EXTENDING POSSIBILITIES OF DEVELOPERS IN LARGE BUSINESS APPLICATIONS BY INTEGRATING VAADIN FRAMEWORK

Michał Szczygieł

Bachelor's Thesis
May 2013

Degree Programme in Information Technology

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

DESCRIPTION

Author	Type of publication	Date
SZCZYGIEŁ Michał Piotr	Bachelor´s Thesis	15/05/2013
	Pages 48	Language English
	Confidential ( ) Until	Permission for web publication (X)

Title
EXTENDING POSSIBILITIES OF DEVELOPERS IN LARGE BUSINESS APPLICATIONS BY INTEGRATING VAADIN FRAMEWORK

Degree Programme
Information Technology

Tutor
PELTOMÄKI Juha

Assigned by
Descom Oy

Abstract
The purpose of this bachelor’s thesis was to introduce Vaadin framework to for an existing Java EE project incorporating Spring MVC technology. This thesis was connected with a 4 month internship at Descom company, during which the developer team decided to introduce the said framework.

The aim of this thesis was to facilitate the creation of new features and, at the same time, save developers' effort. The introduction of this framework is expected to accelerate application development while maintaining high quality of both source code and functionality accessible to end-users.

This thesis provides information on both the theoretical for the entire project, as well as the practical part, which allows developers to carry out such an integration. The examples presented in this thesis are issues that affect deep layers of the application. However, the whole architecture of the project and the integration process which has taken place in the project could not be presented due to confidentiality of corporate data.

In the conclusion chapter is presented prove the usefulness and benefits of integrating Vaadin framework inside large Java EE based business applications.

Keywords
Vaadin, Integration, Spring, JSP, Java EE

Miscellaneous

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

Nowadays, more and more important for corporation businesses are solutions in investment areas with which customers seek competitiveness and profitability. Such solutions often require different sectors of activities. These sectors perform in various areas of their services and work in a variety of methodologies. The whole complex system in a company always requires integration between sectors. Integration is an inseparable part of many business processes, as well as is required many more abstract things. Also, this thesis focuses on the issue of integration.

The major importance of business solutions among others are web applications. Why web applications, not desktop applications? Over the last few years, there is no difference between the limitations characterized in web applications and desktop. The advantages of web application are their ability to be updated and maintained without distributing and installing software on every customer's computer. Inherent support for cross-platform is the main advantage of choosing this technology.

Implementing a web-based solution, even though there is a wide choice of development tools they often have to be limited to the constraints imposed by a customer. Often, many of these technologies have to use a few, however, that requires integration. Sometimes the integration of these tools can be made at various stages of application development, which is often problematic. That is the main motivation for the choice of the subject of this thesis: to show how to make integration for web development tool; what kind of benefits this integration brings into a project. The experience gained during the internship can touch the heart of the problem.

The main objective was to integrate the Vaadin inside an existing large web application, thereby extending the capabilities of developers. This procedure required extensive integration of both the presentation layer and the deeper-lying layers of the business. This junction allows developers to reduce the cost of application creation dedicated to the reduction of the time.

2 THEORETICAL BASIS

2.1 Overview

In this part of the thesis there is necessary information to understand the essence and structure of integration. All of these items create a single unit of the application.

2.2 What is a business web application?

2.3 Java 5 EE and why this old one

2.4 WebSphere Server Application v 7.0 introduction

2.5 About frameworks used in the application

2.6 JSP technology

2.7 Enterprise JavaBeans

The sub-chapter called “What is a business web application?” explain basic principles of business applications.

The next sub-chapter titled “Java 5 EE and why this old one” shows the concept of the platform, and explains why this technology has been used at the project.

In the following sub-chapter called “WebSphere Server Application v 7.0 introduction” is presented using technology provided by IBM.

"About frameworks used in the application" sub-chapter contains a frameworks description used in the project such as Spring, Hibernate, Vaadin.

The following chapter titled “JSP technology” provides a brief introduction to this technology.

Finally, in the last chapter called "Enterprise JavaBeans" shows architecture for modular construction of enterprise applications.

2.2 What is a business web application?

There is no strict definition of explaining what is a business web application and what standards shall be fulfilled. A business web application is nothing more than a part of enterprise software.

Enterprise application software is the foundation for large corporations, and often this application is an integral part of all departments in the company.

Also, this software aims to improve solving the main problems for industrial companies by providing business logic support functionality for this product.

> "Enterprise applications are about the display, manipulation, and storage of large amounts of often complex data and the support or automation of business processes with that data."
> (Fowler, Patterns of Enterprise Application Architecture, 2002).

This topic is very broad, therefore the remainder will refer to the project where the integration took place. Services provided by the project are oriented to enterprise application integration. The most accurate term for the operation and idea of the project is the term supply chain management, which describes solutions serving the company to manage the supply chain network. With this it is possible to synchronize the flow of materials between the cooperating parties. Also, this project is based on Java EE 1.5 and the products offered by IBM like WebSphere Application Server 7 (WAS7).

2.2 Java 5 EE and why this old one was chosen

In the world of information technology, time of creating and designing new enterprise applications is expected to cost less money, and to be produced faster. Thus engineers have created a platform such as Java EE, to facilitate the work.

The Java EE platform provides a great deal of advantages for enterprise:

- Establishes standards for enterprise computing which use database connection, enterprise business components, web-related components, message-oriented middleware, communication protocol and interoperability.
- Uses open standards.
- Irrespective of the infrastructure provided by vendors' products, the time to market is reduced by implementation in Java EE standards.
- Promotes a standard platform for developing software components which are portable in vendor implementations.
- Java EE technology is based on Java language, which allows developers to learn fast this technology.
- Java EE provides inter-operate within existing heterogeneous environments

(Alur, Crupi & Malks, 2003, 8)

The Java Enterprise Edition 5 (Java EE 5 also called Java EE 1.5) is server-based platform to write software in Java language. Java EE is an extension to the Java SE and it provides a powerful API for running and developing enterprise software. Attributes that characterize this technology are that is: scalable, transactional, portable; it has security, and reliable server-side applications. In order to reduce the effort of developers annotations support was introduced, XML deployment descriptors are optional. Information about annotations contained in Java source files, is automatic executed by Java EE server during deployment and runtime.

Java EE 5 also provides dependency injection which can be applied to all resources. This approach effectively hides the creation and lookup of required resources, thus saving a developer effort on writing boilerplate code. Dependency injection is also used in EJB containers, web containers and application clients. After injection Java EE container automatically combines references to other needed components or resources where annotations were used.

The JPA (Java Persistence API) is new in Java EE 5. The Java Persistence API allow for Object-Relational Mapping (ORM) to manage relational data in enterprise beans.

FIGURE 1 shows components of Java EE running on a single machine.

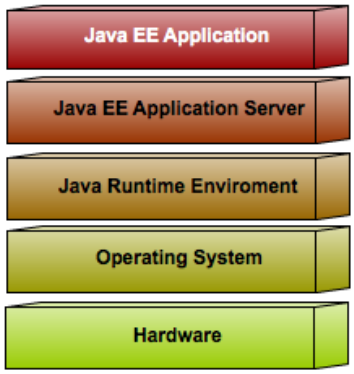


FIGURE 1: Java EE application execution model

The Java EE platform based is multitier application model, where the logic of application is divided into components. Each part is installed on a different machine.

FIGURE 2 shows this model divided into the tiers.

- Component (client applications and applets) with client tier is running on the client machine.
- Component (Java Servlet, JavaServer Pages and JavaServer Faces) with web tier is running on the Java EE server.

- Component (Enterprise JavaBeans) with business tier is running on one the Java EE servers.

- Component Enterprise information system tier is running on the EIS server.

(Jendrock, Ball, Carson, Evans, Fordin & Haase, The Java EE 5 Tutorial, 2007)

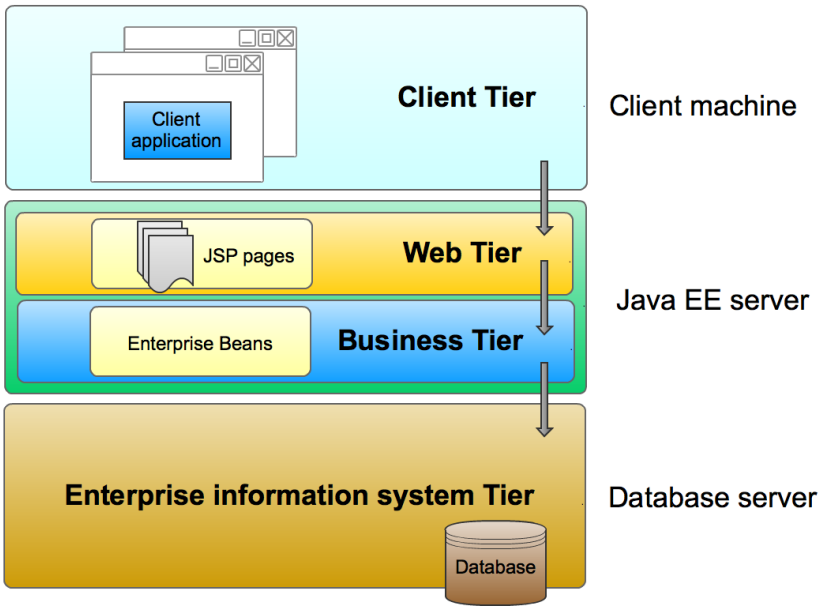


FIGURE 2: Multitiered applications

It could be asked “why is this old Java EE 5 used to this project?”.
As of today, the latest version of Java is Java EE 1.7, which offers a wealth of tools. The answer is very easy; the use of this technology is closely dependent on WebSphere Server Application architecture. This WebSphere Server Application in version 7 does not support a higher version of Java EE than 1.5. More about this technology is described in the following subchapter.

2.3 WebSphere Server Application v 7.0 introduction

A necessary component for software developers in the world of web business applications are servers on which the developed applications are installed.
Very popular solutions are offered by IBM in their products. One of them is the WebSphere Application Server. This is a cross-platform family, which can runs on from laptops up to the largest mainframe computer. This distributed platform which is based on single process model and also contains the Java Virtual Machines (JVMs) provides a powerful application server to deploy. The base logical application servers are executing on multiple JVMs, and each is executing in different address spaces, which are called Servant Regions (SR).In FIGURE 3 is presented the concept of logical architecture.

