Type of the Paper: Original scientific paper
Received: 23. 12. 2021.
Accepted: 21.1. 2022.
DOI: https://doi.org/10.18485/edtech.2022.2.2.1
UDC:

Designing and implementing the "Education Support" web application using Spring and Hibernate frameworks

Vladimir Bošković [1], **Svetlana Jevremović** [1]

[1] ITS – Information Technology School, Zemun, Belgrade, Serbia
* vladimir2417@its.edu.rs, svetlana.jevremovic@its.edu.rs

Summary: In this paper, we present the process of developing and implementing the web application "Education Support" using Spring and Hibernate frameworks. The objective of the paper is to explore the modes of designing and implementing this web application that aims to provide an accessible way of searching and selecting between various educational institutions, as well as to provide an easy way for educational institutions to communicate with their potential clients. This application aims to enable users to collect information, plan ahead and secure their enrolment with special benefits. Educational institutions would benefit from this app business-wise, thanks to having an efficient and user-friendly list of their subscribers, along with a list of the requests to schedule an informative visit to their institution, with the option to send promotional messages to users. We chose this subject as we noted that no such web application is available on our market. We have applied Larman's methodology in each development phase of this web application.

Keywords: web-application; Spring; Hibernate; Larman's methodology

1. Introduction

The "Education Support" web application is based on the current Java web technologies Spring and Hibernate. Therefore, to design this application, for the server side, we used the Spring framework, including Servlets, Maven, Hibernate and Thymeleaf. For the client side, we used HTML, CSS, JS, and libraries JQuery and Bootstrap. We used IntelliJ IDEA Ultimate as our development environment and Servlets for the database.

The first part of the paper presents the Spring framework, with a special emphasis on Spring MVC module concepts, which we used for the development of the application "Education Support". The second part of the paper is dedicated to the subject of object-relational mapping and data persistence through using the Hibernate ORM (object-relational mapping) tool. The central part of the paper is dedicated to the development of the web application "Education Support".

While developing this application, we used Larman's software development methodology during the planning, analysis, design, implementation and testing [10]. Larman's methodology is based on an iterative and incremental model of software lifecycle; it consists of Use cases and utilises object-oriented development methodology with UML (Unified Modelling Language) diagrams. This methodology is software-developer-oriented, since its focus is on the analysis and design, and it is unique for its operation contracts [8].

2. Technologies applied

Before we analyse the design and implementation documentation for the web application "Education Support", we will analyse the technologies we used during the phases of the web application development. We focused on the characteristics and solutions provided in the Spring framework, in the context of web application development.

2.1. The Spring framework

One of the most important characteristics of the Spring framework lies in its modularity which allows programmers to use only the modules that are required in a specific situation instead of the entire framework. Thanks to the modular approach, Spring can be used to develop Java applications of various levels of complexity. Depending on the application requirements, one can use any Spring module, independently from the rest of the modules. For instance, we can use the Spring core container to manage the business logic of the application and create other parts of the application by using some other technology. In addition to its modularity, Spring also allows integration with various other technologies.

The Spring framework architecture consists of the following six modules and can be seen in Figure 1 [5]:

» Core;
» AOP (aspect-oriented programming);
» DAO (Data Access Objects);
» ORM (object-relational mapping);
» JEE (Java Enterrrise Edition);
» Web.

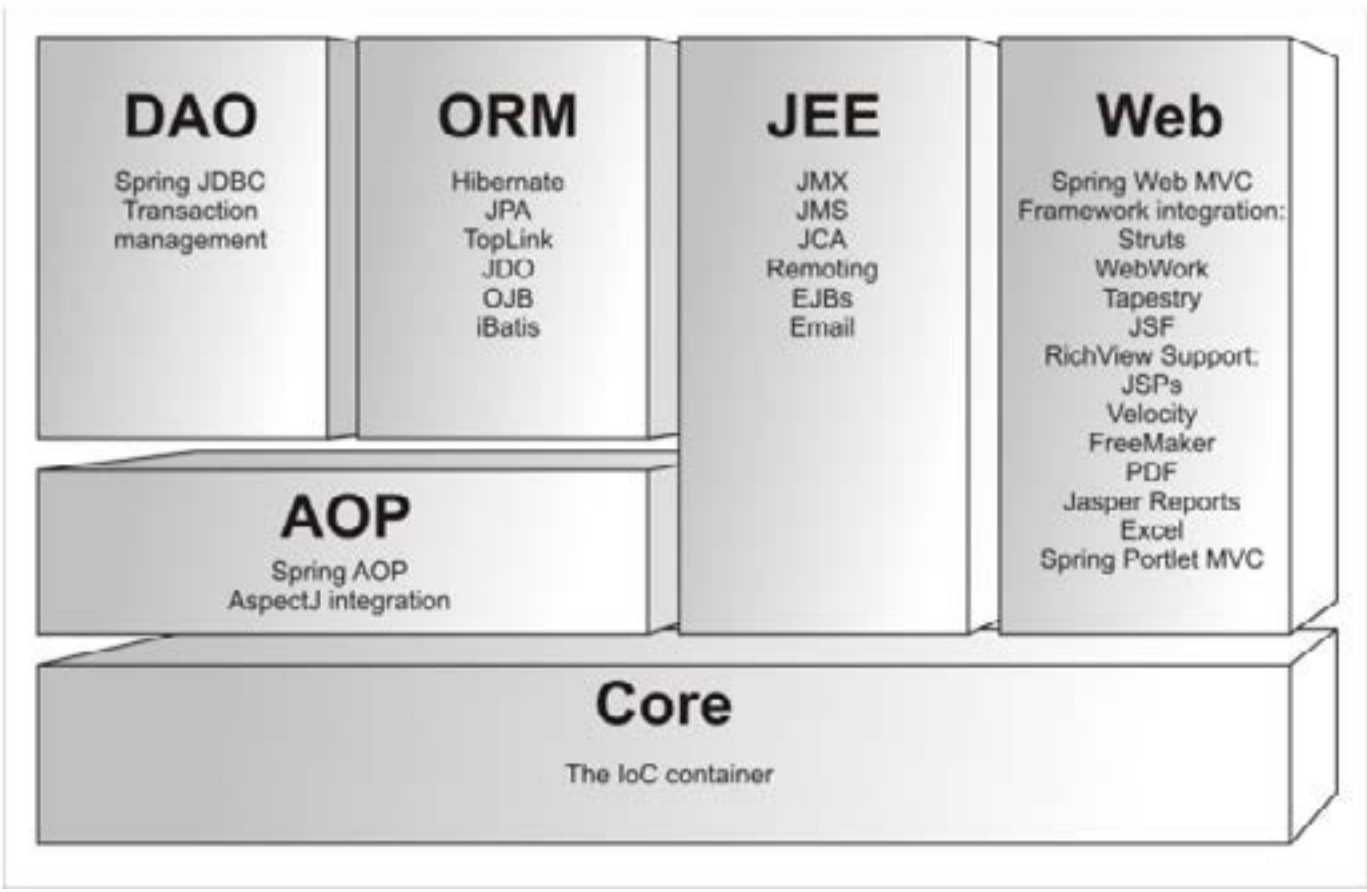

Figure 1. Spring framework modules [5]

