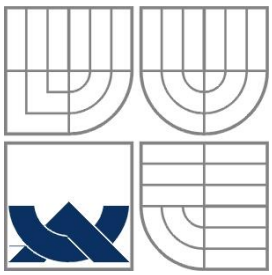
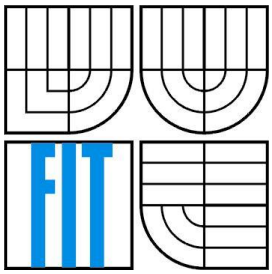


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

INTEGRACE DAT DO CRM SYSTÉMU

DATA INTEGRATION INTO A CRM SYSTEM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

DAVID POŘÍZEK

VEDOUCÍ PRÁCE
SUPERVISOR

Doc. Dr. Ing. DUŠAN KOLÁŘ

BRNO 2016

Abstrakt

Práce se zabývá analýzou firemního CRM systému se zaměřením na integraci externích dat. Na základě výsledků práce je navržena aplikace, která zjednodušuje zpracování dat a zpřístupňuje ji běžným uživatelům z řad zákazníků firmy. Návrh aplikace vychází z existujících omezení a požadavků firmy. Aplikace je následně implementována tak, aby předešla většině chyb, které mohou nastat při přípravě a zpracování zdrojových dat.

Abstract

The thesis analyzes of the company's CRM system and focuses on the external data integration aspects of it. Based on the results, an application is designed. The application simplifies data processing and it enables it to common users, who are customers of the company. Design of the application is influenced by company's restrictions and requirements. Then, the application is implemented to prevent most of the errors, which could occur during the data integration process.

Klíčová slova

CRM systém, integrace externích dat, ASP.NET, C#, JavaScript, informační systém, verifikace dat, párování dat, tabulkový procesor, optimalizace aplikace

Keywords

CRM system, external data integration, ASP.NET, C#, JavaScript, information system, data verification, data mapping, spreadsheet, application optimalization

Citation

Pořízek David: Data integration into a CRM system, bachelor thesis, Brno, FIT VUT in Brno, 2016

Rozšířený abstrakt

Cílem této práce je navrhnout a implementovat aplikaci, která by zjednodušila proces vkládání externích dat do CRM systému firmy. Aplikace by měla být uživatelsky přívětivá natolik, aby ji byli schopni používat i méně zkušení uživatelé. Dále by měla přinést kontrolní a opravné mechanizmy, které by zabránily chybám během integrace dat a napovídali uživateli úpravu tak, aby odpovídala požadovanému formátu. Následně by aplikace měla být schopna zpracovat odpovědi databáze obsahující identifikaci chyby a řádku, na kterém chyba nastala, a zobrazit je uživateli. Chyby, které jsou častější, by aplikace měla eliminovat, případně uživateli navrhnout, jak je opravit.

Pro vývoj aplikace je dále podstatné, aby dokázala pracovat s velkými objemy dat. Vstupem by mohl například být textový soubor obsahující destitisíce řádků. Proto je potřeba celý systém navrhnout a implementovat tak, aby efektivně spravoval paměť.

Práce je logicky strukturovaná do kapitol, ve kterých se postupně věnujeme analýze, technologiím, návrhu, implementaci aplikace a jejímu testování. Následující kapitola popisuje detailně firemní systémy, které se zapojují do procesu integrace dat. Dále se zaměříme na strukturu souborů, které jsou vyžadované na vstupu procesu a popíšeme strukturu souborů, které jsou výsledkem celého procesu.

Ve třetí kapitole je podrobně popsán návrh aplikace. Ta je členěna do dvou podkapitol. V první zdůvodníme výběr technologií a popíšeme jejich úlohu v aplikaci. V následující vysvětlíme jednotlivé technologie a jejich obecný popis.

Samotná implementace je popsána v kapitole čtvrté, která je rozčleněna na podkapitoly podle funkčních prvků použitého návrhového vzoru. Rozebereme zde převážně důležité procesy a algoritmy aplikace, popíšeme jak fungují a zdůvodníme motivaci pro jejich zavedení.

Postup testování je vysvětlen v kapitole páté, která začíná popisem základních metod a způsobů testování. Následně porovnáme původní uživatelský proces s procesem po zavedení naší aplikace a zjistíme, zda naše rozšíření přineslo jeho očekávané zkvalitnění pro uživatele.

V šesté kapitole vyhodnotíme získané výsledky práce a následně je porovnáme s požadavky. Závěrem se zmíníme o možnostech budoucích rošíření této aplikace.

Data Integration into a CRM System

Declaration

Hereby I declare that this bachelor's thesis was prepared as an original author's work under the supervision of Doc. Dr. Ing. Dušan Kolář.
The supplementary information was provided by Mr. Christophe Ratton.
All the relevant information sources, which were used during preparation of this thesis, are properly cited and included in the list of references.

……
David Pořízek
16. May 2016

Acknowledgements

I would like to thank my supervisor, Doc. Dr. Ing. Dušan Kolář, who has helped and supported me by providing constructive criticism, advices, and interesting suggestions during my bachelor thesis writing. I would also like to thank Mr. Christophe Ratton and other employees of IMS Health for this opportunity, their support, and all provided explanations.

Contents

1 Introduction

The aim of this thesis is to design and implement an application, which would make the process of the data integration into customer relationship management (CRM) system easier and faster for IMS Health customers. The motivation for writing this thesis was a chance to design and implement an application, which would simplify the whole data integration process and assist its user in a more advanced way than a usual text editor. It was also an opportunity to cooperate with an international company.

IMS Health is engaged in recording, managing, collecting and selling the data. The data they sell mostly contain details about health care professionals (HCP) and health care organizations (HCO). This information can then be sold to their customers and they would then adjust their selling strategies according to the provided data. Another important part of what IMS Health offers to pharmaceutical companies is a data management. They provide multiple applications, which allow their customers to upload and then manage data about sales, progress of their representatives, success with doctors when promoting a product, etc. This thesis application is focused on hospital sales data, which describe pharmaceutical drugs sales success in hospitals. These data are uploaded by customers and managed by IMS Health.

While designing the application, it is important to keep the design simple and easy to use for all users with limited software experience. Comparing designs of other applications that the company provides, will help when deciding the overall design of this project. It will also help customers to get used to the new application more easily. Another thing to keep in mind is to design the application in a way that it detects and can correct elementary errors made by a user when inputting the data, before they are sent into the database system, thus reducing the traffic and giving customer immediate response.

As for the development of the application, one of the requirements is memory efficiency and good performance optimization, because the application needs to be able to work smoothly with tens of thousands of rows imported from customer's database. Application should accept Microsoft (MS) Excel output format, since it is the most common format that customers use.

The thesis is logically structured into chapters, where we gradually go through technologies that IMS Health uses, analyze their CRM system, design the application, describe the implementation and talk more in depth about some implementation details, finally discuss testing and the conclusion we can draw from the testing. The following chapter describes and analyses CRM system's components in depth. It also explains the whole integration process and all the sub-processes. We will also talk about a structure of the input file required by the system and the output file the system produces as a response. This application's main goal is to help create the input file and process the response.

Design of the application is outlined in the third chapter. It explains reasoning behind all the technologies used and also describes each of the technologies in detail. Then it defines who are the target users, restrictions of the user interface and also important elements of the application.

In the fourth chapter we will specify the implementation. This chapter is divided into subchapters based on the components of the application.

Testing process and protocol is defined in the fifth chapter. Firstly, we will go through and describe basic testing methods. Secondly, we will compare user process flow and how the application improved it. Finally, we review the testing protocol and see how it helped improve the application.

The final chapter contains summarization of the whole thesis and comparison to the requirements. We will also propose possible improvements of the application.

2 Analyzing the CRM System

This chapter is based on information provided by IMS Health [13] and verbal explanation from IMS Health employees. We will describe the IMS Health's CRM system and examine each of its components, their role, data, and dependent systems. In Subchapter 2.1 we will debate what IMS Health provides and who its customers are. It is also important to understand the general idea behind the systems, which we will talk about in Subchapter 2.2. Then we will describe each interacted component in Subchapter 2.3. Finally, we will describe the process of integrating the data into the system in Subchapter 2 and in Subchapter 2.5 we will debate system's deficiencies and suggest possible improvements.

An understanding of data integration principles and flows is crucial for this project, because the goal of the thesis is to simplify the process for all users and make it more user-friendly while not losing any features provided by the current interface.

2.1 IMS Health

IMS Health is a service company which vertically focuses on the pharmaceutical market. It is important to understand, that one of its main products is information. It collects and updates various contacts and other descriptive details about health care professionals (HCP) and health care organizations (HCO) in the pharmaceutical industry. These data are called OneKey data and it is the key product that IMS Health provides to its customers. The company also manages private data of pharmaceutical companies, such as sales data. These types of data are secured and not shared across customers contrasting to OneKey data.

All the above mentioned data can be managed by a CRM system developed by IMS Health. The CRM system is highly configurable to suit customer's needs and can be integrated in customer's system. It provides features to monitor and record activities and results of customer's pharmaceutical representatives when promoting their products. The main requirements on the system are good performance for a mass data integration, continual accessibility, and user-friendly interface for data integration, data management, and data analysis.

Our application will work with the private sales data linked to HCP or HCO. It will use interfaces provided by the CRM system, which are currently very difficult for average users. Our goal is to provide a new and simplified interface for data integration to the end users.

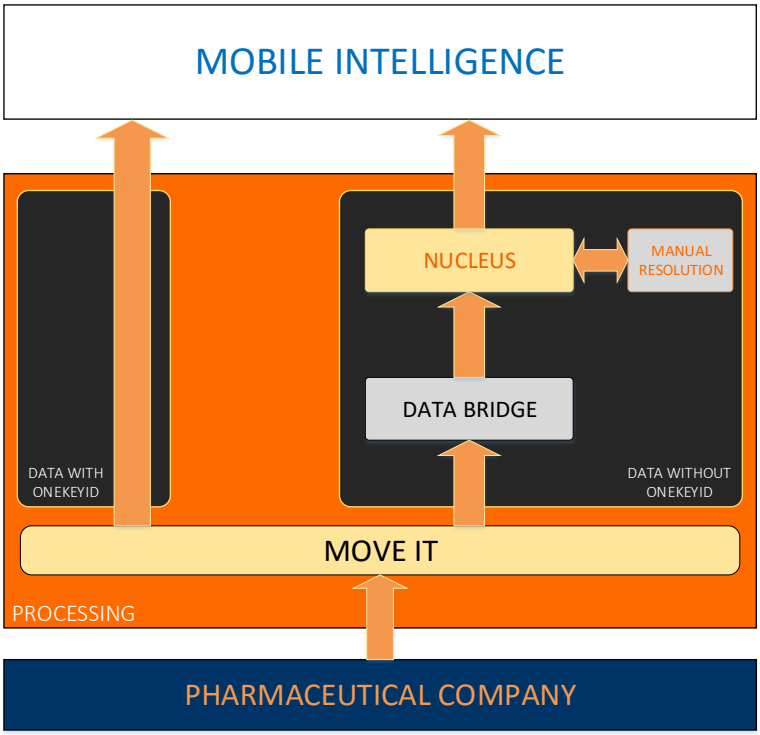
2.2 General System Description

The CRM System from IMS Health provides various reports and statistical analyzes to customers. These outputs can be achieved only if all components communicate together in order to map customer's HCP and HCO identifications on IMS Health's identifications.