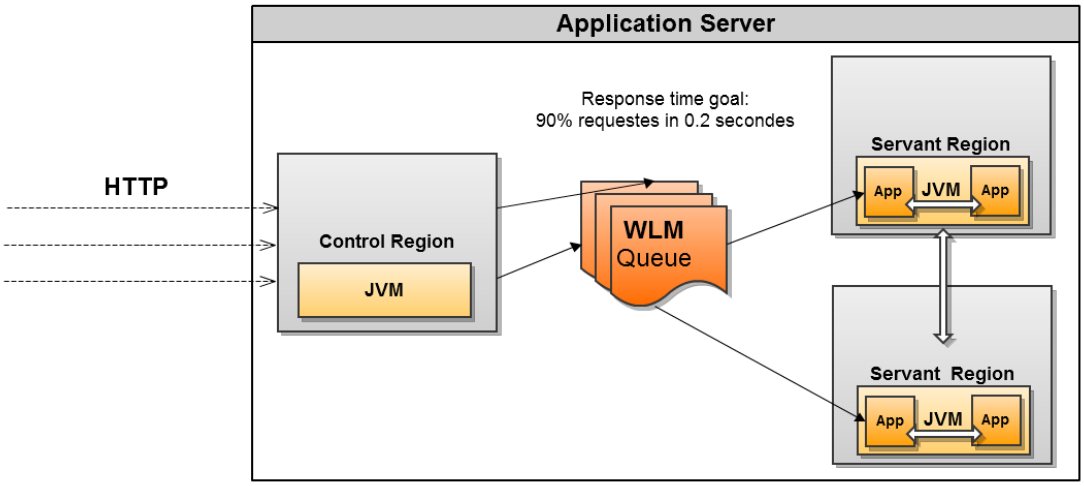


FIGURE 3: Architecture of application server

The basic concept of this architecture consists of one control region on one application server. The Servant Regions are statically defining and also there exists a possibility of adding new Servant Region. The limits of amount Servant regions are defined by available physical memory on the system. The main task of the control region is to take responsibility for incoming connections, which are dispatching to the request in Workload Manager (WLM) queues using their own JVM. The WLM queue stores information for further processing. This WLM uses FIFO (first-in-first-out) mechanism to represent a service class.

In the next step, the appropriate information goes to Servant Region, and also one Servant Region can serve to be one service class. Therefore, the Servant Region is a component of the application server. Here the application runs and other issues like transactions, EJB and Web container are processed in the instance of a JVM. (Sadtler et al. 2009)

2.4 About frameworks used in the application

This sub-chapter contains information about all used frameworks in the project. Each framework has its different purpose. Often, these all have similar requirements and the tasks to be solved become easier. The time saved by developers by the use of the framework makes the cost of creating applications lower because functions that would normally have to be implemented by a programmer are already created within a framework in this project among others the used open source frameworks are such as:

1. Spring

FIGURE 4: Spring logo

2. Vaadin

FIGURE 5: Vaadin logo

3. Hibernate

FIGURE 6: Hibernate logo

2.4.1 Spring

This framework created by Rod Johnson is open-source. The main objective of the creation of this framework was to facilitate the creation of enterprise applications. Spring is often referred to as a lightweight framework for building enterprise applications. The package for entire Spring framework is arranged in a single JAR file, whose the size of which does not exceed 1 MB. Inversion of Control (IoC) is the basis for the core of the framework. This technique manages objects in such a way that their dependencies are given passively instead of looking for or creating dependencies for objects. This behaviour is called Dependency Injection, because injection of dependencies is done at runtime.

(Harrop & Machacek, Pro Spring, 2005)

The structure of the Spring framework is divided in to seven well-defined modules. In FIGURE 7 are presented the modules which give everything needed to develop enterprise applications. Each module is an independent tool.

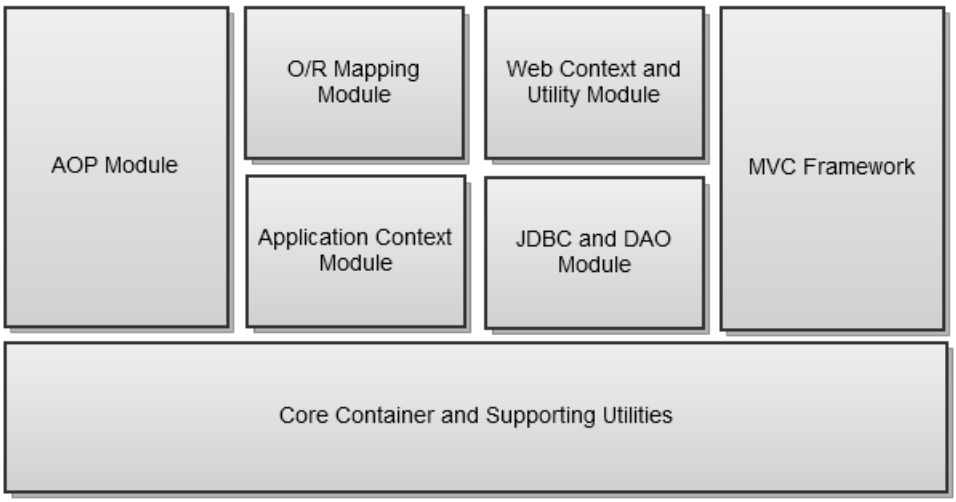


FIGURE 7: The Spring framework modules

All Spring modules are built over the core container. This container defines a set of beans which creates, manages and configures and provides the fundamental functionality. Here is localized `BeanFactory`, a pattern that applies IoC.

Spring's AOP module supports aspect-oriented programming. This module serves as a basis, but also provides support from other AOP frameworks. The Spring AOP module also allows using annotations to its own code source instructions.

Object/relational mapping (ORM) module in Spring framework provides hooks into ORM solution of other frameworks such as Hibernate, iBATIS SQL Maps and JDO.

Application context module extends the concept of `BeanFactory`. This module supplies enterprise services, JNDI access, EJB integration, remoting, email and scheduling. Among other things, this module provides a support for internationalization of application life-cycle events, messages and validation.

Web Context and Utility module provide a proper context for web-based applications and support web-oriented tasks such as among others programmatic binding of request parameters and transparently handling multi-part request.

JDBC abstraction and the DAO module simplify the process of writing the code related to business logic, bypassing the process of creating a great deal of boilerplate code. That module also supports display of database errors, as understandable exception. In addition, it provides transaction for management services.

Spring framework also contains a module which provides full-featured MVC (Model-View-Controller) template. Developers receive a high level of control over the template via the interfaces of the strategy. Also, this module can be easily integrated with other MVC frameworks.

(Walls & Breidenbach, Spring in action, 2005)

Spring MVC architecture could be divided into a series of layers. These layers separate the basic functionality of the application from the user interface to the persistence. All other layers are dependent on the Domain Model. MVC basically contains five layers of abstraction (see FIGURE 8):

- user interface
- web
- service
- domain object model
- persistence

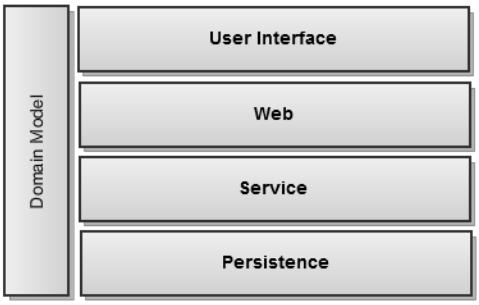


FIGURE 8: Spring MVC application layers

The main task of the user interface layer (often referred to as the View) is to present a front-end for an application. That part renders a generated response as a result of response to client requests. The concept consists of the fact in the entire chain processing that in this layer the final result for rendering is transferred to the client as bytes. The reason why this layer is separated from others is because the system can process other request with valuable information such as database connection.

Web layer is a layer that fulfils two basic functions. One of them is navigation logic, and the other one is to provide and ensure consistency between the service layer and HTTP request. Navigation logic provides simple mapping of a single URL to a single page. Responsibility of this layer means to move the user to the correct page view, maintaining the correct sequence. Web layer also takes responsibility for forwarding business exceptions such as the error message for the end user. Spring MVC also contains the work flow for processing requests which extend the handled request. Spring MVC for this layer provides a rich library with Controller interface, and it has a very complex solution where there is a possibility to use work flow.

The service layer provides access to the methods in stateless manner, also it provides a coarse-grained interface to use for system interactions. Methods represented in the service layer are a part of transactional units of work, where the methods executing many instructions are performed under a single transaction.

The domain object model contains the most important part of the layered Spring MVC

architecture which is the business logic. Business logic is centralized inside POJOs (Plain Old Java Objects), which make it possible to use polymorphism and inheritance. Spring can also enhance the domain model using AOP.

The data access layer takes responsibility for persistence mechanisms, in order to obtain and store data into database. The detached this layers from other has to simplify the management of data held by the mechanisms in this layer. Spring framework provides support for other toolkits like Hibernate, JDBC and IBATIS for all data access operations.

This isolated architecture of modules is aimed to increase testability and reduce coupling. Each layer can be easily tested in isolation. Being a separate business logic from the view and the transaction layer it allows to focus on each issue separately, while maintaining the structure of object-oriented model.

(Ladd with Davison, Devijver & Yates, Expert Spring MVC and Web Flow, 2006)

2.4.2 Vaadin

In Finnish folklore culture, Vaadin is a mythological creature. This animal spirit was part of shamanistic trance. More about this historical creature is explained in the Finnish epic poem “Kalevala”. Anyway, Finnish developers introduced this name to this project. The story of Vaadin framework started in 2000 in IT Mill. They had a desire to create new programming paradigm. The first application was developed in 2001, which supported the creation of user interface. The library was called Millstone Library. The next release of this library took place in 2006 which offered a new AJAX-based engine. At the end of 2007 IT Mill Toolkit 5 was released. In this edition the user interface was rewritten using Google Web Toolkit. This step allowed for developers the use of server-side and client-side. The next opportunity for developers was released in this framework under the Apache License 2. At the beginning of 2009 the next version called Vaadin was released. Also, IT Mill was transformed to Vaadin Ltd. In the next year the community of Vaadin boosted very fast. The version of Vaadin 7 which is already used in the current project, has changed very much from its predecessor. This version is more concentrated on aims like web-orientedness, stability and performance particularly in the case of Internet Explorer.

Vaadin is an open source framework for Java web application development. Vaadin technology supports server-side and client-side programming models. The client-side is located in the view layer and can be extended by using GWT (Google Web Toolkit). Using this powerful and rich framework allows to concentrate on the application logic without wasting time on graphic layout. The server-side takes care of communications between the browser and the server. FIGURE 9 present this communications.

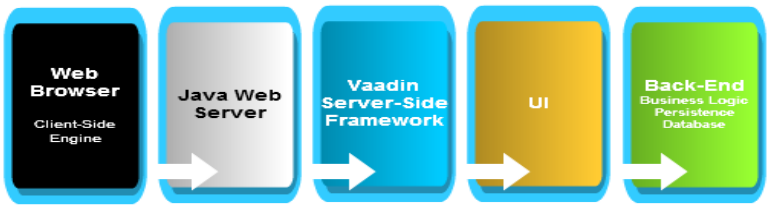


FIGURE 9: Vaadin application architecture

The client-side is executed as pure JavaScript, and consequently, other plugins such as Flash are not needed. Interaction with user interface to the server is done by the low level Java-based web server, where all business logic is located. Vaadin implicitly uses technologies such as AJAX (Asynchronous JavaScript and XML), GWT, CSS (Cascading Style Sheets) and SaSS (Syntactically Awesome Stylesheets). Vaadin framework has a structure of separated modules, which allows for separate development of each.