Core module is the central module of the Spring framework. The Inversion of Control (IoC) mechanism, i.e. dependency injection (DI) mechanism, is realised in the Core module. The realisation of the DI mechanism in the Core module takes place via the BeanFactory interface, i.e. through the implementation of this interface. The AOP module supports aspect-oriented programming that allows the developer to easily separate the components of the system by implementing logically separated functionalities. The DAO module provides the JDBC (Java Database Connectivity) layer that removes the need for traditional database access by using JDBC management software. The use of JDBC implies writing the code for opening a connection, creating statements, processing results and for closing a connection; however, with the Spring JDBC framework, the entire job is abstracted and simplified. The ORM module provides a layer for integrating the Spring framework with popular ORM tools. In addition to the Hibernate framework, whose integration with the Spring framework we used in this paper, Spring also allows integration with the following ORM tools: iBatis SQL Maps, JDO, Apache OJB, Oracle TopLink. In addition to the integration with these ORM tools, the ORM module enables the use of declarative transaction management. The JEE module enables the integration of the Spring framework and EJB (Enterprise JavaBeans) components, as well as the integration with JMS (Java Message Service) components, which enables asynchronous creation, along with sending and receiving of messages. The Web module contains a complete implementation of the MVC4 framework used for web application development. Spring Web MVC implementation provides a complete separation of the application business logic from its presentation layer, at the same time enabling the use of all IoC characteristics of the framework during web application development.

2.1.1. Dependency injection

Spring IoC container is based on the dependency injection (DI) mechanism. The very name tells a lot about the relationship between the application and the container, and the way in which dependencies between application components are resolved. The application can be seen as a set of components that cooperate and depend on each other during the execution. With Java applications, these components are Java class instances, i.e. objects. Business objects of an application run by Spring IoC container are not in charge of collecting resources and creating other objects they work with and depend on; instead, the container creates and configures their dependencies, which allows us to place more focus on the business logic that needs to be executed by the class methods while creating business classes.

There are three basic types of the DI mechanism supported by the Spring container:

» setter injection;
» constructor injection;
» method injection.
»

2.1.2. Spring container

Dependency injection mechanism is the essence of the Spring framework. The implementation of this mechanism is realised through two interfaces constituting the core of the Spring IoC container:

» org.springframework.beans.factory.BeanFactory
» org.springframework.context.ApplicationContext
»

BeanFactory is the main Spring container interface that enables configuring and connecting beans by using DI mechanisms. When the BeanFactory interface is created, Spring framework conducts bean configuration validation. A singleton bean is created when the framework is launched, and other beans are created upon the user's request. In addition to this, the BeanFactory interface provides a mechanism for bean lifecycle management, but this only applies to singleton beans, since when it comes to prototype beans, the container loses control over them once they have been created.

ApplicationContext interface inherits the BeanFactory interface. Its implementation also enables bean creation and bean lifecycle management. However, it also provides support for integration with the Spring AOP module, message resource support, as well as application events propagation. In addition to this, it also provides specific application layer contexts such as the WebApplicationContext interface used in web applications.

Therefore, the BeanFactory interface enables framework configuration, as well as basic functionalities of such configuration, while the ApplicationContex interface adds new and more complex functionalities. The ApplicationContext interface constitutes a comprehensive superset of the BeanFactory interface, i.e. it enables everything that BeanFactory does. The relationship between these two interfaces is illustrated in Figure 2.