One of the features the system provides is a possibility to compare the customer's Hospital Sales data with their pharmaceutical representative's activity stored by the system and see how effective promotion of a certain product was. It is important to allow customers to make future market plans and investments based on their current sales and its development.

In addition, cross-platform availability, including tablets and phones, is crucial to the system, because it is regularly used outside office by pharmaceutical representatives to schedule a meeting with an HCP or HCO and then record the progress and result of meetings. These records are stored in the CRM system and then later used to generate comparisons and statistics for the customer.

Our application will handle a process of external (sales) data integration into the CRM system using an existing interfaces built from DataBridge and Nucleus360 (Nucleus). Simplified mapping process is demonstrated in Picture 1, to make the main communication flow of the CRM system clearer.



Picture 1: Simplified MI identifier mapping process

In this example we use a pharmaceutical company as a data provider. Its Hospital Sales data are uploaded to MOVEit in the correct format. The input data can be of two different types, one with Mobile Intelligence (MI) HCP or HCO identifiers (OneKey Id) and the other one is without the OneKey Ids. Data with correct identifiers are verified and uploaded directly into MI skipping the whole mapping process. Data without MI identifier have to go through the whole process to get the correct identifiers provided by MI.

The mapping process starts with DataBridge separating master data (HCP or HCO details) that are required in order to retrieve a corresponding MI identifier and transactional data. **Transactional data** are more dynamic compared to master data, they contain information about sales, profits, dates, etc. and vary with every integration request. **Master data** on the other hand do not change often and contain only HCP or HCO details. Both data are saved on DataBridge and master data are sent to Nucleus (Subchapter 2.3). Nucleus then processes master data and tries to identify the correct MI identifier. In case Nucleus is unable to identify the correct MI identifier, it is left for administrator to match the MI identifier manually. Once all the master data have been appended with MI identifiers, Nucleus sends them back to DataBridge. DataBridge combines new master data with stored transactional data and uploads them to MOVEit and the process continues the same as with the first type of data.

The CRM system is quite complex and further details about the process of sales data integration can be found in the data flow diagram in Picture A.3.

2.3 System Applications

In this chapter, we will describe in detail each of the components engaged in the mapping process namely Mobile Intelligence, DataBridge, and Nucleus. We will talk about their role, how they transform the data and their interaction with other components.

2.3.1 Mobile Intelligence

This is the core part of the CRM system. Majority of the data are saved in the system. The data are then used by other components, such as Nucleus. It also provides most of the system features to customers such as data analyzes, graph plotting, data comparison, etc.

MI database contains identifiers, which uniquely and internally identify every HCP or HCO that is in the database. Pharmaceutical companies use different identifiers in their own databases. Consequently, IMS Health needs to match both identifiers provided by their customers, pharmaceutical companies, and its own MI identifiers. The purpose is to enable all the features provided by the CRM system regardless the identifiers used in the source data.

Customers upload their sales data to MI database where it is stored and available for generating statistics. They do not access MI directly for security and performance reasons, all the traffic goes through DataBridge, which checks data for possible issues that may occur during the integration process.

MI is available on multiple platforms, because it has to be accessible not only to the administrators who manage the customer's data, but also their representatives who work with an HCP and HCO personally, to allow them to report their meetings in the field. Every meeting or report they create with their application is uploaded into MI database when there is an available Internet connection.

There are two different versions of MI client available, one of them is web interface and the second is an executable application. The first one is targeting managers and administrators, as it provides big variety of graphs and data comparisons that can be made and gives users total control over what will be compared from the general data to a very specific ones describing sales of one product for example. The second one is what representatives use to schedule their meetings, record results of meetings, organize meetings and so on. It is also connected to MI database, allowing representatives to setup meetings and link them with drug from MI database, an HCP or an HCO.

2.3.2 DataBridge

This chapter is based on DataBridge documentation available in "*Nucleus-DataBridge_System v0.pdf*" available on CD Content. The purpose of the DataBridge is to enable integration of external sales and transaction data without customer ID into Mobile Intelligence. One of the roles of DataBridge is a verification of the input data. It also serves as an entry point of the data integration process for users into the whole system and therefore it acts as an interface between users, MI and Nucleus. It is there to ensure the whole integration process successfully finishes and to report any potential causes of a failure back to the user immediately, before the system starts to integrate the

data. It duplicates MI database triggers and integrity rules and verifies the input data using almost identical checks as MI would.

The integration process is divided into several steps that are described in "*Nucleus-DataBridge_System v0.pdf*" available on CD Content.

- Import files from MOVEit.
- Save imported files into stage tables.
- Validation of imported files.
- Send not valid customers (unknown customers) to Nucleus.
- Import feedback from Nucleus to DataBridge.
- Run change request process.
- Export data to MI.

DataBridge periodically checks linked MOVEit storages for new files and downloads any file that meets its format and file naming criteria, thus starting the integration process.

Data are loaded into staging tables with status "Pending". It will be modified during the next steps.

After the import, DataBridge starts with data validation. It is done based on the following criteria:

- Control if customer is new and need to be matched by Nucleus system.
- Control if provided data are correct before it forwards them to MI (check if in provided data have all required fields).

DataBridge performs these data quality controls detailed in "*Nucleus-DataBridge_System v0.pdf*" available on CD Content:

- HCP/HCO: Check if provided record exists in `dict_customer` table (this table stores earlier matched MI and customer's identifiers) and if we have all required identifiers (like Nucleus id)
- Products: Check if provided product exists on MI side.

Records that have no all required field are rejected and inserted to error report. Then DataBridge uploads the error log file to MOVEit as a response file, containing the error description and the row where the error occurred. Users then have to manually look through the error file and find and modify rows in the original file and re-upload it. This error log is analyzed by our application.

If the input data pass all the verifications, DataBridge sends the data either directly to MI for data integration, or in case some rows are missing MI identifiers, it is sent to Nucleus to append these rows with matching MI identifiers. All these HCP/HCO are submitted to Nucleus. Then Nucleus returns identifiers as the response to DataBridge. All sent customers have their status updated to "Sent to Nucleus".

In the next step DataBridge imports information about identifier for customers that are required to specify. All HCP/HCO validated by Nucleus are completed by transactional data and then sent to MI. In addition, DataBridge stores this matches in its database. Therefore, consequent match requests of the same customers are done without any delay and communication with Nucleus.

When some HCP/HCO cannot be matched efficiently by Nucleus Match Engine, DataBridge generates Change Requests record and submit it to MI. Then MI creates them in the master database. DataBridge identifies concerned customers with a specific code "Change Request Sent". With a next scheduled process, DataBridge checks whether appropriate HCP/HCO exists and submits them for integration.

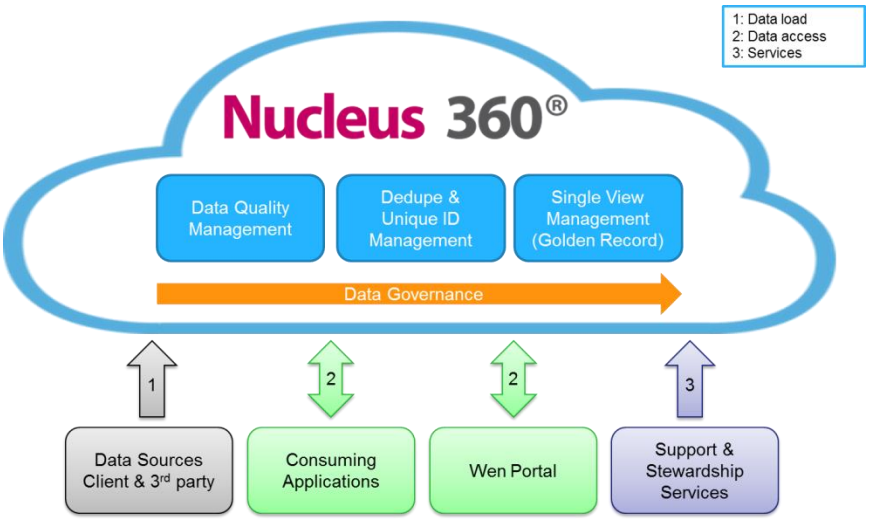
2.3.3 Nucleus360

Nucleus is the software product designed to capture, centralize, and help manage customer information. With Nucleus, pharma companies can create an interactive customer master repository that is accessible across the enterprise and is governed to maintain ongoing customer data quality.

Nucleus can interact with existing and new data sources and systems (operational, marketing, CRM, etc.) to combine data quickly and efficiently. This interactive master repository becomes a resource for accurate and current customer information that can serve many purposes.

All data entering the system are cleansed and validated according to business rules determined by the clients before being applied to the repository. The system then matches customer records to detect duplicate entries and issues merge notifications to contributing systems. In the process, Nucleus creates a single record from data linked across multiple sources using a variety of configuration-driven attribute survival rules.

The interactive master repository can then be used as the source of quality customer data for CRM, and internal systems; data warehouses and external vendors; alignment and compliance programs (sample management and aggregate spend); customer targeting activities; and as the foundation for advanced analytics.



Picture 2: Solution components [13]

The MDM (Master Data Management) repository together with available matching and standardization rules is implemented as a component of the CRM system where OneKey database delivers HCP and HCO (Data Source) and Mobile Intelligence is one of the Consuming Applications.

For purpose of this thesis, we analyzed an existing implementation where Nucleus interacts with DataBridge and where Nucleus performs the process of mapping rows with an HCP or an HCO details provided in the input file with sales data to the MI identifiers. Nucleus periodically downloads changes from MI database and retrieves MI identifiers for mapping purposes. The online access is not necessary here and scheduled access is sufficient because the HCP and the HCO details do not change often and relieves workload from MI system.

Data flow process was also described by data flow diagram in Picture A.3. There are many interactions between various systems and the main principles are explained in Chapter 2.2. Nucleus

stands in the middle of data integration and matches data from different data sources, such as MI and customer's data.

Nucleus uses matching score to determine how accurate its match was and when the score is too low it identifies the matching as too inaccurate. In this case, Nucleus is unable to properly match a row with the MI identifier. The mapping process skips these rows while finishing the mapping process for the rest of the records. Rows that could not be identified are sent to the administrator for manual resolution.

There are several configurable features to answer various needs of matching and standardization in Nucleus. A matching function responsible for matching customer records, within a data source and across data sources, ensuring two or more records are recognized as the same customer or recognized as a unique customer. The match function is also responsible for managing customer split and merge scenarios. A standardization function responsible for ensuring customer data, across data sources, is structured to ensure matching and resolution is efficient and accurate.

In case the MI identifier is missing and no matches are possible, Nucleus informs DataBridge that a new MI identifier needs to be created. This is done by returning an empty MI identifier field in the response file back to DataBridge.

Nucleus supports a number of standard data interfaces for both importing and exporting data using point-to-point communication channels – secure FTP, through message-brokering and secure web services architectures. The web services offer many possibilities to read, submit updates, enabling real-time searches of the database using wild cards, etc. Within our implementation, Nucleus and DataBridge are connected through MOVEit which offers secured communication. Another connection between Nucleus and MI is based on web services.

Integral part of Nucleus is also a portal used by administrators, data stewards, and customers. It is a web-based application that enables users to view and maintain customer information within the Nucleus repository. However, this is not utilized by our application and therefore we will not detail it further.

