

**A web application solution for symbol file storing and  
distribution**

Jaakko Leinonen

University of Tampere  
School of Information Sciences  
Computer Science  
M.Sc. Thesis  
Supervisor: Marko Junkkari  
November 2013

University of Tampere  
School of Information Sciences  
Computer Science

Jaakko Leinonen: A web application solution for symbol file storing and distribution

M.Sc. Thesis, 65 pages + 10 pages of appendices

November 2013

---

**Abstract**

Data dispersion is a common problem in many companies. Valuable files of the enterprise are not well managed and decentralization of the files causes problems for finding and using the needed data. This implies that the efficiency of the work decreases. Symbol files are not an exception and companies like Microsoft and Mozilla have created their own systems for symbol file storing and distribution. Both of the mentioned companies provide their own solution for the centralized symbol server. The solutions are notably easy to use and automated as much as possible.

In this thesis I present a web application solution for data dispersion problems for symbol files storing and distribution. The developed solution differs from the Microsoft's and Mozilla's symbol servers by providing also a web UI that enables effective data browsing and searching features.

This thesis attempts to describe all the needed parts of the web application development area that must be taken into account. The provided solution meets the set requirements for storing and distributing data effectively inside the company. The presented solution is also easily extendable for storing any kind of files and the solutions for distribution are versatile.

**Keywords:** Web Application Development, Symbol, Debugging, Data Dispersion, PHP

## ACKNOWLEDGMENTS

I would like to thank many people who made this thesis possible. First of all I would like to thank my thesis supervisor Marko Junkkari for his support and great professional guidance throughout the writing process.

I would also like to thank Pasi Kauraniemi who always believed in me and pushed me forward when I had some challenging times with the thesis. I would like to thank Janne Koski and John Ervoky for making the subject of the thesis possible. Petri Määttä deserves my gratitude for helping me with some of the most challenging topics of the thesis.

Special thanks to my wife Riina and the rest of the family for supporting me at all times and making my life as great as it is.

Tampere, November 2013

Jaakko Leinonen

# CONTENTS

<b>TERMS AND ABBREVIATIONS .....</b>	<b>V</b>
<b>1. INTRODUCTION .....</b>	<b>1</b>
<b>2. BACKGROUND .....</b>	<b>4</b>
2.1. DATA DISPERSION .....	4
2.2. SYMBOLS .....	5
2.3. SYMBOLIC DEBUGGING .....	6
2.4. SYMBOL SERVERS .....	9
<b>3. WEB APPLICATIONS .....</b>	<b>11</b>
3.1. PRINCIPLES OF THE WEB .....	11
3.1.1. <i>Client-Server Architecture</i> .....	12
3.1.2. <i>Web Server</i> .....	13
3.2. DATA MANAGEMENT.....	14
3.2.1. <i>DBMSs</i> .....	15
3.2.2. <i>SQL</i> .....	16
3.3. MVC .....	17
3.4. PROGRAMMING LANGUAGES USED.....	18
3.5. AJAX.....	20
<b>4. PHP FRAMEWORKS .....</b>	<b>22</b>
4.1. FEATURES.....	23
4.2. PERFORMANCE .....	24
<b>5. SYMBOLDATABSE .....</b>	<b>28</b>
5.1. USE CASES .....	29
5.2. REQUIREMENTS.....	31
5.2.1. <i>Web Application</i> .....	31
5.2.2. <i>Database</i> .....	33
5.3. TECHNIQUES.....	34
5.4. ARCHITECTURE AND DESIGN.....	35
5.4.1. <i>Structure of Yii Framework Application</i> .....	36
5.4.2. <i>Commands</i> .....	36
5.4.3. <i>Components</i> .....	37
5.4.4. <i>Config</i> .....	37
5.4.5. <i>Controllers</i> .....	38
5.4.6. <i>Models</i> .....	41

5.4.7.	<i>Logging</i> .....	41
5.4.8.	<i>Views</i> .....	42
5.5.	SYMBOL COLLECTOR.....	42
5.6.	WEB UI.....	43
5.6.1.	<i>Introduction</i> .....	43
5.6.2.	<i>Views</i> .....	45
5.7.	INTERFACES.....	51
5.7.1.	<i>Content Usage</i> .....	51
5.7.2.	<i>IAD Usage</i> .....	52
5.7.3.	<i>Web Service</i> .....	53
5.8.	DATABASE.....	54
<b>6.</b>	<b>SUMMARY AND CONCLUSIONS</b> .....	<b>56</b>
	<b>REFERENCES</b> .....	<b>57</b>
	<b>APPENDIX A: DATABASE TABLES</b> .....	<b>60</b>

## TERMS AND ABBREVIATIONS

API	Application Programming Interface.
Collectionset	A set of files which share the same identifier (ROM ID).
CGI	Common Gateway Interface. Standard method for web server software to delegate the generation of web content to executable files.
Compiler	Program that translates source code into object code (usually binary code).
Crash Decoder	A tool for decoding binary crash files.
Data Dispersion	Decentralized homogeneous data inside a system or network.
Debugger	A tool which is used to test and debug other programs.
GUI	Graphical User Interface.
Firmware	Combination of persistent memory, program code and data stored in it.
IAD	Independent Application Delivery. A packet contains an application and symbol files.
Metadata	Descriptive data about data.
ROM	Image and Symbol file formation.
ROM ID	Checksum calculated from the image creation for the ROM file. Identifies collection set.
ROM Image	File that contains a firmware for a mobile device.
SIS	Software packet which can be installed into the S60 device.
SOAP	Simple Object Access Protocol.
SUI	Symboldatabase UI. Web UI application.
Symbol	Contains debugging information of the image file.
UID	Unique Identifier.

## 1. INTRODUCTION

*Data dispersion* is a common problem in many companies. Valuable files of enterprise may be decentralized in multiple locations and or even in employer workstations. Reasons for data dispersion can be miscellaneous. One of the biggest reasons is a lack of communication inside the company. Decentralization of files not only causes possibilities to loss of valuable information, but it also inevitably affects to work efficiency. Because of the mentioned reasons, there might be situations where a worker uses more time for searching the needed files to be able to perform a certain task, than the actual task execution would take. Undoubtedly, this is a major data loss, performance and efficiency problem which need to be solved.

Decentralized storages like different servers and shared network drives still host a lot of data nowadays. It is a good and simple solution for relatively small amount of data that does not need any extra information than the filenames or folder names provide. When the amount of stored files expands or more metadata about the files is needed, the limitations of such a solution will start causing problems. Different kind of solutions for file storing and distribution must be developed to be able to improve the existing processes or to develop novel methods.

Web solutions are used more and more in creating applications. Due to the web servers the web applications have a very good accessibility to all related users by using internet or intranet. In data dispersion point of view a web application is in many cases the most easily accessible location inside a company. Advantages of the web application are that a web application can provide features for data management and distribution. A web application also contains easily manageable and extensible GUI which helps to provide files to users with more specific information and advanced features like searching, authentication and caching.

The subject for this thesis came up from the real working project. In Nokia Corporation the *symbol files* which contain the *debug information* of the device software do not have centralized repository from where the files are extensively distributed to different kind of clients. This is not problematic only for the single developer but also for the automatic systems and scripts which require those files to work properly.

In an early solution the symbol files were uploaded to servers shared network drive by build teams. The build teams are responsible for releasing software versions for the devices. Then a processing script wrote a *ROM ID* value and a path of the files to a TXT file. A decoding tool reads the whole TXT

file while searching a matched ROM ID value to be able to use the correct symbol files for the decoding purposes.

Problems of the early solution were that tens of thousands of files were located on a single folder and files did not pass other information than their filename. Also reliability problems with the uploaded content were usual, because the files were uploaded by different teams around the company with different processes and practices. In addition, most of the developers were not aware of the symbol file storage repository, so users had to seek the needed files from elsewhere. There was a definite need to create a centralized storage for the symbol files which could also collect and distribute its content widely inside the company.

For solving the mentioned problems a system called *Symboldatabase* was developed. Symboldatabase is a system which consists of the following components:

- Server Application
- Database
- Web UI.

Server Application is responsible for collecting stored files from the multiple locations and providing *API's* for distribution. Database stores *metadata* of the stored files. The stored metadata includes valuable information about the files such as filename, file type and file path. With metadata information the files are introduced and distributed to clients. Symboldatabase is also responsible for providing those files to Crash decoding tool. Symboldatabase provides a dynamic *HTTP interface* and a *web service*, which provides files and metadata as requested by a client tool. Symboldatabase also provides a web UI that can be used for content browsing, searching and downloading. With the web UI a user can effectively and easily browse and download files.

Selected solutions for Symboldatabase implementation were selected after feasibility study. PHP was selected for the web UI due to its popularity and because it provides a good API and documentation. The *Yii PHP Framework* was selected after some features and performance comparison of the PHP mostly used frameworks. The *Ext JS JavaScript* framework on top of Yii Framework was chosen because it provides better UI components than Yii Framework. For the database solution a relational database was found most useful and *MySQL* was chosen of it is easy to use and maintain and *Yii PHP Framework* provides advanced libraries for handling and managing the *MySQL* database.



This thesis includes six chapters. Chapter 2 introduces data dispersion and what are the stored and distributed symbol files and for what purpose they can be used. The chapter also provides information about the existing symbol servers. Chapter 3 provides information about the architecture and mainly used solutions for the web applications. The chapter is a research section where different parts of the web application are under microscope. Chapter 4 introduces PHP frameworks. It also contains evaluation of the PHP framework versus pure PHP development. It answers to questions: What are the advantages of the PHP frameworks and are there any disadvantages which prefer pure PHP developing.

Chapter 5 presents Symboldatabase. It contains use cases, requirements and design for Symboldatabase. This is thesis implementation section and it focuses also on the database structure and *GUI* for providing Symboldatabase content to the users. Chapter 6 summarizes and concludes the thesis.

## 2. BACKGROUND

### 2.1. Data Dispersion

Data dispersion means decentralization of the similar data in a large scale system like intranets, not just in a single computer storage component. Practically data dispersion means that same kind of data is not stored into a centralized and properly managed repository. In many cases the location and accessibility of the data is not what it should be to make the usage of the data effective. It can be said that the fragmented data is not stored and distributed in the best possible way. Data dispersion is an infrequent part of any plan.

In practice, data dispersion is a common problem in many companies. The valuable files of enterprise can be decentralized in multiple locations and/or even in single employer workstations. The more data is divided into several locations, the greater is the risk to lose it. Decentralized storages like different servers and shared network drives still host huge amount of data. It is a good and simple solution for relatively small amount of data that does not need any extra information than the filenames or folder names provides. When the amount of stored files expands or more metadata about the files is needed, the limitations of such a solution will start causing problems. Other solutions for file storing and distribution must be developed to be able to improve the existing processes to develop novel methods.

Reasons for data dispersion can be complex. In a well-organized company a data is also well-organized and it is easily accessible for all the users that find it useful. Reasons for data dispersion in companies systems are usually linked to bad quality of processes. Another well-known reason is the lack of communication and unclear roles inside the organization. In these cases one can say that the flow of information is not working inside the company. Globalization and cultural differences can also cause problems to practicalities inside the company and that affects also to data storing. The lack of work motivation and other individual reasons can also cause data dispersion and losing valuable data. Whatever the reasons are for individual employer's problems, reasons for decentralized data problems can be seen as problems with processes and how the works are supervised.

On large scale companies, where the organization, communication and cross-team collaboration are not properly working, data dispersion consequences can become much more expensive than the company leaders can understand. When teams are working together it is important that those who produce the data and those who use the data communicate. Both sides have to

understand each other to make the cooperation to work. This is not possible in some cases if the data is coming from the outside of the company, but usually also when there is communication needed to know how the data should be used. In many cases, the importance of the metadata is not fully understood. When the created data does not contain any metadata or the metadata is not available, the user of the data does not know how to use it properly. This may cause that metadata is lost and data becomes unusable.

In my daily work as a software developer I have noticed that dispersed data inside the company causes lots of problems and makes working less effective. To be able to work properly, developer might have to use a lot of effort to find the correct data and tools to perform the task. In many cases it means that doing the actual job takes less time than finding the correct data or metadata. I believe that this is something that is not taken into account enough when creating processes and managing information flow inside companies.

## 2.2. Symbols

In C++ programming, when a software is coded by a developer, a *compiler* translates the code into the computer language that computer uses to execute the created software. When the compiler is creating a *ROM image* or other *executable file* from the source code, it also generates symbolic information related to it. A ROM image file or other executable program is aimed to keep as simple as possible. This keeps the size of the executable relatively small and the structure well organized. A symbol is a metadata record associated with a ROM image file or other executable file. The valuable *debug* information is stored into symbols files.

Symbols provide a footprint of the *functions* that are contained in executable files. In addition, when *debugging* an application, symbol files can help point to the function calls that led to a failure by helping the user to view his/her application's full *call stack*. [Darka, 2012] Symbol files share debug information about software for the device.

Typically, symbol files, also known as *private symbol data*, might contain the following content [Microsoft 3, 2013]:

- *Global variables*
- *Local variables*
- *Function names and the addresses of their entry points*
- *Frame pointer omission (FPO) records*
- *Source-line numbers.*

A global variable is a variable that is accessible in every entity of the computer program, whereas a local variable is only accessible from the function or block in which it is declared. A function is a *callable unit* also known as a *subroutine*. A function performs a specific task that is packaged as a unit. An entry point refers to a memory address, corresponding to the point in the code which is intended as the destination of a long jump. A frame pointer omission is a specific class of compiler optimization that deals how the compiler accesses local variables and stack-based arguments. The last item source-line numbers refers to the line in the code where symbol information refers to.

Individually each of these items is called a symbol. A single symbol file *release1.symbol* might contain hundreds of symbols, including global variables, function names and local variables. Software companies usually release two versions of each symbol file: a full symbol file that contains both *public symbols* and *private symbols*, and a stripped symbol file that contains only the public symbols. [Microsoft 3, 2013] The public symbol usually contains only function and global variable names and a general rule is that, the public symbol contains only those items that can be accessed from one source file to another. [Microsoft 4, 2013]

Table 2.1: Example content of a symbol file

<i>Address</i>	<i>Class::function</i>	<i>Object name (format)</i>
809a7351 :	CDataOutput :: ~CDataOutput()	CDataOutput.o (.text)
809a73e7 :	CDataOutput :: ~CErvoky ()	CDataOutput.o (.text)
809a73f7 :	CDataOutput :: ~DumpEventLogL()	CDataOutput.o (.text)

In Table 2.1 lines from a sample symbol file are shown. The first column expresses the address in the memory. Next column indicates the name of the class and its method/variable/constant that is executed. The last value indicates the name and format of the object that is created.

### 2.3. Symbolic Debugging

A debugger is a computer program itself. It lets a developer to run a program, line by line and examine the values of variables. A debugger also shows the values passed into different functions. This action from user is called debugging. Debugging helps the developer to figure out why his/her program is not running the way he/she expected it to. [Bolton, 2013] A debugger is very valuable tool for finding the program errors also known as *bugs*. Generally debugging is a process for finding and reducing the number of bugs and *defects* in software.

Farokhzad *et al.* [2010] claims that the most requested area of improvement in embedded design support are debugging tools. Debugging has always been amongst the biggest concerns of designers. Further, one issue is that embedded systems are becoming increasingly complex and have unique constraints which make them hard to debug. [Farokhzad *et al.*, 2010]

In McInerney's *et al.* [2000] patent for demand-based generation of symbolic information he describes debugging with symbols like this.

*High-level symbolic debuggers represent an attempt to let a programmer view running programs with the same level of abstraction as the original source code. To present the programmer with a view of their program that closely matches their own perception of how the program is operating, high-level symbolic debuggers must use symbolic debugging information provided by the compiler to perform a reverse translation from the machine instructions and binary data, back into the source code. [McInerney *et al.*, 2000]*

In other words symbols contain the mapping between a compiler-generated *machine code* and the developer created source code. Symbols help the user to understand what the program was processing when the problem occurred. Without proper symbol file the debugger may present *disassembly* output like

*call App+0x1234abcd(1234abcd).*

For a developer that kind of information does not say much since he/she does not know what this address means and what the program was doing at the time. With a help of symbols information debugger shows the addresses of functions, *parameters* and *variables*. When valid symbols are available, the debugger can print to equivalent function name found from the symbol file like

*call App+!ExampleClass::OnCheckboxSelected.*

Now the developer gets more specific information what is going on in his/her program and in what part of the code the problem exists. The problem occurred when the method *OnCheckboxSelected* in *ExampleClass* class was executed. With this information he/she will probably solve out why the program does not work as planned. He/she can *trace* and fix the bug more easily.

One problematic question is that how a developer knows what symbols should be used and where to get those for debugging the program? When developing some small program by itself, it easy to find and use proper symbol

files. On more complex systems like Symbian C++ device software, finding the proper symbols is much more complicated task, because the user may not know where the needed symbols exists or even know which symbols he/she should use. When using third party applications or libraries, useful symbols might be impossible to find. This could lead to situation where a program has a bug, but the developer cannot find or understand it.

For Symbian C++ there is a simple solution for matching a ROM image and the correct symbols together. It is called ROM ID, which is a 32-bit checksum of the ROM image. Without any proper identification, wrong symbols can be used for decoding. Using the wrong symbols a developer will receive totally wrong information about the bug and that is why one cannot fix it.

A compiler uses *symbol tables* to store debug information. Symbol table is a compile-time data structure. A compiler uses symbol tables to remember all the declarations, so it can find inconsistencies and misuses. Symbol table maps names into called attributes. For example it maps a variable name Y to its type *int*. A symbol table stores the following information [Singh *et al.*, 2008]:

- Type definitions of type names
- Each variable and its type
- A type and value of each constant name
- Formal parameter list and output type of each function or procedure.

Coplien [1994] describes how a truly object-oriented debugger in C++ uses symbol tables to debug a program. In this case a user has added a *breakpoint* into code. The breakpoint defines a point in the code that the debugger should investigate [Coplien, 1994]:

- The debugger analyses the “break pointed” object and parses it into a name of an *object pointer* and function related to that object.
- The object pointer is used to search for a variable from the symbol table. The class name, type and memory address are the properties that are found from the symbol table entry. A unique class and function names are generated and those are searched from the symbol table. The function addresses are collected from the entry.
- A breakpoint header table is searched for an entry containing the found function address. The breakpoint header table stores information about the breakpoints and how to proceed with the debugging. If there are other breakpoints found the debugger processes also those.

