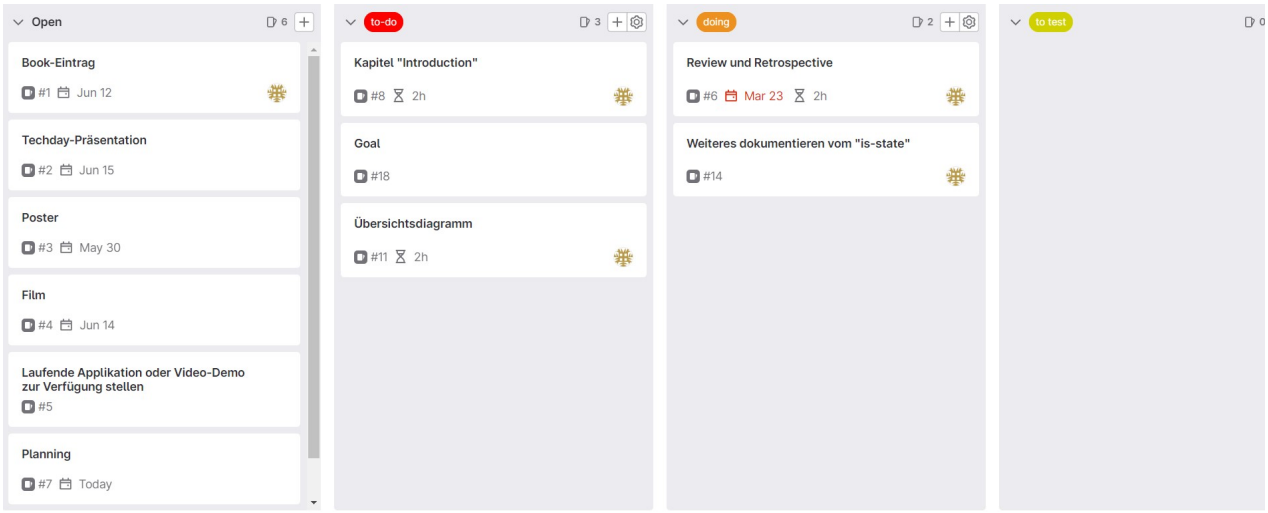

Illustration 32: Concept Phase - Kanban Board

16.2 Appendix B – Sprint 1

The Illustration 33 shows detailed information about Sprint 1: The goal, duration and the tracked time.

