

# E-COMMERCE PLATFORM INTEGRATION DEVELOPMENT

Case: Liana Technologies Oy

Klimenko, Artem

Bachelor's thesis  
School of Business and Culture  
Business Information Technology  
Bachelor of Business Administration

2019

<b>Author</b>	Artem Klimenko	Year	2019
<b>Supervisor</b>	Yrjö Koskenniemi		
<b>Commissioned by</b>	Liana Technologies Oy		
<b>Title of Thesis</b>	E-commerce Platform Integration Development		
<b>Number of pages</b>	43		

---

Modern businesses often have to deal with multiple software systems involved in their process of value creation. This fact often creates an excessive amount of overhead, which results in higher costs. In the diverse software context, the solution to this problem can be found by developing system integration. This thesis was focused on system integration development applied to a customer case.

This research had three main objectives. The first objective was to provide an understanding of what system integration is and how it can be developed. The second objective was to study how communication to web services can be established. The final objective of this thesis was to fulfill a customer case by developing the integration of an e-commerce platform with two external web services.

Constructive research approach has been used throughout this research work. This research targeted to solve a practical problem. Therefore, the theoretical knowledge was collected through a literature review in order to be applied to produce a new practical solution.

The outcome of this research was an e-commerce platform, which was complemented with an integrated logic, provided by external web services. The platform fulfills customer requirements and is deployed to a production environment. Additionally, the platform integration has been tested in order to mitigate possible risks.

**Key words** E-commerce platform, system integration, PHP, Plugin development, Plugin architecture

## CONTENTS

1	INTRODUCTION .....	6
1.1	Background and Motivation .....	6
1.2	Research Topic and Objectives .....	7
1.3	Scope and Limitations .....	8
1.4	Thesis Structure .....	8
2	RESEARCH QUESTIONS AND METHODOLOGY .....	10
2.1	Research Questions .....	10
2.2	Research Methodology .....	10
3	SYSTEM INTEGRATION .....	12
3.1	System Integration .....	12
3.2	Web Services .....	13
3.3	API Integration .....	14
4	LIANACOMMERCE PLATFORM .....	15
4.1	E-commerce Platform Definition .....	15
4.2	Plugin Architecture .....	15
4.3	Hook System .....	18
4.4	PHP and Liana Framework .....	18
5	WEB SERVICE COMMUNICATION STANDARDS .....	19
5.1	Web Service Standards .....	19
5.2	SOAP .....	19
5.2.1	SOAP Message Structure .....	19
5.2.2	WSDL .....	20
5.3	RESTful Web Services .....	21
5.3.1	Client-Server .....	22
5.3.2	Stateless .....	22
5.3.3	Cache .....	23
5.3.4	Uniform Interface .....	23
5.3.5	Layered System .....	23
6	REQUIREMENT ANALYSIS .....	24
6.1	Requirement Analysis Description .....	24
6.2	Stakeholder Analysis .....	24
6.3	Functional Requirements .....	25
6.4	Non-Functional Requirements .....	25

7 AVENUE WEB SERVICE INTEGRATION.....	27
7.1 Specification Analysis.....	27
7.2 Plugin Structure.....	28
7.3 Service Integration.....	30
7.4 Unit Testing.....	34
8 DYNAMICS CRM INTEGRATION.....	35
8.1 Specification Analysis.....	35
8.2 Service Integration.....	36
9 CONCLUSION.....	41
BIBLIOGRAPHY.....	42

## SYMBOLS AND ABBREVIATIONS

API	Application Programming Interface
B2B	Business-to-business
CLI	Command Line Interface
CRM	Customer Relationship Management
CRUD	Create Read Update Delete
ORM	Object-Relational Mapping
REST	Representational State Transfer
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SSL	Secure Sockets Layer
UI	User Interface

## 1 INTRODUCTION

### 1.1 Background and Motivation

In the 21st century, software systems are widely used across various spheres of human life. These spheres are extensively interconnected and there is a demand for interlinking the software systems to achieve better efficiency and performance in the field of application. The sphere of e-commerce is not an exception.

There are often multiple software systems involved in the process of online retail. This is clearly visible in the B2B sector when various business partners are utilizing software systems from different vendors and of a different application domain. In this context, system integration of various software services becomes crucial for business success. The system integration process is highly dependent on the internal architecture of the systems involved; thus, it is difficult to generalize the set of requirements and automate it.

This thesis studies the process of software integration development with the example of LianaCommerce e-commerce platform. LianaCommerce is cloud-based software, developed by Liana Technologies Oy. This study provides an overview of how e-commerce software can be integrated with other cloud-based software systems. More specifically, this thesis focuses on an integration of LianaCommerce platform with Microsoft Dynamics CRM (hereinafter Dynamics) and Avenue digital product management (hereinafter Avenue) systems.

The choice of this thesis topic has been fostered by the researcher's interest in software development and his professional involvement in the development of LianaCommerce platform. This work helps the researcher to broaden his knowledge on the topics of system integration development and web service communication. The possibility of professional growth as a back-end developer has been considered as one of the key motivating factors.

## 1.2 Research Topic and Objectives

The topic of the thesis is system integration development. More specifically, this research focuses on studying how Liana Technologies should develop system integration of their LianaCommerce e-commerce platform with Dynamics CRM and Avenue systems. This integration is a customer case designed to fulfill the requirements of a company in the education management industry (hereinafter EMI company), who is the user of LianaCommerce platform. Therefore, a set of user requirements is to be considered throughout the implementation phase and is provided by the EMI company.

This thesis has three main objectives. The first objective is to develop an understanding of how system integration can be developed. This objective requires theoretical information on software architecture, design patterns and module development approaches used in the target system.

The second objective is to learn how data transfer between various software systems can be done. To achieve this objective, API and other technical documentation related to the systems of integration will be studied. Information about protocols and service architectures used to share the messages between the web services will be obtained and analyzed. Based on the findings, relevant tools and development approaches will be selected to deliver the service integration implementation. Test requests to the systems of integration will be made to provide a proof of concept before proceeding with the implementation phase.

The third objective is to fulfill the customer case by providing an integrated e-commerce environment with custom business logic. This objective will be achieved by creating a system integration based on customer requirements. The implementation will result in the client being provided with an e-commerce solution which implements custom business logic tailored according to the requirements of the client.

### 1.3 Scope and Limitations

The scope of this thesis is limited to studying the technologies and tools which are relevant to the system integration of LianaCommerce platform with external systems. In other words, this study does not intend to observe various ways which could contribute to an integration of a software system. Instead, the study provides a detailed description of a particular system and system integration development within its technical context.

The limitations of this study derive from an existing technical environment and development guidelines established in the case company. These factors are limiting the choice of development tools and force the researcher to follow a defined coding style, utilizing a set of tools adopted in the company. However, the limitations mentioned above do not put any notable impact on the external validity of this research. Its results can be re-applied within a similar context in another technical environment with a minimal amount of adaptations.

Another limitation is related to Liana Technologies' security policy and is affecting publishing of the source code. Therefore, during the implementation phase, only the code which is crucial to the research objectives of this thesis and does not expose any business, security and privacy-critical information will be published.

Final limitation considers the database structure. During the implementation phase, the database structure and data storing mechanisms are not discussed, due to extensive utilization of object-relational mapping (hereinafter ORM) and object serialization by LianaCommerce platform.

### 1.4 Thesis Structure

To sequentially cover each topic, the structure of this thesis has been divided into 9 chapters. Chapters 1 and 2 are providing a discussion on what the thesis is about and describe the theoretical framework used throughout this work. Chapter 3 defines what software integration is and describes the type of



integration developed as part of this research's outcome. Chapters 4 and 5 provide an overview of the technical context of this thesis. Chapter 4 describes LianaCommerce platform, its plugin system, and development environment. Chapter 5 focuses on web services, their architectures and how the architecture defines the communication style used to communicate to the service. Chapter 6 collects and analyzes the requirements to be fulfilled during the implementation phase. Chapters 7 and 8 apply the theoretical knowledge, presented in the previous chapters, to provide an overview of the plugins developed as an outcome of this research. Chapter 9 draws a conclusion by outlining the results of this thesis and providing ideas for further improvement.

## 2 RESEARCH QUESTIONS AND METHODOLOGY

### 2.1 Research Questions

This study addresses the following research questions:

1. How can a system integration for an e-commerce platform be developed?

This question tries to identify a set of technological constraints which have to be fulfilled to integrate external systems to LianaCommerce platform. To answer this question, the module system and architectural patterns used inside of LianaCommerce platform are studied carefully.

2. What are the characteristics of the final system suitable for the case company?

This question identifies what the client company is trying to achieve with the system integration. It studies the requirements, business logic provided by the external systems and the desired business logic of the e-commerce platform.

3. How can possible risks related to the system integration be mitigated?

The purpose of this question is to identify how does the risk domain expand when external services are involved in the delivery of business logic. It aims to find solutions to minimize the risk if any is to be identified during the implementation phase.

