Integration of Visma Administration and SiteVision

Integration av Visma Administration och SiteVision

Jonathan Fredriksson Lind

Faculty of Health, Science and Technology

Computer Science

Bachelor's Degree, 15HP

Supervisor: Leonardo Martucci

Examiner: Donald F. Ross

2015-06-01

Integration of Visma Administration and SiteVision

JONATHAN LIND
Department of Mathematics and Computer Science

Abstract

The ability to reach a wider public using the Internet has brought the world a wide variety of online retailers offering their customers to purchase almost anything from the comfort of their home. With this increase in online retailing, a need for automation has been developed by the users. To be able to decrease the administrative tasks to a bare minimum allows retailers to focus on their core business instead. With this new request for automation, business systems are becoming more and more popular and to be able to integrate the web site with a business system results in lower administrative workload. A variety of systems are already available for the public to use for integrating a web shop to a number of different business systems. These systems are however often restrictive and will provide a framework for developing the web shop themselves. One web content management system that lacks this kind of support is SiteVision. On behalf of Soleil IT, a Proof of Concept was developed to investigate the feasibility of integrating a web shop developed in SiteVision, with a business system for automation of administrative tasks. The implementation included features such as retrieving products from the business system and displaying them on the web site, and the ability to purchase a set of products which should generate an order in the business system. RESTful web services was used to transmit data between the business system and the web shop. This project resulted in a successful integration of the business system Visma Administration and a web shop developed in SiteVision. The project showed the feasibility of performing such an integration and also exposed any restrictions that the setup may have.

Keywords: Business system, Content management system, Visma Administration, Sitevision, RESTful services, System integration, Web shop

This thesis is submitted in partial fulfillment of the requirements for the Bachelor's degree in Computer Science. All material in this thesis which is not my own work has been identified and no material is included for which a degree has previously been conferred.

_____

Jonathan Lind

Approved,

_____

Supervisor: Leonardo Martucci

_____

Examiner: Donald F. Ross

Acknowledgements

I would like to thank my supervisor at Karlstad University, Leonardo Martucci, for his support on writing this thesis. I would also like to thank my supervisor at Soleil IT, Johan Andersson, for all the technical guidance he provided during the development of this Proof of Concept. I also want to express my gratitude to Gray Gatehouse, who helped me review the thesis and providing valuable feedback. Finally, I would like to thank Soleil IT for giving me the opportunity to do this thesis project.

Karlstad, June 4, 2015 Jonathan Lind

Contents

1 Introduction

Soleil IT [17], a consulting company in Karlstad with a main focus on web development using SiteVision [16], a content management system (see section 3.2), were interested in evaluating the capability of integrating SiteVision with a business system (see section 3.1) for customers who wish to automate parts of their administrative tasks. To gain experience they first wanted this evaluation to be performed prior to any complete implementations. For this purpose, a Proof of Concept (POC) was to be developed and investigated to provide Soleil IT with a report on the matter. The POC for integrating Visma Administration with a web shop developed in SiteVision was developed and evaluated.

1.1 Goal

The goal of this project is to produce a POC for investigating the possibilities of integrating SiteVision with a business system to provide automation of administrative tasks for prospective customers. There are no documented integrations performed with SiteVision and a business system which made both the product company SiteVision, and Soleil IT, interested in researching the subject. The POC was to show the possibilities of such an integration by presenting a web shop that could display products managed in the business system as well as being able to complete purchases of one or more products where an order would automatically be created.

1.2 Contribution

This project showed the possibilities of performing an integration between a business system and a web site developed in SiteVision and led to continued investigation and research on the possibilities to extend the integration towards more than just this specific business system and further application to suit a broader customer base than just merchandising.

1.3 Scope

The project revolved around the development of the POC, meaning that it was not intended to be used as a finished product to be delivered to any customers. The purpose was to provide evidence of the integration being feasible when using SiteVision to produce a web site. The implementation did not include a security evaluation nor did it verify personal information of any customers wishing to purchase products from the web shop. The scope included features such as displaying articles in the web shop and the ability to select and buy one or more products with the automation of order creation in the business system. For further reading about the requirements, see chapter 2.

1.4 Organization of the thesis

This thesis is organized into seven chapters. Chapter 2 presents the functional and non-functional requirements that were decided upon before starting the project. Chapter 3 introduces the tools and techniques that were used in the implementation. Chapter 4 describes

the design of the system and chapter 5 describes the system setup and details on the implementation of the designed system. Chapter 6 describes the results of the implementation and compares the pre-determined requirements to what was accomplished. Finally, Chapter 7, provides the conclusion of the project.

2 Requirements

This chapter presents the requirements for the project. They are divided into two separate categories, functional- and non-functional requirements. The functional requirements specify requirements that are directly linked to the functionality of the system such as for example displaying articles. The non-functional requirements include areas such as adherence to security standards, computational adherence, or scalability.

2.1 Functional Requirements

This section presents all the functional requirements that are used to verify that the POC has been developed according to expectations. The requirements have an assessment criteria that can be used to determine whether or not the requirement has been met. The requirements are summarized in table 1 below.

Table 1: Functional requirements

ID	Name	Description	Assessment criteria
1.1	Show products	All the products stored in Visma should be presented in the web shop.	Articles with information such as price, quantity in stock etc. are visible in the web shop.
1.2	Purchase products	Customers can buy one or more products at a time.	Customers can buy one or more products at a time.
1.3	Add/remove products	Customers can add or remove products before a purchase has been completed.	Customers can add or remove products before a purchase has been completed.
1.4	Order should be created	An order should be created in Visma when a purchase is made in the web shop.	An order should be added to Visma after a purchase has been completed in the web shop.
1.5	Quantity in stock should be updated	When the purchase has been made and the order created, the quantity in stock of the given article(s) should be updated accordingly.	When the purchase has been made and the order created, the quantity in stock of the given article(s) should be updated accordingly.

2.2 Non-Functional Requirements

Table 2 below presents the non-functional requirements that are used to describe criteria that are not related to the functionality of the program, but rather implementational features that describe the environment in which the POC will be running.

Table 2: Non-functional requirements

ID	Name	Description	Assessment criteria
2.1	Web shop in SiteVision	The web shop should be developed in SiteVision Content Management System.	Web shop developed in SiteVision
2.2	Independent of CMS	The integration should be independent of what CMS the web shop has been developed in.	N/A

3 Background

This chapter explains the different tools used as parts of the development of the POC. The concept a business system is explained. The chapter also explains the business system that was decided upon for this POC. A content management system called SiteVision was used for developing and hosting the web shop. This non-functional requirement was imposed by Soleil IT as part of the project. A part of the investigation was to find out if it was feasible to integrate a business system with this specific CMS. This chapter also explains the general techniques used when working with RESTful services[8] to communicate information between different networks.

3.1 Business Systems

This section provides a short introduction to business systems. It gives an understanding of what they can be utilized for as well as a couple of example systems that are available on the market. The section also provides more information on the specific business system used in this POC to give the reader a better understanding of what this system can offer for the solution. As part of the project specification, Soleil IT wanted to investigate the possibilities of integrating a business system with the content management system called SiteVision which is explained in section 3.2. Soleil IT had investigated a number of business systems to choose from and Visma was chosen because it is popular among small and medium businesses in Sweden. Visma also has a variety of sub-systems to choose from that could fit the requirements for supporting an integration with a web shop. Visma and its sub-systems are further described in section 3.1.2.

3.1.1 Introduction to Business Systems

A business system can be defined as a standardized business comprehensive support system [23]. Most business systems are commercial off-the-shelf software (COTS), meaning that they are standard solutions often focused on specific areas of business. As a result of most systems being standardized solutions for specific areas of business, it is crucial to find a business system that has been developed to suit the specific needs for the given business it is intended to be used for [23].

There are a number of business systems available for those who wish to automate parts of their daily business. Depending on the size and focus of the organization there are more or less suitable options available. One example of a business system is Dolibarr, which is an open source system that is most suitable for smaller organizations and freelances [3]. Another example that is more suitable for larger enterprises is SAP ERP, which is a more complex solution that includes features covering everything from logistics features such as inventory, to production features such as project management [11].

3.1.2 Visma

Visma is one of the more popular business system providers on the Swedish market with over 340 000 customers in Northern Europe mostly focusing on small- and medium size businesses, although they also offer products for larger businesses and enterprises [18].

