Santeri Pulkkinen

DEVELOPMENT OF AN ERROR LOG MODULE

Thesis
CENTRIA UNIVERSITY OF APPLIED SCIENCES
Bachelor of Engineering, Information Technology
December 2023

ABSTRACT

Centria University of Applied Sciences	**Date** 17.12 2023	**Author** Santeri Pulkkinen
Degree programme Bachelor of Engineering, Information Technology		
Name of thesis Development of an error log module		
Centria supervisor Henry Paananen		**Pages** 27
Instructor representing commissioning institution or company Tom Cygnel		

The purpose of this thesis was to implement an error log module for Raisoft.net SaaS platform. This task was commissioned by Vitec Raisoft oy. The implementation had to be modular so it can be integrated with the main software Vitec Raisoft Oy provides to their customers. The task was divided in two parts, backend, and frontend. The backend work focused on fetching the desired data from database, as frontend work consisted of building a way to display error log entries in the software. The aim of the project was to connect backend and frontend implementations to display error log entries in readable format in the main software.

Information logging in larger systems is important part of improving user experience. By logging information such as errors that has occurred in the system makes identifying underlying issues of the system easier. Administrators can use this information to detect and possibly prevent issues from happening in the future. Error log is used to collect errors occurred in the system to spot system vulnerabilities. This thesis aims to visualize errors occurred in the system for the administrators of the software Vitec Raisoft Oy provides. The implemented error log allows administrators to view and inspect error log entries collected in a table format. Administrators can filter error logs based on time interval or username to narrow down search. Error log table can be exported as Excel, PDF, or CSV format. Tools for this task were provided by Vitec Raisoft Oy, including Delphi and Visual Studio Code as integrated development environments and SQL Server Management Studio for relational database management. Common web programming languages such as PHP and JavaScript were used in the development for user interface.

The project work was divided in two parts. The work started with the implementation of the backend functions to fetch desired data from the database table. Once the backend part was implemented, the development process for the frontend part was able to begin. The frontend part implementation focused on developing user interface, required functionalities and connection with the backend. The frontend implementation had to stylistically align with rest of the software. After the work had been completed, test cases were implemented, and feature was partly tested. Due to time limitations frontend part is still to be tested, however the development part for the project was completed successfully.

Key words
Delphi, error log, information logging, JavaScript, PHP, programming, SQL

CONCEPT DEFINITIONS

Delphi

Delphi is a programming language and an IDE based on Object Pascal programming language.

HTML

Hypertext Markup Language, used to define structure of a web page.

ICT

Information and Communications Technology

IDE

Integrated Development Environment

MIT

MIT is an open-source software license.

MPL

Mozilla Public License is an open-source software license.

OOP

Object-oriented programming (OOP) is a programming paradigm.

PHP

General purpose scripting language to write dynamic web pages.

QA

Quality assurance

SaaS

Software as a Service (SaaS), application accessed by user via internet.

SQL

Structured Query Language (SQL) is used to access and manipulate data in relational databases.

SSMS

SQL Server Management Studio

UI

User interface

XML

Extensible Markup Language (XML) is used for data transfer and storing.

ABSTRACT
CONCEPT DEFINITIONS
CONTENTS

1 INTRODUCTION.....1

2 TASK OVERVIEW.....2

3 THEORETICAL BACKGROUND.....3
3.1 Information logging.....3
3.2 Tools.....4
3.3 Programming languages.....6
3.3.1 Delphi.....6
3.3.2 SQL.....7
3.3.3 PHP.....8
3.3.4 JavaScript.....8

4 DEVELOPING AN ERROR LOG MODULE.....10
4.1 Planning.....10
4.2 Backend function for fetching data.....11
4.3 UI module for error log.....15
4.4 Testing the implementation.....21
4.5 Results.....24

5 CONCLUSION AND DISCUSSION.....26

6 REFERENCES.....28
APPENDICES

FIGURES
FIGURE 1. SQL server management studio, a graphical tool for relational database management.....5
FIGURE 2. Debugging in Delphi IDE.....6
FIGURE 3. SQL Retrieving top transactions with just few lines of code.....7
FIGURE 4. PHP file can have mixed content.....8
FIGURE 5. JavaScript code example.....9
FIGURE 6. Dataflow between frontend and backend.....14
FIGURE 7. Folder structure for UI module part.....15

CODES
CODE 1. Implementation of TErrorLog child class.....11
CODE 2. Implementation of TLog parent class.....12
CODE 3. Implementation of TErrorLogFilter child class.....12
CODE 4. Part of the implementation of GetXML function.....13
CODE 5. Example output of GetXML function.....14
CODE 6. The error log html div-element.....16
CODE 7. XML string creation.....17
CODE 8. Creating table with tabulator library.....18
CODE 9. Creation of date range input fields with jQuery.....20
CODE 10. Implementation of TErrorLogFilterTests class.....22

CODE 11. Implementation of Setup and TearDown procedures 22
CODE 12. Implementation of TestParamDate test case 23

TABLES

TABLE 1. Main functionalities of files discussed in this chapter 20

1 INTRODUCTION

Logs are used to gather log entries that represent information. In information systems, logging can be used to keep track of events such as user activity, logins, logouts, or error tracking. Logging information in information systems can be critical and useful to spot vulnerabilities in the system. Gathering the log information is usually executed automatically by the system, and the logged information can then be processed by authorized people. In information systems, logging is the process of gathering useful information that can be later used to enhance the system performance, user experience or security. However, logging too much information can cause memory issues and make the logs not readable for human eye. Because of that, it is important to know what events to log, what to include in one log entry and to provide effective filters for the log. Based on these key points, I developed an error log module for SaaS platform. This task was commissioned by Vitec Raisoft Oy.

Vitec Raisoft Oy is an information and communications technology (ICT) company that operates in medical sector. They offered me this task of developing an error log module for SaaS platform they provide for their customers. The platform already had other logs, so my work had to stylistically align with them. Scope of the task was limited to implementing backend and frontend. Backend part required retrieving error log entries from database. The frontend part consisted of creating new module for the SaaS platform that would display the error log. In this thesis, I will present how I developed an error log module for Vitec Raisoft Oy. Theoretical part covers information logging, technical tools that were used, and programming languages. Development part covers the steps of the development process. This includes planning phase that goes through implementation workflow in more detail. Followed by backend implementation that covers, how the data from the database can be fetched. At the end of this thesis, I will describe development of the user interface (UI), how testing was handled and the end results.

The task objective is to have working error log module that can be attached to the SaaS platform. This error log module should stylistically align with rest of the software. The error log should not log too much information, but still enough for administrations of the system to identify problems. Filtering based on username and time interval must be implemented. The error log should be exportable as Excel, CSV, and PDF formats. My personal goals while working with this thesis included familiarizing myself with Raisoft.net SaaS platform, learning Delphi programming language and improving myself as a software developer. I had no prior experience in Delphi, so to accomplish this task, I had to do some deeper learning.

2 TASK OVERVIEW

This part covers the overview of the task including introduction to the company and requirements to accomplish the task. To grasp the concepts of this thesis, it is important to understand for whom this thesis has been written for and the reasons behind the development.

Vitec Raisoft Oy is an ICT company that operates in medical sector. The company was grounded in 2000 as “Raisoft” but since Vitec acquired Raisoft in December 2022, name has been changed to Vitec Raisoft Oy (Vitec Raisoft Oy 2023). The company offers cloud platform for data collection and reporting, called “Raisoft.net”. This software has been built to be modular, secure, and compatible with integrations. The centre of the application are assessment instruments which are developed by interRAI. These instruments are used to measure health care need for people of all ages. Raisoft.net SaaS platform is not tied with any enterprise resource system, and it is widely in use all around the world. (Vitec Raisoft Oy 2023.)

My job was to implement an error log module for the platform. The implementation started by first looking at the requirements. The requirements included filtering of the log entries based on time interval and username. To fill the requirements, I took a deeper look at the other logs in the system and how filters were implemented on those. The backend had to be implemented with Delphi programming language, which I had not used before, so I reserved some time for learning it myself. The scope of the task seemed to be large at first, so I decided that the best way to implement the error log module was to program it in two parts. First, I had to implement a backend function to fetch the data from relational database. The second part consisted of creating a UI that displayed logs to the user. The detailed implementation is described in chapter 4 of this thesis.

The goal of the task was to have an error log that displays log entries of errors in readable format. These log entries hold error data, such as timestamp, username, and error message. The errors are already collected automatically to the relational database. A way to fetch the errors and display them needed to be implemented. This information is used to enhance user experience and to detect vulnerabilities of the system.

3 THEORETICAL BACKGROUND

Developing and maintaining SaaS platform is a challenging job. Customer's standards are constantly tightening and the only way to keep up with the customers level of requirement is to enhance user experience of the software. To enhance user experience, developers need to have the capability to capture and store errors encountered by the users. For this, it is important to implement an automated error logging system. In this thesis, I am going to implement an error log for Raisoft.net SaaS platform. This chapter discusses the concepts of information logging, as well as technical tools needed for development work. At the end of the chapter, programming languages used for backend and frontend development are described in more detail.

3.1 Information logging

It is necessary to understand the very basics of logs and information logging before implementing error log module. Log entries, sometimes referred as log messages, are the single lines of information in logs. To use these log entries effectively, the key information of the log entry must be extracted. The information extracted from the log entry is called log data. Cumulatively, log entries build the complete log record, referred as log. Logs are effective source of information, for instance computer application management or security related processes. The general categories of log entries are, informational, debug, warning, alert, and error. (Schmidt, Phillips & Chuvakin 2012, chapter 1.) The focus of this chapter is in error log category.

Error logs are implemented to store, for instance, errors occurred in the software. One log entry of an error log can contain, for example, timestamp, username, and error message. The log entry is descriptive, giving information about the user who got the error, when the error occurred and detailed information about the error itself. Error logging must be automated to have accurate and updated error log. In today's software development, monitoring behaviour of the software is essential to identify problems. These problems can harm the entire business and cause absurd costs. (Candido, Aniche & Deursen 2021.)

Implementation of software logs is done by developers during the development process. Log entries are stored to databases, either relational or non-relational. Complete logs can grow to be a size of terabytes even in smaller scale businesses so during the implementation of the log, it is important to know exactly what information to log. This prevents unnecessary information to be stored. (Schmidt, Phillips & Chuvakin 2012, chapter 4.) During this thesis work, I had to learn basic concepts of information logging. Since the errors were already collected to relational database table, I had to analyze the table schema and determine what information is necessary to include in the error log. Before the development work started, it was necessary for me to familiarize myself with the given tools for the development. Next chapter discusses development tools in more detail.