2.4 Data Integration Process

This section will describe file formats used in the data integration process. Formats from Subchapters 2.4.1 and 2.4.3 are important for this thesis, because the application will have to process them and display them to a user. We will also talk about formats, which are used temporarily during the process and are hidden to a regular user.

2.4.1 Input File Format

The input file format has very specific requirements and is described in a Data Interface Document (DID) and it is called an Interface Data Load (IDL) file. There are many different formats of the IDL file, because every customer has his own modified CRM system. The IDL files have to be modified in order to allow data integration using the DataBridge interface.

Despite the fact that there are many different IDL file formats, we can define an IDL file format generally. An IDL file is a text file representing data in tabular form with an *idl* suffix similar to Comma Separated Values (CSV) format. It does not use comma as a separator, instead it uses vertical bar character. An IDL file contains lines split into columns by a vertical bar character, where each line represents a row in the tabular form. The line is terminated by Microsoft Windows

end of the line (EOL) carriage return (CR) and line feed (LF) of Linux EOL LF. Each row is divided into columns by the separator. Columns that are optional can be left empty, but it is still required by the format specification that all the rows are split into an exact number of columns. Also the IDL file has to be archived in a *zip* archive, otherwise it will not be accepted.

This thesis focuses on Hospital Sales IDL file format, because the data integration process flow is unique to this IDL file format compared to any other. Reasons why it is unique will be discussed in Subchapter 2.5. An example of IDL file is in Algorithm 1.

```
20160216|I|STAT|SNIASDFR_Hospital_Sales_20160216000000_01.idl|STAT|FR13904
8|ORG|UNK|||POLYCLINIQUE DE FURIANI||||||Y||||||||||||1|ROUTE NATIONALE
193|||FURIANI||20600|FR|Y|||||||||||||||||||||||||FR139048PRO_10102016-
01-01|PRO_1010|2016-01-01|6|149||
```

Algorithm 1: Example of one row in DID file format

The row has exactly 68 columns and this number is required to be the same for every row, as long as we are using Hospital Sales format. If the number of rows is different, the file will be marked as invalid and will not be used in the data integration process. Definition of each row is specified in the documentation provided by IMS Health and can be viewed in the CD Content.

2.4.2 Master Data File Format

DataBridge has to separate master data and transactional data from the input data before sending them to Nucleus for mapping. Because there is no need to send the whole row to Nucleus, DataBridge only sends details, which are needed for the matching purposes. Further details are explained in Chapter 2.3.3.

DataBridge creates new separate file which contains required details and sends it to Nucleus. After the mapping process is finished DataBridge receives a similar file, but with one newly appended field that contains the matched MI identifier.

There are two different results of the mapping process that can happen. In the first case, the mapping process was successful and all the rows have appended MI identifiers. In this case, DataBridge merges the original file and the one with MI identifiers and uploads it to MOVEit where it will be integrated into MI on a schedule. In the second case, the MI identifier could not be found and has to be created in MI. Nucleus informs DataBridge, that it could not find an identifier. Based on the information, DataBridge creates a change request file that MI recognizes, uploads it to MOVEit and pauses the integration process until the request has been processed by MI. Once MI added changes, DataBridge accesses MI directly on a schedule and appends the newly added identifiers to its file. Then DataBridge merges the original file and the one with MI identifiers and uploads it to MOVEit where it will be integrated into MI on a schedule.

2.4.3 DataBridge Response Log

The response log file provides a way for DataBridge to communicate with a user in case of a failure or a success during the integration process. The log format is very similar to the DID file format, except it has 3 more columns and uses “err” suffix. An example of DataBridge response is in Algorithm 2.

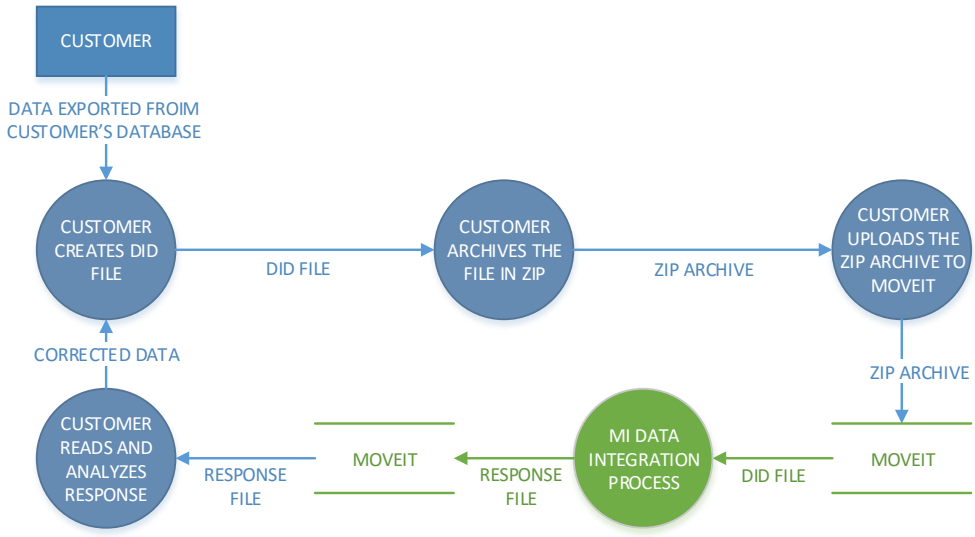
```
Main_DataBridge_process|Field count is different. Count of field in data
file is 69.Count of fields in the layout defined in task is
68|20150828|I|STAT|SNIASDFR_Hospital_Sales_20150828000000_01.idl|STAT|FR13
9071|ORG|UNK|||POLYCLINIQUE FRANCHEVILLE||||||Y||||||||||||1|34 BD DE
VESONE|||PERIGUEUX
CEDEX||24004|FR|Y||||||||||||||||||||||||||FR139071PRO_560 Camera
System2015-05-01|PRO_560 Camera System|2015-05-01|-
1|0|||SNIASDFR_Hospital_Sales_20150828000000_01.idl
```

Algorithm 2: Example of a row in the response file

In the example we can see that there are two new fields in the beginning of the row and 1 new field at the end of the row. The first prepended field describes the process which detected the issue. The second prepended field is the error information. Most of the errors returned are Oracle database errors, because MI uses an Oracle database. Therefore, DataBridge verifies the rows against similar database rules. In the example in Algorithm 2 the user uploaded IDL file with wrong number of fields allowed in a row. Such violations are caught before the actual verification of database rules and so it is one of the exceptions when the error is not from the Oracle database. The source file name that this response was generated for is specified in the last added field.

2.5 System Deficiencies

The main issue with the current CRM system is that it was not designed for manual integration of the external data directly by the pharmaceutical end user (IMS Health customer). When the system was designed, the initial idea was that the sales data integration would be done by IMS Health support team. Therefore, the data integration interface requires technical skills and provides no graphical user interface (GUI). Consequently, the whole process is not very user friendly, as we can see in Picture 3.



Picture 3: Data integration process from customer's perspective

As we can see in the Picture 3, customer has to do the whole process of the DID file creation manually. This is not only inconvenient but also inevitably increases chances of making an error in one of the actions. It is also very impractical for the customer to read the response file and match the specific errors to rows in the original file manually.

Currently, there is a workaround for IDL creation available for the customers. It is a Microsoft Excel file which provides an option for users to create IDL files. It has many limitations, because it is still only an interactive file and is not very extensible.

The inconvenience of the current interface is the main motivation for this thesis. It should provide a user-friendly graphical interface which would eliminate the requirement of doing most of the actions manually.

3 Application design

IMS Health uses many different technologies to provide their customers with as much comfort as possible while using their products and services. Most common technology is ASP.NET, which is used to implement the majority of the applications the company offers. ASP.NET is part of .NET Framework which we will describe more in this chapter. Another important technology is the data transfer service MOVEit [12] by Ipswitch. We will talk about MOVEit role in the company system in Chapter 2, while in this chapter we will focus more on the general description and basic features.

3.1 Technology analyzis

.NET Framework and ASP.NET were chosen as the frameworks for the implementation of the application. Main motivation for the choice was the fact that IMS Health already uses the framework for its own services and applications. This will make the implementation into the IMS Health system easier. Also using the same technology as the customers are used to, might make it be easier for the customers to transition to a new application. The framework also provides many advantages for the application development. It allows us to re-use pieces of code already implemented by the framework instead of having to write it ourselves. For example, we can implement authentication interface and provide our application with basic identity, user groups and membership authentication.

JavaScript is used to offer users website with dynamic content. It is possible to create excel-like spreadsheet inside our web application. Furthermore, it allows us to display the current progress of requests which might take longer to process and so giving the user immediate response whether his request is being processed or not.

jQuery simplifies and standardizes the usage of JavaScript on web pages. It helps us to implement most of our functions with cross-browser support without the need to check browser versions manually.

Chart.js offers simple to use and nice looking charting functionalities. It generates charts for users in the account statistics.

Cascading Style Sheets (CSS) are used to style the table elements on a web page to make them look similar to Microsoft Excel spreadsheet.

Bootstrap provides styling to the whole web application and most of its basic HTML elements. It also makes the design responsive for variety of resolutions, even tablets and phones.

Regular Expressions are crucial for this application in terms of database response identification and correction. They provide easily modifiable string matching, which allows us to save them in database to be later modified or added by administrators, without the need to rebuild the application.

MOVEit is a secure file storage. Its use is required by the current data integration process. Therefore, this application will have to implement access to it as well.

3.1.1 .NET Framework

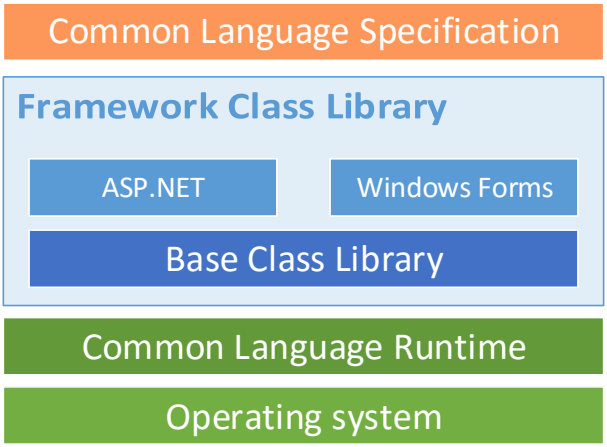
.NET Framework is a software development framework developed and maintained by Microsoft. It provides a controlled programming environment which can be used for programming, installing,

and executing software. The framework runs primarily on Microsoft Windows, even though there are similar implementations cross-platform which are implemented above .NET Core that has been released as open-source and thus share core features, for both the runtime and framework libraries [3].

.NET Framework is designed to provide the following advantages to developers [2]:

- **Language independence** – .NET defines the Common Language Specification (CLS) which makes any language to work under the framework if the specification is followed.
- **Common runtime** – Common Runtime Engine, also known as Common Language Runtime (CLR), provides a consistent object-oriented programming environment with services such as exception handling, security, and memory usage.
- **Language interoperability** – The Common Language Infrastructure allows applications written with language that implements CLI to be run in a common runtime environment instead of a language-specific one. Additionally, two applications can share access to base classes, functions, and data structures, even if they are written in different languages.
- **Base Class Library** (BCL) – As the name suggests it is a base library, meaning it contains basic, fundamental data types, commonly used functions, collections, security attributes, etc.
- **Security** – Applications developed in .NET are based on a common security model

The Framework consists primarily of the CLR and .NET Framework Class Library (FCL). There are also many other components, which are defined above CLR and FCL and are part of the framework such as ASP.NET, LINQ, Entity Framework, etc. We will talk about each of the components in this chapter, but first we should examine the architecture of .NET and see what layers it is split into.



