SILESIAN UNIVERSITY OF TECHNOLOGY

FACULTY OF AUTOMATIC CONTROL, ELECTRONICS AND COMPUTER SCIENCE

Final Project

Web application supporting diabetes care

Author: Paweł Smolarski

Supervisor: dr inż. Krzysztof Dobosz

Consultant: mgr inż. Tomasz Jastrząb

Gliwice, January 2018

Table of contents

1. Introduction

This chapter contains information about the main goals of the project and presents the idea of document organization.

1.1 Description of topic

This disease is a problem of people all over the world. According to WHO's research in 2017 there were 422 million patients which means around 8.5% of all the adult population. [6] Diabetes is disease which main feature is high level of glucose (or blood sugar), which can consequently lead to heart problems. There can be listed two most common types of diabetes – type 1 and type 2. First one is connected with chronic, autoimmunological disease process which leads to slowly destroying pancreatic islets used for production of insulin. Second one is the most common one. The difference between this one and diabetes type 1 is that in second one the pancreas is not able to create insulin. The number of cases of diabetes increases all the time. Therefore it is very important to make the society aware about diabetes.

1.2 Goal of project

The main goal of the project is creating a web application allowing diabetics to do selected exercises and tasks which concern the awareness of the dangers posed by diabetes. The scope of project includes:

- various types of tasks and exercises allowing to expand knowledge about diabetes,
- competitiveness between players who are involved in the game,
- administrator panel which makes opportunity to modify a database of exercises.

The application provides two types of users:

- player – user which has possibility to take part in the game and rival with another players,
- administrator – user which has possibility to modify tasks' content.

The application has to be developed by using Java programming language. For the implementation there is used *Java EE 7* which provides convenient ways of developing web applications.

1.3 Document organization

The document is divided into 8 chapters. The second chapter presents the tools and technologies used in the project and describes wider specification. The third chapter brings the reader closer to the requirements of project – both functional and non-functional. The next chapter contains the description of project internal organization which means the way of creating software. The fifth chapter presents the methods used for the implementation of the application. The sixth chapter presents the external specification which means requirements and process of installation, types of users containing in application and also simple usage scenarios. The seventh chapter provides the information about testing techniques used during the development of the application. The last chapter is a summary of project.

After this chapters there are two additional ones. First provides references to literatures used in project, while the second is a description of attachments located on the CD-ROM.

2. Analysis of the task

This chapter describes the main technological assumptions of the project. They include, first of all, the tools used during application development, the technologies such as the programming language, the frameworks and the external libraries and also non-functional issues which arose while developing the software.

2.1 Problems encountered in the project

One of the biggest challenges of project is to make interactive game which not only teaches players about diabetes but also makes the time spent in front of computer enjoyable.

The second very important issue of the project is to make it as simple as possible, so the player would intuitively know how to finish the quiz. It is widely known knowledge that modern user of internet is used to fast and easy solutions. In the world where time is money it is necessary to economize resources of potential client.

The last thing which has to be implemented elegantly was the administration panel. It is supposed to be simple and powerful at the same time. The main thing that should be contained in administrator panel is possibility to modify data. In the project there are provided Create, Retrieve, Update and Delete (CRUD) functionality. There are four basic functions used in working with persistence storage by means of which there is possibility to have full control of data.

2.2 Tools used in the project

There are presented the tools used during project development:

- *NetBeans IDE* – integrated development environment for Java programming language. It is fully open-source project under CDDL (Common Development and Distribution License) or GPL (General Public License). It can be developed by third party programmers. *NetBeans IDE* is cross-platform. It can be run among Microsoft Windows, Linux, Solaris, macOS and any other system which delivers appropriate JVM (Java Virtual Machine). IDE mentioned above was created primarily for making

applications in all Java versions – *Java SE*, *Java ME*, *Java EE* and *JavaFX*. *NetBeans IDE* provides also possibility to install additional modules. It can be used not only for programming in Java but also C/C++, PHP and many others. In project *NetBeans IDE* in version 8.2 was used, because it provides many enhancements in *HTML5/JavaScript*, *Java Editor* and *Profiler*, debugger and *C/C++*. It also delivers support for *PHP 7*, *Docket*, *ECMAScript 6* and *Experimental ECMAScript 7*. [7]