The user interface is rendered by Vaadin Client-Side Engine. Definitions of rendered widgets are located in components on the server-side. FIGURE 10 shows the flow between the client-side and server-side.

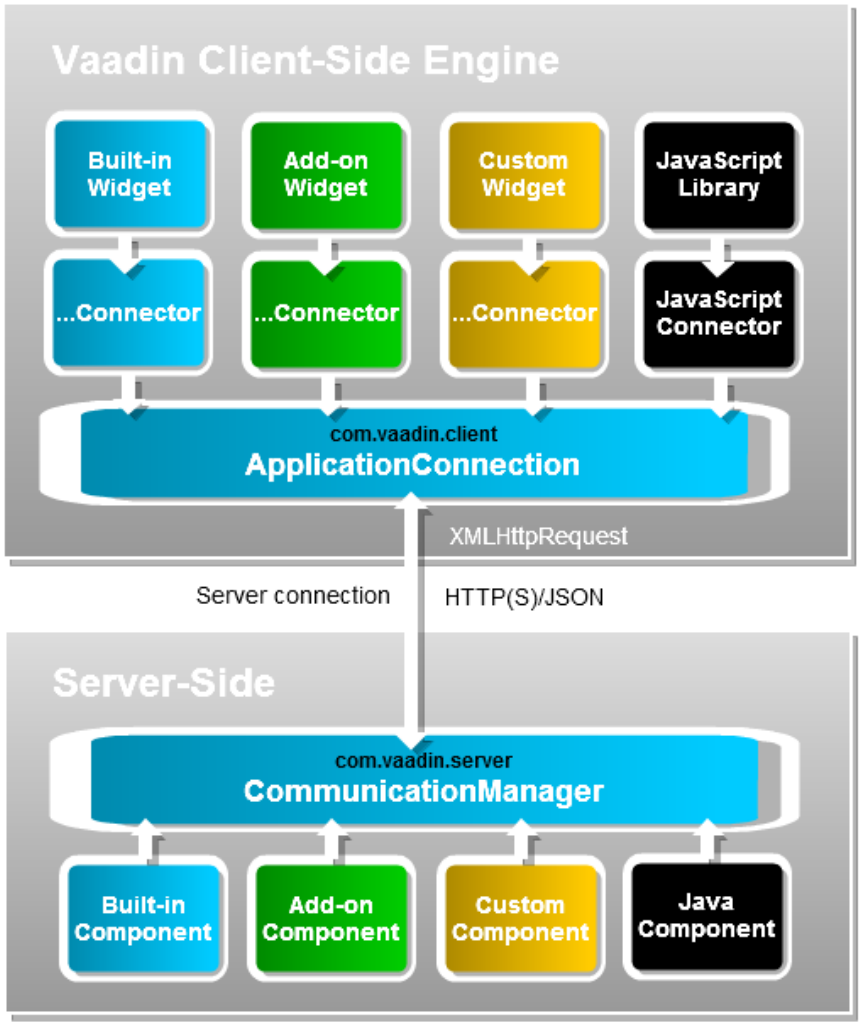


FIGURE 10: Vaadin Client-Side engine

In Vaadin Client-Side the framework exists of two different kinds of built-in widgets. These are Vaadin widgets and the others are GWT widgets. Both communicate with the server-side using `ApplicationConnection`. The client-side can easily be extended by new widgets written in Java. Between the two sides the serialization of component occurs transparently, and it also includes the RPC (Remote Procedure Call) mechanism.

Vaadin Server-Side framework is located on the Java servlet side. Code can also be ran on the portlet. Vaadin framework offers rich API for developing user interfaces. Components communicate transparently with other widgets on the client-side. The components of the user interface are implemented by Vaadin UI class. The components also contain event listeners in which it is possible to bind this component directly to data. The layout of the application can be defined by CSS or SCSS, by defining annotations `@Theme` in the application extends UI class.

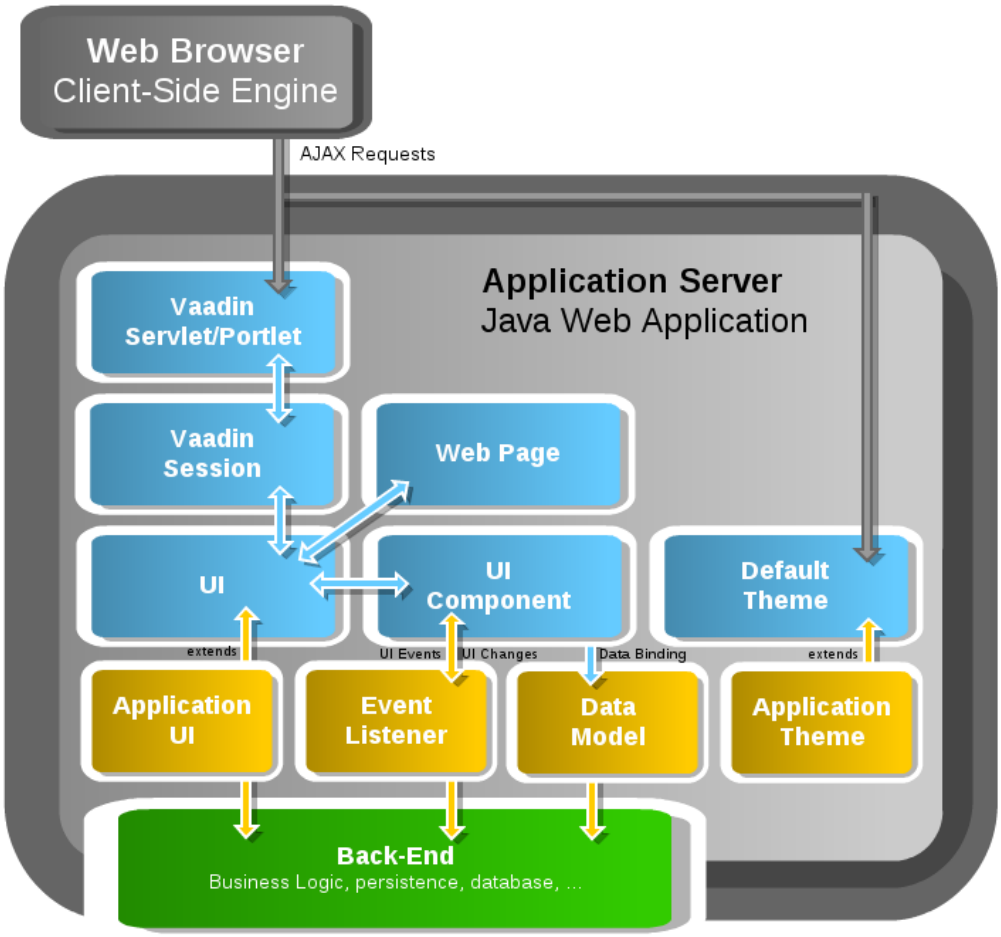


FIGURE 11: Server-Side application architecture

FIGURE 11 presents the server-side architecture with basic elements:

- UI – is the abstract class, which should be extended by at least one class application. Also, this extended class should override the `init()` method. UI is a kind of a bridge between the web page and part of the application logic. This represents a part of the HTML that works on the application side in web page. UI is deployed as a Servlet Container as a part of the a Java Servlet. Basically, this part is a viewport, connected to a user session, which can be associated with other windows.

- Web Page – this object is associated with UI. This part represents the web page and browser windows. The page object can be easily accessed globally from the application by using `Page.getCurrent()` method.

- Vaadin Session – object stores the session of a current user in the application. The session starts when the first UI of Vaadin application is initialized. The end of the session occurs when the session expires or is completed.

- User Interface Components – are created under the instance of an application. These components are hierarchically laid out in the structure where layout root on the top of hierarchy is contained. User interaction causes events that are captured by these components.

- Events and Listeners – are interfaces whose mechanisms allow to register the events.

- Themes – are section that is responsible for presenting what has been defined in the CSS or the SCSS. This presentation part is separated from the logic of application.

- Data Binding – is a process in which the values are bound directly with the component. Items such as tables or lists can be easily bound with data source collected in a container.

- Resources – are additional items that can be displayed by the user interface, such as images are content for download. All these items can, be external or internal, because they are handled as resources by Vaadin. In FIGURE 12 are presented two interfaces: `Resource` and `ConnectorResource` which are provided by a servlet.

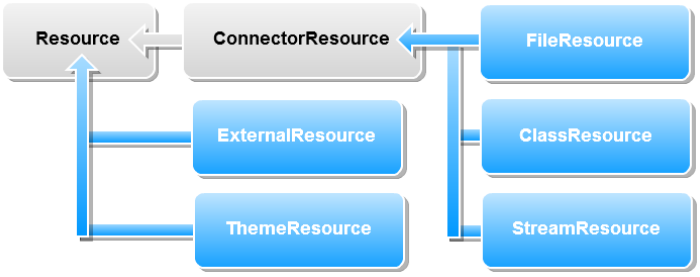


FIGURE 12: Resource interface

Vaadin framework is based on the most common three-layer architecture:

- User Interface (presentation) layer
- Domain layer
- Data store layer

Like any similar technology, it is characterized by the same features; in the presentation layer located on the top of architecture is the end-view for user, in the domain model is the defined business logic of application. Vaadin framework provides a direct bind user interface with the data source. The domain logic of enterprise solutions used by Java EE and Enterprise JavaBeans is located in the Domain Layer. Data from this layer are persisted by containers which are bound with Data Access Layer. FIGURE 13 shows the concept of this architecture.

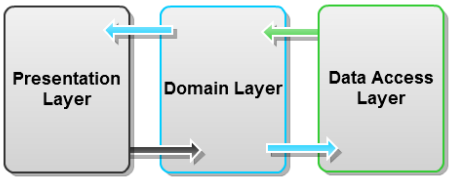


FIGURE 13: Three-layer architecture

(Grönroos, Book of Vaadin, 2013)

2.4.3 Hibernate

Hibernate is a framework for the Java language which provides transparent persistence for POJOs, however, to understand what Hibernate is one needs to know what persistence is.

Persistence is a fundamental concept for developing an application which uses an infrastructure layer. Persistence in Java stores data in a relational database, beyond the scope of the JVM and it can also be re-created at a later time.

Hibernate is a project that focuses on solutions for the problem of managing persistence of data. This facility allows that developers do not need to care about the business logic. This framework provides mapping for an object-oriented domain model to a relational database. Problems with object/relational paradigm mismatch are solved in that framework.

The applications architecture using Hibernate in business layer can be divided into two layers, Business Layer and Persistence Layer. In FIGURE 14 is presented the concept of architecture with the use of Hibernate in an application.