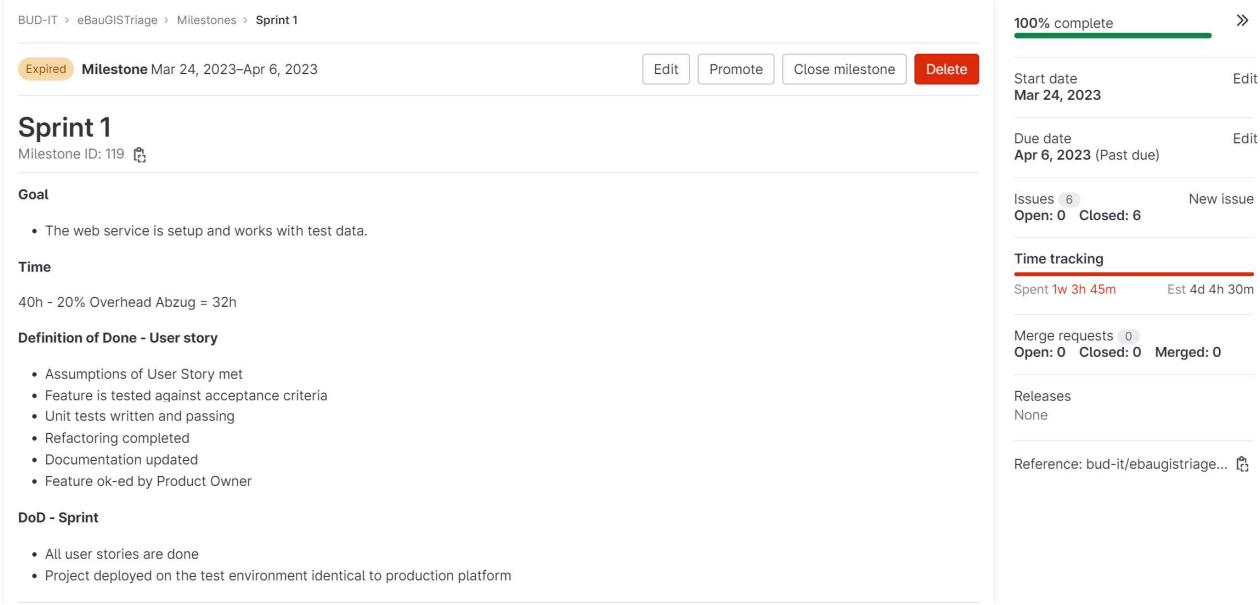

Illustration 33: Sprint 1 - Gitlab Milestone

The Illustration 34 shows the Gitlab board at the end of Sprint 1.

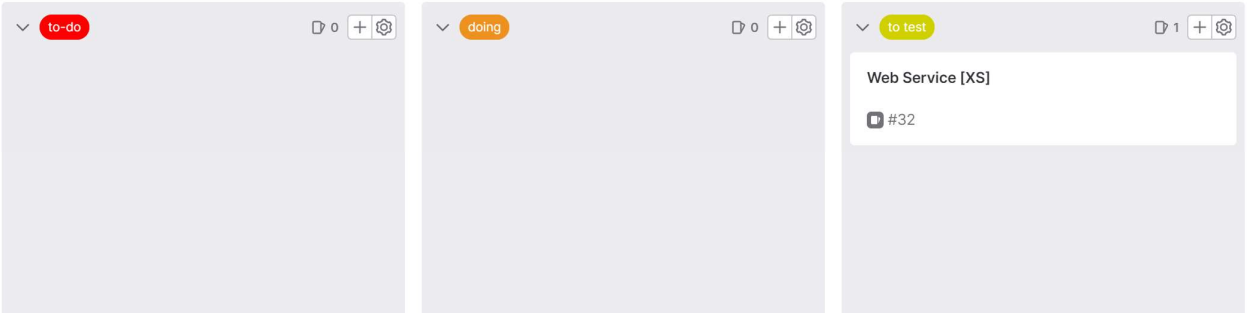

Illustration 34: Sprint 1 - Kanban Board

16.3 Appendix C – Time Tracking Report "Web Service"

The Illustration 35 shows the time tracking report of the task "Web Service":

Time tracking report

Spent at	User	Time spent	Summary / note
March 26, 2023, 16:09 (UTC: +0200)	Timo Schlatter	2h	Entscheidung zwischen gRPC und JSON-RPC + Dokumentation Nächster SChritt: Start mit JSON-RPC Server bzw. Tutorial
March 27, 2023, 20:58 (UTC: +0200)	Timo Schlatter	1h	Mit dem Tutorial gestartet. Allerdings Probleme beim Part mit dem JsonRpc da ich es nicht mit Nuget installieren kann: " [nuget.org] Unable to load the service index for source https://api.nuget.org/v3/index.json. Schlüssel ist im angegebenen Status nicht gültig."
March 28, 2023, 21:44 (UTC: +0200)	Timo Schlatter	2h	Nicht mehr direkt am Tutorial weitergemacht. Warum soll ich wie gemäss Tutorial ein Websocket verwenden, wenn ich den JSON Request auch einfach über HTTP Post senden kann? Dann kann ich den Endpunkt wie bereits bekannt analog REST API aufbauen. Allerdings habe ich diese Lösung noch nicht funktionsbereit
March 30, 2023, 10:09 (UTC: +0200)	Timo Schlatter	2h	Vorbereiten, Meeting mit Dominik, Nachbereiten
March 30, 2023, 16:57 (UTC: +0200)	Timo Schlatter	6h	Gestartet mit dem Postrequest. Allerdings noch Probleme mit der Validierung vom Input. Nächster Schritt: Nochmals genau überlegen, wie die Daten daherkommen müssen, wie das validiert werden kann. Dann serialisieren und deserialisieren und whatever.
April 1, 2023, 12:06 (UTC: +0200)	Timo Schlatter	2h	API + Daten mappen und checken hinzugefügt
April 1, 2023, 18:50 (UTC: +0200)	Timo Schlatter	2h	a67580eb, b74fe76b und mich über Unit Tests in .net informiert.
April 2, 2023, 11:54 (UTC: +0200)	Timo Schlatter	2h	Mich über unit testing in .net informiert. Projekt und ersten Test hinzugefügt, siehe commit 125bb856
April 2, 2023, 21:39 (UTC: +0200)	Timo Schlatter	2h	Mich über INtegrationstests informiert und begonnen, diese zu erstellen, siehe commit e3889882
April 3, 2023, 21:24 (UTC: +0200)	Timo Schlatter	2h	Habe mich informiert, wie ich meine Tests dokumentieren soll + neuen Test hinzugefügt siehe commit 3ebf2f0b
April 5, 2023, 22:01 (UTC: +0200)	Timo Schlatter	2h	Alle bisher geplanten integrations tests hinzugefügt, siehe commit aff3ee12
April 5, 2023, 23:10 (UTC: +0200)	Timo Schlatter	1h 15m	Unit tests für queriesserivce hinzugefügt. Siehe commit 3b4f1f14
April 7, 2023, 12:40 (UTC: +0200)	Timo Schlatter	2h	Tests abgeschlossen (siehe commit c7c1a11f), Security muss ich mit Dominik besprechen. Mit Swagger Dokumentation angefangen.
April 7, 2023, 16:50 (UTC: +0200)	Timo Schlatter	3h	Swagger doc abgeschlossen siehe commit a98b893f
April 8, 2023, 12:32 (UTC: +0200)	Timo Schlatter	2h	Refactoring, siehe commits c39dd142 und ba49844a
April 9, 2023, 13:41 (UTC: +0200)	Timo Schlatter	15m	Refactoring abgeschlossen, siehe commit e94e550d
April 9, 2023, 16:58 (UTC: +0200)	Timo Schlatter	2h 45m	Dokumentation aktualisiert (source code doc: mvc, testing und readme.md). Muss noch das klassendiagramm aktualisieren
April 10, 2023, 09:43 (UTC: +0200)	Timo Schlatter	1h 30m	Klassendiagramm und API in der Dokumentation aktualisiert
		4d 5h 45m	