### 2.2 Research Methodology

The primary methodology used throughout this research is constructive research approach. Piirainen & Gonzalez (2014) define it as a methodology which utilizes an existing theory to create an applied solution to a practical problem. According to Lassenius et al. (2001), constructive research seeks for

novel solutions. This thesis does not extend the existing theoretical base. Instead, it utilizes theory, collected by means of literature review, to create a practical outcome, which is an e-commerce platform integration. The theoretical knowledge is gained through literature reviews and documentation analysis. In addition, developers and software architects of the case company are interviewed to obtain technical guidelines and suggestions for the implementation.

### 3 SYSTEM INTEGRATION

#### 3.1 System Integration

System integration is a process of making separate applications to compose unified application logic by establishing automatic communication to each other (Viitala 2017). The system integration process should be seen as the utilization of various design patterns and architectural guidelines to compose a single application. As discussed in the introductory chapter, modern companies are utilizing a set of various distributed software systems which have a narrow application domain. The integration of those systems into a single one can result in better control over multiple business processes and benefit the entire organization. For example, information can be automatically shared between various applications, reducing the risk of an error which can occur if a user does manual input.

Hasselbring (2000) defines three architectural levels of an organizational unit at which integration can be performed. These units are presented in Figure 1 below.

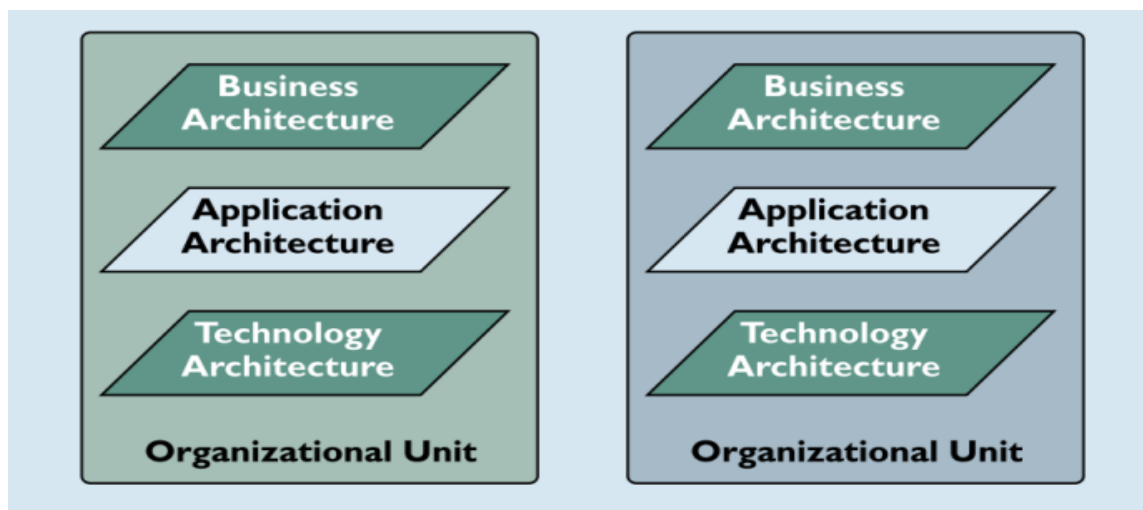


Figure 1. Fragmentation of Organization Units (Hasselbring 2000)

Business architecture layer contains a representation of the system's functionality in logical units, which are understandable for the users of the system. Application architecture layer contains an implementation of the

application's business logic. In addition, it is providing an interface between the technology architecture layer, which is responsible for the ICT infrastructure, and the business logic unit.

### 3.2 Web Services

Introduction of distributed and cloud-based computing has induced the popularity of web services. Services in computer science can be referred to as fine-grained units of logic, containing a limited set of functionality. This implies that web services are the services which are operating on the web. There exist multiple alternative meanings for the term "web service", shared across multiple domains; however, this research utilizes the following definition. According to W3C (2004), a web service is a software system designed to support interoperable machine-to-machine interaction over a network.

In order for the service to be reachable by other systems, it has to provide its interface description also known as API. According to Christensson (2016), an API is a set of functions and commands, used by programmers to interact with an external system. API is used to expose the functional interface of the service to the developers. It consists of a service descriptor, describing the parameters to be given; resources to be called and the structure of the data returned as a response.

The communication between the web services is typically done by sending XML or JSON messages transferred through Hypertext Transfer Protocol (hereinafter HTTP). The HTTP protocol is used due to it being widely supported across the browsers and operating systems. In case additional security layer is required, more secure version of HTTP – HTTPS is used.

Modern software applications are utilizing various languages and technological stacks and in order to facilitate communication, web services should be built context-agnostic. This means that they should comply with strictly-defined standards, which ensure correct functioning regardless of the technological context of the service requestor or the service provider. Therefore, web services

should fulfill two basic principles – statelessness and composability (Alemu, 2014, 14). Composability means that a web service can be combined with other web services to form an application. Statelessness characteristic requires that service’s response does not depend on the client metadata. Instead, it is defined by the request parameters and the resource identifier (Alemu, 2014).

### 3.3 API Integration

The integration type used throughout this research is called point-to-point API integration. This type of integration directly uses API of an external web service to obtain the necessary data. Viitala (2017, 11) defines point-to-point as a fast and simple way to connect two applications. Figure 3 represents a model of point-to-point integration.



Figure 3. Point-to-Point Integration (Linthicum 2001 as cited in Viitala 2017, 11)

The point-to-point pattern has been praised for its efficiency and is known to be cost-effective and easy to implement. It is often utilized in small enterprises when no customizations and interface changes are expected. Therefore, this integration pattern is relevant to the scope of this thesis, as the target integration is utilized by a single client and is not requiring any customizations. This pattern assumes utilization of middleware on the client side, which will provide message creation and connectivity to the web service. The technologies facilitating implementation of the middleware are discussed in the next chapter.

## 4 LIANACOMMERCE PLATFORM

### 4.1 E-commerce Platform Definition

LianaCommerce is an example of an e-commerce platform. E-commerce platform is software, specifically designed to provide merchants with a set of tools to control various aspects of their e-businesses. According to Oracle NetSuite (2018), an e-commerce platform is a software, which encapsulates core business functions under a single software solution; it fosters improved collaboration, aligns operational processes and provides real-time data visibility within organizations. An e-commerce platform cohesively delivers the following functions:

- Analytics and reporting
- Customer support
- Order and inventory management
- Procurement
- Content management
- Marketing
- Pricing and promotions

(Oracle NetSuite 2018).

For many companies, e-commerce platforms serve as a hub for their supply-chain management applications, such as CRM and ERP systems. This requires from the platforms to be flexible and to provide an easy API for integrating solutions from the other vendors into their workflow. Architectural solutions which ensure flexibility of LianaCommerce platform are discussed in the following sections.

### 4.2 Plugin Architecture

LianaCommerce platform allows extending its functionality employing external plugins, which are managed by its plugin system. This system can also be used to create middleware for developing a system integration to other cloud-based

services. LianaCommerce plugin system is done by using a microkernel architecture pattern (also known as plugin architecture pattern). Architecture patterns define on a high level how the application's code has to be organized in order to provide a solution to a common problem. According to Richards (2015, 21), microkernel architecture pattern allows the application's functionality to be extended by means of fine-grained units of logic, called plugins. Figure 4 illustrates the elements of the plugin architecture pattern.

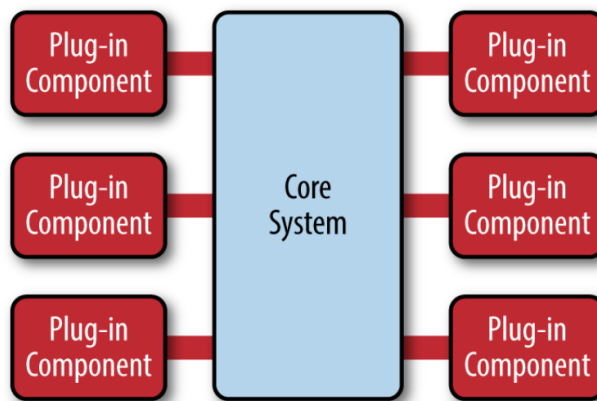


Figure 4. Plugin Architecture Pattern (Richards 2015, 22)

The core system is responsible for the delivery of the main functionality of the application. Plugins can be seen as isolated logical units, which extend the core system's logic and are responsible for a limited set of functionality. Plugins are often isolated from each other and are built according to the loose-coupling and separation of concerns (hereinafter SoC) principles. These principles state that every unit of logic should be responsible for a defined set of features and should not be tightly interconnected with the rest of the system (Richards 2015, 22).

The core system gets the knowledge of what plugins are available through a plugin registry, which contains information on what is the name of the plugin, what functionality does it expose for calling and what kind of configuration data does it require (Richards 2015, 23).

Plugin architecture pattern can be seen as a possible solution to a common software entropy problem illustrated in Figure 5 below.