Visma offers its customers a wide range of applications based on their needs. Table 3 below shows some of the more popular products that Visma has to offer.

Table 3: Visma products

Name	Description	Focus group
Visma .net	complete business system with support for cloud.	All businesses
Visma eEkonomi	Accounting and invoice management.	N/A
Visma Administration	Accounting, invoicing, order management, warehousing etc.	Small and medium business.
Visma Proceedo	Invoicing, order management etc.	Freelancers and small and medium business.

The project required a business system that handled articles, invoices, customers and orders. After investigating the different business systems that Visma offered, Visma Administration was chosen as it had support for all of the functionality that was required as well as having good support for integration with third party software through its API Visma Integration. This will be further described in section 3.1.4.

3.1.3 Visma Administration

Visma Administration is one of the business systems that could offer control over invoicing, accounting, order management, warehousing etc. that a web shop would require. Visma Administration comes in four different packages, depending on what you need for your business. Table 4 below specifies the four different packages available for Visma Administration.

Depending on the customer's business, it is possible to choose one of the above packages. In our case, where a prototype was to be developed to investigate the possibilities of integrating Visma with a web shop developed in SiteVision, Visma Administration 2000 was chosen. Visma Administration is in itself a complete business system where the customer can via a graphical user interface (GUI), control and manage for example invoicing and warehousing. The goal in our case however, was to integrate some of the functionality in Visma Administration, with the web shop. This integration required an application programming interface (API) for Visma Administration, called Visma Integration.

3.1.4 Visma Integration

Visma Integration[19] is the API used to integrate web shops or other third party products with Visma Administration. This API is supported by any programming language that can import and use a dynamic-link-library file (dll), i.e. a Microsoft library file[6]. Besides the actual API, which is basically just the dll file ADK.dll (or for .NET: ADKNetWrapper.dll), Visma Integration comes with extensive documentation and example programs which are executable and integrate some functionality with Visma Administration.

Table 4: Visma Administration packages

Package	Description
Visma Administration 200	Offers payment control and accounting.
Visma Administration 500	It builds upon Administration 200 and, in addition, it offers invoicing capabilities.
Visma Administration 1000	It builds upon Administration 500 and, in addition, it offers ability to manage orders and warehousing.
Visma Administration 2000	It builds upon Administration 1000 and, in addition, it offers ability to generate offers to customers.

3.2 Content Management Systems

This section gives a brief introduction to content management systems as well as provide some examples of such systems that could be used when developing the web shop.

3.2.1 Introduction to Content Management Systems

Content management systems (CMS) are systems that can be used to collect, support, organize and publish information on the Internet and intranets [22].

There are several content management systems available today. Many of them are freely available and therefore could be better suited than SiteVision for smaller services or personal use. One popular example of a freely available CMS is Wordpress. Wordpress started out as a blogging system but over the years it has developed to a full content management system that is both suited for private users and larger companies or organizations [21]. Another popular and freely available CMS is Joomla. Joomla has been used to develop millions of web sites. It also offers great customizability[4].

3.2.2 SiteVision

SiteVision is a CMS that allows its users to develop web sites with minimal programming effort. It also allows for content to be managed by the so-called editors. An editor is someone usually working for the end customer that will be able to edit what is displayed in the predefined content areas of the web site. This means that the web site can be easily adapted and be more dynamic, without it requiring any programming skills from the editor [16]. SiteVision offers around 100 separate modules that include everything from simple text editors to interactive modules such as e-mailing as well as integration modules [14].

3.3 RESTful Services

RESTful services are services which follow a set of guidelines and best practices for communicating over a network and are not tied to any specific platform or technology [7]. A RESTful service typically communicates by using HTTP. Table 5 provided by Microsoft Developer Network [7] shows some of the methods defined in the HTTP with a short description of what they do as well as if they are considered safe methods or not.

Table 5: Some HTTP methods

Method	**Description**	**Safe**
GET	Requests a specific representation of a resource	Yes
PUT	Create or update a resource with the supplied representation	No
DELETE	Deletes the specified resource	No
POST	Submits data to be processed by the identified resource	No
HEAD	Similar to GET but only retrieves headers and not the body	Yes
OPTIONS	Returns the methods supported by the identified resource	Yes

RESTful services are often preferred over other web service techniques such as Simple Object Access Protocol (SOAP) as they do not require the same bandwidth as SOAP. This makes it more suitable for when working with the internet [24]. When sharing data using REST, a developer would typically choose between extensible markup language (XML) [20] and JavaScript Object Notation (JSON) [5]. XML involves reading an XML file from the designated web page, which should include all the requested information. XML is by many considered more verbose than JSON [1].

JSON seems according to most developers to be preferred for its straightforward use over the Internet. If the web site uses JavaScript, JSON would be considered the natural choice with the ease of parsing a JSON object [2]. Most of the arguments for using JSON over XML and vice versa seem to be subjective, although there are some measurements showing the performance increase associated with using JSON over the alternative XML. A simple test performed by www.codeproject.com shows a comparison between the two alternatives where the same set of information was compressed, then sent over the internet and finally decompressed. The test shows that with the JSON representation, the data was considerably smaller than with the XML representation and as such, JSON offered an advantage in performance. The XML representation of the text was 84.38% larger than the JSON representation [1].

Although it would be beneficial with a larger test suite to obtain a more accurate set of readings, it indicates that besides the more subjective advantages with JSON there might also

be some hard facts supporting the decision of using JSON rather than XML.

3.4 Summary

This chapter provided information about what business systems and content management systems are. The chapter also provided some alternative systems that could be used if the products used in this POC are not feasible or requested. It has also explained the REST technology that can be used to transfer data between servers on separate networks. In the next chapter, the design of the POC that was developed will be explained and in the following chapter, how this design was implemented and resulted in a complete POC.

4 System Design

This chapter describes the flow of information that is required to exchange data between Visma Administration and the web shop. Chapter 5 will describe how this has been set up and implemented.

4.1 Open Company in Visma Administration

Before the system can start the communication to share information between Visma Administration and the web shop, it must first establish a connection to a specific company administered in Visma Administration. To establish a connection with Visma Administration, a company needs to be opened. When installing Visma Administration 2000 on a server, a practice company is provided with example invoices, articles and orders. This is the company that has been used for this POC. This practice company is provided in a demonstrational environment, i.e. no invoices or orders will be physically produced.

4.2 Get All Products

When the connection with the company in Visma Administration has been established, information can start to flow between the end points. The first and foremost task for the POC is to be able to get all the products from the company to the web shop that was developed on a SiteVision server. This involves two steps: firstly, the information has to be collected by the SPCS integration module from Visma Administration (see chapter 5.3). Secondly, the information has to be forwarded from the SPCS Integration Module to the SiteVision server. The information about products that are being shared is in this POC the following:

- Article number
- Article name
- Quantity in stock
- Price
- Description of the article

4.3 List All Products

When all the products have been shared with the SiteVision server hosting the web shop, they can be displayed for future customers to inspect.

Figure 1 shows the initial mockup of how products are intended to be displayed on the webpage [1]. The customer is be able to view the price, a picture and a short description of the product. They are also able to add the product to a shopping cart or continue their shopping and then pay for all items at once.

[1] The product image has been borrowed from Apple Inc. (http://store.apple.com/se/buy-iphone/iphone6)

Figure 1: Products on web shop mockup

4.4 Shopping Cart

The shopping cart is a collection of products that the customer is able to use in the check out phase and complete a purchase with his/her contact information.

Figure 2: Shopping cart mockup

The shopping cart allows products to be removed or the quantity increased, and as explained above, to be checked out and generate an order.

4.5 Create Order

When a customer selects which products he/she wants to purchase, he/she would check out the shopping cart and fill in customer information consisting of:

- Social security number
- Full name
- Street address for delivery
- Zip code
- City
- Email

All this information is sent to the SPCS integration module and from there, an order in Visma Administration is created using all of the above-mentioned data. Visma Administration then calculates the total price of all the products so this information does not have to be transmitted. Visma also updates the warehouse status of the products involved by removing the given quantity of each product selected. An order will then be created and can be viewed

in the Visma Administration user interface which can then be printed and if required, sent along with the order. This step is however not included in the POC as it is related to the logistics end of the process, rather than integrating Visma with the web shop and is as such a decision for the logistics department of the company using this system.

5 System Setup and Implementation

This chapter describes how the system has been set up and how the design described in chapter 4 has been implemented to produce a complete POC.