Illustration 35: Time Tracking Report: Task "Web Service"

16.4 Appendix D – Sprint 2

The Illustration 36 shows detailed information about Sprint 2: The goal, duration and the tracked time.

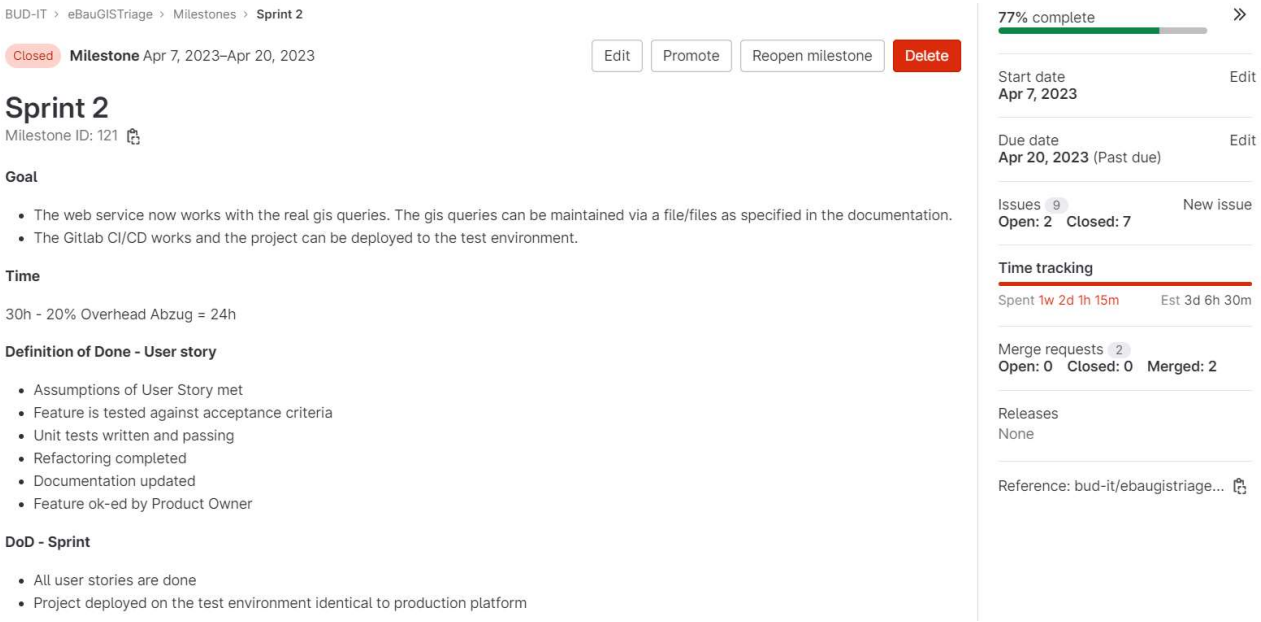

Illustration 36: Sprint 2 - Gitlab Milestone

The Illustration 37 shows the Gitlab board at the end of sprint 2.