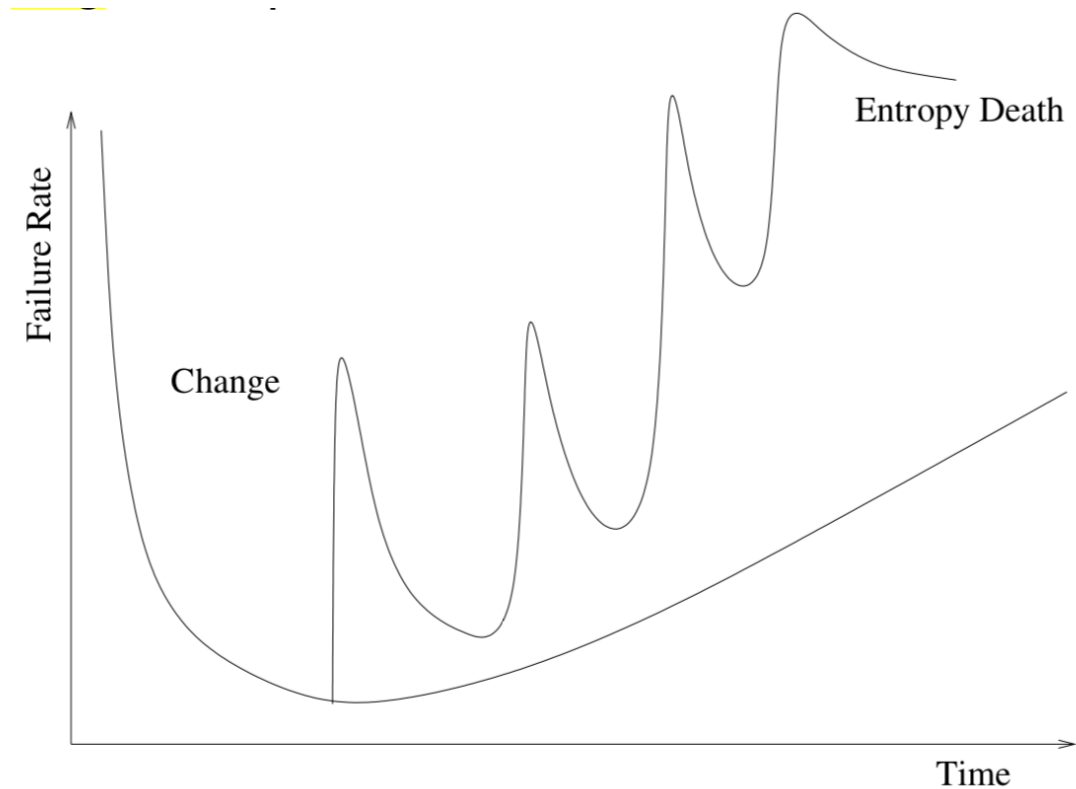


Figure 5. Software Failure Curve (Pressman & Maxim 2015, 6)

During its lifetime, software gets altered which leads to an increase in the failure rate. Consecutively, patches for the existing problems are released, and the software gets updated again. These steps are repeated multiple times, and according to Pressman & Maxim (2015, 5-7), the failure rate begins to rise contributing to software deterioration. When the changes affect the software so much that it becomes hard and unprofitable to fix the existing errors, the development team usually abandons the current version of the software and starts developing a new version from scratch.

The scenario mentioned above can be partially avoided by means of plugin architecture pattern. The software functionality is modified using loosely-coupled plugins, which are integrated into the core logic. This way every single part of added software functionality is encapsulated into a plugin and can be maintained without the rest of the software being affected. This approach can significantly reduce maintenance costs and simplify risk management.

### 4.3 Hook System

In order to manage different plugins, LianaCommerce is utilizing a system of hooks and filters. According to Wordpress (2018), hooks are defining a way for one piece of code to interact or modify another. A hook is placed inside of a function and serves as a function-trigger. There exist two types of hooks – actions and filters. (Wordpress 2018). Actions are calling a function inside of a plugin class, serving as function triggers. Whereas filters are calling a function with a parameter given by a reference. Called function is modifying the given data and the data is used further during the execution. Hook system is used to connect the plugin functionality with the core system's logic so that they can perform as a whole.

### 4.4 PHP and Liana Framework

LianaCommerce platform is written using PHP language. This language is widely used across the Web and is one of the main languages there. In order to extend the essential features of a programming language and provide advanced functionality, software frameworks are used. A software framework is a set of essential functions, provided through an API, which can be re-used by the developers. Software frameworks are often based on widely-adopted architecture design patterns, which positively affects the code quality.

LianaCommerce platform development is done using Liana framework, developed by Liana Technologies; therefore, this framework is utilized during the practical part of this research. Liana framework provides a variety of useful features some of which involve ORM, which allows automatic serialization of objects into the database. Plugin registry and the hook system are also provided by the framework and do not need to be implemented from scratch. The framework helps to maintain a high quality of code by providing access to design pattern implementations, which can be reused by the developers. Also, framework utilization significantly improves the development speed and code security by providing encoding, encryption, authentication, data transfer, and connectivity functionality.

## 5 WEB SERVICE COMMUNICATION STANDARDS

### 5.1 Web Service Standards

As the practical part of this thesis deals with web services, it is crucial to define and describe technologies laying behind the communication and implementation of them. In order for a service to be accessible through the web, it should be built in compliance with a set of web service architecture standards. Those standards in their basic form define the way a web service is providing its functionality, which protocols are used for the communication and what message structure is utilized. Nowadays, there are two widely-used web service implementation approaches – SOAP web services and RESTful HTTP. The following chapter provides a detailed description of both of them, as well as, compares them against each other. The web service standards ensure interoperability among various operating systems, programming languages and cloud services. In order to understand web services, understanding of their standards should be obtained. (Chen et al. 129).

### 5.2 SOAP

Simple Object Access Protocol (hereinafter SOAP) is one of the most widely-used web service standards. According to W3C (2007), SOAP is a lightweight protocol dedicated to exchanging information in a distributed and decentralized environment. SOAP is an XML-based communication protocol; therefore, all the data transferred through SOAP has to be serialized into XML format first. XML messages used inside of SOAP are called SOAP envelopes and contain response data from the service.

#### 5.2.1 SOAP Message Structure

SOAP protocol defines a structure for an XML message; therefore cross-platform and cross-language compatibility is ensured, which significantly reduces development overhead. According to Christensson (2006), Each SOAP message is contained in an envelope, which always includes the header and

the body elements. These elements are known as required parts of the message. SOAP message is encoded as an XML document, consisting of an <Envelope> element, an optional <Header> element and a <Body> element. <Body> element can optionally contain a <Fault> element for error reporting (IBM Knowledge Center 2018). An envelope is a SOAP message root element used for identifying the beginning and the end of a SOAP message. A header is an optional element of a SOAP message, and it contains information on how the message should be processed. SOAP body is a mandatory element containing data payload transferred to the message receiver. SOAP fault element is used to provide verbose information if an error occurs during SOAP request processing. Figure 6 provides an example of a basic SOAP message structure.

```

<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
               soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">
  <soap:Header>
    ...
  </soap:Header>
  <soap:Body>
    ...
    <soap:Fault>
      ...
    </soap:Fault>
  </soap:Body>
</soap:Envelope>

```

Figure 6. SOAP Message Structure

## 5.2.2 WSDL

Web Services Description Language (hereinafter WSDL) is an essential part of a SOAP web service. WSDL contains a machine-readable description of all the service endpoints exposed by a web service and provides the client with a description of the service request and response parameters. According to W3C (2001), WSDL uses an XML format to describe network services which communicate using messages with a document- and procedure-oriented information. This implies that WSDL is not restricted to any specific message format or a network protocol, making it applicable to protocols other than SOAP.

### 5.3 RESTful Web Services

The second widely-used way for implementing web services is Representational State Transfer (hereinafter REST). There is a common misconception, that REST is a web service communication protocol. However, REST is much more than that. REST concept has been defined by Roy T. Fielding. Fielding has formulated REST by inheriting constraints from various existing software architecture styles and extending them with a set of new ones. Introduction of constraints was motivated by a necessity to formalize the communication between the services and provide a reliable standard which would eliminate the diversity of API implementations on the Web. Fielding (2000), defines REST as an architectural style to be used with distributed hypermedia systems. According to Shaw & Clements (1997), architectural style defines components and connectors used to create a system. In contradiction to SOAP, which is protocol-neutral, REST was designed intending to extensively utilize HTTP protocol, which is the main application-layer protocol of the internet. HTTP provides a variety of methods, which facilitate various CRUD operations such as GET, POST, PUT and DELETE. There exist many more, but the abovementioned ones are considered to be the sufficient minimum for a REST implementation. Figure 7 provides a correlation of CRUD functions with the respective HTTP and SQL functions.

<b>Action</b>	<b>SQL</b>	<b>HTTP</b>
<b>Create</b>	Insert	PUT
<b>Read</b>	Select	GET
<b>Update</b>	Update	POST
<b>Delete</b>	Delete	DELETE

Figure 7. CRUD Actions

As the REST architecture has been already covered it is time to clarify the term “RESTful.” A RESTful web service is a service built according to the REST architecture. Following sub-sections describe REST constraints which need to be fulfilled by a web service in order to implement the REST architecture.