- *MySQL Workbench* – tool used to manage and design *MySQL* databases. By using this application it is possible to edit configuration of the server and its components. It can be also used to create schemas of database (visual representation of views, tables, and connections). There is support for developing SQL queries. It is an open-source application under GPL. This version provides some changes. The most important which can be shown are migration from the command-line not only by GUI, generator of SSL certificate, SQL editor auto completion and many others. [8] In project this tool was used for creating database which contained all necessary data such as players' results and tasks descriptions. The wider description can be found in 4.5 subsection.

- *Mozilla Firefox* – open-source web browser based on Gecko engine. It is one of the most popular browser all over the world. In my project this program was used for testing the front-end view. There is much expanded debugging console in which the changes can be noticed easily.

- *GlassFish Server* – open-source Java EE application server under CDDL or GPL licenses. At the beginning developed by Sun Microsystems, nowadays by Oracle. The version used in the project (4.1.1) supports *Java 7 EE*, *EJB* (Enterprise JavaBeans), *JPA* (Java Persistence API), *JSF* (Java ServerFaces), *JSP* (Java ServerPages), servlets and many others. It is supported by NetBeans IDE so starting own server is very simple.

2.3 Technologies used in the project

There are presented technologies used during the project development:

- *Java 1.8 Standard Edition* – object-oriented programming language created by team from Sun Microsystems under James Gosling's leadership. Java is a language in which source code is compiled into bytecode. It allows running applications on every device which has delivered JVM (Java Virtual Machine). It means that it is platform-independent. Other main assumptions are distributed computing, automatic garbage collector and inheritance. Version used in project compared to earlier ones there have been implemented such components as Lambda Expressions and Stream API for comfortable work with collections. These changes are implementation of functional programming which is becoming more and more popular.

- *Java 7 Enterprise Edition* – server platform used for creating business applications merging set of standards for Java language which should simplify complex and repeatable problems related to software development. It is supplied by Oracle after taking over Sun Microsystems Inc. Main elements provided by Java EE are *EJB* (Enterprise Java Beans), *Jax-RS* (Java API for RESTful Web Services), *JMS* (Java Messaging Service), *JTA* (Java Transaction API), *JPA* (Java Persistence API) and *JSF* (Java ServerFaces). The last two elements are used in the project and are described wider later. Version 7 was used in the project, providing a lot of updates of core components and added support for *JSON*.

- *Git* – Distributed Version Control System created by Linus Torvalds to support developing Linux's kernel. In connection with its open-source feature it is one of most popular software for its category. Server used during producing the application was GitHub which provides free public repositories. The most important attributes of *Git* are decent support for branched process of developing application, possibility to have own copy of repository locally, support for protocols such as *HTTP(S)*, *FTP*,

SSH, high speed of execution, revision of whole images of project. Primary operating system for *Git* is GNU/Linux but it is also provided for other systems.

- *JSF* – framework based on Java designated for creating web applications. It simplifies creating of User Interfaces (UI) by using MVC architecture pattern. It defines a lot of standard UI components. It makes possibility to reuse this components. It provides chance to create new UI components libraries or extend existing ones. In application this framework was used for connecting front-end and back-end. It was also used for keeping consistent of program. The wider description of usage can be found in 4.4 subsection.

- *MySQL* – open-source system for managing relational model databases. It is supported by most of the operating systems. It is characterized by high speed of execution, flexibility and simple way of handling. In the process of taking over *MySQL* by Oracle an alternative version of this software was developed – *Maria DB*, which is based on the same source code and keeps backwards compatibility. The main purpose of this system is high performance and for long time it has not been compatible with SQL, for example for most of time it could not operate transactions. Considering lightness and speed, *MySQL* is very frequent choice.

- *Hibernate* – framework for implementation persistence layer. It is providing translation of data between relational model databases and ORM (Object-to-Relational Mapping). It is extending official standard – *JPA*. From development point of view there is possibility to operate on objects called "entities" and saving results into relation database by "EntityManager" object. The way in which objects and connections between them are translated into elements of database is defined by annotations (directly during writing source-code) or by configuration XML files. Apart from the standard functions, *JPA* provides also *JPA Query Language* similar to SQL. The version used in the project was 4.3, whose main feature is providing support for the *JPA 2.1*.