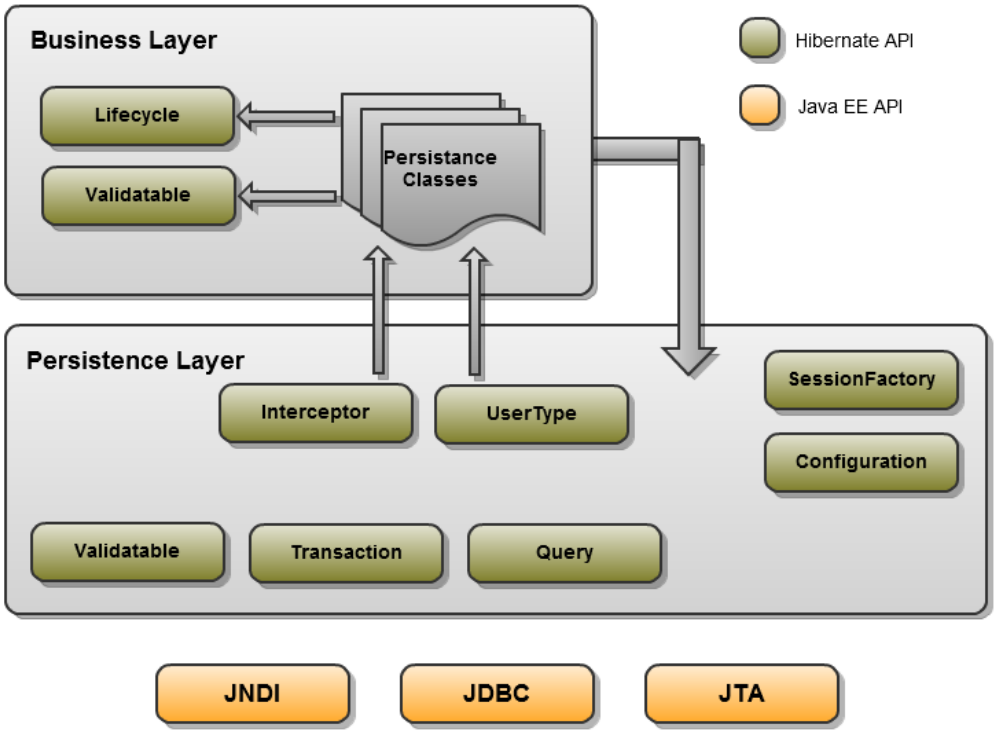


FIGURE 14: Application architecture with Hibernate API

CRUD (Create, read, update and delete) operations are performed by interfaces which include `Session`, `Transaction` and `Query`.

Application infrastructure interfaces contain the `Configuration` class, which includes the Hibernate configuration.

`Interceptor`, `Validatable` and `Lifecycle` so called Callback interfaces, allow the application to react on events ongoing in Hibernate.

`IdentyfierGenerator`, `UserType` and `CompositUserType`, are interfaces responsible for mapping functionality.

(Bauer & King, Java Persistence with Hibernate, 2007)

2.5 JSP technology

JSP, otherwise called JavaServer Pages is a technology which allows easily creating and also maintaining web content for dynamic web pages. JSP is a family of Java technology, which provides rapid development of web applications. JSP page is nothing more than a text document which contains two different types of data:

- static data usually expressed such as HTML, SVG, WML or/and XML
- JSP elements, which build dynamic construction of web application.

This technology separates the layer of presentation (user interface) from generation of content, such as in FIGURE 15.

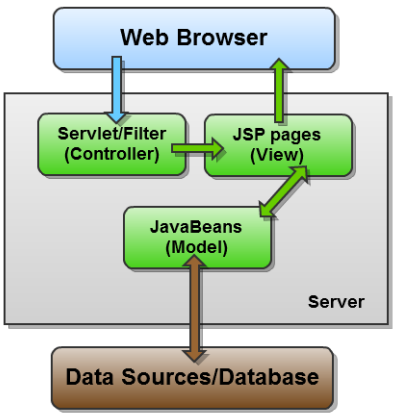


FIGURE 15: JSP layer separation

2.6 Enterprise JavaBeans

Enterprise JavaBeans (EJB) is a part of Java EE and provides services such as transactions, persistence, distribution, security, multiple access, and so on. This technology runs on the server-side as components called beans. EJB components are embedded into container on the server application (EJB container) which provides them to perform locally or remotely.
The idea of EJB contains three main types of EJB beans:

1. Session EJB like Singleton, Stateless and Stateful session beans.

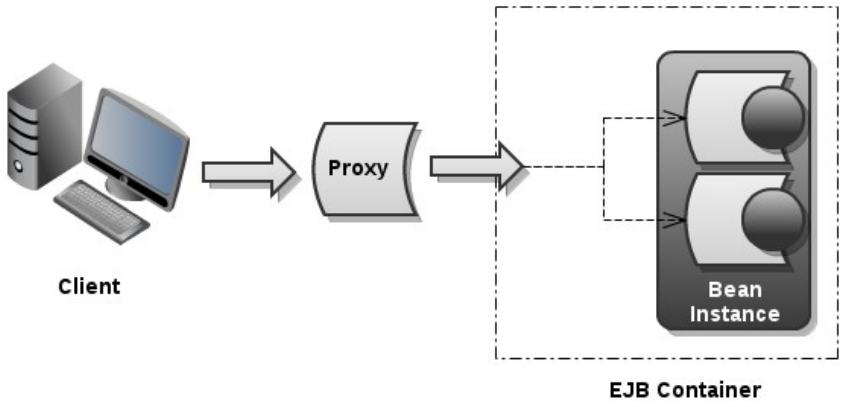


FIGURE 16: Delegating object during EJB session

Because the client does not have direct access to the EJB, it uses the proxy to connect with the container instead. The proxy creates a request with reference that is a delegate to the correct instance and returns the appropriate response. The example is illustrated in FIGURE 16.

Stateless session EJB is a business object which is not associated, which means that the instance is limited to one client at a time, and access to this EJB is disallowed during this associating. This process provides thread-safe session.

Stateful session EJB is totally different than the Stateless session EJB, because a proxy object is running in isolated session context, and another session does not affect the other one.

Singleton session EJB is a business object which has a shared state in entire the JVM. The instance of a singleton bean is controlled by the container.

2. Message-driven EJB

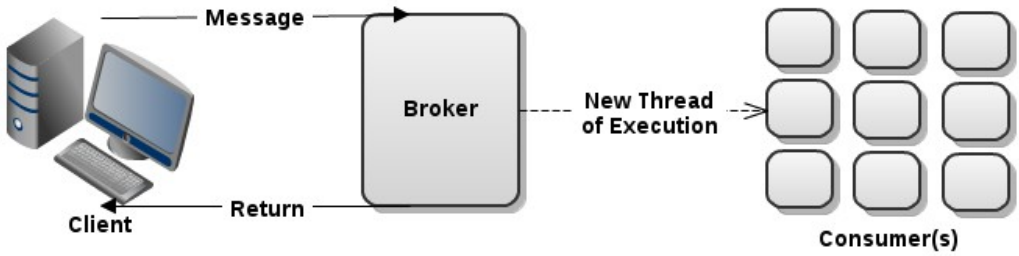


FIGURE 17: Listener of events in message-driven bean

Message-driven EJB is a business object which listens to the consumed messages and can execute them directly or pass them to further processing within the EJB (see FIGURE 17). Message-driven beans provide other messaging protocols, among others asynchronous and synchronous. Differences between message-driven beans and session are ways to calling methods and messaging.

3. Entity EJB

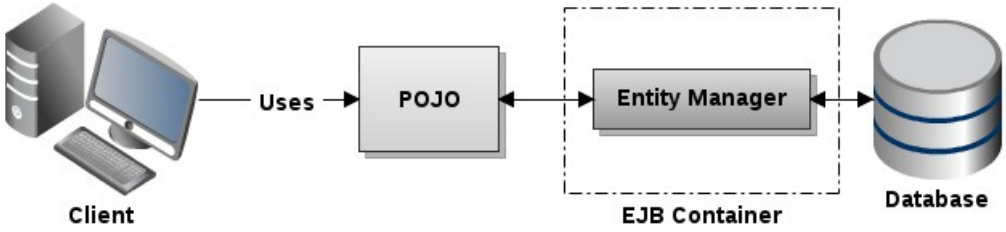


FIGURE 18: EntityManager which mapping POJO object state with persisting to database

The main goal of Entity EJB is “to express an object view of resources stored within a Relational Database Management System (RDBMS)—a process commonly known as object-relational mapping”. In FIGURE 18 is illustrated the task of EntityManager which supplies container service that synchronizes with database changes during the tracking state.

(Lee & Burke, Enterprise JavaBeans 3.1, 2010)

3 INTEGRATION

3.1 Structure of existing project

Before attempting to describe the process of integration, acquaintance with the architecture of existing, large business applications is integral part. All technologies presented in the theoretical part form a unity in this project. This chapter focuses more on the MVC structure in application. Entire application is launched on own servlets. Data for end users is displayed in JSP files. Requests are moved to flows, where values are passed to processing. Dispatcher servlet is responsible for request mapping and Controller forwards the request to model class. In FIGURE 19 is presented the basic concept of Spring MVC flow. The communication between the database layer and the model classes is done by Hibernate.

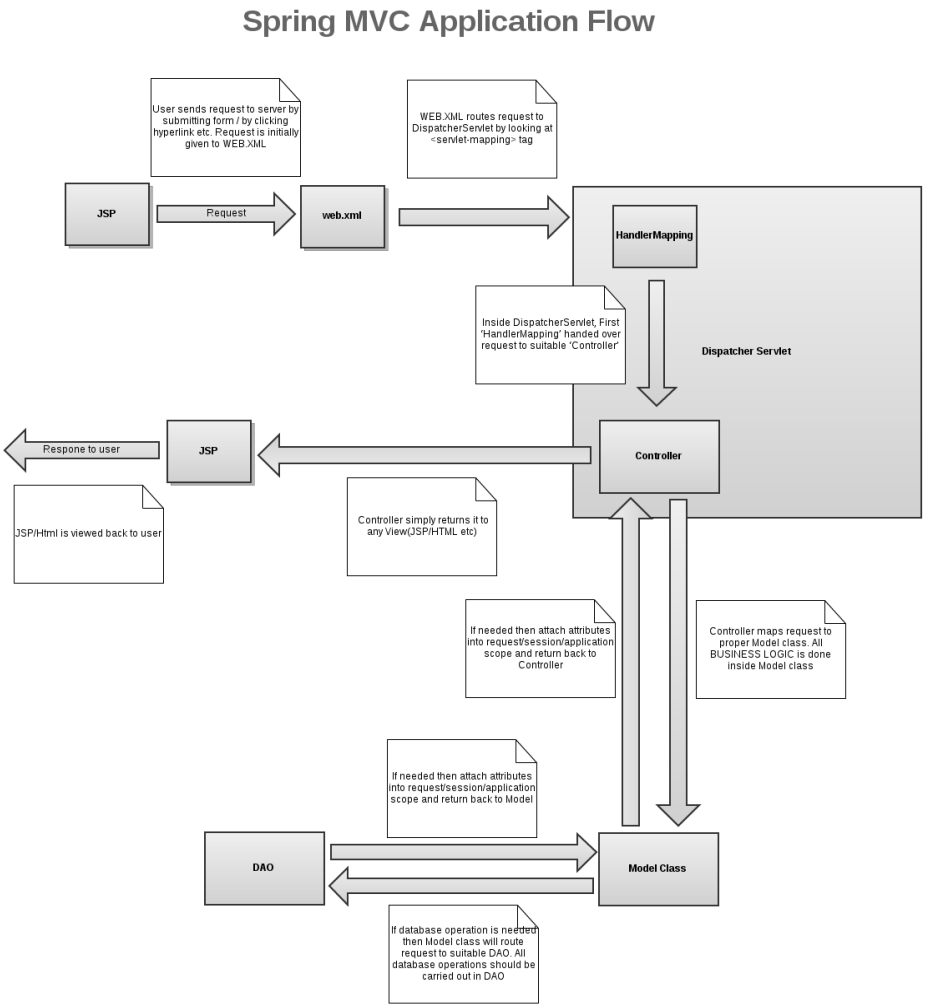


FIGURE 19: Flow diagram in application

3.2 Structure of Vaadin framework integration

The mechanism of action in a Vaadin application is very similar to that of a Spring-based application, considering the data flow. In the presentation layer, Vaadin page sends a JavaScript request in the JSON format to the component. In the next step, the component communicates with a servlet through UIDL. Servlet serves as a kind of bridge between the model classes and the end view. FIGURE 20 shows a simplified mechanism of action in the project's Vaadin application.