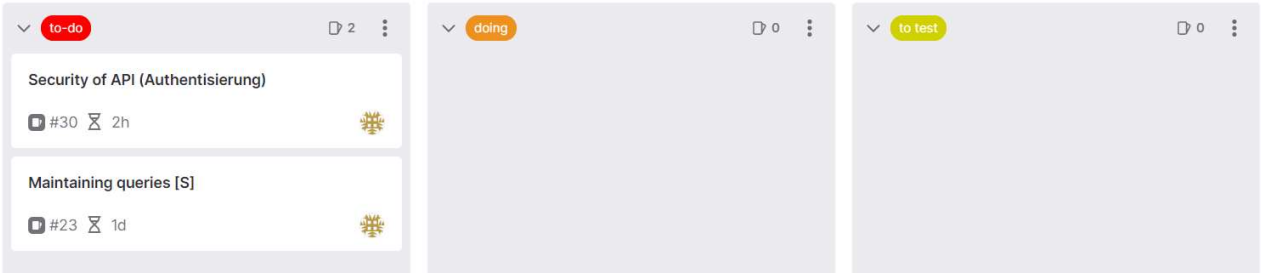

Illustration 37: Sprint 2 - Kanban Board

16.5 Appendix E – Time Tracking Report "Queries against GDWH"

The Illustration 38 shows the time tracking report of the task "Queries against GDWH":

April 15, 2023, 11:08 (UTC: +0200)	Timo Schlatter	1h	Gdwh context und DI hinzugefügt, siehe commit 22256bd1	
April 15, 2023, 18:45 (UTC: +0200)	Timo Schlatter	2h	ExecuteQueries, siehe commit e5d286a7	
April 16, 2023, 18:15 (UTC: +0200)	Timo Schlatter	4h	Grundlegende Funktionalität gegeben. Muss das noch genauer testen und entsprechende Integration/unit-tests schreiben.	
April 17, 2023, 21:30 (UTC: +0200)	Timo Schlatter	2h	Am Integrationstest für Agile Story 2.1 gearbeitet. Mit verschiedenen Fehler bei der Json Deserialisierung herumgekämpft. Jetzt am Schlussspurt, nur noch die Assertion schlägt fehl	
April 20, 2023, 10:12 (UTC: +0200)	Timo Schlatter	45m	Besprechung Dominik	
April 20, 2023, 10:12 (UTC: +0200)	Timo Schlatter	1h 15m	Test Post_MultipleInput_ReturnExpectedGisResults funktioniert jetzt	
April 20, 2023, 21:15 (UTC: +0200)	Timo Schlatter	2h		
April 22, 2023, 12:04 (UTC: +0200)	Timo Schlatter	3h	Mich über parallelität in .NET informiert und mit ParallelLForEach() gestartet. Frage mich aber wie ich den DB context per partition und nicht per query erstellen kann.	
April 22, 2023, 17:01 (UTC: +0200)	Timo Schlatter	2h	Multithreading eingebaut. Jetzt geht es noch darum, die Zeit von singlethreade <> multithreaded zu messen um zu sehen ob ich wirklich einen Vorteil erhalten habe. Danach refactoring	
April 22, 2023, 21:52 (UTC: +0200)	Timo Schlatter	1h	Unit Test für Zeitmessung erstellt, gibt noch fehler	
April 23, 2023, 21:00 (UTC: +0200)	Timo Schlatter	1h 30m	Mit dem TEst weitergemacht	
April 24, 2023, 21:49 (UTC: +0200)	Timo Schlatter	2h	Tests gefixt und mit dem Refactoring begonnen.	
April 26, 2023, 21:45 (UTC: +0200)	Timo Schlatter	2h	Middleware um Problemdetails für 500 errors hinzuzufügen. Muss noch den Test fertigstellen für database not reachable. Danach noch logging hinzufügen.	
April 27, 2023, 12:18 (UTC: +0200)	Timo Schlatter	1h	Besprechung Dominik	
April 27, 2023, 12:19 (UTC: +0200)	Timo Schlatter	3h	Herumkämpfen mit dem DB Test. Wird eher ein Unit Test mit Mock DB falls möglich	
April 27, 2023, 16:59 (UTC: +0200)	Timo Schlatter	4h	Angefangen das Repository pattern zu verwenden um einfache unit tests für die Datenbank schreiben zu können. Fast fertig, muss noch den Schluss im parallelLforeach fertig machen. Danach testen und Test schreiben	
April 28, 2023, 09:50 (UTC: +0200)	Timo Schlatter	2h	Repository pattern umgesetzt und angefangen mit refactoring	
April 28, 2023, 12:02 (UTC: +0200)	Timo Schlatter	2h	Refactoring, tests gefixt	
April 28, 2023, 16:17 (UTC: +0200)	Timo Schlatter	2h 30m	Dokumentieren	
		1w		