3.2 Tools

This chapter describes the development tools and integrated development environments (IDE) used in process of implementing the error log module. Picking up the right tools can speed up the development process. For software development, there are tools available for different purposes. Integrated development environments are used for debugging and writing the code. Version control systems have their own tools, for instance, git. Database management tools helps developers to access, modify and visualize data in databases. Programming languages can be considered as tools for developer to communicate with computers. Different programming languages are utilized for different purposes. However, tools for this task were already decided by the commissioning company. These tools are described in more detail in this chapter.

Visual studio code is an IDE which supports multiple programming languages. It can be used as lightweight desktop application or web application. There are multiple extensions offered for visual studio code that can simplify development process. These extensions offer for instance, debuggers and syntax correction. Git has been integrated to the Visual studio code, meaning that git commands can be run directly from the Visual studio code's own terminal. Customizable configurations and extensions can enhance developers experience, although Visual studio code functions seamlessly without additional configurations or extensions. (Visual Studio Code 2023.) Visual studio code was used as code editor during the frontend development of an error log module.

For development, version control system is crucial. It enables tracking of changes applied to the code. Git is a version control system that allows developers to create parallel versions of the code called branches. These branches are used to implement changes to the code; however, these changes can be reverted at any time without affecting the original code. Once the code in branch is finished, it can be merged with the original code. Git configured repository starts with one branch, called master where the original code is stored. (Blischak, Davenport & Wilson 2016.) During the development work of this thesis, git was utilized, and tasks were developed on different branches. This allowed work to be tracked and the original code base stayed intact. The code was reviewed and tested by other developers before it merged with the master branch.

The development of an error log module included analysing the error log entries stored into relational database. SQL server management studio (SSMS) was used to access, configure, and test the data during the backend development of an error log module. It is a graphical tool to connect, manage and edit multiple relational databases. Visualisation of the structure of the databases allows developers and administrators of all skill levels to easily work with relational databases, as seen in FIGURE 1. SQL server management studio contains components which can be utilized for different purposes. For instance, "visual database tools" component is utilized for building queries, tables, and diagram databases. In the development phase of the error log module, the component called "Object explorer" was utilized the most. It allows developers to view and manage databases, tables, and objects of SQL server. (Microsoft 2023.)

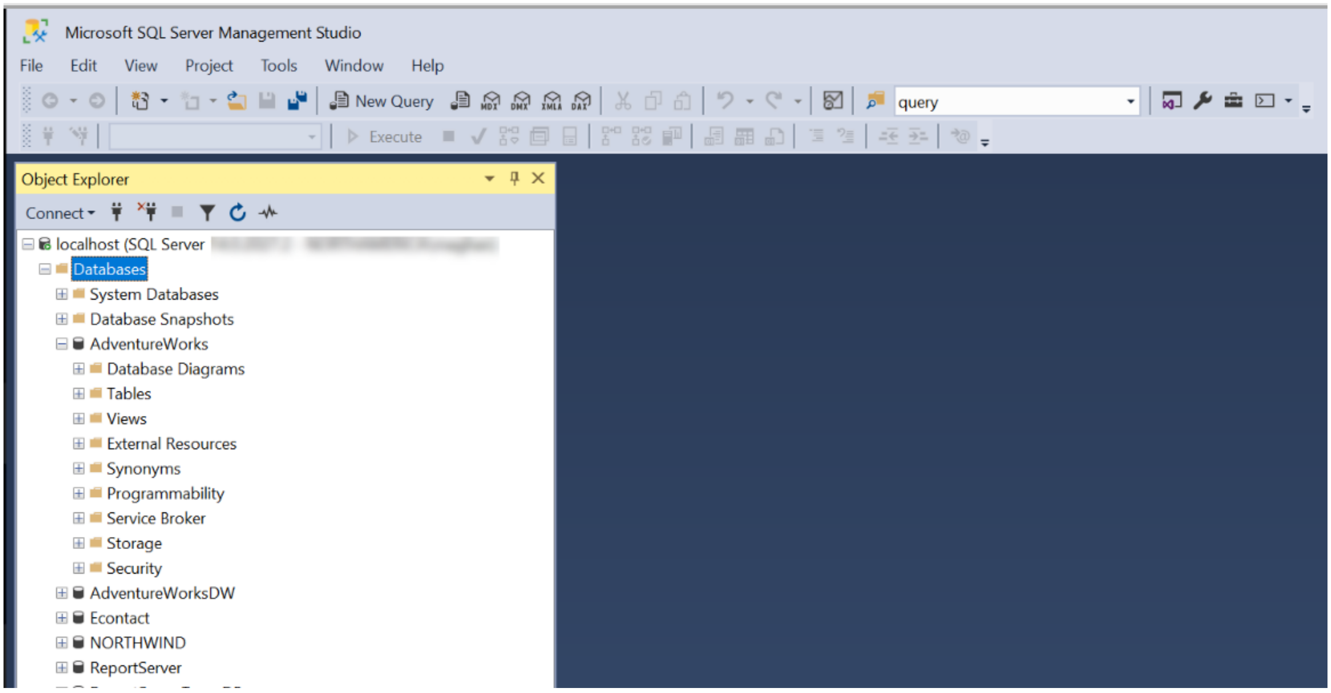


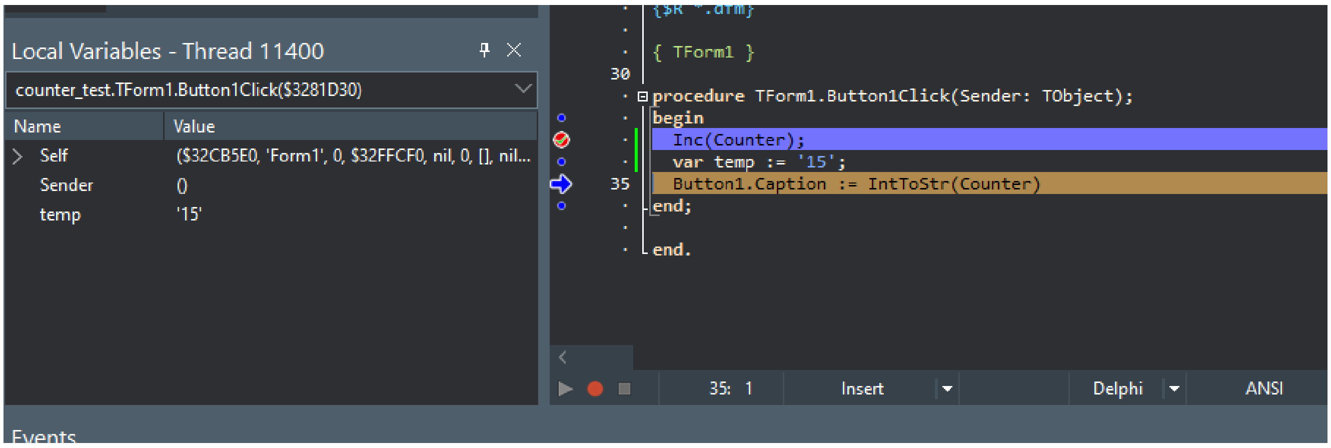
FIGURE 1. SQL server management studio, a graphical tool for relational database management (Microsoft 2023).

3.3 Programming languages

The part of web application that handles data storage and logic for the application is called backend, also referred as server-side. This part of the code does not have direct user access. Indirect access is achieved by user via frontend. The layer between user and server is called frontend. Frontend, also referred as client-side is the visible and interactive part of the application. It aims to have optimal performance and responsiveness, to make sure that user experience is pleasing. Visual aspects, such as buttons, forms and menus are part of frontend. (GeeksforGeeks 2023.) For frontend and backend there are multiple programming languages available, and this chapter will present languages used for this thesis work.

3.3.1 Delphi

Delphi is a programming language and an IDE based on Object Pascal programming language. It is used to build applications for multiple platforms and operating systems (Embarcadero 2023). Delphi is high-level and strongly typed programming language. Object-oriented practises are extensively integrated into Delphi. With Delphi, multiple operations can be encapsulated into procedures and functions. The difference is that functions return value, while procedures do not. Units in Delphi refer to pieces of source code, which are separated Delphi files with the extension of “. pas”. With Delphi, coding is simplified with visualization. This can be noted while debugging with the editor, as executable lines of code are marked with blue dots which can be used to set break points. (Głowacki 2017, chapter 2.)


Local Variables - Thread 11400
counter_test.TForm1.Button1Click($3281D30)
Events

Name	Value
Self	($32CB5E0, 'Form1', 0, $32FFCF0, nil, 0, [], nil...
Sender	()
temp	'15'

```
{$R *.dfm}

{ TForm1 }

procedure TForm1.Button1Click(Sender: TObject);
begin
  Inc(Counter);
  var temp := '15';
  Button1.Caption := IntToStr(Counter)
end;

end.
```

```
35: 1    Insert    Delphi    ANSI
```

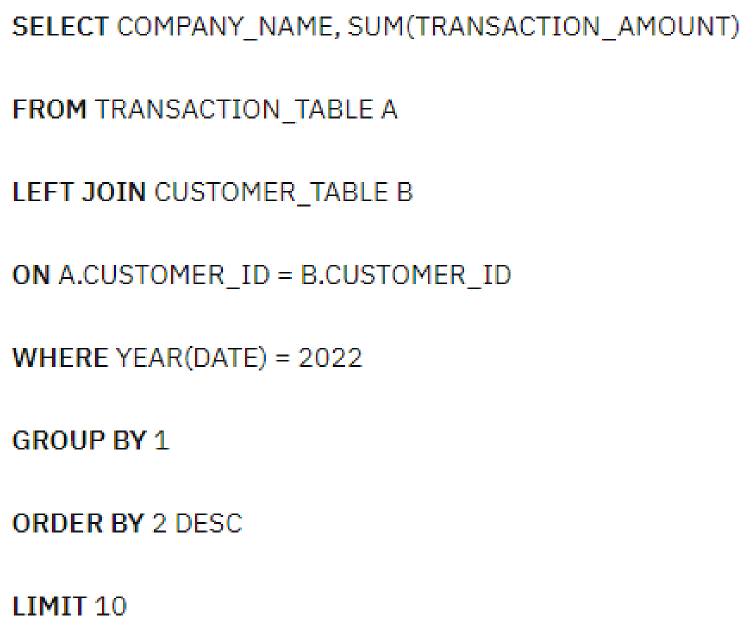
FIGURE 2. Debugging in Delphi IDE.

