Developing Rich Web Applications with PrimeFaces in Java EE7

Leon Dobnik, Marcel Šalej

Contact: dobnik.leon@icloud.com, marcel.salej@icloud.com

Summary: *Ever since the appearance of HTML5 several years ago, the area of web applications has seen an increasing presence of JavaScript libraries, as well as other technologies and standards that make it possible to build dynamic web pages, portals, and ERP environments. Some of these standards are implemented by Java EE, which is being developed by Oracle, industry experts, and open source organizations. JavaServer Faces in combination with PrimeFaces makes it possible to create dynamic web applications. We have illustrated the use of JavaServer Faces, PrimeFaces, and other Java EE standards with a practical example of creating online lumber auctions. This article presents the technologies used in the application.*

1. INTRODUCTION

In recent years, we have witnessed numerous changes and innovations in web application development. This has affected languages (HTML5, CSS3, JavaScript closures), technologies (REST web services, WebSockets), and conventions in web application development (HTML5 applications calling web services, server-side JavaScript, etc.). Furthermore, it has almost become a necessity to provide a mobile web application along with the usual web interface. All of this quickly leads us to the question: "What kind of approach should be taken in order to follow technological progress, to stay market-competitive, to ensure a good foundation for further development, to maintain the application with as little funds as possible, to achieve the expected performance level, and to justify the expenses?" The answer differs from one case to another. It depends on a number of factors: the development group's knowledge, the infrastructure, possible target customers, the expected performance level of the application, the type of the application, and so on.

Since we are developing an application for online lumber auctions, we looked for a reliable platform based on standards and good practice; a platform which makes it possible for us to immediately focus on developing the application's functionality with ready-made components. We wanted to find an open source platform with the capacity for further extensibility (e.g. a mobile interface, integration of additional libraries, etc.).

2. THE TECHNOLOGIES USED

2.1. Java Server Faces

2.1.1. What is Java Server Faces?

Java Server Faces (JSF) is "a server-side component framework for building Java technology-based web applications." [16] The framework consists of the following:

- an API for representing components and managing their state; handling events, server-side validation, and data conversion; defining page navigation; supporting internationalization and accessibility,
- tag libraries for adding components to web pages and for connecting components to server-side objects.

The typical JSF page is created from two parts [16]:

- an XHTML[1] file in which Facelets (a page declaration language) is used. The file requires Facelets tags to define components (e.g. *<h:inputText />)*, and the use of EL (Expression Language) to connect to managed beans (e.g. *#{managedBean.string}*). In practice, such a page will also contain HTML tags (e.g. **), CSS[2], JavaScript[3], and the like. An example of a simple page in JSF is provided in Image 1:

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:f="http://xmlns.jcp.org/jsf/core">
    <h:head>
        <title>Dobrodošli</title>
    </h:head>
    <h:form>
        <h:outputLabel value="Vnesi niz" for="niz" style="color: blue;" />
        <strong>
          <h:inputText id="niz" value="#{podpornoZrno.niz}" required="true" />
        </strong>
        <br />
        <h:commandButton value="Pošlji"
                         actionListener="#{podpornoZrno.prilepiKlicaj()}"
                         action="dobrodosli" />
    </h:form>
</html>
```

Image 1: The dobrodosli.xhtml (English: welcome.xhtml) file in a JSF web page

- a managed bean, which is a Java class labeled with fixed annotations. It contains methods and data used by components defined in the XHTML file; at the same it specifies for how long the data is kept server-side (the scope) and provides an interface to the application's business logic level. Image 2 presents an example of a managed bean for the XHTML file in Image 1. The annotations *@Named* and *@RequestScoped* signal that this is a managed bean which creates itself at every request by the browser. The browser retains the data from the input field (*String niz)* and contains the method *prilepiKlicaj()* (English: *attachExclamationMark()*).

```
@Named
@RequestScoped
public class PodpornoZrno {

  private String niz;