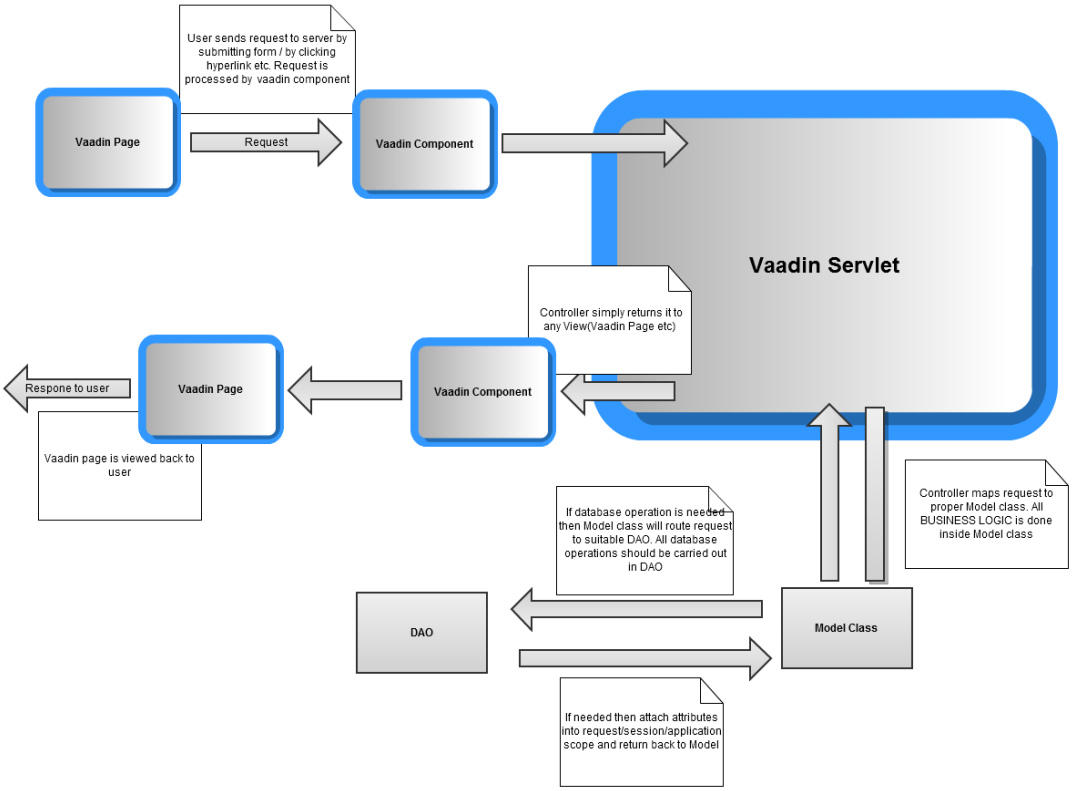


FIGURE 20: The data flow for the implementation of Vaadin

In the FIGURE 21 basics of Vaadin framework injection into legacy application are presented. The idea of this integration is to keep the behaviour of all the properties of the previous applications that offer Spring framework. To do so it is necessary to create a separate servlet for Vaadin that provides all the functionality of this framework.

A more detailed diagram of the structure of integration is in Appendix 1: Architecture of integration Vaadin framework into Spring MVC.

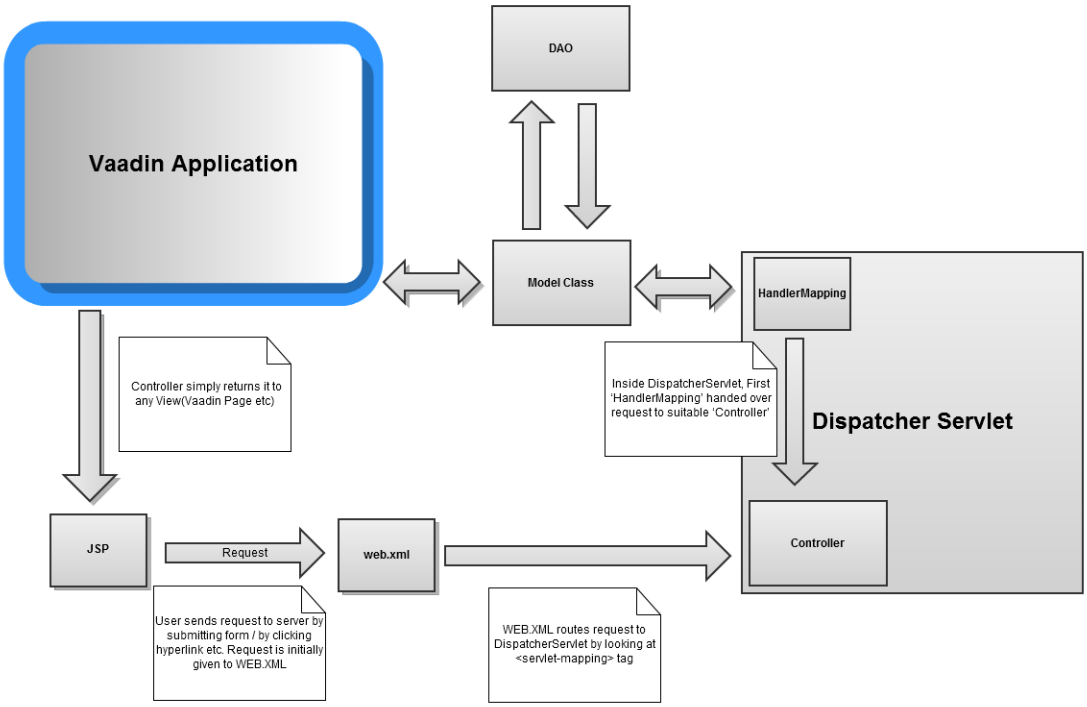


FIGURE 21: Architecture of integration Vaadin framework into Spring MVC

However, several different scenarios of usage need to be considered. The first of these is when a developer wants to create a page independent from other project pages. For this purpose, a pure Vaadin page must be created and embedded in the JSP page. This is the simplest case, which does not require more interference in Spring framework (see FIGURE 22).

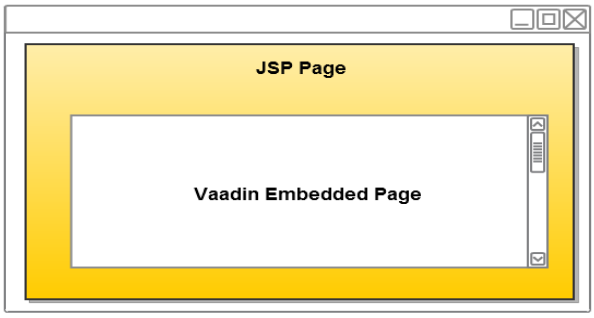


FIGURE 22: Example of placement embedded Vaadin page in JSP page

Another example might be a call from the Vaadin other pages which are contained in the project and will store the value for page from which they are called. It looks like the Vaadin page calls the JSP page on the Spring side, which performs an operation and returns the value for the Vaadin page as has been illustrated in FIGURE 23.

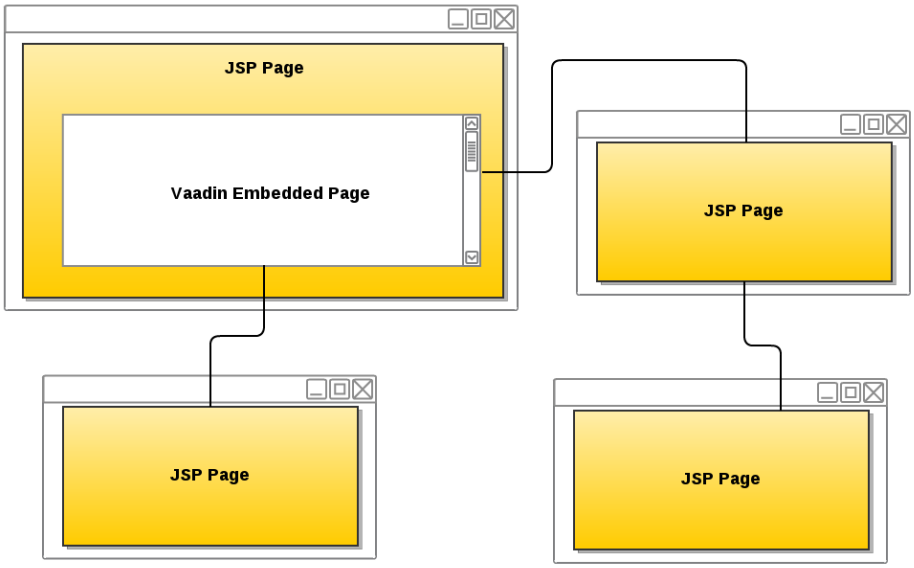


FIGURE 23: Example of placement embedded Vaadin page in JSP page, which invoke other pages for getting proper values

Another case which is a special case of the first, when developer wants to call the JSP page Vaadin. Creating subpages this way is easier because the programmer does not need to focus on the structure of the JSP file - only on the Java source code. It can be pure Vaadin page and also can be embedded page. For the lower layers of the application as Domain and Data Access to Vaadin and Spring use the same object class. Sometimes, however, in addition needs to set the property for Hibernate, if it is not defined in the relevant files with the properties. Returning to Appendix 1: Architecture of integration Vaadin framework into Spring MVC, an important factor which revealing is a servlet on Vaadin side. Construction of servlet is done, when created and initialized applications on the server. Behaviour of this servlet is like normal the life cycle of a servlet. This servlet calls the main class of Vaadin UI, which is listener of request. When this class get proper request, generates right page for the view.

