Re-engineering Web Applications from MSO to SSO

A centralized customer database

Simon Sabelis

Re-engineering Web Applications from MSO to SSO

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER ENGINEERING

by

Simon Sabelis
born in Heemstede, the Netherlands

Software Engineering Research Group
Department of Software Technology
Faculty EEMCS, Delft University of Technology
Delft, the Netherlands
www.ewi.tudelft.nl

MediaMij BV
Grasweg 63-65
Amsterdam, the Netherlands
www.mediamij.nl

© 2011 Simon Sabelis.

Re-engineering Web Applications from MSO to SSO

Author: Simon Sabelis
Student id: 1050737
Email: simon@sabelis.nu

Abstract

Nowadays a lot of small companies maintain multiple websites that require the user to sign up for ordering products or extended interaction. The users have to create an account for each website the company owns, creating duplicate information for the company and forcing the customer to remember multiple accounts. Where big companies can solve the problem with commercial solutions, small companies do not have the funding. MediaMij, a small publisher, has to deal with exactly this situation.

In this project we investigate options to find a solution for MediaMij. We research ways of re-engineering the current web applications to enable a single central customer database with Single Sign On capabilities. The research renders an approach for the situation which will be verified by conducting 2 case studies on 2 of the current websites of MediaMij. With the verified approach we were able to form a set of guidelines describing how to approach projects that are alike.

Thesis Committee:

Chair: Prof. Dr. A. van Deursen, Faculty EEMCS, TU Delft
University supervisor: Dr. M. Pinzger, Faculty EEMCS, TU Delft

Preface

With the writing of this thesis the end of my Msc project also has come. It has taken some more time then expected, but nonetheless I am happy with the result. When I look back I can actually say I am proud of the result, certainly keeping in mind the personal events that interrupted the progress. Finishing the the project will transform these events into something positive besides the negativeness I already experienced.

Therefore I want to thank my girlfriend Ilona foremost, for all the support she gave me before the project, during the project and the support she will give after the project. She gave me the motivation to keep on going at the times I needed being motivated.

For the real motivation and great feedback and also the nice meetings I want to thank my supervisor Martin Pinzger. His help for setting up the project, the brainstorm sessions we had to investigate possible solutions and the help during the writing of this thesis were vital to the success of this project.

Lastly I want to thank MediaMij for allowing me the use of their databases and applications for the project.

Simon Sabelis
Delft, the Netherlands
August 2, 2011

Contents

Preface iii

Contents v

List of Figures vii

List of Tables ix

1 Introduction 1
- 1.1 Problems with current design . . . 1
- 1.2 Requirements for the new situation . . . 2
- 1.3 Contributions . . . 3
- 1.4 Outline of the thesis . . . 3

2 Background and related work 5
- 2.1 Web Reverse Engineering . . . 5
- 2.2 SSO . . . 8
- 2.3 Migration . . . 10
- 2.4 Summary . . . 11

3 Approach 13
- 3.1 Phases of engineering . . . 13
- 3.2 Inventorisation . . . 15
- 3.3 Development . . . 18
- 3.4 Refactoring the existing websites . . . 23
- 3.5 Migration . . . 24

4 Case Studies: 2 MediaMij Websites 27
- 4.1 Welke.nl . . . 27
- 4.2 Huisplan.com . . . 29
- 4.3 Inventorisation . . . 29

4.4 Development . . . 40
4.5 Refactoring . . . 44
4.6 Actual migration . . . 47
4.7 Validation of requirements . . . 50

5 Lessons learned 55
5.1 Website reverse engineering . . . 55
5.2 SSO Design . . . 57
5.3 Migration . . . 58
5.4 MSO to SSO Guideline . . . 58

6 Conclusions and Future Work 63
6.1 Contributions . . . 63
6.2 Future work . . . 64

Bibliography 65

A Glossary 69

B API definition 71
B.1 Backend API . . . 71
B.2 frontend API . . . 74

List of Figures

3.1 Input and result diagram of the Approach 14
3.2 Project work breakdown structure 15
3.3 Inventorisation WBS 16
3.4 Development WBS 18
3.5 SSO Database design 19
3.6 SSO design 20
3.7 SSO register sequence 21
3.8 SSO login sequence 22
3.9 Refactoring WBS 23
3.10 Generic SSO implementation within the website 24
3.11 Migration WBS 25

4.1 Database design schema of old version of welke.nl 30
4.2 Use case structure of old version of welke.nl 33
4.3 Implementation details of old version of welke.nl 35
4.4 Database design schema of old version of huisplan.com 37
4.5 Use case structure of old version of huisplan.coml 38
4.6 Implementation details of old version of huisplan.com 39
4.7 Database design for the SSO system 41
4.8 SSO system integration 42
4.9 SSO login sequence 43
4.10 SSO welke implementation 45
4.11 SSO huisplan implementation 46

5.1 Steps of the guideline 59

List of Tables

2.1 Different levels of SSO 9
2.2 SSO Solution comparison 10

4.1 Welke.nl migration schema 49
4.2 huisplan.com migration schema 50
4.3 Measurements of slowdown while executing registration scenario 52
4.4 Measurements of slowdown while executing login scenario 52

Chapter 1

Introduction

Media Mij. BV is a publisher of living related magazines and maintains websites corresponding to the magazines and real estate business. Recently a new web-platform has been released for the primary magazines, the so called Welke magazines, with buyer info for kitchens, bathrooms, floor, etc. During the development of the new platform it was discovered that there are many ways in use to store client information, client accounts and profiles. Both the way the information is gathered (account creation, signup forms, magazine order, etc) and the underlying technologies (php, javascript, java, email-forms) differ a lot and are not standardized nor exchangeable. With the new platform and the current ages standardization, integration but also privacy are becoming more and more important. Media Mij. BV has the need to analyze the various databases, datasources and technologies, and to migrate to a newly created central organized database with single sign-on possibilities and excellent security and privacy.

1.1 Problems with current design

The current situtation as described has a number of shortcomings which is the main reason behind starting up this project. The shortcomings can be categorized as follows:

1.1.1 Multiple accounts

Users of the websites of MediaMij have to create accounts for all of the websites seperately and so they have to type in their personal information all over again when they want to subscribe at an other website owned by MediaMij. Secondly the administrators of these websites also have a seperate account for each website.

1.1.2 No standard for account uniqueness

The websites have different meanings about account uniqueness, whereas welke.nl for example samples the email address as an unique property of the account, huisplan.com says an account is unique if the street address of the person differs. The latter option has an immediately visible flaw as it does not take into account movals of people, so for example if someone moves but forgets to update his or her account, no one else can register on that address.

1.1.3 Data synchronization / profile format

The profile information stored at the different websites differs, name prefixes are not allways stored and sometimes the name is stored fully concatenated, firstname and lastname in one field, this makes comparing profiles companywide difficult.

1.1.4 No central management

Evidently as there is no central customer database and there is no real comparison possible, overall statistics and analysis is difficult, profiles cannot be removed/added centrally.

1.2 Requirements for the new situation

The new situation has to cope with the shortcommings of the old design as a starting point and take care of the newly wanted features by the company. This leads to a set of functional and non-functional requirements which will determine if the project is a success in the end or not.

1.2.1 Functional requirements

1. Central customer database
   a) Declare the emailadres as the unique identifier for each customer of MediaMij
   b) All information entered at one of the web application of MediaMij can be stored in the customer database
   c) Behavioural information gathered about the customer can be stored in the customer database
2. Customers need to login once and the login session will be active for all of the web applications of MediaMij
3. External API at the SSO system offering access to the customer database for the web applications of MediaMij
   a) Web applications of MediaMij can check the validity of a customer session
   b) Web applications of MediaMij can request for the user role of a customer
   c) To be developed MediaMij applications can gather behavioural information about customers for analysis
   d) To be developed MediaMij applications can store generated behavioural or statistic information about customers through the API

1.2.2 Non functional requirements

1. Slowdown of the application due to using an other sign on system then the one provided with the web application is at most 500ms
2. Communication between the web application and the SSO system is secure

1.3 Contributions

In the project patterns and tools found in the background & related work described in Chapter 2 are used to reverse engineer the web application, to design the new situation and to migrate from the old situation to the new situation. The combination of these patterns and tools dellivered the final product.

Some patterns were interpreted more freely or combined with other patterns or tools which rendered good results. This is described in Section 3.2.2 and applied in Section 4.3.1.2 and Section 4.3.2.2.

Another contribution of this project is the guideline describing the approach to projects that are alike. The guideline can be found in Section 5.4.

1.4 Outline of the thesis

In Chapter 2 we describe backgrounds and related work, in which we find challenges and discussions about topics similar to the ones found in this project. This knowledge is used as a basis for the approach to the project and is described in Chapter 3. To prove a succesfull project 2 web applications are used as a case study for checking the succesfullness of the approach. This case study is detailed in length in Chapter 4, here the details of the progress of applying the approach to both applications is described.
During the case study several advantages, disadvantages and issues were found about the approach, which are used as a basis for a guideline describing a generic approach to similar projects, featured in Chapter 5.
The final conclusions and a list of contributions are found in Chapter 6. At the end of that Chapter we sum up also a list of work that would enhance the result but was declared out of the scope of this project due to time constraints.

Chapter 2

Background and related work

During the seperate phases of the project challenges arose which can be related to known challenges in other scientific work. In this project the knowledge of scientific research is used as background and inspiration for the approach of the project. In this chapter the various areas of knowledge used in the project are discussed and related to existing studies.

2.1 Web Reverse Engineering

Web applications differ from standard applications in many ways, whereas the use of an application to use them, the iternet browser installed at the user's computer, is the most important difference. The web application generates its output as HTML code which is then intepreted by the browser to nicely present the web application pages on the screen for the user. The biggest challenge are the numerous technologies available for web applications which make the process of reverse engineering web applications difficult. Whereas at the backend of the web application, the part running on the web server, the choice of many programming and scripting languages hinder the process, at the frontend the process is even harder as scripting code which dynamicly updates the content for the user, or even triggers tasks at the web application backend itself, exist.

Completing the challenge with the allmost allways absence of (updated) documentation of the web application, similarly observed in [33, 13, 34], the reverse engineering of web applications seems allmost impossible.

For reverse engineering it is important to understand the development methodologies available for web applications, as there are many, not all can be investigated and documented. For the scope of the project we looked into development methodologies which are used as a basis for the reverse engineering strategies. A variety of methodologies is reviewed by Patel et al in [33].

First we look into UML based methodologies which use UML to describe the web application structure. Conallen proposes in [6] a number of extensions aiding the development of web applications. Some of the ideas proposed by Conallen are included in several reverse engineering techniques, found in Section 2.1.1 below. This type of reverse engineering methodologies form the major part of all of the methodologies available.

A different methodology is one that tries to describe the relational model used in the web application for the storage of data. Isakowitz describes the Relationship Management Model (RMM) in [25] and extends it in [24] as a basis for developing a web application. The model is the output of techonologies reverse engineering web applications to discover the relational data model of the application.

Also we look into methodologies sepcifically targetting dynamic (Ajax) web applications, as the web applications handled in this project are also implemented using Ajax [21] for dynamic updates of the web pages. We see in [2] that statistical analysis of web application is insufficient. This statement is confirmed by Matthijssen in [28] with his research on Ajax web applications. With pure statistical analysis it is hard to find out upfront which code is reached inside the web application by the dynamicly executed JavaScript code at the client.

Finally we investigate other existing methodologies with interesting starting points.

2.1.1 Structure Modelling

As UML is among the most widely accepted modelling methodologies, it supplies a familiar environment for engineers. With UML modelling we want to model the structure and behavioural details of the web application in UML based diagrams and among the results will be uml use-case diagrams and sequence diagrams describing the internal structure of the web application.

Tramontana writes about the WARE project in [41] describing the usage of the WARE tool to extract the structure of a web application, cluster the pages into use cases as to provide visualization of the various usage scenario's of the web application. The WARE tool is described in detail by Di Lucca et al in [16], [18] and [17]. WARE offers the combination of static source code analysis and dynamic code analysis.

The tool starts with a static analysis of the source code and proceeds with an analysis during the executing of the web application to discover the dynamic content an behaviour. Then using automatic clustering (as described in [15]) the tool clusters the discovered structure into sets of clusters representing uses of the web application. The seperate clusters contain a description of the functionality they implement within the web application. The downside however is that the tool needs human intervention for the dynamic analysis phase, as the tool needs input values to execute the analysis.

Other tools and methodologies based on the same principle of combining static and dynamic analysis with clustering as to generate UML models of the web application are for example but not limited to: WebUml, described by Bellettini in [5], the methodology of di Francesomarino et al described in [12] and the method of Pu et al in [35].

2.1.2 The relational model of web applications

Another method of reverse engineering a web application available in the literature is discovering its entity relation model as to comprehend what kind of data the web application uses and what operations it executes on that data.

Antoniol proposes a method resulting in a RMM in [1] and Di Lucca et al propose in [14] an other method based on Ubiquitous Web Applications (UWA) or omnipresent web applications [7]. The result of both methodologies is the entity relational model (ERM) of the web application.

2.1.3 Dynamic web applications

Dynamic web applications are web applications containing client code that dynamicly updates the web page the client is viewing without refreshing the whole page. The latter has been the standard for web applications before the introduction of JavaScript. Ajax is the name for the set of technologies enabling the development of interactive web applications. As the content is updated by dynamicly executed code, the reverse engineering of the code is harder then with non Ajax web applications.

In [28] some methods and tools are described enabling the understanding and ultimately the reverse engineering of dynamic web applications. The thesis by Matthijssen mentions several existing tools and describes the development of a tool called FireDetective, enabling full understanding of Ajax applications.

Matthijssen describes in his thesis the existing tool Firebug[1]. Firebug is a plugin for the Firefox[2] webbrowser and allows insight in the executed Javascript code and also shows Ajax calls to the web application. The plugin lacks however the possibility to show a stack trace.

FireDetective, the tool developed by Matthijssen, combines a browser plugin handling the dynamic analysis (events, requests, JavaScript executions) at the client side with a server side plugin, dynamicly analyzing the execution of the web application at the server. During the dynamic analysis excution traces are gathered client and servide side. The execution traces can grow enormously as both at client and at the server the full stack traces are gathered. Therefore the tool uses 2 trace reduction algorithms described in [8] to reduce the traces. The first reduction is the filtering of library calls to keep the trace to only contain information specific to the analyzed web application. Secondly the tool offers a start/stop button to time slice the traces gathered labeled into sections, allowing the user to view trace information about a particular interaction with the wen application.

The tool has a third component which gathers all information of the browser plugin and the server side plugin to allow visualization of the execution traces, helping the user to get an understanding of the web application. The visualization of the data is loosely based on the visions of Schneiderman in [39].

Other tools allowing insight into Ajax applications are for example a) Script Insight, using dynamic analysis to record dynamic page updates, which is described by Li et al in [27] and b) FireCrystal, allowing an user to view a timeline of dynamic page updates and events, described by Oney et al in [30]. Both tools however are not as complete as FireDetective.

[1]See http://getfirebug.com

[2]See http://www.firefox.com

2.1.4 Other methodologies

Aside form the above methodologies and tools there are a set of other methodologies which are also operating on the same area of reverse engineering web applications, however they are based on other principles. For the project they are interesting to discuss nonetheless.

2.1.4.1 ReWeb

Comibining source code analysis with reachability, flow and traversal analysis the tool generates a colour coded evolutional model of the web application. The tool is described by Ricca and Tonella in [36] and [37].

2.1.4.2 JsPick

JSPick is a tool purely targeted at reverse engineering Java Server Pages (JSP). The tool analysis existing web applications that are developed with JSP and generates a documenation of the whole system interface in an easy to read specification language. The tool also generates a GUI browser for the system, so the developer is able to browse the abstract syntax tress, type information, warning and the linkage structue of the system. The tool is proposed by Draheim et al in [19].

2.1.4.3 Revangie

Draheim et al describe in [20] Revangie, a tool performing source code independent reverse engineering. The tool operates purely on analysis of the HTML code generated by the web application and so is independent of the web application implementation. The tool recovers form-oriented analysis models in 3 modes of operation:

a) crawl mode Operating at client side, in this mode the tool works like an automated web browser. The tool starts at a specified entry URL for the application, retrieves the pages and scans for links to follow and forms to submit to continue the analysis.

b) snoop mode In this mode the tool installs itself as a proxy server to monitor the session of several users or integrates itself into the web server to monitor all the users. The tool collects data of actual sessions in this mode for analysis. The result of this mode can be used for models of navigational probability, distributions for turnaround time or sample input.

c) guided mode This mode combines the snoop and crawl modes. The tool executes in the crawl mode, however the tool asks for user input if an analysis step requires that, for example web forms with input fields.

2.2 SSO

Single Sign on systems are commonly used nowadays and are deployed for various reasons. The most known reason is off course offering your users logon credentials usable to authorize themselves with all the systems, instead of having separate credentials for each system. Having this enabled also offers the more advanced SSO possibility to allow the users to logon once and keep them logged on for all the systems instead of asking the credentials at each system again.

Starting with our own knowdledge and experience combined with the views by the Open Group[3] and the ideas of Pashalidis et al in [32] and De Clercq in [9] we describe a list of SSO implementation levels. The different levels are described in Table 2.1 togethere with the advantages per level.

SSO level	**Advantages for the user**	**Advantages for the owners**
One set of credentials	One account to remember	One list of accounts to monitor One access list to manage One central blacklist possible
One account session	Logon once only	
Central user database	Only fill in profile once	Central profile information Enforcing uniqueness easy Allows central profiling

Table 2.1: Different levels of SSO

2.2.1 Existing SSO software

There are several requirements for the SSO solution in this project:

1. Customer database functionalities (profile storage)
2. Extendable profile database (adding extra database tables)
3. API available / extendable
4. Connector for the PHP programming language
5. Storage of the profile data on servers owned by the company
6. No yearly costs for licence

At the start of the project a list of available and suitable SSO solutions where gathered and checked against the requirement for viableness, these solutions are compared in the table sumarized in Table 2.2.