As seen from FIGURE 2, developers can go through the code execution step by step from the set breakpoint. The red dot indicates the breakpoint, and the blue arrow indicates the current execution point.

Variables and their values are seen on the left box which simplifies the developer's ability to observe if something is wrong. If the code contains a significant number of variables, it might be difficult to track the desired variables and their values. Delphi solves this problem by allowing the developer to choose variables to track. Delphi was already used as backend language for Raisoft.net platform; therefore, it was mandatory to use it for backend part of the error log module.

3.3.2 SQL

Relational databases stores data in structured format. Database consists of tables which hold unique information in rows and columns. Row contains each information entity and column define what data is available for that entity. Structured Query Language (SQL) is the standard way to retrieve data from relational databases. The possibility to join relational database tables together allows user to create meaningful information. With SQL, data can be retrieved with just a few lines of descriptive code as seen in FIGURE 3. (IBM 2023.)



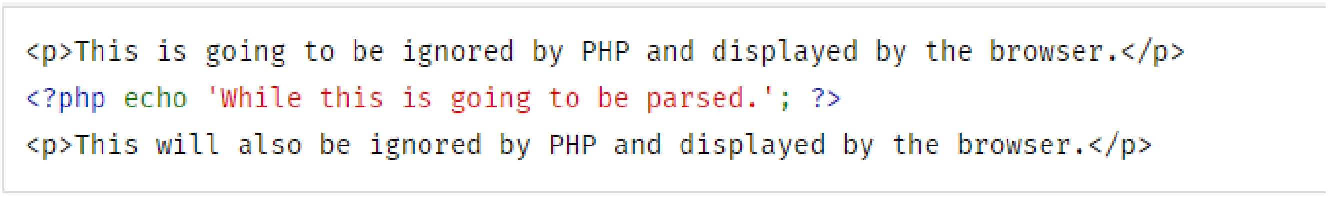
```
SELECT COMPANY_NAME, SUM(TRANSACTION_AMOUNT)

FROM TRANSACTION_TABLE A

LEFT JOIN CUSTOMER_TABLE B

ON A.CUSTOMER_ID = B.CUSTOMER_ID

WHERE YEAR(DATE) = 2022

GROUP BY 1

ORDER BY 2 DESC

LIMIT 10
```

FIGURE 3. SQL Retrieving top transactions with just few lines of code (IBM 2023).

SQL syntax is easy to understand and read. In this query, data is collected from two tables that are joined together using customer information. Only data from 2022 is collected and results are grouped, ordered, and limited to top ten. SQL was used to retrieve error data for the error log module.

3.3.3 PHP

PHP is an open-source scripting language, which main goal is to allow users to create dynamic web pages quickly. With PHP users can create command-line applications as well as desktop applications; however, the language is best utilized for web development and more specifically server-side scripting. PHP code is usually embedded in HTML with PHP-tags and the tags are then executed by server and content is displayed on client-side. When the server is processing code and creating the results, the users cannot see the code which can be considered as security advantage. What makes PHP versatile is the ability to work on major operating systems, the ability to code either procedural or object-oriented code and simplicity of the language for newcomers to learn. (PHP 2023.) An example of PHP script syntax can be seen in FIGURE 4.



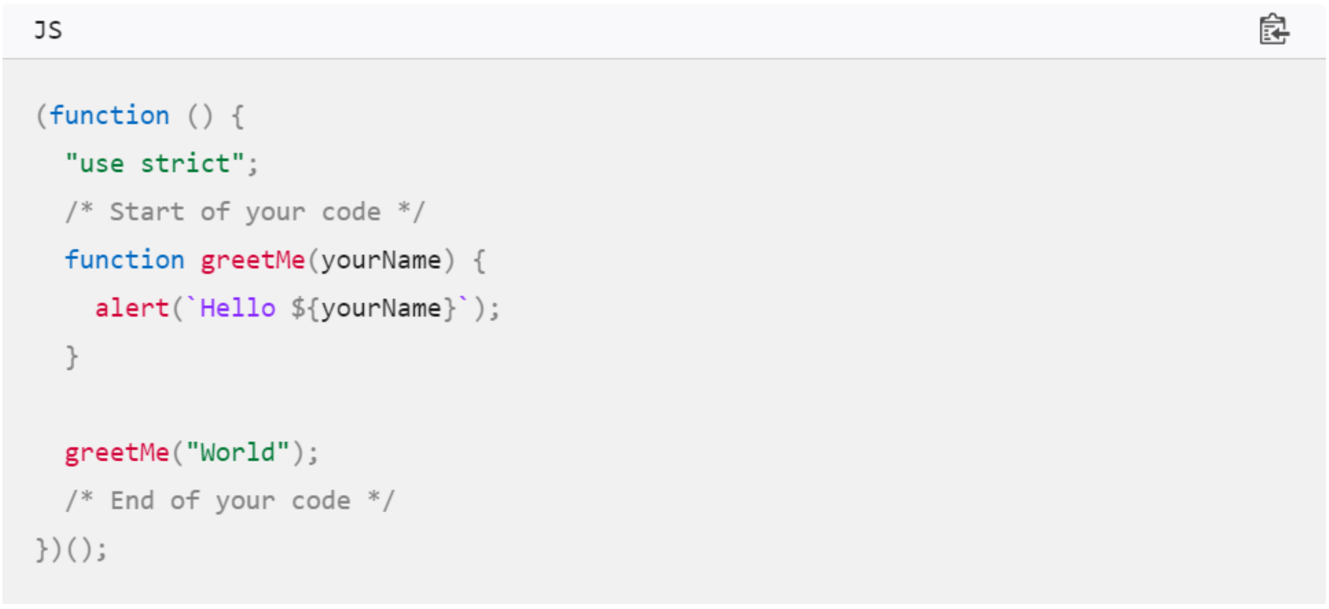
```
<p>This is going to be ignored by PHP and displayed by the browser.</p>
<?php echo 'While this is going to be parsed.'; ?>
<p>This will also be ignored by PHP and displayed by the browser.</p>
```

FIGURE 4. PHP file can have mixed content (PHP 2023).

PHP code starts with the opening tag and ends with closing tag as seen in FIGURE 4. These tags are marks for PHP parser to process code between the tags. Any other content outside of the tags is treated as normal HTML content. This enables mixed content to be applied to PHP files. The frontend part of this thesis work was built with PHP and functionality was added with JavaScript.

3.3.4 JavaScript

JavaScript is the most known programming language for web development. It supports different programming styles such as object-oriented and declarative. The main point of JavaScript is to add dynamic aspect for web applications, by making content interactive. JavaScript can be used either server-side with technologies such as Node.js or client-side via web browsers. Web browsers have built-in JavaScript engine that runs JavaScript code directly on users’ machines. Applying JavaScript to web application can be accomplished by writing code directly to HTML or PHP file and wrapping it with script tags. However, it is considered poor practise since it reduces code readability and maintainability. (MDN contributors 2023.) Best practise is to create file with “.js” extension and put the JavaScript code there, which can be referred to in HTML or PHP files. Example of a JavaScript code can be seen on FIGURE 5.



JS

```
(function () {
  "use strict";
  /* Start of your code */
  function greetMe(yourName) {
    alert(`Hello ${yourName}`);
  }

  greetMe("World");
  /* End of your code */
})();
```

FIGURE 5. JavaScript code example (MDN contributors 2023).

The example JavaScript code in FIGURE 5 declares function “greetMe” which takes “yourName” as parameter and utilizes alert function to display “Hello” and the given parameter in user’s browser. In the example greetMe function is called with “World” string so running this code would result alert in user’s browser displaying “Hello World”. This example displays the simplicity of function declaration with JavaScript. JavaScript has been used for the error log module frontend functionality. The next chapter discusses the development process as the main tools and technologies have been presented up to this point of this thesis.

4 DEVELOPING AN ERROR LOG MODULE

The implementation of the error log module is described in this chapter. I will describe how implementation went, the workflow and how development tools described in previous chapters were utilized. The goal was to have working error log module that could be attached to Raisoft.net platform. I decided to split the work in two tasks, one for the backend part and one for the frontend part. Since the backend part of the implementation was programmed with Delphi, I had to spend some time to learn the programming language. That also strengthened my object-oriented programming (OOP) skills since I had to use the approach during the development work. The frontend implementation was programmed with popular web programming languages, PHP, and JavaScript. I had prior experience on those which made the development process easier for the frontend part. At the end, I managed to implement working error log module for the Raisoft.net SaaS platform.

4.1 Planning

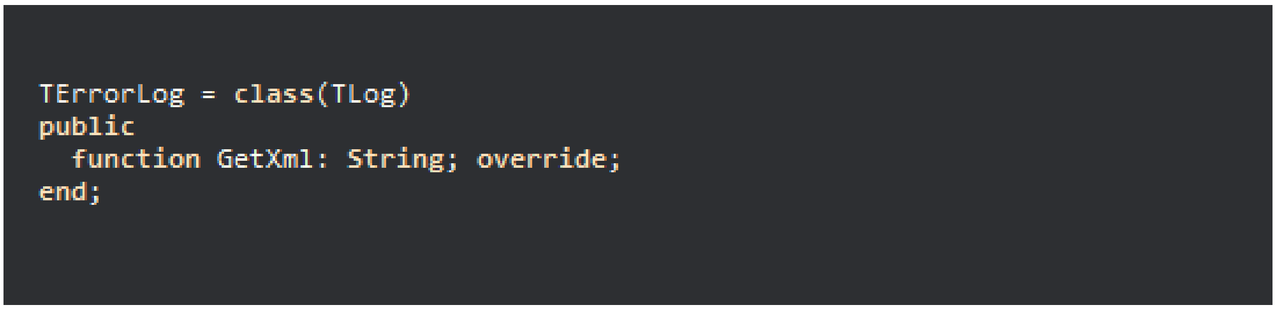
It is important to allocate some time to list requirements, steps and expected outcome before starting the implementation phase. This is seen as the planning phase. The commissioned company informed me about the expected results, gave me list of requirements and tools for the work. Since this work had no due date, it was up to me to schedule the implementation. Task was already split in two parts, backend, and frontend.

The plan for implementing backend was clear. I spent some time investigating other log modules and their implementation. Since the errors were already stored in relation database table, I only had to fetch the correct data. For that I had to implement a function that handles fetching the data when UI calls the function. Filters for username and time interval were requested features and so had to be implemented to the backend. For me to understand structure of the database table and what data would be fetched, it was crucial to utilize SSMS. With this tool I was able to access the existing database table and identify the correct columns that needed to be accessed with backend database queries. After I had analysed the database table schema with the help of SSMS and planning of the backend work was done, it was time to plan work for the frontend part.