### 5.3.1 Client-Server

The client-server constraint is based on the separation of concerns principle. This constraint is dedicated to separate service requestor and service provider logic so that these components can be altered separately. This constraint introduces an abstraction to the implementation of the service. Communication is done through a well-defined interface, and the client does not know the implementation details of the server. (Fielding 2000). Figure 8 provides a schematic description of a client-server constraint.

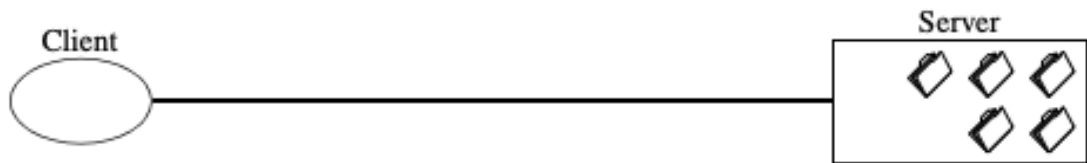


Figure 8. Client-Server Constraint (Fielding 2000.)

### 5.3.2 Stateless

Stateless is another REST constraint defined by Fielding. It enforces the communication between the client and the server to be stateless, inducing the information provided as a response to be affected only by the parameters provided in the request. Response information must not be affected by any client-related data which is not stored on the server; therefore, all the session-related data has to be managed by the client. This constraint helps to improve the performance of the server and positively influences the scalability of the service as the responses can be cached and delivered to several clients (Fielding 2000.) It also improves the key characteristics of a service – scalability, visibility, and reliability. Figure 9 illustrates the concept of stateless constraint.

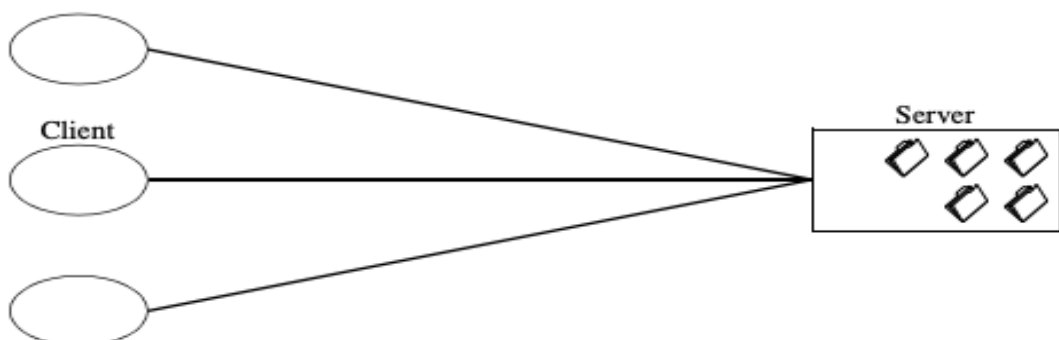


Figure 9. Stateless Constraint (Fielding 2000.)

### 5.3.3 Cache

Cache constraint states that when a client obtains a response from the server, the payload might be cached by the client so that the same request does not have to be done multiple times. The usage of caching on the client side can significantly reduce the server load and therefore is a widely-adopted practice even beyond REST context. Roy Fielding in his dissertation states, that “the most efficient network request is the one that doesn’t use the network” (Fielding 2000).

### 5.3.4 Uniform Interface

This constraint is what makes REST different from any other service architecture. Uniform Interface constraint means that the interaction with the service is done through a finite set of requests which are generic. Resources of the service are manipulated via a Unique Resource Identifier (hereinafter URI), using HTTP verbs such as GET, POST, etc. (Reese 2012).

### 5.3.5 Layered System

Layered System constraint in its basic form allows the RESTful service to encapsulate any degree of complexity by representing an infinite set of interlinked sub-systems. This constraint helps to implement scalable services by allowing the use of various load balancers and distributed computing software. However, the complexity of such systems should not affect the client as his interaction with the service is restricted to the uniform interface. (Reese 2012; Fielding 2000).

## 6 REQUIREMENT ANALYSIS

### 6.1 Requirement Analysis Description

Requirement analysis is an initial starting point of any development project. According to Kotonya & Sommerville (1998, 3), requirement analysis is a process, which determines conditions to be met by a software project in order to satisfy its stakeholders. It plays a significant role in the success of the overall project.

At first, the stakeholders for this project are identified. After that, functional and non-functional requirements are gathered and analyzed. Finally, the outcomes of the requirement analysis are used to construct the sequence diagrams presented in the specification sections of the implementation chapters.

### 6.2 Stakeholder Analysis

The primary stakeholder for this project is the EMI company. The company is providing various educational and leadership services, consequently, they are in need for an e-commerce solution which would satisfy their current requirements and would allow them to distribute their products online. The potential users of the system are the merchants of the EMI company who need to manage the product stocks, prices and promotions, as well as, to do the inventory and content management.

This project involves project managers who are the direct stakeholders of it. Project managers are present on both sides – at Liana Technologies and the EMI company. Their main areas of concern are activities which facilitate the project development and ensure, that the project is executed according to the schedule.

The researcher is a primary internal stakeholder as he is directly involved in the project development. The researcher is responsible for the development and



delivery of the project according to the requirements as well as ensuring that the customer is satisfied with the result.

### 6.3 Functional Requirements

This project is intended to deliver a fully-functional e-commerce platform, the outcome should include all the basic features provided by the LianaCommerce platform plus the features identified by the requirements listed in this section. The custom functionality expected by the customer is described below.

According to the client, the web store built on the platform has to provide digital product licenses alongside the other products. Implementation of this feature requires integration of LianaCommerce platform with Avenue digital product system. A meeting has been initiated with the client and workflow for the desired web store has been identified. The desired workflow is as follows: web store user adds a product license to cart and creates order. When the order is paid, LianaCommerce platform requests a product link from the Avenue web service. When the link is obtained from Avenue system, the web store should send it to the user via email.

During the communication session, it also has been noted, that the LianaCommerce functionality should be extended with integration of Dynamics CRM system into its workflow. The workflow of LianaCommerce must be modified in such a way that every time the user creates an order, the order information is transferred to the Dynamics CRM with the data defined by the documentation for the Dynamics service of the EMI company.

### 6.4 Non-Functional Requirements

According to the functional requirements listed above, the technical part of this project is highly dependent on the communication between individual web services. This shifts the focus of the non-functional requirements mainly to the area of web service communication and availability.

Separate vendors provide the Avenue and Dynamics CRM web services; therefore, Liana Technologies cannot predict the availability times of these services. The downtime of service can result in data transfer failure or lead to an inconsistent data state and data loss. Thus, a mechanism for error handling and data consistency check should be considered during the implementation phase.

From a developer's perspective, during the technical implementation special attention must be paid to the code style and its compliance with the organization's technical guidelines. The code must be readable and easy to understand. This can be achieved by extensive usage of Liana framework functionality and the usage of design patterns. Additionally, the code should be self-documenting, e.g. variables should be given descriptive names and comments should be used where necessary.

Furthermore, it is crucial to ensure the stability of the platform after the new functionality has been added. This can be done by covering the new functionality with unit tests. If an implementation is using a network, the responses from the services should be mocked. This contributes to faster test execution and eliminates the test failure on service denial event.

Security is another topic to be considered during the implementation. All the data transfer during the service communication must be done with security in mind, therefore data should be encoded and stored according to the organization's security guidelines.

## 7 AVENUE WEB SERVICE INTEGRATION

### 7.1 Specification Analysis

Avenue plugin is a part of LianaCommerce platform, which serves as a connecting component between the platform and the Avenue web service. Technical specifications for implementation of this plugin derive from requirements gathered during the requirement analysis section. The outcome of the analysis is a sequence diagram provided in Figure 10 below. According to Bell (2004), sequence diagrams are helping to define the behavior of the future system as well as provide a refinement of the collected requirements. This diagram represents the requirements as an interaction of objects and serves as a technical reference throughout the plugin development process.