## 2.4. Symbol Servers

Microsoft provides its own solution for symbol server. Microsoft supports C++ debugging, which is the base of the Symbian C++ language. Microsoft provides tools for symbol storing and distribution. User can set up his own symbol server by using *SymStore* tool or use *SymSrv* tool to deliver the symbol files from Microsoft's centralized symbol stores. [Microsoft, 2013] Visual Studio also provides automatic symbols downloading to users for debugging a Visual Studio project. [Microsoft 2, 2013]

Developer can download either all symbols of some specific Windows version or use a simple HTTP interface to get some specific symbols only. The HTTP interface does not support browsing which is a bit surprise, because it is a useful feature as shown in the present study. Overall Microsoft has done symbol storing and distribution very easy and automatic. A debugger uses public, intranet and local stores all in a same way and user actions are kept simple and minimal.

Mozilla has a symbol server for Firefox Windows builds. Mozilla provides symbols via an HTTP interface. Similarly to Microsoft Symbol Server, the Mozilla symbol server can also be configured in Visual Studio options to download and use symbols automatically (Figure 2.1). Mozilla symbols can be downloaded via *symchk.exe* tool which is part of the Microsoft's Debugging Tools set for Windows. Unlike Microsoft, the Mozilla symbol server itself does not provide source-level debugging. Function names and call stacks are available, but without links to the source code. [Mozilla, 2013]

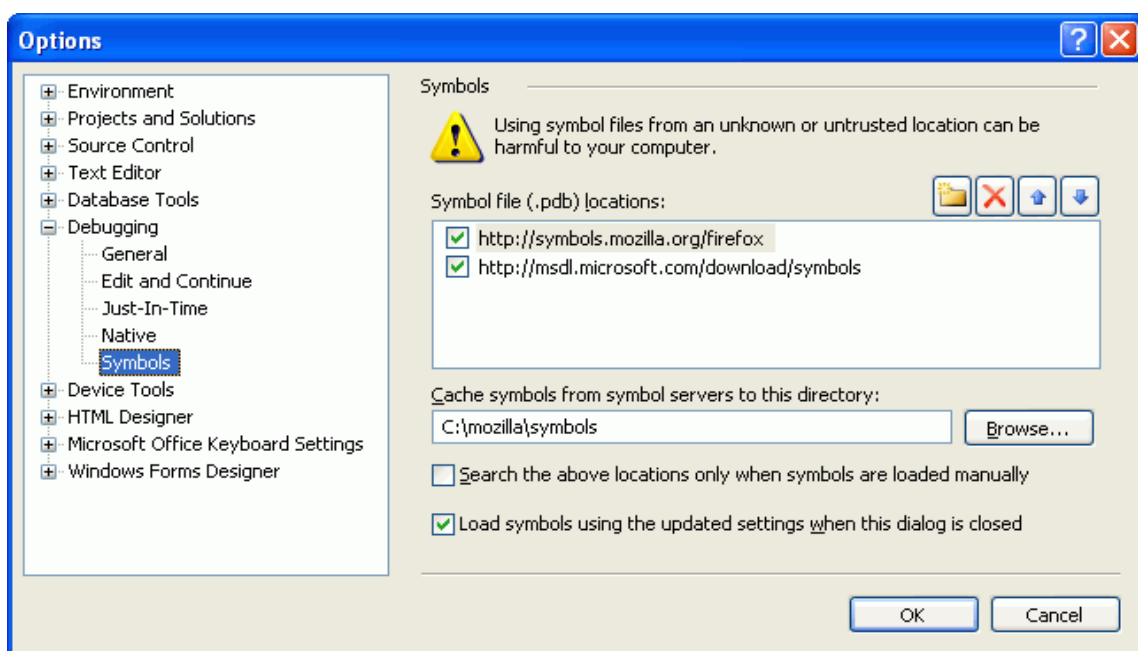


Figure 2.1: Visual Studio configuration for symbol debugging [Mozilla, 2013]

In addition to symbol server, Mozilla provides a source server, letting a developer to do source-level debugging and inspection on demand. To be able to use the source server, the symbol server must also be in use. When debugging is executed, the debugger will download the source code from the Mozilla source server and highlight the current line for the user. [Mozilla 2, 2013]

Uploading symbols to the Mozilla server is also possible. When a developer is creating a release of his/her application to Firefox or Firefox OS, it is recommended to also upload the symbols to Mozilla's symbol server. The upload process is done through a *SSH connection* to the symbol server and the user needs to make an account for that purpose. Mozilla provides scripts for generating and uploading symbols on a user local machine. If the user has possible sent crash reports from his/her own build to a Mozilla's crash reporting server, it is vital that also the symbols are uploaded to the symbol server. Without proper symbols the crash reports that developer sent will not contain much actionable information. [Mozilla 3, 2013]



### 3. WEB APPLICATIONS

What makes programming usually hard is largely a question of scale. Things that might be easily programmed on a single computer are much more difficult to distribute to other users [Stobart and Parsons, 2008].

A *Web application* can be defined as a *software system* that is accessible over the web. It uses technologies for the web and strives to use standard technologies where feasible [Jablonski *et al.*, 2011]. Acunetix [2011] defines that web applications are computer programs which allow website visitors to submit and retrieve data to/from a database over the internet or intranet using their preferred web browser.

Earlier, the software was built on a certain computer platform such as Windows or Linux. This is not the case with the web applications, because they can render the same content regardless of the system from which the page was loaded. The server can run on a different platform from client applications. “*Software above the level of a single device*” is one key pattern of the *Web 2.0*. It means that a web application can span in different types of devices such as desktop PCs, mobile phones and tablets. [Stobart and Parsons, 2008].

#### 3.1. Principles of the Web

The web as an environment has become more attractive because of its main principles of *openness*, *simplicity* and *ubiquity*. A web application can be deployed free of patent fees and other charges, which may not be the case with other environments. There are two aspects for principle of simplicity. Those are simplicity of use and simplicity of programming. The principle of ubiquity, in other words *omnipresence*, can be challenging, because it has span heterogeneous systems and has to use universal *transport* and *communication protocols*. [Jablonski *et al.*, 2011]

Web applications are getting more and more popular, because people can nowadays access the web almost where ever they are and whatever the time is. Hence scalability is an issue that must be always taken into account [Jablonski *et al.*, 2011].

*Accessibility* is an important property of the application and the internet has helped companies to share their applications easily to customers. Web applications are not just easily accessible, but they are nowadays also quite easy to implement and publish. Easy in this case does not mean that it would be easily done without a proper development processes. Thanks to rapidly developed frameworks and different helper tools the developing process and

many useful features are available for all. Those applications enable developer to create and publish great web applications with less knowledge and effort every day.

### 3.1.1. Client-Server Architecture

The *client-server architecture* is architecture between two components, the *client* and the *server*. Figure 3.1 represents the situation where the client makes requests for service to server component that satisfies those requests. In a web application a client and a server communicate by using the HTTP protocol. For each service, the client makes a request that the server-side operates and then returns a response to the client. HTTP is inherently a client-server protocol which is designed for implementation using the client-server architecture [Grove, 2010].

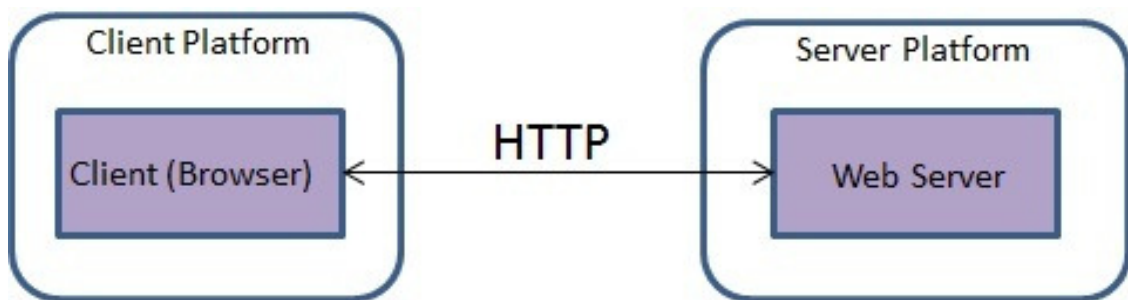


Figure 3.1: Client-server communication

Client-server architectures can be categorized by the number of *tiers* or *layers*. Typically tiered components include *UI*, *business logic*, *data management*, *firewall* and *web server* (Figure 3.2). *Two-tier*, *three-tier* and *multi-tier* (also known as *N-tier*) are some of the standard architectural design styles.

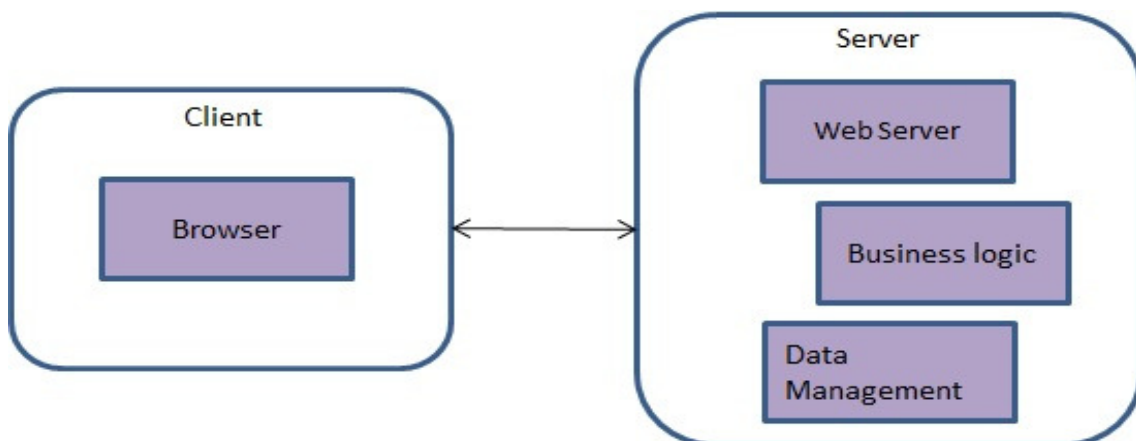


Figure 3.2: Typical Thin-Client configuration

In Figure 3.2, the typical *Thin-Client* configuration is described. Thin-Client uses a web browser to execute the application. This means that there is no special software client required to use the application. All *application logic* (business logic) is executed on the server-side. Data Management operations are also handled on the server-side. Web Server is needed on the server-side for enabling the client to connect to the server. [Grove, 2010]

Advantages of a Thin-Client are that configuration does not require much computing capacity from the client platform. This means that the application can be used for larger scale of computers. A disadvantage of the Thin-Client is that it needs a lot of communication between client-server and this can in some cases cause problems.

In Figure 3.3, the client tier is responsible for the user interface presentation and processing of the application. The server tier is responsible only for the data management which means data storing and executing database transactions [Grove, 2010]. The *Thick-Client* configuration needs more computing capacity from the client platform and it keeps the connection to server as minimum. Compared to the Thin-Client configuration, the Thick-Client is much simpler and does not require a web server on the server-side.

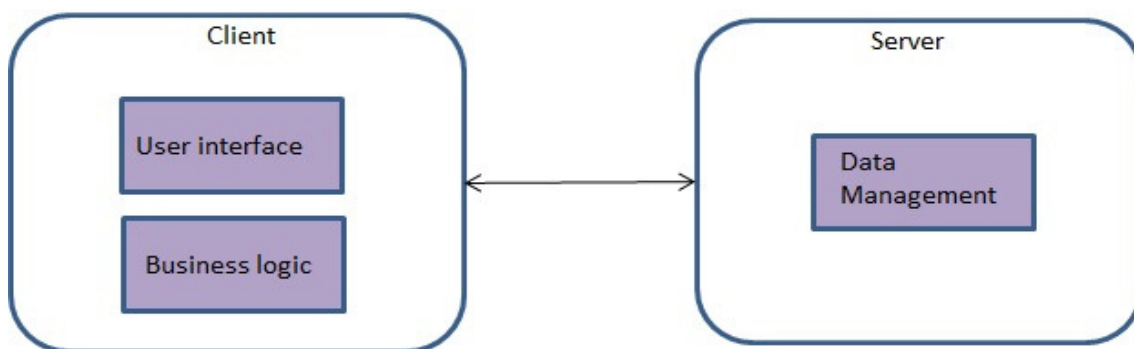


Figure 3.3: Typical Thick-Client configuration

### 3.1.2. Web Server

A web server stands for a set of different *software modules* bend together. Those must be installed onto an internet/intranet host computer to be published into web. A web server handles HTTP requests and retrieves documents as a response. [Jablonski *et al.*, 2011] Figure 3.4 represents the main functionalities of a web server.

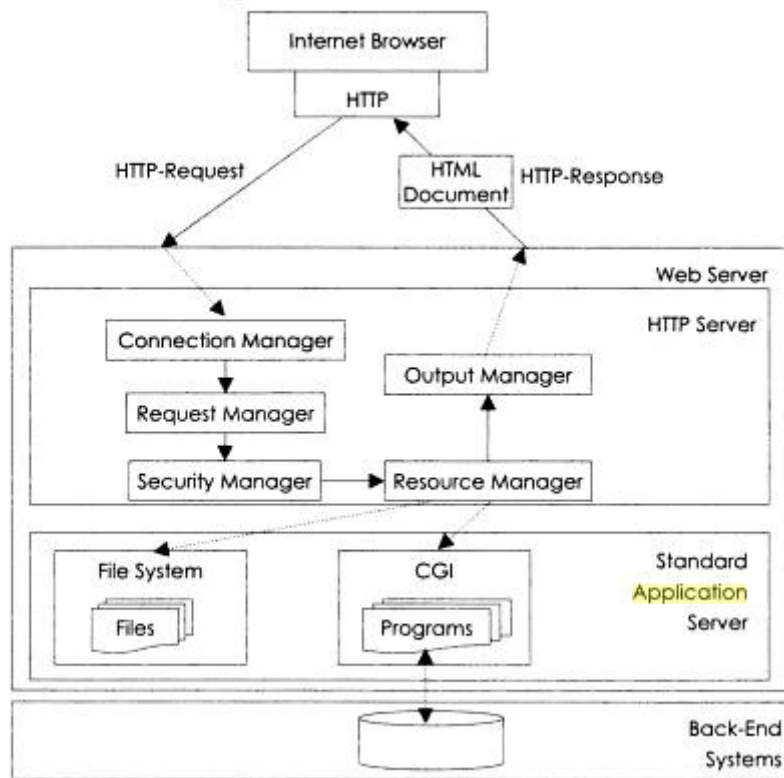


Figure 3.4: Functional structure of a web server [Jablonski *et al.*, 2011]

The user uses a web browser and creates a HTTP-Request to the web server. The web server directs the request to inner HTTP Server where modules *Connection Manager*, *Request Manager* and *Security Manager* lead the request to *Resource Manager*. The *Resource Manager* module is responsible for managing and handling the *File System* and server-side programs using the *CGI technology*. The *File System* and *CGI programs* are laid on *Standard Application Server* layer. *CGI programs* can use *Back-End Systems* like databases to get the requested information. After the requested content is found and collected from the server, the *Resource Manager* sends information to *Output Manager* which is responsible for sending the HTTP-Response message back to the user's web browser.

### 3.2. Data Management

In web applications the data management is usually operated by using a *database*. The database is an integrated collection of logically-related records or files consolidated into a common pool that provides data for one or more multiple uses.

Being responsible for managing updates and allowing simultaneous access from a web server is a part of the database functionality. The database also

provides security functionality, ensures the *integrity* of data and support to important services like backup of the data. Quickness and flexible access are characteristics features of a good database. Most Database Managements Systems (DBMSs) and servers are designed and implemented in a way that software complexity is hidden. [Williams and Lane, 2004] Many of the databases are using some standardization which makes their use congruent.

Importance of a careful database designing cannot be highlighted enough. Specifically designed database enables an effective and comprehensive usage of the stored data. Mistakes done in the designing phase could lead to an irremediable situation, where efficiency is dramatically dropped or even the whole database data might become unusable. For effective use of the database the user must have skills to design a well-structured *database schema* and execute queries using the data definition and manipulation languages.

### 3.2.1. DBMSs

Databases are used for storing the content or metadata of the content. There are a lot of different DBMS types available like Relational, Object-Oriented and Hybrid just to mention few.

Relational DBMS (RDBMS) is based on a relational model of data. Codd [1972] describes that any formatted database is a collection of time-varying relations of assorted degrees. In the Relational model the database is a collection of relations where each relation resembles a table of values or a file of records. In more details, each row in the relational model table typically corresponds to a real-world entity or relationship [Elmasri and Navathe, 2000]. Names of the tables and columns should be descriptive. For example, a table PERSON could have columns like NAME and AGE that present facts about a single person.

Object-Oriented DBMS (ODBMS) integrates database capabilities with capabilities of an object-oriented programming language. ODBMS makes database objects to work like a programming language objects in many existing programming languages. ODBMS extends the programming language with many database capabilities like: *transparently persistent data, concurrency control, data recover* and *associative queries*. ODM is a system that integrates ODBM's capabilities to relational or another non-object DBMSs [Cattell and Barry, 1999].

Object-Relational DBMS is a hybrid solution which has features from both Object-Oriented DBMS and Relational DBMS. The markets for Object-Relational DBMS are estimated to be in between RDBMS and ODBMS.

Figure 3.5 represents the classification that Stonebraker [2003] has defined. With simple data, Relational DBMS is the most suitable for query languages and file system should be used when query languages are not used. With the

more complex data Object-Relational DBMS is the best solution for query languages and Object-Oriented DBMS is the best solution when queries are not used. Stonebraker [2003] admits that the problem with the split is defining when the data is simple and when it is complex. Usually, a developer has to make a decision based on his/her experience, because there are no valid measurements metrics available to help with the decision.