The plan for frontend implementation required more time. The required functionalities for UI part included visual aspects of the error log. The error log had to be implemented following the styles of other logs and the functionality had to align with the backend implementation. The frontend had to show clear filter options for the user, and the ability to export log file as PDF, Excel, or CSV. I decided to start from the backend implementation. The reasoning behind the decision was that testing of the UI part would be easier when UI is able to get logs from database. Next part of this thesis discusses the implementation of the backend functionalities in more detail.

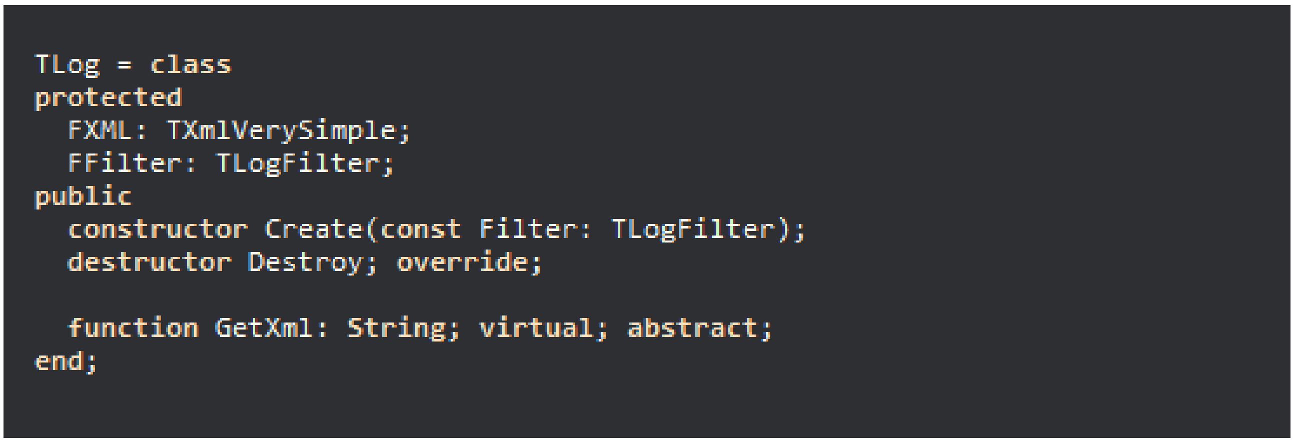
4.2 Backend function for fetching data

Programming IDE and language for backend part was decided to be Delphi. I started the development work by creating a new class for the error log module called "TErrorLog". Classes and objects are the key features of OOP. Objects are created from classes, and they inherit all class properties and methods, however each object has different values for the inherited properties. (W3Schools 2023.) This allows developer to write class once but utilize it for multiple instances. TErrorLog class inherited procedures, functions, and properties of an already made parent class for other logs. The parent class had declared abstract method "GetXML" for fetching the data from database. The method being abstract required implementation by the "TErrorLog" class.


```
TErrorLog = class(TLog)
public
  function GetXml: String; override;
end;
```

CODE 1. Implementation of TErrorLog child class

TErrorLog class gets created and it inherits properties, functions, and procedures from TLog class. Public function GetXML is declared. The output type for the function is set to be string and it will be set to override function from the parent class TLog. Other properties from the parent class does not need to be declared, yet TErrorLog class can still utilize them. The implementation of TErrorLog can be seen on CODE 1.

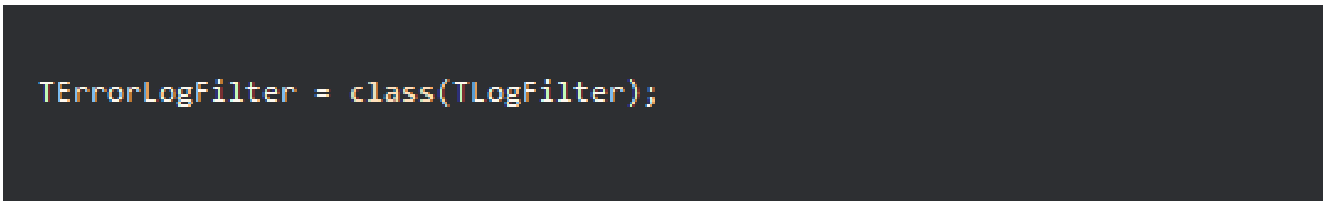


```
TLog = class
protected
  FXML: TXmlVerySimple;
  FFilter: TLogFilter;
public
  constructor Create(const Filter: TLogFilter);
  destructor Destroy; override;

  function GetXml: String; virtual; abstract;
end;
```

CODE 2. Implementation of TLog parent class

In CODE 2, implementation of the parent class is presented. Properties under the protected keyword enable accessibility of the properties for this class and its child classes. Public keyword defines constructor which takes filter for the log as parameter when creating object from that class. Destructor is called when an object is deallocated from memory. At the end, there is a declaration for GetXML function, which has been implemented in TErrorLog class. However, while only GetXML function is implemented, all other functions and properties are available for TErrorLog class to utilize. In the later sections of this thesis, I will be implementing filtering on UI side, which a user can use to filter logs. This filter is sent to backend as input, which is used to create object of TErrorLog class. This object will be used to return desired error log entries based on the given filter.

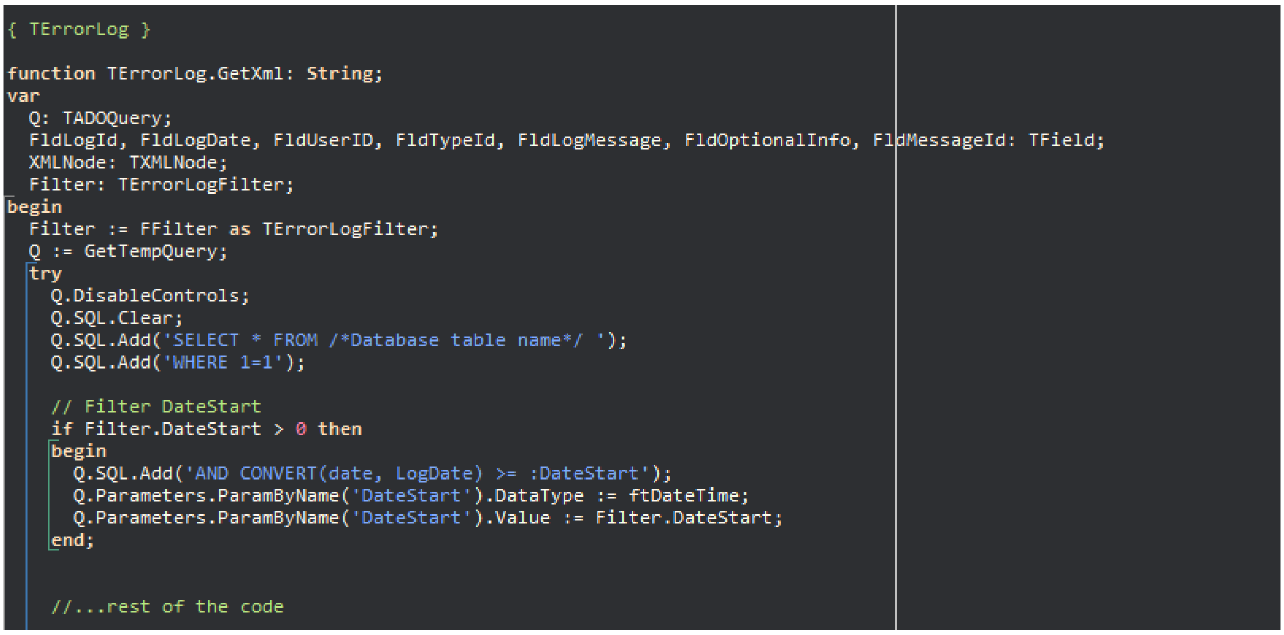
Before implementing the method “GetXML”, I had to develop the filter, because it would be used in the SQL queries and when creating an instance of the TErrorLog. An abstract class for filtering logs with necessary functions and properties already existed in the code base. I created a child class from that which I would be utilizing when creating an instance of the TErrorLog. Creation of the child class can be seen in CODE 3.



```
TErrorLogFilter = class(TLogFilter);
```

CODE 3. Implementation of TErrorLogFilter child class

Now that the filter was created, I started working on the GetXML method. The purpose of this method is to retrieve data from relational database. For that I had to use SQL queries with the filter I created before. Queries retrieved data from relational database table line by line while applying filters and generating XML string. The XML language is a markup language designed to store and transfer data. The XML language encapsulates data inside descriptive tags, which software can then read and parse. (W3Schools 2023.) The most issues I had during this implementation came from understanding how to use SQL queries with Delphi. After investigation, I found that there was a ready-made component that I could use for the queries.

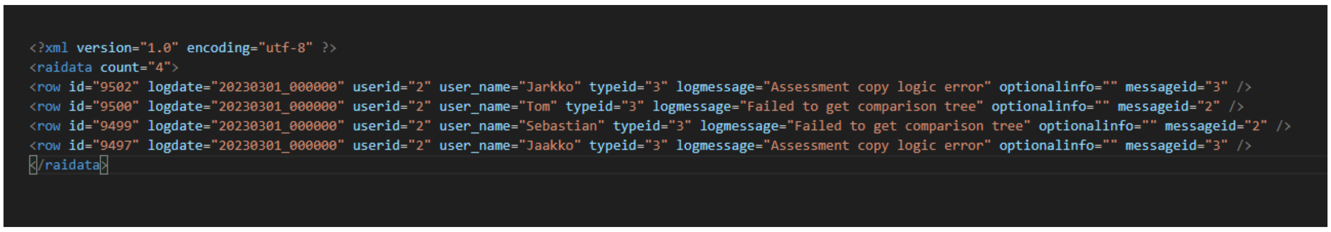


```
{ TErrorLog }

function TErrorLog.GetXml: String;
var
  Q: TADOQuery;
  FldLogId, FldLogDate, FldUserID, FldTypeId, FldLogMessage, FldOptionalInfo, FldMessageId: TField;
  XMLNode: TXMLNode;
  Filter: TErrorLogFilter;
begin
  Filter := FFilter as TErrorLogFilter;
  Q := GetTempQuery;
  try
    Q.DisableControls;
    Q.SQL.Clear;
    Q.SQL.Add('SELECT * FROM /*Database table name*/ ');
    Q.SQL.Add('WHERE 1=1');

    // Filter DateStart
    if Filter.DateStart > 0 then
    begin
      Q.SQL.Add('AND CONVERT(date, LogDate) >= :DateStart');
      Q.Parameters.ParamByName('DateStart').DataType := ftDateTime;
      Q.Parameters.ParamByName('DateStart').Value := Filter.DateStart;
    end;


    //...rest of the code
```

CODE 4. Part of the implementation of GetXML function

With TADOQuery it is possible to access relational database tables with SQL statements. Retrieving, inserting, deleting, and updating data in relational databases is achievable with this component. (Embarcadero 2014.) I started by initializing a filter, TADOQuery, XML node and fields. I cleared TADOQuery with clear method to make sure it does not contain any unwanted statements. Once the query was clear I started to build it step by step by declaring the select statement first. With “SELECT * FROM” I could retrieve all columns from the database and by including “WHERE 1=1” allows me to continue the statement with “AND” statements. After that I started to apply filtering for the query based on the filter object. For instance, the first filter checks whether the start date is set as see in this CODE 4. When all the filters had been applied to the query, it was started, and database table rows were collected to field list. Now that I had all the wanted rows as the result of the query, I implemented a while loop where I created XML string for each row collected and the XML string had attributes for each column collected.