Also to be noted is that OpenSSO actually has commercial support plans for their software so they will be ruled out together with Cams for this project because of the requirement of having no yearly licence costs (6).

Further JOSSO and CAS allow for extra user properties above the standard credentials in the form of a flat table but do not allow for custom tables as defined in requirement 2. OpenId finally is based on an user account of a OpenId provider, for example a google.com

[3]See http://www.opengroup.org/security/sso/sso_intro.htm

Name	licence/costs	Technology	Database	Connectors
CAS	opensource	Java	LDAP / JDBC	Java / PHP
OpenSSO	opensource	Java	LDAP	Java / PHP
OpenID	opensource	Java	application specific	Java / PHP
JOSSO	opensource	Java	LDAP	Java / PHP
Cams	commercial	Java	SQL / LDAP / XML	Java + API prepared for PHP

Table 2.2: SSO Solution comparison

or yahoo.com account so requirement 5 will not be met. There is however a possibility of creating a custom OpenId provider, however that wipes out the advantage of of being able to allow account credentials already known by consumers.

2.2.2 SSO techniques

Important in this project is the possibility to reach the data of the customers in the form of an API translating database acces to remote procedures that can be executed from authorized web applications owned by the company. A similar task is described by Del Castillo et al in [10], the book by Sneed in [40] and by Turner et al in [42].
This requirement together with the requirement of the extendability of the profile information stored in the SSO, was the reason for the decision to design a custom SSO instead of using one of the SSO's discussed in the previous section.

Armando et al describe in [4] (and in [3] they describe a flaw in the protocol) the Single Sign-On methodology in use by the Google Apps[4], the SAML 2.0 protocol. In this protocol the consumer request for a protected url at the web application, the web application then returns a redirect to the SSO with a generated string identifying the request. Then the SSO returns a form to the consumer and on form submit the credentials will be checked by the SSO and if succesfull the consumer will be redirected to the orignally requested url.

In [26] Jeong proposes another SSO scheme offering mobile users supplying their credentials to the home network for obtaining accesses to another network and allows the mobile user to store only a generated token to identify itself with on the home network as to lower the usage of resources for the mobile device. The schema is based on the SAML protocol, like with the technology of Armando et al.

Similar SSO schemes are described by Hillenbrand et al in [23] and Pashalidis et al in [31].

2.3 Migration

Works by Haller in [22], Rüping in [38] and Wagner et al in [43] all sketch the same situation that data migration in a project is not to be underestimated. According to Rüping the migration even should be considered as a seperate project. The migration software developed specifically for the project should optimally only be executed once but in practice it is

[4]See http://apps.google.com

executed more then once to ensure the requirements of the migration are met in terms of validity of the data, efficiency of the migration process or other custom requirements specific for the project.

In [44] the butterfly approach is proposed, a 5 phase model, migrating from the old system to the new sytem. In order the phases are:

1. analyzation of the legacy schema
2. create data mappings and the target schema
3. develop a sample datastore in the target system
4. incrementally migrate the system components to use the new system without migrating the data yet
5. step-by-step data migration

Finally the systems will all be switched to use the new situation.

More information about the project management aspect of data migration can be found in a book by Morris [29]. Morris focuses on the project manager view and describes the most important technical issues on a high-level overview.

2.4 Summary

In the area of reverse engineering web applications there are many tools and techniques available. We have seen the UML based techniques which analyze the web application staticly and some also dynamicly during the execution and result in an UML model of the structure. There are cross-platform methodologies available like WARE, and there are tools targeted at solely one platform like Jspick. The most interesting technique is the tool of Matthijssen, however unfortunately the tool was not available at the time the case study in the project was done. As Firebug was available at that time, we have used it in the project to investigate the execution of the web application in the browser. For analysis of the server based code a static analysis similar to the strategies found in Section 2.1.1 are used in the project.

The other methodologies based on the capture of the relational model, have also been a source to the approach for this project. In the approach we discover the relational model as a analysis method used for determining the requirements of the new database model for the SSO application.

In [4] we see the details of the SSO protocol in use by google for their web applications ”Google Apps”[5]. The researched methodology we use as a starting point for our own implementation design of the SSO functionality in the project. As an extra security measure MediaMij offers an extra separate network for the communication between the SSO application and the web application, so all data will be transferred in private.

To migrate to the new SSO environment we have seen the incremental butterfly approach. This approach will serve as a beginning for our migration approach.

[5]See http://apps.google.com

Chapter 3

Approach

In the project it became clear an approach was needed to get the project from the old situation to the new situation. The approach describes what needs to be done to transform the input to the results following the requirements found in Chapter 1. In this chapter the phases are described together with the methodologies and procedures they use to consume the inputs to produce the expected results.

3.1 Phases of engineering

We define the phases as following:

- The inventorisation phase, where the current systems are inventoried on persistent data storage and authentication integration in the actual website code
- The development phase, where the new SSO system is designed with its database and the API's are created together with a javascript client for the frontend API
- The refactoring phase, where the code of the current websites is refactored to allow connection to the SSO API
- The migration phase, where a migration plan is setup and executed to migrate and merge the current web applications and data to the new situation

In Figure 3.1 the different phases are shown in order of execution, the inputs of a phase are connected with an incoming arrow to denote the dependencies. The outgoing arrows show the results per phase. We see that the inventorisation phase depends on 3 inputs, the source code, the current database schemas and the development team, the latter by means of inteviews or demos. The inventorization generates the captured models, an use case diagram, an implementation diagram and the relational model of the existing web application as its result.

Following to the inventorization is the development phase. The development phase depends on one input, the requirements for the new situation. The phase results in the implementation of the SSO system, presented by the documented API, the new database schema and the implementation diagrams of the new SSO system.

With the SSO API developed the refactoring phase can start as it depends on the API documentation for connecting the existing web applications to the API. The result of the phase is the refactored code of the existing web application that allows for connection to the SSO API.

The final phase is the migration to the new situation, this phase needs the old and new relational models to develop the migration plan which is used during the rest of the phase. So the migration plan is a result of this phase but also an input for the rest of the phase, where the old situation is migrated to the new situation according to the steps in the migration plan.

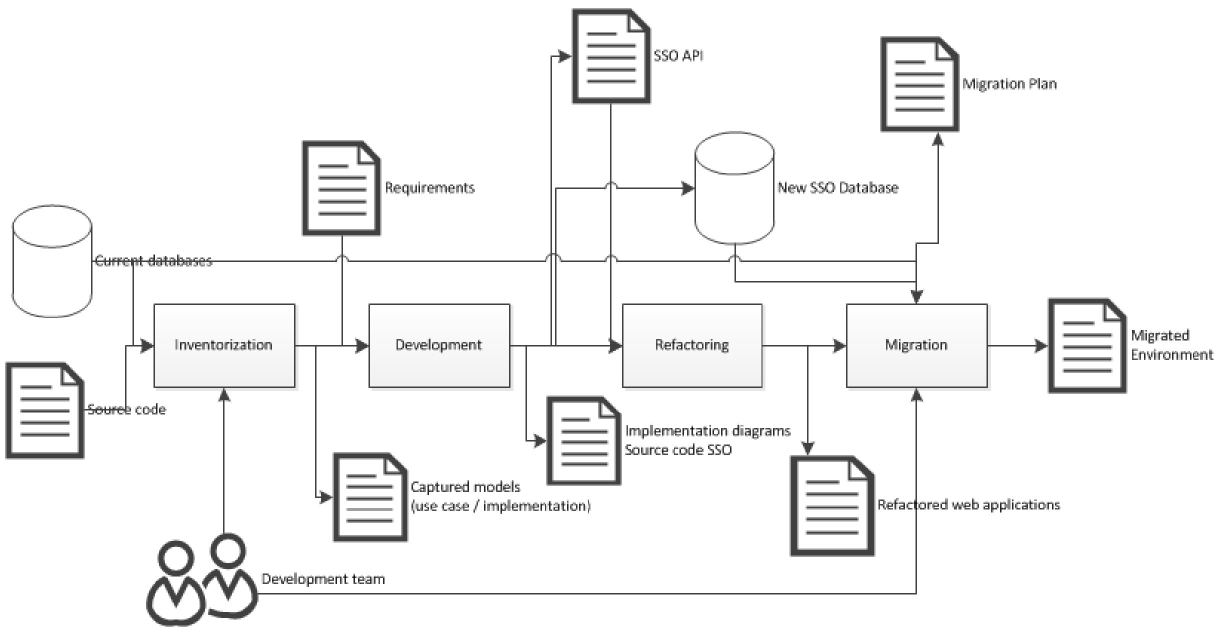

Figure 3.1: Input and result diagram of the Approach

3.1.1 Work division

The phases are further divided into subphases using the work division strategy known in Project Management into the work breakdown structure visible in Figure 3.2. The work breakdown structure is used as a core for the whole project and is referred to in its whole or to a part throughout the thesis.

In the work breakdown structure an extra item is visible, the Project management work unit. This work unit contains work items for planning and project change management to account for time that is spent not for progress in one of the phases.

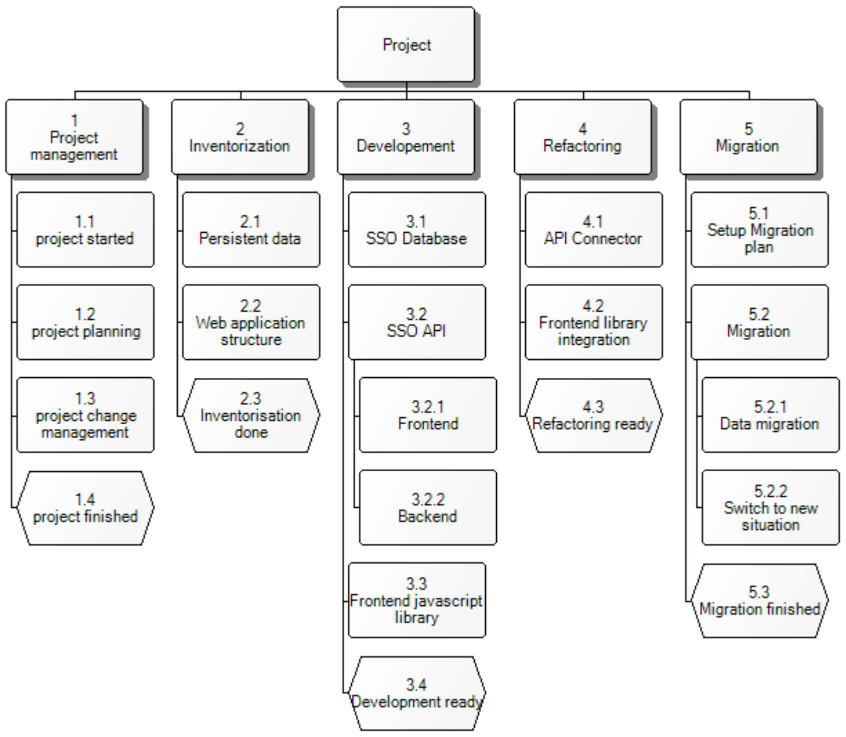

Figure 3.2: Project work breakdown structure

3.2 Inventorisation

The division of the inventorisation phase of the project visible in Figure 3.2 renders the tasks seen in Figure 3.3 which comprise the inventorisation that has to be done to determine what needs to be reverse engineered and redesigned, before continuing with the design of the new SSO system. For the inventorisation an analysis of the persistent data (2.1) is needed to be able to comprehend what is stored already and what needs to be stored in the new system. Secondly the existing websites have to be reverse engineered (2.2) to discover where the customer registration, authentication and data gathering is done in the current websites to make a list of to be re-engineered code. Completing the inventorisation brings the project to milestone 2.3.

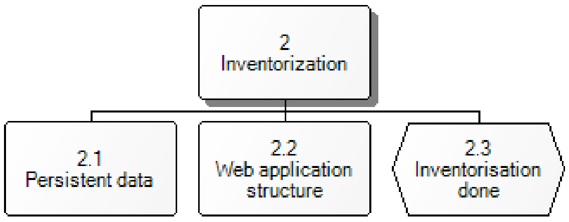

Figure 3.3: Inventorisation WBS

3.2.1 Analyzing the persistent data (2.1)

With customer databases it is important to know what kind of information needs to be stored. MediaMij depends on the information supplied by and gathered from the customers visiting and registering at their websites, reading their magazines and ordering brochures. So everything that is usefull for MediaMij to improve the process of informing the customer is required to be stored in the database. This means the database needs to be able to store a lot of varying information about an user.

As the target is a single customer database for all the customers registering at and visiting the websites of MediaMij, an inventarization is needed of all the fields used in the current databases that are used by the existing websites of MediaMij. The strategy of the project is the one of the ”Analyze the persistent data” strategy described in book [11] and is similar to the static analys done similarly by the reverse engineering methodologies described in[1] and [14]. This analysis produces a class diagram of the classes representing the user, its profile information and other classes that are related to the user which are stored in the database. This class diagram is the one of the results of the inventorisation process.

Also important to gather from the existing databases is which authentication levels exist, if they exist at all, and how we can map them to a single structure for the new database. A single authentication level for the customer is presumed together with some extra authentication levels for administrators and suppliers of MediaMij.

Functional requirement 3c and 3d state that to be developed applications of MediaMij want to be able to gather data and store data through the SSO API and so the design needs to cope with that requirement by including a seperate set of tables. The mentioned web applications are no part of the project so the design of the extra tables in the database is done in co-operation with the developerment team of MediaMij responsable for new developments. So allthough the integration of those web applications is not part of the scope of this project, the design of the extra tables is included in the final design of the new relational model.

3.2.2 Existing user database interaction (2.2)

Using the analyzed data structure, represented by the captured models in figure 3.1, a further understanding about how the database is used by the web application is obtained by further analysis of the web application code. Using the class diagram found in the analysis of the persistent data a couple of strategies are used to get the detailed implementation diagrams of the source code using the database for user profile data. The result consists of 2 diagrams, a use case diagram describing all the scenarios throughout the code that touch user profile data and finally also the implementation diagram which describes the flow through the code of those scenarios.

A few obvious interaction scenarios are introduced as a starting point for the further analysis:

- User registration
- User login
- User logout

Completion of the list is done by using a set of methodologies. At first a session is organized to meet with the current development team including the team manager, to see how they think of the project and what they know about the current scenarios touching the user profile data. This session delivers a rough list of all the scenarios that the team can think of which is to be nearly complete.

To complete the list and to get a better undertanding of the scenarios the code is studied for a fixed amount of time, according to the strategy ”Read the code in one hour”. This methodologie is similar to the strategies of the UML based reverse engineering strategies in [17], [5] and others. The code reading session delivers enought information to draw the use case diagram as a result.

Then for the third strategy a session is organized with a developer to do an interview during demo. This is necessary to complement the information found during the code reading session with information of the developer and results in data which is used to draw an initial implementation diagram.

To finish the implementation diagram a detailed understanding is needed of the code. This detailed understanding is captured by using the stepping through the execution technique of the [11], similar to the dynamic analysis of the reverse engineering strategies of the strategies discussed in Section 2.1.1. To analyze the Ajax requests that occur during the stepping though the execution, we use the Firebug tool as described in Section 2.1.3. With the understanding we then obtained the detailed implementation diagram can be filled in completeley. When the detailed implementation diagram is finished, we also have formulated a list of code locations on which the refactoring phase has to have effect as we know now where the authentication and registration functionality is handled in the website.

3.3 Development

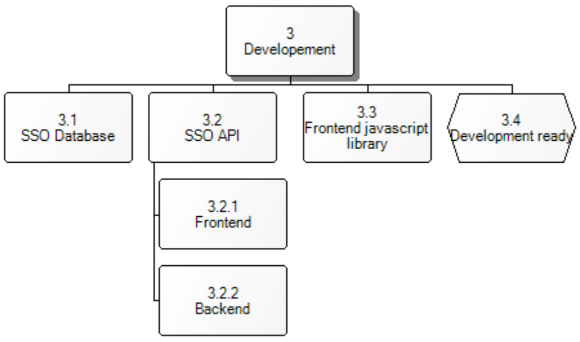

Figure 3.4: Development WBS

Figure 3.4 describe the tasks that the development phase is composed of. The development phase is divided into 3 subphases to structurize the development process. As the design of the database is the fundamental of an SSO system, this design is made in the first subphase (3.1) and approved by MediaMij before the development is continued onto the design and development of the SSO system. After the database is designed and implemented, an API is created for both backend en frontend of the SSO system (3.2). After that to reach milestone 3.4 a javascript library is designed to genericly enable websites to use the frontend API of the SSO system (3.3).

3.3.1 Database (3.1)

The input of the inventorisation phase is used to create the initial design of the database. To further enhance the design the requirements by MediaMij are processed and translated into needed extra database table columns or even tables. Then the resulting incorporates all the findings and extra requirements so it fits to all the needs of MediaMij. This design is presented as an relational database model which represents the database to be used by the SSO system.

To ensure compliance the design is checked several times during the process with MediaMij. Before transforming the design into an actual database, the final design is first submitted for approval, which ensures that the created design is flexible and suits all needs and therefore supported by Mediamij for integration in the SSO system.

Out of the first survey a basic desing of the new database is made, this design is visible in Figure 3.5. The profile table holds the credentials for all users centrally in the database, where each profile can be assigned an user role to support storing users with different access levels. Thirdly a table exists in the database to store the online status of a profile as to enable the possibility for a website to check the online state of an user.

Then the database schema is further designed by implementing the requirements found throughout the previous inventorisation phase. This process results in the complete database design for the SSO system depicted by Figure 3.5.