5.1 System Setup

The system consists of four separate intercommunicating components. Figure 3 illustrates the four components and how the setup looks. Visma Administration 2000, Visma Integration API and Soleil SPCS integration module, are located on a server. A separate server hosts SiteVision. This was done to investigate the situations where customers might request separate systems for Visma. To have SiteVision on a separate server then, is not a requirement for setting up the system, but rather for demonstrational purposes. The fourth component is the SiteVision client, which is run in the web browser of any visitors to the web shop and to use when developing the web site.

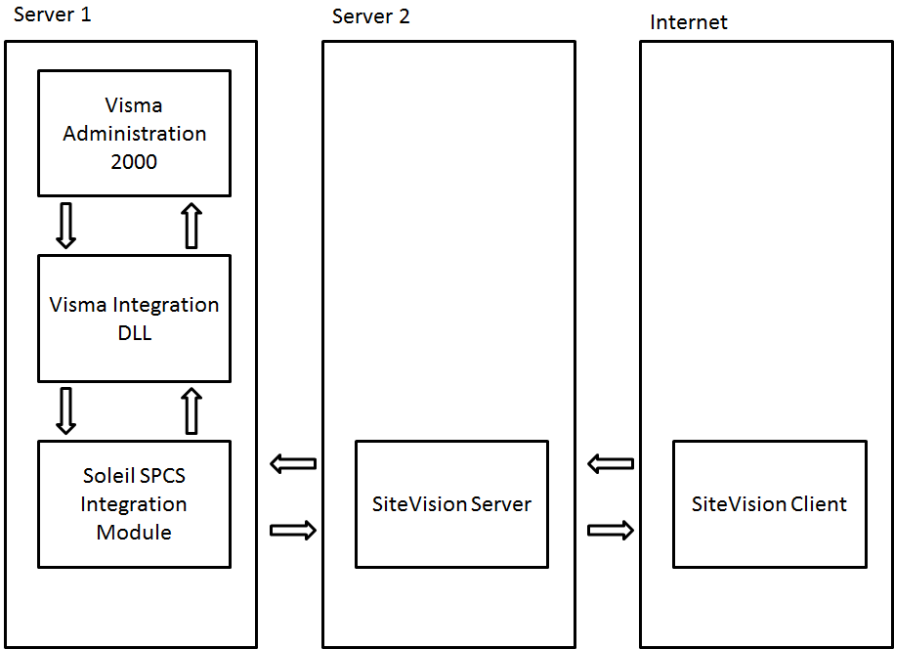

Figure 3: System setup

5.1.1 Server Environment

This sub-chapter will briefly explain the server environments for the servers used in the POC with minimum system requirements.

Server 1. Visma Administration is built on a Microsoft platform and requires a Microsoft environment to run in. Server 1 has the following system requirements [18]:

- Platform: Microsoft Windows Vista/ Windows 7/ Windows Server 2003 or newer
  - No Home or Basic versions of Microsoft Windows are compatible
- Disk space: Minumum 900Mb

Server 2. SiteVision was developed in Java and as such, is platform independent. It can be run on any platform using a modern Java virtual machine (JVM) [15].

- Platform: Any Microsoft, Linux, Unix or Macintosh operating system
- Java 5 or newer is required on this server.
- CPU: minimum 2 GHz, preferred dual or quad core processing
- Disk space: Highly dependent on what is being developed, at least 50 GB is recommended where 300 Mb is allocated for SiteVision installation and the rest for web sites resources.
- Memory: Minimum 2Gb

Client. To develop the web site in SiteVision a browser with JavaScript activated is required and a Java-plugin needs to be installed. Internet Explorer 6 or newer, Mozilla Firefox 2 or newer for Windows, Safari 2.0 or newer for Mac as documented, however any newer versioned browser would work for development [15].

5.2 Visma Integration API

In this section fundamental parts of the Visma Integration API are explained, including most used functions. Only a handful of the functions and data structures that the API has to offer is included in this report. I have selected those that offer the greatest value to anyone attempting to develop a similar system to the one presented in this thesis.

5.2.1 Installation

The API consists mainly of the file Adk.dll which is a library constructed by Visma. The file will be placed in the same folder as Visma Administration and is tied to its version. This means that with any updates of the version, the Adk.dll file will automatically be updated. For this to properly function, the Adk.dll file can therefore not be moved. When developing a .NET application there are two extended .dll files that can be used instead of Adk.dll, they are however just extensions of this file and as such, they require the original Adk.dll to be current.

5.2.2 Programming Languages

When programming an integration system to work with Visma Administration through Visma Integration API, you need to use one of the programming languages that support native windows dll files. This includes the whole C-family, along with for example Java.

5.2.3 AdkOpen and AdkClose

AdkOpen is a function that is used to open the connection to a database for an existing company on Visma Administration. AdkOpen requires two parameters when called. The first parameter is a string that points towards where Visma Administration has been installed. The second parameter, also a string, points towards where the company is located on the server.

Upon calling AdkOpen, the credentials of Visma Administration are checked to verify the version of Visma to ensure that it is compatible with the current API. A check to verify that the company has been created using the correct version of Visma Administration is also performed when calling this method.

AdkOpen returns a structure of the type ADK_ERROR (see section 5.2.4) which can present information regarding the result of calling the method.

AdkClose simply closes the connection to the opened company that has been created calling AdkOpen. This method does not take any parameters nor does it return any value.

5.2.4 ADK_ERROR

ADK_ERROR is a structure used for error handling in the API. It consists of 5 attributes all represented as the datatype long:

- lRc: The single most important attribute. This attribute contains the actual error message. The message is represented as a code. When lRc is equal to 0, no errors have occurred. If lRc has a value other than 0 it means that some type of error has occurred. Which error the code represents can easily be found in the documentation provided with the API. There is also a function to get a description of the error that can be used instead.
- lDbTable: This attribute will show which database table was being used when the error occurred.
- lField: This attribute will show which actual field in the database that was being used when
- lFunction: The called functions code which is returning an error.
- lProgramPart: The code for the part of the API where the error occurred.

5.2.5 ADK_DATA

In the communication between Visma Administration and the integration module, a data structure called ADK_DATA is used. This data structure is used to send information back and forth between the integration module and Visma Administration. To create this data structure, the function AdkCreateData is used. It returns a pointer to the ADK_DATA data structure. This function takes one parameter that will specify what type of data structure it should return.

Below is an example of how to create an ADK_DATA data structure. As mentioned above, AdkCreateData returns an integer representing a pointer towards the data structure. The parameter in this example, ADK_DB_CUSTOMER shows that the type of ADK_DATA that will be created is of the type Customer.

```
Int pData = AdkCreateData(ADK\_DB\_CUSTOMER);
```

5.2.6 AdkGet and AdkSet

AdkGet is a collection of functions that is used to get data from the given data structure (ADK_DATA). Depending on what data is to be retrieved, the AdkGet call will have different parameters. The collection consists of 5 different functions that will all return different data types. All of the AdkGet functions return a data structure of the type ADK_ERROR (see section 5.2.4).

AdkGetString. AdkGetStr will get a string from a field in the given (ADK_DATA) data structure.

```
ADK_ERROR error = AdkGetStr(pData, ADK_CUSTOMER_NUMBER, ref custNum, 16);
```

The example above shows how AdkGetStr is used to get a customer number (typically, this line of code would be used while reading the table from first to last entry) and place it in a variable called custNum. The last parameter (16) indicates the size of the data, which for a customer number is 16 bytes. The function will then return a data structure of the type ADK_ERROR to display any errors occurring during function call.

AdkGetBool. AdkGetBool will get a Boolean from a field in the given (ADK_DATA) data structure.

```
ADK_ERROR error = AdkGetBool(pData, ADK_CUSTOMER_EU_CUSTOMER, ref check);
```

The above example shows how AdkGetBool is used to get information to see if a customer is an EU-customer or not. The last parameter check is to show where the result of the question will be placed. This is an integer that will represent true or false depending on the customer.

AdkGetData. AdkGetData will get a pointer from a field in a given data structure.

```
ADK_ERROR error = AdkGetData(pData, ADK_OOI_HEAD_ROWS, i, pRows);
```

The above example shows how AdkGetData is used to get a pointer from a field in the data structure pData in the table ADK_IIO_HEAD_ROWS. The second to last parameter "i" points to a specific index and the last parameter pRows is where the result will be stored.