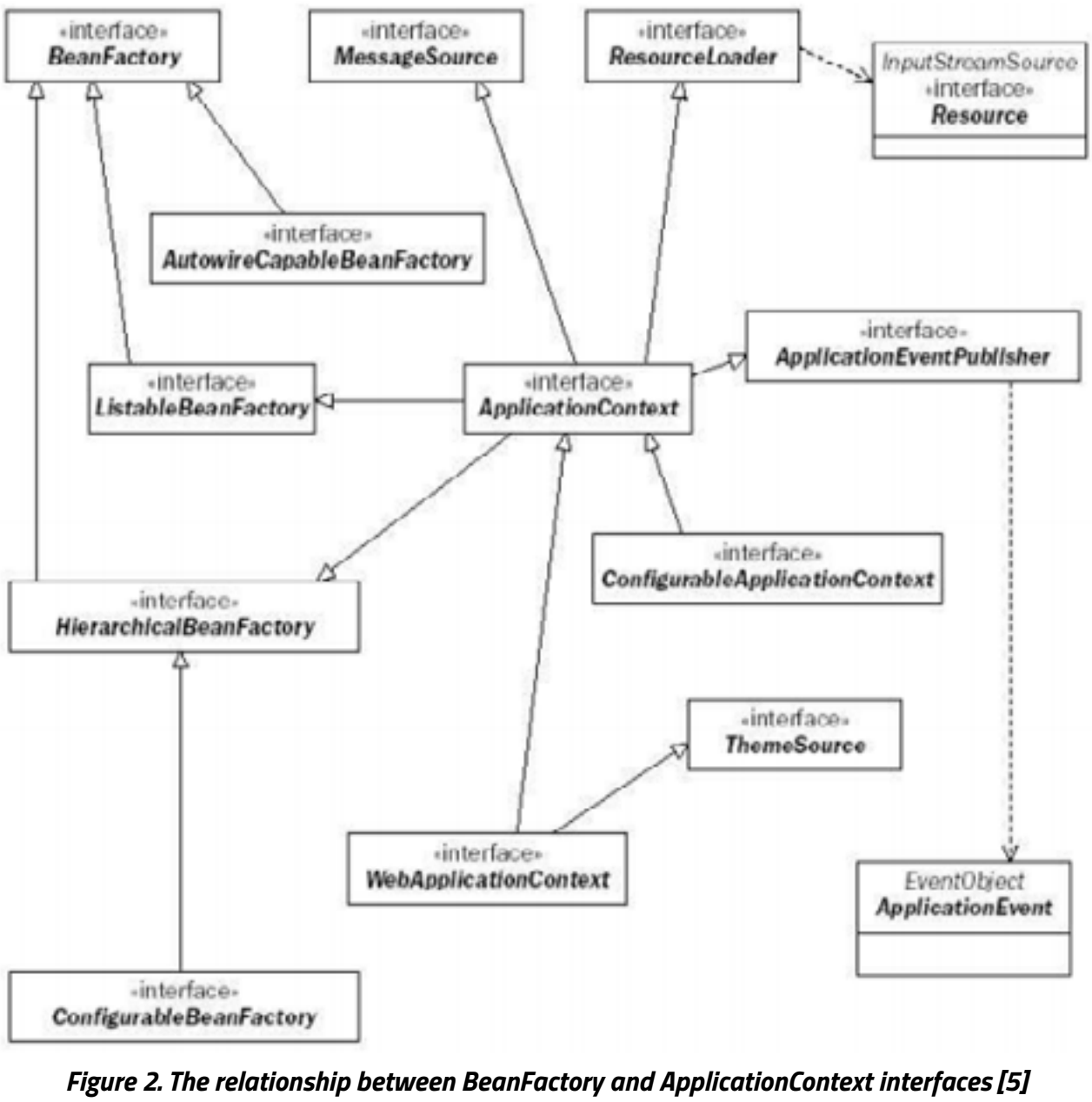

Figure 2. The relationship between BeanFactory and ApplicationContext interfaces [5]

2.1.3. Working with JavaBean components

Bean is a software component that can be reused and visually manipulated by using an appropriate tool [4]. In the context of apps based on the Spring framework, the term bean refers to any object created and managed by the Spring container [5]. When defining beans, the configuration file consists of one (root) bean element and one or two bean elements. Validation of this XML file is conducted in relation to the XML DTD file spring-beans.dtd, which describes in detail all the valid attributes and elements that the configuration file might contain.

In most cases, the bean identifier is the first element defined. It is important to have in mind that it is not necessary to define the identifier, in which case, the bean will be treated as an anonymous bean. A Bean name can be defined by using the bean element's name or id attribute. Using the id attribute while naming beans is recommended because this attribute is of the XML IDREF type. Therefore, in case other beans reference it, the XML parser will be able to decide whether the reference is valid or not. The IDREF type comes with certain limitations that make it inapplicable in some situations.

Beans are usually created with the bean constructor. When defining a bean, the name of the bean class is stated in the class attribute. When the container requires a new instance of this bean, it will internally execute an operation equivalent to using the new operator in Java code.

A bean can also be created by using the static factory method. In that case, one needs to define a class whose role will be to encapsulate the process of bean creation within a static method. Then, using the class attribute, the bean is defined, and then the factor-method attribute is used to specify the bean method in charge of creating the bean.

The next approach to creating beans is using the non-static (instance factory) method. In this case, a method from another bean already created by the container is used.

2.1.4. Spring Web MVC framework concepts

MVC framework is an architectural pattern; it consists of three key components: Model, View and Controller [6]. The Model is a component that contains the business system structure, along with its operation, i.e. it contains data and data processing operations. The View component provides a user interface via which the user communicates with the system. It also sends reports to the user. These reports are obtained from the Model. The Controller is the component in charge of managing the execution of system operations. It accepts client requests, calls an operation defined in the model and controls its execution.

The architecture of the Spring Web MVC framework implies the existence of controller servlets as central input points for all incoming requests. In the Spring MVC framework, this component is realised via DispatcherServlet class. The application business logic layer is realised via the controller component. In the presentation layer, Spring MVC supports various presentation technologies. The most prominent characteristic of the presentation layer is the possibility of realising full independence between the business logic and the concrete presentation technology.

The central component of the Spring Web MVC framework is the class DispatcherServlet, which constitutes the main entry point for any incoming request addressed at a Spring Web MVC application. This class is fully integrated into the Spring IoC container, which ensures that all properties of the Spring framework can be utilised.

Figure 3 shows the conceptual model of request processing using DispatcherServlet class as the central controller. As can be seen in the picture, the central controller is the entry point for any incoming request. Upon the reception of the request, the central controller delegates the request to the appropriate controller in charge of request processing. The controller creates a model and processes the request, and then it passes the model and the logical name of the view to the central controller. The model contains the attributes which the view is supposed to show to the client. Based on the logical name of the view, the view that will be returned to the client as a response is rendered through mapping to a concrete realisation of the view.