- *Primefaces* – open-source library of components for *JSF*. Main goal of this software is creating website with dynamic graphic interface elements in simple and fast way. It features over 100 components. Besides version for JSF there exists also other such as *PrimeUI*, *PrimeFaces Mobile*, *PrimeFaces.NET*, *PrimeNG*, *PrimeReact*, *PrimeUI Widgets*. The most important features besides a lot of various components are high performance, simple way of handling, ease of configuration, independence from others libraries. The components are supplemented by JavasScript's library - *jQuery* which depends on *AJAX*. The usage is describe wider in 5.3 subsection.

- *CSS* – Cascading Style Sheets language which describes the presentation part of WWW pages. It is used for formatting *HTML* elements. The list of rules are located inside *.css* file or directly inside code. In the application this language was used for providing background image for whole page and specific tasks.

- *JS* – JavaScript scripting programming language mostly used in developing websites. It is used to ensure interaction through reacting on events, validating input data or creating advanced visual effects. On the server side it can work as *node.js* or *Ringo*. There is also possibility to develop standard applications.

3. Requirements of the project

This chapter describes in detail the functional and non-functional requirements that were to be met by the developed project.

3.1 Functional requirements

In software engineering a functional requirement defines what system should do, what is its main goal. It is describing what inputs, the behavior and outputs system should provide.

The most important functional requirements which had to be fulfilled were:

- providing several different types of tasks and exercises which would require some knowledge about diabetes and as a consequence it would teach and make people aware of the dangers posed by diabetes .,
- providing saving game status in the session and keeping its. It means that user should not have possibility to start new game before finishing the old one unless the session is lost,
- administration panel, which would allow creation, modification and removal of task types and particular exercises for the given type. The main issue is to prevent the access to the administration pages by unauthorized users. The information necessary to login as administrator are contained in database.

In Figure 3.1 can be seen UML Use Case Diagram of functional requirements.