AdkGetDouble. AdkGetDouble gets a value of the data type double and is used much like AdkGetStr but for other tables that are not returning strings but doubles.

```
ADK_ERROR error = AdkGetDouble(pData, ADK_OOI_HEAD_DOCUMENT_NUMBER, ref InvNum);
```

The above example shows how AdkGetDouble is used to get the invoice number in the given data structure (pData) and store it in the variable InvNum.

AdkGetDate. AdkGetDate gets a date from a given data structure in the database.

```
ADK_ERROR error = AdkGetDate(pData, ADK_OOP_HEAD_DOCUMENT_DATE2, ref date);
```

The above example shows how the date of a given document (order or invoice) is retrieved and stored in the variable date.

AdkSet is, similarly to AdkGet, also a set of functions but used to set data in the database. These functions are used in a similar fashion to AdkGet and they are, as in AdkGet, a set of 5 functions (String, Bool, Data, Double and Date).

5.3 Soleil SPCS Integration Module

The Soleil SPCS integration module is the component of the system that communicates and shares information between the two end points Visma Administration 2000 and SiteVision Web shop. The communication with Visma Administration 2000 is based on, and is built using the given API Visma Integration that was described earlier in this chapter. The communication with the SiteVision web shop has been implemented using RESTful services. In this chapter it will be explained how the SPCS integration module was implemented for this POC by explaining what RESTful services is and how it was used as well as explaining how Visma Integration API was being used.

5.3.1 RESTful Services

When using Visual Studio 2013 for developing a REST Service in C#, the developer might use a standard template provided by Visual Studio for developing web services, called Windows Communication Foundation (WCF). WCF is actually first and foremost built to support SOAP but is still able to use REST services since it was released as part of the .NET 3.0 Framework [8].

Upon starting the implementation of REST services for the SPCS integration module, a decision was made to use JSON representation for the transmitted data rather than XML. The main reason for this was due to a smaller file size and hence more efficient transmission when using JSON as JavaScript was being used throughout the web shop (3.3).

In the POC there are two separate calls sent between the web shop and the SPCS integration module. The POC performs a GET call to the SPCS integration module to get all

```
1.  public static void Register(HttpConfiguration config)
2.  {
3.      config.Routes.MapHttpRoute(
4.          name: "DefaultApi",
5.          routeTemplate: "api/{controller}/{id}",
6.          defaults: new { id = RouteParameter.Optional }
7.      );
8.
9.      var appXmlType = config.Formatters.XmlFormatter.SupportedMediaTypes.FirstOrDefau
    lt(t => t.MediaType == "application/xml");
10.     config.Formatters.XmlFormatter.SupportedMediaTypes.Remove(appXmlType);
11.     config.Formatters.JsonFormatter.SupportedMediaTypes.Add(new MediaTypeHeaderValue
    ("text/html"));
12. }
```

Figure 4: Set JSON as default media type

products, and a POST call to add a purchase of a set of products. The developed WCF application follows a Model View Controller (MVC) setup and to navigate through to the right method from the URL when calling the SPCS integration module, one would follow the guidelines that can be seen in figure 4 before the media type has been set to JSON. The HttpConfiguration's route is determined by a call to HttpConfiguration.Routes.MapHttpRoute and the route is there set to "api/controller/id".

Controller is in the URL the specific controller that the request wants to communicate with and id is the parameters that the request can include. This is however optional.

```
1.  public List<Product> Get() {
2.
3.      return GetAllProducts();
4.  }
```

Figure 5: GET-method that returns a list of products

```
1.  public int Post(List<String> values)
2.  {
3.      /* SET UP READERS TO GET THE REQUEST PARAMETERS */
4.      StreamReader reader;
5.      HttpContext.Current.Request.InputStream.Position = 0;
6.      reader = new StreamReader(HttpContext.Current.Request.InputStream);
7.
8.      /* READ THE STREAM INTO STRING */
9.      string str = reader.ReadToEnd();
10.
11.     /* PARSE THE STRING TO JSON OBJECT */
12.     JObject json = JObject.Parse(str);
13.
14.     return 0;
15. }
```

Figure 6: POST-method with a JSON string as parameter

5.4 System Implementation

This section describes in detail how the information flow described in section 4 has been implemented to give the reader a good understanding, using code examples, on how the system has been constructed from a more in-depth point of view.

5.4.1 Open Company in Visma Administration

To start the communication between the end points, a connection to a company located on Visma Administration first has to be established. This is called "Opening a Company" and can be performed using Visma Integration API from the SPCS integration module by specifying the path to the company's location on the server as the location of where Visma is installed.

5.4.2 Get All Products

This section explains how the products are retrieved first from Visma Administration to the SPCS integration module, and then how the SPCS integration module transmits this data to the web shop.

Get all products from Visma Administration 2000. Products are retrieved from Visma Administration 2000 by using the Visma Integration API and stored in a list of the type Product. A product is declared to have a set of attributes that represent the data required to present it in the web shop. These attributes have been specified in the System Design chapter. Below in figure 7 the reader can see how the type Product can be declared to include the given attributes earlier specified.

```
1.  public class Product
2.  {
3.      Public int Id { get; set; }
4.
5.      public string Name { get; set; }
6.
7.      public string Desc { get; set; }
8.
9.      public int Price { get; set; }
10.
11.     public int Stock { get; set; }
12.
13.     public byte[] Image { get; set; }
14. }
```

Figure 7: Product is described by a set of attributes

The SPCS integration module calls Visma Administration to get the previously specified attributes (section 4.2) from all articles in the opened company by using the AdkGet functions described in section 5.2.6 Visma Integration API and store them in local variables. All of the articles are added to a list of the type Product. When all the products have been added to the list, the list is returned to the Get-method called by the web shop. Figure 8 illustrates how the attributes of an article can be loaded from Visma Administration 2000 to the SPCS integration module for further processing.

```
1. Api.AdkGetStr(pDataArt, Api.ADK_ARTICLE_NUMBER, ref ArtNummer, 16);
2. Api.AdkGetStr(pDataArt, Api.ADK_ARTICLE_NAME, ref ArtNamn, 50);
3. Api.AdkGetStr(pDataArt, Api.ADK_ARTICLE_REMARK1, ref ArtDesc, 60);
4. Api.AdkGetDouble(pDataArt, Api.ADK_ARTICLE_PRICE, ref ArtPris);
5. Api.AdkGetDouble(pDataArt, Api.ADK_ARTICLE_QUANTITY_IN_STOCK, ref ArtStock);
6. Api.AdkGetStr(pDataArt, Api.ADK_ARTICLE_IMAGE_PATH, ref ArtImgPath, 100);
```

Figure 8: Attributes are captured from article and stored in local variables

In figure 8, line 6 it can be noticed that ADK_ARTICLE_IMAGE_PATH is used to get the path to the picture for a given article. This path can then be used to load in the image. In figure 9 below it is shown how the image is loaded in. When sending data using JSON, the image should be converted to a string. One way of doing this is by converting the image to a byte array and to rebuild it in the web shop.

```
1. if(ArtImgPath != ""){
2.     myImg = Image.FromFile(imgPath);
3.
4.     img = imageToByteArray(myImg);
5.
6. }
```

Figure 9: Image is loaded and converted to byte array

When all attributes have been saved to local variables, and the image has been loaded in and converted to a byte array, a new product can be added to the list of products which will later be returned to the web shop. Figure 10 illustrates how this can be achieved.

```
1. products.Add(new Product
2. {
3.     Id = Convert.ToInt32(ArtNummer),
4.     Name = ArtNamn,
5.     Desc = ArtDesc,
6.     Price = Convert.ToInt32(ArtPris),
7.     Stock = Convert.ToInt32(ArtStock),
8.     Image = img
9.
10. });
```

Figure 10: New product is added to the product list

Get all products to Sitevision web shop. The web shop uses REST to get the data for all products represented by a JSON string and parses and displays them for the customer to inspect. In figure 11 it is shown how the SiteVision server calls the SPCS integration module with a GET request to get all products.