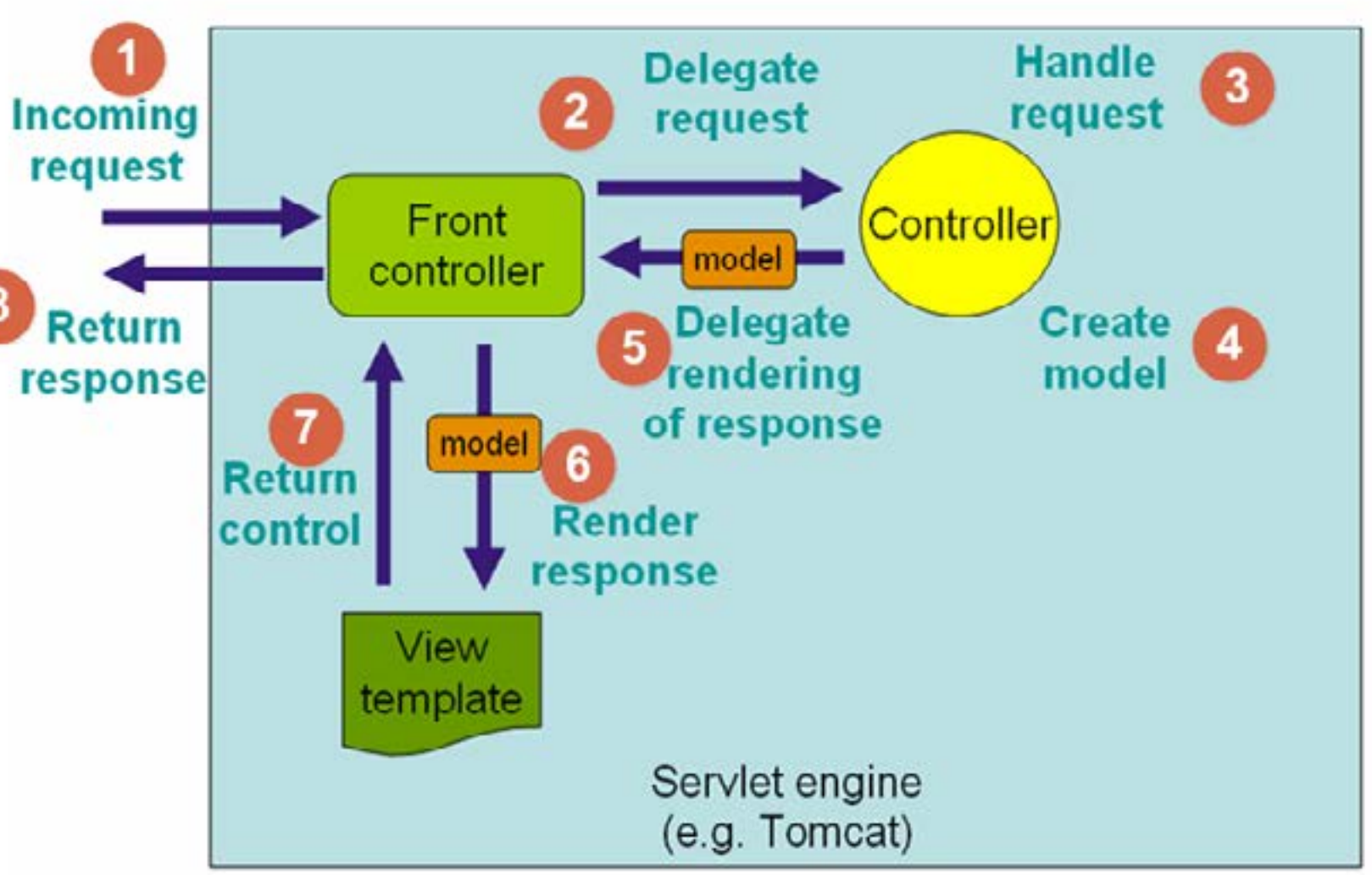

Figure 3. Conceptual model of data processing flow [9]

Class DispatcherServlet is, in fact, a servlet (derived from class javax.servlet.http.HttpServlet). It is the only servlet that needs to be declared and configured in a web application's deployment descriptor. In addition to servlet declaration, the deployment descriptor should contain the definition of request mapping to the central controller (class DispatcherServlet).

2.2. Hibernate framework

Data persistence is one of the fundamental concepts of software system development. Generally, data are said to be persistent if they outlive the programme that created them. There are several definitions that concern data persistence in the context of object-oriented software development [6].

»
» An object is persistent if it can be materialised and dematerialised.
» An object is persistent if it continues to exist after the programme that created it stops working (G. Booch).
» Materialisation is the process of transforming database syllables into objects of the programme.
» Dematerialisation is the process of transforming object s from the programme into database syllables.
» A persistent framework is a set of interfaces and classes that provides persistence to objects of different classes.

The most popular form of data storage in today's apps is using relational databases, and data persistence in Java applications usually implies storing Java objects in a relational database. Relational databases have become a standard of sorts in the realm of data persistence.

In object-oriented applications, persistence ought to provide storing of objects in a relational database. This does not refer merely to storing individual objects, but to storing an entire network of interconnected objects that represents an object model. In addition to permanently stored objects, object-oriented apps also contain a large number of the so-called transient objects. Transient objects are objects whose duration is limited by the duration of the app that created them. Such apps usually contain a subsystem in charge of materialising and dematerialising persistent objects, i.e., their transformation into a form conducive to storage in relational databases – relational model. Therefore, persistence in object-oriented apps that use a relational database can be viewed as a process of transforming the object model into a relational one and vice versa.

2.2.1. Associations

The problem of associations concerns the transformation of the relationships established between objects in the object model and the relationships created among the relations in the relational model. In the relational model, these connections are realised using external keys: an external key in a referencing table represents a primary key in a referenced table. In the object model, there are several types of associations: 1-1 (one-to-one), * - * (many-to-many), 1 - * (one-to-many). In the relational model, these associations need to be provided using external keys [7].

Transformation is usually performed by representing two objects by a single relation. In this way, one object's data is expanded by another object's data, and they become a unique relation. The attributes of the relationship become attributes of both objects.

Many-to-many transformation is performed by creating, in addition to the relationships created for each object type, an additional, aggregated relationship comprising the primary keys of the relations in question. When an object is creating a relationship with several objects of a certain type, the transformation is performed by creating separate relationships for each of these object types, and the primary key of the object that creates the relationship is memorised for all objects it is connected to, i.e., the primary key of the relation on the one side is represented as the external key of the relation on the many side.

2.2.2. Hibernate framework architecture

The Hibernate framework is a very flexible tool that provides a choice between several different approaches when it comes to choosing the framework's service that will be used in the development of an application. The three major services (components) of the Hibernate framework are connection management, transaction management, and object-relational mapping.

Figure 4 shows the two basic architectures: Lite and Full Cream. This classification is based on the components of the framework that are used in app development. Lite architecture uses only the components for object-relational mapping, while transaction management and the provision of JDBC connections are left to the app. Full Cream architecture uses all three components.