<b>Query</b>	<b>Relational DBMS</b>	<b>Object-Relational DBMS</b>
<b>No Query</b>	<b>File System</b>	<b>Object-Oriented DBMS</b>
	<b>Simple Data</b>	<b>Complex Data</b>

Figure 3.5: A classification of DBMS applications [Stonebraker, 2003]

### 3.2.2. SQL

Relational databases are the most popular and SQL (Structured Query Language) has been established as their standard query language. SQL is based on the relational model. The client sends SQL queries to DBMS server. The DBMS server processes the requested queries and sends answers back to the client.

There are a lot of commercial and open source relational database implementations available. Among open source solutions, MySQL is the most used. This is illustrated in Figure 3.6.

PostgreSQL is also very popular and both of them are capable of challenging commercial databases. Whereas MySQL is known for its speed and ease of use, PostgreSQL is known for its reliability and more comprehensive feature support. In practice MySQL and PostgreSQL have all the features needed in an extensive web application.

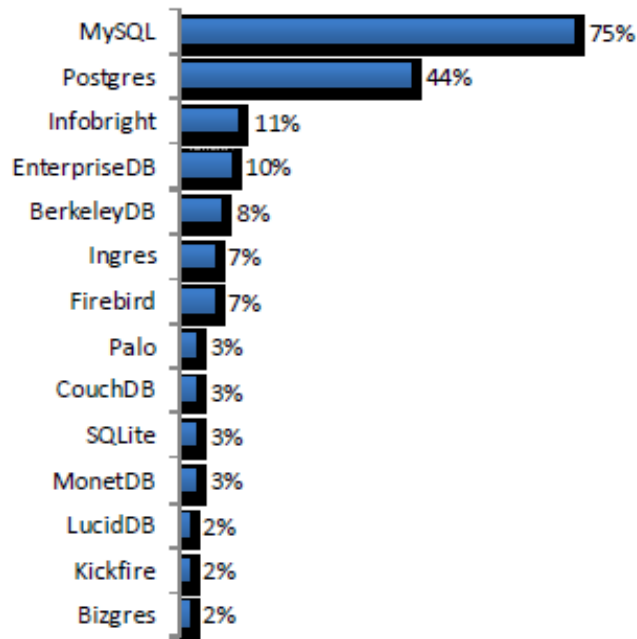


Figure 3.6: The most used open source databases [Madsen, 2010]

### 3.3. MVC

*Model-View-Controller (MVC)* is design pattern which is often used in the end user applications. The MVC pattern is the most used pattern for today's web applications. It has been used for the first time in Smalltalk and then adopted and popularized by Java. Nowadays, there are more than a dozen PHP frameworks based on the MVC pattern [PHP MVC, 2009]. MVC can be seen as a base structure of a modern PHP framework. Understanding the process of the MVC is crucial when using MVC frameworks for web application development. Its main components are [PHP MVC, 2009]:

- The **model** is responsible for managing the data. It is used to store and retrieve entities used by an application from the database. It can also contain some logic implemented by the application that relates to data handling.
- The **view** is responsible for displaying the data provided by the model. In the view the displayed format can be defined by the user.
- The **controller** handles the model and view to work together. The controller receives a request from the client-side and handles it. It requests the model to perform the requested operations and sends the received data to the view.

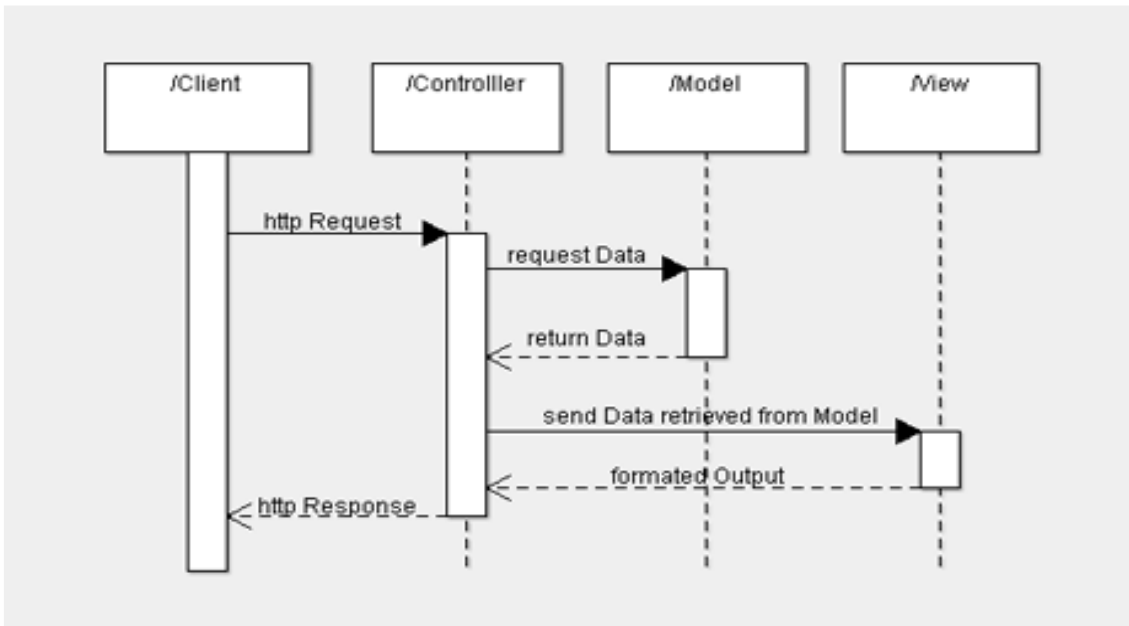


Figure 3.7: MVC explained [PHP MVC, 2009]

Figure 3.7 shows how the MVC pattern works. A process starts when a client application sends a HTTP request to the controller. In the controller the request is usually validated. The controller sends a request to the model to perform the requested operations. The model returns a response which controller module passes to the view. The view represents the data in a defined format and the response is sent back to the client through the controller.

An advantage of the MVC is that its every module has its own responsible area that is easily manageable. Using the MVC pattern makes also code maintaining easier compared to situation where no pattern is in use. A well known problem in the software development is that many software projects does not use any design patterns or architectures. Poorly designed software causes a lot of problems especially in situations where the original developer leaves the project and someone else has to start maintaining the code. In those cases where the MVC (or other standard) design pattern is used, a new developer gets to know the code probably much faster and is capable of adding new features to software faster than without MVC or other standard design pattern.

### 3.4. Programming Languages Used

*Server-side programming languages* perform the requested task on the server-side. Figure 3.8 presents the most used programming languages on the server-side. It is notable that websites may use more than one server-side programming



language. As seen in Figure 3.8, the situation between the top 7 technologies has not changed much in a couple of years. PHP is by far the most used server-side programming language and its usage is still increasing. ASP.NET is the second most used technology, but its popularity is decreasing. Only 4.1% of the websites use Java as a server-side programming language, which is not much compared to Java's popularity on the other platforms. ColdFusion, Perl, Ruby and Python are not much used technologies and none of them has increased its popularity.

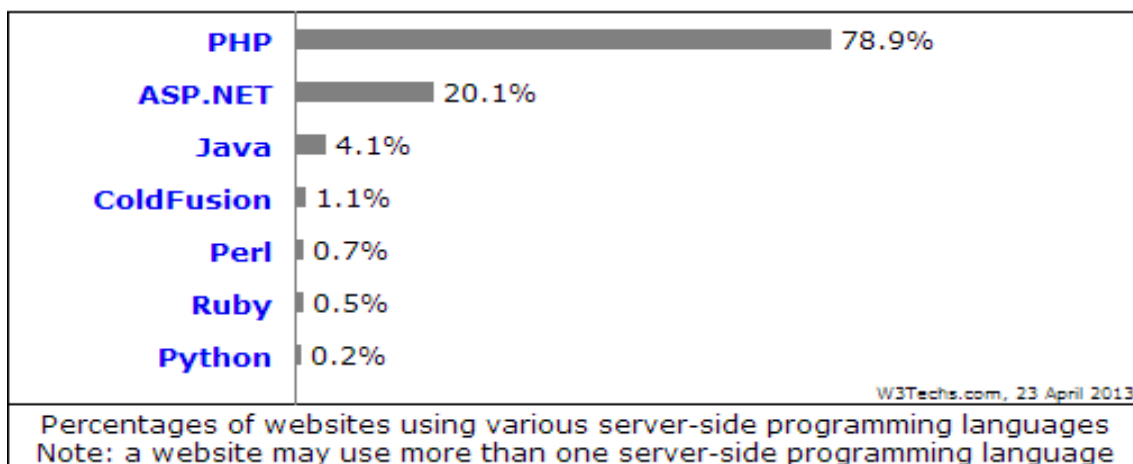
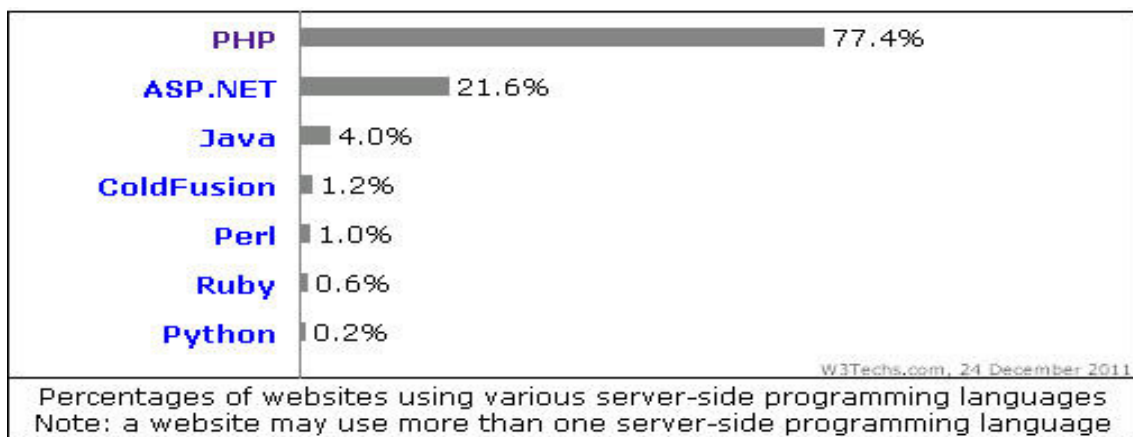


Figure 3.8: Server-side programming language usage statistics 2011 and 2013 [W3techs 2, 2011-2013]

The client-side is responsible for executing tasks only its own side. As Figure 3.9 shows, 10.9% of the websites does not use any of the *client-side programming languages*. In these cases the websites are usually very simply. JavaScript is used for most of the websites with 88.8% share and Flash has 18.4% share of the websites. It is notable that a website may use more than one client-side programming language. [W3thechs, 2013]

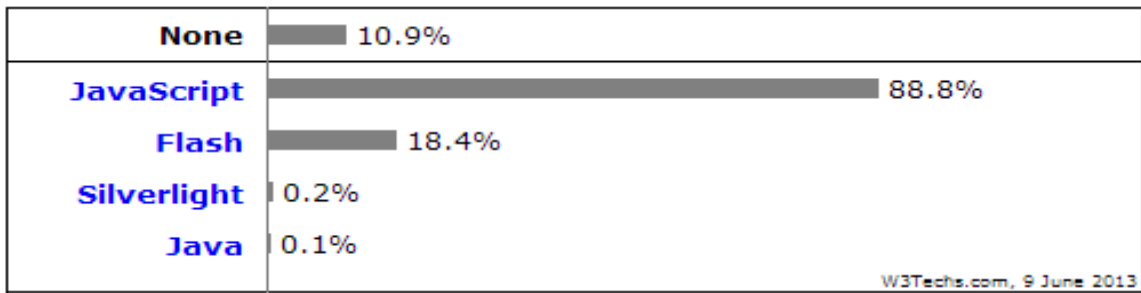


Figure 3.9: Client-side programming languages [W3techs, 2013]

### 3.5. Ajax

Ajax (Asynchronous JavaScript and XML) is a technique that helps creating fast and dynamic web pages. Ajax allows web pages to be updated *asynchronously* by exchanging data with the server. Ajax makes possible to update parts of a web page without reloading the full page. [W3schools, 2011]

Ajax took web applications into new level. It can be said that Ajax opened new ways of providing content on the web applications. Earlier it was difficult to provide much information at once because an application could not react to user inputs effectively. With Ajax, a single web application page can provide the same information that earlier needed page reloading and multiple pages.

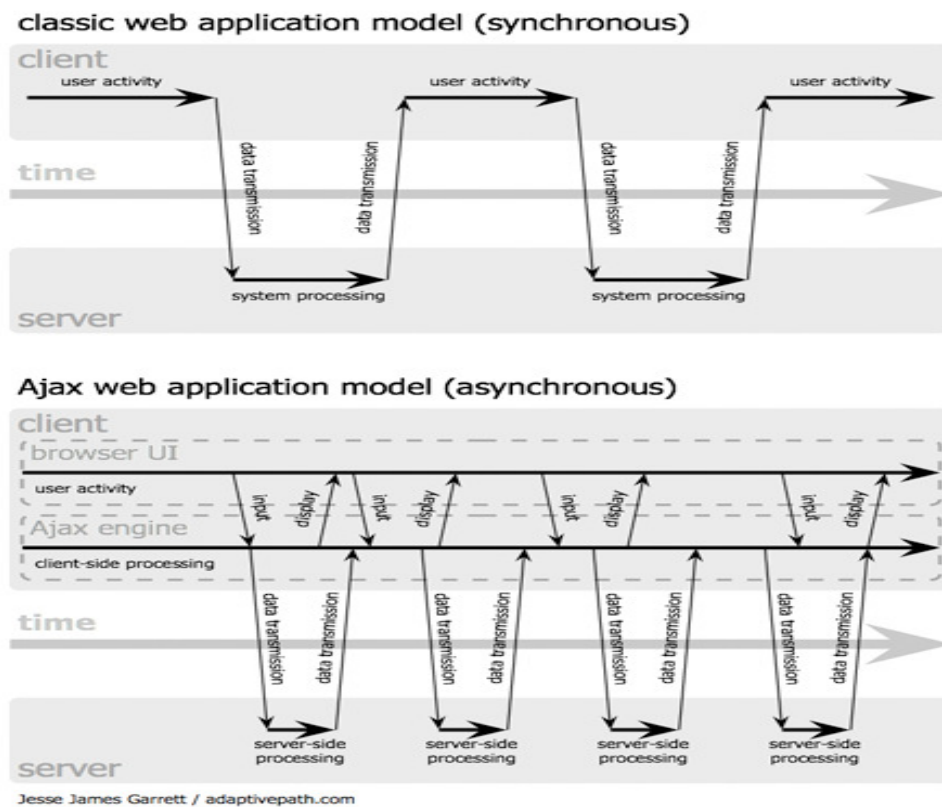


Figure 3.10: Classic and Ajax web application models [Spahr, 2011]

Figure 3.10 shows the differences between a classic web application model and an Ajax web application model. In the classic model all the requests are synchronous, which means that the request execution process blocks the application usage until the request has been performed. Further, in the classic model the client must refresh the whole page to get the new data into use from the server-side.

The Ajax web application model works asynchronously, which means that it is not blocking the functionality of the application when awaiting a response from the server-side. When a user activity creates a request to the server-side, it is directed to the Ajax engine which is responsible for communicating with the server-side. In this way the application keeps working as usual and only the necessary parts of the web UI will be updated, when the response has been received from the server-side through an Ajax engine layer.

## 4. PHP FRAMEWORKS

In the context of the present study, PHP (Hypertext Preprocessor) was chosen for server-side programming language on multiple reasons. As seen from the Figure 3.8, PHP is clearly the most used server-side programming language. It is also very well documented. I also had a good history with PHP and I have noticed that it is really powerful and effective language for creating an extensive web applications.

PHP is a widely-used scripting language for web development. The goal of the language is to allow web developers to write dynamically generated pages quickly [PHP, 2010]. PHP was designed to resemble C in structure, which makes PHP an easy adoption for developers who knows C, Perl, and other similar languages [PHP, 2013].

PHP is mainly focused on server-side scripting, but PHP can also been used for writing desktop applications. Since graphical user interfaces are not the main focus area of the PHP language it is not much used for that purpose. PHP works on all major operating systems, including Linux, many UNIX variants (including HP-UX, Solaris and OpenBSD), Microsoft Windows, Mac OS X and RISC OS. A support for most of the web servers today contains Apache, IIS, and many others is also provided. Further, the support covers all the web servers that can utilize the *FastCGI PHP binary*, like *lighttpd* and *nginx*. PHP can be executed as either as a CGI processor, or a module. [PHP, 2010]

Many people links PHP automatically with *HTML*. It is true that it is a very much used web application solution where HTML + CSS create the web UI and HTML files has embedded PHP scripts for processing the requested actions on the server-side.

A web application framework is a set of tools which helps developers to build a web application. These frameworks typically provide core and basic functionalities for the web application. The commonly provided functionalities are such as *user session management*, *templates* and *data persistence*. When selecting a framework which functionalities are suitable for the designed web application, a developer can often save a significant amount of time when building a web site [DocForge, 2010]. Another great advantage is that a user does not have to implement same features again for the different web applications when they both can use the same features provided by the framework. Many of the frameworks provide also scripts for integration database to the web application.

Next we introduce on features of the PHP frameworks. The section also provides some comparison information between the most used frameworks and pure PHP.

#### 4.1. Features

Many of the PHP Frameworks share the very same kind of feature set that is found useful for several kinds of web applications. The features area a stone base for the application. This does not mean that it is not possible to develop a great web application without using some framework. Next let's have a closer look of the main functionalities of the PHP frameworks [PHP 2, 2013].