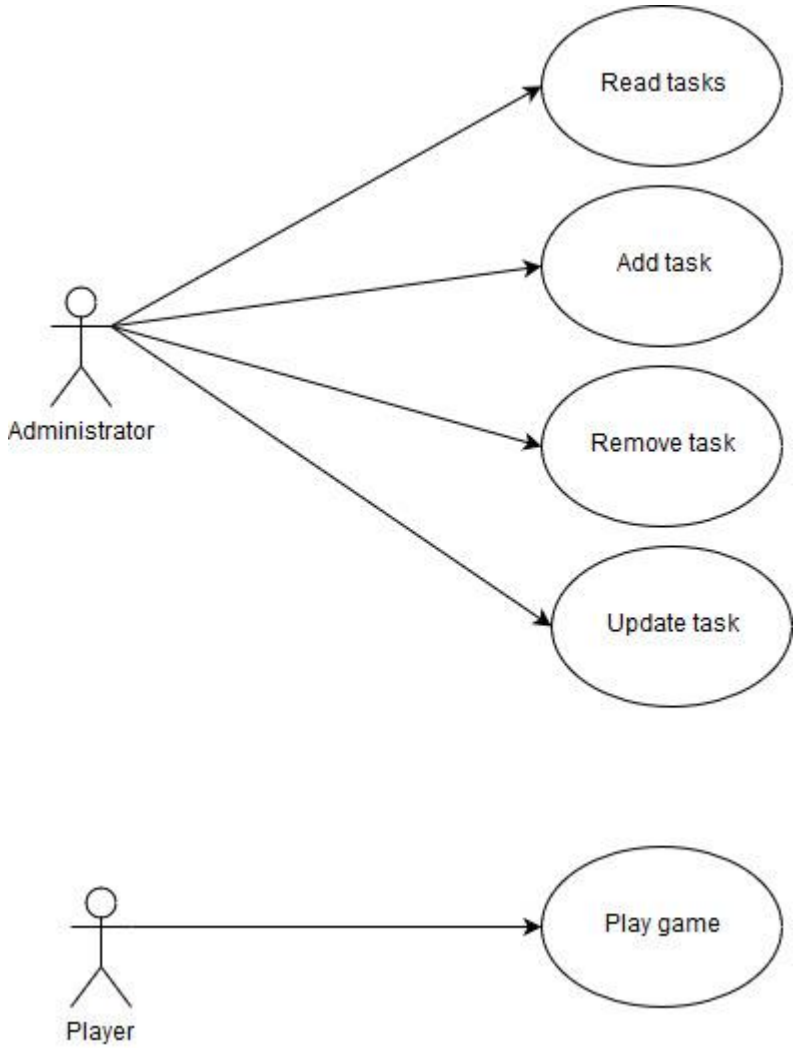


Fig. 3.1. UML Use Case Diagram

3.2 Nonfunctional requirements

In software engineering a nonfunctional requirement defines quality goals of application. It is describing how the system works.

The most important nonfunctional requirements which had to be fulfilled were:

- remaining independent of various operating systems. The only requirement for running program is using one of the browsers which support the frameworks used,

- high performance. Nowadays, when everybody hurries, the application cannot be slow because it can make the user bored easily. There should be something that will keep the user interested,
- providing real-time system. It means that result of work of application depends on the time of development of this outcome,
- reliability. Application shared all over internet can be accessed by many users simultaneously what means that the server must not allow the situation in which it is down,
- security. In relation of the confidential data related to people suffering from diabetes, it is obligatory not to allow users to influence or access other people's data.

4. Internal organization

This chapter describes the structure of the project. It presents the main ideas such as the design and architectural patterns and the organization of the application.

4.1 General description

Along with technological development, diabetes has become a global problem. The willingness of people to do sports and eat healthy food is disappearing. It is possible to order anything without leaving home. It get people used to fast and easy solutions. For these reasons the number of people suffering from diabetes has increased tremendously. In order to slow down this process, it is necessary to create tools to make the community aware about this disease. In Figure 4.1 is presented simplified diagram of project's design.

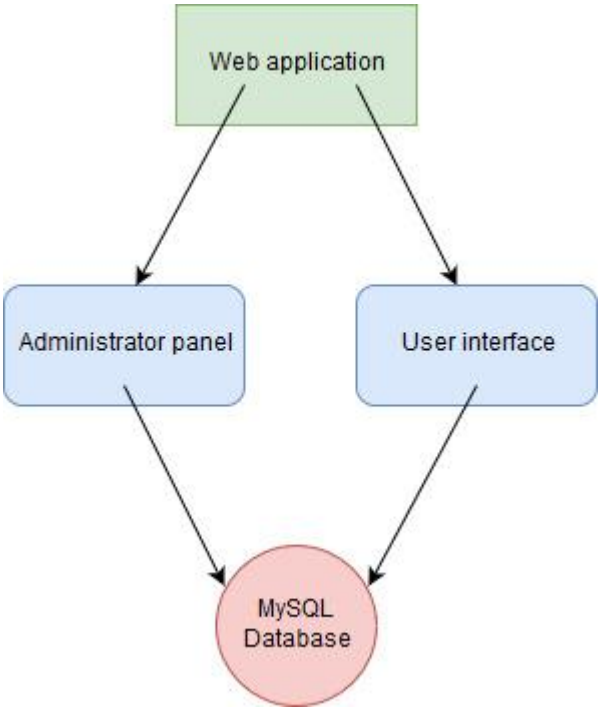


Fig. 4.1. Simplified project design

4.2 Architectural and design patterns

Architectural and design patterns are one of most important things in modern programming. They let the project to keep freshness and possibility to extend its modules in the future. The first mentioned type of pattern is an approved and tested method for

solving problems appearing at the moment just before the start of project development. It defines the general structure of the given system. The design pattern are universal, tested solutions for repetitive problems appearing during application development. They allow to make the source code clearer, more immune for mistakes and easier for refactoring.

The project was created according to the SOLID rules, proposed by Robert C. Martin. [9] It describes five basic assumptions of object-oriented programming.