  public String getNiz() {
    return niz;
  }

  public void setNiz(String niz) {
    this.niz = niz;
  }

  public void prilepiKlicaj() {
    niz = niz + "!";
  }
}
```

Image 2: The managed bean for the dobrodosli.xhtml (English: welcome.xhtml) file

[1] XHTML (Extensible HyperText Markup Language): a markup language for building HTML pages.
[2] CSS (Cascading Style Sheets): a style sheet language used for describing the look and formatting of a document (mostly web pages) written in a markup language.
[3] JavaScript: a script language for building dynamic and interactive web pages.

A simple page (presented in images 1 and 2) is generated in a browser as an input form with one input field. Whenever a user clicks on the button *Send*, an exclamation mark is attached to the end of the string which is sent, and the page is refreshed.

2.2. JSF and Java EE 7

The Java Platform, Enterprise Edition 7 (Java EE 7) comes with a new version of JSF – version 2.2, defined in the JSR 344 specification[7]. Important changes include the following:

- Facelets is now used to define view since JSP[4] was deprecated and does not support all of the innovations introduced by Facelets;
- CDI beans are used to define managed beans, which were deprecated as well. Back in the times of Java EE6, numerous articles already advised the use of CDI[5] beans [9] because they offered more than managed beans. However, the problem with CDI beans in the times of EE6 was the lack of scope of the *ViewScoped* bean. Scope refers to the bean preserving its status as long as the user is sending requests from the same page;
- The Faces Flows feature is introduced as the standard for creating flows. This means that pages with the same scope can now be much more easily defined. This comes useful when, for example, creating a wizard. Faces Flows standardizes the flows which the frameworks Spring and ADF have been offering for some time under the names Spring Flows and ADF Task Flows;
- HTML5 support has been added, which means that standard HTML5 tags can be used as JSF components [1] and connected to the JSF framework through a few attributes;
- We also need to mention:
    - *<h:inputFile />,* the new component for transferring files to the server;
    - the ability to define stateless JSF views – the so-called "view" is not stored on the server;
    - And Resource Library Contracts, which allow us to dynamically reuse Facelets templates in the entire application while it is being executed. This means, in other words, that the system which allows us to produce different appearances and themes in an application has been standardized [3].

2.2.1. Recently resolved problems in the world of JSF

As a standard or specification, JavaServer Faces (JSF) is defined in Java Specification Requests (JSRs); for example, JSF version 2.2 is defined in the previously mentioned JSR-344 specification. These specifications are implemented by the developers. Two implementations of JSF are currently being used: one is Oracle's Mojarra, the other is Apache's MyFaces. Some application servers (e.g. GlassFish) come with Oracle's Mojarra JSF 2.x, which used to be tormented by two problematic bugs:

- If we created a JSF page with a large number of JSF components (e.g. 250 and more), the page rendering time increased significantly [3]. It was confirmed that the problem lay with Mojarra [20]. In version 2.1.22 the problem was resolved, as confirmed by repeated tests [2];
- A problem occurred when using JSF ViewScoped managed beans: some JSF tags (e.g. *<f:convertNumber />*) did not function properly [23] – the so-called "chicken-egg" bug, officially managed under the number 1492. In version 2.1.18 the problem was resolved [6].

[4] JSP (Java Server Pages): A technology used to create dynamically generated web pages.
[5] CDI (Context and Dependency injection): A collection of services which make it easier for developers to connect enterprise beans to the JSF technology. (Specification JSR-299)

2.2.2. What is missing in JSF?

Once we were acquainted with the concepts of development in JSF —the scope of managed beans, implementation of Faces validators, implementation of Faces converters, building XHTML pages, etc.— and once we looked over JSF components, we learned a few things:

- JSFs allow an application's presentational and business levels to be clearly separated. The use is simple when it comes to the i18N standard, server-side validation and conversion of input data, and specifying for how long data should be kept server-side (achieved with the scope of managed beans). AJAX[6] is supported as well;
- While creating the structure of the page itself, we learned that the framework does not include any layout manager components, nor does it include any skins, and the set of graphic components is basic at best. We would like to see, for example, a calendar component; the table component *<h:dataTable />* does not support defining filters, sorting, pagination, displaying graphs, image galleries, and such;
- We noticed some problems in the connection between an XHTML file and a managed bean:
    - The JSF framework does not allow direct addressing of constants (unless we define the *getter* and *setter* methods);
    - If we want to establish a connection between a managed bean and a component displaying itself as a drop-down menu (e.g. *<h:selectOneMenu />*) via a Java object, it is necessary to implement a data type converter;
    - If executing an AJAX request results in a JSF error and no additional functionalities are implemented, the end user is not even aware of it (the error is visible only in the browser console).

JSF is fundamentally a standard which seeks to provide an adaptable and extensible framework for developing web applications. If we take that into account, we can understand why it does not include any rich components for browser display. The framework would have to include JavaScript libraries as well in order to display components client-side. The question is whether the market offers any JSF framework extensions which can solve this problem. The answer is, of course, yes – there are three large projects which include rich widgets for developing web applications: ICEfaces, RichFaces, and PrimeFaces. Based on Google Trends statistics [5] and some comparative tests, PrimeFaces is the best choice [14].

2.3. PrimeFaces

2.3.1. What is PrimeFaces?

PrimeFaces is an open source component suite which extends the JSF 2.0 (or newer) framework and enables fast development of rich web applications. The latter is possible due to:

- More than a hundred JSF components for displaying graphic interfaces such as image galleries, calendars, or tables which support sorting, filtering, pagination, an HTML editor, and so on;
- Prepared stubs for data components which enable custom implementation of sorting, filtering, and pagination. An example of such a component is a table (tag *<p:dataTable />*);
- Refined support for exporting data from tables to other formats (XML, XLS, PDF, and CSV);
- The supported framework for creating themes, which brings with it more than 35 themes defined in advance and JSF components for defining page layout;
- Improved and simplified use of AJAX as opposed to basic JSF, and support for the "Push" technology through the Atmosphere framework;

[6] AJAX (Asynchronus JavaScript): a group of techniques used client-side to create asynchronous web applications.

- The included library for developing mobile applications, and the library of rich HTML5 widgets which are equal to HTML widgets into which JSF components are mapped.

We have mentioned that PrimeFaces is an open source project. However, we need to emphasize the fact that only the major PrimeFaces releases (e.g. 4.0, 5.0) are open source [8]. Maintenance, bugfix-only releases (e.g. 4.0.10) are now available exclusively to those who pay for them. There are two types of support available to us: the first is PrimeFaces ELITE, which provides access to maintenance releases and gives rights in the PrimeFaces issue voting system; the second is PrimeFaces PRO, which offers full support – e.g. critical fixes, counseling in development, and so on. We believe that the open source release is good enough for prototyping and/or learning, whereas for production working we advise you to consider investing in PrimeFaces ELITE.

2.3.2. Creating a custom theme in PrimeFaces

As pointed out before, PrimeFaces includes a framework for creating themes. It is based on the JQuery ThemeRoller web application[7], with which we can easily create our own, unique theme. We import it in our own application in the form of a JAR archive which has to have the same structure as shown in Image 3.

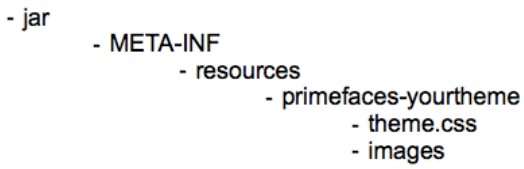


Image 3: The structure of a theme in a JAR archive

Before we compress the theme, we need to change[8] the paths to images in the main style sheet (theme.css) and add a path to it in the *web.xml* file. The theme's framework consists of two types of style sheets:

- Structural CSS – an external style sheet which defines the framework of all PrimeFaces components, as well as other skinning features important for website architecture (borders, padding, dimensions, and positions);
- Skinning CSS – an external style sheet which defines colors, the shape of components, the shape of borders, the size of the font, etc. It already includes selectors in which we can change the skinning attributes.

We can assign each component a different style by defining it in the .XHTML file. Above the component we wish to change, we add a call to the selector by using the command *»styleClass«* (or *»class«)*, while we enter the properties of the selector into the style sheet. In doing so, we have to be careful about the already defined attributes in selectors. If necessary or possible, we override the attributes of individual components via custom styles. We can also use the command *»style«,* through which we can use style commands in the .XHTML file itself.

The PrimeFaces documentation is quite meager when it comes to creating themes, since it only describes the architecture and adding themes to an application. In our application, we generated a basic theme with the help of the JQuery ThemeRoller, we predefined selectors in the style sheet, and added custom selectors (e.g. *labelLogin*, *button*, *button:hover*). We designed other widgets used in the application similarly. A practical example of predefining the shape of a framework is presented in Image 4.

---

[7] JQuery ThemeRoller: http://jqueryui.com/themeroller/

[8] We need to change the path to an image. Example: There is a path to an image in an exported theme *–images/ui-bg_highlight-soft_75_cccccc_1x100.png–*, which needs to be changed to: *#{resource['primefaces-yourtheme:images/ ui-bg_highlight-soft_75_cccccc_1x100.png']}*.

Image 4: Editing the shape of a login form

PrimeFaces itself does not support responsive design. This can be solved by including frameworks which help adjust display across different-sized screens (Bootstrap, Fundation, Skeleton, and others). The developers of PrimeFaces recommend using the TwitterBootstrap framework.

2.3.3. PrimeFaces and support for filtering, sorting, and lazy loading

We often use tables to present data on web pages. In our opinion, *<p:dataTable />* is one of the most perfected widgets in PrimeFaces since it supports numerous functionalities: editing data within a table, labeling one or more lines, sorting, filtering, lazy loading, etc. As long as we are working with a small amount of data (e.g. a few dozen lines) and need only filtering and sorting, we can use the pre-implemented methods in PrimeFaces. The two methods are executed on application server level as shown in the code segment below [17].