3.3 Environment settings

3.3.1 Overview

In this sub-chapter called "Environment settings" the fundamental steps to start creating Vaadin UI applications inside existing large business application are presented. The basic steps for Vaadin framework integration are described in the following sections:

- 3.3.2 Maven configuration for Vaadin integration
- 3.3.3 Servlet configuration
- 3.3.4 Class resolver implementation
- 3.3.5 Spring injection inside Vaadin framework

3.3.2 Maven configuration for Vaadin integration

Highly facilitating the process of developing business applications is using a software build automation tools such as Maven. This tools provide a rich development infrastructure. The concept is based on a Project Object Model (POM) which may , among others: compilation, testing, documentation, collaboration and reporting are part of Maven tool. Behaviour of application that uses Maven is customized in the POM, which is stored in pom.xml.

The basic settings for Vaadin:

```
<repository>
        <id>vaadin-addons</id>
        <url>http://maven.vaadin.com/vaadin-addons</url>
</repository>
```

This is an optional setting in the project because the project uses its own repository that already has nested this server.

```
<repository>

        <id>central</id>
        <name>Nexus mirror</name>
        <url>http://nexus.descom.fi:8081/nexus/content/groups/public</url>
        <releases>
                <enabled>true</enabled>
        </releases>
        <snapshots>
                <enabled>true</enabled>
                <updatePolicy>always</updatePolicy>
        </snapshots>
</repository>
```

The next step is an add required Vaadin libraries.

```
<!-- Vaadin -->
<dependency>
        <groupId>com.vaadin</groupId>
        <artifactId>vaadin-server</artifactId>
        <version>7.0.4</version>
</dependency>

<dependency>
        <groupId>com.vaadin</groupId>
        artifactId>vaadin-client-compiled</artifactId>
        <version>7.0.4</version>
</dependency>

<dependency>
        <groupId>com.vaadin</groupId>
        <artifactId>vaadin-client</artifactId>
        <version>7.0.4</version>
        <scope>provided</scope>
</dependency>

<dependency>
        <groupId>com.vaadin</groupId>
        <artifactId>vaadin-themes</artifactId>
        <version>7.0.4</version>
</dependency>

<dependency>
        <groupId>com.vaadin</groupId>
        <artifactId>vaadin-client-compiler</artifactId>
        <version>7.0.4</version>
        <scope>provided</scope>
</dependency>
```

The required libraries are:

- The library *vaadin-server-7.0.4.jar* is a part responsible for developing server-side. This library is associated with two other libraries in the *vaadin-themes* and the *vaadin-shared*.

- The library *vaadin-client-compiled-7.0.4.jar* contained a precompiled Client-Side Engine of Vaadin framework. If application using compiled widgets, deploying this library is not necessary.

- The library *vaadin-client-7.0.4.jar* is a part responsible for developing client-side, which including basic Vaadin-specific widgets and GWT API. The library vaadin-client-compiler is required to compile client-side modules. If is used only precompiled Client-Side Engine, deploying this library is not recommended.

- The library *vaadin-shared-7.0.4.jar* contains shared functions to development for both the server and client side.

- The library *vaadin-themes-7.0.4.jar* is required for basic use of custom CSS themes. This library also provides custom SASS themes.

- The library *vaadin-client-compiler-7.0.4.jar* compiles Java-to-JavaScript. This library is needed for building client-side modules.

(Grönroos, Book of Vaadin, 2013)

After adding the necessary dependencies, the next step is to set the plugin to build the widget. The correct plugin should be set as below:

```
<!-- Compile custom GWT components or widget dependencies with the GWT
  compiler. Compilation is invoked by running 'mvn gwt:compile'. There is no
  need to compile the widget sets unless they are changed. -->
<plugin>
        <groupId>org.codehaus.mojo</groupId>
        <artifactId>gwt-maven-plugin</artifactId>
        <version>2.5.1</version>
        <configuration>
                <webappDirectory>
                        ${basedir}/src/main/webapp/VAADIN/widgetsets
                </webappDirectory>
                <extraJvmArgs>-Xmx1024M -Xss1024k</extraJvmArgs>
                <runTarget>xyz-web</runTarget>
                <!-- Plugin should find the modules on its own <modules>
        </configuration>
</plugin>
```

Compiling widgets can take place in two ways, either by defining it properly in Maven or by compiling in the console.

To compile needs to use : `$ mvn gwt:compile -P<profile_name>`

The compile process is time consuming, and the averages machine, it can take a few minutes.

3.3.3 Servlet configuration

Any application that uses Vaadin framework, must have at least one servlet. In this project, is needed to use a custom servlet that would extend the class of `VaadinServlet`

```
package com.vaadin.server;

/**
 * This class is a representation of the configurable servlet, extended
 * by VaadinServlet.
 *
 * @author ext-mszczygi
 *
 */
public class XyzVaadinServlet extends VaadinServlet {

        /**
         * Default serial version UID.
         */
        private static final long serialVersionUID = 1L;

}
```

The body of the class is empty, but the process of extending this servlet is necessary for proper operation of application. Also, here is possibility of define own methods for this servlet.

Deployment descriptor is the main configuration file for the Web application. This is an XML file named web.xml, which is placed directly in the WEB-INF directory. At that file is defined descriptor for custom Vaadin servlet class. There is defined URL address of servlet. This servlet handles the HTTP request. In this initialization of servlet, are needed parameters such as:

- The main class of view, extending UI.
- Parameter for Bean System Messages.
- The GWT class, which contains settings for widgets.

Below is a piece included the deployment descriptor to the issue of integration.

```
<!-- Vaadin servlet -->
<servlet>
	<servlet-name>Vaadin Application Servlet</servlet-name>
	<servlet-class>com.vaadin.server.XyzVaadinServlet</servlet-class>
	<init-param>
		<description>Vaadin UI to display</description>
		<param-name>UI</param-name>
		<param-value>com.xyz.vaadin.VaadinResolverViewUI</param-
value>
	</init-param>

	<init-param>
		<param-name>systemMessagesBeanName</param-name>
		<param-value>DEFAULT</param-value>
	</init-param>

	<init-param>
	<description>Application widgetset</description>
		<param-name>widgetset</param-name>
		<param-value>com.xyz.vaadin.AppWidgetSet</param-value>
	</init-param>
</servlet>
<servlet-mapping>
	<servlet-name>Vaadin Application Servlet</servlet-name>
	<url-pattern>/public/vaadin/*</url-pattern>
</servlet-mapping>
```

3.3.4 Class resolver implementation

This is the class, which is loaded as the main class in the deployment descriptor (web.xml). In VaadinResolverViewUI is presented the core of resolving name to class. The snippet below shows the class.

```
@Theme("descomtheme")
@Title("VaadinResolverViewUI")
public class VaadinResolverViewUI extends UI {

	private static final long serialVersionUID = -2892523868922028481L;
	private String pathInfo;

	public String getPathInfo() {
		return pathInfo;
	}

	@Override
	protected void init(VaadinRequest request) {
		this.pathInfo = request.getPathInfo();
		initView(this.pathInfo);
	}

	public void initView(String pathInfo) {
		VaadinClassLoader.componentResolve(pathInfo, this.getUI());
	}

	public void setPathInfo(String pathInfo) {
		this.pathInfo = pathInfo;
	}
}
```

In method init(), which is called just after the constructor is defined method responsible for generating the proper view. Before this method is called the function that gets from VaadinRequest the name of path mapping. This is a very important process, because the value is passed to the method of generation view (initView(String pathInfo)). Inside this method is invoked static method in VaadinClassLoader class. The source code of this class is in snippet.

```
public class VaadinClassLoader {

        public static final String SAMPLE_PAGE = "/samplePage";
        public static final String VAADIN_TEST = "/vaadinTest";


        public final static CustomComponent componentResolve(String
                                    className, UI parent) {
                CustomComponent component = null;

                if (SAMPLE_PAGE.equals(className)) {
                        component = new SamplePageUI(parent);
                }

                if (VAADIN_TEST.equals(className)) {
                        component = new VaadinTestUI(parent);
                }

                return component;
        }
}
```

This static method compares the path mapping with patterns. If method finds proper path mapping, then generates the right view. Necessary step is send the reference of VaadinResolverViewUI to the proper view. This is needed to set up content inside that class.

3.3.5 Spring injection inside Vaadin framework

An inseparable element to use Spring classes to perform the injection is need create special class which will do that. The first step is to define the ApplicationContext, which will be in the child class has set SpringContextHelper for VaadinServlet. First of all is necessary setting listener for ContextLoaderListener in web.xml

```
<listener>
        <listener-class>
                org.springframework.web.context.ContextLoaderListener
        </listener-class>
</listener>
```

However, to be able to use Spring in application which using Vaadin, is needed to define in the Maven file (pom.xml) following dependencies:

```
<!-- SPRING -->
<dependency>
        groupId>org.springframework</groupId>
        <artifactId>spring-core</artifactId>
        <version>3.2.2.RELEASE</version>
        <type>jar</type>
</dependency>

<dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>3.2.2.RELEASE</version>
</dependency>

<dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-beans</artifactId>
        <version>3.2.2.RELEASE</version>
</dependency>
```

To get access to Spring managed beans is need create a helper class. This class will get the relevant session information.

```
import javax.servlet.ServletContext;
import org.springframework.context.ApplicationContext;
import org.springframework.web.context.support.WebApplicationContextUtils;

public class SpringContextHelper {

	/**
	 * Application context.
	 */
	private ApplicationContext context;

	/**
	 * Constructor.
	 *
	 * @param servletContext
	 */
	public SpringContextHelper(ServletContext servletContext) {

		context =
WebApplicationContextUtils.getRequiredWebApplicationContext(servletContext);
	}

	/**
	 * Get bean.
	 *
	 * @param beanRef
	 * @return the context of bean
	 */
	public Object getBean(final String beanRef) {
		return context.getBean(beanRef);
	}

}
```

The easiest way to get this context for Vaadin application, is create method which will gets this ApplicationContext inside class which needs it use.

```
/**
 * Spring context helper.
 */
private SpringContextHelper helper;

/**
 * Initialise Spring Context Helper.
 */
private void initSpringContextHelper() {
	helper =
   new SpringContextHelper(VaadinServlet.getCurrent().getServletContext());
}
```

3.4 Integration into Spring MVC

In this subchapter will show you an example of integration Vaadin application in already existing legacy application. It is a process that applies to various parts of the project to make a proper integration which is due to pre-set the environment as shown in 3.3 Environment settings.