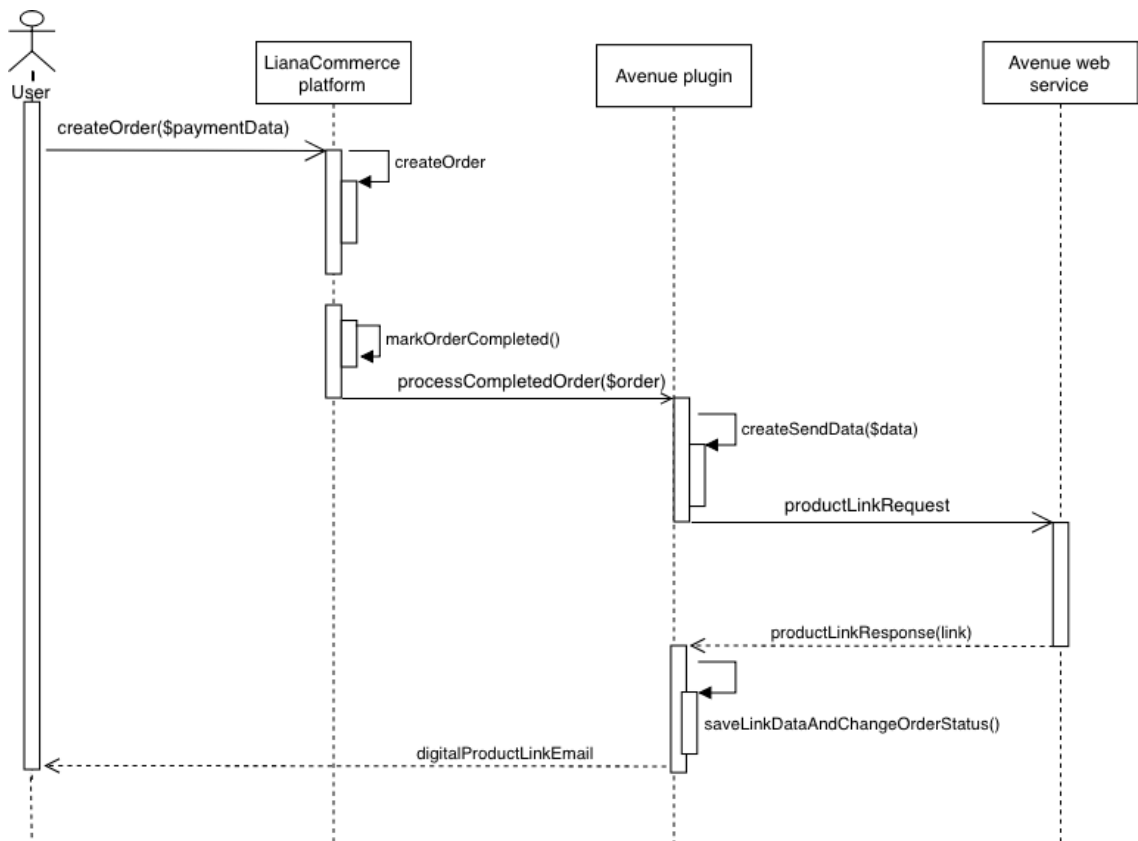


Figure 10. Avenue Plugin Sequence Diagram

According to the diagram above, the user is an event-trigger, who creates and pays an order. Once LianaCommerce platform has validated the payment data,

the order status is set to 'paid'. The status change event is tracked by LianaCommerce plugin system. The plugin system does a 'processCompletedOrder' function call to the Avenue plugin, providing an order object as an argument. The plugin does processing of the order-related data and performs a POST request to the Avenue web service. Once a response from the server is obtained, Avenue plugin performs a validation of the response payload. If the payload contains a valid JSON data with a URL inside, then the response is considered successful. The plugin saves the link to the database and sends an email with the digital product URL to the user. Finally, Avenue plugin marks the order as handled, by setting a success status to it. Alternatively, if an error occurs and the digital product URL is not obtained, the order is marked as failed by setting a 'failed' status.

## 7.2 Plugin Structure

In LianaCommerce, software plugins are managed by Liana framework's built-in hook system. Therefore, there exist several constraints which plugin should follow in order to function correctly. One such constraint is that all the plugins implementing an integration to external services should be located under Plugin2 directory. Therefore, all the Avenue plugin-related data is placed at the store/Plugin2/Avenue directory.

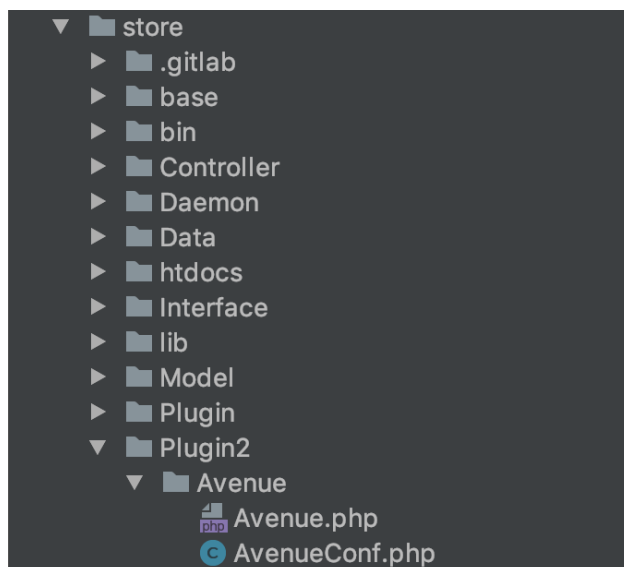
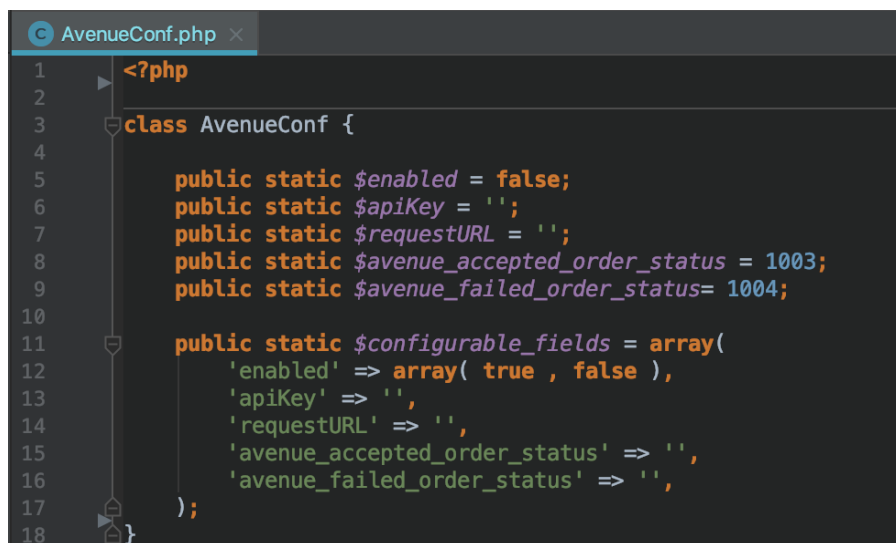


Figure 11. Avenue File Structure

As shown in Figure 11 above, Avenue folder contains two files: Avenue.php and AvenueConf.php. Avenue.php is the core file of Avenue plugin. It contains Avenue class with an implementation of the plugin's logic. Special attention has to be paid to the naming of the files, folders, and classes. Liana framework contains an autoloader script, which refreshes the class structure of a web store every time the store is updated. The class structure is providing a correct namespace mapping ensuring that functions used across several classes can be correctly used throughout the project.

AvenueConf.php is a configuration file, containing the plugin's settings. These settings are automatically imported by LianaCommerce's upgrade script, which is used by the LianaCommerce team to distribute the updates across the web stores. The upgrade script reads the configuration file and automatically inserts the default values for the settings to the database's plugin configuration table.

The image shows a code editor window titled 'AvenueConf.php'. The code is as follows:

```
1 <?php
2
3 class AvenueConf {
4
5     public static $enabled = false;
6     public static $apiKey = '';
7     public static $requestURL = '';
8     public static $avenue_accepted_order_status = 1003;
9     public static $avenue_failed_order_status= 1004;
10
11     public static $configurable_fields = array(
12         'enabled' => array( true , false ),
13         'apiKey' => '',
14         'requestURL' => '',
15         'avenue_accepted_order_status' => '',
16         'avenue_failed_order_status' => '',
17     );
18 }
```

Figure 12. Avenue Plugin Configuration

Figure 12 illustrates the structure of the Avenue plugin configuration file created according to the plugin's specification. Static fields at the beginning of the file are defining the default values for the settings, while the '\$configurable\_fields' array defines the fields which will be configurable by the admin user from the UI. Figure 13 shows the web store's plugin configuration interface. This interface has been automatically created by the LianaCommerce's upgrade script from the configuration provided in the AvenueConf.php file.

Avenue	Setting	Current value	Set value
enabled	Enabled	true	true <input type="checkbox"/>
apiKey	Avenue_Plugin2-apiKey	da8c7ddca23e47bac84a30e9e60cab4f	<input type="text" value="da8c7ddca23e47bac84a30e9e60cab4f"/>
requestURL	Avenue_Plugin2-requestURL	http://avenue.staging.fi/invitation-link-management/api/invitationlink	<input type="text" value="http://avenue.staging.fi/invitation-link-i"/>
avenue_accepted_order_sta	Avenue_Plugin2-avenue_accepted	1000	<input type="text" value="1000"/>
avenue_failed_order_statu	Avenue_Plugin2-avenue_failed_o	1001	<input type="text" value="1001"/>

Figure 13. Avenue Configuration Interface

LianaCommerce's module system supports feature toggle functionality. According to Hodgson (2017), feature toggle is a part of the continuous delivery process, allowing in-progress features to be merged to master branch while still keeping the branch production-stable. Feature toggle is a handy technique as it serves several purposes. Firstly, it allows to do the software versioning and keep the module's functionality isolated from the rest of the code in case that a specific client does not require the functionality. Secondly, feature toggle contributes to the stability of the web store during the development and testing of the plugin. The plugin can be enabled for a short time for the testing purposes and can be easily turned off after the testing is done. Thus, if the module contains any bugs, they will not affect the business-critical logic of the system if the module's feature toggle is turned off.