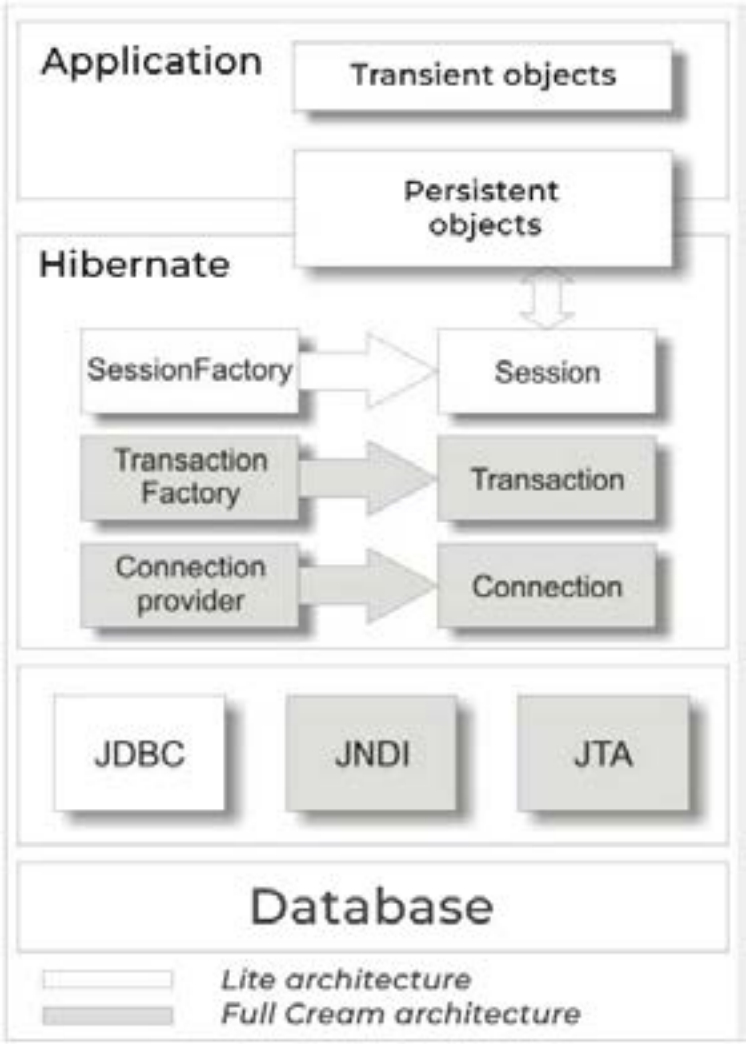

Figure 4. Hibernate framework architecture

2.2.3. Configuring the Hibernate framework

As a persistent framework, Hibernate is able to communicate with numerous different SUBPs and can be executed in various environments. Hibernate's adaptability to different SUBPs and environments in which it is executed requires different modes of framework configuration. Regardless of the environment and the database which the application communicates with, framework configuration can be logically divided into two parts. In the first part, there is the configuration data that the framework needs in order to access the database, and the second part consists of configuration data that enables mapping between the application's persistent classes and the relevant tables in the relational database. The central class used for configuring and starting the Hibernate framework is the class org.hibernate.cfg.Configuration. On creation, this class has to be provided with configuration data based on which the Configuration object will create a singleton of the SessionFactory class (Figure 5).

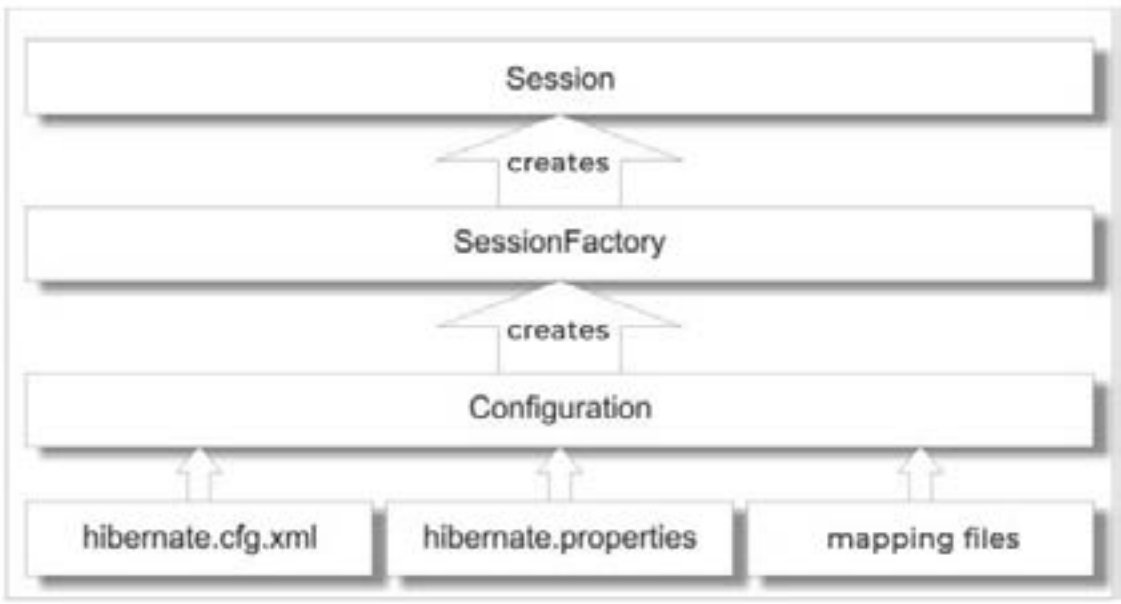

Figure 5. Configuring the Hibernate framework

A singleton of the SessionFactory class essentially constitutes a completely configured Hibernate framework that provides communication with a single database. In addition to the Configuration object creating a singleton of the SessionFactory class, the state of this object becomes unchangeable after the creation. The SessionFactory class is in charge of creating objects of the Session class during each interaction with the database.

3. Designing and implementing the web application "Education Support"

The research of the suitable technologies was followed by detailed documentation. First, we will present a verbal description of the model, and we will use it to define the system actors' Use cases.

3.1. Request specification

A web application which would help its users find and select a suitable educational institution needs to be designed and implemented. Figure 6 shows three types of actors: user, administrator and institution.

The system has to enable the user to register on the website if the user is not already registered. After registering, the user will receive a welcome message with the option to log in to the website. After logging in, the user would be able to search through the advertisements and assign several search criteria on the homepage, and then they would see a list of all the advertisements that match the parameters the user has given. If the users are interested in an ad, when they click on it, they can view the ad in detail, as well as learn some details about the institution. An additional possibility is a free subscription to a certain institution, where the users confirm that they would like to receive promotional emails from the institution. With each subscription, the user collects points. After accumulating a certain number of points, the user receives an electronic voucher as a present. This voucher allows them to receive a discount on the scholarship fee for certain institutions. If they would like to learn more about an institution, they can schedule a visit to the institution, with the realisation dependent upon the confirmation by an employee of the institution. If the institution does not confirm the request 24 hours before the tour, the request is automatically deleted.

The app should allow the system administrator to add new institutions, change information about an institution and remove institutions. On adding a new institution, an email would automatically be sent, containing login parameters. The administrator would also be able to change and delete ads and users.

The institution receives login parameters from the app administrator via email. After login, the institution can publish, change and delete ads. Also, if a user subscribes to an institution before opening its ad, the institution, once logged in, can see which users have logged in, and send them a promo message. If a user has requested a tour of the institution, the institution can accept or reject the request. If the institution accepts a request, the user will receive an email informing them that the request has been accepted; otherwise, they will receive an email notifying them that the request has been denied.

3.1.1. Use cases

Based on the verbal model, the following Use cases have been identified: Login, Registration, Add search, Add view, Subscription to an institution, Sending a visit request, Adding new institutions, Institutions overview, Changing institution info, Deleting institutions, Adding new ads, Changing ad information, Deleting ads, Users overview, Deleting users, Subscribed users overview, Sending promotional messages, Visit requests overview, Approved visit requests overview, Accepting visit requests, Denying visit requests (Figure 6).