Illustration 38: Time Tracking Report: Task "Queries against GDWH"

16.6 Appendix F – Sprint 3

The Illustration 39 shows detailed information about Sprint 3: The goal, duration and the tracked time.

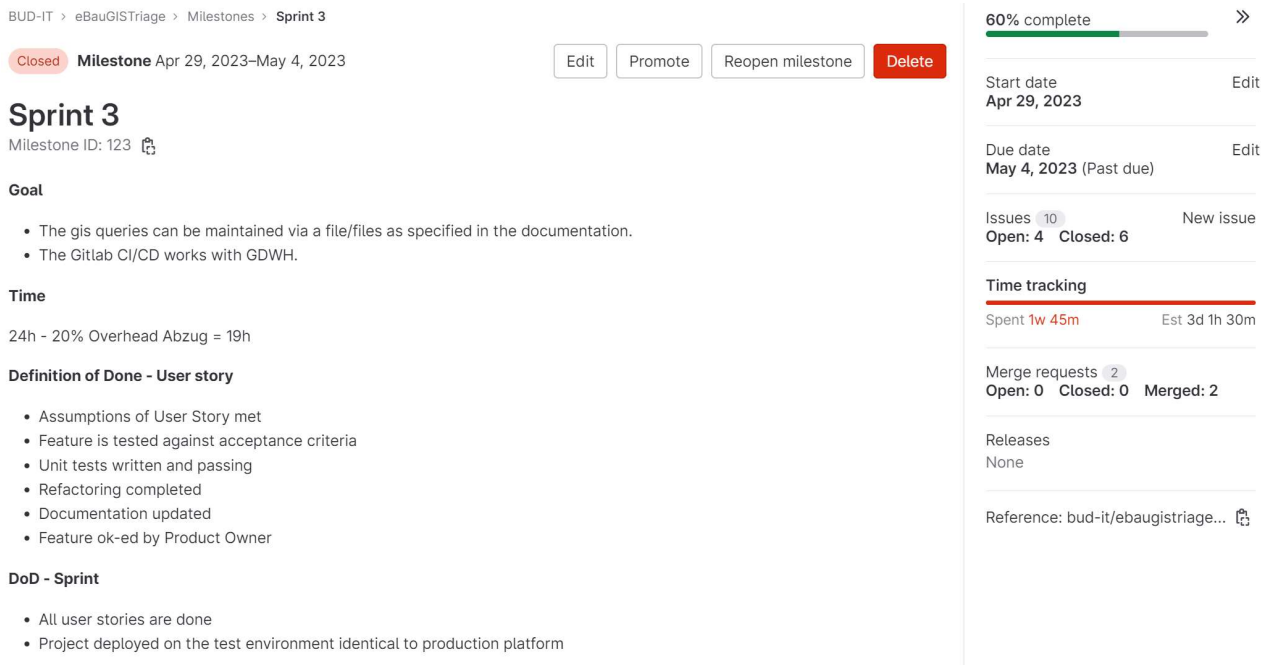

Illustration 39: Sprint 3 - Gitlab Milestone

The Illustration 40 shows the Gitlab board at the end of sprint 3.