### 7.3 Service Integration

Avenue web service provides a RESTful API for communication. According to the service documentation, the interface consists of a single 'invitationlink' resource. Call to this resource is performed via a standard HTTP POST method. The service requires authentication; thus, the 'X-DigiAPV-Api-Key' authentication key has to be provided as an HTTP header.

```

1  {
2    "product": {
3      "id": "a61352bd-025f-4885-b0ad-1c12fe54b484",
4      "role": "",
5      "modules": [],
6      "tags": []
7    },
8    "employer": {
9      "name": "",
10     "address": {
11       "postalCode": "",
12       "country": "",
13       "region": "",
14       "city": "",
15       "street1": "",
16       "street2": "",
17       "street3": ""
18     }
19   },
20   "payer": false
21 }

```

Figure 14. Invitationlink Request Body

Avenue invitationlink resource is expecting a set of POST parameters to be contained inside of the request payload. The structure of the request body is illustrated in Figure 14 above. This request has a single mandatory parameter – id of the purchased digital product. This id corresponds to a digital product id inside of the Avenue system.

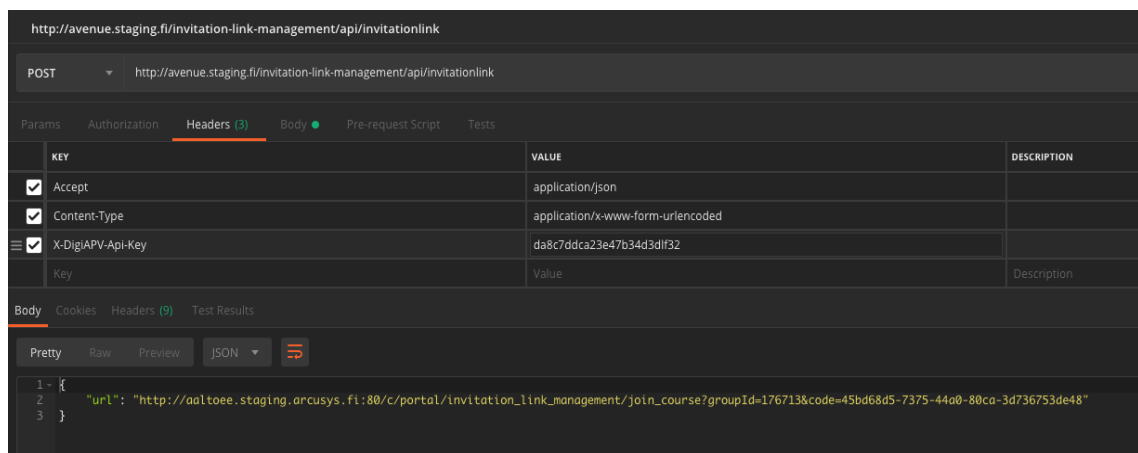


Figure 15. Invitationlink Postman Call

In order to proceed with the plugin implementation, a test call has to be done to the system. The data obtained through a test call serves as a validation of correct API utilization and serves as a good reference throughout the implementation. The test call is done using Postman software, which is a special tool for working with various APIs. According to the test call, Avenue service responds with a single parameter – the digital product URL.

```

1 <?php
2
3 $curl = curl_init();
4
5 curl_setopt_array($curl, array(
6     CURLOPT_URL => "http://avenue.staging.fi/invitation-link-management/api
7     /invitationlink",
8     CURLOPT_RETURNTRANSFER => true,
9     CURLOPT_ENCODING => "",
10    CURLOPT_MAXREDIRS => 10,
11    CURLOPT_TIMEOUT => 30,
12    CURLOPT_HTTP_VERSION => CURL_HTTP_VERSION_1_1,
13    CURLOPT_CUSTOMREQUEST => "POST",
14    CURLOPT_POSTFIELDS => "",
15    CURLOPT_HTTPHEADER => array(
16        "Accept: application/json",
17        "Content-Type: application/x-www-form-urlencoded",
18        "Postman-Token: d736c2c7-7b3d-4097-93f9-a9824c510ec5",
19        "X-DigiAPV-Api-Key: da8c7ddca23e47b34d3d1f32",
20        "cache-control: no-cache"
21    ),
22 );
23 $response = curl_exec($curl);
24 $err = curl_error($curl);
25
26 curl_close($curl);
27

```

Figure 16. Invitationlink Request Code

Postman can be useful not only for testing various APIs, but it can also significantly reduce the development time by automatically generating client code snippets. Figure 16 shows an auto-generated PHP code snippet, implemented with the use of PHP Client URL library (hereinafter cURL). This snippet is used inside of plugin's 'sendRequest' function with minor changes.

According to LianaCommerce's plugin system, every plugin requires function triggers to connect to the core logic of LianaCommerce. These triggers are provided by Liana framework and are called hooks. According to the plugin specification, the request to the web service should be made when the order is completed, and its status is changed. Figure 17 shows how Avenue hooks are configured inside of Hook.php file. Array keys 'order:order\_completed' and 'order:status\_change' are the names of event-triggers, defined at specific places of the Model/Order.php file. The 'from' element defines the name of the plugin, and the 'call' element defines the callback function to be called. When the trigger call is reached during the runtime, the framework checks the configuration inside of each sub-array one-by-one and if the plugin configuration flag is set to 'enabled', does a function call.



```

/** @var array PLUGIN2 hooks are defined here */
protected static $hookables2 = array(
    'order:order_completed' => array(
        array(
            'from' => 'Avenue',
            'call' => 'processCompletedOrder',
            'type' => '::',
        )
    ),
    'order:status_change' => array(
        array(
            'from' => 'Avenue',
            'call' => 'onStatusChange',
            'type' => '::',
        )
    )
);

```

Figure 17. Avenue Hook Configuration

When 'onStatusChange' and 'processCompletedOrder' functions are called, they are given an order object as a parameter. This object contains Avenue product id and other order-related data which is serialized as JSON string by 'createSendData' function. The service request is done through 'sendRequest' function, which contains cURL PHP implementation from the Postman API development environment.

#### Extra descriptions:







 avenue_code:	9dasdd20c-3fs6-4209-b6ab- ea82gfe3b8ac	
 2017-11-27 at 14:45	 avenue accepted	
Comments:	 Avenue link received successfully	
Order extra 3:	<a href="https://avenue.com:443/c/portal/invitation_link_management/join_course?groupId=1231232&amp;code=5231230e-9mde-4d30-af7e-65fb478df933">https://avenue.com:443/c/portal/invitation_link_management/join_course?groupId=1231232&amp;code=5231230e-9mde-4d30-af7e-65fb478df933</a>	

Figure 18. Digital Product Information

When the service responds with a correct URL, 'processCompletedOrder' function sets the response URL to "Order extra 3" field. After that, 'sendLinkToCustomer' function sends an email to the user, and 'avenue\_accepted\_order\_status' is set as a new status of the order. Figure 18 shows web store admin UI with a product URL saved to 'Order extra 3' field.

If an error occurs and no product link is returned from Avenue service, then an 'avenue\_failed\_order\_status' is set, and the store's administrator is notified with an email, containing verbose error data. When the problem is resolved, the additional request to the service can be triggered manually by changing the order status manually to 'avenue\_accepted\_order\_status'.

#### 7.4 Unit Testing

In order to ensure that the plugin performs as defined, unit testing is done. According to Fowler (2007), a unit is the smallest software part, which could be tested. Thus, unit testing as a method to ensure that separate units are performing according to their specification (Kolawa & Huizinga, 2007). Tests for avenue plugin are intended to verify that the system performs correctly given invalid and valid responses from the web service. For this purpose, a mocked-up JSON response is passed to 'processCompletedOrder' function. To assert that the values are correct a Liana PHP assertion library is used. Figure 19 below shows the results of the avenue unit test execution.

```
ok      1      64      | Cart created Order
ok      2      87      | category has id
ok      3     141     | Avenue has responded with invalid url, status set to avenue_failure
ok      4     146     | Avenue has responded with valid url, status set to avenue_success
ok      5     147     | Order_extra3 field is not empty
```

Figure 19. Unit Tests

## 8 DYNAMICS CRM INTEGRATION

### 8.1 Specification Analysis

Dynamics plugin is facilitating interaction between LianaCommerce platform and Dynamics CRM system. The initial set of requirements discussed in Chapter 3 has identified a need for a service downtime management mechanism to be considered within this implementation. The communication with the Dynamics service is done via SOAP protocol over HTTP. Dynamics plugin serves as an example of how a SOAP service can be integrated with an application. Together with Avenue plugin, it comprises EMI web store's custom business logic.