Picture 4: .NET Framework simplified architecture

As we can see, .NET Framework is implemented above operating system, primarily Microsoft Windows and is divided into layers, where each layer provides certain service or functionality. The first and lowest layer is the CLR which provides the basic infrastructure and is a core of the whole platform.

The BCL is part of the FCL and is built above the CLR and together they provide basic and advanced structures for the framework. The FCL also provides other libraries such as ADO.NET,

LINQ, ASP.NET, etc. which help developers shape their final application and provide advanced functionalities to it.

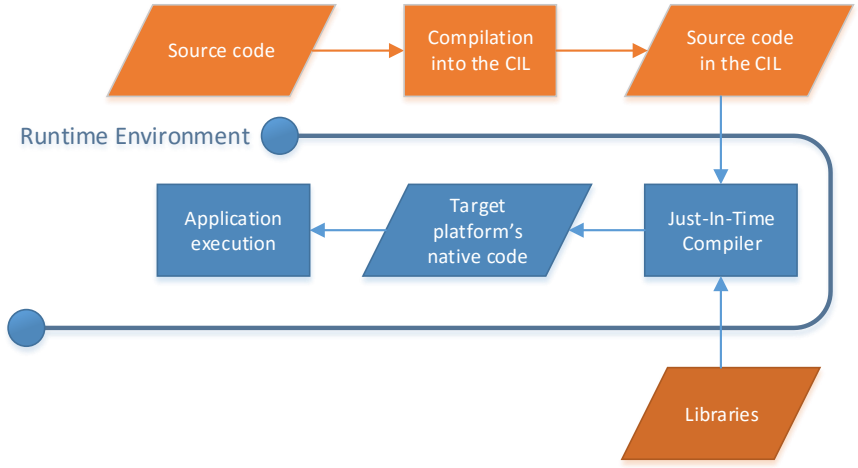
Final layer is common language specification which defines standards that developers need to follow if they want their application to use .NET. Any language can be used as long as the CLS is followed, but the most commonly used languages are Visual Basic, C#, F# and C++.

.NET Framework is still under development and is currently in version 4.6 [1].

Common Language Runtime

The Common Language Runtime (CLR) is a term that is comparable to the term virtual desktop. It is a virtual environment for which the applications are compiled and they are also ran there. This approach ensures safe execution of the code, including code created by unknown or semi-trusted third party applications. It also provides consistency for developers even across different types of applications and manages memory. The CLR provides many services such as memory management, thread execution, code execution, code safety verification, compilation, etc. which are inherent for managed code [4].

Even though the CLR is designed to support developers in creating new applications, it also provides interoperability between managed and unmanaged code, allowing developers to continue to use necessary COM components and DLLs.



Picture 5: .NET code compilation and execution flow

The basic flow of .NET code compilation starts with the original source being compiled into Common Intermediate Language (CIL), then the compiler compiles the code with other required dependencies into a native code of the platform. Once finished, the compiled application is stored in assembly cache for a specific amount of time and then executed. This way, there is no need for recompilation and consequent executions are significantly faster. This is one of the advantages of Just-In-Time (JIT) compilation [5].

ASP.NET

ASP.NET is part of FCL and it was initially released in .NET Framework 1.0. As of March 28, 2012 Microsoft has placed ASP.NET under open-source license. ASP.NET is a server-side web application framework designed for web development and the creation of dynamic web pages, web

applications and web services. It is implemented above CLR, allowing developers to use any supported .NET language when developing an ASP.NET application.

ASP.NET offers three frameworks for creating web applications: ASP.NET Web Forms, ASP.NET MVC and ASP.NET Web Pages. Each of these frameworks is mature and allows us to develop complete web applications with them [6].

ASP.NET Web Forms

The Web Forms framework offers WYSIWYG drag-and-drop designer, making it popular choice for developers, who are looking for rapid application development (RAD) environment. It is a good choice for developers who prefer declarative and control-based programming, such as Microsoft Window Forms (WinForms) and Windows Presentation Foundation (WPF).

Web Forms allows developers to program events similar to a client application in WinForms or WPF. It also automatically preserves states between HTTP requests, which are helpful for programmers who are not used to creating stateless web applications. Web Forms provides rich assortment of a data access and a data display, which helps to create whole websites without having good knowledge of HTML.

ASP.NET MVC

ASP.NET MVC encourages separating the logic layer of the application from its presentation layer. By dividing the application into the model (M), views (V) and controllers (C), complex MVC applications are easy to manage and allow teams of developers to work separately on each of the parts.

In MVC application, developers work directly with HTML markup and are given more freedom when modifying the website compared to Web Forms. It allows us to have a complete control over what the application is doing.

Razor Inline Syntax

We will describe Razor inline syntax (Razor) under MVC, despite the fact that it can be used separately. It is usually associated with MVC, since it was first released as part of the framework.

Razor's markup is primarily designed to work with C# code, as it is most commonly used language to create ASP.NET web pages [22]. It provides all the power of ASP.NET, but uses a simplified syntax. The server executes the Razor code first, before it sends the HTML page to the client. It is possible to execute commands, which would not be possible in HTML web page, for example accessing a database. When browser receives the web page, it is no different than static HTML content [21].

ASP.NET Web Pages

The Web Pages framework development is in some ways comparable to PHP development. HTML pages are created and then appended with server-based code to dynamically control how the page is rendered. It is targeted at developers who are experienced with HTML, but might not have broad programming experience. Thanks to its simple approach, we can generally open up a source file and think of the logic as executing top-to-bottom as we would with PHP for example.

Web Pages are the easiest way to get into ASP.NET development and provide plenty of helpers which implement actions that would be tedious or complex to code from scratch and can be accessed through few lines of code.

Language Integrated Query

The Language Integrated Query (LINQ) is a .NET Framework component that adds database-like querying capabilities to .NET languages. It extends the languages by adding query expressions, similar to SQL statements, and can be used to extract data from arrays, enumerable classes, XML documents, relational databases and third-party data sources [8].

LINQ also defines set of rules which allow compiler to understand query expressions written in a fluent code style and compile them into expressions using these method names, lambda expressions and anonymous types. This provides the ability to write a more readable code [7].

An example comparing fluent code and progressively constructed code:

```
string[] names = { "Tom", "Dick", "Harry", "May", "Jay" };

var final = names
            .Where((string name) => name.Contains("a"))
            .OrderBy((string name) => name.Length)
            .Select((string name) => name.ToUpper());
```

Algorithm 3: Fluent syntax example [9]

```
string[] names = { "Tom", "Dick", "Harry", "May", "Jay" };

var Filtered = names.Where((string name) => name.Contains("a"));
var Sorted = Filtered.OrderBy((string name) => name.Length);
var Transformed = Sorted.Select((string name) => name.ToUpper());
```

Algorithm 4: Progressive syntax example [9]

In the examples from Algorithm 3 and Algorithm 4, we can see that the Algorithm 3 is much easier to understand and it allows developer to focus primarily on writing the query.

There are many LINQ providers which implement interfaces for a particular data store. The most common ones are LINQ to Objects, LINQ to XML, LINQ to SQL and LINQ to DataSets. While using some of the providers, it can be fairly challenging to create a well optimized query, because of sub-optimal LINQ implementation patterns.

Entity Framework

Entity Framework (EF) is an object-relational mapping (ORM) framework for ADO.NET. It used to be part of .NET Framework, but it was separated from it with a release of version 6. It enables developers to work with the relational data as domain-specific objects, eliminating the need for most of the data access code that is usually required. Developers can simply query domain-specific objects using LINQ, then retrieve data and manipulate them as needed and then, if required, actualize database with them. [11]

There are three different approaches based on what is available when developing an application using the EF [10].

Database First

The database first approach is useful in cases when a database has been designed by database administrators (DBAs) or there is already an existing database in place that we use by our application. Entity framework then generates Plain old CLR objects (POCO) entities by itself. To modify entities in the application, it is necessary to modify the original database and then update the existing model from database using EF features.

Picture 6: Generate data access classes from an existing database (Database First)

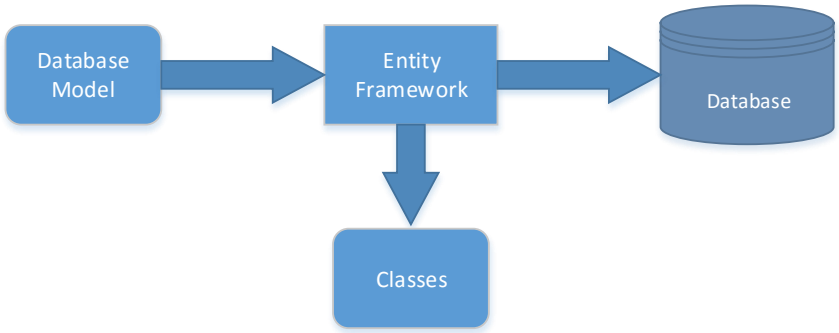
Model First

This approach is convenient for developers who prefer using a graphical designer instead of writing the code or already have a designed model available. It does not provide as much control as other approaches, but enables fast and simple database and POCO entities generation.

Picture 7: Generate database tables from defined model classes and context (Code First)

Code First

The code first approach provides full control over the code. Since developers write most of the code, there is no automatically generated code which might be hard to modify. Since EF 4.3 code first approach allows developers to update database schema, when the mode changes without losing any existing data or other database objects.



Picture 8: Generate classes and database tables from model design (Model First)

3.1.2 JavaScript

JavaScript is an interpreted, prototype-based, dynamic scripting language with first-class functions. It is most well-known as a scripting language for web pages. JavaScript is supported in all major web browsers, but its implementation is not always consistent. Therefore, we have to always look out for compatibility issues when developing a JavaScript application. [15]

3.1.3 jQuery

jQuery is a JavaScript library designed to simplify client-side scripting of HTML. It is under MIT license[1] and publicly available for download[2]. It simplifies HTML Document Object Model (DOM) traversal and manipulation, event creation and handling, animations, and asynchronous JavaScript and XML (AJAX) requests. It basically provides prototypes of DOM elements with added functionalities which are supported across a multitude of browsers. [17]

It is important to note, that despite jQuery being simpler to use, it may not always be the most performance efficient solution. The performance of Algorithm 5 and Algorithm 6 were tested with 200 000 elements inside the `<body>` element. The Plain JavaScript traversed the document in 13-15 milliseconds and jQuery took 164-168 milliseconds. That is more than ten times difference. The performance of jQuery can be significantly improved if we use a query selector '`* body`'. The plain JavaScript traverses the HTML document in the same time 13-15 milliseconds, but jQuery now takes only 28-32 milliseconds. As we can see, the plain JavaScript is still faster, but the performance difference is negligible. This application will usually work with similar amount of data, so keeping performance in mind is crucial when implementing this solution.

```
var items = $('*');
 for (var i = 0; i < items.length; i++)
    var $this = items[i];
```

Algorithm 5: jQuery iteration through all DOM elements

```
var items = document.getElementsByTagName("*");
for (var i = 0; i < items.length; i++)
    var $this = items[i];
```

Algorithm 6: Vanilla JavaScript iteration through all DOM elements

3.1.4 Chart.js

Chart.js is a JavaScript framework. It provides a variety of different types of charts. It is built to be responsive and to have a good rendering performance across all modern browsers. It is available under MIT license and be downloaded from the official web site[3].