```
<p:dataTable var="car" value="#{tableBean.cars}"paginator="true"
             rows="10" filteredValue="#{tableBean.filteredCars}">
  ...
    <p:column headerText="Model" sortBy="model" filterBy="model">
      #{car.model}
    </p:column>

    <p:column headerText="Year" sortBy="year" filterBy="year">
        #{car.year}
    </p:column>
  ...
```

Image 5: A code segment which shows how sorting, filtering, and pagination are defined

As can be seen in the *<p:dataTable />* tag in the image above: we simply add pagination (XML attributes *paginator* and *rows*), define the list in which PrimeFaces stores the filtered values (attribute *filteredValue*), and define in each column (tag *<p:column />*) by which field it is sorted (attribute *sortBy*) or filtered (attribute *filterBy*). For all these functionalities, there needs to be no additional implementation in the managed bean, apart from defining a temporary variable for storing the filtered list and the variable's *getter* and *setter* methods. On the other hand, whenever we wish to assume control over the sorting, filtering,

pagination, and/or when we are working with large amounts of data, we can use PrimeFaces' lazy loading interface. It is a Java class with a pre-defined method into which PrimeFaces delivers data about pagination, sorting, and filters. In case of a table, we inherit from the class *org.primefaces.model.LazyDataModel* and implement the method *loadData*, as shown in Image 6[18].