SiteVision utilizes JavaScript to perform a GET request to a specific URL and read the return values by using the InputStream. The InputStream can then be converted to a string for further use in the SiteVision Application. This string will be formatted for JSON as configured in the SPCS integration module (figure 4). This JSON String can be parsed to a

```
1.  var url = new Packages.java.net.URL("http://192.168.1.26:9091/api/product");
2.  var connection = url.openConnection();
3.  connection.setRequestMethod("GET");
4.  connection.setRequestProperty("Accept", "application/json");
5.  connection.connect();
6.
7.  var rd = new BufferedReader(new InputStreamReader(connection.getInputStream(), "UTF-
    8"));
8.  var sb = new Packages.java.lang.StringBuilder();
9.  var cp;
10.
11.
12. while((cp = rd.read()) != -1)
13. {
14.     sb.append(String.fromCharCode(cp));
15. }
16. var str = sb.toString();
17.
18. var object = Function("return " + str)();
```

Figure 11: GET-request to get articles

JSON Object for easy access to the data it contains. This can be done as figure 11 illustrates on line 18.

5.4.3 List All Products

When listing the products retrieved using the GET request to the SPCS integration module, a design framework like Bootstrap [2] can be used to make it easier for the developer. Bootstrap offers a developer the ability to quickly produce a design using the provided CSS and JavaScript which will automatically be responsive for developing web sites that should work on different sized browsers such as smart phones, desktops etc. In this POC, the products are contained in thumbnails, which are containers provided by the Bootstrap framework as can be seen in figure 12.

The article images as well as the information regarding the articles such as price and names have been borrowed from `www.komplett.se`.

Figure 12 shows how the articles have been displayed using Bootstrap thumbnails. The information is retrieved using GET request and is parsed and displayed. Customers are provided with an indication of whether the quantity of a given product in stock is sufficient or not, or if there are no articles left at all. This is done by checking the ADK_ARTICLE_QUANTITY_IN_STOCK value received with the GET request. The customer can also press the shopping cart icon to add the article to his/her shopping cart.

The picture of the article is received as a byte array (explained in 5.4.2) and can be converted back to an image using JavaScript as depicted in figure 13.

5.4.4 Shopping Cart

The shopping cart is intended to give the customer the possibility of purchasing more than one product per order. This has been implemented using an Open Source JavaScript solu-

[2] `http://getbootstrap.com/`

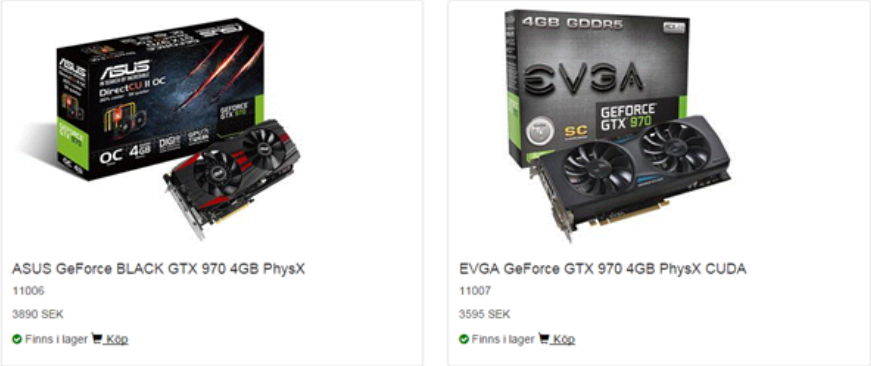

Figure 12: Bootstrap thumbnails are used to display the articles

```
1.  var img = "" + object[i].Image;
```

Figure 13: Image byte array converted back to image

tion called SimpleCart JS. It utilizes local storage to store the products and provides payment solutions like PayPal. The POC does not utilize the payment methods provided by SimpleCart JS because the preferred payment method was by invoicing which is supported through Visma Administration. Also, the payment solution provided by SimpleCart JS does not take in consideration that an order might need to be fully placed before an invoice is issued which also led to the decision of not using this feature. SimpleCart JS provides the developer with the ability to customize the information stored for each product by configuring the cart columns. This can be performed as illustrated in figure 14 below.

```
1.  simpleCart({
2.      cartColumns: [
3.          { attr: "name" , label: "Name" } ,
4.          { attr: "price" , label: "Price", view: 'currency' } ,
5.          { view: "decrement" , label: false , text: "-" } ,
6.          { attr: "quantity" , label: "Qty" } ,
7.          { view: "increment" , label: false , text: "+" } ,
8.          { attr: "total" , label: "SubTotal", view: 'currency' } ,
9.          { view: "remove" , text: "Remove" , label: false }
10.     ]
11. });
```

Figure 14: Customize columns in simplecart JS

It is also possible to change the default currency for the products. This can be achieved by following the example provided in figure 15 below.

To display the shopping cart in SimpleCart, an element with the class simpleCart_items can be created. This element will be updated every time a product is added or removed from the shopping cart. There is also the possibility of increasing and decreasing the quantity of

```
simpleCart({
    currency: "GBP" // set the currency to pounds sterling
});
```

Figure 15: Change the currency in simplecart JS

a given product in the shopping cart. The design of the shopping cart could be customized using CSS. As per default there will be no styling provided for the shopping cart and could for some be hard to read. Figure 16 shows the POC shopping cart with no styling added. Due to time limitations and low priority of design, no styling has been added to make the shopping cart more readable for the customer.

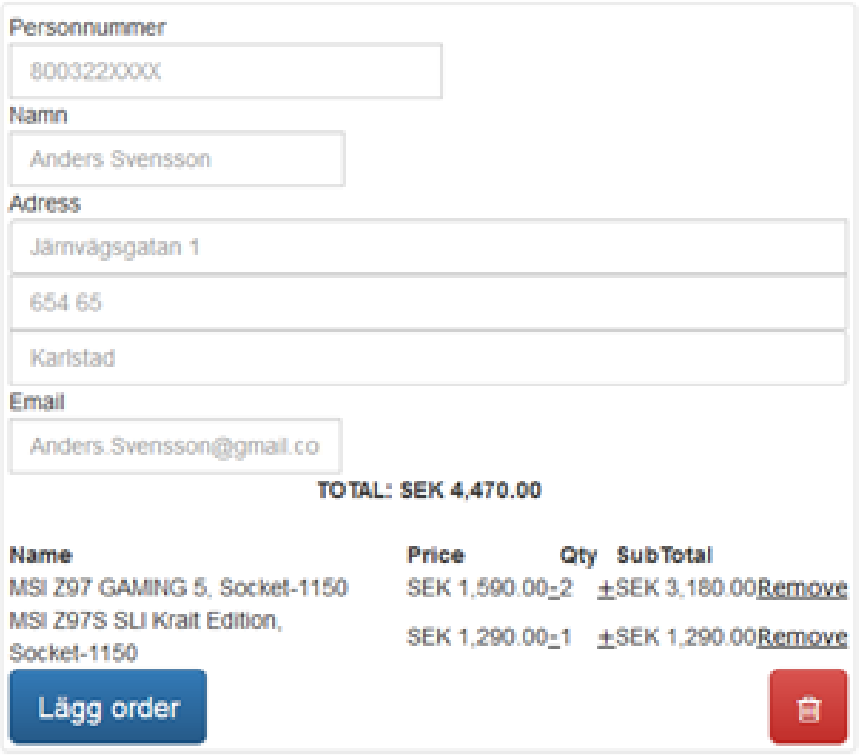

Figure 16: Shopping cart provided by simplecart JS

Above the product information in figure 16 there are also fields provided for the customer's contact information before placing an order. This contact information is later used when the order is placed and when a customer is created in Visma Administration 2000. This will be further explained in section 5.4.5. The contact information fields are not standard in SimpleCart but are instead added, along with the shopping cart, to a Bootstrap thumbnail. In the bottom of the thumbnail there are two buttons; one button for placing the order and

one button for removing all products from the shopping cart.

5.4.5 Create Order

In this section, the process of creating an order from the customer's shopping cart is explained in detail. I will describe how the POC creates a purchase object in the web shop, which passes it along to the SiteVision server which finally calls the Post method in the SPCS integration module where the order will be created in Visma Administration 2000.