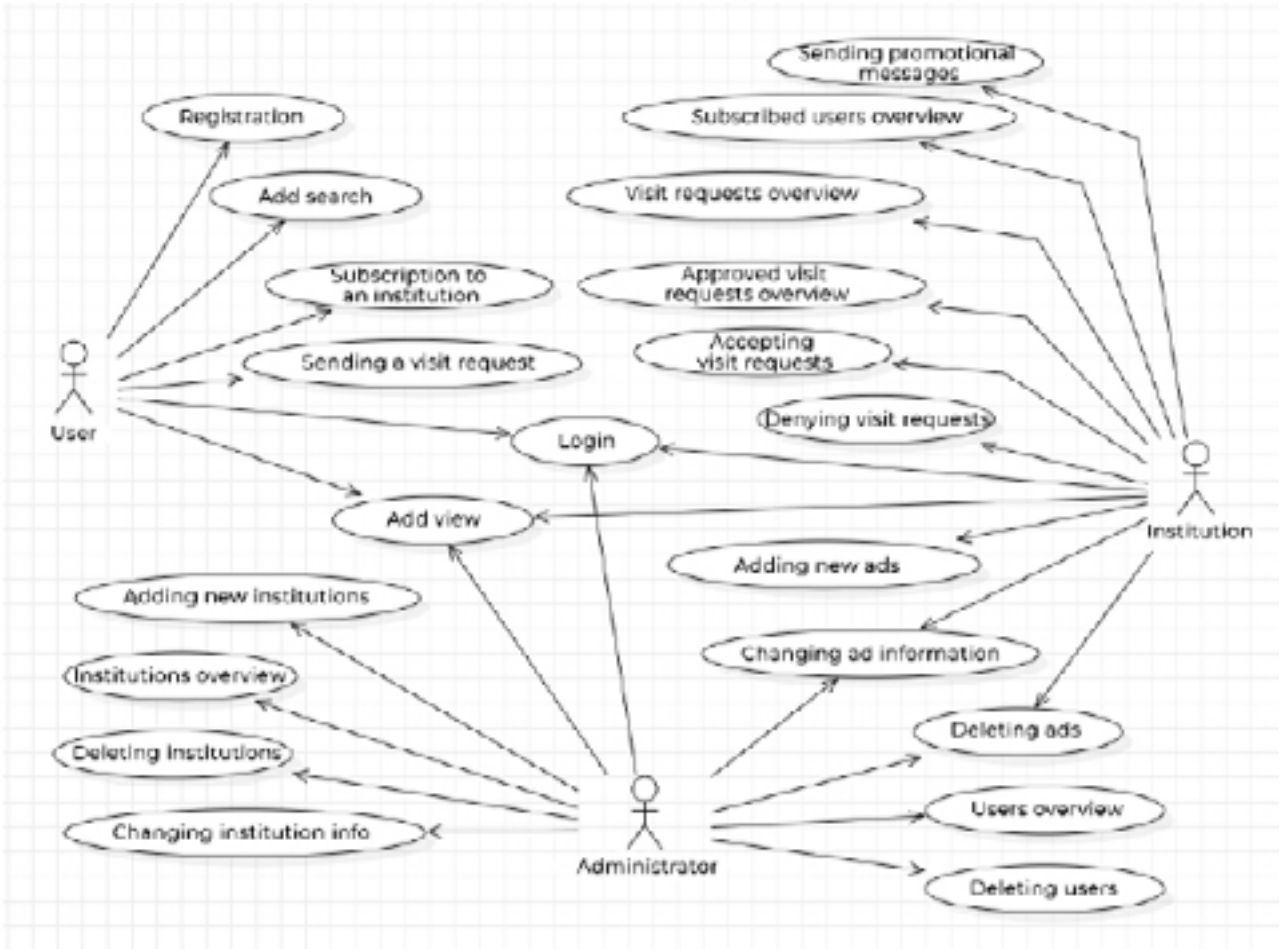

Figure 6. Use case diagram of all Use cases

Due to the limited scope of the paper, we are providing a description of one Use case as an example (UC4).

UC4: Ad view

Name: Ad view

Actors: User, Administrator, Institution

Participants: User, Administrator, Institution and system

Prerequisites: The system is active, the user is logged in and the page with ads is displayed

Basic scenario:

1. The user asks the system to show detailed ad info (APSO).
2. The system finds the request details (SO).
3. The system shows the requested details to the user (IA).

Alternative scenario:

2.1. The system cannot connect to a database and displays an appropriate message (IA).

3.2. Analysis

In the analysis phase, we describe the logical structure and the behaviour of the software system. We describe the structure of the software system using a conceptual and relational model. We describe the behaviour of the system using sequence diagrams (UCSD) made for each Use case and using the system operation contracts received on the basis of the previous diagrams. Now follows an example of a system sequence diagram.

UCSD4: Ad view

Basic scenario:

1. The user requests that the system display advertisement details (APSO).
2. The system displays advertisement details to the user (IA).

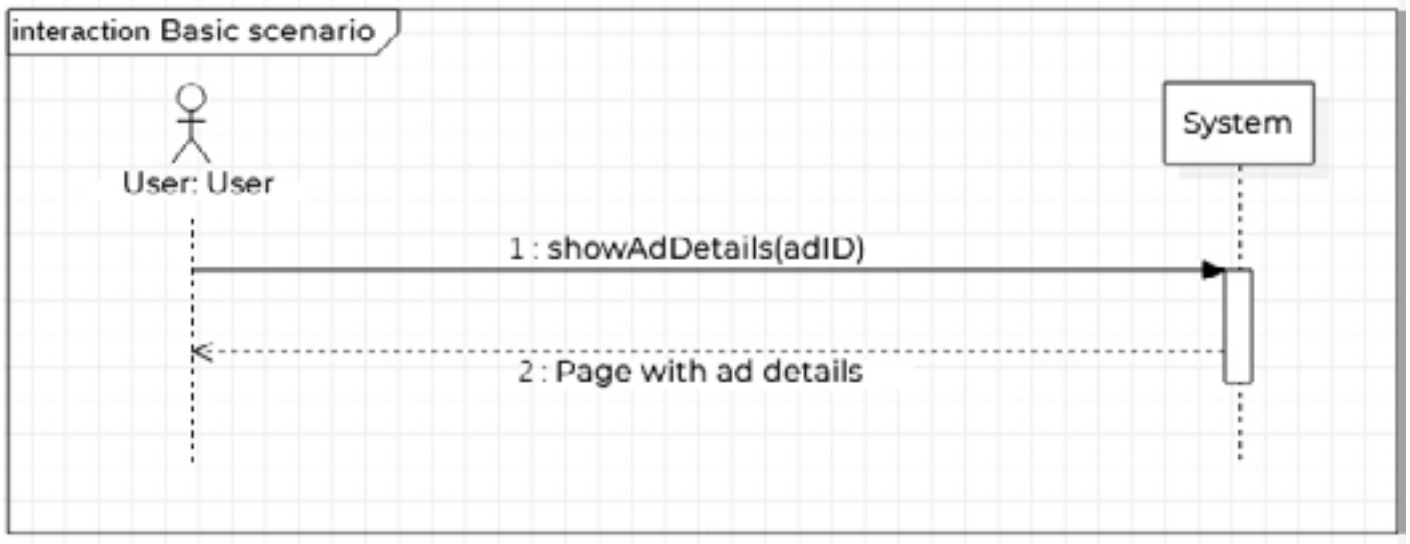

Figure 7. UCSD4 – Viewing an ad

We introduced a system operation:

1. showAdDetails(adID)

For each detected system operation, a contract is made that describes what the operation does, but not how it does it, and one contract is connected with one system operation [8]. Now follows an example of a contract (UC18).