Dynamics CRM is an example of an application, which is designed for providing services to multiple software systems, assuming that the communication stability is crucial. However, during the implementation process, it has been identified that the service does not scale well – the responses tend to get stuck during the transfer as well as the service response time fluctuations are difficult to predict. Since Apache server configuration used at Liana's production server does not allow requests to last longer than 30 seconds, an asynchronous mechanism for data exchange has to be considered.

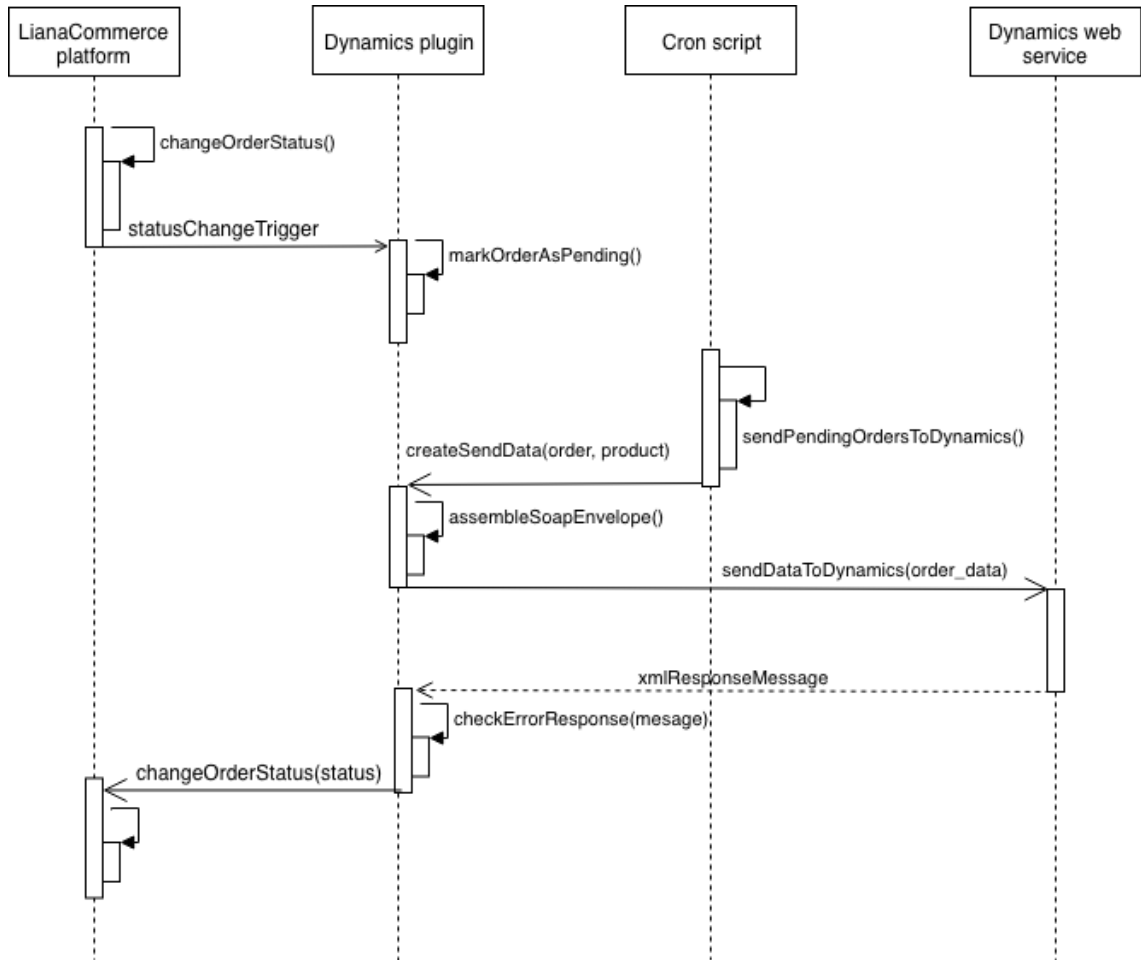


Figure 20. Dynamics Plugin Specification

As shown in Figure 20, Dynamics plugin is triggered on the order status change event. Plugin checks if the order has not yet been exported to Dynamics system and marks it as 'pending'. Cron script has been added to the workflow to eliminate service interruptions and guarantee that the data is transferred successfully. Cron is constantly monitoring the order changes in the background. Once an order has been marked for transfer, Cron performs a 'createSendData' call to the Dynamics plugin. The plugin serializes order-related data as a SOAP envelope and performs a call to the Dynamics web service. After the response is obtained, the plugin does a payload check and sets a new status to the order.

## 8.2 Service Integration

The structure of the Dynamics plugin shares the majority of structural aspects with the Avenue plugin, due to both of them being constructed according to the

guidelines of Liana framework. As can be seen from the Figure 21, the only difference between Avenue and Dynamics plugin file structures is the Dynamics.bin.php file. This file contains a PHP script which is managed by the Cron daemon.

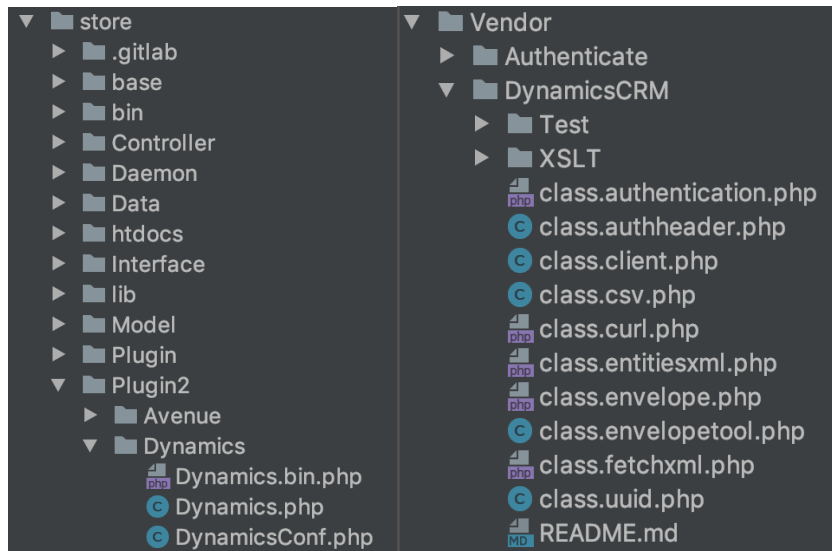


Figure 21. Dynamics File Structure

Plugin configuration file illustrated in Figure 22 contains configurable fields, used to connect to the Dynamics service. SOAP used in Dynamics is having several differences from the one described in Chapter 5. It does not provide WSDL or any other kinds of service description, which could be used directly by a SOAP client. Instead, WSDL is created dynamically by using a DynamicsCRM PHP library. This library provides a convenient API for an envelope creation and data passing. It is important to note, that Dynamics uses custom data structures, which have to be used carefully. The service is using static data types, and a single type conversion error would lead to the payload being rejected. Thus, special attention has to be paid to the typecasting of date, float, and double variables. It is especially important to remember because the implementation is done in PHP, which is a dynamically-typed language.

```

class DynamicsConf {
    public static $enabled = false;
    public static $username = '';
    public static $password = '';
    public static $endpointUrl = '';
    public static $entityName = '';
    public static $dynamics_success_order_status = Order::TYPE_PROCESSED;
    public static $dynamics_failed_order_status = Order::TYPE_ERROR;

    public static $configurable_fields = array(
        'enabled' => array(true, false),
        'username' => '',
        'password' => '',
        'endpointUrl' => '',
        'entityName' => '',
        'dynamics_success_order_status' => '',
        'dynamics_failed_order_status' => ''
    );
}

```

Figure 22. Dynamics Configuration

From the developer's perspective, Dynamics CRM can be seen as a remote database, which is accessed by sending SOAP envelopes to it. Like in any other database there are tables. In Dynamics CRM, the tables are called entities. Entities contain attributes, which are having data types and are functioning as columns. Service requests are done through an endpoint URL, which is defined in the plugin configuration panel. Every request should contain a name of a target entity with data attributes mapped to it. Figure 23 shows an example of a request body, containing 'fs\_firstname', 'fs\_lastname' and 'fs\_jobtitle' attributes of string type. The LogicalName tag below contains a target entity's name.

```

<s:Body xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <e:Create xmlns:e="http://schemas.microsoft.com/xrm/2011/Contracts/Services">
    <e:entity xmlns:b="http://schemas.microsoft.com/xrm/2011/Contracts">
      <b:Attributes xmlns:c="http://schemas.datacontract.org/2004/07/System.Collections.Generic">
        <b:KeyValuePairOfstringanyType>
          <c:key>fs_firstname</c:key>
          <c:value xmlns:d="http://www.w3.org/2001/XMLSchema" i:type="d:string">Artem</c:value>
        </b:KeyValuePairOfstringanyType>
        <b:KeyValuePairOfstringanyType>
          <c:key>fs_lastname</c:key>
          <c:value xmlns:d="http://www.w3.org/2001/XMLSchema" i:type="d:string">Klimenko</c:value>
        </b:KeyValuePairOfstringanyType>
        <b:KeyValuePairOfstringanyType>
          <c:key>fs_dateofbirth</c:key>
          <c:value xmlns:d="http://www.w3.org/2001/XMLSchema" i:type="d:dateTime">2888-01-14</c:value>
        </b:KeyValuePairOfstringanyType>
      </b:Attributes>
      <b:EntityState i:nil="true"></b:EntityState>
      <b:FormattedValues xmlns:c="http://schemas.datacontract.org/2004/07/System.Collections.Generic"></b:FormattedValues>
      <b:Id>00000000-0000-0000-0000-000000000000</b:Id>
      <b:KeyAttributes></b:KeyAttributes>
      <b:LogicalName>fs_dynamicentity</b:LogicalName>
      <b:RelatedEntities></b:RelatedEntities>
      <b:RowVersion i:nil="true"></b:RowVersion>
    </e:entity>
  </e:Create>
</s:Body>