[1] https://opensource.org/licenses/MIT
[2] https://jquery.com/
[3] http://www.chartjs.org/

3.1.5 Cascading Style Sheets

CSS define the way HTML elements are displayed on a web site. It was primarily designed to separate the document content from document presentation. This separation provides more flexibility and control over the document's presentation and allows multiple HTML documents to share the same visual style. [16]

CSS specifications are maintained by World Wide Web Consortium[1] (W3C).

3.1.6 Bootstrap

Bootstrap is an HTML, JavaScript and CSS framework developed by Twitter under MIT license and is available for download for free[2]. It provides templates for typography, forms, buttons, navigation, and other interface components to HTML web sites which use it. It also offers optional JavaScript extensions. The main goal of the framework is to deliver a simple start-up for creating clean looking and dynamic web sites to developers. [18]

3.1.7 Regular Expressions

Regular expression (RegEx) is a sequence of characters that define a search pattern. It is commonly used in pattern matching with strings. Regular expressions were introduced by mathematician Stephen Cole Kleene when he described regular languages using his mathematical notation called regular sets. The modern use of regular expressions is fundamentally different to the original one and no longer describes regular language, but far exceeds it. [14]

3.1.8 MOVEit Transfer

The following chapter is paraphrased from [12].

MOVEit Transfer (MOVEit) is a secure file transfer server for important or sensitive data transfers developed by Ipswitch. It provides good visibility and control over file activity, allowing its users to manage how sensitive information is sent and who is eligible to receive such information, while monitoring all data transactions.

MOVEit also provides centrally managed user authentication through integration into already existing systems. Its flexibility allows it to be integrated into data loss prevention (DLP) and antivirus systems, Identity Provider systems (IdP) and Active Directory (AD) or Lightweight Directory Access Protocol (LDAP).

Clients for MOVEit are available on majority of platforms and systems including: mobile devices, web browsers and even Microsoft Outlook client. Consistent environment is ensured across all clients.

Security of transfers is ensured with Federation Information Processing Standard (FIPS) 140-2 validated cryptography, receiver authentication, and delivery confirmation. MOVEit also allows administrators to enforce user, system, and file security policies.

MOVEit supports encrypted transfer over FTP, FTPS, SFTP, HTTP, HTTPS, SSL and SSH protocols allowing developers to implement protocol which would be the most suitable for their

[1] https://www.w3.org/
[2] http://getbootstrap.com/

network solution. The protocols are only used to establish connection and ensure data transfer. The encryption is provided by MOVEit applications.

Accessing MOVEit

MOVEit provides a solution for developers in terms of library which provides Component Object Model (COM) interface to any .NET powered application. The full source code of the library is provided to customers and is used to implement an example application of a remote management console for a MOVEit storage.

It is also possible to implement the MOVEit API from scratch using one of the supported protocols and following implementation specifications which are described in MOVEit documentation. This option is useful in case an application is unable to take advantage of the provided MOVEit library.

3.2 Design Requirements

Target Users

First of all, it is important to define who are the target users for this application. IMS Health customers are usually companies from the pharmaceutical industry. Majority of the users is not very skilled in the information technology (IT) and have difficulties with using the current interface that IMS Health provides. Based on that, it is important to implement the application user-friendly and simple to use.

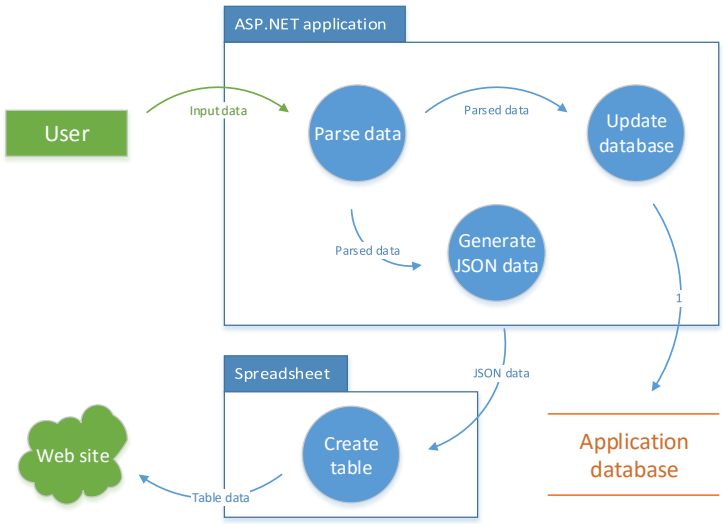
Existing restrictions

It is also important to establish what options we have available from IMS Health in terms of technologies. As we mentioned earlier, IMS Health provides many different services to their customers. Majority of them are implemented as web services and applications. Main reason for this approach is to provide cloud-based services and also the security of the CRM system. IMS Health does not allow an access from an external network to their internal databases. It is possible to access them only from their internal network or in specific cases using database replication. Since web services and applications can run on a server which is inside the IMS Health network, this security restriction is not an issue and still allows IMS Health to provide their services. This leads to an advantage for the customers, which is simplicity and straightforwardness of the usage of applications and services. There is no installation process and prerequisites are managed by server.

Application Structure

The application consists of 2 core components. The first is the spreadsheet, which deals with data editing logic and provides Microsoft Excel-like behavior. The second one is the ASP.NET application. It parses the input data to Javascript object notation (JSON) format for the spreadsheet, manages user accounts and sessions, creates DID file and uploads it to MOVEit, and processes the database response log file.

Picture 9 shows basic data flow of importing data into the application and describes how the components communicate together during the process:



Picture 9: Application data flow when importing data

Spreadsheet

It is implemented separately to the ASP.NET application and generates hypertext markup language (HTML) table element on the web page, appends dropdown headers with column verification to the table element, offers basic spreadsheet functionalities, and displays database response to the user.

Server Application

The application provides two interfaces. The first interface defines file processing, offering simple extensibility in terms of file formats. The second one defines file storage access, which means it is possible to implement the application into an environment without MOVEit, if the interface is implemented.

The application is implemented using MVC and code-first approach, and it is written in C# programming language. The MVC structure of this application is explained in detail in Picture A.1.

Part of the application is also a database, which stores sessions and session data, earlier data integration requests, and user accounts. It is described by entity relationship (ER) diagram in Picture A.2.

4 Application Implementation

We have already analyzed the full structure of the ASP.NET application, so we will only describe the important methods. In this chapter we will talk more in depth about the rest of the technologies, frameworks, approaches, and application program interfaces (APIs) used in the implementation. We will also describe interfaces, objects, and important classes and functions in detail.

4.1 ASP.NET

Structure of the application is shown in the class diagram in Picture A.1. In this chapter, we will describe in detail some of the methods important for the data integration process.

Input Data

The application accepts three input file formats: CSV and Microsoft Excel spreadsheet file (XLS) or Microsoft Excel open extensible markup language (XML) spreadsheet file (XLSX). Interface for all these file types is already implemented. In case there is an empty cell in the input data, it is stored as an empty string.

The input file is then parsed and the data are stored in an internal object, which provides a simple export of the data in a JSON format. The parsed data are also saved into a database and linked with the current user session.

IDL File Creation Process

The correct structure of an IDL file was already explained in Chapter 2.4.1. The IDL file format which the application uses to create the final file is stored in its database. It is a string, which is structured in a similar way like the final IDL file and an example of the string is shown in Algorithm 7. The application constructs each row of the final IDL file separately in `CreateRow(headers, row, format, fileName)` function. The output of the function is a string, which represents one row in the final IDL file. It uses `_X`, where `X` is a column identifier (`HeaderFormatId` in ER diagram in Picture A.2), to specify which cell from the source data row should be put in its place. It replaces all header identifiers this way, appending the output string with user's source data. Furthermore, there are many cells, which need to be either left blank or filled with default values. Every cell text from the format string that is not recognized as a header identifier, except `CDATE` and `FILENAME`, is appended to the string without modification. Finally, there are keywords which are replaced with generated data. `CDATA` is replaced with the current date in `YYYYMMDD` format, where `YYYY` represents current year displayed with 4 digits, `MM` represents current month displayed with 2 digits, and `DD` represents current day with 2 digits.

```
CDATE|I|_3|FILENAME|_5|_6|ORG|UNK|||_11||||||Y||||_21|_22|||||_27|_28|1|_3
0|||_33||_35|_36|Y|||||||||||||||||||||||||_63|_64|_65|_66|_67|0
```

Algorithm 7: Example of a string that defines DID file format

The final IDL file is progressively created by calling `CreateRow()` and appending every returned row to the final file. After that, it is archived in ZIP archive. Then, the establishes the connection to the MOVEit file storage specified in `Access` (from ER diagram) and selected for this request by a user.

MOVEit Connection

MOVEit API is implemented using Visual Basic (VB), but by utilizing the language interoperability of .NET languages and COM objects, it can be used in C# language as well.

The API provides a public constructor for an object `clientObj`. This object then provides all the methods required to work with MOVEit storage. The Ipswitch does not provide detailed documentation for the API, but rather offers an implemented example and its source codes, so developers can copy the implementation.

DataBridge Response Processing

DataBridge uploads the response to MOVEit archived in ZIP archive. This archive contains few different files and one of them is the ERR file that user wants to see and analyze. DataBridge names the archive based on the date and time when the check was done. The application has to check MOVEit storage regularly to verify whether DataBridge uploaded new response or not. It has to download every new ZIP archive and look through files inside the archive, because the only filename it can identify is the ERR file filename. The ERR filename is generated based on the original IDL file uploaded to MOVEit.

When the application matches an ERR filename to the filename of a request, it sets the request state to "Response received" state. User can then open the request and the response will be processed.

Response file contains error identification on each row. The application has regular expressions specified in `MatchString` (in `Response` from ER diagram) and it uses them to identify the error. Once the error is identified, the application saves `CorrectionString` and `Message` values from the same `Response` entry and sends it to JavaScript.

Optimization

Since one of the requirements for the application is to handle big amounts of data, some optimizations had to be made. Firstly, `DataContext`, which can be used to access object relation mapped data in database inside EF, is very slow when inserting around few thousands of cells into the database. The performance was tested and the results were unacceptable. Inserting around a million of cells took over an hour. Consequently, an alternative process was implemented to handle inserting big amounts of data, which utilizes `SqlBulkCopy`[1] class provided by Microsoft to insert bulk data.

It was also important to implement most of the operations to work with as few data as possible. Therefore, most operations in the application work with list[2] of strings, where each string represents a cell in the source data and each row is represented by a separate list. By using this method, we are relying on the application to provide correct structure of lists in order for the application to

[1] https://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqlbulkcopy%28v=vs.110%29.aspx
[2] https://msdn.microsoft.com/en-us/library/6sh2ey19%28v=vs.110%29.aspx

work properly. But we eliminate the need of storing additional information which we would have to, if we used list of `Data` class representations provided by `DataContext`.

Account Management

The application provides basic account management and stores FTP details for the application, so it can access file storages to upload and download IDL files from. All the user information is stored in the application's database and can be accessed upon user authentication.

Registering an account is not provided to guests, but it is restricted to administrators only. Same applies to the file storage access details, they can only be added by administrators. The reason for this, is that the storage access is assigned by IMS Health, so there is no need for users to modify it.