Example output can be seen in CODE 5. This XML string is the results backend returns to UI when user is searching error log entries. Each row presents a row in the database table where information got fetched. The frontend part then parses this information and displays it in a more convenient way.



```
<?xml version="1.0" encoding="utf-8" ?>
<raidata count="4">
<row id="9502" logdate="20230301_000000" userid="2" user_name="Jarkko" typeid="3" logmessage="Assessment copy logic error" optionalinfo="" messageid="3" />
<row id="9500" logdate="20230301_000000" userid="2" user_name="Tom" typeid="3" logmessage="Failed to get comparison tree" optionalinfo="" messageid="2" />
<row id="9499" logdate="20230301_000000" userid="2" user_name="Sebastian" typeid="3" logmessage="Failed to get comparison tree" optionalinfo="" messageid="2" />
<row id="9497" logdate="20230301_000000" userid="2" user_name="Jaakko" typeid="3" logmessage="Assessment copy logic error" optionalinfo="" messageid="3" />
</raidata>
```

CODE 5. Example output of GetXML function.

Backend functionality for fetching the data has been pretty much implemented up to this point. Now I had to implement a function for UI to call. It would be placed somewhere between backend and frontend as an interface function. This way UI does not have direct access to the backend. I implemented this interface function based on how similar functions were implemented for the other logs. Basically, the function would take XML string as an input and use that to create a filter. The filter is then used to implement TErrorLog object and as a result would return result from GetXML method. In normal state the error log would just display certain amount of log entries. If user sets filters based on, for example, user or time, then these conditions are passed to the backend as parameters. Dataflow between UI and backend can be seen in FIGURE 6.

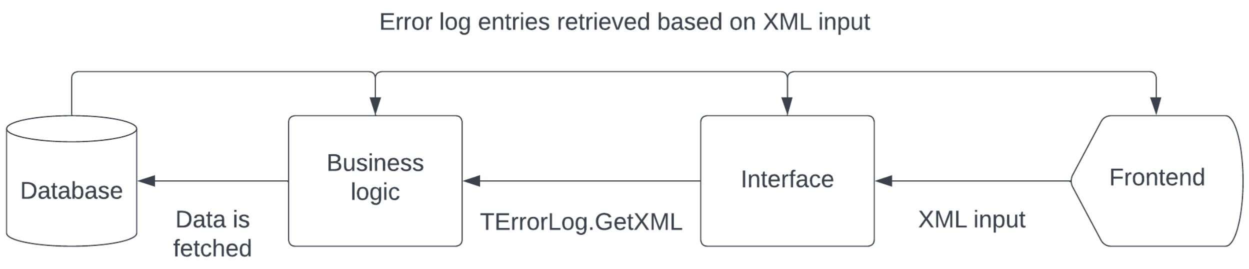


FIGURE 6. Dataflow between frontend and backend

This part of the code was programmed by creating multiple smaller classes that all had their own job to handle. I was not confident with OOP practices as they were heavily involved in this task, however, as I progressed with the implementation, my OOP skills got better. At the end, I implemented unit tests for the backend functions, as well as test cases for manual testing. I will describe the testing process later in this chapter. Now that the backend implementation was completed, it was time to focus on the UI side. Next chapter describes the implementation of the user interface.

4.3 UI module for error log

I started the implementation by creating a new folder for the error log module. This folder would include all the necessary files for the frontend part of this project. I wanted to split the files in different folders based on the functionality. The main folder at the top layer contains all the necessary subfolders and files for the frontend part. As seen from FIGURE 7, the files have been categorized based on the functionality they have, and this significantly increases code readability. As stated before in chapter 3.3.3, this part of the project was developed with PHP and JavaScript. The files that handle data processing, communication with backend and rendering the UI content have been programmed with PHP. The main functions of the error log have been programmed with JavaScript. The Programming for frontend part was done utilizing Visual Studio code IDE. The implementation can be split into three different steps, creating an empty page, implementing communication between frontend and backend, and the last part would be adding UI side functionality. In the actual development work, I was jumping between files since some parts of the code were used in multiple places, however, for the clarity of this thesis, I will go through the implementation following the steps in order, as mentioned. The most important functionalities are explained, but not all the files are going to be explained in detail.

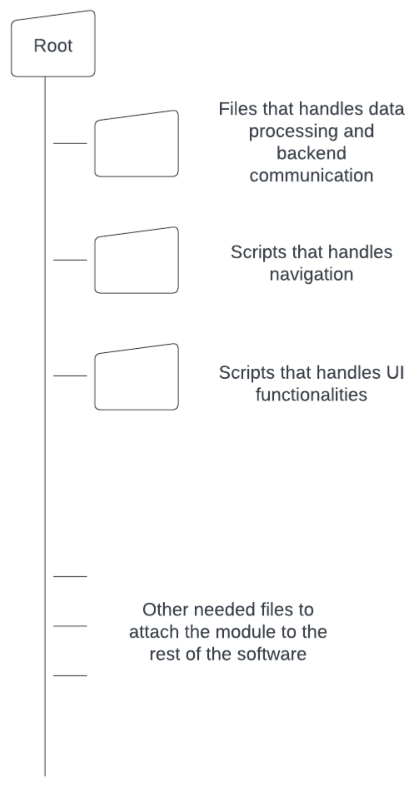
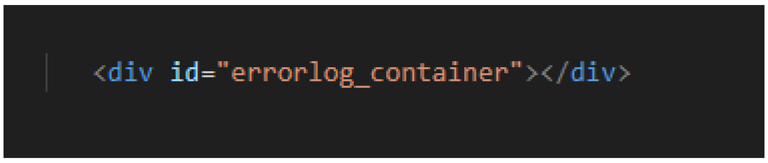

Root
Files that handles data processing and backend communication
Scripts that handles navigation
Scripts that handles UI functionalities
Other needed files to attach the module to the rest of the software

FIGURE 7. Folder structure for UI module part

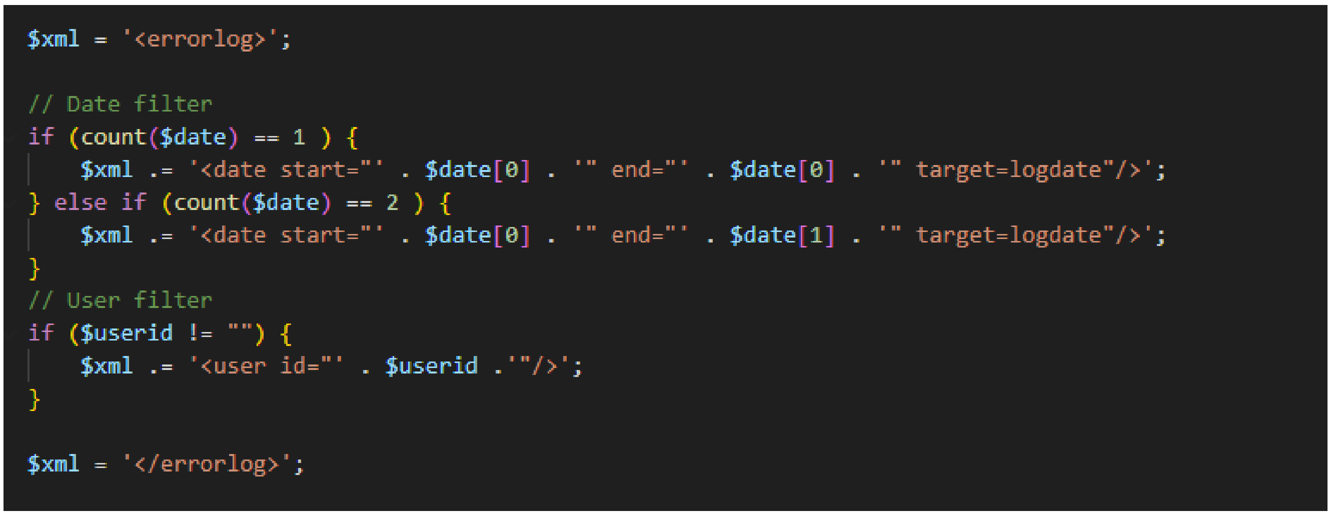
The implementation for the empty error log page did not require any communication with the backend. Basically, it was just creating an empty page in the software that had the layout which stylistically aligned with the other logs. I started to develop this, first by looking at the general structure of the other logs and how their frontend rendering had been implemented. I created a PHP file that contains HTML elements that would be then parsed on the server side. These elements define the structure of the page, in this case they were yet empty elements acting as placeholders. For instance, the main container for the error log table that would show all the log entries was built using div-element with an id-attribute. The code that fills the table would then access this unique id to show the log entries. This allowed me to write clean PHP file that hold only the structure of the error log page and rest of the functionality is then implemented elsewhere. The error log element can be seen in CODE 6.


```html
<div id="errorlog_container"></div>
```

CODE 6. The error log HTML div-element.

In CODE 6, the basic structure of HTML element can be seen. The element starts with an opening tag which holds the name of the element, in this case div. Opening tag can hold attributes which contain additional information, but it will not be shown in the content (Mozilla 2023). I used id-attribute so the code that generates the actual content can find the right element via this attribute. At the end of the element there is a closing tag which marks the end of the element (Mozilla 2023). Normally element content would be placed between the opening and closing tags, however, in this case I have left it empty since it is going to be filled with the error log table. The other UI elements like filters and navigation bars were implemented the same way by using the same PHP-file with placeholder HTML elements. I had split the code into multiple files, for instance UI side functionalities I put on JavaScript files and data processing methods on PHP files. However, there were also general files that other modules used as well, that I had to use in this project. Luckily, the files were easily accessible in PHP file with include function. In the next part of this thesis, the development of the communication between frontend and backend is described in more detail.