```

Figure 23. Request Body Example

Dynamics service is using preemptive authentication, meaning that every service request should contain authentication data. This data consists of username and password fields which are defined in the Dynamics configuration file. According to the Dynamics plugin specification, it should transfer the order data to Dynamics when status is set to 'paid'. To implement this, 'onStatusChange' function is created inside of the plugin. Figure 24 illustrates the 'onStatusChange' function. This function performs a status check and marks the order as pending by adding "Dynamics pending" string into order's "notes" field.

```
public static function onStatusChange($hook_data) {
    if (DynamicsConf::$enabled !== true || DynamicsConf::$entityName === '' || DynamicsConf::$endpointUrl === '' || DynamicsConf::$username === '') {
        return null;
    }
    /** @var Order $order */
    $order = $hook_data['order'];
    if ($order->notes === 'Data has been sent to Dynamics' || ($order->status !== Order::TYPE_PAID)) {
        return null;
    }
    $order->notes = 'Dynamics pending';
    $order->save();
}
```

Figure 24. Dynamics onStatusChange Function

To connect the plugin to the core logic of LianaCommerce, hook mapping is done inside of Hooks.php model. Dynamics plugin has two hooks assigned to it - 'order:status\_change' and 'order:order\_completed'. When the hooks are triggered, 'onStatusChange' static function is called.

According to non-functional requirements, connection interruption, which might occur during the message transfer should be prevented. This requirement can be addressed by delegating the message transfer to a daemon. Daemon is a background process which is listening for a specific event to occur, rather than being triggered by the user (The Linux Information Project, 2005). Linux systems have a built-in daemon named Cron. Cron can execute the desired script according to a schedule. Cron schedules are configured inside of Crontab files, which contain a definition of which script should be running at which point of time.

Dynamics.bin.php is a Cron script, which is constantly checking if there are any orders marked as "Dynamics pending". When such an order is found, the script tries to transfer it to the Dynamics service. The script itself does not handle

service connections. Instead, it triggers a 'createSendData' function inside of Dynamics.php plugin file. This function serializes order data into a JSON string and passes the string to 'sendDataToDynamics' function which converts JSON data into an XML envelope and makes a request to the Dynamics service.

When a successful response is obtained, the Cron script marks the order as transferred and sets it a "Dynamics success" status. In case any error has occurred during the data transfer, the Cron will try to send the order during the next 24 hours and if the error persists, it will mark the order as failed. Also, on failed order status the error is written into a log file and the web store administrator is notified via email.



## 9 CONCLUSION

This thesis work has achieved all the desired objectives by implementing system integration of the LianaCommerce platform with external services. During this research, the theoretical concept of system integration has been studied with an example of point-to-point integration. The obtained theoretical knowledge has been tested against the real-world case during the implementation phase, extending the theoretical understanding with practical knowledge. The stability of the integration has been verified by covering the implementation with unit tests and ensuring the successful data transfer with a Cron script. In addition, the researcher has extended his knowledge on web services and their architectures, improved his professional skills by following code style guidelines and utilizing various software architecture patterns.

The customer case has been fulfilled by delivering the client a working e-commerce platform with integrated logic provided by external services. The platform has been deployed to a production environment and has been used as a web store. The client has given positive feedback on the provided e-commerce solution.

The integration done during this research has proven itself to be sufficient in the existing context; however, it does not support scalability. This fact provides a fertile environment for future studies. One example of such study could be to consider implementation of an integration service, which would serve as middleware between LianaCommerce platform and other systems. This type of middleware could facilitate changes and allow system integrations to be developed with less manual input required from the developer's side.

## BIBLIOGRAPHY

Alemu, M., B. 2014. REST API Implementation with Flask-Python. Lapland University of Applied Sciences. Faculty of Communication, Transport and Technology. Bachelor's Thesis.

Bell, D. 2004. UML basics: The sequence diagram. IBM developer. Accessed 22 December 2018.  
<https://www.ibm.com/developerworks/rational/library/3101.html>.

Chen, M., Chen, A., K. & Shao, B., M. 2003. The Implications and Impacts of Web Services to E-Commerce Research and Practices. *Journal of Electronic Commerce Research*, 128-129.  
<https://pdfs.semanticscholar.org/698e/fa2bfd0afa2cd5d70de80c355fd720f9ce07.pdf>.

Christensson, P. 2006. SOAP Definition. Accessed 12 December 2018  
<https://techterms.com>.

Christensson, P. 2016. API Definition. Accessed 13 December 2018  
<https://techterms.com>.

Fowler, M. 2007. Mocks aren't Stubs. Accessed 4 January 2019.  
<https://martinfowler.com/articles/mocksArentStubs.html>.

Fielding, R., T. 2000. Architectural Styles and the Design of Network-based Software Architectures. University of California, Irvine.  
<https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>

Hasselbring, W. 2000. Information system integration. *Communications of the ACM*. Volume 43, Issue 6, 32-38.

Hodgson, P. 2017. Feature Toggles. Accessed 14 January 2019.  
<https://martinfowler.com/articles/feature-toggles.html>.

IBM Knowledge Center. 2018. The structure of a SOAP message. Accessed 16 December 2018.  
[https://www.ibm.com/support/knowledgecenter/en/SSMKHH\\_10.0.0/com.ibm.etools.mft.doc/ac55780\\_.htm](https://www.ibm.com/support/knowledgecenter/en/SSMKHH_10.0.0/com.ibm.etools.mft.doc/ac55780_.htm).

Kolawa, A. & Huizinga, D. 2007. Automated Defect Prevention: Best Practices in Software Management. Wiley-IEEE Computer Society Press.

Kotonya, G. & Sommerville, I. 1998. Requirements Engineering: Processes and Techniques. Chichester, UK: John Wiley and Sons.

Lassenius, C., Soininen, T. & Vanhanen, J. 2001. Constructive Research Methodology workshop. Helsinki University of Technology.

Linthicum, D. 2001. B2B Application Integration. Boston: Addison-Wesley.

Oracle NetSuite. 2018. What is an ecommerce platform? Accessed 30 October 2018. <http://www.netsuite.com/portal/resource/articles/ecommerce/what-is-an-ecommerce-platform.shtml>.

Piirainen, K., A. & Gonzalez, R., A. 2014. Constructive Synergy in Design Science Research: A Comparative Analysis of Design Science Research and the Constructive Research Approach. Nordic Journal of Business. No. 3-4, 206-234.

Pressman, R., S. & Maxim, B., R. 2015. Software Engineering: A Practitioner's Approach. 8<sup>th</sup> Edition. McGraw-Hill Education.

Reese, G. 2012. The REST API Design Handbook. 1st Edition. Amazon Digital Services LLC. EBook.

Richards, M. 2015. Software Architecture Patterns. 1st Edition. O'Reilly Media, Inc.

Shaw, M. & Clements, P. 1997. A field guide to boxology: Preliminary classification of architectural styles for software systems. Twenty-First Annual International Computer Software and Applications Conference, 10-12.

The Linux Information Project. 2005. Daemon definition. Accessed 17 January 2019. <http://www.linfo.org/daemon.html>.

Viitala, K. 2017. Integration Architecture Development for Pori Energia. Satakunta University of Applied Sciences. Master's thesis.

W3C 2001. Web Services Description Language (WSDL) 1.1. W3C Note 15 March 2001. Accessed 3 January 2019. <https://www.w3.org/TR/2001/NOTE-wsdl-20010315>.

W3C 2004. Web Services Architecture. Working Group Note, 11 February 2004. Accessed 11 November 2018 <https://www.w3.org/TR/ws-arch/>.

W3C 2007. SOAP Version 1.2 Part 1: Messaging Framework. 2<sup>nd</sup> Edition. W3C Recommendation 27 April 2007. Accessed 3 January 2019. <https://www.w3.org/TR/soap12-part1>.

WordPress. 2018. Plugin handbook: Hooks. Accessed 27 December 2018. <https://developer.wordpress.org/plugins/hooks>.