First of all,it is necessary to define the bean in XML file for sample JSP file.

```
<bean classtype="com.xyz.web.component.MenuComponent">
        <set-property property="resourceKey" value="page.vaadinTest"/>
        <set-property property="internalUrl"
        value="/public/controller/vaadin/vaadinTest" />
        <set-property property="role" value="ROLE"/>
</bean>


<definition name="vaadinTest" extends="adminBase">
        <put-attribute name="selected" value="page.vaadinTest"/>
        <put-attribute name="content-main"
        value="/WEB-INF/jsp/settings/vaadinTest.jsp"/>
</definition>
```

An example JSP file is located in "/WEB-INF/jsp/settings/vaadinTest.jsp". Vaadin application is embedded in this file and the latter part of this chapter explains the structure of the file. There is also a set mapping for that JSP file in the bean. "/public/controller/vaadin/vaadinTest" this mapping will redirect to the correct JSP file located under that directory "/WEB-INF/jsp/settings/vaadinTest.jsp".

To get a request mapping additional controller class is needed. By adding @Controller annotation to class, then serves the role of controller. This allows avoiding the reference to the Servlet API. Before adding this, an annotation is required to be set up in webflow.xml component-scan and the base-package to be set.

```
<!-- Scan all classes in base-package and look for @Component or @Service
        annotations. If such an annotated class is found, automagically instantiate
        a bean out of it. -->
<context:component-scan base-package="com.xyz" />

<mvc:annotation-driven conversion-service="mvcConversionService" />
```

The dispatcher will be scan the annotated class and detecting @RequestMapping annotations. An example of controller class is presented in the snippet:

```
@Controller
@RequestMapping(value = "/vaadin")
@RolesAllowed("ROLE")
public class VaadinController {

	@Autowired
	private DomainObjectService domainObjectService;

	/**
	 * @return the domainObjectService
	 */
	public DomainObjectService getDomainObjectService() {
		return domainObjectService;
	}

	/**
	 * @param domainObjectService
	 *            the domainObjectService to set
	 */
	public void setDomainObjectService(DomainObjectService
			domainObjectService) {
		this.domainObjectService = domainObjectService;
	}

	@RequestMapping(value = "/vaadinTest")
	public String viewVaadinTestPage() {
		return "vaadinTest";
	}
}
```

To map URLs is used @RequestMapping annotation like @@RequestMapping(value = "/vaadinTest") and refers to a particular handler method, however, it can also refer to an entire class. Typically these annotations map HTTP requests such as GET or POST inside the controller.

The next step is to create a sample class, demonstrating the integration of the Vaadin into existing application.

```
/**
 *
 * Test class, created for the purpose of demonstration.
 *
 * @author ext-mszczygi
 *
 */
@SuppressWarnings("serial")
public class VaadinTestUI extends CustomComponent {

	/**
	 * Variable keeping reference to parent.
	 */
	private UI parentReference;

	/**
	 * Constructor for VaadinTest, which initialise all Component. After call this
	 * method, setParentReference() is need.
	 */
	public VaadinTestUI() {
		...
	}

	/**
	 * Constructor for VaadinTest, which initialise all Component.
	 *
	 * @param parent
	 */
	public VaadinTestUI(UI parent) {
		setParentReference(parent);
		initLayout();
	}

	/**
	 * @return the parentReference
	 */
	public UI getParentReference() {
		return parentReference;
	}

	/**
	 * Initialise layout.
	 */
	private void initLayout() {
		// Creates vertical part of layout
		...
		parentReference.setContent(layout);
	}

	/**
	 * @param parentReference
	 *          the parentReference to set
	 */
	public void setParentReference(UI parentReference) {
		this.parentReference = parentReference;
	}
}
```

To get the bean inside this class configuration explained in 3.3.5 Spring injection inside Vaadin framework is needed. For defined SpringContextHelper, it is possible to get a bean, for example "staticService" like that:

```
/**
 * @return the staticService
 */
public StaticService getStaticService() {
        return staticService = (StaticService) helper.getBean("staticService");
}
```

For VaadinResolverViewUI class to render the appropriate class, is needed define the path pattern in VaadinClassLoader class. Mapping patterns are compared as a String. When the proper pattern is found, creates a new object of this class and is passed the reference of this VaadinResolverViewUI class in order to set the appropriate view.

```
public static final String VAADIN_TEST = "/vaadinTest";
...
if (VAADIN_TEST.equals(className)) {
        component = new VaadinTestUI(parent);
}
```

The final step is to embed Vaadin applications in JSP file. The file named *vaadinTest.jsp* should contains this elementary issues:

- Set up the page header

```
<head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
        <meta http-equiv="X-UA-Compatible" content="IE=9;chrome=1" />
        <title>Vaadin Integration</title>
</head>
```

The embedded UI application inside JSP file should conform to the standard XHTML. Encoding for characters must be set to UTF-8. For better compatibility meta declarations can be set.

- Contains the *vaadinBootstrap.js*

```
<!-- Loads the Vaadin widget set, etc. -->
<script type="text/javascript" src="/xyz-web/VAADIN/vaadinBootstrap.js">
</script>
```

The *vaadinBootstrap.js* boots up UI application. This script should be called before initialization UI. The source path for *vaadinBootstrap.js* should be relative.

- Includes a GWT history frame

```
<!-- GWT requires an invisible history frame. -->
<!-- It is needed for page/fragment history in the browser. -->
<iframe tabindex="-1" id="__gwt_historyFrame" style="position: absolute;
        width: 0; height: 0; border: 0; overflow: hidden" src="javascript:false">
</iframe>
```

GWT needed an invisible history for tracking the page history.

- Defines for Vaadin application

```
<!-- So here comes the div element in which the Vaadin -->
<!-- application is embedded. -->
<div id="vaadinTest" class="v-app">

        <!-- Optional placeholder for the loading indicator -->
        <div class=" v-app-loading"></div>

        <!-- Alternative fallback text -->
        <noscript>You have to enable javascript in your browser to use an
                application built with Vaadin.
        </noscript>
</div>
```

- The UI application will be embedded into this html element. Also, this element must contain unique id in the page. In addition, should contain three more elements:

1. In must be define *v-app* style class.

2. Inside elements should be defined nested element, which contain *v-app-loading* style class. This item is just to indicate the loading status of UI application.

3. Required is also <noscript> element, which contains informations about an unsupported JavaScript by browser.

- Script initializes the Vaadin UI

```
<script type="text/javascript">//<![CDATA[
        if (!window.vaadin)
        alert("Failed to load the bootstrap JavaScript: "+
        "VAADIN/vaadinBootstrap.js");
```

The UI application is rendered after calling method vaadin.initApplication(). Before calling that method, should be checked if the vaadinBootstrap.js was loaded properly.

```
        /* The UI Configuration */
        vaadin.initApplication("vaadinTest", {
        "browserDetailsUrl" : "/xyz-web/public/vaadin/vaadinTest",
        "serviceUrl": "/xyz-web/public/vaadin/",
        "widgetset" : "com.xyz.vaadin.AppWidgetSet",
        "theme": "descomtheme",
        "versionInfo" : { "vaadinVersion" : "7.0.4" },
        "vaadinDir": "/xyz-web/VAADIN/",
        "heartbeatInterval": 300,
        "debug": true,
        "standalone": false,
        "authErrMsg": {
                        "message": "Take note of any unsaved data, "+
                        "and <u>click here<\/u> to continue.",
                        "caption": "Authentication problem"
                        },
        "comErrMsg": {
                        "message": "Take note of any unsaved data, "+
                        "and <u>click here<\/u> to continue.",
                        "caption": "Communication problem"
                        },
        "sessExpMsg": {
                        "message": "Take note of any unsaved data, "+
                        "and <u>click here<\/u> to continue.",
                        "caption": "Session Expired"
                        }
        });//]] >
</script>
```

The vaadin.initApplication() method takes two parameters.

1. The UI identifier, the same as the unique in element.

2. Associative map that contains the following parameters:

   - "browserDetailsUrl" – This parameter must set the URL path, which should point to the Vaadin servlet.

   - "serviceUrl" – This parameter is need to communicate by UIDL. The value of this parameter should be set the same as at <servlet-mapping> for Vaadin UI in web.xml.

   - -"widgetset" – This parameter must be define exact class name for widgets. The extension of .gwt.xml need be omitted. If any specific widgets are not used, default widget set set can be com.vaadin.DefaultWidgetSet.

   - "theme" – The parameter sets theme for UI application. The theme can be custom or built-in (runo, chameleon or reindeer).

   - "versionInfo" – This parameter contains the associative map. The parameters for this map are optional, but contain the number version of currently using Vaadin version.

   - "vaadinDir" – Path to VAADIN directory. Location is relative to the URL address, where is located embedded page.

   - "heartbeatInterval" – This parameter sets the message sending frequency, preventing session time-out.

   - "debug" – Specifies whether the window debugging is enabled.

   - "standalone" – Usually this parameter should be set as false. This defines whether the UI application is generated in the browser window or another context.

- `"authErrMsg"` – The parameter defines authentication error. This parameter contains an associative map with two key-value pairs: message and caption.

- `"comErrMsg"` – The parameter defines communication error. This parameter contains an associative map with two key-value pairs: message and caption.

- `"sessExpMsg"` – The parameter defines error for session expiration. This parameter contains an associative map with two key-value pairs: message and caption.

(Book of Vaadin, Marko Grönroos, 2013)

An example configuration of the entire embedded page is in Appendix 2: Example code of embedded Vaadin UI in JSP file.

3.5 Calling Spring pages from the Vaadin page

Using already created items that exist in the project, is a very common thing. Use of Spring components in situation as shown in FIGURE 23 on the page 27 is an inevitable process. To be able to call up the Spring page from the embedded Vaadin page is necessary to send a request to webflow. In Vaadin page of UI need to add:

```
EditLink edit = new EditLink("/xyz-web/sample-flow.xyz?
_eventId=editSomething&elementId=" + element.getId() + "&vaadin=true");
```

Class for EditLink is attached in Appendix 3: Source code of EditLink class.

Next step in sample-flow must be set correct scopes for obtained values.

```
<on-start>
        <set name="flashScope.samplePageId"
             value="requestParameters.elementId"/>
        <set name="flowScope.vaadin" value="requestParameters.vaadin"/>
</on-start>
```

Also is required set up proper <decision-state> for <subflow-state>:

```
<decision-state id="decideEditSomething">
        <if test="requestParameters._eventId == 'editSomething'"
            then="editSomething" else="..."/>
</decision-state>
...

<subflow-state id="editSomething" subflow="editsomething-flow">
...
</subflow-state>
```

In the next step, the view will be transitioned to the *editSomething.jsp* page. Inside this file different activities can be made. Also, this JSP file contains its own webflow. The following are settings for this webflow file:

```
<on-start>
        <set name="flowScope.vaadin" value="requestParameters.vaadin"/>
</on-start>

<decision-state id="decideSelect">
        <if test="flowScope.vaadin == 'true'" then="selectVaadin" else="select"/>
</decision-state>

<view-state id="selectVaadin"
view="externalRedirect:contextRelative:/public/controller/vaadin/samplePage/
selectSample?elementId=${flowScope.elementId}&amp;elementId=
${flowScope.selectedElementId}"/>
```

After the appropriate executions, view back to the embedded Vaadin page.