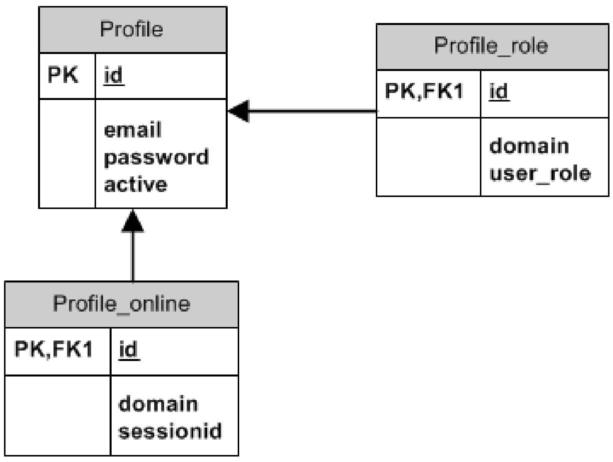

Figure 3.5: SSO Database design

3.3.2 SSO (3.2)

Using the database design as a starting point we need to define how the data will be used by the web applications to enable SSO. To design this system we used the knowledge about the Google Single Sign-On methodology described in [4] and we conducted some further analysis done by using the FireBug tool. The final design visible in Figure 3.6 shows how the web applications will communicate with the SSO system.

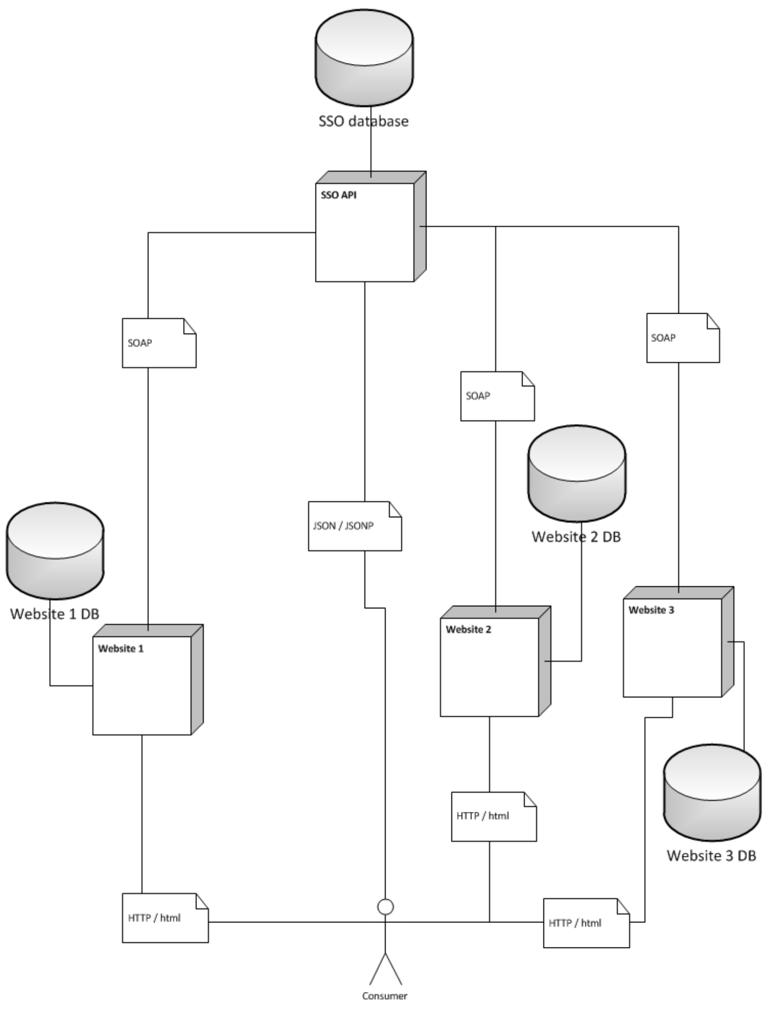

Figure 3.6: SSO design

The SSO needs to offer functionalities to the backends of the websites and functionalities to the frontend website users. To enable this without without offering each system (and the client) direct database access, a remote invocation model is designed and implemented. To separate client and backend access, 2 seperate API's are designed and developed. The frontend API is the one that is used by the browser of the customers while the backend API is restricted to be used by the websites themselves only.

The backend API is available to the websites by using the SOAP messaging format while the frontend API offers an JSON interface over XMLHttpRequests. When using this seperation adding websites in the future will be no more hassle then connecting the website to the existing API as depicted by Figure 3.6.

Registration and login are used as an example to demonstrate the design of the SSO further. Visible in Figure 3.7 shows the registration of a new user. The user submits the required form fields (e.g. email address, password) to the web application server. The web application then submits the information to the SSO using the backend API, creating a new user at the SSO system. The SSO will inform the web application if the action was successful.

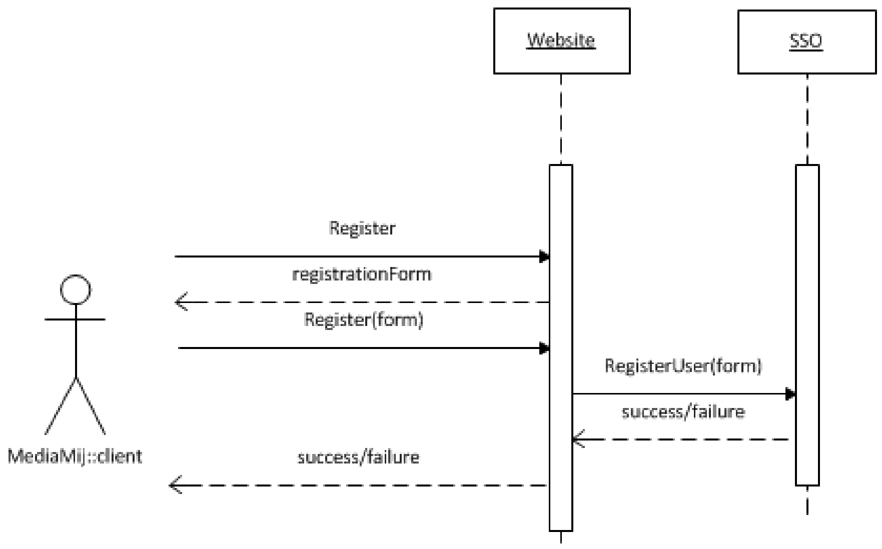

Figure 3.7: SSO register sequence

The login scenario is slightly more complicated as the login information will not be submitted directly to the web application anymore, but to the frontend API of the SSO system instead, as depicted by Figure 3.8 The SSO will return the status of the login attempt to the user and if successful the response will include the session the user obtained from the SSO system. The session information is then submitted to the web application which in turn is able to retrieve the userrole of the user based on that session.

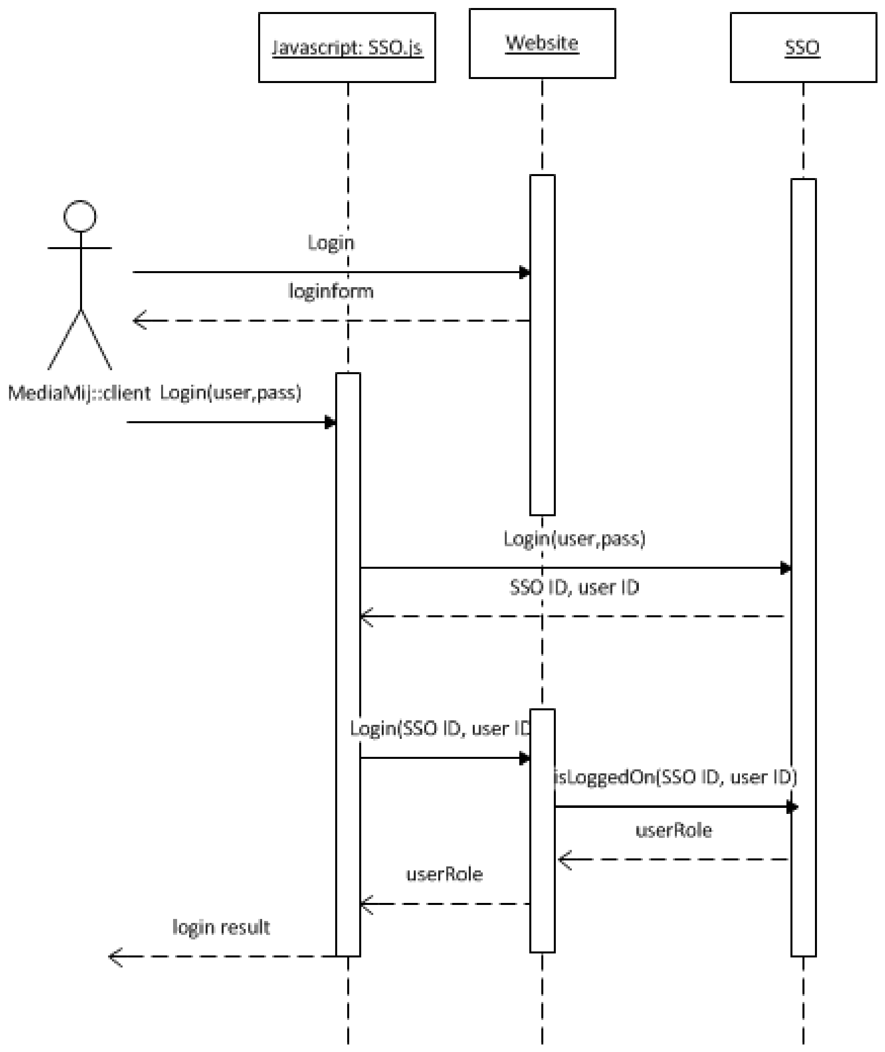

Figure 3.8: SSO login sequence

3.3.3 frontend javascript library (3.3)

As the frontend API is reachable by XMLHttpRequests, it is easy to create a generic frontend library in the javascript language. This technique is supported by most client software and offers easy access to XMLHttp Webservices with json data. The result of this subphase is the javascript client enabling access to the frontend API, which is used during the refactorings of the web applications. Also future websites of MediaMij will be able to use this client to connect to the frontend API.

3.4 Refactoring the existing websites

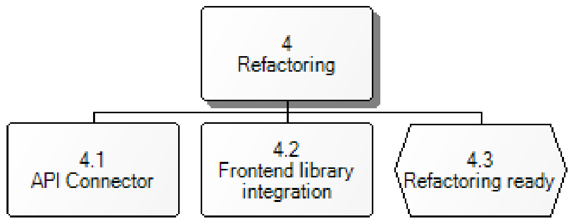

Figure 3.9: Refactoring WBS

As we obtained a working SSO system with its API's offering the functionalities needed to enable the SSO process and centralized profile information storage for the existing websites of MediaMij, the last step is refactoring the old websites to use the new SSO system, this process consists of 2 tasks as visible in Figure 3.9. These 2 tasks are connecting the websites of welke.nl and huisplan.com to the backend API of the SSO system (4.1) and the integration of the frontend javascript library to connect to the frontend API (4.2) to enable its usage by the websites. This implementation of the SSO within the existing websites is visible in Figure 3.10 and completes the refactoring phase, reaching Milestone 5.3.

3.4.1 Backend API connector

As both the websites are now written in the PHP language we can instead of duplicating the SSO backend integration code, create a simple PHP library that connects to the backend SSO API and offers its functionalities to the website backend. The library consists of all the functionality needed to use the full possibilities of the backend API of the SSO, so the existing web application is able to use this library to handle all calls to the user database residing on the SSO system.

The backend API connector is depicted by the SSOController component in Figure 3.10. The RegistrationForm, if the user submitted valid data, calls SSOController to write the registration of the new user to the database. Instead of writing to the local database, SSOController uses the backend API of the SSO to register the new user in the SSO system instead, by calling the api function ”RegisterUser”. Alternatively if the user object needs to be updated, the SSOController is also leading and uses the ”updateUserData” API call to record the update in the SSO system. The third scenario visible in Figure 3.10 is the user login, if the user succesfully logged on the web application will check the userrole of the user by means of the ”isLoggedOn” API call, this is visualized by the call of the CheckLogin component to the SSOController executing the actual API call.

3.4.2 frontend library integration

In the frontend of the website, the already created frontend javascript library is integrated and is configured to connect to the SSO on the one hand and to the request handler within the website itself on the other hand. This ensures the clients now authenticate through the SSO system.

The frontend API connection is visualized as the Client component block in in Figure 3.10, with in the center of the block the SSO.js component presenting the javascript library. The library sends login and logout calls to the FrontendAPI of the SSO by using the corresponding API calls.

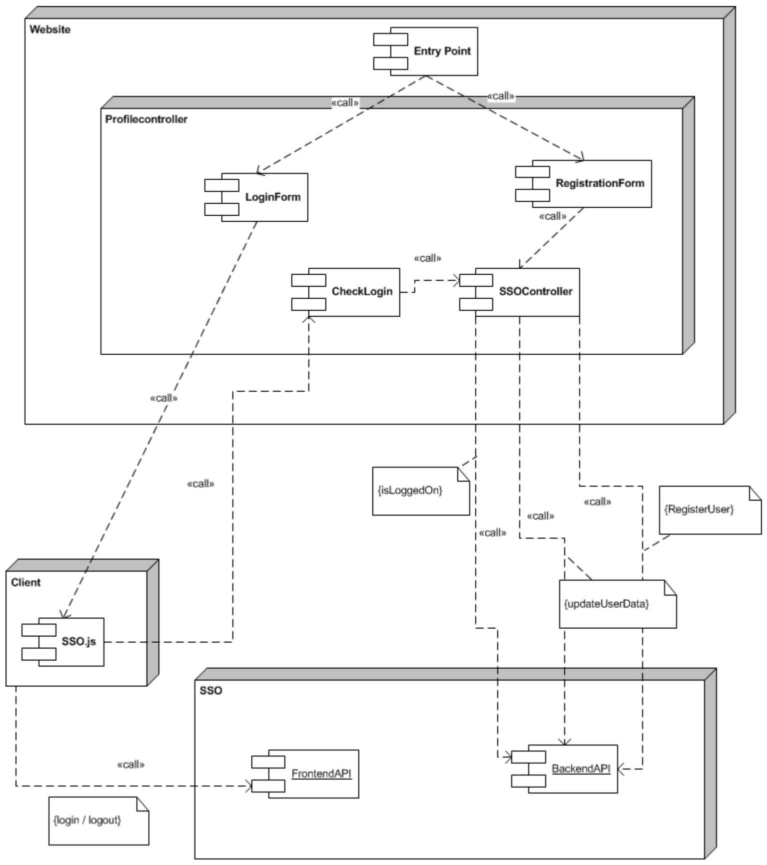

Figure 3.10: Generic SSO implementation within the website

3.5 Migration

As important as the design of the database is the migration of the data to the new system.The existing data consists of profile information gathered on multiple websites of MediaMij over a total span of at least 5 years. This is crucial marketing information for MediaMij which they want to use in the to be developed data analyzation tool.

The tasks comprising the migration are visible in Figure 3.11.

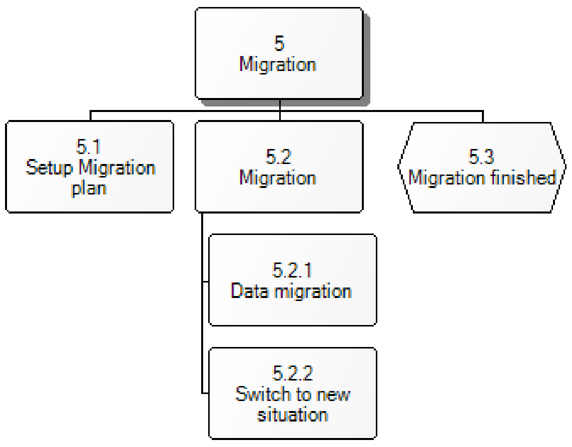

Figure 3.11: Migration WBS

To ensure a successful migration techniques of book [11] are used. Important is to involve the users within the company as they would be using the new system for their future projects, but also the management was involved as they wanted to have more insight into the data submitted by the users of the website, as they want to use the information for profiling and marketing processes. This involvement was the key to success of the migration as the satisfaction of the internal users of MediaMij is a requirement.

To ensure complete satisfaction the migration is done incrementally, similar to the butterfly method in [44], to offer a rollback opportunity after each migration if the new system was not satisfactory yet or would not be satisfactory at all. Both cases would then be reason to improve the system and then try the migration again. The incremental steps are described in a migration plan (4.1), which also details more information regarding the migration. This plan defines which data from which table in the database are copied to which column or row or cell in which table in the new database. Important here is to define where there may be migration problems like duplicate profiles in different databases. This triggers situations where manual intervention is needed to decide which data is preferred, destroying the possibility of automating the process. Therefore a strategy is developed to select which data is said to be more recent or more complete and so a set of rules to circumvent those issues are included in the plan.

In the schedule also the discussed milestones or rollback points are mentioned, these denote the finishing of a migration and the opportunity for a rollback if the system is not satisfactory. Another part of the involvement of MediaMij is the review of the migration plan before continuing with the actual migration, so everything is according to MediaMij's needs. Finishing this migration moves the project to Milestone 5.3.

Chapter 4

Case Studies: 2 MediaMij Websites

These case studies describe the progress of the approach of this project applied to 2 existing MediaMij websites, `http://www.huisplan.com` and `http://www.welke.nl`. It also documents the design and development of the central SSO system and its API. The primary goal of the the case studies is to test the applicability of the approach depicted by Chapter 3 to real web applications. A secondary goal is to see if the rendered situation implements all the requirements of Section 1.2.

4.1 Welke.nl

Welke.nl is a website directly targeted at customers who are or looking for news on products related to living, eg. bathrooms, kitchens, etc., or are looking to inspire themselves how to design/decorate the various areas of their houses or are ultimately looking for specifications and prices of products there are available for them to buy.

The website is divided into the subsections news, inspiration and products. Secondly the website offers a kitchen an bathroom designer, which lets the customer create designs for their bathroom or kitchen online, which are viewable in an 3d viewer. These designs also may be sent to a professional for assistance or questions and they will ultimately send the customer to a showroom to buy their new bathroom or kitchen.

An other section of the website offers ordering modules for the magazines related to the website, which is a paid service, but also for ordering free brochures of partners of MediaMij with product information, like the new kitchen designs of a certain brand.