```
public class LazyCarDataModel extends LazyDataModel<Car> {

  @Override
  public List<Car> load(int first, int pageSize, String sortField,
          SortOrder sortOrder, Map<String, String> filters) {

    // Implementacija nalaganja podatkov - npr. preko JPA

    return null;
  }
}
```

Image 6: PrimeFaces' interface for implementing lazy loading

The example in Image 6 shows that the mode of data acquisition (e.g. via JPA, pure JDBC[9], or a web service call) is hidden in PrimeFaces. If we want to implement lazy loading into the *<p:dataTable />* tag seen in Image 5, we simply add the attribute *lazy='true'*, remove the attribute *filteredValue* and upgrade the managed bean so that it returns the instance of the *LazyDataModel* class. The rest of the class's XHTML code remains unchanged.

2.3.4. PrimeFaces extensions and OmniFaces

The "PrimeFaces Extensions" open source project is being run in parallel with the PrimeFaces project. It includes graphic widgets missing in PrimeFaces. Among the more interesting ones are: the incredibly rich and adaptable HTML editor (tag *<pe:ckEditor />*), the component for selecting time (tag *<pe:timePicker />*), and the timeline (*<pe:timeLine />*). At registration in our application, the user must accept the basic terms and conditions for its use, while the owner of the application wants a simple way to edit these terms and conditions. We united the appearance of terms and conditions at registration with the appearance in the administrative module by using the *<pe:ckEditor />* component. This component is displayed in read-only mode at registration, while in the administrative module it can be edited.

Finally, we will present OmniFaces. OmniFaces is a library which combines a bundle of accessories that make it easier to develop applications in the JSF framework[22]. We used OmniFaces in our application in order to solve problems described in the third paragraph of point 2.1.4.

2.4. JPA

Enterprise Java Beans (EJB) are Java classes executed on the Java EE application server to which we add one or more annotations in accordance with the EJB specification. EJBs can be seen as widgets with which we can implement a business logic level into a Java EE application. There are two major types of EJBs: message driven beans and session beans. Message driven beans are mostly used in connection to the Java Message Service (JMS). Session beans are used for direct communication between a bean and its user (e.g. a CDI bean and an EJB bean). There are three types of session beans: those having state (stateful session beans), those who do not have state associated with them (stateless session beans), and those having a global shared state within a JVM (singleton session beans) [19]. If we want to implement e.g. a shopping cart, we can use a stateful session bean. On the other hand, if we want to use it as an interface for accessing a database, a stateless session bean is generally sufficient.

Java Persistence Architecture (JPA) is a Java specification which describes the management of data between Java objects and a relational database. The specification itself can be divided into four parts: the Java

[9] JDBC (Java Database Connectivity): A technology for accessing databases via the Java programming language.

Persistence API (JPA), the Java Persistence Query Language (JPQL, a language similar to SQL on the level of JPA), the Criteria API (an API for the dynamic building of queries), and metadata for object-relational mapping. JPA can be used in Java EE as well as Java SE applications. It needs to be emphasized though that JPA is merely a specification, which is why our web application must include a library implemented by JPA (e.g. OpenJPA, TopLink, EclipseLink, Hibernate, etc.).

Let us describe in short how we used JPA in our application.

- We defined the *persistence.xml* configuration file in which we configured the operation of JPA and its implementation library;
- We implemented JPA entity classes which represent images of tables from the database. JPA entities are normal (POJO[10]) Java classes in which the classes and their attributes (or even methods) have annotations which define the connection with the database (e.g. by using the annotation *@Table(name = "Table")* we state that the Java class represents a table named "Table");
- We used Java stateless session beans as components which initiate queries. These beans are recommended when using JPA in Java applications [12].

We have not come across any problems with using EJBs so far. We can only commend the simplicity of their use, since a single annotation is enough for a Java class to become an EJB. The following is our experience with using JPA:

- When an entity being queried is associated with other entities, we have to consider whether we require all the data from these associated entities – e.g. we have an entity named Invoice, which includes the list (of) Rates. By default, JPA loads all the data, which can increase the execution time of queries returning a large number of hits, and can unnecessarily increase an application's memory consumption. We turn on lazy loading in annotations with which we define associations – e.g. *@ManyToOne(fetch = FetchType.LAZY)*;
- Whenever an entity (or more associated entities) includes a large number of attributes and we need only a few, we can use the so-called *tupleQuery* [12]. This enables us to query only certain attributes in one or more entities. For example, a schedule of auctions does not require all the arrays which are otherwise used in an overview of auctions. This, too, is one of the application's optimization options;
- Some databases do not include operators for specifying which lines of an SQL[11] query result we require (the mySQL database includes two: *LIMIT* and *OFFSET*). We do not have to deal with this problem when using JPA. Let us take as an example Oracle's database, for which the so-called "N-tier" queries are recommended when implementing pagination. EclipseLink, which we use in our application, follows this recommendation;
- Java EE 7 also includes the 2.1 release of JPA, which brings with it the possibility to call stored procedures via a standardized interface (Oracle, 2013). Oracle's database enables the implementation of functions and procedures inside the database. We called the procedure via this interface with no issues. For a function, though, we could not find a solution; using EclipseLink classes could work because they support it, however they are not part of the JPA specification [27].

2.5. Apache Shiro

There are two approaches to implementing security in Java web applications. The first is "container-managed-security", which uses security mechanisms of the application server on which the application is running. For the simplest of cases it suffices to properly adjust one or more configuration files in the web application. The problem with this approach, though, is that the configuration and collection of security mechanisms differs from one application server to another, which means that the application is not portable between these servers. The second possible approach is using "application-managed-security", which

[10] POJO (Plain Old Java Object): a Java object not bound by any restriction, with the exception of those defined by the Java Language Specification.

[11] SQL (Structured Query Language): A programming language used to extract data out of a database.

implements security on application level, mostly through the use of a security library. Such libraries include Apache Shiro, Spring Security, and Security Filter [21] [25].

Apache Shiro is a framework which enables us to implement security in Java applications. It covers four important areas of application security: authentication, authorization, encryption, and a session manager. The library can be used both in Java SE and Java EE applications. In order to check authentication data and load permission we use "realms" which act as an interface to the authentication data. Supported realms are: the configuration file (enclosed with the application), JDBC, Active Directory, and LDAP[12]. It is possible to contact multiple realms during the authentication process, to define an authentication strategy (e.g. the authentication must function properly in one of the realms on hand), and to read roles and permissions from a separate realm. Apache Shiro uses a system of roles and permissions which enables us to thoroughly check a user's permissions – e.g. we can check whether a user has the A role and the permissions B and C (the operator "and" can optionally be replaced with "or" as well). By using this system we can also avoid defining large groups of roles.

We learned about the Apache Shiro library by reading Bauke Scholtz's blog. Bauke is better known to the internet community for his nickname BalusC and his implementation of the OmniFaces library [24]. His blog is an invaluable source of knowledge from the field of JSF, as well as Java EE in general. One of his blog entries includes a tutorial on how to integrate Apache Shiro into a JSF application; we followed this tutorial as well. Scholtz points out three main weaknesses of the Apache Shiro library in relation to JSF and Java EE: the problem with AJAX requests when a session is terminated, the lack of options when using annotations to define security on the level of CDI and EJB beans, and the lack of tags for using security on the level of JSF beans. Solutions (AJAX and annotations) are provided for the first two problems, and we can confirm that they work perfectly. Image 7 shows an example of how to protect a CDI managed bean method [24].