Call SPCS integration module from Sitevision. When the button for placing an order in figure 16 is clicked; a JSON object is created by taking the article number and quantity for each product added to the SimpleCart shopping cart stored in local storage as well as the contact information that is filled out above the product list. This JSON object will have two keys; "contact" and "cart". This is the JSON object that will be passed to the POST request as a JSON string. A JSON object can be converted to a JSON string by calling the JavaScript function "stringify" . In figure 17 below is an example of how this JSON object can be created and also converted to a JSON string using the "stringify" function.

```
1.  /* CREATE COMPLETE PURCHASE WITH CONTACT AND PRODUCTS */
2.      var purchase = {contact, cart}
3.
4.  /* STRINGIFY COMPLETE PURCHASE */
5.      var jsonStringPurchase = JSON.stringify(purchase);
```

Figure 17: JSON object converted to string

As can be seen in figure 17, the object is first created and assigned to a variable called "purchase". This object contains two separate objects, or keys, named contact and cart. "contact" is all the information that has been gathered by the input fields for contact information, and "cart" is the data gathered for the articles.

This JSON string can be sent with a POST request to the SPCS integration module for completing the order. See below figure 18 as an example of how this POST request can look.

Create purchase object in SPCS integration module. When the above function is executed from the SiteVision server, the post method for the purchase in the SPCS integration module will be called. In this post method, the data passed on as parameter will be read and converted back to a JSON object and stored as a new variable of the type Purchase. An object of the type purchase has two attributes; a list of products called cart, and a Contact attribute, which holds the contact information. Figure 19 shows how the post method is constructed to parse the parameter and store it as a new Purchase before creating a new order in Visma Administration 2000.

In CreateOrder, which is called at the end of the post method showed in figure 19, a new order will be created using the parameter "purchase". Below it will be further described how the order is created using the purchase parameter sent along with the call for it.

Find customer / Create new customer. In Visma Administration 2000, all orders have to be associated with an existing customer number. In this POC, a customer number will be

```
var urlNew;
var connection = null;
try {
  //Create connection
  urlNew = new java.net.URL(url);
  connection = urlNew.openConnection();
  connection.setRequestMethod("POST");
  connection.setRequestProperty("Content-Type",
       "application/x-www-form-urlencoded");

  connection.setRequestProperty("Content-Length", "" +
           Integer.toString(urlParam2.length()));
  connection.setRequestProperty("Content-Language", "en-US");

  connection.setUseCaches (false);
  connection.setDoInput(true);
  connection.setDoOutput(true);
  out.println(urlParam2);

  //Send request
  var wr = new java.io.DataOutputStream (
              connection.getOutputStream ());
  wr.writeBytes (urlParam2);
  wr.flush ();
  wr.close ();
```

Figure 18: POST request sent to SPCS integration module

the social security number of the customer. When creating the order for a customer, the system will first check if the new order is being placed by anyone previously registered as a customer in SiteVision by searching in the existing customer numbers for the given social security number provided. If no customer is found, a new one will be created and the order will be placed with that customer. If a customer number is found, the order will instead be placed with this existing one. How this may be achieved can be viewed in figure 20 below.

By utilizing the ADK_ERROR data structure, the system can verify the results of a customer number search. Error code 115 will be returned if the search for a customer number was not successful. If this happens, a new customer number can be created using the social security number provided.

Create new order. When a customer number has been created or found, the order can be created. This can be achieved by creating an order head and associate a number of rows to it. An order head contains the information about the given order and its associated customer such as delivery address, name etc. The rows added are for the products that the order contains. Figure 21 below gives an example of how the order head can be created.

After the order head has been created, the order rows can also be created. The number of rows to create is determined by inspecting the purchase object and counting the articles in it. Figure 22 shows how the number of rows can be determined and also created.

The last step to be completed before the order can be inserted into the Visma Administration 2000 Database is to combine the order head with the order rows. Figure 23 shows how this can be achieved as well as how to add the order to the Visma Administration 2000 Database.

```
1.  public int Post(List<String> values)
2.  {
3.      /* SET UP READERS TO GET THE REQUEST PARAMETERS */
4.      StreamReader reader;
5.      HttpContext.Current.Request.InputStream.Position = 0;
6.      reader = new StreamReader(HttpContext.Current.Request.InputStream);
7.
8.      /* READ THE STREAM INTO STRING */
9.      string str = reader.ReadToEnd();
10.
11.     /* PARSE THE STRING TO JSON OBJECT */
12.     JObject json = JObject.Parse(str);
13.
14.     int jsonSize = json.Count;
15.
16.     /* SAVE CONTACT INFORMATION FROM JSON OBJECT */
17.     string upnum = json["contact "]["upnum"].ToString();
18.     string uname = json["contact "]["uname"].ToString();
19.     string uaddress = json["contact "]["uaddress"].ToString();
20.     string uzip = json["contact"]["uzip"].ToString();
21.     string ucity = json["contact"]["ucity"].ToString();
22.     string uemail = json["contact"]["uemail"].ToString();
23.
24.     Contact contact = new Contact(upnum, uname, uaddress, uzip, ucity, uemail);
25.
26.     /* SAVE PRODUCT INFORMATION FROM JSON OBJECT */
27.     List<Cart> cart = new List<Cart>();
28.
29.     foreach (var prod in json["cart"])
30.     {
31.
32.         string id = prod["id"].ToString();
33.         string quant = prod["quant"].ToString();
34.         cart.Add(new Cart(id, quant));
35.
36.     }
37.
38.     /* CREATE A PURCHASE OBJECT FROM CONTACT AND PRODUCTS */
39.     Purchase purchase = new Purchase(contact, cart);
40.     CreateOrder newOrder = new CreateOrder(purchase);
41.     int orderNum = Convert.ToInt32(newOrder.ordernum);
42.     return orderNum;
43. }
```

Figure 19: POST method called for new order

Get order number. There might be a request to get a confirmation back to the customer that the order has been successfully created by providing him/her with an order number. The order number may be found by using the AdkGetDouble method and using the data structure used for creating the order earlier. This can be seen in figure 24 below.

5.5 Summary

This chapter described the environment as well as the setup of the system. It has also described how the design specified in chapter 4 has been implemented using RESTful services for communication between different networks as well as by using the Visma Integration API for communication between the Soleil SPCS integration module and Visma Administration

```
1.  public int findCust(string pnum)
2.  {
3.      pDataCust = Api.AdkCreateData(Api.ADK_DB_CUSTOMER);
4.      error = Api.AdkSetSortOrder(pDataCust, Api.ADK_CUSTOMER_NUMBER);
5.
6.      error = Api.AdkSetStr(pDataCust, Api.ADK_CUSTOMER_NUMBER, ref pnum);
7.      if (error.lRc != Api.ADKE_OK)
8.      {
9.          String errortext = new String(' ', 200);
10.         int errtype = (int)Api.ADK_ERROR_TEXT_TYPE.elRc;
11.         Api.AdkGetErrorText(ref error, errtype, ref errortext, 200);
12.         Console.WriteLine(errortext);
13.         pData = 0;
14.     }
15.
16.     error = Api.AdkFind(pDataCust);
17.     if (error.lRc == 115)
18.     {
19.         /*COULDNT FIND CUSTOMER NUMBER, CREATE A NEW ONE */
20.         error = Api.AdkAdd(pDataCust);
21.         if (error.lRc != Api.ADKE_OK)
22.         {
23.             String errortext = new String(' ', 200);
24.             int errtype = (int)Api.ADK_ERROR_TEXT_TYPE.elRc;
25.             Api.AdkGetErrorText(ref error, errtype, ref errortext, 200);
26.             Console.WriteLine(errortext);
27.             pData = 0;
28.         }
29.
30.         /* VERIFY THAT CUSTOMER IS CREATED AND IT CAN BE FOUND */
31.         error = Api.AdkFind(pDataCust);
32.         if (error.lRc != Api.ADKE_OK)
33.         {
34.             String errortext = new String(' ', 200);
35.             int errtype = (int)Api.ADK_ERROR_TEXT_TYPE.elRc;
36.             Api.AdkGetErrorText(ref error, errtype, ref errortext, 200);
37.             Console.WriteLine(errortext);
38.             pData = 0;
39.         }
40.     }
41.     return error.lRc;
42. }
```