Developing the connection between frontend and backend started by looking at the backend code I had created before. Implementing the connection would require frontend to call the interface function with XML string as parameter. First, I created a new PHP-file that would be mainly handling creation of the XML string. In the PHP-file, the code first retrieves inputs from UI, as the values of filters. These values consist of date and username. After inputs from the UI have been collected, the code generates the XML string based on the input values. Once the XML string is complete, the code calls the interface function I created before. XML string gets passed as parameter and result gets passed to a variable. Generation of the XML string can be seen in CODE 7.

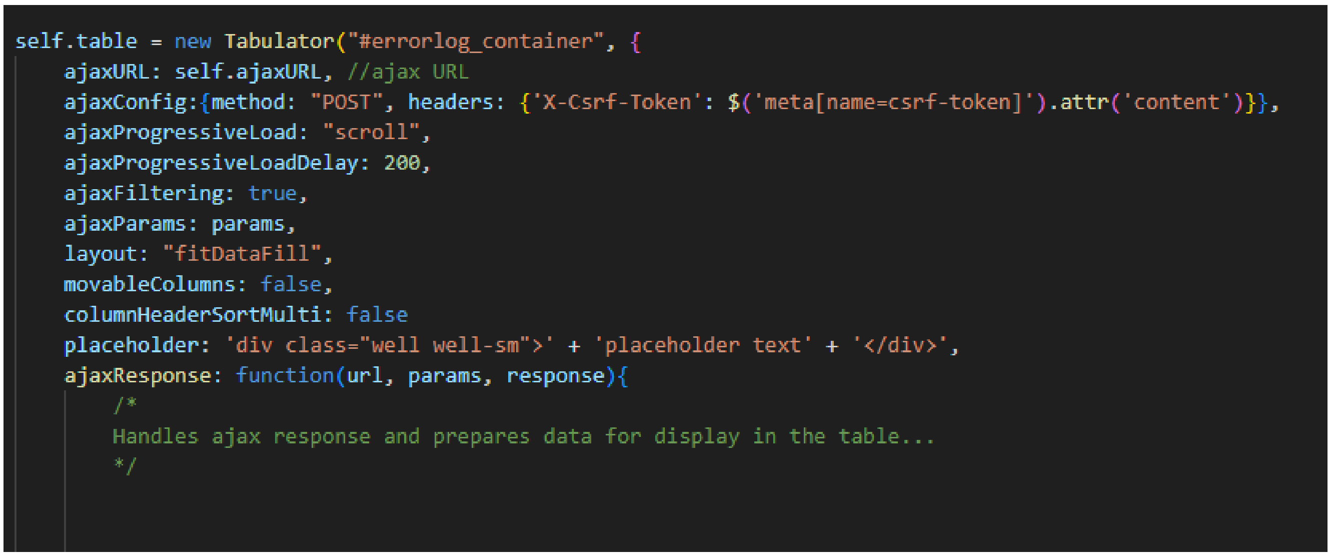


```php
$xml = '<errorlog>';

// Date filter
if (count($date) == 1 ) {
    $xml .= '<date start="' . $date[0] . '" end="' . $date[0] . '" target=logdate"/>';
} else if (count($date) == 2 ) {
    $xml .= '<date start="' . $date[0] . '" end="' . $date[1] . '" target=logdate"/>';
}
// User filter
if ($userid != "") {
    $xml .= '<user id="' . $userid .'"/>';
}

$xml = '</errorlog>';
```

CODE 7. XML string creation.

The XML string generation has been implemented step by step as seen in CODE 7. It was requested that error log entries would be able to filter based on time interval or username. These filters were added to the XML string that would be used as parameter for interface function which returns error log entries for UI. When the user has chosen the desired time or username filters for error log, the values get passed to this PHP file that handles XML string generation. Filters are applied to the XML string, starting at the date filter. The dates are passed to an "date" array and the first entry of the array holds start date and the second entry holds end date. In the code, I implemented an if-statement that checks whether the date-array holds one or two entries. If there is one entry in the array, I assume that user wants to retrieve error log entries from that specific date and so I set the start and end dates with the same value. If there are two entries, I assume that the user is trying to set up interval from which the errors should be retrieved. The username that gets passed as filter, gets mapped to a user id earlier in this PHP file. For the implementation of the user id filter, I have implemented basic if-statement that adds user id filter if the variable is not empty string. At the end I closed the XML string. This XML string will be passed to the interface function which will then create a filter based on that. The filter will be then used to create an object of TErrorLog class which have been described in chapter 4.2 of this thesis. At the end, results retrieved from backend are returned to the UI. The purpose of this PHP file is to retrieve UI inputs and use them to call interface function, and this connects frontend with backend. I have also included generating PDF, CSV and Excel files to this PHP file, however, I will not go through the implementations in this thesis. Now that the empty UI page and connection between backend and frontend have been implemented, it was time to populate the UI with error log entries and filters. In the next part I will describe populating the empty UI page with content in more detail.

Server-side implementation had been done so far for everything that was required. The final step would be to populate the UI with content. For this, I created a JavaScript file that would handle all the main functions of the module as well as creation of filter HTML elements. In this thesis I will not explain all the functions, but the focus is on the most important functions. For populating the error log on the UI side, I used Tabulator library. The Tabulator library is simple to use, contains plenty of features and can be used to create interactive data tables with plain JavaScript. It is available under the MIT license. (Tabulator 2023.) Populating error log table with Tabulator library can be seen on CODE 8.

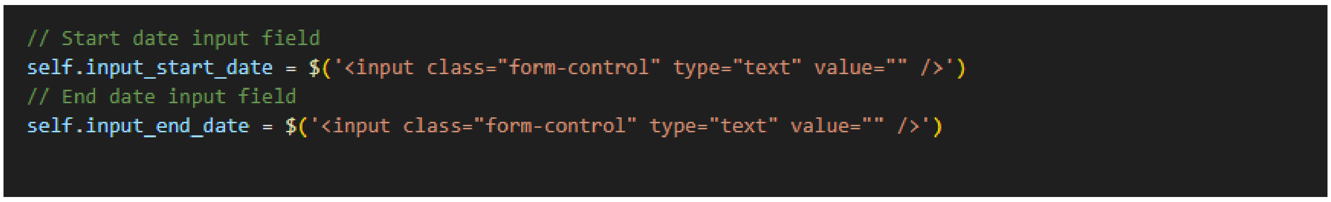


```
self.table = new Tabulator("#errorlog_container", {
    ajaxURL: self.ajaxURL, //ajax URL
    ajaxConfig:{method: "POST", headers: {'X-Csrf-Token': $('meta[name=csrf-token]').attr('content')}},
    ajaxProgressiveLoad: "scroll",
    ajaxProgressiveLoadDelay: 200,
    ajaxFiltering: true,
    ajaxParams: params,
    layout: "fitDataFill",
    movableColumns: false,
    columnHeaderSortMulti: false
    placeholder: 'div class="well well-sm">' + 'placeholder text' + '</div>',
    ajaxResponse: function(url, params, response){
        /*
        Handles ajax response and prepares data for display in the table...
        */
```

CODE 8. Creating table with tabulator library.

As seen from this CODE 8 the Tabulator library allowed me to configure the table in a convenient way. Not all the configurations are going to be explained in detail, but the focus is on the important features that Tabulator library allowed me to do. When creating an instance of Tabulator, I was able to determine the target where the table would be created. The first line of CODE 8 initializes a new instance of Tabulator, sets the table to be created in HTML element with id attribute, “errorlog_container”. At this point is good to refer to the previously explained CODE 6, which holds the div element with the correct id for Tabulator to populate. After the instance of the Tabulator had been created, I configured settings, like “ajaxURL” which I used to point to the previously explained PHP-file that handled retrieving error log entries from backend. Now the data for the table is fetched from that PHP-file using AJAX. AJAX (Asynchronous JavaScript and XML) is a set of technologies to build interactive web pages. It utilizes XML for data transfer between server and client application. AJAX utilizes asynchronous JavaScript for retrieving data in XML format from server without needing to refresh the web page. This allows user to continuously use the web application while the content of the web page is updating. (IEEE 2005.) The “ajaxConfig” determines the type of request I am sending. The PHP file that handles fetching requested data from the backend is taking inputs like username and date and use those to create the parameter XML string for the backend call. So, to get the data from “ajaxURL” the type of request needs to be “POST”. For the request, I needed to have the input values from the filters configured to be in Tabulator property called “ajaxParams”. These parameters, I have retrieved earlier in this code and passed to variable “params”. The “placeholder” property is presenting the content that is shown when table is loaded, or the content is not available for some reason. Once the error log entries have been retrieved from backend, the data is prepared for display in the “ajaxResponse” function. I will not go through other properties of the Tabulator in this thesis. Populating the error log page with table of error log entries was just one of the main functions that got implemented in this JavaScript file.

Next step for me was to implement filters that were placed on the UI. For that I used jQuery to handle HTML document manipulation on JavaScript side. jQuery is a fast and small JavaScript library that provides plenty of features to handle HTML elements and AJAX. jQuery is available under MIT license. (jQuery 2023.) With jQuery I was able to create HTML elements and input fields on JavaScript side. After the required fields were created, I combined them and pushed them to the container that would be shown on the UI side. For instance, the required fields for date filter included input fields for start and end date. Of course, these input fields would need to be linked to keep aligned styling, so I had to push the fields inside of div-element which I created with jQuery. Creation of the date range input fields can be seen in CODE 9.



```
// Start date input field
self.input_start_date = $('<input class="form-control" type="text" value="" />')
// End date input field
self.input_end_date = $('<input class="form-control" type="text" value="" />')
```

CODE 9. Creation of date range input fields with jQuery

The error log was required to be able to filter based on username as well. I implemented the username filter similarly to date range filter, so I will not go into details. Besides creating filters and an error log table, I included other functionalities to this JavaScript file such as refreshing the logs and buttons for exporting the error logs as PDF, CSV, and Excel. However, the main purpose of this JavaScript file is to create a table for error log entries and create filters for username and date.