```
package com.example.controller;

import javax.enterprise.context.RequestScoped;
import javax.inject.Named;

import org.apache.shiro.authz.annotation.RequiresRoles;

import com.example.interceptor.ShiroSecured;

@Named
@RequestScoped
@ShiroSecured
public class SomeBean {

    @RequiresRoles("ADMIN")
    public void doSomethingWhichIsOnlyAllowedByADMIN() {
        // ...
    }

}
```

Image 7: An example of protecting a CDI bean method (retrieved from BalusC's blog)

To add: implemented annotations do not function with managed beans (they were deprecated with the arrival of Java EE7). Here is some of our experience with using the library:

- The documentation of the project is incomplete, and the home page is not updated. The 1.2.3 version was released when this article was being written, but we could not yet find the list of differences between this

[12] LDAP (Lightweight Directory Access Protocol): a standard for accessing and maintaining distributed directory information services over an Internet Protocol (IP) network.

version and the previous one (1.2.2). The only good aspect of the documentation is the description of the architecture and widgets (e.g. the realm) used by the framework;

- The architecture of the framework is understandable and easily extensible. Once we looked over the documentation, some web sources, and source code comments, we were able to implement certificate authentication without difficulty;
- There is an absence of tags for using Apache Shiro in JSF's XHTML files. We solved this issue for the moment by creating a managed session bean with methods which can be called by JSF pages via EL expressions. The methods delegate the call to the framework – more specifically the *org.apache.shiro.subject.Subject* class;
- In the configuration file, we can specify which roles have access to certain addresses (e.g. /admin/admin.xhtml); if we specify a number of roles for one address, the framework reads it as if access to the address requires not just one, but all of those roles. So, what is missing is the support for an "or" operator in configuration files;
- While testing the library, we learned that the framework has no built-in protection against session cookie theft – if an attacker manages to intercept a session cookie, he can use the application without logging in (if, of course, the session for that cookie is still active);
- The framework lacks the option to define multilevel authentication – e.g. the user enters their username and password, then they receive a one-time password via some other channel (SMS, e-Mail); they enter the one-time password and the already-prepared components in order to perform certificate authentication.

2.6. Quartz Scheduler

The Quartz library is used to create time schedules of jobs in an application (executing certain segments of code at certain time intervals). It can be integrated into all types of Java applications (Java SE, Java EE). The library is independent from the application server and supports multithreading.

The library's main functionality is the Scheduler interface, which implements simple operations for scheduling jobs at certain repeating or non-repeating intervals (triggers). "Jobs" are the code segments we wish to execute. Each one is executed in its own thread, and they can be stopped or restarted. We can add input and output parameters to the jobs, and arrange jobs into developer-friendly groups. When we enter jobs into the scheduler, the data about jobs and triggers (input data, run time, etc.) is stored into either:

- The system memory (RAMJobStore): The advantages of this approach are simple configuration and fast execution, while the weaknesses are memory consumption (an increased number of jobs and triggers) and loss of data when an application error occurs or when the system is restarted.
- Or the database via JDBC (JDBCJobStore): Storing is compatible with different databases (Oracle, PostgreSQL, MySQL, MS SQL Server,...).

The library has proven itself to be very adaptable since the jobs can be run in any sort of time sequence. By using it we were able to send emails to the users of our application at registration, whenever an auction started, and whenever they forgot their passwords. We also plan to implement certain jobs with which we could run security copies of the system daily at night time.

2.7. JodaTime

The date library JodaTime is a quality replacement for the Date and Time API. Its architecture enables us to create different types of calendars (Gregorian, Julian, Buddhist, Ethiopian, Islamic) while the library itself remains simple to use. It enables us to use different time zones connected to the tz (Time Zone) database which is updated several times a year. JodaTime supports time periods and operations between them, different date formats, and different types of parsing. It uses the Apache License, Version 2.0 which means that it can be incorporated in applications freely [11].

We, of course, tested both libraries (Date and Time API, and JodaTime) before implementing one into our project. In test projects of both libraries (as shown in Image 8) we created a million instances of the Date

class and changed the date format into the Slovene one. With these tests we wanted to determine the test code's run time in each of the two libraries. The first test was executed using a "formatter" (SimpleDateFormat, DateTimeFormatter) implemented outside the "for" loop; in the second test, we implemented both inside the "for" loop.