Figure 20: Search for customer and create new one if it does not exist

business system.

```
1.  /* CREATE AN ORDER HEAD */
2.  pData = Api.AdkCreateData(Api.ADK_DB_ORDER_HEAD);
3.
4.  /* CONNECT CUSTOMER NUMBER, NAME, DELIVERY CREDENTIALS TO THE ORDER HEAD */
5.  error = Api.AdkSetStr(pData, Api.ADK_OOI_HEAD_CUSTOMER_NUMBER, ref purchase.
    contact.Upnum);
6.  error = Api.AdkSetStr(pData, Api.ADK_OOI_HEAD_CUSTOMER_NAME, ref purchase.
    contact.Uname);
7.  error = Api.AdkSetStr(pData, Api.ADK_OOI_HEAD_DELIVERY_CITY, ref purchase.
    contact.Ucity);
8.  error = Api.AdkSetStr(pData, Api.ADK_OOI_HEAD_DELIVERY_ZIPCODE, ref purchase.
    contact.Uzip);
9.  error = Api.AdkSetStr(pData, Api.ADK_OOI_HEAD_DELIVERY_MAILING_ADDRESS1, ref purchas
    e.contact.Uaddress);
```

Figure 21: Create order head

```
1.  /* DETERMINE THE NUMBER OF ROWS TO ADD TO THE ORDER */
2.  int nRows =  purchase.products.Count;
3.  int pRowData = Api.AdkCreateDataRow(Api.ADK_DB_ORDER_ROW, nRows);
4.
5.  /* CREATE A TEMPORARY DATA STRUCTURE THAT WILL BE GIVEN ONE OF THE ROWS */
6.  int i = 0;
7.  int pTempData = Api.AdkGetDataRow(pRowData, i);
8.
9.  /* CONNECT ARTICLES WITH THE ORDER */
10. foreach (Cart products in purchase.products)
11. {
12.     int index = 0;
13.     pTempData = Api.AdkGetDataRow(pRowData, i);
14.     error = Api.AdkSetStr(pTempData, Api.ADK_OOI_ROW_ARTICLE_NUMBER, ref product.Id)
    ;
15.     if (error.lRc != Api.ADKE_OK)
16.     {
17.         String errortext = new String(' ', 200);
18.         int errtype = (int)Api.ADK_ERROR_TEXT_TYPE.elRc;
19.         Api.AdkGetErrorText(ref error, errtype, ref errortext, 200);
20.         Console.WriteLine(errortext);
21.         pData = 0;
22.     }
23.     error = Api.AdkSetDouble(pTempData, Api.ADK_OOI_ROW_QUANTITY2, Convert.ToDouble(
    product.Quantity));
24.     if (error.lRc != Api.ADKE_OK)
25.     {
26.         String errortext = new String(' ', 200);
27.         int errtype = (int)Api.ADK_ERROR_TEXT_TYPE.elRc;
28.         Api.AdkGetErrorText(ref error, errtype, ref errortext, 200);
29.         Console.WriteLine(errortext);
30.         pData = 0;
31.     } i++;
32.     index++;
33. }
```

Figure 22: Rows are created and articles are added to them

```
/* SET NUMBER OF ROWS TO THE HEAD */
error = Api.AdkSetDouble(pData, Api.ADK_OOI_HEAD_NROWS, nRows);
if (error.lRc != Api.ADKE_OK)
{
    String errortext = new String(' ', 200);
    int errtype = (int)Api.ADK_ERROR_TEXT_TYPE.elRc;
    Api.AdkGetErrorText(ref error, errtype, ref errortext, 200);
    Console.WriteLine(errortext);
    pData = 0;
}

/* COMBINE ORDER HEAD WITH ROWS */
error = Api.AdkSetData(pData, Api.ADK_OOI_HEAD_ROWS, pRowData);
if (error.lRc != Api.ADKE_OK)
{
    String errortext = new String(' ', 200);
    int errtype = (int)Api.ADK_ERROR_TEXT_TYPE.elRc;
    Api.AdkGetErrorText(ref error, errtype, ref errortext, 200);
    Console.WriteLine(errortext);
    pData = 0;
}

/* ADD ORDER TO THE DATABASE */
error = Api.AdkAdd(pData);
if (error.lRc != Api.ADKE_OK)
{
    String errortext = new String(' ', 200);
    int errtype = (int)Api.ADK_ERROR_TEXT_TYPE.elRc;
    Api.AdkGetErrorText(ref error, errtype, ref errortext, 200);
    Console.WriteLine(errortext);
    pData = 0;
}
```

Figure 23: Combine order head and rows and add the order to the database

```
public double getOrderNum()
{
    double order = new double();
    error = Api.AdkGetDouble(pData, Api.ADK_OOI_HEAD_DOCUMENT_NUMBER, ref order);
    if (error.lRc != Api.ADKE_OK)
    {
        String errortext = new String(' ', 200);
        int errtype = (int)Api.ADK_ERROR_TEXT_TYPE.elRc;
        Api.AdkGetErrorText(ref error, errtype, ref errortext, 200);
        Console.WriteLine(errortext);
        pData = 0;
    }
    return order;
}
```

Figure 24: Get order number of created order

6 Evaluation

The main goal of this project was to investigate the possibilities of integrating a business system with a web shop which was developed in the content management system SiteVision. The project included building a POC where a fictive customer could inspect the products of a fictive merchandiser as well as purchase one or more of these products. This chapter will discuss the results of the project.

6.1 Results

To investigate the success of this project, the reader is referred to tables 1and 2 where both functional- and non-functional requirements were specified. These requirements will be further discussed and results will be evaluated in this sub-chapter.

6.1.1 Functional Requirement 1.1: Show products

The first functional requirement, 1.1, revolved around the product visibility in the web shop. All the products in Visma should be presented in the web shop for the customer to inspect. The customer should also be able to get information about the products such as price, quantity in stock etc. Figure 12 shows that the products are displayed with an image, as well as the name of the product, the article number, price and whether or not the quantity in stock is satisfactory or not. Additional information such as a description could be added to this as well but due to the lack of available fields in Visma Administration that this could fit into, this was discarded. Only a short description consisting of no more than 60 characters could be added.

6.1.2 Functional Requirement 1.2: Purchase products

Requirement 1.2 explained that customers should be able to purchase one or more products in the same purchase. To solve this, a shopping cart was introduced (figure 16). By utilizing the local storage of the browser, the products can be stored in an array and are accessed with the free software SimpleCart JS.

6.1.3 Functional Requirement 1.3: Add/Remove products

The third functional requirement specified that customers should be able to add and/or remove products before the purchase was completed. This was solved by using the functions provided by SimpleCart JS where the customer can increase/decrease the quantity of any given product in the cart as well as remove all products from the cart.

6.1.4 Functional Requirement 1.4: Order should be created

Requirement 1.4 specifies that an order should be created automatically in Visma Administration as soon as the purchase was completed by the customer. To solve this, the integration module uses the Visma Integration API to first create a new customer number, or by looking up an existing customer number, based on the social security number of the customer (as can

be seen in figure 20). It then creates a Visma order (figures 21 to 23). To verify that an order has been successfully created, the SPCS integration module looks up the order number and returns it to the web shop where it can be further handled to confirm for the customer that the order was successful (figure 24).

6.1.5 Functional Requirement 1.5: Quantity in stock should be updated

The last functional requirement specified that when an order had been successfully created, the quantity of the given articles should be updated.

By creating an order in Visma Administration with a set of articles, the stock of those articles will automatically be updated by Visma Administration without any interaction from an administrator or, in this case, from the SPCS integration module.

6.1.6 Non-Functional Requirements

As for the two non-functional requirements 2.1 and 2.2 (table 2), these are considered to be met as the web shop was developed using the SiteVision content management system and that the SPCS integration module is independent of the platform used to develop the web shop. This is achieved by using RESTful services to communicate between the SPCS integration module and the web shop. Since RESTful services are not platform dependent they can be used with any implementation [9].

6.2 Implementation Evaluation

As can be read in section 6.1, all the requirements agreed upon before starting the project have been met and the POC was completed successfully. The POC may be extended to incorporate additional functionalities that will be discussed in sections 6.2.1 to 6.2.3. These properties were agreed upon, when starting the project, to only be mentioned as possibilities due to time limitations of the implementation.

6.2.1 Security

Whenever two systems located on separate networks communicate, there are certain security risks that need to be taken into consideration. The communication that is the most vulnerable to attacks in this particular setup is the one taking place between the web shop client (both client side and server side) and the SPCS integration module where RESTful services are being used. Some of the areas that require attention to ensure the integrity of the system are described below.

Authentication. Whenever a RESTful service is used, there should be some sort of authentication to ensure the communication taking place is the one the system is intended for. This can be done by for example using a session token or by using an API key as a body argument in a POST call. This data, whether it is a session token or an API key should never be displayed in the URL as they can easily be captured by web server logs [10].