Users can view their personal information and statistics by accessing their account page by clicking on their username in the right hand corner of any web page.

The application provides statistics to users about the number of requests they created and percentage of failed, pending, and successful requests. It displays the statistics in a simple graph, so the information is easily readable.

Security

Security is an important part when developing any application and main focus for security precautions was the account management and database security. The application secures user's passwords and MOVEit account passwords.

The application secures user's passwords to access the application using Bcrypt hashing function. It is based on Blowfish cipher and has been around since 1999 and still remains uncracked. The algorithm uses salt implicitly so it is more resistant to Rainbow Table attacks, compared to algorithms without salt implementation [25]. Bcrypt is one of the most widely used hashing functions and it still remains uncracked. An only option for cracking a Bcrypt hash is a brute-force[1] method and during a cracking competition in 2012 KoreLogic, only 76 Bcrypt hashes were cracked in the span of 48 hours [24].

Since we need to retrieve MOVEit login details in order to use the file storage and we still need to secure the details, the application uses an AES encryption to secure the details. It is used by U.S. to secure sensitive but unclassified information, so we can assume it is secure enough. [23]

4.2 JavaScript

The use of JavaScript was already briefly explained in Chapter 3.1. Here we will describe important functions and algorithms for the application.

Database Access Mode

The spreadsheet application was implemented as an integral part of the data integration process. We were also debating whether it would be possible to implement the application as a live data editor to allow users to get an immediate response from the database. Unfortunately, IMS Health does not allow direct access into MI database to regular users. DataBridge is the only interface providing the access which is used by the customers. Furthermore, it would be very difficult to

[1] http://searchsecurity.techtarget.com/definition/brute-force-cracking

replicate all the security checks and data mapping used in the whole data integration process. Regardless of the fact that such application would be very hard to implement to be flexible enough to adapt to all the changes that may occur in the system. Consequently, we decided against live approach and instead we focused on implementing features to guide user through the whole process of data integration.

Output Format Verification

The JavaScript application allows users to map columns from the DID file format to the columns of the source data. The mapping is done through option selection from a dropdown menu. Dropdown menus area available for each of columns in the spreadsheet.

The spreadsheet also verifies, whether all the required headers were selected. The headers are loaded into the spreadsheet by sending an asynchronous JavaScript and XML (AJAX) request to the server and the server responds with all the headers, each defined in a separate JSON object. The object also contains attributes, which contain verification RegEx string, correction RegEx string, information whether the header is required, and the name of the header. It uses then the verification string to verify the text format in each cell of the selected column, if it meets requirements specified by the DataBridge. If some of the cells does not meet the required format, it uses the correction string in combination with JavaScript's function `Regex.replace()` and suggests corrections to the user. An example of correction string is in Algorithm 8, it shows that the string is split into 2 parts using the `":"` character as a separator. The RegEx part is used to match the incorrect parts and the other part defines what character should be the incorrect parts replaced with. In this case we match all non-word characters and replace them with a dot.

```
[^a-zA-Z0-9.]+:.
```

Algorithm 8: Example of a header correction string

DataBridge Response Display

Response from the DataBridge is processed and sent to the spreadsheet application by the server. It is sent as a response and each of the errors is defined in a separate JSON object. The object contains row and column identification, which is used by the spreadsheet to display which cell did not pass the verifications specified by the DataBridge. The JSON object also contains the original message and a correction string, which has the same structure as the header correction string from Algorithm 8.

Displaying Progress

Since some of the data that will be uploaded to the application will probably be pretty big, it is important to show the users progress of the upload, because some of the operations might take long. It is possible to implement such behavior in ASP.NET using remote procedural calls (RPCs). It is possible by utilizing SignalR Hubs API provided by Microsoft[1].

The server communicates with the client by sending messages through `SendMessage()` function and client actualizes the progress based on the messages it receives.

[1] http://www.asp.net/signalr/overview/guide-to-the-api/hubs-api-guide-javascript-client

Optimization

From performance testing done with the application, it turned out that browsers have issues with more than few thousands of DOM elements in a web page. It usually becomes unstable and starts to lag. Since we want users to use our application without many issues, optimizing the spreadsheet was an important step. The application displays only data that are visible to the user at the moment and generates more of them when the spreadsheet table gets scrolled.

Cross-browser Compatibility Issues

The cross-browser compatibility was important to resolve, because it is impossible to predict, what browsers are users going to use and it is incorrect to support only some. The application was tested with all the modern browsers and no issues were found. jQuery allowed for a unified implementation of most of the functions.

The spreadsheet provides its own implementation of an **undo** feature. It has a behavior similar to Microsoft Excel. The motivation for the implementation was to provide a consistent behavior across all the browsers.

Each change is saved as an object into an array with all the changes. It has 3 properties:

- `id` – This property specifies which cell has been changed
- `old` – It contains the original value before the change
- `new` – It contains the value that the cell changed to

The array of changes is saved in an object called `changeLog` inside its property called `changes`. It has one more property called `id` that identifies which change will be reverted in the next call of `Undo()` function.

There is also an **autosave** feature which saves the changelog object into the application's database by sending an AJAX request to the server. It is then possible to retrieve the object back into the application by requesting it from the server again by sending an AJAX request.

Additionally, the **copy and paste** default behavior had to be modified in order to provide users functionalities similar to Microsoft Excel. It is also possible to copy the selected area of cells as well as paste it into the editor.

Both of the actions are compatible with Microsoft Excel. It is possible to copy data from Microsoft Excel to the editor and also to copy the data from the implemented editor to Microsoft Excel. Although, cell and text formatting is not supported by purpose by the copy and paste features.

5 Testing

This chapter will introduce the application testing, its importance in software life cycle, and it will describe which approach was selected for this implementation. Then we will discuss the test scenarios and test results. The following chapter is based on CD Content.

5.1 Methodology

The purpose of a testing phase is to check the implemented application whether it fulfills the criteria defined during the requirements specification and whether it can successfully operate in the desired environments.

Software developers follow different types of methodologies in order to validate an application before its release to end users. Each implementation requires different approach and there is not only one correct testing strategy. For this application, we will focus on three areas: unit testing, integration testing, and system testing.

The unit testing will focus at each component separately and testing will be performed when the implementation of the component is completed.

Once all units are developed and tested, we can move to the next step where we perform the integration testing. In this phase, we will prove that all components work together and all the requirements are met.

Finally, the system testing is performed in the target environments to verify whether it works as required in connection with the other systems.

5.2 Unit Testing

Application development is usually divided into smaller units which can be described, understood, developed, and tested separately. The unit tests are usually performed by developers once they implement the unit. The unit is tested separately from the system or other application units.

During the development we have identified key units and they are specified in the following list below:

- ASP.NET application
    - Login
    - Navigation bar
    - Wizard
    - Session Manager
    - Response Manager
- JavaScript spreadsheet
    - Undo
    - Copy/paste
    - Column verification
    - Response view
    - Area selection

All the unit tests were completed repeatedly and all identified issues and problems were fixed before we started to work on another unit. Once the development and unit testing was completed, we moved to the next stage where we do the integration and system testing.

5.3 Integration and System Testing

The integration testing is performed in order to check whether all the units provide the desired functionality and meet the initial requirements together. Sometimes it is called a scenario testing [19]. We will also perform the system testing because the interaction with other systems is crucial for our application.

Integration and system testing should be defined, prepared, and performed by nominated testers and not by developers themselves. This approach provides an independent view on the application's behavior. In case of this project we could not afford to have fully dedicated and independent testers therefore we followed a compromise approach and we did most of the integration tests personally and only some parts of the system tests were done by IMS Health employees.

Firstly, we defined the initial requirements. Then, we used the requirements to form the features that we tested in the test scenarios. For each requirement we identified steps with expected results to check whether the feature works as expected or not. The failed scenarios were analyzed and the application was modified according to the test results. Then, the test scenario was repeated.

In the first round of tests 7 out of 59 tasks failed:

Result	Count	Percentage
Failed	7	12%
Passed	52	88%
Grand Total	**59**	**100%**

Picture 10: First round test results

One of the failures (test case 6.21 from test protocol on CD Content) came from the CRM system and it was not caused by the application. The goal of the test was to verify whether the hospital sales data were integrated correctly and visible in the CRM system. However, one of the hospital for which we loaded sales data was not visible in the system. The CRM support team was approached and necessary changes were applied to make required hospitals visible. Then the scenario was retested and all the data were displayed correctly in the system.

Finally, a new version was developed and the whole scenario was repeated. This second round of tests was completed with 100% success rate.

5.4 Other tests

On top of the initial planning we also performed a non-functional test. Then, we compared the current process that end user had to complete to load data into the CRM system with the situation using our application. Most of the actions done by a user were replaced by an automated process done by our application. The only remaining tasks were an uploading of the source data file and an eventual modification of the data, based on the response log file processed and displayed by the application. We can see the achieved improvements in Picture A.4 showing data integration process with our application:

6 Conclusion

This thesis focuses on analyzing elements of CRM system of IMS Health and implementing an application for its customers based on findings. We explained the external data integration process in the CRM system. We pointed out possible CRM system deficiencies, and we designed the application to address them. Then the application was implemented according to the design specification. Finally, we performed many tests in order to verify, whether the application meets specified requirements.

The application was then presented to the project team of IMS Health. They tested the application and found its improvements beneficial for the future end users and noticeably data integration process more easy to use.

One of the goals of the thesis was to implement an application. Its purpose is to provide a user-friendly data interface. It was created in C# using ASP.NET framework. It will replace the current data load process, which is not very comfortable for users who are not IT skilled.

A complex spreadsheet was developed for the application. It provides various features including undo, area of cell selection, area copy and paste, copy and paste compatibility with excel. Furthermore, it provides a key functionality to the user to map source data column to the DID file interface field. Also, it verifies whether the source data meet the format requirements defined by the DID interface. Another benefit is also configurability of the DID interfaces. All rules are stored in the database therefore future changes in the DID format will not require new version of application.

Another feature supporting daily usage is a session manager. It offers possibility to save current status of work done in the spreadsheet. It also provides an autosave feature.

The biggest added value of the application, which is unique across all the currently provided solutions by IMS Health, is the ability to process the DataBridge response log file and map the returned errors to rows in the source data file. In addition, the application suggests to the user possible corrections of the common errors returned from DataBridge.

There were few challenges during the application development. The first issue was identified during the unit testing and was caused by browser's incompatibility and different implementations of some of the JavaScript features. The code had to be modified several times to ensure consistent behavior across all the major browsers. Another issue was discovered during the implementation of the MOVEit file storage access. The MOVEit API had to be reverse engineered using its provided source code, because Ipswitch does not offer detailed documentation to understand the API enough to implement it.

Comparison with the existing interface

To show what advantages the application provides and how it improves the process of data integration for user, we can compare Picture 3 (page 11) and Picture A.4 (page 38). From the comparison, we can clearly see majority of the actions are now done by the application instead of the user. The only remaining tasks done by the user are to upload the source data file and to eventually modify the data, based on the response log file processed and displayed by the application. This is one of the main benefits of the application to the customers.

It is worth mentioning that the application goes beyond capabilities of the current CRM data interface by providing sessions and displaying database response logs on top of the advanced DID format verifications, which is partly provided in the current data integration process.