- **PHP5** support is the core feature of the PHP framework. Support for PHP4 could also exist, but nowadays PHP5 is the main version used and that is why it is supported by all the modern PHP frameworks.
- **MVC** indicates whether the framework has an inbuilt support for a Model-View-Controller. See Section 3.3 for more information about the MVC.
- **Multiple DBs** indicates whether the framework supports multiple databases configuration. Without this feature, the use of the multiple DB connections at the same time and for same data can be a very complicated task.
- **ORM** (Object-Record-Mapping) is a technique that enables converting data between incompatible type systems in object-oriented programming languages.
- **DB Objects** indicates whether the framework provides DB records to be able to handle as objects in object-oriented programming.
- **Templates** are a pre-developed page layouts. The most common templates are different forms.
- **Caching** indicates whether the framework includes a support for caching the data. Caching provides an effective data handling. Since a server can be in heavy traffic it is wise to try to use caching for help. For example caching of database queries helps server to handle more requests.
- **Validation** is a very useful feature for confirming that the content sent to the server-side from the client application is what it is required to be.
- **Ajax** is a technique for creating fast and dynamic web pages. See Section 3.6 for more information about Ajax.
- **Authentication Module** indicates whether the framework has a build-in support for authorization. Authorization is a process of giving access to system, usually based on a username and password. Authorized access systems are more and more used on web applications. Since the web applications are becoming more and more complex and extensive the authentication of the user gives much benefit which could be hard to develop and manage without any framework.

- **Modules** are a group of independent MVC elements. The use of modules allows for re-usability and encapsulation of the code.









PHP Framework	PHP5	MVC	Multiple DB's	ORM	DB Objects	Templates	Caching	Validation	Ajax	Auth Module	Modules
<a href="#">Akelos</a> 	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
<a href="#">CakePHP</a> 	✓	✓	✓	✓	✓	-	✓	✓	✓	✓	✓
<a href="#">CodeIgniter</a> 	✓	✓	✓	-	✓	✓	✓	✓	-	-	-
<a href="#">Prado</a> 	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
<a href="#">Seagull</a> 	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
<a href="#">Symfony</a> 	✓	✓	✓	✓	✓	-	✓	✓	✓	✓	✓
<a href="#">Yii</a> 	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
<a href="#">Zend</a> 	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Figure 4.1: Supported features of some of the most used PHP frameworks  
[PHP 2, 2013]

Figure 4.1 represents supported features of the most used PHP Frameworks. Support for PHP5, MVC, multiple databases, DB objects, caching and validation can be found from all the listed PHP frameworks. ORM, Ajax, authentication and modules are supported in seven of the eight listed PHP frameworks. Template is the least supported feature among the listed frameworks. Overall, it can be seen that the features are very similar among the most used PHP frameworks.

Comparing to pure PHP, the PHP frameworks provide quite impressive range of useful features. It depends much on what the web application is required to do to decide whether to use a framework for developing the web application or not. For example, secure authentication and ORM are usually quite demanding tasks to be implemented by a pure PHP. A feature like validation could save a lot of time from the developer, because the needed rules and functionalities are not a small task to be implemented.

## 4.2. Performance

When the PHP frameworks share the very same kind of features, the performance of the PHP framework could make the difference. Many of the PHP framework are claimed to be the fastest framework available by their developers. High performance is a keyword for almost every framework. Many frameworks share on their web pages *benchmark charts* showing that their own framework is usually claimed to be the best available. There are not many neutral and extensive comparison data available. Figure 4.2 represents Apache

benchmark of requests per second for the most used PHP frameworks. Yaf is the fastest and Zend Framework is by far the slowest framework in the list. Zend's position is quite awkward because Zend's another project *Zend Engine* interprets PHP. Probably the best known frameworks Yii, CodeIgniter and Symfony are also really slow compared to top-5.

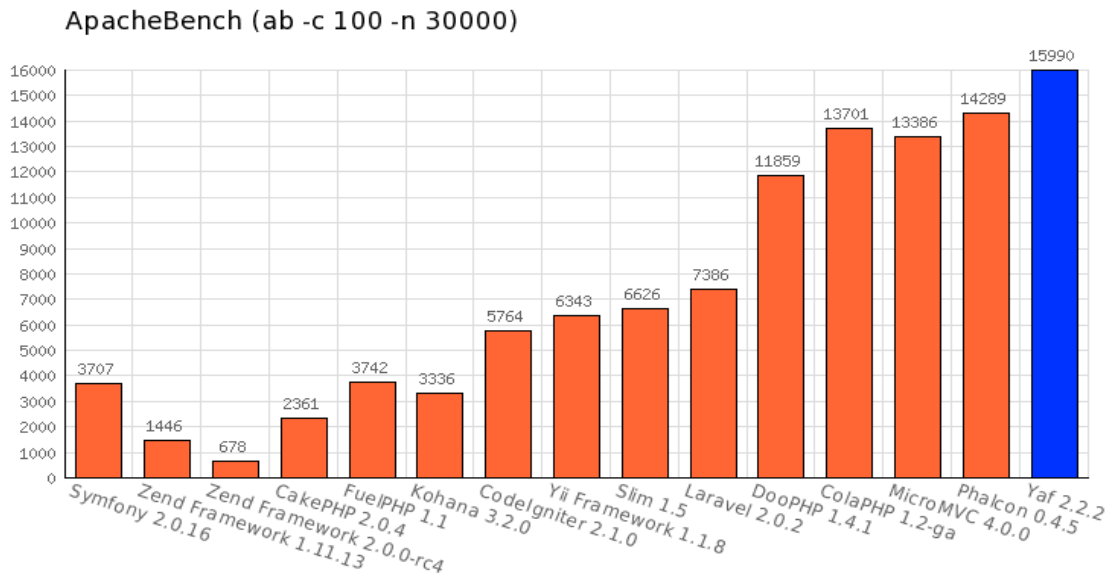


Figure 4.2: Performance benchmark of the PHP frameworks [Ruilog, 2013]

PHP frameworks use of system memory has been calculated in Figure 4.3. The less memory is used for a task, the better. Yaf is again on top of the comparison, whereas Zend seems to be by far the lowest quality framework in use of memory. The results for Zend are incredibly bad, which makes the reliability of the benchmark a bit questionable. Compared to results in Figure 4.2 it seems that the order for frameworks is quite the same.

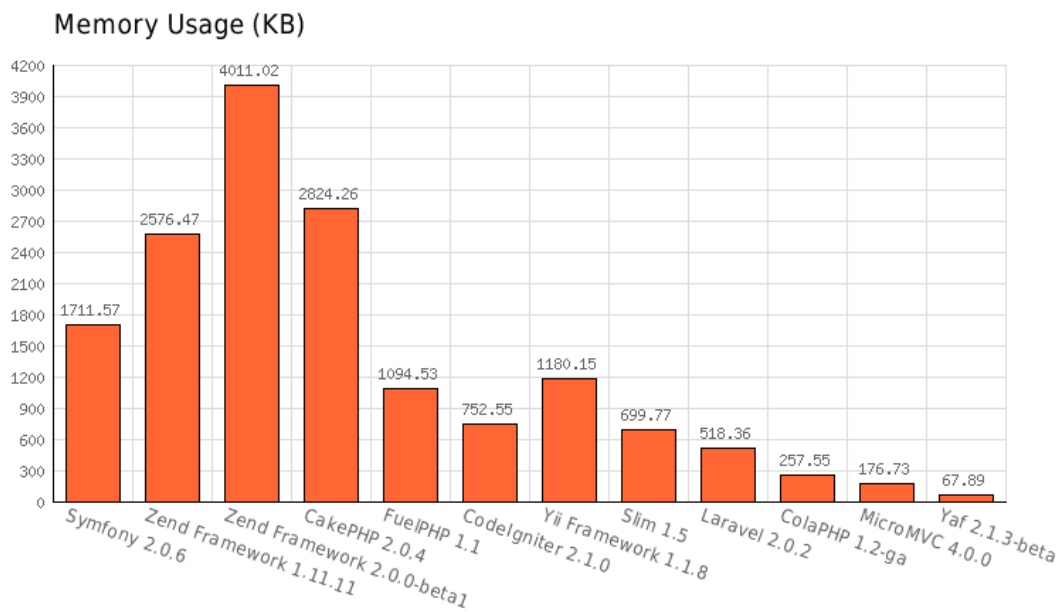


Figure 4.3: Memory usage of the PHP frameworks [Ruilog, 2013]

Below are the results from [Shelmandu, 2010] PHP MVC framework performance benchmark:

- Raw PHP – 740 req/sec – 100%
- PHP Pro MVC 0.0.4 – 200 req/sec – 27%
- DooPHP 1.2 – 170 req/sec – 23%
- Yii Framework 1.1.1 – 130 req/sec – 18%
- Kohana PHP 2.3.4 – 55 req/sec – 7.5%
- CodeIgniter 1.7.2 – 38 req/sec – 5%
- Zend Framework 1.10 24 req/sec – 3%.

Results from Shelmandu [2010] are not very extensive as only six different PHP frameworks are taken into comparison. The thing that makes the results interesting is the results of the raw PHP. Raw PHP seems to be able to perform requests per second almost four times faster than the best PHP framework available. It is not a surprise that pure PHP works faster than a framework, because the framework has to make much more function calls because of its architecture structure.

Raw PHP will lose the benefit if the developer starts adding more features. The more features, the more performance will decrease; and sooner or later the developer finds out that he/she has implemented many of the features that would have been found from any of the PHP framework. Overall, pure PHP has better performance, but there is a risk that the web application will get more complicated than first thought and the situation turns upside down. Basic rule



is that, the more complicated web application, the more advantage you will gain from the PHP framework.

## 5. SYMBOLDATABASE

In this chapter, a web application solution called *Symboldatabase* is described. Symboldatabase is a web application running on a server that is used for storing and distributing symbol files. The main reason for starting Symboldatabase project was that there was a need for centralized storing and distribution system for symbol files. The symbol files are needed for Symbian crash *decoding*. Symboldatabase focuses on solving the problems described on introduction and data dispersion section. A centralized solution that stores all the available symbol files is needed to make the use of the symbol files more easy and effective. Compared to Microsoft's<sup>1</sup> and Mozilla's<sup>2</sup> symbol servers, Symboldatabase is designed to be more user friendly by providing a user web UI , extensive *interfaces* and *web services* to satisfy the needs of the different use cases.

This thesis focuses on those parts of Symboldatabase which handle data collection, storing, and distribution by using a web application. Actually Symboldatabase is more than just these components, but describing the whole system is not relevant in this thesis. Figure 5.1 represents the overall design of Symboldatabase.

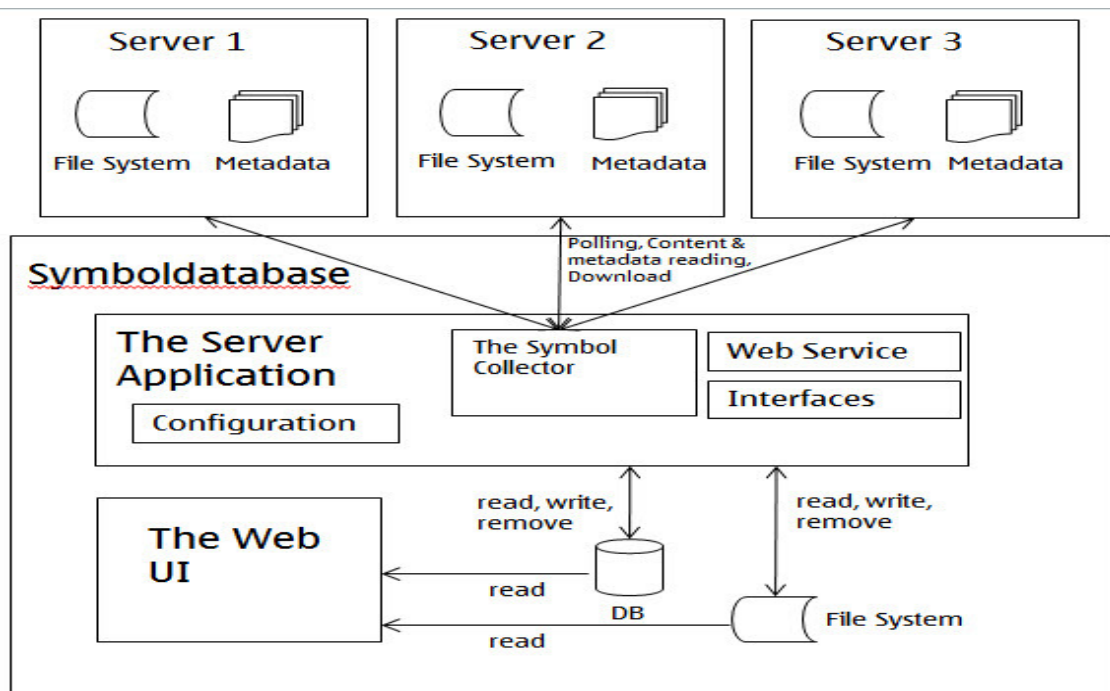


Figure 5.1: Overall design of Symboldatabase

<sup>1</sup> [http://msdn.microsoft.com/en-us/library/windows/desktop/ms680693\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/desktop/ms680693(v=vs.85).aspx)

<sup>2</sup> [https://developer.mozilla.org/en-US/docs/Using\\_the\\_Mozilla\\_symbol\\_server](https://developer.mozilla.org/en-US/docs/Using_the_Mozilla_symbol_server)

Symboldatabase contains two main parts the Server Application and the web UI. The Server Application is the only component of the Symboldatabase that can read, write or remove the stored data whereas the Web UI can only read the information from the database and file system.

The Server Application has three different components. *The Symbol Collector* is responsible for collecting the symbol files from the different servers. The Symbol Collector polls new content periodically from the servers and then uploads new data and metadata to Symboldatabase. The web service provides communication between Symboldatabase and some other system inside the intranet by using the network. The interfaces are views that can be called with a HTTP request to get information about Symboldatabase stored data. The web UI is responsible for providing the symbols to users.

Section 5.1 describes the use cases that were found. Section 5.2 lists all the requirements which are required from the web UI and database. Section 5.3 describes the chosen technologies and the reasons why those were selected. In Section 5.4 the architecture of the system is presented such that each subsection represents a subpart of the system. Section 5.5 presents shortly how the symbol files are collected. Section 5.6 describes the web UI for the users and Section 5.7 focuses on the interfaces that Symboldatabase provides. Finally, Section 5.8 presents the database used in Symboldatabase.

## 5.1. Use Cases

Symboldatabase is focused on three main use cases:

1. A developer checks what files are available for the specific ROM ID

Table 5.1: Use case 1

<b>Actors</b>	Developer, Symboldatabase web UI
<b>System Boundary</b>	Developer PC
<b>Pre-Conditions</b>	Symboldatabase is operational.
<b>Basic Flow</b>	Developer provides ROM id values as an argument quick search. The web UI provides information about collectionsets found which match with the given ROM id value.
<b>Alternate Flow</b>	ROM id value is not valid; ROM id cannot be found from the Symboldatabase.
<b>Post-Conditions</b>	Desired output has been produced.

2. Developer wants to download some specific symbol files from Symboldatabase

Table 5.2: Use case 2

Actors	Developer, Symboldatabase web UI
System Boundary	Developer PC
Pre-Conditions	Symboldatabase is operational.
Basic Flow	Developer uses filtering options to find the correct collectionset. Developer opens collectionset view and selects files he/she wants to download and presses download button.
Alternate Flow	Desired collectionset cannot be found; Desired files are not part of the collectionset.
Post-Conditions	The desired files have been downloaded.

3. Crash Decoder wants to use symbol files from Symboldatabase

Table 5.3: Use case 3

Actors	Developer, Crash Decoder, Symboldatabase
System Boundary	-
Pre-Conditions	Symboldatabase is operational.
Basic Flow	Developer uses Crash Decoder to decode a crash file. Crash Decoder uses Symboldatabase HTTP GUI interface for getting information of the symbol files.
Alternate Flow	Requested content cannot be found.
Post-Conditions	Requested data has been found and used straight from Symboldatabase.

Table 5.1 represents the first use case where a developer wants to check what files are available for the specific ROM ID. In this use case a developer is willing to know if Symboldatabase has the symbols that match with the provided ROM IDs. The developer browses the web UI and seeks for information about the *collectionsets* matching the given ROM ID values. The user inputs the values and Symboldatabase provides a list of available files to the developer. *Alternative Flow* is that Symboldatabase does not have the matching symbols and it informs the developer about it.

Table 5.2 presents the second use case where a developer wants to download some symbol files from Symboldatabase. The developer opens SUI

and uses filtering options to find the matching collectionset. The developer wants to download only a couple of symbols, not the all available. The developer uses download functionality to get the symbol files on his/her local PC. Alternative Flow is that a desired collectionset cannot be found or the needed symbol files are not included to the collectionset.

In Table 5.3, the third use case is described. In this use case a tool called Crash Decoder wants to use symbols files provided by Symboldatabase. This is the only use case where actors include some external client tool. Basic Flow for use case is that the Crash Decoder needs proper symbol files to be able to decode a Symbian crash file. Crash Decoder needs to send a HTTP request to Symboldatabase server to receive information of the symbol files. Alternative Flow is that the requested content cannot be found from Symboldatabase and Crash Decoder cannot perform the decoding process properly.

## 5.2. Requirements

In this section the general and functional requirements for the Symboldatabase web application and database are described. The main requirements for Symboldatabase are to collect, store and distribute the specific data. This thesis focuses only for the web application and database of Symboldatabase. Since the content collection requirements are not listed here.

Subsection 5.2.1 focuses on the general requirements for the web application. These requirements are quite common web application requirements. Also functional requirements are not anything new to web applications since content browsing, searching and downloading can be listed as basic functionalities for the web application.

Section 5.2.2 describes the general and functional requirements for the database. The database solution of Symboldatabase must provide easily stored content which can be related to each other.

### 5.2.1. Web Application

Below the general and functional requirements for Symboldatabase web UI (*SUI*) are listed.

#### **General:**

##### 1.1 Accessibility

The SUI must be accessible to all Nokia intranet users. All the functionalities are freely accessible and there is no need for user permissions.

## 1.2 Scalability

The SUI must be easily expandable to handle new kind of content. All the content must be presented in a way that supports effective use of the data.

## 1.3 Performance

Accessing the SUI must be nice and smooth and changing between the views should work without any notable delay. Loading the requested data from the server-side should not take more than one second per request.

### **Functional:**

#### 2.1 Content searching

The SUI must provide a search functionality that makes content finding easy and effective. Quick search functionality for ROM ID must exist in all views.