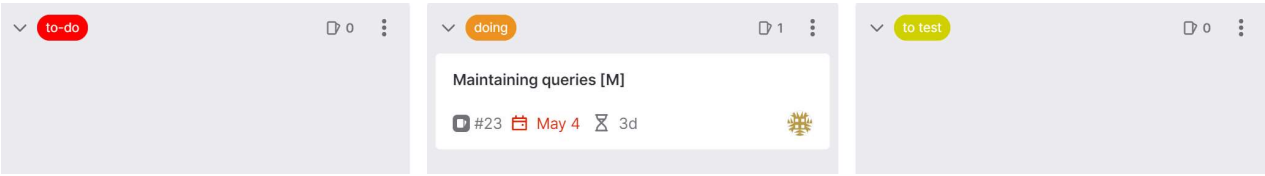

Illustration 40: Sprint 3 - Kanban Board

The Illustration 41 shows the subtasks of “Maintaining queries [M]”.

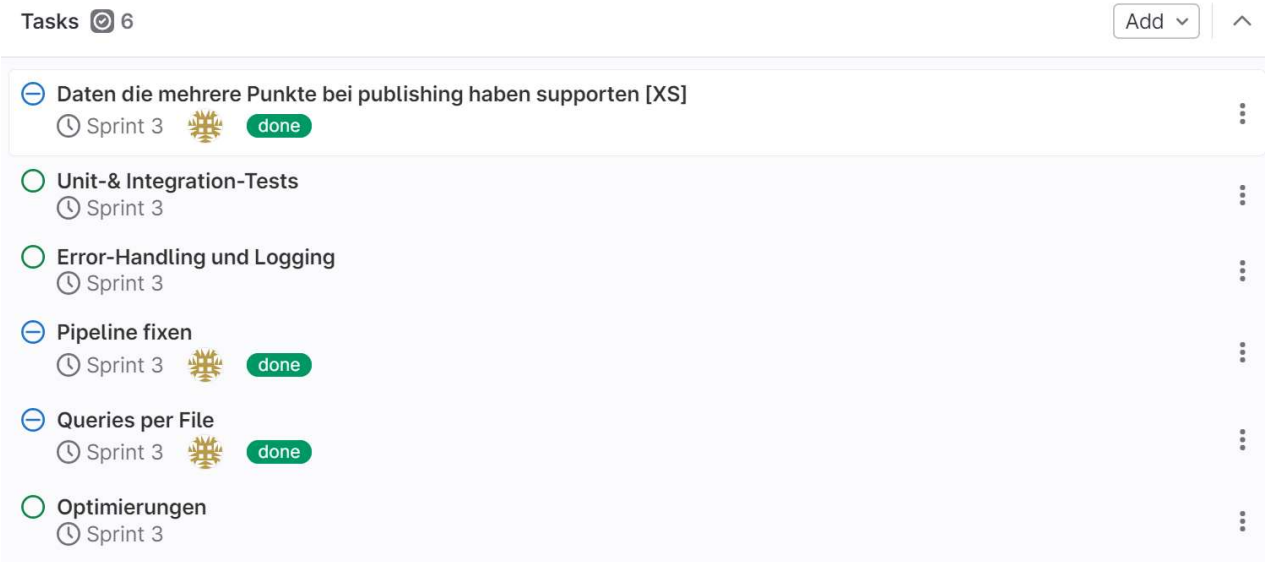

Illustration 41: Subtasks of the Task "Maintaining queries"

16.7 Appendix G – Initial Sprint Plan

Table 21 shows the initial sprint plan.

Sprint Nr.	Goal	Duration (Friday – Thursday)
Sprint 1	The web service is setup and works with test data.	24.03. – 06.04.2023
Sprint 2	The web service returns real gis results.	07.04. – 20.04.2023
Sprint 3	The web service returns real, production responses.	21.04. – 04.05.2023
Sprint 4	The gis query checker works with the web service.	05.05. – 18.05.2023
Sprint 5	Business user can maintain the rules through a web interface. The poster is finished and delivered.	19.05. – 01.06.2023
Closing	Bug fixing, small features, documentation, wrap up. The movie is finished and delivered. The book entry is finished and delivered. The presentation is finished and ready to present.	02.06. – 15.06.2023

Table 21: Initial Sprint Plan

16.8 Appendix H – Sprint 4

The following illustration shows detailed information about Sprint 4: The goal, duration and the tracked time.