Future development

The application is designed and developed with extensibility in mind. Therefore, it is simple to implement additional source file format and file storage access support. The application could be extended to support new customer's source file formats. These formats would come directly from the customer's database, which would simplify the data integration even further. By utilizing the already implemented session manager, it would be possible to store the customer's columns identifications and use them in the future data integration processes.

We could also extend the text verifications and correction suggestions to be more advanced, which would be capable of prediction and more accurate error matching process.

Bibliography

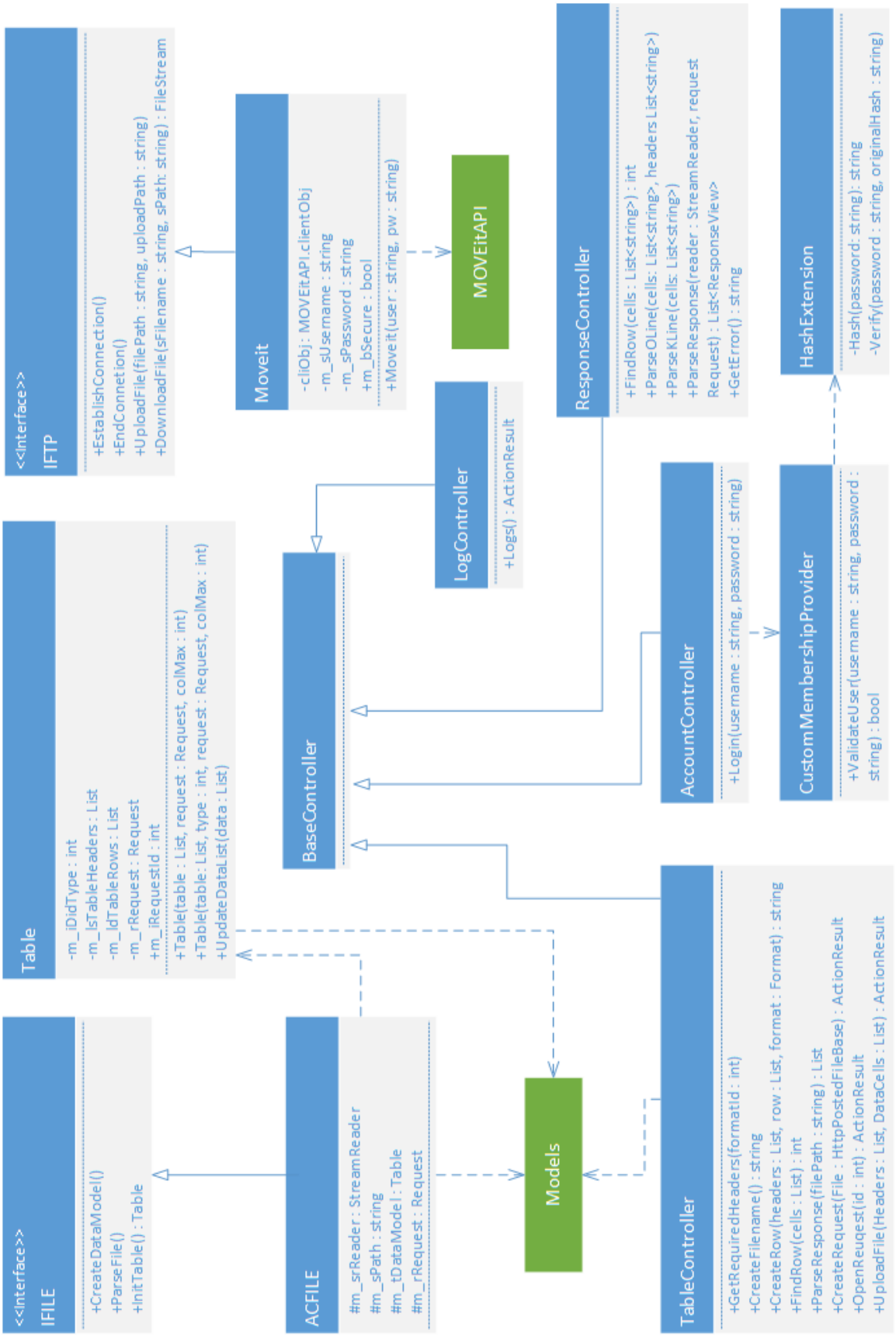
1. Overview of the .NET Framework. *MSDN Developer Network.* [online]. © 2016 [cit. 2016-05-02]. Dostupné z: *https://msdn.microsoft.com/en-us/library/zw4w595w%28v=vs.110%29.aspx*
2. .NET Framework (.NET) . Techopedia. [online]. © 2016 [cit. 2016-05-02]. Dostupné z: *https://www.techopedia.com/definition/3734/net-framework-net*
3. About .NET. .NET CORE. [online]. © 2016 [cit. 2016-05-02]. Dostupné z: *http://dotnet.github.io/about/*
4. Common Language Runtime (CLR). *MSDN Developer Network.* [online]. © 2016 [cit. 2016-05-16]. Dostupné z: *https://msdn.microsoft.com/en-us/library/8bs2ecf4%28v=vs.110%29.aspx*
5. What is CLR (Common Language Runtime) . *Carlos Bercero.* [online]. 16.11.2009 [cit. 2016-05-02]. Dostupné z: *http://carlosbercero.com/post/?post=What_is_CLR_%28Common_Language_Runtime%29*
6. Overview of the .NET Framework. *ASP.NET Overview.* [online]. © 2016 [cit. 2016-05-02]. Dostupné z: *https://msdn.microsoft.com/en-us/library/4w3ex9c2.aspx*
7. LINQ – Overview. *TutorialsPoint simply easy learning.* [online]. © 2016 [cit. 2016-05-02]. Dostupné z: *http://www.tutorialspoint.com/linq/linq_overview.htm*
8. LINQ: .NET Language-Integrated Query. *MSDN Developer Network.* [online]. © 2016 [cit. 2016-05-02]. Dostupné z: *https://msdn.microsoft.com/en-us/library/bb308959.aspx*
9. Linq. *YMD Tech.* [online]. 11.8.2015 [cit. 2016-05-16]. Dostupné z: *http://www.diranieh.com/NETCSharp/LINQ.htm*
10. What is Entity Framework?. *Entity Framework Tutorial.* [online]. © 2016 [cit. 2016-05-02]. Dostupné z: *http://www.entityframeworktutorial.net/what-is-entityframework.aspx*
11. LERMAN, Julia. *Programming Entity framework*. 2nd ed. Sebastopol: O´Reilly, c2010. ISBN 978-0-596-80726-9.
12. Secure File Transfer Server for the Enterprise . *ipswitch.* [online]. © 2016 [cit. 2016-05-02]. Dostupné z: *https://www.ipswitch.com/resources/data-sheets/moveit-file-transfer-datasheet*
13. *Imshelath.* [online]. 2016 [cit. 2016-05-02]. Dostupné z: *http://www.imshealth.com/*
14. Regular Expressions in ASP.NET. *MSDN Developer Network.* [online]. 1.3.2004 [cit. 2016-05-05]. Dostupné z: *https://msdn.microsoft.com/en-us/library/ms972966.aspx*
15. JavaScript. *MDN Mozilla Developer Network.* [online]. 2.5.2016 [cit. 2016-05-05]. Dostupné z: *https://developer.mozilla.org/en-US/docs/Web/JavaScript*
16. Cascading Style Sheets. *W3C.* [online]. 5.5.2016 [cit. 2016-05-06]. Dostupné z: *https://www.w3.org/Style/CSS/Overview.en.html*
17. What is jQuery?. *jQuery.* [online]. © 2016 [cit. 2016-05-06]. Dostupné z: *https://jquery.com/*
18. Bootstrap Get Started. *w3schools.com.* [online]. © 1999-2016 [cit. 2016-05-06]. Dostupné z: *http://www.w3schools.com/bootstrap/bootstrap_get_started.asp*
19. Testing Methodologies. *Inflectra.* [online]. 25.8.2015 [cit. 2016-05-07]. Dostupné z: *https://www.inflectra.com/Ideas/Topic/Testing-Methodologies.aspx*

20. *PATTON, Ron. Testování softwaru. Praha: Computer Press, 2002. Programování. ISBN 80-722-6636-5.*
21. ASP.NET Razor – C# and VB Code Syntax. *W3C.* [online]. © 2016 [cit. 2016-05-15]. Dostupné z: *http://www.w3schools.com/aspnet/razor_syntax.asp*
22. Introduction to ASP.NET Web Programming Using the Razor Syntax (C#). *ASP.NET.* [online]. 7.2.2016 [cit. 2016-05-15]. Dostupné z: *http://www.asp.net/web-pages/overview/getting-started/introducing-razor-syntax-c*
23. Advanced Encryption Standard (AES) . *TechTarget.* [online]. 1.11.2014 [cit. 2016-05-15]. Dostupné z: *http://searchsecurity.techtarget.com/definition/Advanced-Encryption-Standard*
24. CMIYC 2012 has ended!. *KoreLogic security.* [online]. 1.11.2014 [cit. 2016-05-15]. Dostupné z: *http://contest-2012.korelogic.com/stats.html*
25. ASSOCIATION, USENIX. Proceedings of the FREENIX track, 1999 USENIX annual technical conference: June 6 - 11, 1999, Monterey, California, USA. Berkeley, Calif: USENIX Ass, 1999. ISBN 18-804-4632-4.