During the setup of the project some hours were spent talking to the developers and setting up the sessions needed to get all the information needed to start with the inventorization phase. During the first sessions basic information about the current website and its technologies was gathered.

The welke.nl website was built with the JAVA programming language and was developed against an open source cms also running on JAVA technology, the developers however mentioned the website uses ajax technology a lot for loading dynamic content, api requests of the 3d designer and for the customer registration module.

The database is split in two databases, one based on mysql technology used for product information and customer information and one based on a java content repository used by the cms to store articles and images. In the mysql database the customer information is stored and according to the developers 2 types of users that could login at the website, namely the customers and the advisors who would be able in the future to comment on designs made by the customers, are stored.

The whole website runs on the model view controller principle, meaning a controller object existed for a group of actions (eg customer registration), which uses the model, the object presentation of the database table for the customer, and represented the result of the action based on the available view, which were written in jsp. For integrating the mvc principle in the website, the technology of the open source project Spring MVC is used[1], and it also supported by annotation usage to define the entry points for web site request handling (eg /register maps to profileController::doregister()).

The communication between the model objects and the database tables was handled by the technology of the opensource project Hibernate is used[2], likewise with the spring annotation, hibernate annotation makes it possible to denote which table corresponds to the model object and the mapping of attributes to columns.

As we obtained sufficient knowledge of the website and its technologies a deeper investigation was made to discover the exact representation of the persistent data, the data flow in the code and the code operating on customer data. This is necessary to see where the adjustments to the code have to be made to migrate the website to the usage of the SSO environment.

4.1.1 New website

During the reverse engineering of the welke.nl website there were already rumours in the company that the website would be redeveloped. As the website was built in cooperation with an external company, when the relation between the companies cooled down, MediaMij debated internally how to proceed with the website as half of the requested functionality still was not implemented and the requirements list grew as fast as the delay on the still to be delivered functionalities.

MediaMij had to decide whether to continue developing the website and cms delivered by the external company or to start from scratch. The website and cms both are developed in the JAVA language and as the company only has one developer with enough JAVA experience to continue on the development, they had to hire more employees if they wanted to continue developing on the existing website. Otherwise MediaMij could develop a new website in php, which all developers had mastered as the other website of Mediamij, huisplan.com, already was developed in PHP. Mostly due to the economical of hiring more developers, MediaMij decided to abandon the existing website and to start developing a new website in PHP.

[1]See: http://www.springsource.org

[2]See: http://www.hibernate.org

Developing the website and cms from scratch had a major advantage for this project, namely the authentication module could be designed in such a way that switching to the newly developed SSO environment wouldn't cost much time. Also the database model for the storage of customer data would be designed in a way that the data can be migrated to the SSO database allmost by a 1 on 1 copy action.

Also for the development team the rewrite had a major advantage, they now only had to deal with one technology for all websites and were motivated to generalize the new cms in such a way it can be immediately used for the other websites as well in the future. So that all websites would profit from improvements made in the new general cms of MediaMij. The project motivated the development team to build a clean an modular cms as they were left a lot of freedom in the design and it would be their own cms. For the project is was also an advantage as the code that had to be refactored to let the website use the SSO system would be significant less and for sure a lot more documented and structured.

However at this point in time, the inventorisation of the existing welke.nl website had already been executed, so the results are documented in the case study to state how the website used the authorization and registration functionalities, as the fundamentals of the customer profiles of the website wouldn't change.

4.2 Huisplan.com

Huisplan.com is a website targeted at customers who are in the process of buying a house, addressing questions in the areas of negotiations, real estate agents, mortgages and many more. The website contains a lot of information on the mentioned problems and gives the customers insight into how to approach certain aspects of buying a house. The website also offers the current interest rates for a lot of known mortgage providers. But more importantly customers can fill in little forms, called tools, on the website to determine their monthly costs, maximum mortgage, etc.

The developers of the website indicated that we encounter again 2 types of users that can log in, namely the customer wishing to search for information, but also a real estate agent that can register new customers for the website. Although these 2 types of users existed, the real estate agent user should be left out of scope as it is not used anymore, which leads us to one usertype to use in the analysis of the application.

All the data of the website (articles, customer profile information) is stored in a single mysql based database. Both the website and CMS are developed in the PHP language and do not use any libraries out of the standard php functional library, and furthermore no general framework or standard CMS product was used, meaning all code was developed custom for huisplan.com. And also huisplan.com uses ajax technology for functionalities like login and registration and dynamic content.

4.3 Inventorisation

The inventorisation of the 2 websites of MediaMij is described in the following sections, the process follows the approach and follows the structure that is defined in Figure 3.3.

4.3.1 Welke.nl

4.3.1.1 Persistent data (work unit 2.1.1))

The database used by the welke.nl website can be divided in parts. At first there is the content repository used by the CMS to store news, backgrounds and the corresponding images. Secondly we have the bathroom and kitchen products in the mysql database, also in the mysql database we have the tables that are the most important for the project, namely the customer tables and the tables in use for customer generated content.

The cms repository and the product database are left out of scope and are not included in the designs as they do not include any information about the customer registration or authorization processes.

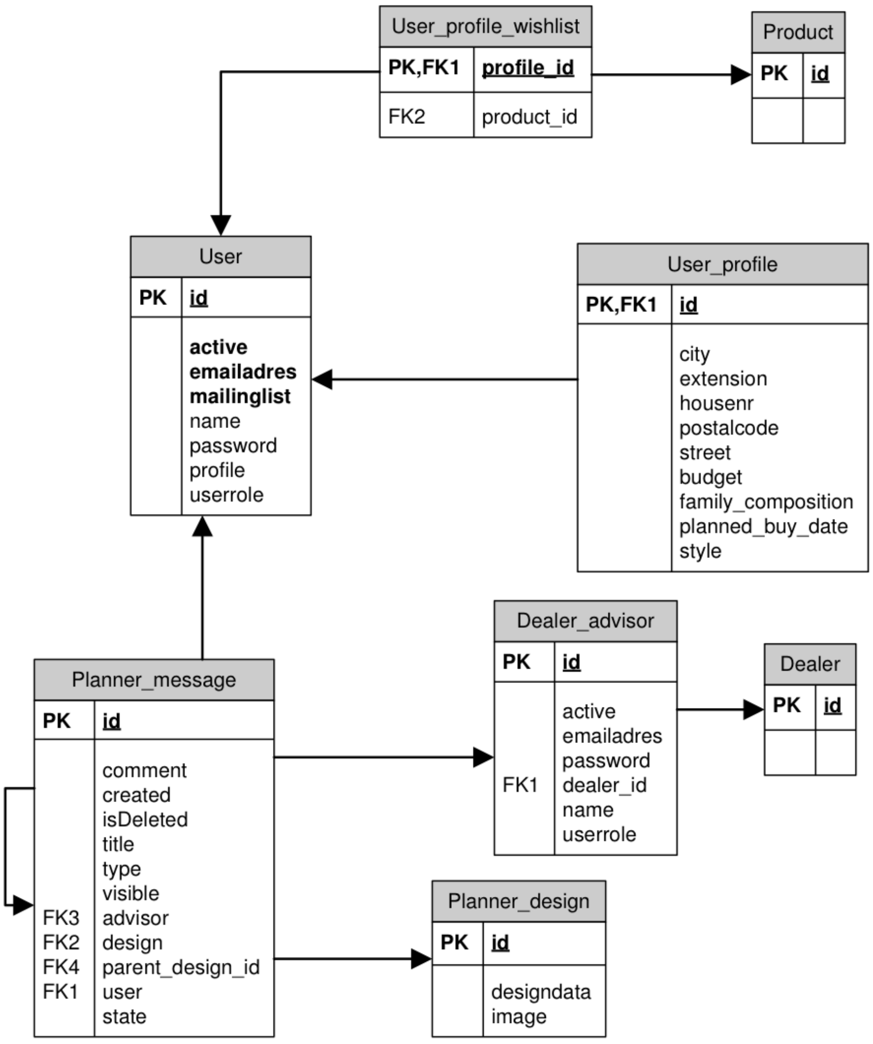

Figure 4.1: Database design schema of old version of welke.nl

Analyzing the persistent data resulted in the database schema visibile in Figure 4.1. As seen in the schema the user table contains the basic information to identify a customer user with its credentials and information about the status of the account, namely the activety status in the active field and in field mailinglist optin stores if the customer wants to recieve mailings or not. As seen in the diagram the table also includes a field to store the authorization level of the user in ”userrole”. However none of the rows in the table set a value for this field so we conclude the field is not in use at the moment.

We see that the relation between the profile and user tables is 1 on 1, so the profile table is in fact an extension to the user table and here more detailed data of the customer is stored. This table stores address information, family composition, preferred style and some more profiling information typical for users of the welke.nl website.

As the website offers a simple form of favorite products, the database has a table that stores unique user identifier with unique product identifier pairs to allow that functionality.

A different type of user of the website, namely an advisor, is stored in a seperate table in the database, dealer_advisor. Only basic information of this user is stored in the table, no address information, as that can be retrieved by querying the corresponding dealer table.

The planner_message table allows for an entity to be saved that has references to itself, to enable threaded communication, and links to the customer user or advisor user that saved the design. The actual design is found in the planner_design table that is also linked by the planner_message table consisting of the design in xml form and a preview image.

4.3.1.2 Website structure (work unit 2.2.1)

After determining the persistent data, the next step is to investigate the structure of the website. This is to discover where the login/authorization processes where hidden and so to discover the integration points. Following the approach a session was organized with the current developers of the website, including the manager of the team. This session delivered scenario’s involving the creation, authorization and altering of user information for a customer browsing the website.

For instance during the session the registration process was discussed thoroughly, discovering the fact that registration of a new user will first create the account and email the registration details to the user. The email consists of a link to the website confirming the registration of the account, which is needed before the user can actually use his account.

This list of scenario’s was then used as input for a code reading session to verify the scenario’s in the code and to discover if some were missed or left out during the session with the development team. The code reading was set for a maximum of 2 hours as at first the scenario’s already known were located in the code and then starting at the code altering or querying the database for the user tables an investigation for more scenario’s was done.

Finally with the information gathered at this point a diagram was made that resembles an use case diagram and points out the various use cases in the website touching the user databases, which is visible in Figure 4.2.

The diagram shows the various scenarios available in the website. The registration action as learned from the developers during the first session, is split in 2 separate scenarios, registration and Activation. Registration consists of the user filling in a form, which is validated and if valid the data is stored in the database and an email is sent out to the user.

Activation is triggered by an user clicking on the activation link in the email he or she received after registration. The link will trigger an update of the account enabling access and will automatically login the user to the website.

When an user request for the login form or for an action not available to anonymous users, the login form will be presented. After validation the login form is validated by Spring Security[3] and if successful the current page will be refreshed.

Aside from the website there is another website running on a separate domain, `http://bestel.welke.nl`. This subwebsite offers the brochure and magazine ordering features to the user, but use a totally different codebase. As the mentioned ordering system is not integrated in the codebase of the website, the inventorisation will not deal with this system. Toge Due to the non-integration of these ordering modules, they are further left out of scope of the project as they will be picked up in a seperate project by the company later on as they want to redesign and redevelop them anyway from scratch.

[3]See: http://static.springsource.org/spring-security/site

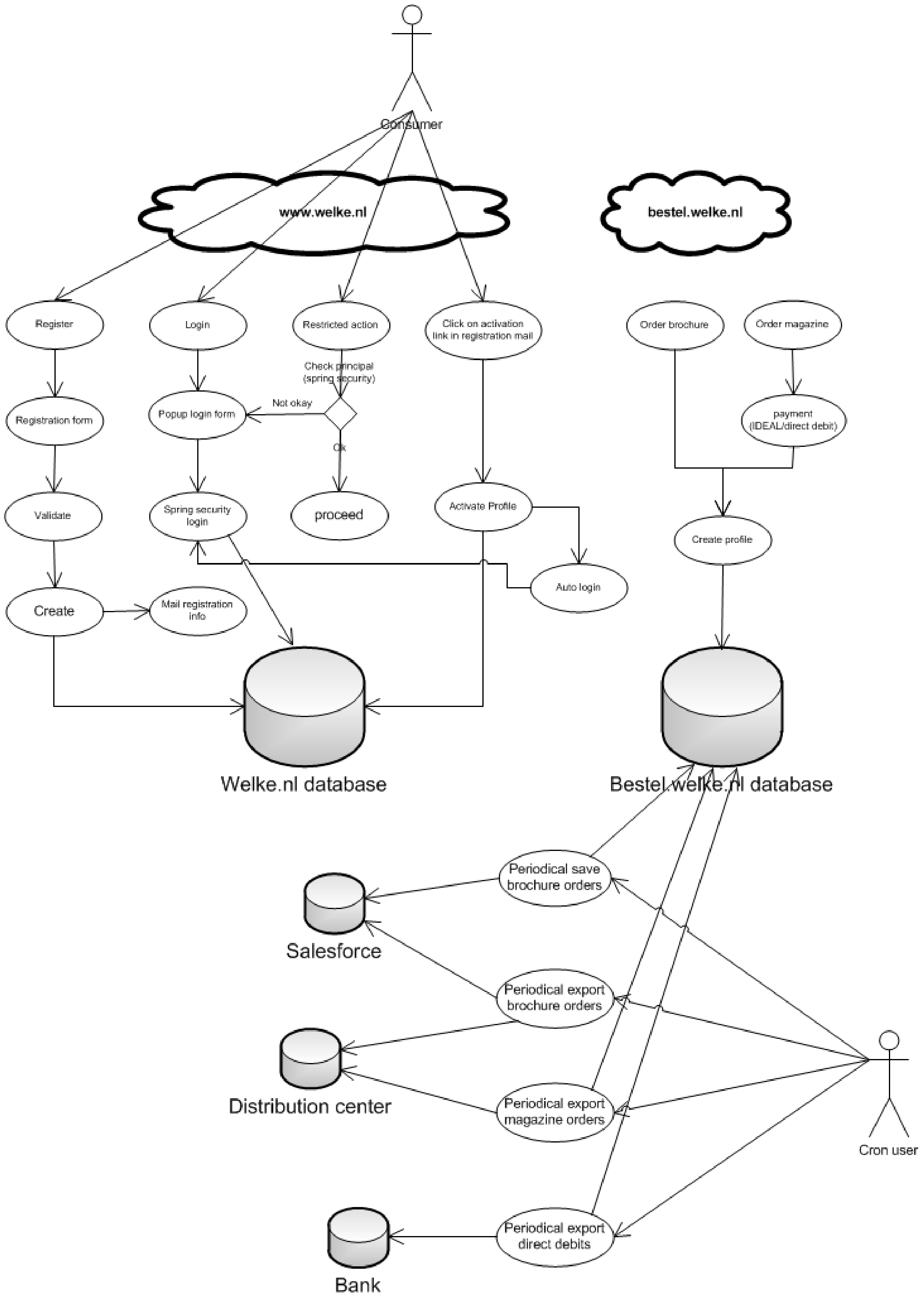

Figure 4.2: Use case structure of old version of welke.nl

As a detailed understanding is needed to discover the integration points in the code, the above diagram does not suffice. To get a better understanding an interview with the development team was held during a demonstration of the website. This identified subsystems of the website and offers entry points for the deeper investigation done by stepping through the code. Stepping through the code delivered the actual flow of data through the website and shows the code touching the database in detail (entry points, method names, etc). With this information the implementation diagram was graphed and visible in Figure 4.3.

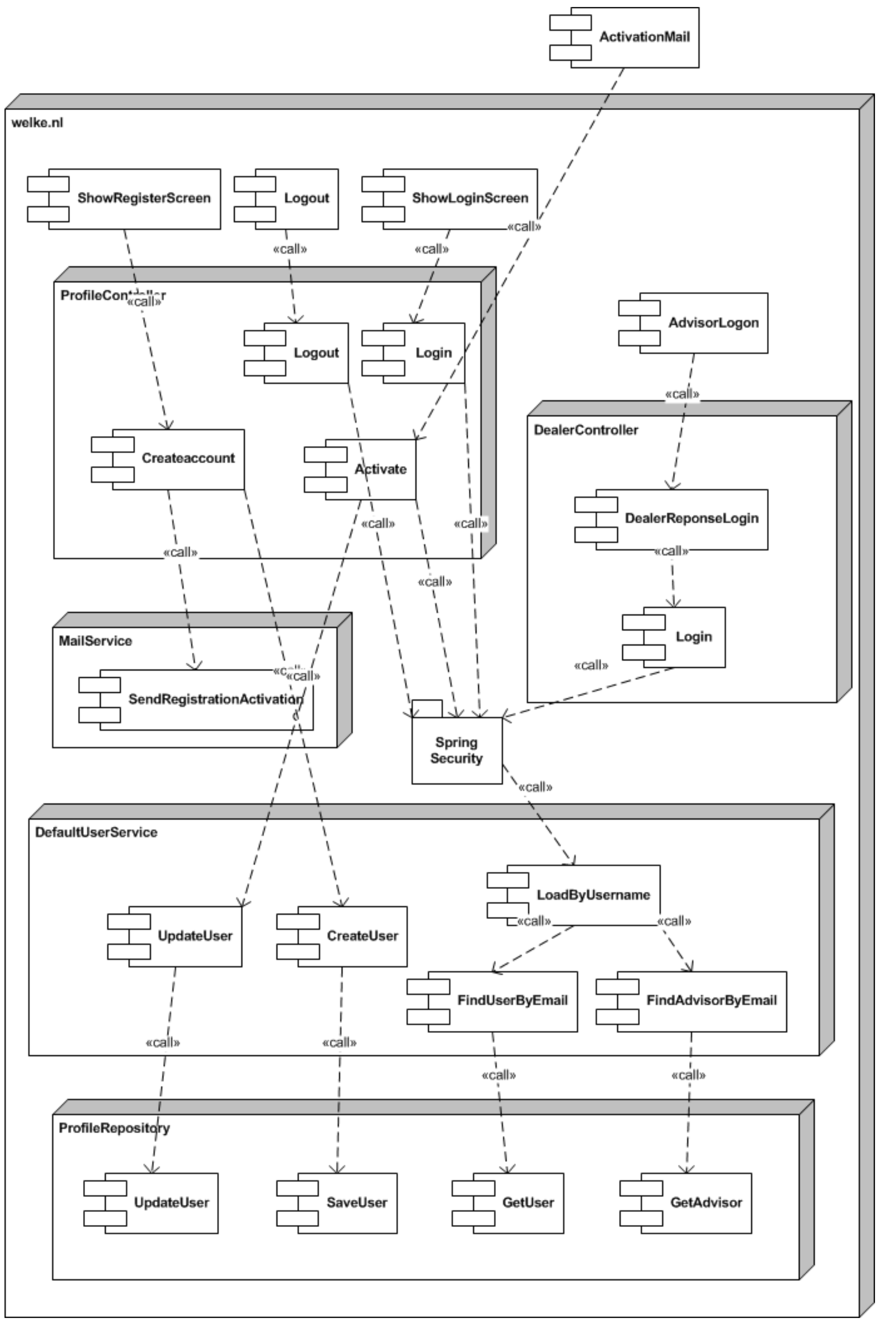

Figure 4.3: Implementation details of old version of welke.nl

The most important packages in the implementation schema are the ProfileController, DealerController and DefaultUserService. The ProfileController and DealerController are handling the action an user executes at the websites. They offer functionality for registering a new customer user and offer the functionality to login/logout an user (be it customer the

ProfileController kicks in, be it an advisor then the DealerController handles the request).