#### 2.2 Content usage

There must be a GUI that enables a client application to request the content easily without any UI browsing from Symboldatabase. The GUI must be dynamic and provide always the latest information about the requested content.

#### 2.3 Content browsing

Content of Symboldatabase must be easily browsed with filtering options and a tree view. Filtering must provide the following options: *Product Family, Release, Product, Category* and *Date*.

The tree view for filtering must present the content of Symboldatabase in a logical hierarchy order of the content:

- Product family, Release, Product, Category and Collectionset.

#### 2.4 Content downloading

Symboldatabase must provide functionality for file downloading. Each file in Symboldatabase must be downloadable and the SUI should provide single- and multi-file downloading functionality.

#### 2.5 Content metadata

All metadata available about the selected content should be shown in the content views. Metadata must be shown for releases, collectionsets and single symbol files.

## 5.2.2. Database

Next, the general and functional requirements for database of Symbol database are described. The database is needed for data management and it must be fast and reliable.

### **General:**

#### 3.1 Scalability

Content stored in a database must have good scalability. The structure of the database schema and tables must be easily expandable.

#### 3.2 No duplicate data

Database must avoid of storing duplicate records. This keeps the size of the database as small as possible.

#### 3.3 Performance

Database queries must have a reasonable response time. Indexes should be used for tables to make the queries work faster.

#### 3.4 Maintenance

Database must be easily maintainable. The structure should be kept as simple as possible.

#### 3.5 Backup

Backup of the stored data is vital for the database. Backup from the database must be taken at least once a week and needs to be stored to trusted location.

#### 3.6 Suitability

Selected database paradigm must work easily and effectively with the selected web application technique(s). Connection establishment and the needed queries should be easily implemented into code.

### **Functional:**

#### 4.1 Collectionset

The name of the Collectionset must have a ROM ID, category and release information. This gives a good description about content of the collectionset.

#### 4.2 File

Files must have filetype, uploader and filepath information. All files stored into server must have the proper metadata stored into database and must be related to at least one collectionset.

#### 4.3 IAD file

IAD files must store name of the component, UID and version information. This enables proper identification for the IAD file.

#### 4.4 Filetype

A Filetype table where only the allowed filetypes are stored is needed. Every file record must be related to some filetype record.

#### 4.5 Uploader

All files and collectionsets which are added to database must have content uploader information stored.

#### 4.6 Used count

All collectionsets and files must have a store counter value for utilizing statistics. Those statistics give valuable information about which files are used and which are not.

#### 4.7 Download count

All collectionsets and files must store the download counter value. This value is valuable information for statistics.

### 5.3. Techniques

This section shortly presents the techniques that were chosen for the implementation of Symboldatabase. These techniques were chosen based on the background research. Each of these selected techniques supports each other and as a unit they fill the needs described in requirements.

#### **Yii PHP Framework**

The functionalities of server-side web UI were implemented with PHP and the MVC framework called Yii. Yii Framework provides basic functionalities for the Web UI. Yii Framework is a component-based PHP framework for developing web applications. Yii is an acronym for “Yes It Is!” [Yii, 2010].

Yii Framework was selected based on three different reasons. Firstly, I had some earlier experience working with the framework. Secondly, Yii Framework contains all the needed features (see Figure 4.1) that are needed to build an extensive web solution. Third reason was that the documentation of the framework is probably the best available. Creating a new project like Symboldatabase was very easy thing to do, thanks to great tutorials and example projects.

Yii Framework provides also functionalities for creating a web UI with its own components. For Symboldatabase I noticed that it still did not provide all



the needed components I had in my mind. So, I had to investigate what JavaScript could offer to meet the requirements for the web UI.

## **EXT JS**

Ext JS is a cross-browser JavaScript library for building web applications. Ext is available with Commercial and Open Source licenses. [Sencha, 2010]. Ext JS is a client-side web application and it is claimed to be very fast and extensive.

Ext JS was chosen to Symboldatabase, because it provides very good functionality to build a layout as described on the requirements. I had also experience from Ext JS on earlier projects and I liked Ext JS's style of presenting the data. Ext JS application is like desktop software, but it only ran as a web application. It does not try to be fancy or really nice looking. Actually, the layout is quite reduced, but that is a part of the plan. The main purpose of Ext JS is to provide versatile components with great efficiency.

## **Data Management**

For the database application MySQL was selected because of multiple reasons. MySQL is an open source relational database and it meets best the set requirements for the database. MySQL is well supported by the Yii Framework and it provides a great performance. MySQL is also very easy to use and maintain. Further, one of the requirements was the database backup must be performed at least once a week. MySQL provides easy to use tools for taking backup dumps from the database.

## **5.4. Architecture and Design**

Usually architecture and design are made before choosing the actual techniques for the implementation. In PHP Frameworks this is not the case since to be able to choose such a framework the architecture and design has to match how the framework works. Main application for Symboldatabase is created by using the Yii PHP Framework. In the following sections the structure of Symboldatabase application is described.

Symboldatabase can be seen as a single Yii Framework project that can be split into two different components. The components are the Server Application and the web UI. Because of the structure of the Yii Framework project this split is not always so clear. Both components share many functionalities, configurations and resources.

Server Application consist most parts of Symboldatabase. Actually the Server Application is not a single application but a collection of different components which are used to collect, store and distribute symbol files.

#### 5.4.1. Structure of Yii Framework Application

To be able to fully understand how the Yii Framework works, knowledge about MVC design pattern is needed. In Figure 5.2, the basic structure of a Yii based application is presented. *Index.php* is an access point to web UI from where the whole structure of an application is created. The application may have relations also to other application components. The application is structured based on MVC design pattern. A controller handles that a model and a view work together and directs the data from one component to another. Widgets can be used to intensify the application.

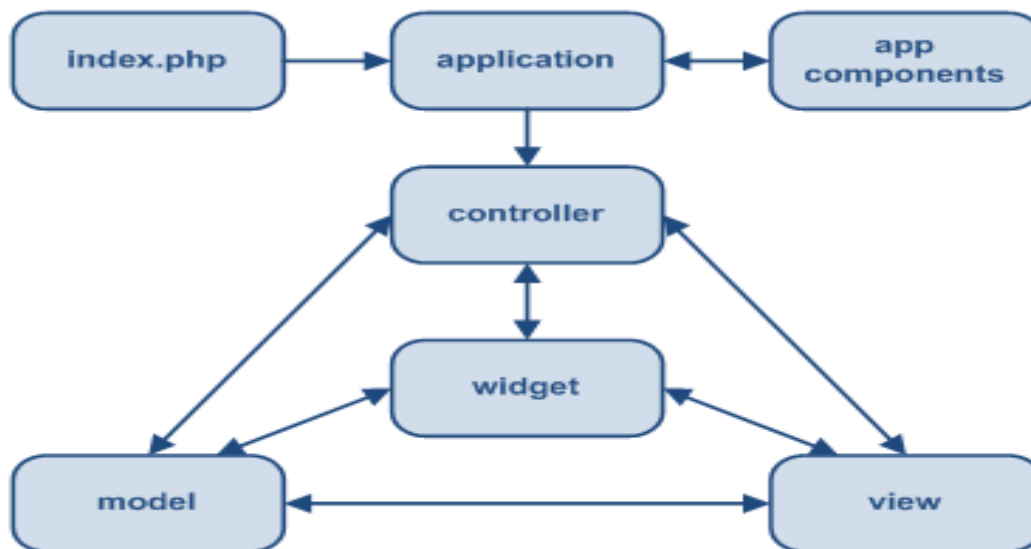


Figure 5.2: The structure of Yii Framework application

#### 5.4.2. Commands

Commands are individual parts of a Yii Framework project. The commands are command-line executable objects that are created to execute some task from the command-line. The only command that Symboldatabase uses is the Symbol Collector (Table 5.4). The command files are usually designed to be executed as a scheduled task.

Table 5.4: Commands of Symboldatabase

Command name	Description
Symbol Collector	The Symbol Collector is responsible for collecting new symbol files. See Section 5.6 Symbol Collector.

### 5.4.3. Components

Components are also individual parts of a Yii Framework project like the commands. Components are normally used to process a certain tasks like providing an HTTP interfaces or LDAP authentication to a web UI or web service. Many components are created to be used as a helper commands to process some functionality like file moving. Table 5.5 represents the components that are used in Symboldatabase.

Table 5.5: List of Symboldatabase components

Component name	Description
ContentUsage	See Subsection 5.7.1 Content Usage
IADUsage	See Subsection 5.7.2 IAD Usage
LDAPUser	An LDAPUser class provides LDAP authentication functionality to web UI.
UserIdentity	A UserIdentity class provides basic user identify functionality to such as username based authentication.
Setting	A Setting component provides helper functions for managing setting.

### 5.4.4. Config

Configure settings for the web UI and console applications are described in Table 5.6. Configurations are centralized to single file to make the configuration easily maintainable.

Table 5.6: List of config files

File	Description
main.php	A configurations file for the SUI. In this file variables, set paths, define database connection and other needed configurations are defined.
test.php	A configurations file for the unit tests.

### 5.4.5. Controllers

Controllers are used to execute server-side functionalities. Since JavaScript works on the client-side, the SUI has to send requests to server-side to get information from the database. Requests are handled by the controllers, to be described below.

All of following classes works individually and sends responses to JS classes which sent the requests. All method names have the word *action* as a prefix. All responses sent back to JavaScript are in the *JSON* format. This is how JavaScript can handle the response information. Controllers can also send just prints as a response value but it is not recommended because requester does not always know how to handle the response. With JSON the response format is always the same.

The naming of controllers indicates what action will be provided. For example *actionList()* method of *CollectionSetController* class provides a list of Collectionsets with the given parameters. In Figure 5.3 all controller classes with methods and member variables are shown. The purpose of each class is described below.

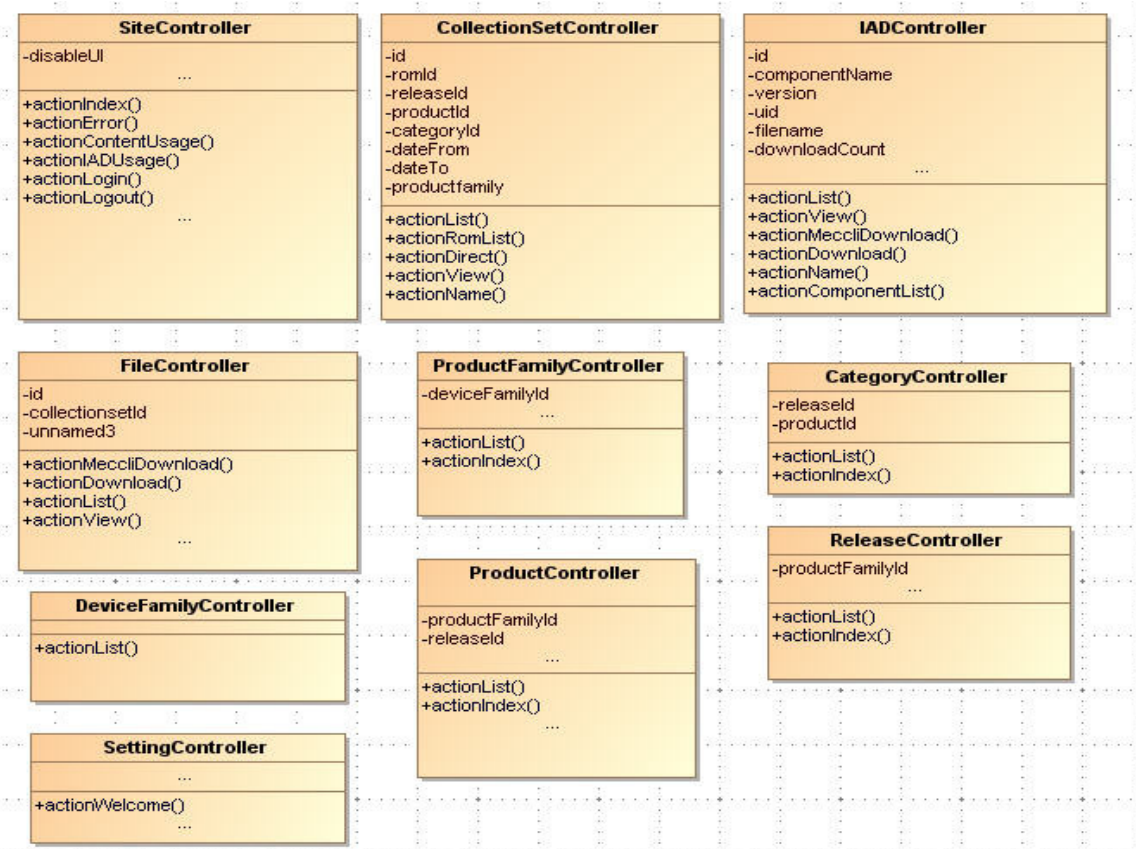


Figure 5.3: Controller classes

### The server-side controller classes:

- **SiteController** handles site changes and authentication for the web UI. All functionality related to overall usage of the web UI are also located here.
- **CollectionSetController** is responsible for providing collectionset related information from the database. This class provides functionality for getting a list of all stored collectionsets, but also more specific information about a single collectionset.
- **IADController** handles all requests regarding IAD content. This class provides IAD file information and also single IAD file downloading from the server.
- **FileController** handles all actions related to stored files. For example, `actionDownload()` method handles file downloading from the server. This class provides also information related to file.
- **ProductFamilyController** provides productfamily information from the database to the web UI.
- **CategoryController** provides information about the categories used in Symboldatabase.

- **DeviceFamilyController** provides devicefamily information from the database.
- **ProductController** handles all requests related to products information.
- **ReleaseController** handles all requests related to release information.
- **SettingController** handles settings stored into Symboldatabase. Currently only welcoming message is handled via this controller.

Figure 5.4 represents *actionName()* method in the Collectionset controller class. Based on the MVC-model, the controller is responsible for handling that the model and view work together.

```

/**
 * Returns name of the collectionset
 */
public function actionName()
{
    $id = 0;

    try {
        //Check that GET parameter is set
        if (isset($_GET['id'])) {
            $id = $_GET['id'];
        } else {
            echo JsonResult::failure(array('message' => 'Cannot process
            the request. Id value must be set'));
        }

        $Criteria = new CDbCriteria();
        $Criteria->select = 'name';
        $Criteria->condition = "id = $id";

        //Use the set criteria to find the name of the collections
        $result = CollectionSet::model()->findAll($Criteria);

        //Check the results
        if (count($result) > 0) {
            //Name was found.
            echo JsonResult::failure(array('name' =>
            $result[0]['name']));
        } else {
            //No collectionset found with the given ID
            echo JsonResult::failure(array('message' => 'No
            collectionset available for the requested id'));
        }
    } catch (CException $e) {
        echo JsonResult::failure(array('message' => $e->getMessage()));
    }
}

```

Figure 5.4: Controller method *actionName()*

In this method, a client-side provides *id* of the collectionset and it is sent as a *GET parameter* to a server-side controller. The received GET parameter is validated and a database criterion is build. It is notable that the actual query is formulated by using the Collectionset model. A developer does not have to write any SQL code to the controller to search content from the database. After the model has provided the results, an HTTP response message in JSON-format is created.

### 5.4.6. Models

Models are used to keep data and their relevant rules. A model represents a single data record in the database. It means that there is a model class for every table in the database. Models can also have functions for handling and storing the data in the database. Models can also be used to create relations between different database tables which makes possible to use data from multiple tables, without having to make additional queries.

Models define rules for the stored content. Figure 5.5 represents method *rules()*, which is used in the File model. All these rules are related only to user inputted data, meaning that this is a part of the validation functionality.

```
/**
 * @return array validation rules for model attributes.
 */
public function rules()
{
    // NOTE: you should only define rules for those attributes that
    // will receive user inputs.
    return array(
        array('filename, filetype_id, uploader_id', 'required'),
        array('filetype_id, langpack_id, uploader_id, used_count,
            download_count', 'numerical', 'integerOnly'=>true),
        array('filename', 'length', 'max'=>255),
        array('filepath', 'length', 'max'=>255),
        array('md5', 'length', 'max'=>50),
        array('description', 'length', 'max'=>200),
        // The following rule is used by search().
        // Please remove those attributes that should not be searched.
        array('id, filename, filepath, filetype_id, langpack_id, md5,
            description, upload_date, uploader_id, latest_download,
            used_count, download_count', 'safe', 'on'=>'search'),
    );
}
```

Figure 5.5: Validation rules defined in the File model

The *rules()* method contains the number of arrays that specifies what rule is used for which column in the related table. For example *required* is a rule which is defined for columns: *filename*, *filetype\_id* and *uploader\_id*. Required means that the defined fields are mandatory and they must be set to be able to add a new record for the File table.

### 5.4.7. Logging

Yii Framework provides logging for the web UI and console application (Table 5.7). Both files are stored under `\runtime` folder. Logging is not enabled by default and all the configurations related settings are located in the `main.php` file.

Table 5.7: Logging

File	Description
application.log	Log for SUI actions: The level of the logging can be set from the <i>main.php</i> config file. Default levels are <i>error</i> , <i>warning</i> , <i>trace</i> and <i>info</i> .
console.log	Log for Console actions: The level of the logging can be set from the <i>main.php</i> config file. Default levels are <i>error</i> , <i>warning</i> , <i>trace</i> and <i>info</i> .

#### 5.4.8. Views

A view is a PHP script consisting mainly of user interface elements. Table 5.8 presents the views that are needed for Symboldatabase. The layout of the SUI is implemented by using a JavaScript framework. That is why the count of views is kept in minimum and only the needed interfaces are created by using Yii Framework views.

Table 5.8: Views of the web UI

File	Description
layout/main.php	The most important file for the Yii Framework web application. This is an entry point to the web UI initialization. Because the SUI layout is made by using the Ext JS, all JS files in this file are executed to build the layout.
site/contentusage.php	The interface view for Content usage.
site/iadusage.php	The interface view for IAD usage.
collectionset/index.php	This view enables direct HTTP access to collectionset in SUI. For example: <a href="http://localhost/symboldatabase/index.php/collectionset/direct/romid/0xabcd1234">http://localhost/symboldatabase/index.php/collectionset/direct/romid/0xabcd1234</a>

### 5.5. Symbol Collector

Symbol Collector is a part of Symboldatabase and it is responsible for collecting the symbol files from the defined locations. Symbol Collector is based on a Yii



Framework's command application and it is executed from the command line as a scheduled task. The tool polls the defined target locations and tries to find new symbol files. When new data is found the tool uploads the data into Symboldatabase.