```
  public static void main(String[] args)
  {
    Date date = new Date(System.currentTimeMillis());
    long start = System.currentTimeMillis();
    for(int i = 0;i < 1000000; i++){
      SimpleDateFormat sdf = new SimpleDateFormat("dd.MM.yyyy");
      String datum = sdf.format(date);
    }
    long end = System.currentTimeMillis();
    System.out.println("Time is:"+ (end - start));
  }

}
```

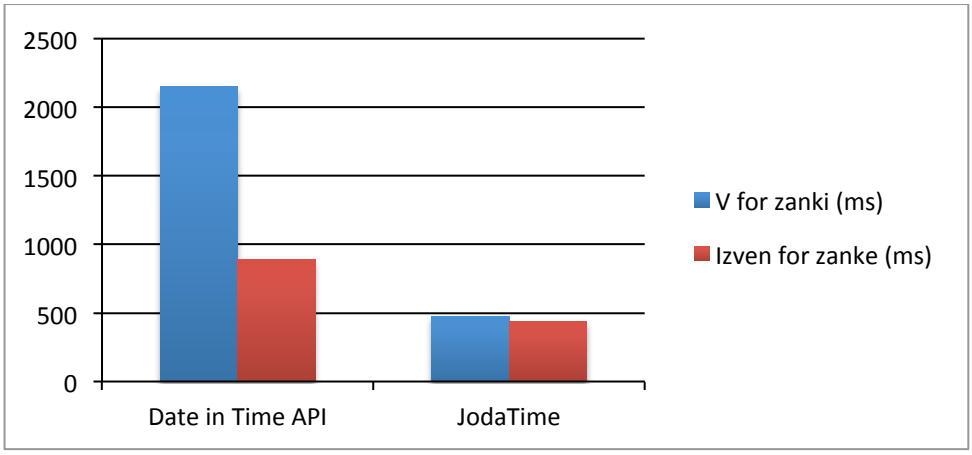
```
  public static void main(String[] args)
  {

    DateTime dt = new DateTime(System.currentTimeMillis());
    long start = System.currentTimeMillis();
    for(int i = 0;i < 1000000; i++){
      DateTimeFormatter fmt = DateTimeFormat.forPattern("dd.MM.yyyy");
      String date1 = dt.toString(fmt);
    }
    long end = System.currentTimeMillis();
    System.out.print("Time is: " + (end - start));
  }