During the implementation of the UI side for the error log, I created considerable number of other files. To mention some, I created a tutorial page for first time visitor, which explains all the functionalities for the user. Also, I created md-file which can be used to collect future improvements that arise from other developers' feedback. The main functionalities of the PHP and JavaScript files described in this thesis can be seen in TABLE 1. Implementation of the UI part was unorganized from my side. Developing and jumping back and forth between multiple files was challenging for me. The files and functions seen in TABLE 1 were mostly discussed in this chapter. I implemented multiple other functions and files as well, but they were not included in this thesis. For instance, implementation on how this error log module will be integrated to the main software has not been included for security reasons. Now that the backend and UI side had been implemented, it was time to create some tests. In the next chapter, testing for UI and backend are discussed in detail.

TABLE 1. Main functionalities of files discussed in this chapter.

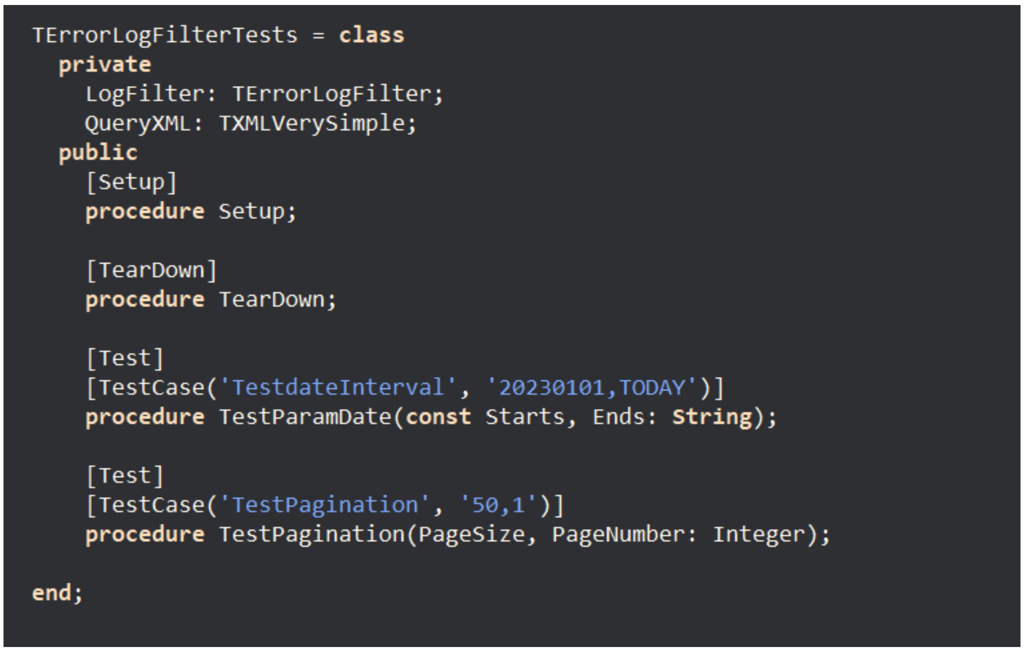
PHP-file 1	PHP-file 2	JavaScript-file
Creating layout for error log page (empty page)	Getting user input	Populating empty error log page with UI elements (table, filters)
	Generating XML string	
	Fetching error log entries from backend	

4.4 Testing the implementation

The commissioned company required that every new feature must be tested and passed before it can be part of the main application. Normally, manual test cases are written by hand by QA developers, but for this project I wrote the manual test cases myself. Unit tests are written by the software developer. This chapter discusses test cases I created for this project, and the focus is on the backend unit tests.

I wrote unit tests for the backend to check that the filter is created according to the given parameters. Unit tests were accomplished with DUnitX framework. The framework is for building and executing tests for Delphi applications. DUnitX is open source and provides Assert-class with multiple functions to help testing (Embarcadero 2019). I started the implementation of unit tests by first creating a class that defines the test cases, called TErrorLogFilterTests. Then I initialized LogFilter and QueryXML objects, these are needed to create filter for testing. QueryXML object was created from TXMLVery-Simple class which is third-party unit that helps handling and manipulating XML data (Dennis1000

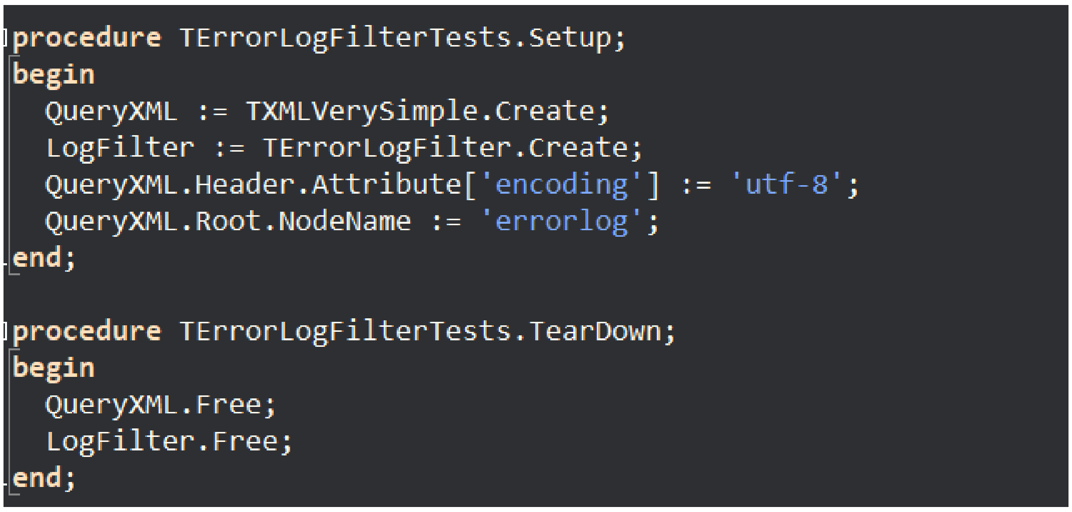
2019). TXMLVerySimple is available under MPL license. Before initializing any tests, I created Setup-procedure that would be run before each test case, this allowed me to set initial conditions for each test. Similarly, I also created TearDown-procedure that would be run after each test case, so necessary objects would be deallocated to prevent memory leaks. At the end, I created test cases for testing the date range and pagination in filter. In DUnitX framework, test cases are marked with TestCase-attribute which includes name of the test case and potential values for the test case. The procedure that handles the actual testing is listed right below TestCase-attribute declaration and it takes the values from TestCase-attribute as parameters for the test. Required procedures and tests have been created up to this point, but the implementation is missing. Next, I will describe the implementation for these class procedures. The implementation of TErrorLogFilterTests class can be seen in CODE 10.



```
TErrorLogFilterTests = class
  private
    LogFilter: TErrorLogFilter;
    QueryXML: TXMLVerySimple;
  public
    [Setup]
    procedure Setup;

    [TearDown]
    procedure TearDown;

    [Test]
    [TestCase('TestdateInterval', '20230101,TODAY')]
    procedure TestParamDate(const Starts, Ends: String);

    [Test]
    [TestCase('TestPagination', '50,1')]
    procedure TestPagination(PageSize, PageNumber: Integer);

end;
```

CODE 10. Implementation of TErrorLogFilterTests class

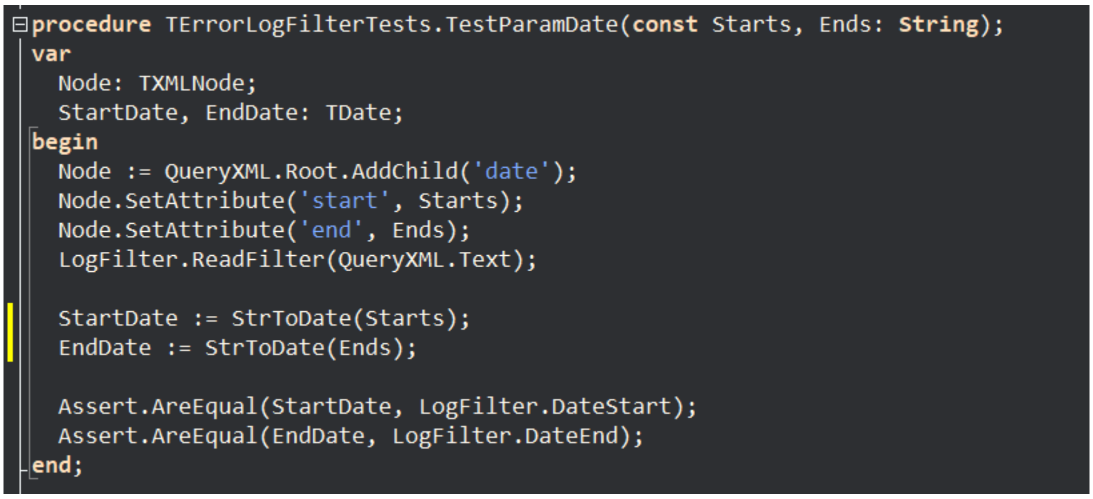
Implementation of Setup-procedure was straightforward since it was the starting point for test cases. All I had to do was to create instances of the required classes for the tests. In this case, I had to create instances of TErrorLogFilter and TXMLVerySimple classes. Now every time a new test case is executed, instances are created. The TearDown procedure on the other hand does the cleanup job after each test case has been executed by freeing the instances. Implementation for Setup and TearDown procedures can be seen in CODE 11.



```
procedure TErrorLogFilterTests.Setup;
begin
  QueryXML := TXMLVerySimple.Create;
  LogFilter := TErrorLogFilter.Create;
  QueryXML.Header.Attribute['encoding'] := 'utf-8';
  QueryXML.Root.NodeName := 'errorlog';
end;

procedure TErrorLogFilterTests.TearDown;
begin
  QueryXML.Free;
  LogFilter.Free;
end;
```

CODE 11. Implementation of Setup and TearDown procedures

The implementation of test cases for date range and pagination were quite similar. Date range was tested in a procedure called TestParamDate. This procedure takes start and end dates as parameters and creates log filter based on those. At the end filter values are compared with the initial parameter values. Implementation of TestParamDate procedure can be seen in CODE 12.



```
procedure TErrorLogFilterTests.TestParamDate(const Starts, Ends: String);
var
  Node: TXMLNode;
  StartDate, EndDate: TDate;
begin
  Node := QueryXML.Root.AddChild('date');
  Node.SetAttribute('start', Starts);
  Node.SetAttribute('end', Ends);
  LogFilter.ReadFilter(QueryXML.Text);

  StartDate := StrToDate(Starts);
  EndDate := StrToDate(Ends);

  Assert.AreEqual(StartDate, LogFilter.DateStart);
  Assert.AreEqual(EndDate, LogFilter.DateEnd);
end;
```

CODE 12. Implementation of TestParamDate test case.