Contract UG18: showVisitRequests
Operation: showVisitRequests(institutionsID)
Connection to UC: UC18
Preconditions: Requests for visits exist in the database
Postcondition: The list of requests for visits is displayed

After defining all the contracts, a conceptual model was created on the basis of the data from the functional requirements and Use cases (Figure 8).

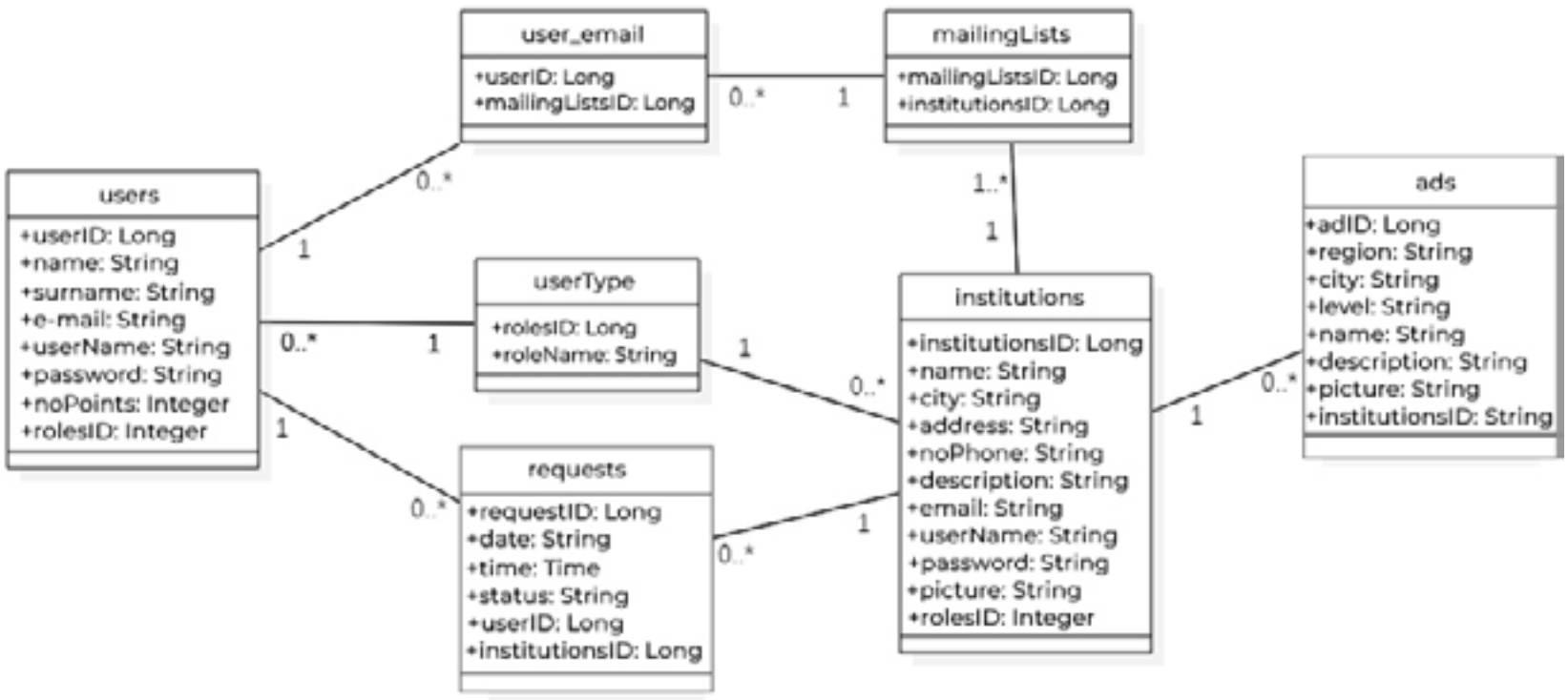

Figure 8. Conceptual model

The relational model is created on the basis of the conceptual model. It constitutes the basis for database design.
userType(rolesID, roleName)
users(userID, name, surname, e-mail, userName, password, noPoints, rolesID)
users(rolesID) references userType(rolesID)
institutions(institutionsID, name, city, address, noPhone, description, email, userName, password, picture, rolesID)
institutions(rolesID) references userType(rolesID)
ads(adID, region, city, level, name, description, picture, institutionsID)
ads(institutionsID) references institutions(institutionsID)

mailingLists(mailingListsID, institutionsID)
mailingLists(institutionsID) references institutions(institutionsID)
user_email(userID, mailingListsID)
user_email(userID) references users(userID)
user_email(mailingListsID) references mailingLists(mailingListsID)
requests(requestID, date, time, status, userID, institutionsID)
requests(userID) references users(userID)
requests(institutionsID) references institutions(institutionsID)

3.3. Design

The phase of design describes the physical structure and the behaviour of the software system, i.e. its architecture. As such, it includes the design of application logic and the design of logical structure and behaviour of the software system.

3.3.1. Application structure design

Contract UG18: showVisitRequests (Figure 9 and Figure 10)
Operation: showVisitRequests(institutionsID)
Precondition: Requests for visits exist in the database
Postcondition: The list of requests for visits is displayed