The configuration for the tool is defined in the Server Applications *console.php* file. The configuration has the attributes listed in Table 5.9.

Table 5.9: Symbol Collector configuration

Attribute	Description
Url	URL to target server(s). Multiple servers can be defined by using semicolon as a separator.
Poll	Polling period for the tool. Default value is 60 minutes.
Filetypes	Comma separated list of file types to be collected. File types must match the file types stored into database.

Symbol Collector tries to collect the symbol files with all the available metadata. In most cases the symbol files are parts of the newly created firmware and Symbol Collector collects the files from the known locations. This is a very reliable way to collect the symbol files, but the problem is that these are usually done only for the official releases.

Unofficial firmware may also be important. Unofficial means in this case that the firmware is not planned to be published and it may contain some special features that still needs to be tested. In these cases the symbol files are usually in a single directory so the metadata is very limited. The metadata is read from the filename if it is possible.

## 5.6. Web UI

### 5.6.1. Introduction

The Symboldatabase web UI (SUI) is designed to be a well-structured and easy to use. The SUI allows the user to browse, search and download symbol files from Symboldatabase. The SUI also provides functionality for administrator management. One of the main benefits of the SUI is that a user can download different variants of the symbol files. The variant files can share the same ROM ID as an identifier as the default symbol files. Using the different variant files is

not possible via interfaces, but the SUI shows all the available symbol files including the possible variants to the user.

SUI is a PHP web application that uses JavaScript for the layout. As seen in Figure 5.6, the layout of SUI is divided into three different logical parts that are named as menu, dataset and data view. This is quite common style to split SUI into different logical parts. The layout section *Menu* takes 10% of the height in SUI. The purpose of the Menu panel is to provide shortcuts to main functionalities.

The *Dataset* is a layout section that contains different options for content filtering. This layout takes 30% of the Web UI width. Rest of the space is for the *Data View* layout section which is used to present the actual data.

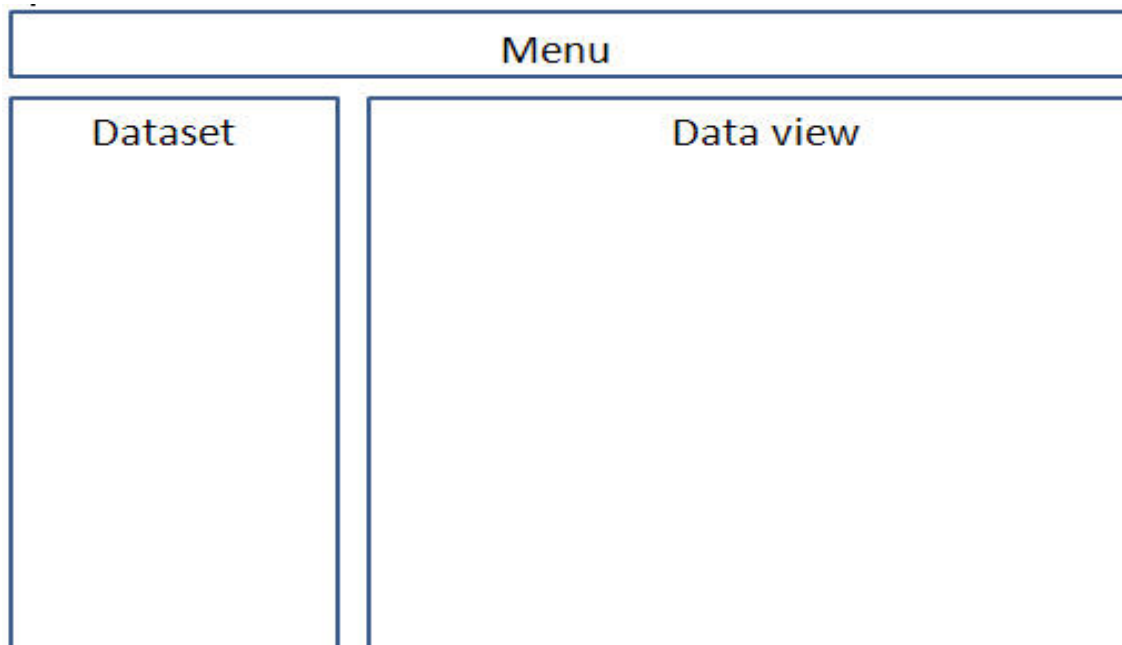


Figure 5.6: Layout sections of the SUI

Client-side JavaScript classes of the SUI is depicted in Figure 5.7. *SD.App* and *SD.Layout* classes provide the stone base for the SUI. These classes are responsible for building a layout. The main classes are described in Table 5.10.

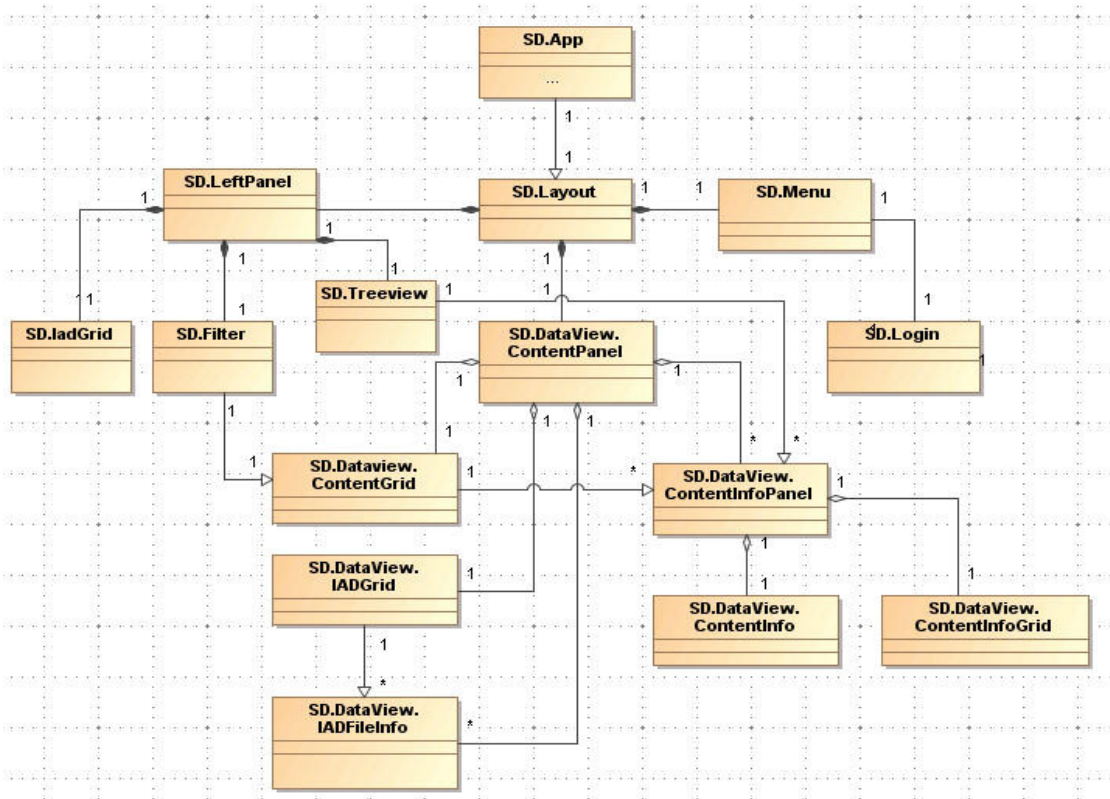


Figure 5.7: Client-side classes of the SUI

Table 5.10: Web UI classes

SD.App	The class is a starting point of the SUI. SD.App initializes the layout and handles variables used for layout.
SD.Layout	The class contains the layout of the SUI. Layout generates a structure of the SUI
SD.Menu	SD.Menu class provides functionality for the menu.
SD.LeftPanel	LeftPanel class handles all the UI components related to Dataset section of the layout.
SD.DataView.*	These classes share the same namespace, because all of them purpose is to show different kind of data.

### 5.6.2. Views

The design of the SUI views is described in this section. The views are designed in so that they meet the use cases and defined requirement as much as possible. The layout of the SUI is very simple and easy to use. In the SUI each selected collectionset creates a new tab to the Dataset layout. This functionality enables

multiple collectionsets to be opened at once and helps the user to keep the browsed data in the view.

## Main view

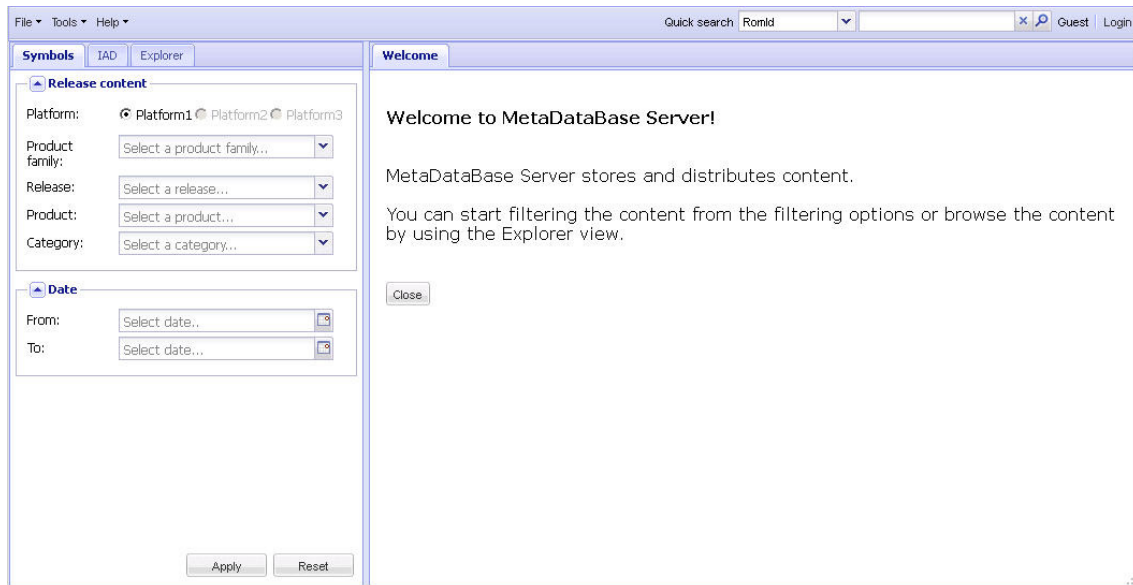


Figure 5.8: Main view of the SUI

The main view of the SUI is shown in Figure 5.8. On the Dataset layout there are three different tabs available. The first tab *Symbols* contains filtering options that can be used for filtering the content of the tool. The second tab *IAD* shows the IAD files that are stored into Symboldatabase. The third tab *Explorer* is a tree node view where the user can browse the content hierarchically.

When the user accesses the page first time, the welcome tab is shown in the Data View layout section. Welcome text is an information section which provides basic information about the SUI and how to use it. The Menu layout section is located on top of the page, where menu items, quick search textbox (requirement 2.1) and login functionality exist.

## List of collectionsets

When the user has used filtering options and pressed the Apply button the matching collectionsets are listed into grid inside the Data tab (Figure 5.9). The list of matching collectionsets is listed in the grid that can be sorted by every available column.

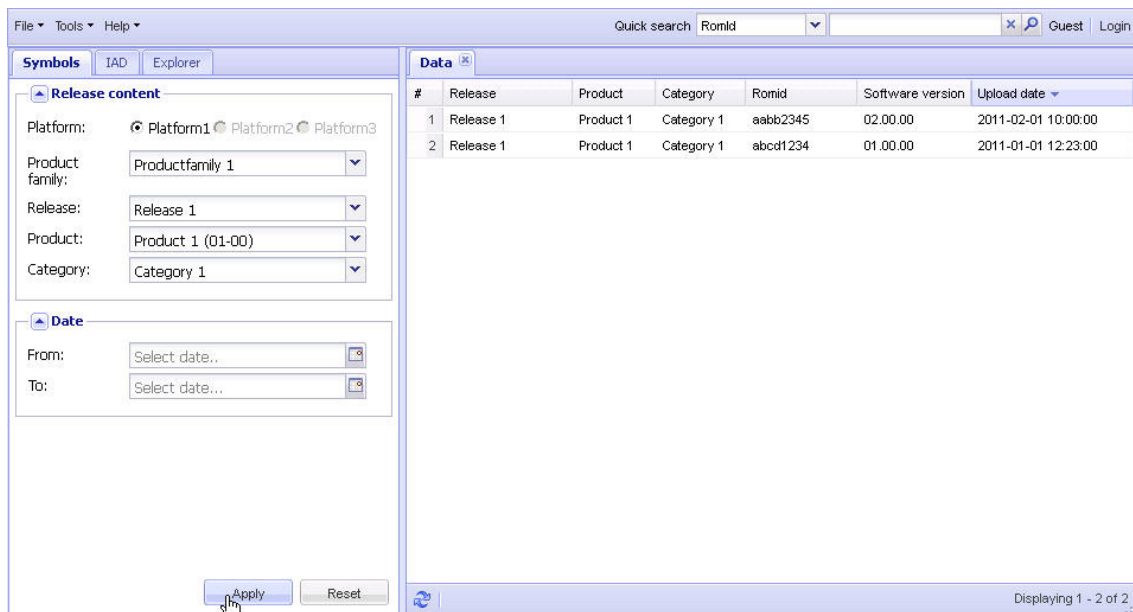


Figure 5.9: List of collectionsets

In the *Data* grid, a part of the collectionset metadata is shown. Available metadata gives all the needed information for the user so he/she can verify that the content is what he/she needs. The given metadata is: *name of the release, product name, category name, ROM ID value, software version information and the upload date*. All these columns are can be sorted and the user can even change the order of the columns. The user can also hide all the columns he/she find irrelevant. By double clicking a single collectionset in the grid, a new tab is created for the collectionset data.

### List of collectionset files

Files of the collectionset are listed and grouped by the types of the files (Figure 5.10). The user can hide the files that he/she finds useless by pressing [-] button on the left side of the type description.

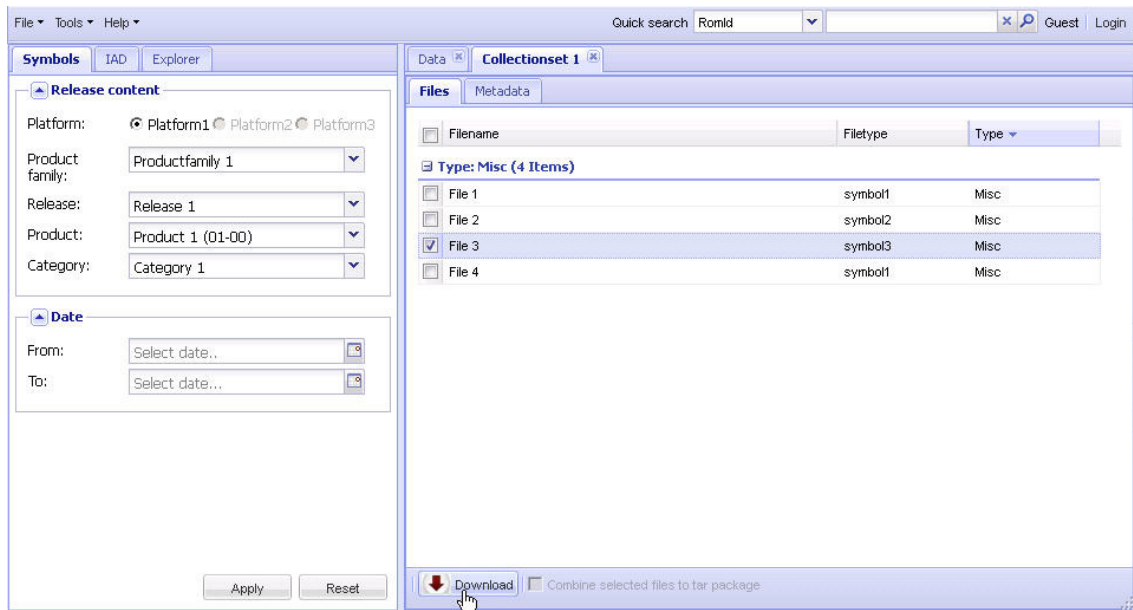


Figure 5.10: List of collectionset files

The collectionset files grid provides *filename*, *filetype* and *type* information of the file to the user. The difference between the filetype and type is that the type is a high level file type (like image to jpg), whereas the filetype gives information about the actual file type in *filename*. This can be valuable information for the user to find the proper files.

From the view the user can download all the available files or just a single file from Symboldatabase. User selects the checkbox of the desired files and presses the Download button. All files can be selected by selecting a checkbox on top of the grid and then pressing download. All files will be downloaded individually.

### Collectionset metadata

Metadata of the collectionset can be found from the *Metadata* tab (Figure 5.11). The metadata tab provides all the metadata information which is stored into the database related to selected collectionset. The metadata can help the user to verify the content that he/she is looking for.