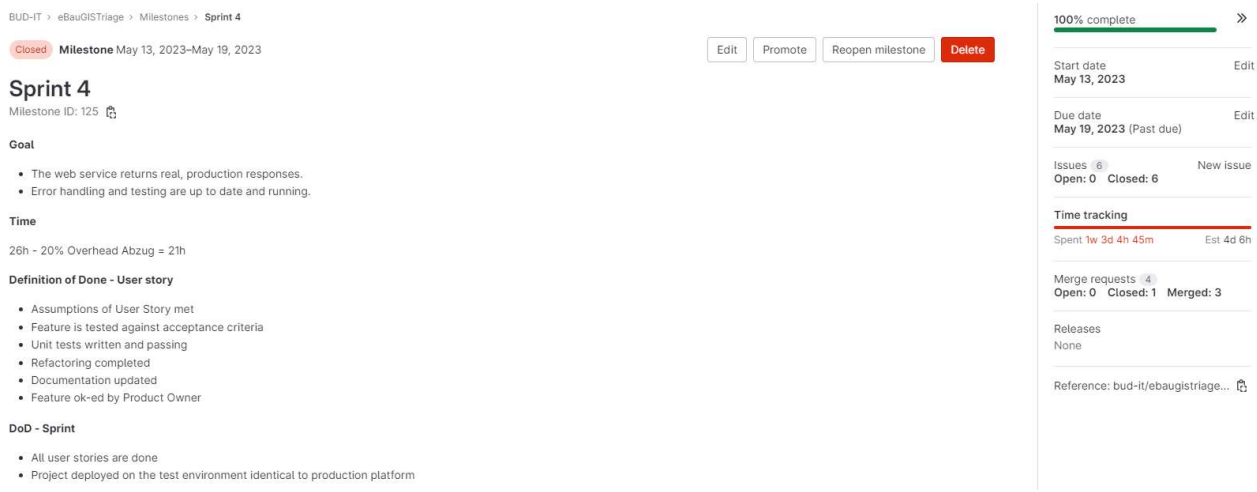

Illustration 42: Sprint 4 - Gitlab Milestone

The Illustration 43 shows the Gitlab board at the end of sprint 4.

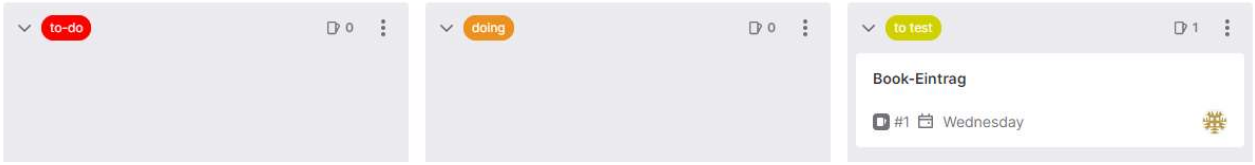

Illustration 43: Sprint 4 - Kanban Board

Erklärung der Diplomandinnen und Diplomanden
Déclaration des diplômant-e-s

Selbständige Arbeit / *Travail autonome*

Ich bestätige mit meiner Unterschrift, dass ich meine vorliegende Bachelor-Thesis selbständig durchgeführt habe. Alle Informationsquellen (Fachliteratur, Besprechungen mit Fachleuten, usw.) und anderen Hilfsmittel, die wesentlich zu meiner Arbeit beigetragen haben, sind in meinem Arbeitsbericht im Anhang vollständig aufgeführt. Sämtliche Inhalte, die nicht von mir stammen, sind mit dem genauen Hinweis auf ihre Quelle gekennzeichnet.

Par ma signature, je confirme avoir effectué ma présente thèse de bachelor de manière autonome. Toutes les sources d'information (littérature spécialisée, discussions avec spécialistes etc.) et autres ressources qui m'ont fortement aidé-e dans mon travail sont intégralement mentionnées dans l'annexe de ma thèse. Tous les contenus non rédigés par mes soins sont dûment référencés avec indication précise de leur provenance.

Name/*Nom*, Vorname/*Prénom* Schlatter, Timo

Datum/*Date* 15.06.2023

Unterschrift/*Signature* T. Schlatter

Dieses Formular ist dem Bericht zur Bachelor-Thesis beizulegen.
Ce formulaire doit être joint au rapport de la thèse de bachelor.