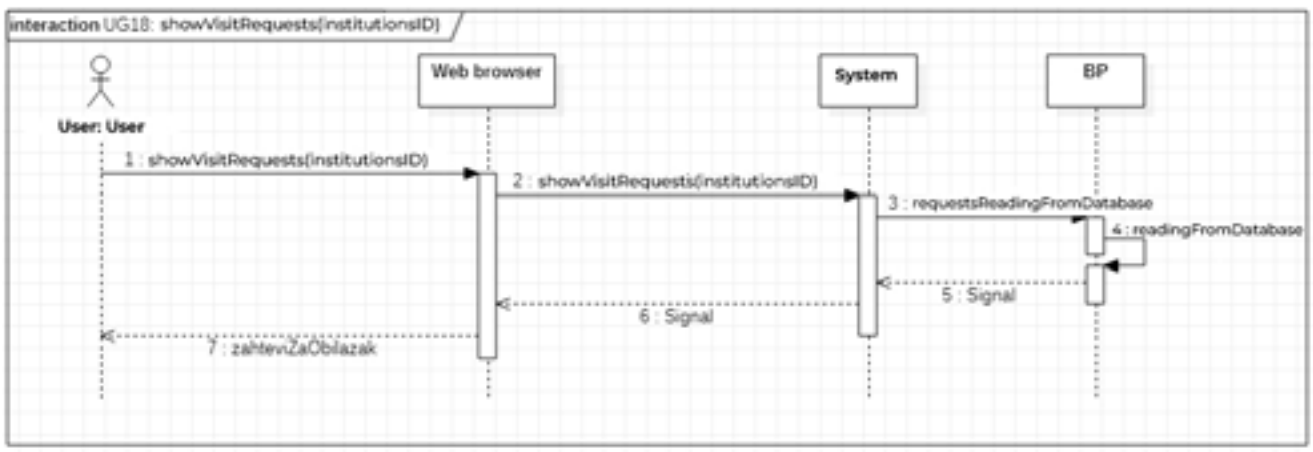

Figure 9. Sequence diagram UG18 – showVisitRequests

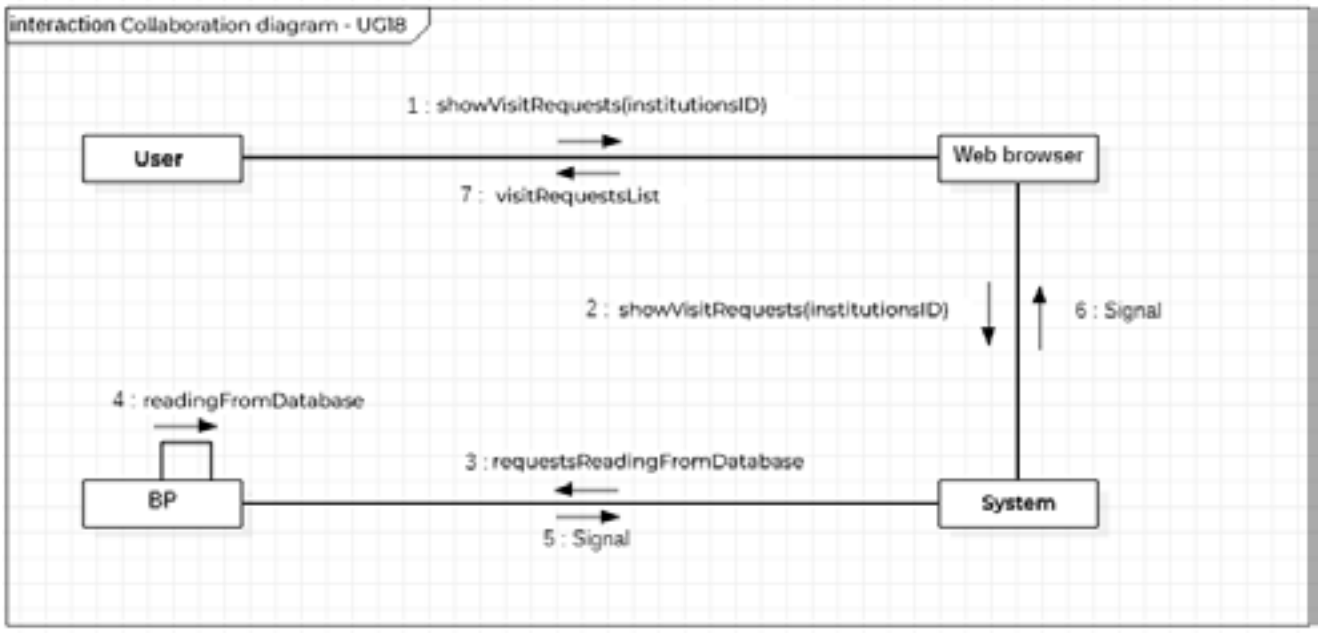

Figure 10. Collaboration diagram UG18 – showVisitRequests

3.3.2 Creating the user interface

What follows is an example of defining a portion of the user interface for the "Education Support" app.
UC4: Ad view
Prerequisites: The system is active, the user is logged in and the page with ads is displayed (Figure 11)

Figure 11. Creating the user interface – UC4

Basic scenario:

1. The user asks the system to display detailed ad information b
   Action description: By clicking on an advertisement, the user asks the system to display detailed information.
2. The system locates ad information
3. The system shows detailed information to the user (Figure 12)

Figure 12. Creating the user interface – UC4

Alternative scenario:

The system cannot connect to a database and displays a corresponding message (Figure 13)

Figure 13. Creating the user interface – UC4

3.4. Implementation and testing

In the implementation phase, the system is coded by utilising certain technologies. The following technologies were utilised for the development of this web application: Spring framework, including Servlets, Maven, Hibernate and Thymeleaf on the server side. On the client side, HTML, CSS, JS, and JQuery and Bootstrap libraries were used. In the testing phase of the "Education Support" web app development, the functionalities of the application itself were tested, including entering various data in order to determine and ameliorate possible irregularities.

4. Conclusion

In this research, a web application was created to facilitate the locating of the desired institutions for further education of the interested users, and thereby contribute to the development of education in our country in general. The application can certainly be helpful in further career development. The technology of choice for the application was Spring, as it is conducive to faster, easier and more secure web application development. Further research would concentrate on the possibilities of enriching the "Education Support" web app with new functionalities.

References

1. Vlajić S, Savić D, Stanojević V, Antović I, Milić M. Projektovanje softvera – Napredne Java tehnologije. Beograd: Zlatni Presek, 2008.
2. Jevremović S. Java programiranje veb aplikacija. Beograd: ITS, 2016.
3. Spring Framework – Reference Documentation, Spring Framework, 2020. Available at: www.springframework.org
4. JavaBeans Documentation, Oracle, 2020. Available at: http://java.sun.com/javase/technologies/desktop/javabeans/docs/spec.html
5. Smeets B, Ladd S. Building Spring 2 Enterprise Applications. US: Apress, 2007.
6. Vlajić S. Projektovanje programa. Beograd: FON, 2004.
7. Hibernate – Reference Documentation, Hibernate, 2020. Available at: http://www.hibernate.org/
8. Anđelić S. WPF i ASP.NET Framework - projektovanje i implementacija softvera. Beograd: ITS, 2016.
9. Spring Web MVC Framework Flow, 2015. Available at: https://www.onlinetutorialspoint.com/spring/spring-web-mvc-framework.html
10. Vlajić S. Projektovanje softvera, skripta. Beograd: FON, 200