Input Validation. Since the RESTful web service can be accessed by anyone, input validation is of great importance. By assisting the user to input correct data, and keeping a log of

all failed input attempts, the risk of a harmful use of the services can be minimized. Some fields, such as zip code, could easily be matched to contain reasonable data to ensure the validation. Other fields, such as plain text fields are harder to validate. Such fields should instead be sanitized. It would also be a good idea to rate limiting the API, as this can prevent abuse of the service by only allowing a certain number of access attempts per hour or day [10].

Output Encoding. The following suggestions refer to any project using JSON to transfer data, rather than XML. For more information regarding XML specific suggestions, the reader is referred to the OWASP web page [3].

To prevent any Javascript remote code execution within the browser, it is important to use a JSON serializer to encode the data provided by the user to prevent any user-supplied input on the browser [10]. It is also important, or at least strongly recommended, to use .value/.innerText etc. instead of .innerHTML to prevent any simple DOM XSS Attack [10].

6.2.2 Customizability

One of the main benefits of using a content management system such as SiteVision is that it enables the owner of the web site to customize it to specific events or specific requirements. This can be especially well utilized when developing a web shop as it gives the owner the ability to adapt to the market. By, for instance, setting up a time limited publication of certain products that depend on the time of the year, the owner can maximize the exposure of the most popular products at any given time with minimum effort. This is just one example of how SiteVision can be utilized to create a more dynamic web shop and if SiteVision were to be used to implement this solution, it would be strongly recommended to use the functionality that comes with it.

6.2.3 Payment Options

This POC has not been developed with any payment automation system due to the time limitation and this will instead be discussed here. Visma Administration includes support for creating invoices based on an order and this can easily be performed automatically by using similar techniques used to create the order as described in chapter 5. The invoices are provided by Klarna AB in Visma Administration. To adjust to the specific market of the web shop it might however be reasonable to include other payment methods as well that are well suited for the specific target group. Invoices are often considered to be the only option for companies but consumers might want to avoid this method of payment. Credit/debit cards and PayPal are two other options that are popular for this target group and could very well be implemented for this solution as there is support in Visma Administration for third party payment by announcing an order is to be pre-paid.

6.3 Summary

I have, in this chapter, discussed the results of the implementation where a successful integration between two separate systems has been performed to automate some tasks for web

[3] https://www.owasp.org/index.php/Web_Service_Security_Cheat_Sheet

shops. The requirements that were decided upon were met and some areas where the time limitations of the project prevented further implementation have been taken up.

7 Conclusions

This project has revolved around developing a Proof of Concept for Soleil IT to investigate the possibilities of integrating a web shop developed in a content management system called SiteVision. A successful integration was performed and showed that it was indeed feasible to perform such integration by using SiteVision and an intermediate integration module named Soleil SPCS Integration Module (see chapter 5.3) that used RESTful web services to communicate over the internet. This chapter will evaluate the project as a whole and present any problems that occurred during the implementation as well as possible work to be done in the future on this implementation.

7.1 Project Management and Execution

Upon starting the project I had little knowledge about system integration, web development and business systems. The language used to develop the SPCS integration module was C# and was a familiar language. Before the implementation started there was a period dedicated to research possible third party solutions to use for the integration and also research revolving around the different systems and techniques that were to be used. With the assistance of a supervisor at Soleil, the learning curve was significantly reduced as valuable knowledge could be transferred through design meetings and day-to-day communication.

This implementation was performed using an agile project method called Scrum [12]. Scrum is widely used and also the preferred project method for Soleil IT. The project was divided into Sprints that separated implementation from research and evaluation to provide a solid time plan for when milestones were to be reached. By using a web based tool called Scrumwise [13], stories could easily be described, estimated and tracked which provided a solid structure to the project (see figure 25 below).

7.2 Problems

The most time consuming item during the implementation of this POC was to understand the structure of a WCF application. Much of the logic has been predefined which makes it harder to see the full concept of the application. Since neither me or my supervisor at Soleil IT had any experience in developing a WCF application, some time had to be spent on researching this area to get a better understanding of the structure of such an application.

The development of the shopping cart in SiteVision also had its complications and this phase required more time than what was originally estimated. The problems revolved around the basic structure of SiteVision script modules which consisted of two separate parts where one part is for Javascript code that ran on the server side and one part is for Velocity which was executed on the client. Since the shopping cart utilized the local storage, it had to be operated from the client side but the RESTful services could not be called from there. How to transfer the data required some unplanned research which caused a slight delay in the time plan.

Figure 25: Scrumwise backlog

7.3 Future Work

As the project revolved around developing a Proof of Concept, rather than a finished product, this prototype will not be used in any current project at Soleil IT. The intention was to provide evidence that it was feasible to use SiteVision for integration with Visma. The findings showed that there are no limitations on SiteVision when it comes to integrating with third party hardware. The product company that developed SiteVision has shown considerable interest in the project and has been kept informed as the project progressed. The POC showed that an integration using SiteVision was possible and has led to further investigations as to how this can be utilized for a broader customer base than just merchandizing. There might also be an interest in investigating the possibilities of making this integration platform more independent to suit more alternative business systems rather than just Visma Administration.

If the product is to be further developed in the future it would benefit from incorporating the features specified in section 6.2.1 regarding security and to provide different payment methods.

References

[1] Codeproject json vs xml: Some hard numbers about verbosity. http://www.codeproject.com/Articles/604720/JSON-vs-XML-Some-hard-numbers-about-verbosity, accessed: 2015-04-03

[2] Digital Bazaar web services: Json vs xml. http://digitalbazaar.com/2010/11/22/json-vs-xml/, accessed: 2015-04-03

[3] Dolibarr ERP/CRM dolibarr erp/crm. www.dolibarr.org, accessed: 2015-04-18

[4] Joomla what is joomla. http://www.joomla.org/about-joomla.html, accessed: 2015-04-18

[5] JSON introducing json. :http://json.org/, accessed: 2015-06-01

[6] Microsoft definition and explanation of a .dll file. :https://support.microsoft.com/en-us/kb/87934, accessed: 2015-06-01

[7] Microsoft Developer Network a guide to designing and building restful web services with wcf 3.5. https://msdn.microsoft.com/en-us/library/dd203052.aspx, accessed: 2015-02-25

[8] MSDN Magazine an introduction to restful services with wcf. https://msdn.microsoft.com/en-us/magazine/dd315413.aspx, accessed: 2015-04-03

[9] MSDN Magazine more on rest. https://msdn.microsoft.com/en-us/magazine/dd942839.aspx, accessed: 2015-05-20

[10] OWASP rest security cheat sheet. https://www.owasp.org/index.php/REST_Security_Cheat_Sheet, accessed: 2015-05-05

[11] SAP om sap. http://www.sap.com/sweden/about.html, accessed: 2015-04-18

[12] Scrum scrum.org. https://www.scrum.org/, accessed: 2015-05-05

[13] Scrumwise scrumwise. https://www.scrumwise.com/, accessed: 2015-05-05

[14] Sitevision färdiga moduler och funktioner. http://www.sitevision.se/vara-produkter/sitevision.html, accessed: 2015-04-19

[15] Sitevision systemkrav i sitevision. http://www.tidaholm.se/download/18.554f65d312e2e65898780005198/Systemkrav.pdf, accessed: 2015-03-29

[16] Sitevision vad är sitevision. http://www.sitevision.se/vara-produkter.html, accessed: 2015-04-19

[17] Soleil IT soleil it. :http://www.soleilit.se/, accessed: 2015-06-01

[18] Visma systemkrav för våra ekonomiprogram. https://vismaspcs.se/produkter/systemkrav, accessed: 2015-03-29

[19] Visma visma integration. :https://vismaspcs.se/produkter/fler-produkter/visma-integration, accessed: 2015-06-01

[20] W3C extensible markup language (xml). :http://www.w3.org/XML/, accessed: 2015-06-01

[21] Wordpress about wordpress. :https://wordpress.org/about/, accessed: 2015-04-19

[22] Karlsson, T., af Gennäs, J.B.: Content Management Systems - Business effects of an implementation. Ph.D. thesis, Chalmers University, Gothenburg, Sweden (2005)

[23] Magnusson, J., Olsson, B.: Affärssystem. Studentlitteratur, Lund, Sweden (Dec 2008)

[24] Margaret Rouse: Techtarget representational state transfer. http://searchsoa.techtarget.com/definition/REST, accessed: 2015-04-03