4 CONCLUSIONS

The subject of this thesis was to show the possibility of integration Vaadin framework and reduction effort of developers work in creating new features. The process of this integration allows the introduction of additional features for the project, which was not possible to create in the Spring framework without the use of additional framework for the presentation layer. In practice, for a similar solution was used in the project Dojo toolkit. However, in comparison with the Vaadin framework, the process of implementation of the solution take place at the level of the JSP file and not, as is the case of Vaadin, in java source files.

The ability to embed elements on the web page, provides component development for project, thus making it faster to develop applications. For developer who previously used Spring MVC for creating pages and subpages, it can be noted how to reduce the file declaration. Usually, developers must create JSP file, webflow and sometimes controller. In the case of Vaadin, it is a single file, without the template that is embedded inside the JSP file.

However, the advantages and disadvantages of using this framework in the project need also to be taken into account.

TABLE 1: Advantages and Disadvantages of using Vaadin in the project.

ADVANTAGES	DISADVANTAGES
Vaadin allows for development both on server- and client-side.	JavaScript-based components make the resulting HTML code heavy and the styling more complicated.
Time to create new functions is reduced significantly; developers do not have to, for example, create files necessary as while using Spring MVC (xml, flow, JSP page, etc.). Instead, everything is done in Java and optional CSS.	Developers need training in a different than previous technology; training and time costs.

Scalability of solutions.	Some project features will be rewritten in Vaadin technology, which essentially make them more costly.
Good documentation and highly active user base.	
Professional support from Vaadin creators.	
Many ready-to-use components and addons.	
Easy integration with other frameworks like Spring and Hibernate.	
Support for all modern browsers.	
Using native open source software has a positive impact on its growth and development.	
Fast and stable solution, albeit relatively young.	
Vaadin has a great Eclipse plugin, which saves a great deal of developer’s time and work.	
Specialized in UI widgets for mobile devices.	

As can be seen there are plenty of advantages, thus the implementation of this integration to existing applications it is profitable for both the developer and the client. The use of Vaadin framework for this application looks promising.

REFERENCES

Martin Fowler, 2002, Patterns of Enterprise Application Architecture, Addison-Wesley Professional.

Deepak Alur, Jhon Crupi and Dan Malks, 2003, core J2EE Patterns, Sun Microsystems Press.

Eric Jendrock, Jennifer Ball, Debbie Carson, Ian Evans, Scott Fordin and Kim Haase, 2007, The Java EE 5 Tutorial, Sun Microsystem, Accessed on 15 April 2013, http://docs.oracle.com/javaee/5/tutorial/doc/

Carla Sadtler, Fabio Albertoni, Leonard Blunt, Micheal Connolly, Stefan Kwiatkowski, Thayaparan Shanmugaratnam, Henrik Sjostrand, Saori Tanikawa, Margaret Ticknor and Joerg-Ulrich Veser, July 2009, WebSphere Application Server V7 Adnministration and Configuration Guide, IBM.

Rob Harrop and Jan Machacek, 2005, Pro Spring, Apress.

Craig Walls and Ryan Breidenbach, 2005, Spring in action, Manning Publications Co.

Seth Ladd with Darren Davison, Steve Devijver and Colin Yates, 2006, Expert Spring MVC and Web Flow, Apress.

Marko Grönroos, 2013, Book of Vaadin, Vaadin Ltd.

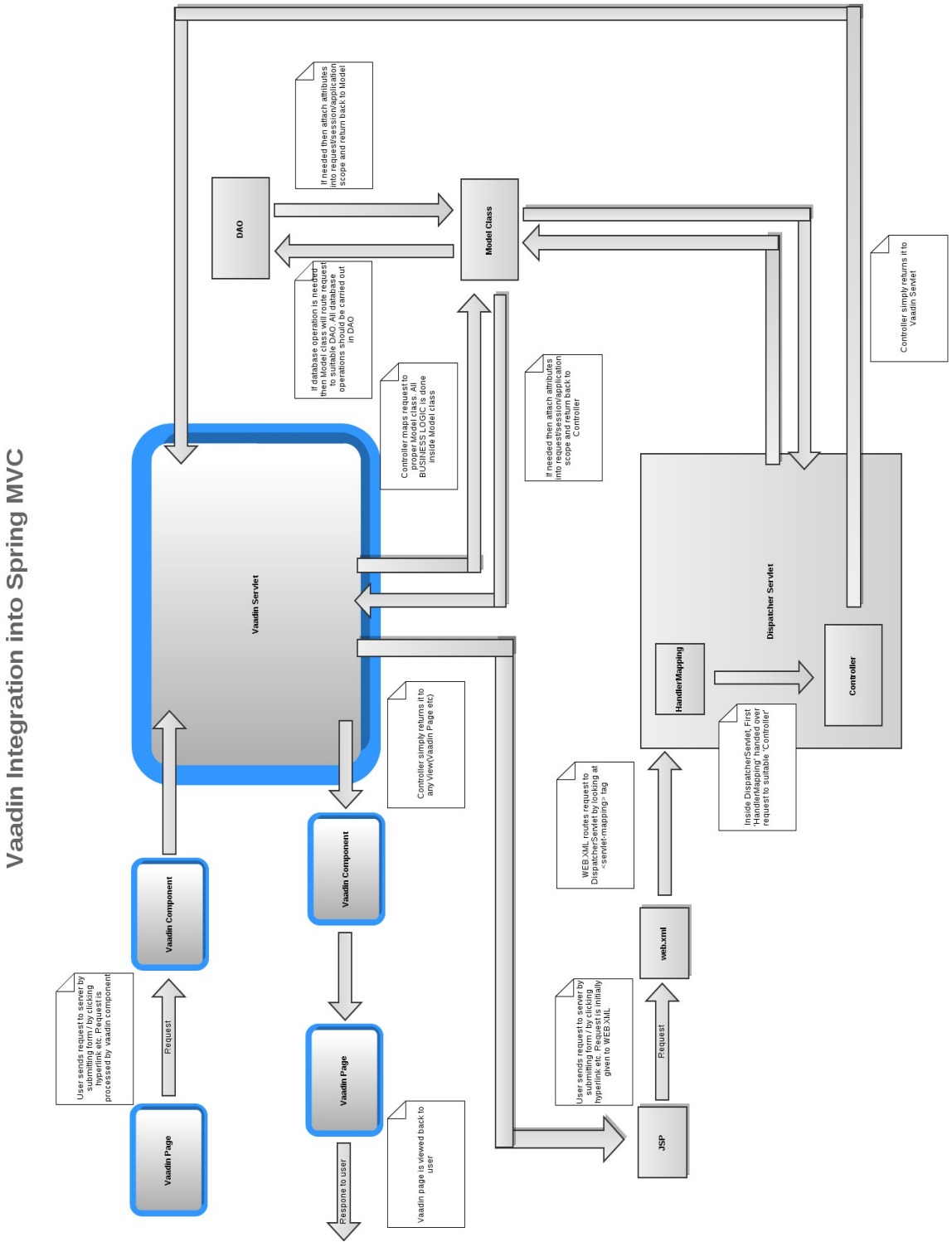
Christian Bauer and Gavin King, 2007, Java Persistence with Hibernate, Manning Publications Co.

Andrew Lee and Bill Burke, 2010, Enterprise JavaBeans 3.1, O'Reilly Media.

Marko Grönroos, 2013, Book of Vaadin, Vaadin Ltd.

APPENDICES

Appendix 1: Architecture of integration Vaadin framework into Spring MVC



Appendix 2: Example code of embedded Vaadin UI in JSP file

```
<!-- Loads the Vaadin widget set, etc. -->
<script type="text/javascript" src="/xyz-web/VAADIN/vaadinBootstrap.js">
</script>
<!-- GWT requires an invisible history frame. -->
<!-- It is needed for page/fragment history in the browser. -->
<iframe tabindex="-1" id="__gwt_historyFrame" style="position: absolute;
        width: 0; height: 0; border: 0; overflow: hidden" src="javascript:false">
</iframe>

<div id="content-main">

        <h3>Vaadin Test Page</h3>

        <!-- So here comes the div element in which the Vaadin -->
        <!-- application is embedded. -->
        <div  id="vaadinTest" class="v-app">

                <!-- Optional placeholder for the loading indicator -->
                <div class=" v-app-loading"></div>

                <!-- Alternative fallback text -->
                <noscript>You have to enable javascript in your browser to use an
                                application built with Vaadin.
                </noscript>
        </div>

        <script type="text/javascript">//<![CDATA[
                if (!window.vaadin)
                alert("Failed to load the bootstrap JavaScript: "+
                "VAADIN/vaadinBootstrap.js");

                /* The UI Configuration */
                vaadin.initApplication("vaadinTest", {
                "browserDetailsUrl" : "/xyz-web/public/vaadin/vaadinTest",
                "serviceUrl": "/xyz-web/public/vaadin/",
                "widgetset" : "com.xyz.vaadin.AppWidgetSet",
                "theme": "descomtheme",
                "versionInfo" : { "vaadinVersion" : "7.0.4" },
                "vaadinDir": "/xyz-web/VAADIN/",
                "heartbeatInterval": 300,
                "debug": true,
                "standalone": false,
                "authErrMsg": {
                                "message": "Take note of any unsaved data, "+
                                "and <u>click here<\/u> to continue.",
                                "caption": "Authentication problem"
                                },
                "comErrMsg": {
                                "message": "Take note of any unsaved data, "+
                                "and <u>click here<\/u> to continue.",
                                "caption": "Communication problem"
                                },
                "sessExpMsg": {
                                "message": "Take note of any unsaved data, "+
                                "and <u>click here<\/u> to continue.",
                                "caption": "Session Expired"
                                }
                });//]] >
        </script>
</div>
```

Appendix 3: Source code of EditLink class

```
/**
 * This class contains the settings for the Link whose behaviour
 * is to remind the edit button.
 *
 * @author ext-mszczygi
 *
 */
public class EditLink extends Link {

	/**
	 * Default serial version UID.
	 */
	private static final long serialVersionUID = 1L;

	/**
	 * Constructor for EditLink.
	 */
	public EditLink() {
		setIcon(new ThemeResource("icons/16/edit.gif"));
	}

	/**
	 * Constructor for EditLink.
	 *
	 * @param resource
	 */
	public EditLink(String resource) {
		setIcon(new ThemeResource("icons/16/edit.gif"));
		setResource(new ExternalResource(resource));
	}

	/**
	 * Constructor for EditLink.
	 *
	 * @param caption
	 * @param resource
	 */
	public EditLink(String caption, String resource) {
		setIcon(new ThemeResource("icons/16/edit.gif"));
		setResource(new ExternalResource(resource));
		setCaption(caption);
	}

}
```