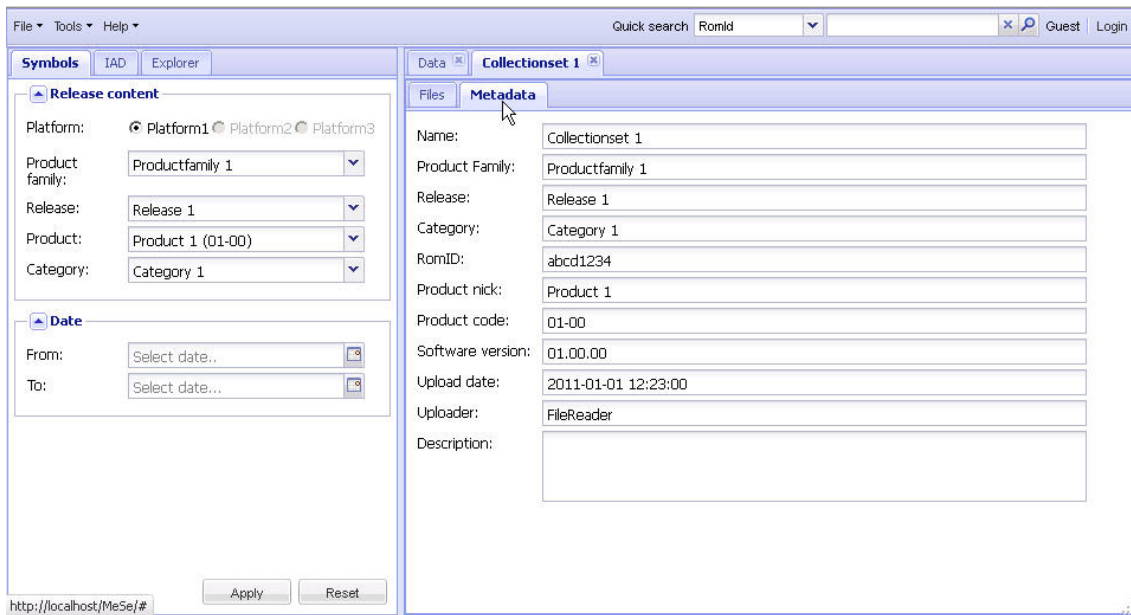


Figure 5.11: Collectionset metadata

## IAD

The IAD view is the second tab on the Dataset layout section (Figure 5.12). The IAD dataset view shows all the available IAD components with the UID information. The user can choose one-to-many different components and then press the Apply button. All the selected components with different versions are listed in the grid. The IAD Data tab is added into the Data View layout section. By selecting a single item from the grid a new tab is created for the selected AID file.

The IAD component view contains metadata information about a certain IAD file. It also contains the download button for the file downloading.

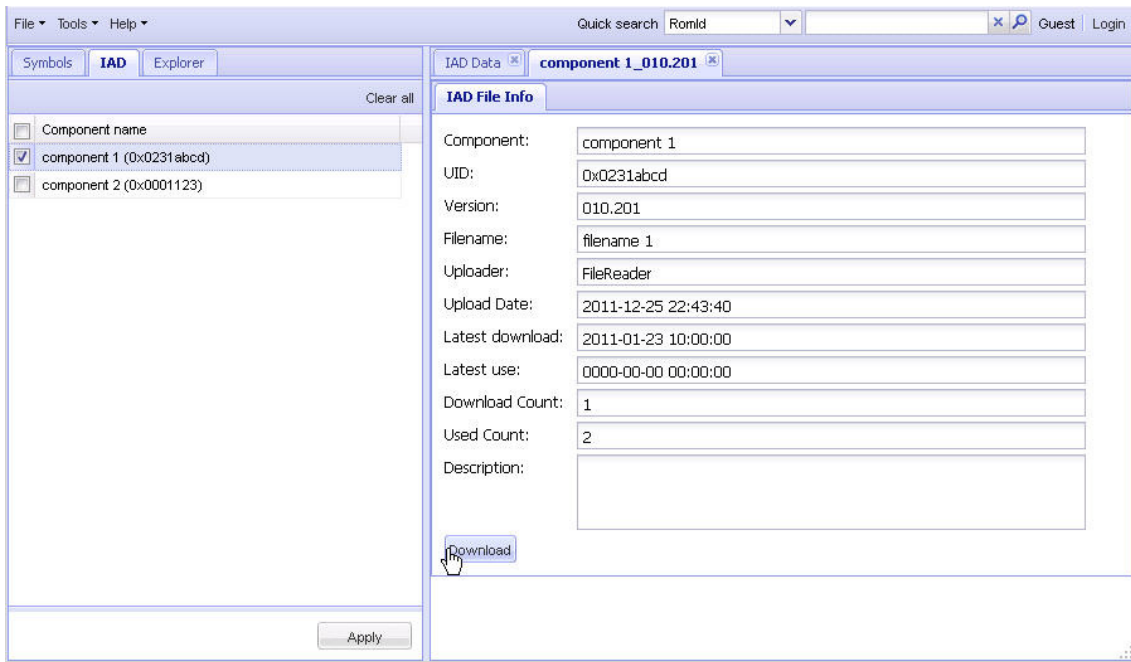


Figure 5.12: IAD file data view

## Explorer

The Explorer view is the third tab of the Dataset layout section. The Explorer view is a tree node, which can be hierarchically browsed by the user. The two root leafs, named as *Builds* and *IAD* are the starting points for the Explorer view.

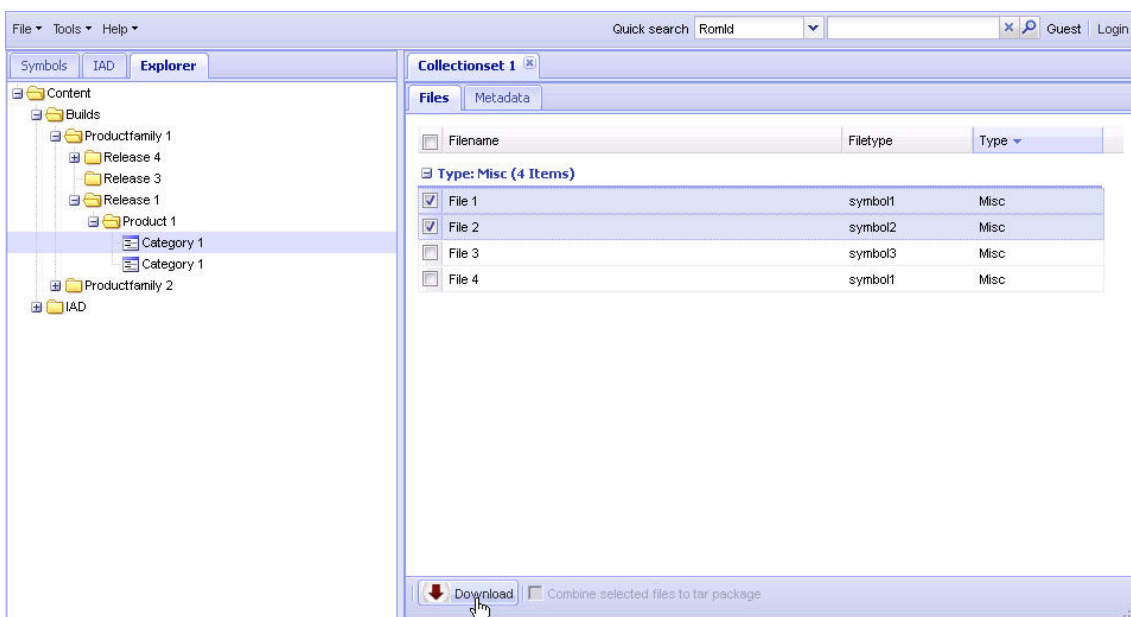


Figure 5.13: Explorer view

The *Builds* branch starts from the product family. By extending the selected value the available releases are shown. By extending some release, the found



products are shown in the tree. Each product may have different categories. When the user clicks a certain category (showed on blue line in Figure 5.13) the matching collectionset is opened in the Data view section.

When extending the IAD branch, all the available IAD components are shown with additional UID information. When a single component is extended, the found version numbers are shown. By clicking a version number, the IAD component metadata view is opened to the Data view (Figure 5.13).

## 5.7. Interfaces

One of the most important tasks of Symboldatabase is to provide symbols to the automated client tools. To be able to satisfy these needs (use case 3); Symboldatabase must provide simple and reliable interfaces which the client tools can use.

Symboldatabase provides two HTTP interfaces which enables symbol and IAD files to be used directly from the Symboldatabase server. Purpose of the interfaces is to provide a dynamic and simple way for the client application to find and use the needed files from Symboldatabase.

All the following interfaces are implemented by using a pure Yii PHP Framework without any JavaScript. The interfaces are simple to use and reduced, as only the very necessary data is shown. Next, the interfaces are described.

### 5.7.1. Content Usage

The purpose of the Content Usage interface is to provide location of the symbol file in the Symboldatabase server. A ROM ID value is used as an identifier to find the matching symbols. The format of the interface call is

*[http://localhost/mese/index.php/site/contentusage?romId=\\$romid](http://localhost/mese/index.php/site/contentusage?romId=$romid).*

Figure 5.14 presents the usage of the Content Usage interface. A client is willing to know whether Symboldatabase has symbols that match the ROM ID abcd1234. Client generates the needed URL and sends a HTTP request to Symboldatabase. Symboldatabase responses and prints the results to view. As seen in Figure 5.14, Symboldatabase does have the desired symbols and it provides the location for the files. The interface is easily accessible and readable for any client tool. "SelgeIni generator" is the starting point of the response and all the requested symbols are grouped by the ROM ID value. Text "done" is the

mark for the end of the results. After reading it the client tool knows that there is no more information available.



```

MetaDatabase Server - Conte...
localhost/MeSe/index.php/site/contentusage?fromId=abcd1234
SelgeIni generator

[INIFILE]

[abcd1234]
CORE_ROM=W:\localhost\metadatabase\collection_sets\devicefamily1\Productfamily 1\Release 1\Product 1\Category 1\abcd1234\File 1
CORE_ROFS=W:\localhost\metadatabase\collection_sets\devicefamily1\Productfamily 1\Release 1\Product 1\Category 1\abcd1234\File 2
CORE_ROM=W:\localhost\metadatabase\collection_sets\devicefamily1\Productfamily 1\Release 1\Product 1\Category 1\abcd1234\File 4

done

```

Figure 5.14: Use of the Content Usage http interface

### 5.7.2. IAD Usage

The IAD Usage interface provides metadata of the IAD content to client. The structure of a HTTP request for IAD content is

*[http://localhost/mese/index.php/site/iadusage?uid=\\$uid&name=\\$name&version=\\$version](http://localhost/mese/index.php/site/iadusage?uid=$uid&name=$name&version=$version).*

To be able to get the needed IAD content, the client tool must provide the name of the component, version number and UID. All of these are used to identify the correct IAD content. The results of the IAD interface can be seen in Figure 5.15.



```

MetaDatabase Server - IAD U: ...
localhost/MeSe/index.php/site/iadusage?name=component%201&uid=0x0231abcd&version=010.201
SelgeIni generator

[SIS_MAP_METADATA_INIFILE]
ini_format_version=1

[0231abcd]
NAME=component 1
VERSION=010.201
FILE=W:\localhost\metadatabase\IAD\component 1\0x0231abcd\010.201\filename 1

done

```

Figure 5.15: Use of the IAD Usage http interface

“SelgeIni generator” means a start of the results and “done” is the end mark of the results. On the interface the results are grouped by the UID and the location of the matching files is shown on the view.

### 5.7.3. Web Service

A web service is a software system designed to support interoperable machine-to-machine interaction over a network. [Yii 2, 2013] Symboldatabase provides web service APIs for the client tools to get metadata information about the stored files. Web service provided APIs can be used by using the SOAP request and responses messages. In order to use the web service provided APIs, a user has to login to Symboldatabase and SOAP sessions information is stored into the database. To be able to use web service the client tool must use the following URL for accessing Symboldatabase:

*[http://mese/index.php/module/web\\_service/api](http://mese/index.php/module/web_service/api)*.

The APIs provided by Symboldatabase are listed in Table 5.11.

Table 5.11 Web Service APIs

Method	Description
login(\$username, \$password)	Login user with the given credentials.
logout(\$username,\$session)	Logout user and close the session.
getCollectionSetsCount(\$search)	Returns a number of collection sets that match with <i>\$search</i> query. This function checks with <i>\$search</i> if a name or sw_version match for the query.
getCollectionSets(\$page=0, \$limit = 10, \$search = "")	Returns all the collectionsets that match with <i>\$search</i> query. This function checks with <i>\$search</i> if name, sw_version or rom_id match for the query.
getCollectionSetFiles(\$collection_set_id)	Returns files that belongs to collection set with id given by <i>\$collection_set_id</i> .
getListOfIADFiles()	Returns all IAD files information.
getFileSize(\$file, \$path)	Returns file size in bytes.
getRomidsForRelease(\$release)	Returns list of ROM IDs related to given <i>\$release</i> name. Returned list is the array of integers.

## 5.8. Database

The purpose of the Symboldatabase database is to store metadata of the stored data. Stored metadata is used for distributing the content to users. A relational database was selected, because it matches best with the given requirements.

Table 5.12 presents the names and descriptions of the database tables. The full schema of the database in schema is described in appendix A.

Table 5.12: Database tables

Table	Description
CollectionSet	The Collectionset table stores information about a set of files related to a product of the one specific release.
File	The File table stores information about files of the collection sets.
IadFile	The IADFile table stores information about IAD files.
CollectionSetFile	The CollectionSetFile table stores relation between a collectionset and a file.
Category	The Categories table stores different categories of the product releases. For example (production, rnd, subcon).
Filetype	The Filetype table stores information about different filetypes. The Filetype table is needed for managing filetypes that are allowed to be stored into the database. Filetypes are also needed for marking stored files.
DeviceFamily	The DeviceFamily table stores information about different device families.
ProductFamily	The ProductFamily table stores information about different product families.
Product	The Product table stores information about different product names (nicknames) and their product type names.
Release	The Release table stores information about releases. Every release is related to some product family and device family.
Setting	The Setting table stores configurations of the command console applications.
User	The User table stores information about users.

Figure 5.16 represents the tables and *foreign keys* used in the database. All the release related files are stored in the File table. All the files must have a *collectionset\_id* as a foreign key value. This is because a collectionset includes ROM ID information which is used as an identifier. That is why the files are not presented and cannot be distributed without valid collectionset information.

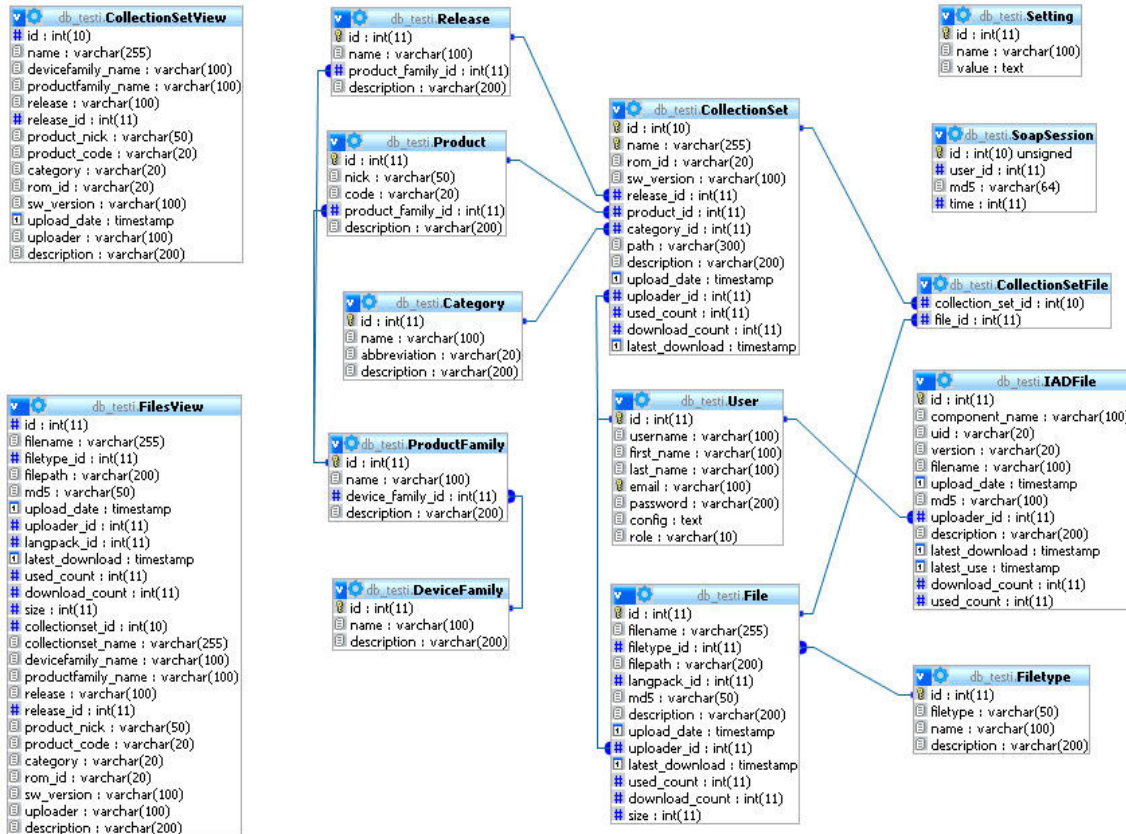


Figure 5.16: Tables and foreign keys of the database

The files in the database must also have *filetype\_id* information which defines the specific filetype for the file. Since Symboldatabase is only for storing and distributing a specific list of different filetypes, it must take care that not allowed filetypes are not stored into the system.

Collectionset must have a valid *category\_id*, *release\_id* and *product\_id* information. It is possible to set unknown values to matching ids on those tables. The most important value of the CollectionSet table is the ROM ID which is used to match the collectionset with the related files. Clients which are using Symboldatabase provide a ROM ID value and Symboldatabase provides the matching files to the user.

## 6. SUMMARY AND CONCLUSIONS

This chapter shortly summaries the key points covered in the thesis. The chapter also describes how the created solution resolves the described problems.

The thesis started with describing what challenges the data dispersion can cause inside a company. Data dispersion is a common problem in companies and Nokia is not an exception. The valuable files of enterprise can be decentralized and the files can be unavailable for the users. I noticed that this was the case with the symbol files and some solution was needed to make the use of the files easier and in some cases even possible.

In the background section I described how the data dispersion affects to company and what the symbol files are all about. I also described Microsoft and Mozilla systems for distributing and storing the symbol files into centralized repository that is easily accessible for all users.

The starting point was that the solution that I was going to make is based on a web application with a rich feature set. Reasons for this are mainly the accessibility inside the intranet and ease to use. The developed Symboldatabase is an application that is designed to resolve the problems with symbol files storing and distribution inside Nokia. Symboldatabase is a Yii PHP Frameworks project which uses the Ext JS framework for the SUI layout whereas MySQL is used as a database for the data management. Yii Framework provides libraries that are used for creating all the needed commands and components for the system. The chosen techniques support each other perfectly and fulfill all the set requirements.