Creation and alteration of an user is done by calling the respectable functions in the DealerUserService, which in turn communicates with the ProfileRepository. The ProfileRepository operates on the database source by using the functionalities of the java library Hibernate.

Login and logout actions are handled by the java library spring security, which keeps track of the authentication session. On login it loads the user object from the database by calling the functionality of the DefaultUserService.

During the sessions it became clear that the customer information of the brochure and magazine ordering modules of the website do not store their information in the welke.nl database but relay their data to a separate database for the ordering information. This information alone was enough to set up another session with the development team for clarification.

In this session the decision was made to leave the ordering database out of scope of the project as it would not be simply merging the database with the main welke.nl database as 3 different operations are dependent on the ordering information and so migrating the data would mean a complete overhaul of these systems, resulting in a project duration completely out of proportion. The mentioned migration and refactoring was put on the schedule of MediaMij for a project to start after the SSO migration.

4.3.2 Huisplan.com

4.3.2.1 Persistent data (work unit 2.1.2)

Huisplan.com has a single mysql database containing all the information shown and processed by the website. According to the developers the website however stores more information about the customers, as it places the customer in different interest area's and stores all information entered by a customer in the calculation tools available on the website. Analyzing the database discovered the design visible in Figure 4.4 and leaves out the tables not concerning customer information.

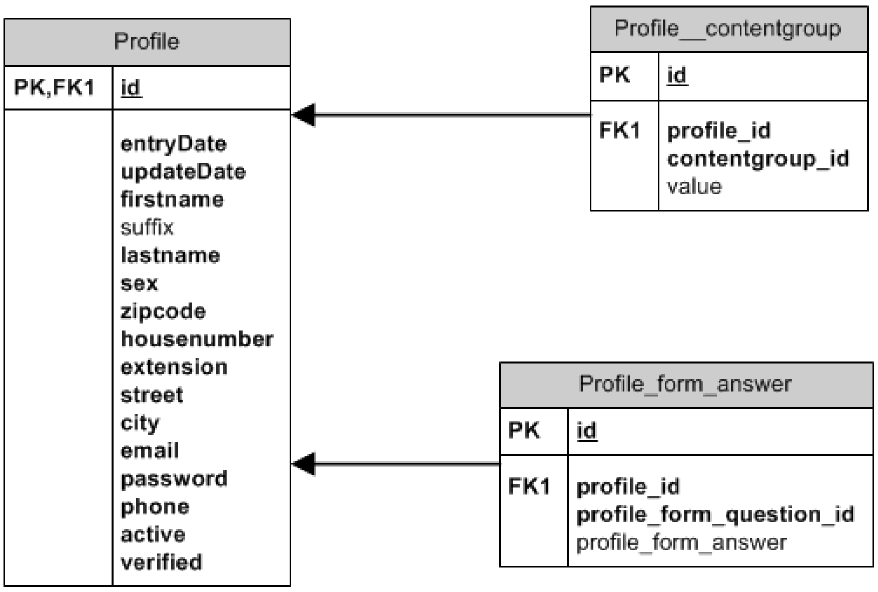

Figure 4.4: Database design schema of old version of huisplan.com

Analyzing the diagram, we see a profile table containing the basic profile information of the customer, which columns have recognizable naming so there are no columns with information that is not clear. However the other 2 tables were not clear at first sight and had to be discussed with the developers of MediaMij. In short the contentgroup table places a profile in a special interest group, e.g. if an customer submits at the huisplan.com he or she wants mortgage information, they are linked to the mortgage special interest group.

The form answer table was made clear to be used as a storage of all the answers of an customer to the extended registration form of the huisplan.com and is filled for reference usage at the moment, but meant for future profiling functionalities.

4.3.2.2 Website structure (work unit 2.2.2)

As with welke.nl the integration points of the website with authentication are important to discover for the project. So for huisplan.com during the first meet with the development team the various scenario's are discovered. These scenario's were then further verified and complemented by reading the code to get the information needed to get an understanding of the scenario's in the website querying or updating customer data in the database. The both sessions resulted in an use case alike diagram visible in Figure 4.5.

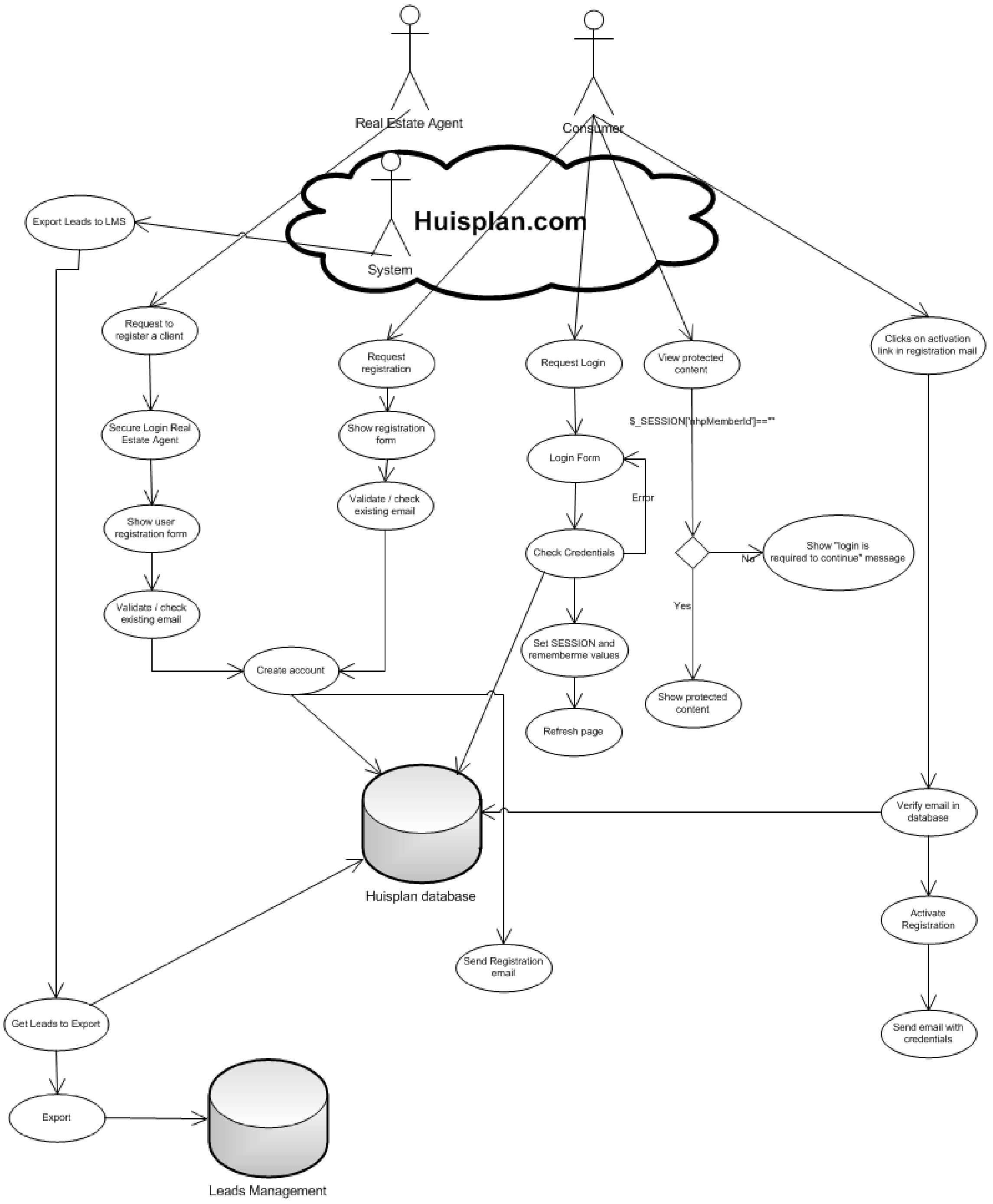

Figure 4.5: Use case structure of old version of huisplan.coml

During the interview during demo to confirm the results the manager of the development team mentioned that the scenario of registering a customer by a real estate agent as visible in the diagram, would no longer be used as all registrations would be done on the website itself from now on.

The rest of the use cases show like with welke.nl the processes the system undertakes for registering the customer, validating the input, sending out activation emails and saving everything to the database. The huisplan.com website offers an extra option for the the leads management system to retrieve an overview of the new registrations which may be interested to be contacted for commecial activities, this overview is downloaded and saved to a seperated database of the LMS itself.

As with welke.nl, the use case diagram is only a general overview of how the authentication works in the website, so to get a detailed understanding, an implementation diagram is needed. To capture the detailed implementation diagram stepping through the code was needed to get to the wanted level of detail and the implementation diagram visible in Figure 4.6.

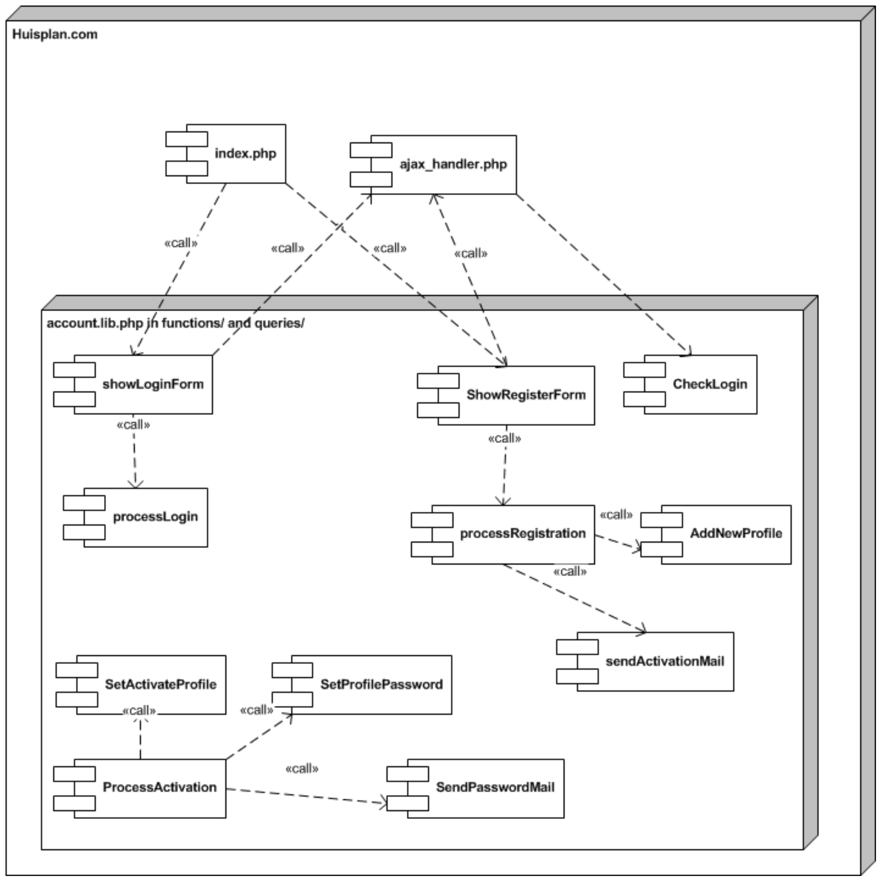

Figure 4.6: Implementation details of old version of huisplan.com

In the implementation diagram it is clear that a customer requests the login or registration form at the website. The login form communicates with the ajax_handler.php which in turn calls the checkLogin functionality to check the supplied credentials. The registration form submits the data to the processRegistration function and that one executes the creation of the user and the sending of the activation email.

For the activation process the function ProcessActivation handles the request, it sets the profile to active, generates a password and then finally it emails the new user again with the registration details including the generated password.

4.4 Development

To facilitate the SSO customer database, a new system had to be developed featuring a database which can hold all the information required by MediaMij. As SOAP was chosen as the favored technology to offer the sign on services to the existing websites, a soap API was designed to offer the services required. On the other hand a JSONP HTTP API was designed to offer authentication for users of the websites with the SSO.

At first the data model is described to clarify how the requirements of MediaMij translate to a persistence model and secondly the api is discussed, as to follow the process defined in Figure 3.4.

4.4.1 Data model (work unit 3.1

The database model needs to store all user related information of users that visit the websites of MediaMij. This includes all data submitted directly by the consumer for example when registering, when submitting a survey of when using one of the tools of huisplan.com. However the database also needs to store generated data, like visit and behaviour statistics, as per requirement 1c) in Section 1.2. There a database model was developed that is able to store all information MediaMij wants to know about an user now but also suits for storage requirements in the future.

The information to be stored is categorized into:

- basic profile information (address, email, password, etc)
- survey and profiling data (extra registration information like income, house costs and also tool input from eg. the maximum mortgage calculator)
- behavioural data (page hits, path of visitors through a website, survey and calculator usages)
- concluded data (the data generated by the analysis tool in development by MediaMij)

As the data has no standardized structure, a separate table for each category is implemented which all are linked on basis of the customer id. It is to be noted here that this model is not optimal for data analysis as it is not possible to see all information of one customer in one query.

The separate tables for each category of profile information also hold a link to the website the information was gathered, if available. Finally the database has a table for storing the active user sessions. This table enables the websites synchronizing the online state of an user. The resulting model is depicted by Figure 4.7

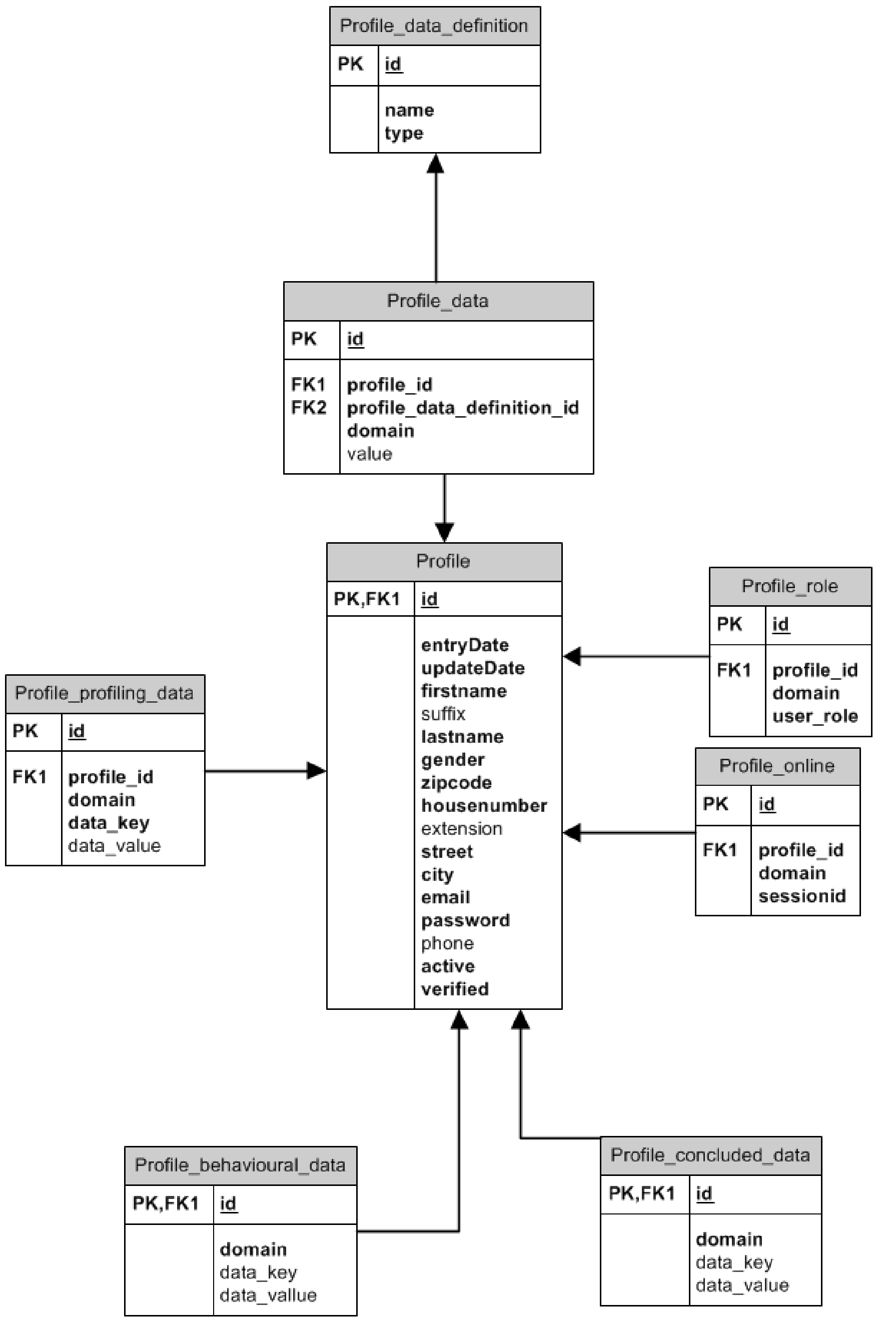

Figure 4.7: Database design for the SSO system

4.4.2 SSO API (work unit 3.2)

Figure 4.8 describes the system setup for the situation where the SSO system is implemented. Both websites of MediaMij have their own databases they use for storing the website data and profile data that is decided not to be stored centrally. The most important data that is stored not centrally but in the database of the web application, are customer generated files like submitted kitchen and bathroom designs. For the development of the websites the decision was made that it would be less complex to have those files stored at the website instead of centrally with the user profile in the SSO system.