}
```

Image 8: A comparison of the Date and Time API library (above) with the JodaTime library (below)

In order to test them, we ran both classes five times and noted the run time. We then calculated the average execution time (in milliseconds), as seen in Graph 1.



Graph 1: Comparison of date classes in Java

From looking at the graph, we see that the JodaTime library helps achieve much better run times compared to the Java Date and Time API. The announced update of the Java environment (Java SE 8) will bring with it a new library for operating with date and time (java.time API); the updated library was developed in association with the creators of the JodaTime library.

The JodaTime library helped us operate with dates in our application; we used it when implementing a data type converter; its ability to calculate intervals between periods helped us manage the schedule of auctions. Because of its simplicity and responsiveness, the library is a good addition to the Java programming language.

2.8. Google libphonenumber

Validating entered telephone numbers can be a problem, because we need to verify syntactic correctness and country calling codes in order to do it. For this reason, developers at Google implemented a library which takes care of parsing, converting, storing, and validation of telephone numbers from around the world (stationery and mobile ones). It is implemented in all major programming languages (Java, C++, JavaScript, PHP, Objective-C). The Java version is adapted for use in developing smartphone applications.

The library provides us with the following functionalities:

- Extraction of data about telephone numbers (from the telephone number, the implementation extracts data about the network operator or the device itself, wherever possible);
- Extraction of data about the user's phone service provider or country;
- Verification of telephone numbers directly at input;
- Searching for telephone numbers based on partially entered telephone numbers.

We used the library mostly for parsing, converting numbers into international format, and verifying the authenticity of telephone numbers. When we verify the authenticity of a user, we therefore gain information about their service provider and regional telephone code [10].

2.9. Atmosphere Framework

It is a framework implemented in the Java or JavaScript programming language, and it is used for developing mobile asynchronous applications. It works with all browsers and all major application servers. The main concept of the framework is independent and transparent development of applications based on client-server communication channels, which means that the developer does not need to worry about compliance or proper functioning of an application in different browsers. A practical example can demonstrate how the framework operates. If a developer decides to implement the client-server connection in the form of the "WebSocket"[13] protocol, the framework will use the HTTP protocol in older browsers and the "WebSocket" protocol in newer ones [4].

In our application, we use the library to notify auction participants whenever the highest offered bid changes. When that occurs, both the bid table and the bid input mask are refreshed for everyone.

3. ARCHITECTURE OF THE APPLICATION

To summarize the architecture presented in Image 9: The database was implemented in the MySQL open source relational database. We connected the database to the application by using a JDBC connector (MySQL Jconnector). Instead of directly using the SQL language for extracting data, we used the JPA technology. On business logic level, we defined EJB controllers and entity classes which represent images of tables from the database. EJB controllers use the defined entities for operations on data, and implement business rules. The presentational level of the application is represented by JSF pages, which consist of XHTML files (page structure) and CDI managed beans. The beans are used to implement supporting operations for the server-side presentational level, and to establish a connection to the application's business level.

[13] WebSocket: a protocol providing full-duplex (in both directions) communications channels over a single TCP (Transmission Control Protocol) connection. It is used for communcation between the browser and the web page.

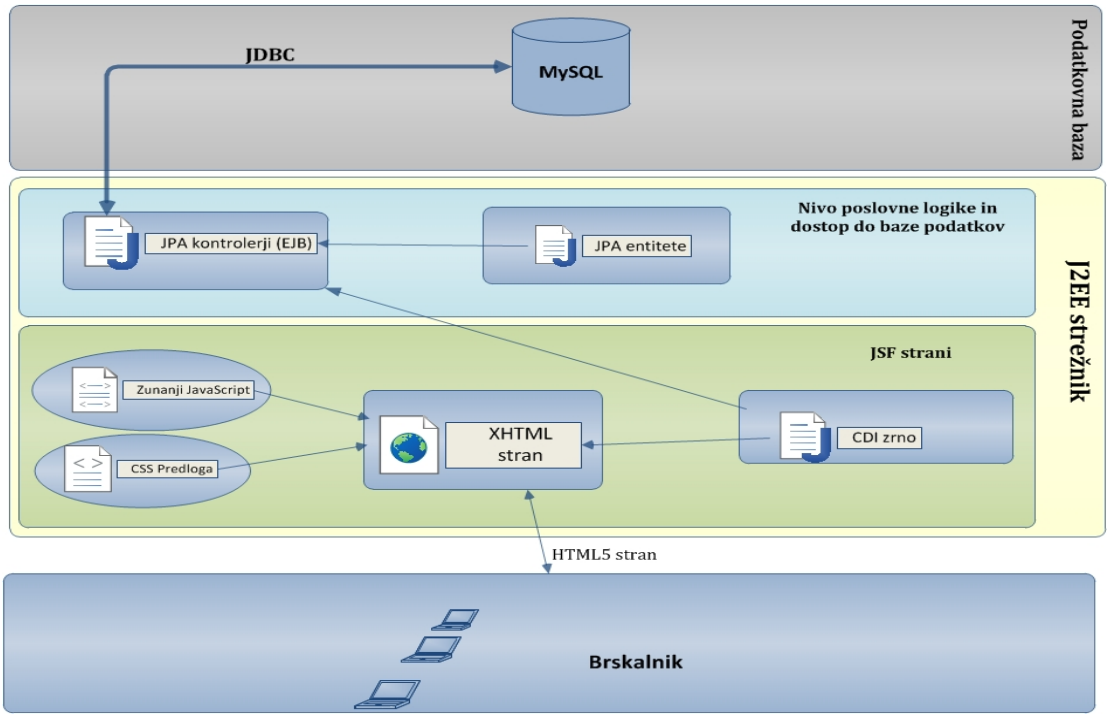


Image 9: The structure of an application for lumber auctions

4. CONCLUSION

We chose the Java EE platform and we believe that it meets our requirements. We were mostly convinced by the standardization on every step of the way (e.g. how to implement validation of input data), the support for solving common issues (e.g. pagination in JPA), and extensibility (e.g. the option to extend JSF tags for graphic components). When JSF "ran out of power" we used PrimeFaces, which includes a truly rich collection of graphic components. We believe PrimeFaces has a future due to its vibrant community, fast development of the framework, as well as the authors' decision to create JSF widgets for mobile applications (PrimeFaces Mobile) and independent libraries of HTML5 widgets, which are identical to those widgets into which JSF widgets translate (PrimeFaces UI). The same could be said for Java EE, because version 7 brings with it the support for WebSockets (JSR-356), standardized processing of JSON (JSR-353), and so on. We only worry about the unclear future of the Apache Shiro library.