The test case for testing that date range values passed to the filter are equal to the values after filter creation can be seen in CODE 12. The implementation is simple and does not require many lines of code. This procedure takes start and end date string type values as parameters. XML node “date” is created and attributes for start, and end dates are set based on the parameters the procedure received. At this point is good to refer to CODE 7 which explains the XML string creation. This “date” node is set to be as child node for Root node which I have set to be “errorlog” in Setup procedure. After creation of the XML string, filter is read and created based on that. Parameter values are converted from string to TDate type to compare them with filter values. DUnitX framework provides an Assertion-class with multiple functions which are used at the end to compare values with the function “AreEqual”. If the values are equal, then the test is passed. I will not go into details of the implementation of the pagination test case since it is like TestParamDate test case.

Besides the backend unit tests, I also wrote manual test cases for QA engineers to run. Usually, the manual test cases are written by QA engineers, but in this case, I wanted to write them myself. These manual test cases were used to test frontend functionality and how backend retrieves data. Since the manual test cases are not that technical, but rather steps and expected results when user goes through every functionality, I will not explain them in detail. However, I could mention that while writing manual test cases, I had to write steps to test for someone not familiar with the feature, which appeared to be quite hard and time consuming in the end. Writing the unit tests went quite smoothly, even though I had not used DUnitX framework before, but it was simple to use and did not require too many lines of code. Overall, I faced little to no problems during implementation of the tests for my project. Unit tests and almost all manual tests passed. Due to the working schedule, the UI side manual tests have not been run yet by QA engineers. In the next chapter I will discuss the results and what was achieved.

4.5 Results

The development task consisted of implementation of an error log module that could be attached to the SaaS platform the task commissioned company provides for their customers. Users of the SaaS platform occasionally encounter errors while using the system, and these errors were already stored in relational database. My job was to develop backend functionality to retrieve stored errors and show them in the UI for the administrators. I had received some requirements about the functionalities, like adding filtering based on date range and username. Required tools and programming languages for the job were also given for me by the company. There was no set time limit for this task, so it was my responsibility plan and schedule the whole project. At the beginning of the project, I evaluated the task and decided to split it in two parts, backend and frontend. I started by implementing the backend functionality to retrieve errors, followed by implementation of the UI, and finishing the task by creating manual tests as well as unit tests for backend. At the end I managed to have working error log module which displays errors in the UI. I managed to implement all the required functionalities and even made tests that were not required from me. The functionalities included filters, export options and table for the error log. Administrators of the system can now easily track errors by username or time interval and use this information to inform developers of the company. This new module has not been yet integrated as part of the main application since UI side manual testing has not been completed yet. I am confident that the error log module that I implemented will be part of the main application during 2024 with new major version release for the software. After all, error log helps to detect bugs occurred in the system, enhancing user experience, and making application safer.

Since there was no time limit set for me, I could freely schedule my work. I did the whole process along side with my daily work at Vitec Raisoft Oy. This project helped me to understand the business logic behind the SaaS platform better. I encountered new programming language Delphi, which I had not heard before, but I was able to learn it quite quickly. During the development process I had to use multiple programming languages, version control system, database management tool as well as debugging tools. Backend part of the project I managed to finish completely, and it was also tested by QA engineers. UI side still needs to be tested by QA engineers, so the project is not yet complete. Even though the project is not quite finished, I am happy with the results I managed to achieve. In the next part of this thesis, I will discuss this thesis work and possible improvements in more detail.

5 CONCLUSION AND DISCUSSION

The aim of this thesis was to visualize errors stored in a relational database. Before, the errors users encountered were just stored with no way to access them later through the platform Vitec Raisoft Oy provided. My task consisted of implementation of an error log module that would show errors for administrators in readable format, allowing them to filter errors based on username or time interval. The task was implemented in two parts, backend and frontend. The backend part consisted of the creation of functions that would be used to retrieve data from database as well as filtering the data based on the user input. The backend part was mainly programmed with Delphi and SQL as programming languages. For debugging, Delphi offered internal tools that were utilized, but for database management and debugging, a tool called SQL Server Management Studio (SSMS) was utilized. The front-end part of the task was implemented with common web programming languages, PHP, and JavaScript. Handling the data flow between frontend and backend was developed on PHP side. PHP also handled creation of the empty error log page, which acted as template for UI. Populating the UI was handled on JavaScript side. Popular JavaScript library jQuery was utilized to create HTML elements like buttons and input fields. These elements were then placed on the PHP template. Tabulator library was used to create dynamic table to show error log entries. The work was developed on two separate feature branches, utilizing git version control system. This allowed me to implement features locally and the code base stayed intact. Visual Studio Code was used as programming IDE for frontend part, because of support tools for PHP and JavaScript. The backend part was programmed with Delphi which provides own IDE. At the end I was able to meet the requirements and complete development tasks. The backend part has been tested by QA engineers and tests passed. The front-end part has not been tested yet, but so far, my work is done. If issues arise from testing of the frontend part, I will have to fix them before this new feature can be integrated to rest of the software. However, I am sure the new error log module will be part of the main application with the new version release in 2024. The error log module I created shows errors in readable format, stylistically aligns with rest of the application, allows filtering for the logs, and provides exporting the error log in CSV, PDF, and Excel formats. The UI is clear and visually appealing. The error log table shows only the relevant information such as timestamp, username, and error message. This information is enough for developers of Vitec Raisoft Oy to start investigating the cause of the error.

Logging information and tracking system behavior is crucial for enhancing user experience. By logging user encountered errors, we can track who and when the error occurred. This helps to detect and even prevent bugs from happening. This thesis work was completed so that Vitec Raisoft Oy can utilize error monitoring in their software and identify possible underlying problems in the system. Identifying problems in time can save a significant amount of money.

In my opinion, I managed to complete my task successfully. At the start, I was unsure how programming with Delphi would go since I had never used it before. However, I was able to learn enough Delphi to complete this task. Although the development process is completed on my part, the project is still on going with the testing. This project has been very educational for me. I am more confident with Delphi and OOP practices. This task taught me useful skills like debugging and at least I feel like it made me a better software developer overall. My objectives before starting this project were to be more confident, improve my skills in OOP, learn Delphi, and achieve a better understanding of how modules of the main software are created. It seems to me that I have reached these objectives.

Although my development is done so far, there are always room for improvement as a developer. There might be forthcoming fixes from the manual testing of the UI which I have to take care of later. Some improvements that crossed my mind during the development process are related to testing. At this point I have created manual testing for UI and backend, as well as unit tests for backend. However, it would be good to create unit tests for the UI side as well. With unit tests on the UI side, I as a developer can be sure that my functions are working as intended. Another possible improvement could be to include a module name in the error log where the error occurred for the user. Currently error log is logging error log messages which can be helpful, but in some cases can be very hard to understand since the software is substantial and errors can happen everywhere. Including module name to the error log entry could be significant improvement to the log. The system administrators would be able to see the exact location where the issues happen. This would result in faster and more efficient debugging. These are my few thoughts on how to improve the error log module.

6 REFERENCES

Blischak, J., Davenport, E. & Wilson, G. 2016. *A Quick Introduction to Version Control with Git and GitHub.* Available at: https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1004668. Accessed 17 December 2023.

Candido, J., Aniche, M. & Deursen, A. 2021. *Log-based software monitoring: a systematic mapping study.* Available at: https://peerj.com/articles/cs-489/. Accessed 17 December 2023.

Dennis1000, 2019. *XML.VerySimple.pas.* Available at: https://github.com/Dennis1000/verysim-plexml/blob/master/Source/Xml.VerySimple.pas. Accessed 4 January 2024.

Embarcadero, 2019. *DUnitX Overview.* Available at: https://docwiki.embarcadero.com/RADStu-dio/Alexandria/en/DUnitX_Overview. Accessed 1 January 2024.

Embarcadero, 2023. *Delphi Frequently Asked Questions.* Available at: https://www.embar-cadero.com/products/delphi/faq. Accessed 17 December 2023.

Embarcadero docs, 2014. *TADOQuery.* Available at: https://docwiki.embarcadero.com/Libraries/Alex-andria/en/Data.Win.ADODB.TADOQuery. Accessed 17 December 2023.

GeeksforGeeks, 2023. *Frontend vs Backend.* Available at: https://www.geeksforgeeks.org/frontend-vs-backend/. Accessed 17 December 2023.

Głowacki, P. 2017. *Expert Delphi: Robust and fast cross-platform application development.* 1st edi-tion. Birmingham, Mumbai: Packt Publishing.

IBM. *What is a relational database?* Available at: https://www.ibm.com/topics/relational-databases. Accessed 17 December 2023.

IEEE, 2005. *Building rich web applications with Ajax.* Available at: https://ieeexplore-ieee-org.ezproxy.centria.fi/document/1516047/authors#authors. Accessed 10 February 2024.

jQuery, 2023. *What is jQuery?* Available at: https://jquery.com/. Accessed 17 December 2023.

MDN contributors, 2023. *Introduction.* Available at: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Introduction#what_is_javascript. Accessed 17 December 2023.

Microsoft, 2023. *What is SQL Server Management Studio (SSMS)?* Available at: https://learn.mi-crosoft.com/en-us/sql/ssms/sql-server-management-studio-ssms?view=sql-server-ver16. Accessed 17 December 2023.

Mozilla, 2023. *Getting started with HTML.* Available at: https://developer.mozilla.org/en-US/docs/Learn/HTML/Introduction_to_HTML/Getting_started#anatomy_of_an_html_element. Ac-cessed 17 December 2023.

PHP. *What can PHP do?* Available at: https://www.php.net/manual/en/intro-whatcando.php. Accessed 17 December 2023.

Schmidt, K., Phillips, C. & Chuvakin, A. 2012. *Logging and Log Management: The Authoritative Guide to Understanding the concepts Surrounding Logging and Log Management.* 2nd edition. USA: Newnes.

Tabulator, 2023. *Info.* Available at: https://tabulator.info/. Accessed 17 December 2023.

Vitec Raisoft Oy, 2023. *Yritys.* Available at: https://www.raisoft.com/fi/yritys/. Accessed 17 December 2023.

Visual Studio Code, 2023. *Getting Started.* Available at: https://code.visualstudio.com/docs. Accessed 17 December 2023.

W3Schools, 2023. *Introduction to XML.* Available at: https://www.w3schools.com/xml/xml_whatis.asp. Accessed 10 February 2024.

W3Schools, 2023. *PHP – What is OOP?* Available at: https://www.w3schools.com/php/php_oop_what_is.asp. Accessed 10 February 2024.