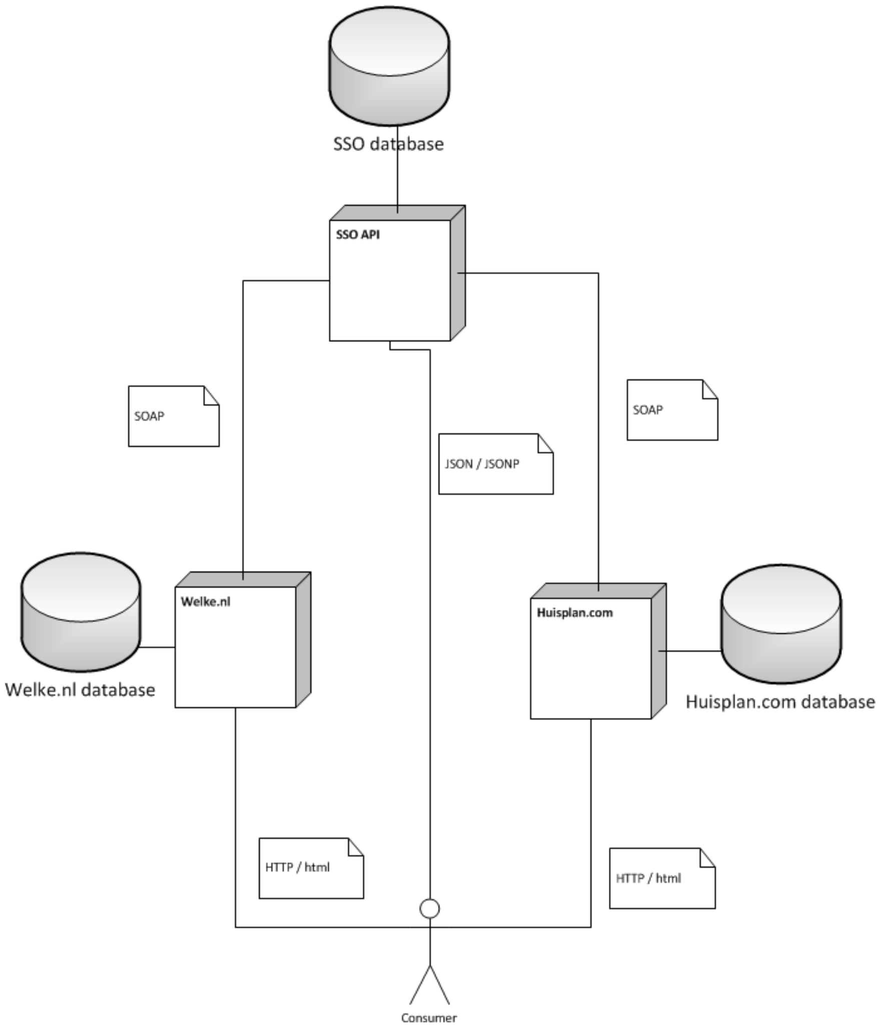

Figure 4.8: SSO system integration

The SSO system on top has its own database for storing the profile data and offers access through SOAP for the websites and for clients it offers JSON/JSONP access.

As described the API of the SSO system is to be divided into 2 seperate API's reachable by the different users. The frontend API is reachable by the website users, offering only login and logout functionalities so the user can authorize himself with the SSO System.

Secondly the backend API, reachable only by the websites of MediaMij, offers the functionalities for registration and administration of the website users. The functionalities

consist of checking the user role, registering a new user, updating user data and administration of profiling data. The whole API is setup to be extensible to the future needs of MediaMij.
As the complete description of the API is a bit lengthy for inclusion in the text, both API's are described extensively in Appendix B, where the backend API can be found in Section B.1 and the frontend api in Section B.2.

4.4.3 frontend javascript library (work unit 3.3)

The frontend library offers all functionality for the websites to enable login and logout through the SSO for its users. The library connects to the frontend API of the SSO system and follows the new logon procedure as shown in Figure 4.9

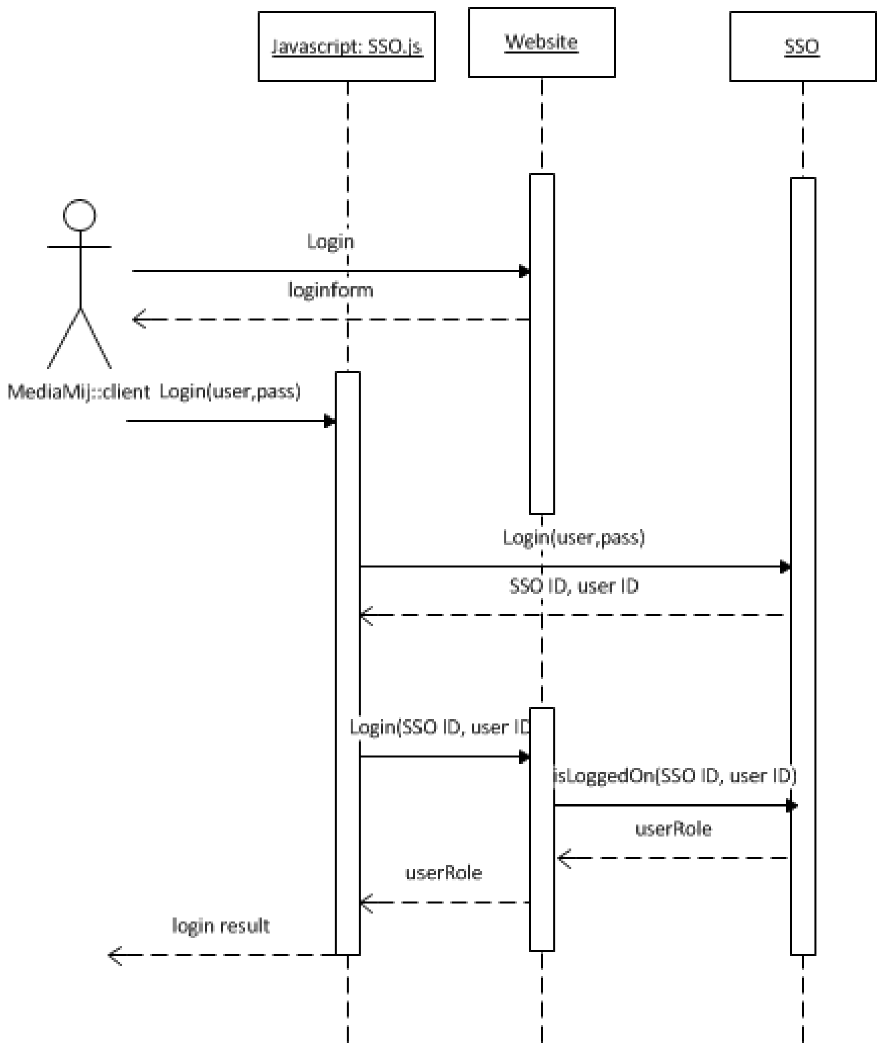

Figure 4.9: SSO login sequence

The user first requests the loginform and then submits the credentials directly to the SSO system. The SSO returns the session ID created for the user and the user ID, these values are submitted by the javascript library to the website. The website then asks the SSO for the userRole by supplying the SSO ID and SSO ID and if successfully the system knows the user and its authentication level.

On a logoff request the SSO is informed as well so it can destroy the SSO session of the user.

4.5 Refactoring

The refactoring of the websites of welke.nl and huisplan.com follows the process defined in Figure 3.9.

4.5.1 Welke.nl (work unit 4.1 / 4.2)

The refactoring for welke.nl was minimized due to the fact that during the development of the new website the authentication and user data collection parts were modularised and documented in such a way the registration and login code can easily be replaced. The replacement code connects to the SSO system instead of using the local database of the web application. So when the replacement code is in place the web application will use the SSO backend API for storing and retrieving profile data.

Seperately a javascript library is developed for usage on the frontend. This library enables usage of the SSO frontend API to logon and logoff customers through the SSO system. The library also provides a callback functionality with the login / logoff result to submit to the website for handling. Eg. on a successfull logon the website has to discover the user role through the SSO backend API by supplying the SSO session ID and user ID that is received through the callback.

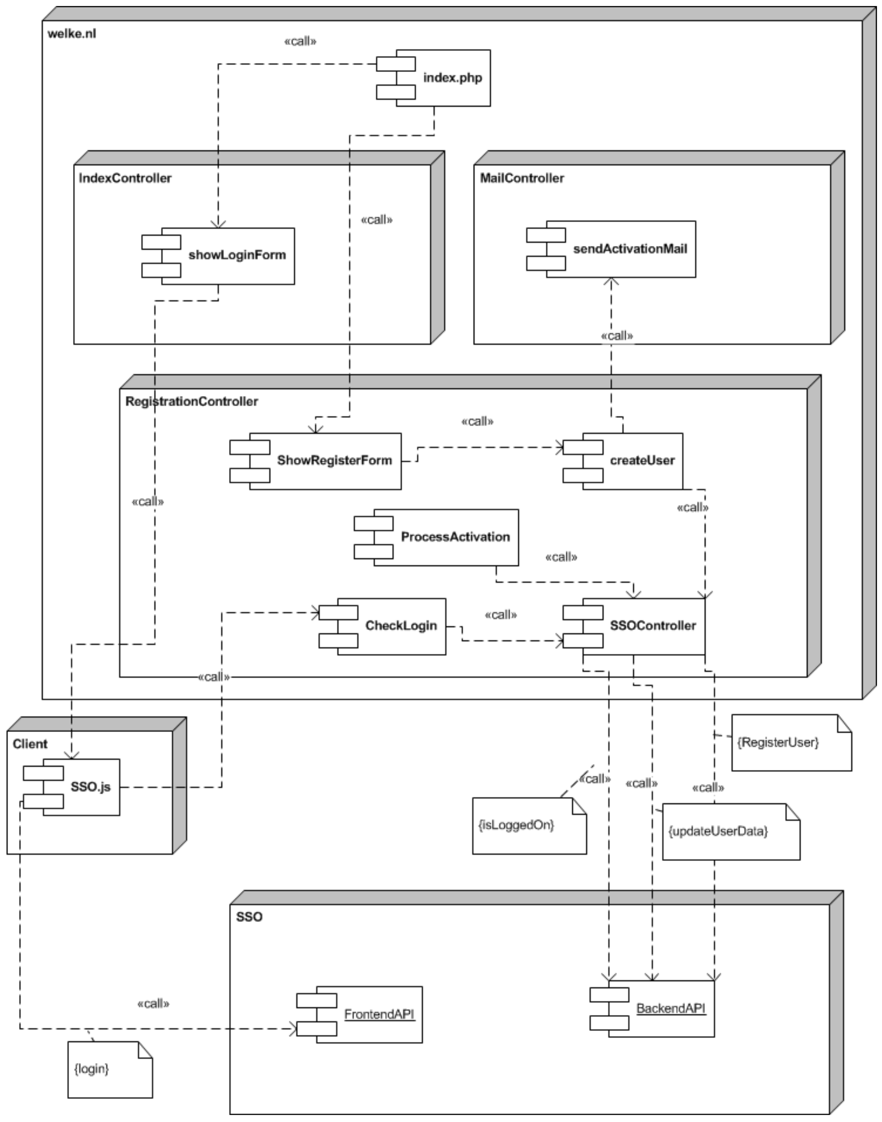

Figure 4.10: SSO welke implementation

As seen in Figure 4.10 the new layout of the welke.nl website structures the various actions of the website into various categories, named controllers. The most important controller inside the website for this project is the RegistrationController which handles all the authorization and profile related functionalities, in the new design this controller was developed with the new SSO system in mind, so for the refactoring only the calls inside the controller were altered from calling the private database of the website to callling the SSO Backend API.

4.5.2 Huisplan.com (work unit 4.1 / 4.2)

For huisplan.com the replacement had to be designed seperately for the existing website offering the same functionality as described above for welke.nl. Also as huisplan.com more actively uses the profile data storing capabilities of the SSO system, some more code had to be refactored to implement the storage of these profile data through the SSO backend API, so that instead of storing it in the local website database it is stored centrally.
Although special refactorings of the authentication handling in the website to retrieve the user role were needed, the javascript library used at the client side can be re-used completely and saves development and maintenance time.

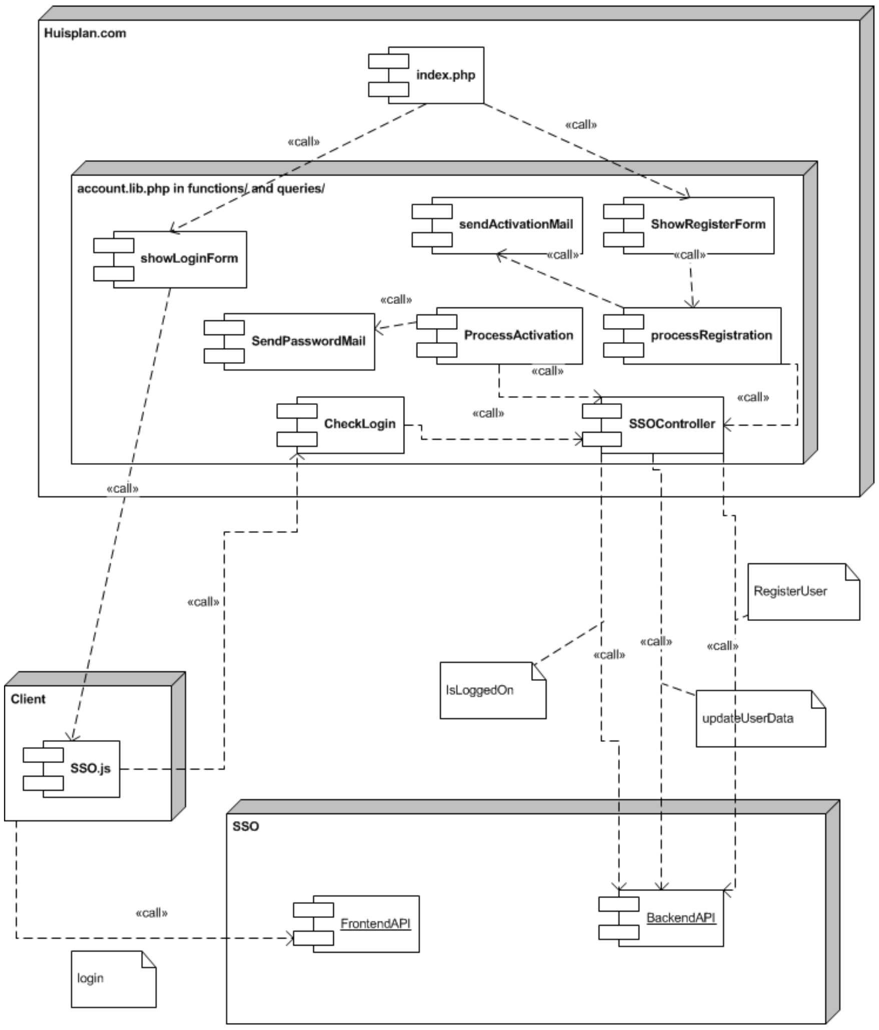

Figure 4.11: SSO huisplan implementation

Figure 4.11 shows where the website is connected to the SSO system, as to say where the current calls are diverted to SSO API calls which handle the functionality requested by the website or client through the SSO.js frontend API javascript library. As huisplan.com has spread all registration and authorization functionalities over different files in the website libraries, the refactoring comprised the altering the different files containing the function calls to the database access functions to using the SSO api call functions.

4.6 Actual migration

The actual migration covers the migration of the profile data of the existing websites to the new SSO system. The process covers the 3 steps defined in Figure 3.11. To ensure a succesfull migration the migration was be done incrementally and during the whole process all the users were involved.

4.6.1 Setting up the migration plan (work unit 5.1)

As the migration comprised of migrating at least 2 databases of user data that are not formatted in the same way and can include duplicate or out of data information, a migration plan had to be developed to tackle all the cases and ensure a smooth migration to the new SSO platform.

Beside the description of the situations the migration has to account for, a schedule of the incremental steps is created to inform the management of MediaMij what had to be done and when it would be done so they could assist in testing the new situation. This schedule is described below in Section 4.6.1.2

4.6.1.1 Merge situations

The uniqueness of the user was defined in its email address, so it the address appeared more then once in both databases it is noted as a duplicate. To simplify the further process MediaMij management decided that when an user existed in both welke.nl and huisplan.com databases, the user is migrated with the newest data of huisplan.com prevailing over that of welke.nl as the information of the user in huisplan.com database always is more detailed then that of the user in the welke.nl database. It is said most recent information of huisplan.com here because an user could subscribe more then once to huisplan.com with the same email address.