5. LITERATURE

[1] Akdogan, H. (2013, August 8). "JSF 2.2: HTML 5 Support." Retrieved May 18, 2014 from *Kodcu.com*: http://en.kodcu.com/2013/08/jsf-2-2-html-5-support/

[2] Asel, T. (2013, May 16). "JSF Performance: Mojarra improves dramatically with latest release." Retrieved May 18, 2014 from *Techscouting through the java news*: http://blog.oio.de/2013/05/16/jsf-performance-mojarra-improves-dramatically-with-latest-release/

[3] Asel, T. (2013, April 8). "JSF-Comparison: MyFaces vs. Mojarra." Retrieved May 18, 2014 from *Techscouting through the Java news*: https://blog.oio.de/2013/04/08/jsf-comparison-myfaces-vs-mojarra/

[4] Async-io.org. (2014, April 29). "Introduction." Retrieved May 16, 2014 from *Atmosphere/Atmosphere Wiki*: https://github.com/Atmosphere/atmosphere/wiki

[5] Breitner, D. (2014, January 15). "Comparing Primefaces, ICEFaces and Richfaces with Google Trend." Retrieved May 18, 2014 from *A blog about Liferay and JSF*: http://liferay-blogging.blogspot.com/2014/01/comparing-primefaces-icefaces-and.html

[6] Burns, E. (2010, August 8). "Component bindings incompatible with View Scope." Retrieved May 18, 2014 from *JIRA*: https://java.net/jira/browse/JAVASERVERFACES-1492

[7] Burns, E. (2014, April 10). “JSR-000344 JavaServer Faces 2.2 - Final release.” Retrieved May 17, 2014 from https://www.jcp.org/aboutJava/communityprocess/final/jsr344/index.html

[8] Cavicii, C. (2013, February 5). “New Maintenance Policy.” Retrieved May 17, 2014 from *PrimeFaces Blog*: http://blog.primefaces.org/?p=2443

[9] Gibson, A. (2012, February 7). “Comparing JSF Beans, CDI Beans and EJBs.” Retrieved May 18, 2014 from *Andy Gibson: Open Source Projects & Technical Writings:* http://www.andygibson.net/blog/article/comparing-jsf-beans-cdi-beans-and-ejbs/

[10] Google. (2013, September 24). “Google's phone number handling library, powering Android and more.” Retrieved May 15, 2014 from *Libphonenumber*: https://code.google.com/p/libphonenumber/

[11] Joda.org. (2013, August 3). “Java date and time API.” Retrieved June 17, 2014 from *JodaTime*: http://joda-time.sourceforge.net/quickstart.html

[12] Keith, M., & Schincariol, M. (2009). *JPA2: Mastering the Java Persistence API.* New York: Apress.

[13] Kyte, T. (2007, January 15). “On Top-n and Pagination Queries.” Retrieved May 16, 2014 from *Ask Tom*: http://www.oracle.com/technetwork/issue-archive/2007/07-jan/o17asktom-093877.html

[14] Marchioni, F. (2010, April 14). “Java EE 6 training. Part 3: Richfaces.” Retrieved May 18, 2014 from *JBoss Aplication Server Tutorials*: http://www.mastertheboss.com/richfaces

[15] Oracle. (2013, October 10). “Applying JPA to Stored Procedures.” Retrieved May 16, 2014 from *Java EE7*: http://www.oracle.com/webfolder/technetwork/tutorials/obe/java/Stored_Proc/StoredProcedures.html

[16] Oracle. (2014, October 4). “The Java EE7 Tutorial: Java Server Faces Technology.” Retrieved May 15, 2014 from *Java EE Documentation*: http://docs.oracle.com/javaee/7/tutorial/doc/jsf-intro.htm#BNAPH

[17] PrimeTek. (2013, April 10). “Complex datatable.” Retrieved May 16, 2014 from *Primefaces Showcase*: http://www.primefaces.org/showcase/ui/datatableComplex.jsf

[18] PrimeTek. (2013, April 10). “Primefaces datatable lazy parsing.” Retrieved May 16, 2014 from *Primefaces showcase*: http://www.primefaces.org/showcase/ui/datatableLazy.jsf

[19] Reese, R. M. (2011). *EJB 3.1 Cookbook.* Birmingham: Packt publishing.

[20] Riem, M. (2012, August 7). “JSF page performance degrades significantly as page size increases.” Retrieved May 18, 2014 from *JIRA*: https://java.net/jira/browse/JAVASERVERFACES-2494

[21] Scholtz, B. (2013, January 24). “Apache Shiro, is it ready for Java EE 6? (a JSF2-Shiro Tutorial).” Retrieved May 16, 2014 from *The BalusC Code*: http://balusc.blogspot.com/2013/01/apache-shiro-is-it-ready-for-java-ee-6.html

[22] Scholtz, B. (2012, May 1). “OmniFaces Showcase.” Retrieved May 16, 2014 from *OmniFaces Showcase*: http://showcase.omnifaces.org

[23] Scholtz, B. (2010, June 5). “The benefits and pitfalls of @ViewScoped.” Retrieved May 18, 2014 from *The BalusC Code*: http://balusc.blogspot.com/2010/06/benefits-and-pitfalls-of-viewscoped.html

[24] Scholtz, B. (2007, August 31). “Welcome.” Retrieved May 16, 2014 from *The BalusC Code*: http://balusc.blogspot.com

[25] StackHunter. (2014, May 5). “Ditch Container-Managed Security To Create Portable Web Apps.” Retrieved May 16, 2014 from *StackHunter*: http://blog.stackhunter.com/2014/05/05/ditch-container-managed-security-to-create-portable-web-apps/

[26] Terracotta inc. (2013, August 3). “Quartz Quick Start Guide.” Retrieved May 18, 2014 from *Quartz scheduler*: http://quartz-scheduler.org/documentation/quartz-2.2.x/quick-start

[27] The Eclipse fundation. (2011, December 19). “EclipseLink/Examples/JPA/PLSQLStoredFunction.” Retrieved May 17, 2014 from *Eclipsepedia*: http://wiki.eclipse.org/EclipseLink/Examples/JPA/PLSQLStoredFunction