Based on the comments from the users and project supervisor, Symboldatabase has been a real success. It has provided easy access to valuable symbol files for the users and the client tools. The client tools can use the symbol files effectively without any user actions. Symboldatabase is still in use in Nokia.

This thesis provides valuable information about the problems there might be with the valuable data inside a company. The described solutions complexity is quite low and its modularity is high. The techniques are kept the same as much as possible and there is no need to run, for example, any external scripts or processes. Symboldatabase is easy to use, accessible for all the internal users and client tools and, the most importantly, it works well.

## REFERENCES

- [Acunetix, 2011], Acunetix, Web applications: What are they, 2011. Available as <http://www.acunetix.com/websitesecurity/web-applications.htm>.
- [Bolton, 2013] David Bolton, Definition of Debugger, 2013. Available as <http://cplus.about.com/od/glossar1/g/debugdefinition.htm>.
- [Cattell and Barry, 1999] R.G.G Cattell, Douglas K. Barry, The Object Data Standard: ODMG 3.0, 1999. Available as <http://www.xtec.cat/~iguixa/materialsGenerics/ODMG30.pdf>
- [Codd, 1972] E. F. Codd, Relational Completeness of Data Base Sublanguages, 1972. Technical Report RJ 987, IBM Research Laboratory, San Jose, CA. Available as <http://www.inf.unibz.it/~franconi/teaching/2006/kbdb/Codd72a.pdf>
- [Coplien, 1994] James O. Coplien, Supporting Truly Object-Oriented Debugging of C++ Programs. In: *Proc. of the 1994 USENIX C++ Conference*, 1994, 99-108. Also available as [http://www.usenix.org/publications/library/proceedings/c++94/full\\_papers/coplien.a](http://www.usenix.org/publications/library/proceedings/c++94/full_papers/coplien.a).
- [Darka, 2012], Jonathan Darka, Debugging Symbols, 2012. Available as <http://www.codeproject.com/KB/debug/symbols.aspx?q=symbol+server+article>.
- [DocForge, 2010] DocForge, Web Application Framework, 2010. Available as [http://docforge.com/wiki/Web\\_application\\_framework](http://docforge.com/wiki/Web_application_framework).
- [Elmasri and Navathe, 2000] Ramez Elmasri and Shamkant B. Navathe, *Fundamentals of Database Systems*, 2000. Available as <http://cecfoces.files.wordpress.com/2010/09/elmasri-navathe-fundamentals-of-database-systems-3rd-ed1.pdf>.
- [Farokhzad *et al.*, 2010] Shahabeddin Farokhzad, Gokhan Tanyeri, Trish Messiter and Paul Beckett, Plug-in Based Debugging for Embedded Systems, 2010. Available as <http://www.clarinox.com/docs/whitepapers/EmbeddedDebugger.pdf>.
- [Grove, 2010] Ralph F. Grove, *Web-based Application Development*. Jones and Bartlett Publishers, LCC, 2010. 17-21.
- [Jablonski *et al.*, 2011] Stefan Jablonski, Ilia Petrov, Christian Meiler and Udo Mayer, *Guide to Web Application and Platform Architectures*, 2011. University of Erlangen-Nuremberg, Dept. of Computer Science 6 (Database Systems).
- [Madsen, 2010] Mark Madsen, Open Source Solutions, 2010. Available as <http://www.dashboardinsight.com/articles/new-concepts-in-business-intelligence/open-source-solutions.aspx?page=6>.
- [McInerney *et al.*, 2000] Peter J. McInerney, Michael D. Wimble and Lawrence L. You, Demand-based generation of symbolic information, 2010. Available as <http://www.google.com/patents/US6067641>.
- [Microsoft, 2013] Microsoft, Symbol Server and Symbol Stores, 2013. Available

- as [http://msdn.microsoft.com/en-us/library/windows/desktop/ms680693\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/desktop/ms680693(v=vs.85).aspx).
- [Microsoft 2, 2013] Microsoft, How to use a Symbol Server, 2013. Available as [http://msdn.microsoft.com/en-us/library/b8ttk8zy\(v=vs.90\).aspx](http://msdn.microsoft.com/en-us/library/b8ttk8zy(v=vs.90).aspx).
- [Microsoft 3, 2013] Microsoft, Symbols and Symbol Files, 2013. Available as [http://msdn.microsoft.com/en-us/library/windows/hardware/ff558825\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/hardware/ff558825(v=vs.85).aspx).
- [Microsoft 4, 2013] Microsoft, Public and Private Symbols, 2013. Available as [http://msdn.microsoft.com/en-us/library/windows/hardware/ff553493\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/hardware/ff553493(v=vs.85).aspx).
- [Mozilla, 2013] Mozilla, Using the Mozilla Symbol Server, 2013. Available as [https://developer.mozilla.org/en-US/docs/Using\\_the\\_Mozilla\\_symbol\\_server](https://developer.mozilla.org/en-US/docs/Using_the_Mozilla_symbol_server).
- [Mozilla 2, 2013] Mozilla, Using the Mozilla Source Server, 2013. Available as [https://developer.mozilla.org/en-US/docs/Using\\_the\\_Mozilla\\_source\\_server](https://developer.mozilla.org/en-US/docs/Using_the_Mozilla_source_server).
- [Mozilla 3, 2013] Mozilla, Uploading Symbols to Mozilla's Symbol Server, 2013. Available as [https://developer.mozilla.org/en-US/docs/Uploading\\_symbols\\_to\\_Mozillas\\_symbol\\_server](https://developer.mozilla.org/en-US/docs/Uploading_symbols_to_Mozillas_symbol_server).
- [PHP, 2010] The PHP Group, PHP: Hypertext Preprocessor, 2010. Available as <http://php.net/index.php>.
- [PHP, 2013] The PHP Group, History of PHP, 2013. Available as <http://www.php.net/manual/en/history.php.php>.
- [PHP 2, 2013] PHPFrameworks.com PHP Frameworks, 2013. Available as <http://phpframeworks.com/>.
- [PHP MVC, 2009] Model View Controller (MVC) in PHP tutorial, 2009. Available as <http://php-html.net/tutorials/model-view-controller-in-php/>.
- [Ruilog, 2013] Ruilog, PHP Framework MVC Benchmark, 2013. Available as <http://www.ruilog.com/blog/view/b6f0e42cf705.html>.
- [Sencha 2010] Sencha, Ext JS: Cross-Browser Rich Internet Application Framework, 2010. Available as <http://www.extjs.com/products/extjs/>.
- [Shelmandu, 2010] Shelmandu, PHP MVC Framework Performance – Part1, 2010. Available as <http://www.sheldmandu.com/php/php-mvc-frameworks/php-mvc-framework-performance-part-1>.
- [Singh *et al.*, 2008] Ravendra Singh, Vivek Sharma and Manish Varshney. Design and Implementation of Compiler, 2008. Available as <http://www.newagepublishers.com/samplechapter/001679.pdf>.
- [Spahr, 2011] Robert Spahr. AJAX web applications, 2011. Available as [http://www.robertspahr.com/teaching/nmp/ajax\\_web\\_applications.pdf](http://www.robertspahr.com/teaching/nmp/ajax_web_applications.pdf).
- [Stobart and Parsons, 2008] Simon Stobart and David Parsons, *Dynamic Web Application Development using PHP and MySQL*. Cengage Learning EMEA, London, 2008, 1-13.
- [Stonebraker, 2003], Michael Stonebraker, Object-Relational DBMS – The Next



Wave, 2003. Available as  
<http://infolab.usc.edu/csci587/Fall2010/papers/Object-Relational%20DBMS-The%20Next%20Wave.pdf>

[W3schools, 2011] W3schools, AJAX Introduction, 2010. Available as  
[http://www.w3schools.com/Ajax/ajax\\_intro.asp](http://www.w3schools.com/Ajax/ajax_intro.asp).

[W3techs, 2013] W3techs, Usage of Client-side Programming Languages for Websites, 2013. Available as  
[http://w3techs.com/technologies/overview/client\\_side\\_language/all](http://w3techs.com/technologies/overview/client_side_language/all).

[W3techs 2, 2011-2013] W3techs, Usage of server-side programming languages For Websites, 2011-2013. Available as  
[http://w3techs.com/technologies/overview/programming\\_language/all](http://w3techs.com/technologies/overview/programming_language/all).

[Williams and Lane, 2004] Hugh E. Williams and David Lane, *Web Database Applications with PHP and MySQL, Second Edition*. O'Reilly Media Inc., 2004.

[Yii, 2010] Yii Software LLC, Yii PHP Framework, 2010. Available as  
<http://www.yiiframework.com/about/>.

[Yii 2, 2013] Yii Software LLC, Web Services, 2013. Available as  
<http://www.yiiframework.com/doc/guide/1.1/en/topics.webservice>.

## APPENDIX A: DATABASE TABLES

### CollectionSet

Table	Type	Extra	Description
id	int(11)	Primary key, auto increment	Id of the collectionset.
name	varchar(100)		Name of the collectionset.
rom_id	varchar(20)		ROM id value of the image creation
sw_version	varchar(100)		Software version information
release_id	int(11)	Foreign key(Release.id)	Foreign key to Release table id value
product_id	int(11)	Foreign key(Product.id)	Foreign key to Product table id value
category_id	int(11)	Foreign key(Category.id)	Foreign key to Category table id value
path	varchar(255)		Path is collection set filepath in the server
description	varchar(200)	Null	Description of the collectionset
upload_date	timestamp	Default: 0000-00-00 00:00:00	Upload date of the collection set
uploader_id	int(11)	Foreign key(User.id)	Foreign key to user table.
used_count	int(11)	Default: 0	Used count of the collectionset.
download_count	int(11)	Default: 0	How many times collection set have been downloaded
latest_download	timestamp	Null	When was the latest download

## File

Table	Type	Extra	Description
id	int(11)	Primary key, auto increment	Id of the file
filename	varchar(100)		Name of the collection set. Format: (release_product_category)
filepath	varchar(250)		Path of the file
filetype_id	int(11)	Foreign key(Filetype.id)	Foreign key to Filetype table which defines what kind of filetypes are allowed to store.
langpack_id	int(11)	Null	Langpack defines language variant number for the image and symbol files.
md5	varchar(50)	Null	Md5 calculated for the file to find out data corruption in data transferring.
description	varchar(200)	Null	Description of the collection set
upload_date	timestamp	Default: 0000-00-00 00:00:00	Upload_date of the collection set
uploader_id	int(11)	Foreign key(User.id)	Foreign key to users table.
used_count	int(11)	Null, Default: 0	How many times file has been used on the server-side. For example symbols for crash decoding.
download_count	int(11)	Null, Default: 0	How many times collection set or just this file have been downloaded

Table	Type	Extra	Description
latest_download	timestamp	Default: 0000-00-00 00:00:00	When was the latest download

## IADFile

Table	Type	Extra	Description
id	int(11)	Primary key, auto increment	Id of the file
component_name	varchar(100)		Name of the component
uid	varchar(20)		Unique identifier for the component
version	varchar(20)		Version information of the component.
filename	varchar(100)		Actual filename in the file system
upload_date	timestamp	Default: 0000-00-00 00:00:00	Upload_date of the IAD file
md5	varchar(50)	Null	Md5 calculated for the file to find out data corruption in data transferring.
uploader_id	int	Foreign key(User.id)	Foreign key to users table.
description	varchar(200)	Null	Description of the IAD file
latest_download	timestamp	Default: 0000-00-00 00:00:00	When was the latest download
latest_use	timestamp	Default: 0000-00-00 00:00:00	When was the latest use (from http interface)
download_count	Int	Null, Default: 0	How many times this IAD file have been downloaded
used_count	int	Null, Default: 0	How many times IAD file has been used on the server-side (from http interface).

## CollectionSetFile

Table	Type	Extra	Description
collection_set_id	int(11)	Foreign key (CollectionSet.id)	Id of the collection set
file_id	varchar(100)	Foreign key (File.id)	Id of the file

## DeviceFamily

Table	Type	Extra	Description
id	int(11)	Primary key, auto increment	Id of the device family
name	varchar(100)	NOT NULL	Name of the device family
description	varchar(200)	NULL	Description of the device family

## ProductFamily

Table	Type	Extra	Description
id	int(11)	Primary key, auto increment	Id of the product family
name	varchar(100)	NOT NULL	Name of the product family
device_family_id	int(11)	NOT NULL	Id of the device family
description	varchar(200)	NULL	Description of the device family

## Release

Table	Type	Extra	Description
id	int(11)	Primary key, auto increment	Id of the release
name	varchar(100)	NOT NULL	Name of the release
product_family_id	int(11)	Foreign key (ProductFamily.id)	Product_family_id is a foreign key to ProductFamily table which defines that which product family the release is related
description	varchar(200)	NULL	Description of the release

## Product

Column	Type	Extra	Description
id	int(11)	Primary key, auto increment	Id of the product
nick	varchar(50)	NOT NULL	Nickname of the product
code	varchar(20)	NOT NULL	Code of the product
product_family_id	int(11)	Foreign key(ProductFamily.id)	Product_family_id is a foreign key to ProductFamily table which defines that which product family the release is related
description	varchar(200)	NULL	Description of the products

## Filetype

Column	Type	Extra	Description
id	int(11)	Primary key, auto increment	Id of the filetype
filetype	varchar(50)	NOT NULL	Filetype definition. (eg. .symbol, .fpsx, map)
name	varchar(100)	NOT NULL	Name of the filetype
description	varchar(200)	NULL	Description of the filetype

## Category

Column	Type	Extra	Description
id	int(11)	Primary key, auto increment	Id of the category
name	varchar(100)	NOT NULL	Full name of the category
abbreviation	varchar(20)	NOT NULL	Abbreviation of the category
description	varchar(200)	NULL	Description of the category

## Setting

Column	Type	Extra	Description
id	int(11)	Primary key, auto increment	Id of the configuration
name	varchar(100)	NOT NULL	Full name of the configuration
value	text	NOT NULL	Value of the setting. Includes all needed information

## SoapSession

Column	Type	Extra	Description
id	int(11)	Primary key, auto increment	Id of the session
user_id	varchar(100)	NOT NULL	Id of the user
md5	varchar(64)	NOT NULL	MD5 value of the session. Used as unique identifier
time	int(11)	NULL	Time for the session

## User

Table	Type	Extra	Description
id	int(11)	Primary key, auto increment	Id of the user
user_name	varchar(100)	Unique, NOT NULL	Username
first_name	varchar(100)	NOT NULL	First name of the user
last_name	varchar(100)	NOT NULL	Last name of the user
email	varchar(100)	Unique, NULL	Users email address
password	varchar(200)	NOT NULL	Password of the user. Scripted
config	text	NULL	Config value of the user
Role	varchar(10)	NOT NULL	Role of the user. 0 = developer, 1 = admin



## Views

### CollectionSetView

Table	Type	Extra	Description
id	int(11)	Foreign key(CollectionSet.id)	Id of the collection set.
name	varchar(100)	Foreign key(CollectionSet.name)	Name of the collection set.
devicefamily_ name	varchar(100)	Foreign key(DeviceFamily.name)	Name of the device family
productfamily_ _name	varchar(100)	Foreign key(ProductFamily.name)	Name of the product family
release_id	int(11)	Foreign key(Release.id)	Id of the release
product_nick	varchar(50)	Foreign key(Product.nick)	Nickname of the product
product_code	varchar(20)	Foreign key(Product.code)	Code of the product
category	varchar(20)	Foreign key(Category.name)	Category abbreviation
rom_id	varchar(20)	Foreign key(CollectionSet.rom_id)	ROM id value of the image creation
sw_version	varchar(100)	Foreign key(CollectionSet.sw _version)	Software version information
upload_date	timestamp	Foreign key(CollectionSet.upload_ date)	Upload date of the collection set
uploader	varchar(100)	Foreign key(User.username)	Username of the collection set uploader
description	varchar(200)	Foreign key(CollectionSet.descript ion)	Collectionset description

## FilesView

Table	Type	Extra	Description
id	int(11)	Foreign key(File.id)	Id of the collection set.
filename	varchar(100)	Foreign key(File.filename)	Name of the file
filetype_id	int	Foreign key(File.filetype_id)	Id of the filetype
filepath	varchar(100)	Foreign key(File.filepath)	Filepath of the file
md5	varchar(100)	Foreign key(File.md5)	MD5 value of the file
upload_date	timestamp	Foreign key(File.upload_date)	Upload date of the file
uploader_id	varchar(100)	Foreign key(File.uploader_id)	User id of the file uploader
langpack_id	int(11)	Foreign key(File.langpack_id)	Id of the language pack information
latest_download	timestamp	Foreign key(File.latest_download)	When was the latest download
used_count	int(11)	Foreign key(File.used_count)	How many times file has been used on the server-side. For example symbols for crash decoding.
download_count	int(11)	Foreign key(File.download_count)	How many times file has been downloaded
size	int(11)	Foreign key(File.size)	Size of the file
collectionset_id	int(11)	Foreign key(CollectionSet.id)	Id of the collection set.
collectionset_name	varchar(255)	Foreign key(CollectionSet.name)	Name of the collectionset
devicefamily_name	varchar(100)	Foreign key(DeviceFamily.name)	Name of the device family

productfamily_name	varchar(100)	Foreign key(ProductFamily.name)	Name of the product family
release_id	int(11)	Foreign key(CollectionSet.id)	Id of the release
product_nick	varchar(50)	Foreign key(Product.nick)	Nickname of the product
product_code	varchar(20)	NULL, Foreign key(Product.code)	Code of the product
category	varchar(20)	Foreign key(Category.name)	Category abbreviation
rom_id	varchar(20)	Foreign key(CollectionSet.rom_id)	Romid of the image creation
sw_version	varchar(100)	Foreign key(CollectionSet.sw_version)	Software version information
uploader	varchar(100)	Foreign key(User.username)	Username of the file uploader
description	varchar(200)	Foreign key(File.description)	File description