4.6.1.2 Migration schedule

The migration schedule was approved as follows:

1. Switch and test welke.nl to use the new SSO on test server
2. Copy basic profile information of welke.nl to SSO database
3. Test welke.nl with migrated data on test server
4. Copy extended profile information of welke.nl to SSO database

5. Test welke.nl with migrated data on test server
6. Milestone 1
7. Develop migration script to import the huisplan.com data with the above merge situations in mind
8. Switch and test huisplan.com to use the new SSO on test server
9. Copy basic profile information of huisplan.com to SSO database
10. Test huisplan.com with migrated data on test server
11. Copy extended profile information of huisplan.com to SSO database
12. Test huisplan.com with migrated data on test server
13. Milestone 2
14. Integration / performance tests
15. Milestone 3
16. Arrange migration day for production websites (with maintenance notice on the websites)
17. Migrate live data following the steps reaching Milestone 1 and Milestone 2 again but now on the production server
18. Release web applications
19. Milestone 4

4.6.2 Data migration: Welke.nl

When the development of the new cms and website for welke.nl started, the development team also setup a new development chain including an acceptance environment. This environment could then be used as a test environment for integrating the welke.nl with the SSO environment. Here tests were run to see how much delay it caused when an user logs in, logs out or registers at the website, eg. the actions for which the website and the SSO have to communicate.

The data migration has been made simple by releasing the new website and cms for welke.nl, with some simple SQL statements the database could be exported out of the database of the new website and imported into the SSO database. However the migration of the data from the old website to the new website had a lot more work to it, as the work for it was put into the scope of the project by the company, a migration schema was setup and can be found in Table 4.1. The migration schema has been used to develop a small software tool which function was to query the database, transform the data to the new schema and insert it into the new database.

Old database table	Old column	New database table	New column
User	active	Profile	active
User	emailadres	Profile	email
User	mailingadres	Profile_data	mailingadres
User	name	Profile	firstname
User	password	Profile	password
User	profile	-	-
User	userrole	Profile_role	-
Profile	city	Profile	city
Profile	extension	Profile	extension
Profile	housenr	Profile	housenumber
Profile	postalcode	Profile	zipcode
Profile	street	Profile	street
Profile	budget	Profile_data	row:budget
Profile	family_composition	Profile_data	row:family_composition
Profile	planned_buy_date	Profile_data	row:planned_buy_date
Profile	style	Profile_data	row:style

Table 4.1: Welke.nl migration schema

It is clearly visible that the last 4 rows of the table mention that they are not mapped to a column in the new table, but are copied as attribute/value rows to a separate table, profile_data. This is necessary due to the requirement of having almost unlimited profile attributes and leads to the usage of an Entity-Attribute-Value table Profile_data. To be noted as well is that the profile column in the old database table User was dropped as both the Profile and User table are merged in the new database into the table Profile.

4.6.3 Data migration: Huisplan.com

The current database of huisplan.com contained numerous users (30000+), migrating the database manually was not an option. Also as the welke.nl database was already migrated to the SSO environment, extra care had to be taken with overlapping data. While migrating a check was needed to see if the user being migrated was already in the new database or not. Here come in hand the decisions made during the setup of the migration plan. As expected a lot of profiles turned up which already existed in the welke.nl database but also because of the email uniqueness declaration, a lot of profiles turned up to be duplicated in the huisplan.com database alone already, so with the rules set in the migration plan a migration schema has been designed and visualized in Table 4.2.

Something to notice here is that the export_lms column is not migrated as this functionality is kept within the website itself, so the huisplan website still has to keep an table for its lead export to state which user is already exported or not. Another thing to notice is that there were 2 columns of the profile table in the old design that where not stored in the profile_data table, namely search_area and movedate, after the migration they are.

Old database table	Old column	New database table	New column
Profile	entryDate	Profile	entryDate
Profile	updateDate	Profile	updateDate
Profile	firstname	Profile	firstname
Profile	suffix	Profile	suffix
Profile	lastname	Profile	lastname
Profile	sex	Profile	sex
Profile	zipcode	Profile	zipcode
Profile	housenumber	Profile	housenumber
Profile	extension	Profile	extension
Profile	street	Profile	street
Profile	city	Profile	city
Profile	email	Profile	email
Profile	password	Profile	password
Profile	phone	Profile	phone
Profile	active	Profile	active
Profile	verified	Profile	verified
Profile	export_lms	-	-
Profile	search_area	Profile_data	row:search_area
Profile	movedate	Profile_data	row:movedate
Profile	*	Profile_data	*
Profile	*	Profile_profiling_datar	*

Table 4.2: huisplan.com migration schema

For the migration process a php script was created to be able to check on uniqueness and so to handle overlapping users. The data of overlapping users found in the huisplan.com simply was used to overwrite that of the user already existing due to the migration of the welke.nl users, whereas new users were just plainly inserted.

4.7 Validation of requirements

With the new SSO system, customers of MediaMij visiting the welke.nl and huisplan.com websites now have one set of credentials for log in. More importantly they only have to register once at one of both websites and they are also greeted by the other. However this was not the main motivation for MediaMij wanting the migration. The main reason was the central customer database they now can use for marketing operations across all the websites instead having to limit to the userbase of one website only.

Also the to be developed marketing tool now can gather its information at all the websites of MediaMij to deduct even more detailed customer behaviour for its analysis, so MediaMij is able to exactly know what the customer wants to see at the websites of MediaMij.

To determine if the project is a success we check all the requirements found in Section

1.2. As seen in the sections below the resulting products meet all the requirements and so fulfill the expectations of MediaMij.

4.7.1 Functional requirement 1: Central customer database

In the migration we have set the email address as parameter for uniqueness, so all customers migrated to the new system are recognizable by their unique email address, satisfying requirement 1a. Requirement 1b is satisfied by inclusion of the table profile_data allowing for storage of all user entered information by the customer at one of the web application of MediaMij. With the tables profile_profiling_data, profile_behavioural_data and profile_concluded_data extra storage is provided for analyzed data of a customer and as multiple rows can be used in the table for a single customer there is ample room for storing the data as per requirement 1c.

4.7.2 Functional requirement 2: Customers need to login once

As a customer authenticates with the SSO system itself, the customer has a session with the SSO system identifying the customer and their userrole, so if the customer accesses a different web application of MediaMij, the user can use the same session at the SSO for authentication.

4.7.3 Functional requirement 3: External API of the SSO system

Appendix B describes the API of the SSO system that is developed. In the API there are various remote methods available for web applications of MediaMij requesting information about a customer. The IsLoggedOn method offers the web application to check the validity of a customer session and retrieve the userrole so they can check permission for the operation requested by the customer, satisfying requirement 3a and 3b.

The methods of the User administration and Global administration sections in the API offer various data retrieval and manipulation methods for web applications to gather data to be analyzed about a customer and submit it back again through the API as per requirements 3c and 3d.

4.7.4 Nonfunctional requirement 1: Slowdown less than 500 ms

As to measure if the slowdown is not more then the required 500 ms a small expirement was made. In cooperation with MediaMij the most used setup of computer was determined and used as setup for the experiment. The majority of the MediaMij customers use Internet Exporer as browser running on the Microsoft Windows Operating System. So the computer used in the test ran on Windows Vista with Internet Explorer 7.0 installed. We decided to use high grade hardware for the expirement as to not let the slowdown depend on it, so we used an Quad Core Intel processor with 4 GB of internal memory.

To get a realistic measurement the 2 most used scenarios are measured in the experiment. At first we measure the Registration scenario as here the registration data is send to

Action	**Try**	**Old Situation**	**New situation**
Registration	1	511ms	535ms
Registration	2	467ms	551ms
Registration	3	501ms	498ms
Registration	4	546ms	680ms
Registration	5	560ms	589ms
Average		517ms	571ms

Table 4.3: Measurements of slowdown while executing registration scenario

Action	**Try**	**Old Situation**	**New situation**
Login	1	211ms	301ms
Login	2	314ms	365ms
Login	3	345ms	289ms
Login	4	289ms	345ms
Login	5	189ms	225ms
Average		270ms	305ms

Table 4.4: Measurements of slowdown while executing login scenario

the SSO by means of the backend API and therefore a slowdown might occur for the user waiting for the SSO to complete the API call.

Secondly the login procedure is measured as the credentials are send to the SSO by using the frontend API and on success the session is send to the web application which in turn verifies the session with the SSO by using the backend API. So as here the user has to wait for completion of an API command twice, we expect a slowdown of the user experience in this scenario.

To measure the slowdown we cannot use the tool FireBug used in the reverse engineering phase as it is a plugin for the Firefox webbrowser only, whereas we are using Internet Explorer for the measurements. So for the measurements we have used Wireshark[4] to calculate the time needed for the web browser to do the requests while executing the scenario.

First we measured the time needed to execute the scenario in the old situation 5 times. Then we switched to using the new SSO system and then we executed the scenario 5 times again. Finally we calculated the averages using the measurements of the expirement for the old and the new situation. As depicted in Table 4.3 the slowdown for the registration scenario is only 54 ms and Table 4.4 shows us an average slowdown of only 35 ms. So we can safely conclude that the slowdown is kept within the set maximum of 500ms.

4.7.5 Nonfunctional requirement 2: Securing communication

A very trivial solution has been created to satisfy the security requirement. As the web applications owned by MediaMij are all running on servers owned and maintained by Me-

[4]See: http://www.wireshark.org

diaMij, a seperate private network has been setup connecting all the web applications to the SSO system, making the SSO backend API reachable only on the private network and so unreachable for devices outside this network. As now all communication between the web applications and SSO system are private, the requirement is satisfied.

For situations where the SSO system is only reachable by connecting through the open internet, this solutions is not usable out of the box. In that situation there are 2 options, one is to raise the encryption used on the SSL / HTTPS connection between the web application and the SSO, another more elegant solution is to create a VPN connection connecting the systems to a virtual secure network.

Chapter 5

Lessons learned

As seen troughout the project, website re-engineering is a challenging subject. During all phases of the projects difficulties were encountered and problems circumvented or solved. Following the proposed structure in the approach (Chapter 3) we describe how to tackle the situation and what to be carefull for during the phases. Also tools and or techniques that help to solve the problem are mentioned. Secondly, we indicate situations where the approach did not went so well and we recommend a different approach.

Everything learned from this project can be formed into guidelines for doing projects that are alike. So as a final section we describe those guidelines so developers with similar projects can use these as a basis for their project.

5.1 Website reverse engineering

The difficulty with websites arise with the separation between code that is running at the webserver and code that is running inside the browser used by the user visiting the website. The first part of the software, running on the webserver, can be developed in one of the many programming languages available. As there are too many we only discuss the languages encountered during the project and of those languages, as there exist a couple of programming strategies as seen in the case study, possible approaches to the encountered programming patterns are discussed.

As seen during the project, also the code running inside the client can effect the whole software process, as (user generated) events may forward the user to a new page in the website or dynamically update the content of the current page. If these technologies are used in the website the reverse engineering may need a different approach, because also the code running in at the client web browser needs to be reverse engineered. For this case we have seen in Section 2.1.3 there are tools available to investigate the execution trace, namely Firebug[1] and Firedetective[2].

[1]See: http://getfirebug.com

[2]See: http://swerl.tudelft.nl/bin/view/Main/FireDetective

5.1.1 PHP websites

The website of huisplan.com and the new version of welke.nl both are written in the php programming language with however a big difference. Where huisplan.com is written with the traditional functional programming strategy that has been the standard for php websites for a time, the new welke.nl however is written completely with OO programming standards in mind and even follows the MVC pattern. As huisplan.com lacked a lot of strucuture in the code it was far more harder to get a grip on that codebase and so the reverse engineering depended more on the results of the demos and interview then on the code reading session.

With the codebase of huisplan.com we had to rely on the information gathered from the meetings with the developers, luckly they had enough knowledge of the system to point out entry points for the code reading session. However due to lack of structure in the code the reading session took an extra hour to get enough information out of it. Therefore we recommended to use one of the tools mentioned in Section 2.1.1, e.g. WARE, to do the static analysis to get a better understanding of the code and entry points before reading the code yourself.

5.1.2 Java websites

Also in JAVA web applications can follow different design patterns, however JAVA is an OO language by origin. Many web applications built in JAVA therefore use a MVC pattern as basis which makes the understanding easier. The old version of welke.nl indeed used the MVC pattern and so the code reading session delivered some good results already.

When a web application is not following the MVC pattern using a tool is recommended to get the same level of understanding. For this process one of the tools mentioned in Section 2.1.1 is sufficient again.

5.1.3 Client code

Like for the code running at the webserver, a lot of solutions are available to run code inside the browser of the user. This code can dynamically alter the content of the page or for example forward the user to a new or previous page. As this effects the web application process, it makes reverse engineering the website a lot more difficult as there are 2 codebases that have to be investigated at the same time. As this was not possible at the time the case studies were done, in the project we used Firebug to investigate the code executed in the browser of the user. Then we tied the results of these investigations to the static analysis so we were able to include the client code interactions in the diagrams of the reverse engineered web application.

With the availability of Firedetective however this can be automized more which as Firedetective is able to link the traces of code running at the browser of the client to code running at the web application. As this removes the need of the manual linking, this is the recommended approach.

5.2 SSO Design

As visible in Table 2.1, SSO can be implemented at various levels, with various reasons explainging the choice for the level of SSO. Therefore one has to determine the exact reason behind the decision, as to discover what SSO level is required for their situation. When one knows why a SSO system is nescessary, the requirements need to be discovered, and most importantly what influences the decision the most is what kind of data needs to be stored centrally and if it really needs to be stored centrally with each update of the information. Crucial for this decision is the requirment of the company or organization owning the system. When the requirement is a fully central customer profile database the decision is easy as was the case with this project. If there is no requirement or at least not the one of a fully central customer profile database, it is recommended to know the relation between the profile data residing in the current web applications. If the data is not so related it is better to opt for only having a common set of credentials extended to one account session if users frequently use the different web applications at the same time.

Secondly, the data structure representing the data to be stored centrally is as crucial as the quantity of the data. Storing a flat profile (key, value) is supported by some systems, where storing a profile with a deeper data structure, like a table with many rows per user, won't be supported. Finally it needs to be clear if the data is privacy sensitive or not, let alone the credentials off course, which are to be kept secret at all times. As there are a laws about the privacy of customer profile data, storing of this information should be done safely and according to the law. For this project we had to adhere to the rules set by the organization[3] guarding the privacy of consumer data.

5.2.1 Custom SSO Design

As some requirements for the SSO in this project were not satisfied by any of the researched existing systems in Chapter 2, we had to develop a custom SSO system. Developing the custom SSO resulted in some challenges that had to be dealt with during the design of the system.

The challenge with building a custom SSO is securing the fact that both at the web application and at the central SSO application, the user authenticating is a valid user and that he or she is assinged the right authentication level. This means that both applications need to agree on the identity of the user and share an identifaction key for this user used for verification of the user at both ends. The web application can then if the user is verified, query the SSO system for the authentication level of the user.

As the web application and the SSO application do not run at the same server and inherently domain, at the client side a challenge arisse for the code submitting the credentials to the SSO system. The challenge is in the fact that the code runs on the domain of the web application, but in this situation the cross domain javascript security policy is enforced and so prohibiting the execution of code based on the result of the call to the SSO system. However as demonstrated there is a solution with JSONP, which allows the call to return code that can be interpreted and in turn is able to act based on the result of the authentication.

[3]See: http://www.dutchdpa.nl/

5.3 Migration

Migrating several databases with profile information to a single database is a tedious task. At first the uniqueness of a profile is defined as profiles can exist in all the databases seperately. Profiles also can exist more then once in a single database already if profiles were not unique or if they are defined to be unique in an other way in the old database then for the new database. In this project we have seen the latter as we set the email adress unique in the new database, while the old huisplan.com database had no uniqueness defined on the email address, but also profiles had to be merged because they existed in both databases.

It is recommended to make the username unique, as otherwise it will be very difficult determining which user is authenticating. As the email adress of an user is one that is known most of the times and the email address is personal this is the recommended way. It is best practice however to notify the users of the merging of their accounts if they existed in more then one database or more then once in one database and to use the information the user entered most recently.

Another challenge arose with the mergal of the seperate columns. Those have to be aligned to the structure of the new database. Here we saw that sometimes the columns could not be copied to columns in the new database but had to be inserted as rows with a relation to the migrated profile. This solution is the best case for this project because migrating them as columns would result in tables with too many rows. It is recommended however to migrate columns to columns and not to key/value table as then the data can be queried with a single statement instead of having to loop over the key/value rows. Only if it is expected that there will be a huge amount of columns sparsely filled for each row, the key/value table should be preferred.

5.4 MSO to SSO Guideline

Keeping in mind everything we learned throughout the project, we define a guideline for projects that are alike, projects that also are about migrating web applications that use their own databases for sign on and storage of profile data, to a system where they use one central database for profile data and sign on. This guideline is a list of steps to undertake during the project to understand the current web applications, describe them in a set of diagrams to get a detailed understanding about how they handle the profile data and authorization and also takes the project through the development and migration phase. The list of steps the guideline consists of is vualized in Figure 5.1.

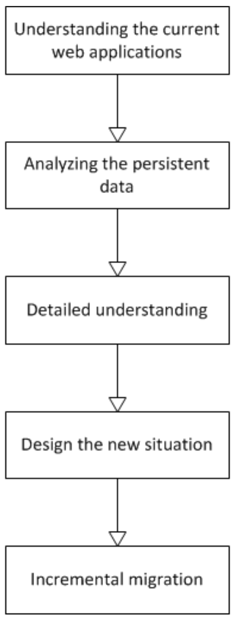
Understanding the current web applications
Analyzing the persistent data
Detailed understanding
Design the new situation
Incremental migration

Figure 5.1: Steps of the guideline

5.4.1 Understanding the current web application

To understand the current web applications first one has to understand why the applications exist and what is the philosophy behind them, otherwhise it is very difficult to understand why the applications need to be moved to a SSO in the first place.

The first step would be meeting up with the current developers and let them tell about the applications and let them see what users can do with them. Let them also describe why users should create an account at the application to get an understanding about what is stored persistently.

Secondly, it is usefull to know the history of the application as it can give insight into the evolutionary process of the application. This can tell us if there allways has been a registration for visitors and if the registration of storage of user data was altered at one point.