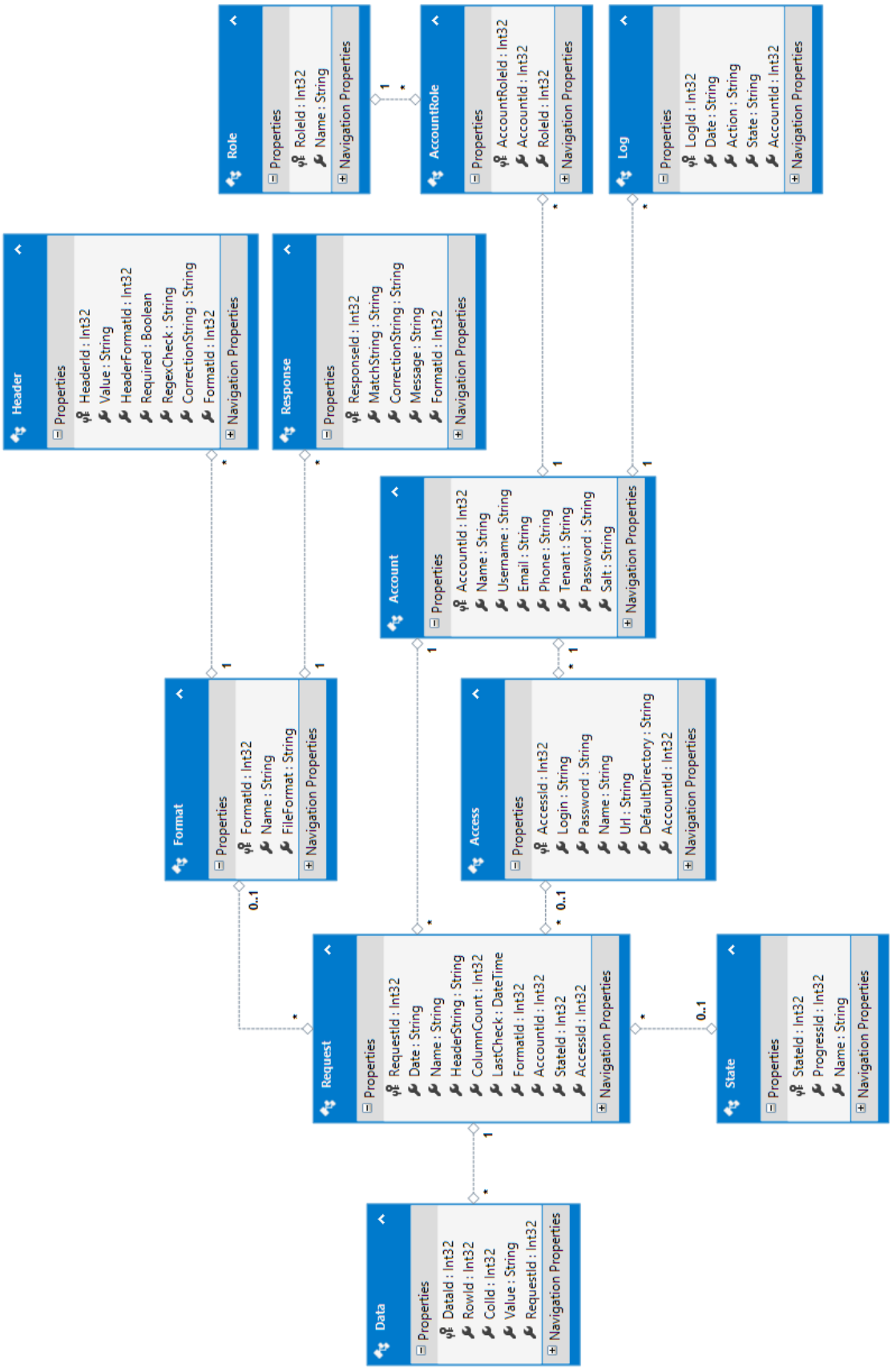
Abbreviations

AD	Active Directory
AJAX	Asynchronous JavaScript and XML
BCL	Base Class Library
CIL	Common Intermediate Language
CLI	Common Language Infrastructure
CLR	Common Language Runtime
CLS	Common Language Specification
COM	Component Object Model
CR	Carriage return
CRM	Customer Relationship Management
DBAs	Database administrators
DID	Data Interface Document
DLP	Data loss prevention
EF	Entity Framework
EOL	End of the line
FCL	Framework Class Library
FIPS	Federation Information Processing Standard
GUI	Graphical user interface
HCO	Health care organizations
HCP	Health care professionals
HTTP	Hypertext Transfer Protocol
IDL	Interface Data Load
IdP	Identity Provider systems
IIS	Internet Information Server
JIT	Just-In-Time
JS	JavaScript
JSON	JavaScript Object Notation
LDAP	Lightweight Directory Access Protocol
LF	Line feed
LINQ	The Language Integrated Query
MDM	Master Data Management
MOVEit	MOVEit Transfer
MS	Microsoft
Nucleus	Nucleus360
ORM	Object-relational mapping
POCO	Plain old CLR objects
RAD	Rapid application development
Razor	Razor inline syntax
RegEx	Regular expression
RPCs	Remote procedural calls
WinForms	Microsoft Window Forms
WPF	Windows Presentation Foundation
WYSIWYG	What You See Is What You Get

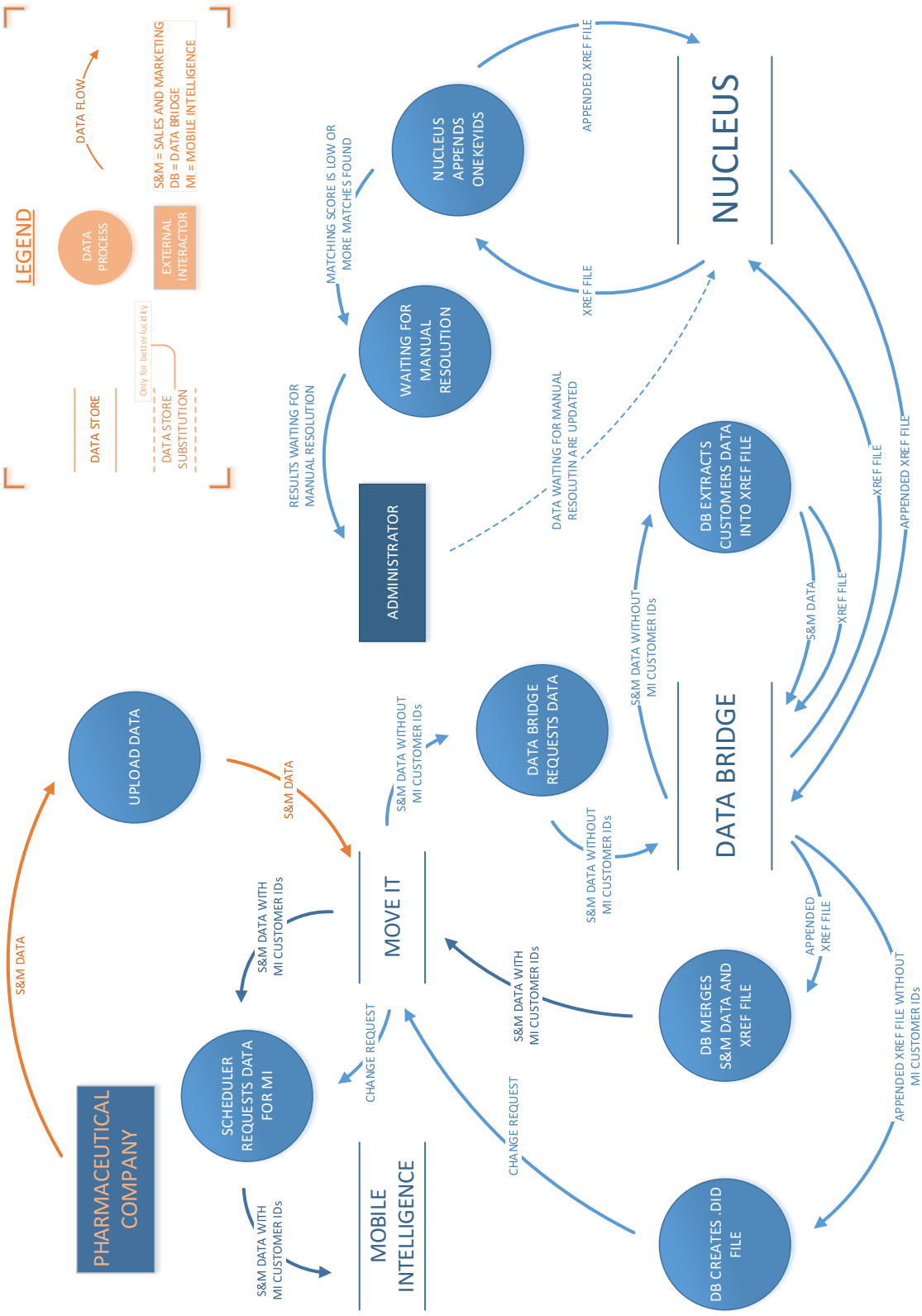
Appendix A: Pictures



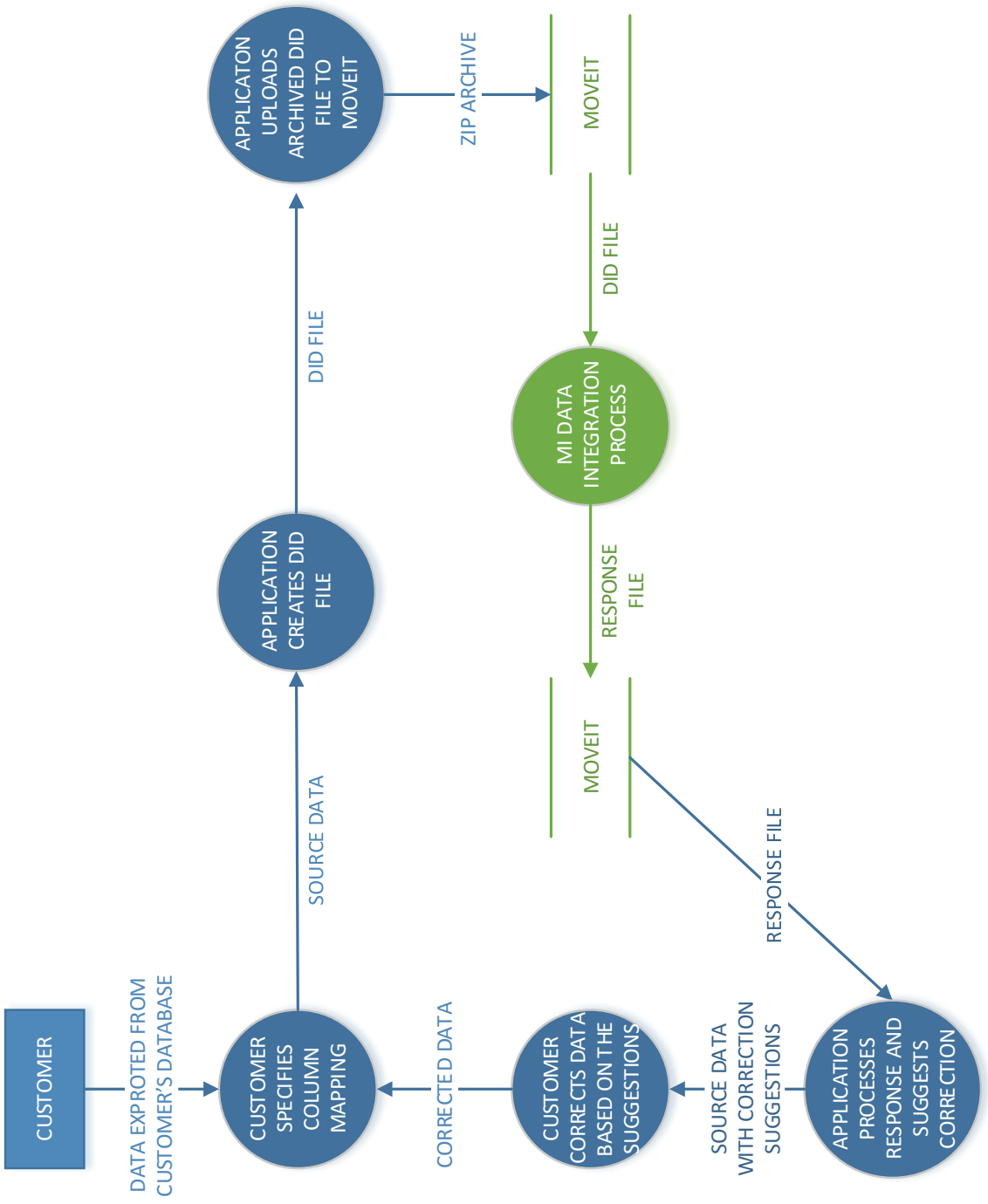
Picture A.1: Class diagram describing simplified structure of the ASP.NET application



Picture A.2: ASP.NET application database ER diagram



Picture A.3: CRM system internal data flow



Picture A.4: Data integration process from customer's perspective with the application

Appendix B: CD Content

CD Directory Structure

- SRC – Directory containing the source codes of the application
- Thesis – Directory containing the word source code
- Documentation – Directory containing all referenced documentations from IMS Health
- Testing – Directory containing test cases protocol and its results
- README.txt – Text file containing information about compiling the application