5.4.2 Analysis of the persistent data

After obtaining a basic understanding of the web application are and what their history is, the next step is to determine what kind of information is stored exactly. For the purpose of the project it is enough to know what kind of information is stored about the website users. However we need to discover if there are any relations between the user data and the rest of the database as these relations may also be required to be stored centrally in the new situation.

As a result of this step a relational database diagram of one kind should be created that is used as documentation throughout the project to fallback on. Also it can be used to verify with the development team to see if something is missing in the diagram.

5.4.3 Detailed understanding

For a complete understanding of the current application we need to know what the various scenarios for user registration, user authorization, user modification are and how they are implemented in the application. With this information we then can draw an implementation diagram detailing the implementation of the mentioned scenarios.

We use as input the basic understanding we got during the meeting with the development team and the analyis of the persitent data and then we use the read the code in one hour pattern of the book [11], to see how the database interaction is done and how this interacts with the rest of the application. The code reading should be done with a time limit as a full understanding by just reading the code is not possible. The results of the session is verified with a developer by doing an interview with the developer while following the scenarios found allready in the application, like described in the interview during demo patter in the book [11].

This interview results in a final list of scenarios that we can use to draw up a use case diagram detailing those situations. This use case diagram is important as we use it as basis to discover the detailed implementation of the scenarios.

To finalize the detailed understanding further there are 2 options, namely using an external tool to analyze the code and draw up an implementation diagram or stepping through the execution of the code manually to get the implementation of the scenarios and then manually draw up the implementation diagram. It is rarely possible to use a tool as the tool tries to investigate the whole codebase while we are only interested in the mentioned scenarios. On the other hand doing the analysis by manually stepping through the code costs a lot more time, however the advantage here is we can limit it to the mentioned scenarios only. For Ajax based website a tool like Firebug or Firedetective of [28] is a must to follow through the execution of the ajax code.

When the implementation diagram is finalized it is good practice to discuss this list with the developers again to make sure it is indeed final and nothing is missing.

5.4.4 Desiging the new situation

Now it is key to determine the exact requirements for the new SSO system, what do we need to store centrally and why. It saves a lot of develoment time if the requirements can be fit in the scope of one of the existing SSO platforms already built. This means that the project only has to develop modules for the existing applications to use the SSO, so no design has to be done of the SSO itself.

If the requirements are too hard to fit into an existing system, like with this project, a design of the SSO system has to be done. A custom system also has its advantages as it can be more flexible and does for sure implement all the requirements. Important here is to design a strict API detailing the services offered by the SSO and to focus on security as with web applications the API needs to be open on the wide internet for the users of the applications.

In both situations the existing web application needs to be refactored to call the SSO API instead of querying the local database, where at the client side the submission of both the login and registration forms have to be redirected to the SSO system for authentication and based on the result a call has to be made to the extended web application to setup the 3 way authentication needed for SSO.

5.4.5 Incremental migration

The final step of the project is the migration from the old to the new system. As we want the project to be a success, everyone should be enthousiastic about the results and wanting to put the SSO into production. Involvement of the users is important and this is done by using a migration plan which is first approved before executed. In the plan the steps needed to migrate the systems are setup together with milestones at which reverting to the old situation is possible if something appears to be working incorrectly. This ensures the original developers feel they are involved in the incremental migration process.

A last thing to note about the migration plan is that it needs to include a detailed mapping of the data migration of all the seperate databases to the central SSO database. This can be tedious as not only data has to be merged vertically, which is the case for duplicate profiles, but also a horizontal merge has to be done to align the data of the old databases to the structure of the new database, this because of columns not existing anymore in the new database and vica versa.

Chapter 6

Conclusions and Future Work

MediaMij BV is a media company serving information about all kinds of redecorating options for houses. It distributes a couple of magazines about those redecorations and those magazines are related to the websites owned by MediaMij. All these websites are bundled in the general welke.nl website, aside from the welke.nl websites MediaMij also owns a website about the process customers go through while searching and buying a house, huisplan.com.

All of these websites are storing information about the visitors helping their search for information by building up a profile about the user. These profiles are stored at the seperate databases running for these websites. The desire of MediaMij was to centrally store the profile information, enabling a single customer database which is usefull for the intended markerting strategies of MediaMij. The second advantage of the desired situation is the ability for the website users to single sign on at one website and use his or her profile at all websites owned by MediaMij.

We set up an approach to this problem in Chapter 3, splitting up the process into the different steps, describing the inputs for each step and the outcome. In the case study, Chapter 4, the approach is applied to 2 of the websites of MediaMij, welke.nl and huisplan.com. Here we described how the approach was applied to the 2 websites and what the approach produced as outcome.

During the project we saw what the implications were of the approach, what went very well and what went not so well. This we described in the lessons learned and with that information the guidelines were developed visible in Chapter 5.

6.1 Contributions

During the project a lot of patterns common with software reverse enigineering were applied to the project. These patterns not allways could be used directly as web applications differ a lot from normal applications, mostly because of the way the users are using the software. As users are not using directly the software but with are viewing the content in a browser application.

The patterns were therefore intepreted more freely and we picked the best out of them to combine with the best of others. A good combination we found was that using interviews and involving the development team with actually reading the codebase or stepping through it, as it delivered good results with the web applications in this project, described in Section 3.2.2 and applied in Section 4.3.1.2 and Section 4.3.2.2.

The guideline described in Section 5.4 is the main contribution of this project. It describes how to approach and execute projects that are like this project, refactoring multiple web application with their own databases to a new situation where the web applications use a central user database with SSO capabilities. The guideline details on how to approach the current situation, how to design the new situation with the central user database and how to migrate from the old situation to the new situation successfully.

6.2 Future work

At the beginning of the project a desision already was made to not include all the websites of MediaMij into the first release of the SSO system, as that would require more time then there was calculated for the project. During the project some parts also were declared to be out of scope as they required too much time, that simply was not available.

6.2.1 SSO control panel

To finalize the migration of all systems of MediaMij to the SSO system there are some tasks left to do where the most important would be to create an administration application to be used by MediaMij to be able to do advanced administrative tasks with the customer database for the intended marketing operations of MediaMij. Also to be included in this application would be the option to extend the API to other systems of MediaMij to offer access for the LMS system and the to be developed marketing tools.

Requirements for this control panel include adding new domains to the SSO database as to enable other websites of MediaMij using the SSO for authentication and registration, adding profile data definitions and a detailed overview with statistics about the users.

6.2.2 Coworker SSO

With the SSO all customers have one login to the websites of MediaMij, so they only have to remember one set of credentials. However the coworkers of MediaMij use the websites as well, be it at the backend, or control panel, where they have a seperate set of credentials for each website to manage the content of the website. So in a future project those accounts have to be integrated in the SSO as well, yielding another layer of complexity there as the accounts of a coworker is totally different from a customer. Luckely multiple levels of authentication are already supported by the userrole implementation, so that does not impact the project duration.

Bibliography

[1] G. Antoniol, G. Canforca, G. Casazza, and A. De Lucia. Web Site Reengineering using RMM. *2nd Int. Workshop on Web Site Evolution*, pages 9–16, 2000.

[2] G. Antoniol, M. di Penta, and M. Zazzara. Understanding web applications through dynamic analysis. *Intl Workshop on Program Comprehension*, pages 120–129, 2004.

[3] A. Armando, R. Carbone, L. Compagna, J. Cuellar, G. Pellegrino, and A. Sorniotti. From Multiple Credentials to Browser-based Single Sign-On: Are We More Secure? 2011. to appear.

[4] A. Armando, R. Carbone, L. Compagna, J. Cuellar, G. Pellegrino, and L. Tobarra. Formal Analysis of SAML 2.0 Web Browser Single Sign-On: Breaking the SAML-based Single Sign-On for Google Apps. 2008.

[5] C. Bellettini, A. Marchetto, and A. Trentini. WebUml: Reverse Engineering of Web Applications. *SAC'04*, 2004.

[6] J. Conallen. *Building Web Applications with UML.* Addison Wesley, 1999. (ISBN 0-201-61577-0).

[7] UWA Project Consortium. Ubiquitous Web Applications. *Proc. of the eBusiness and eWork Conference*, 2002.

[8] M. Cornelissen, L. Moonen, and A. Zaidman. As assessment methodology for trace reduction techniques. *Intl Conf. on Software Maintenance (ICSM)*, pages 107–116, 2008.

[9] Jan De Clercq. Single sign-on architectures. In George Davida, Yair Frankel, and Owen Rees, editors, *Infrastructure Security*, volume 2437 of *Lecture Notes in Computer Science*, pages 40–58. Springer Berlin / Heidelberg, 2002.

[10] R.P. Del Castillo, I. García-Rodríguez, and I. Caballero. PRECISO: A Reengineering Process and a Tool for Database Modernisation through Web Services. *SAC'09*, 2009.

[11] S. Demeyer, S. Ducasse, and O. Niestrasz. *Object-Oriented Reengineering Patterns*. Square Bracket Associates, Switzerland, 2008. (ISBN 978-3-9523341-2-6).

[12] C. Di Francescomarino, A. Marchetto, and P. Tonella. Reverse Engineering of Business Processes exposed as Web Applications. *European Conference on Software Maintenance and Reengineering*, 2009.

[13] G.A. Di Lucca, M. Di Penta, G. Antoniol, and G. Casazza. An Approach for Reverse Engineering of Web-Based Applications. *WCRE'01*, pages 231–240, 2001. Proc. 8th Working Conference on Reverse Engineering.

[14] G.A. Di Lucca, D. Distante, and M.L. Bernardi. Recovering Conceptual Models from Web Applications. *Proceedings of the 24th Annual Conference on Design of communication (SIGDOC06)*, pages 113–120, 2006.

[15] G.A. Di Lucca, A.R. Fasolino, U. De Carlini, F. Pace, and P. Tramontana. Comprehending Web applications by a clustering based approach. *Proc. 10th Workshop on Program Comprehension*, pages 261–270, 2002.

[16] G.A. Di Lucca, A.R. Fasolino, F. Pace, P. Tramontana, and De Carlini U. WARE: A Tool for The Reverse Engineering of Web Applications. *Proc. 6th European Conference on Software Maintenance and Reengineering (CSMR'02)*, pages 241–250, 2002.

[17] G.A. Di Lucca, A.R. Fasolino, and P. Tramontana. Reverse Engineering Web Applications: The WARE Approach. *Journal of Software Maintenance and Evolutions, Research and Practice*, 16:71–101.

[18] G.A. Di Lucca, A.R. Fasolino, and P. Tramontana. Towards a Better Comprehensibility of Web Applications: Lessons Learned from Reverse Engineering Experiments. *Proc. 4th Int Workshop on Web Site Evolution (WSE'02)*, pages 33–42, 2002.

[19] D. Draheim, E. Fehr, and G. Weber. JSPick - A Server Pages Design Discovery. *Proc. 7th IEEE European Conference on Software Maintenance and Reengineering, LNCS*, pages 230–236, 2003.

[20] D. Draheim, E. Fehr, and G. Weber. A Source Code Independent Reverse Engineerint Tool for Dynamic Web Sites. *Proc. 9th IEEE European Conference on Software Maintenance and Reengineering, LNCS*, pages 168–177, 2005.

[21] J. J. Garrett. Ajax: A new approach to web applications, 2005. http://www.adaptivepath.com/ideas/essays/archives/000385.php.

[22] K. Haller. Towards the Industrialization of Data Migration: Concepts and Patterns for Standard Software Implementation Projects. *CAiSE 2009*, 2009.

[23] M. Hillenbrand, J. Götze, J. Müller, and P. Müller. A Single Sign-On Framework for Web Services-based Distributed Applications. *8th International Conference on Telecommunications (ConTEL)*, 2005.

[24] T. Isakowitz, A. Kamis, and M. Koufaris. Extending the Capabilities of RMM: Russian dolls and Hypertext. *Proc. 30th Hawaii Int. Conf. on System Science*, pages 166–186, 1997.

[25] T. Isakowitz, E. Stohr, and P. Balasubramanian. RMM: A Methodology for the Design of Structured Hypermedia Applications. *Communications of the ACM*, 38(8):34–44, 1995.

[26] J. Jeong, Dongkyoo Shin, and Dongil Shin. An XML-based single sign-on scheme supporting mobile and home network service environments. 50(4):1081–1086, 2004.

[27] P. Li and E. Wohlstadter. Script InSight: Using models to explore JavaScript code from the browser inview. *Intl Conf. Web Engineering (ICWE)*, pages 260–274, 2009.

[28] N. Matthijssen. Understanding Ajax Applicationsby using Trace Analysis. Master's thesis, University of Technology Delft, the Netherlands, 2010.

[29] J. Morris. *Practical Data Migration.* British Computer Society, Swinton, UK, UK, 2006.

[30] S. Oney and B. Myers. FireCrystal: Understanding interactive behaviors in dynamic web pages. *Proc. of the Symposium on Visual Languages and Human-Centric Computing (VLHCC)*, pages 105–108, 2009.

[31] Pashalidis, A. and Mitchell, C.J. Single sign-on using trusted platforms. *Information Security and Privacy, 8th Australasian Conference, ACISP 2003*, pages 54–68, 2003.

[32] Pashalidis, A. and Mitchell, C.J. A taxonomy of single sign-on systems. *Information Security and Privacy, 8th Australasian Conference, ACISP 2003*, pages 249–264, 2003.

[33] R. Patel, F. Coenen, R. Martin, and L. Archer. Reverse Engineering of Web Applications: A Technical Review. 2007.

[34] R.S. Pressman. What a tangled Web We Weave. *IEEE Software*, 17(1):18–21, 2000.

[35] J. Pu, R. Millham, and H. Yang. Acquiring Domain Knowledge in Reverse Engineering Legacy Code Into UML. *Proc. Software Engineering and Applications*, 2003.

[36] F. Ricca and P. Tonella. Web Site Analysis: Structure and Evolution. *Proc. 16th IEEE Int. Conf. on Software Maintenance*, pages 76–86, 2000.

[37] F. Ricca and P. Tonella. Understanding and Restructuring Web Sites with ReWeb. *IEEE Software*, 2001.

[38] A. Rüping. From Old To New, Patters for Data Migration. *Proc. of EuroPLoP 2010*, 2010.

[39] B. Schneiderman. The eyes have it: A task by data type taxonomy for information visualizations. *Proc. Symposium on Visual Languages (VL).*

[40] H.M. Sneed. *Migrating to Web Services, in Emerging Methods, Technologies and Process Management*. Wiley-IEEE Computer Society Pr. pages 151-176.

[41] P. Tramontana. Reverse Engineering Web Applications. *Proc. 21st IEEE Int. Conf. on Software Maintenance*, 2005.

[42] M. Turner, D. Budgen, and P. Brereton. Turning Software into a Service. pages 38–44, 2003.

[43] M. Wagner and T. Wellhausen. Patterns for Data Migration Projects. *Proc. of EuroPLoP 2010*, 2010.

[44] B. Wu, D. Lawless, J. Bisbal, R. Richardson, J. Grimson, V. Wade, and D. O'Sullivan. The Butterfly Methodology: A Gateway-free Approach for Migrating Legacy Information Systems. *Proc. of the 3rd IEEE Conference on Engineering of Complex Computer Systems (ICECCS97)*, pages 200–205, 1997.

Appendix A

Glossary

In this appendix we give an overview of frequently used terms and abbreviations.

AJAX: Asynchronous javascript and xml

API: Application programming interface

browser: The application the user is using to view the website

LMS: Leads management system

MVC: Model View Controller software pattern

SSO: Single sign-on

WBS: Work breakdown structure

Appendix B

API definition

In this appendix, the SSO API is described in detail. The first section describes the backend API, while the second section describes the frontend API.

B.1 Backend API

B.1.1 Registration & Authentication

isLoggedOn

userid	int	The userid of the user
ssoid	string	The SSO sessionID of the user
domain	string	The domain of the website the user is currently visiting

1. Lookup the user online status by user ID and SSO sessionID on the given domain
2a. Return the user role if the user is online
2b. Return the anonymous user role otherwhise

userExists

email	string	The email adress to check existence on

1. Check if email adress is already in use by an user in the database
2. Return result

RegisterUser

firstname	string	Firstname
suffix	string	Lastname prefix (eg van der)
lastname	string	Lastname
gender	string	Gender
zipcode	string	Zipcode
housenumber	string	Housenumber
extension	string	Housenumber extension
street	string	Street address
city	string	City
email	string	Email address
password	string	Password
domain	string	The domain the user registered on

1. Checks if user does exist already with email address
2. If not create new user and subscribe user to domain
3. Return generated user ID

subscribeUser

userID	string	The userID
domain	string	The website domain

1. Check if user is already subscribed to the website domain
2. If not, add subscription
3. Return result

B.1.2 User administration

GetUserData

userID	string	The userID

1. Check if user exists
2. Return user data or failure if user does not exist

UpdateUserData

userID	string	The userID
data	hashmap(string,string)	The new user data

1. Check if user exists
2. If user exists, update supplied data
3. Return update success

AddProfilingData

email	string	The email adress to check existence on

1. Check if email adress is already in use by an user in the database
2. Return result

B.1.3 Global administration

GetUsersWithProfileData

filter	string	Profile data filter
domain	string	Website domain

1. Queries the database with the filter and website domain supplied
2. Return all users satisfying the criteriay (could be an empty list

GetUsersForDomain

domain	string	Website domain

1. Return all users subscribed to the website domain requested

GetProfilingDataDefinitions

1. Return all existing profiling data definitions

AddProfilingDataDefinition

name	string	The name of the profiling data element
type	string	The type of the profiling data element

1. Check if data elemen exists
2. If user exists, create profiling data element

B.2 frontend API

This is a JSONP HTTP API offering 2 remote calls:

login

username	string	The username of the user to authenticate
password	string	The password of the user
callback	string	The javascript wrapper function name

1. Tries to authenticate user with supplied credentials
2. If succesfull then save user online state
3. Send back the login result as json data

wrapped by a function call using the function name supplied in the callback parameter

logout

1. Checks if user has an active session
2. If so, destroy session and remove online state