Every letter of the acronym denotes a different rule, namely:

- S – Single responsibility principle – class should not be responsible for more than one thing,
- O – Open/closed principle – classes should be open for extension and closed for modifications,
- L – Liskov substitution principle – there should be a possibility to replace modules of program without impacting other modules. It requires that inherited class should not change sense of base one,
- I – Interface segregation principle – many specific interfaces are better than a single one responsible for everything,
- D – Dependency inversion principle – modules should be connected between themselves by abstraction, high-level objects should not depend on low-level.

There are presented the architectural patterns used in project:

- MVC – Model-View-Controller is one of the most common architectural patterns used in modern applications. The basis of it constitutes the idea of separation of the parts of the system responsible for processing the data from the parts responsible for rendering the view. The lack of such separation makes the modules strictly linked with each other. Isolation of these modules simplifies the introduction of changes in both business logic and in the user interface. Figure 4.2 presents basic schema of MVC. [10]

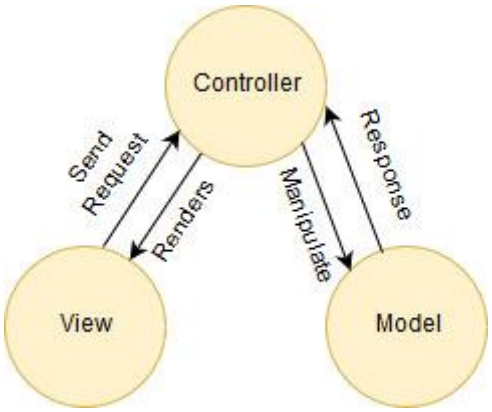


Fig. 4.2 MVC basic schema

- IoC – Inversion of Control is an architectural pattern which reversing controlling duty into used framework which in specific moments execute source code. It is used for removing dependencies in source code. The IoC mechanism can be achieved in three different ways. The first one is the factory design pattern which goal is providing interface used in creation of unspecified objects as connected types , the second one is service locator pattern which depends on sharing container accessibly in whole application and the last one is dependency injection which is described next. IoC is used for making applications more extensible and modular. Using this pattern increases the possibility to test the programs. There is no problem to mock data and write unit tests.

There are presented the design patterns used in the project:

- DI – Dependency Injection is a pattern adopted in modern programming languages both for implementing internal mechanisms and as a means to decouple connections between classes. This pattern is a special case of IoC so the main idea is to reverse control of program execution. Instead of creating dependencies and new objects with the use of “new” operator or by search operation, the needed resources will be injected into target element. This approach has many advantages. Firstly, the client does not need to know about different implementations of the injected resources which simplifies providing modifications inside the project. Secondly, as mentioned before, unit tests can be introduced very quickly.

The next advantage is that the configuration of the program can be easily extended, which results in the reduction of negative effects of changes. The last benefit is that the architecture based on loose connections simplifies making plug-and-play systems. The main technique of dependency injection is changing place of creating objects by using injector which takes over control.

- CDI – Context and Dependency Injection is the mechanism of dependency injection available in the Java EE platform. The types of objects which can be injected are POJOs (Plain Old Java Object), business resources, for example data and collections, remote EJB references, session beans, objects of "EntityManager" type, references to network services, producer fields and returned values of theirs methods.

- DAO – Data Access Object pattern which ensures a convenient data access interface which can be used by various program layers. It is applied to standardize and conclude whole source code which has functionality to access to data into persistent data repository. DAO is managing connection with data source in order to load and save the data. It is used to take care of abstraction and encapsulation of data source isolated business logic and data source implementation. In principle, the DAO pattern allows to change the data source without affecting the application. The implementation of CRUD operations by using DAO pattern can be achieved by low-level interfaces API such as *JPA* and *Hibernate*. [5]

- Front Controller – design pattern in which there is one main controller. All requests are redirected into such controller which after analyzing decide what to do next. This master controller in *JSF* is called *FacesServlet*. The configuration of its parameters can be customized in *web.xml*.

- Singleton – one of the most famous and easiest to implement design patterns. It is a creation pattern which is based on the idea of making only one instance of particular type. The danger of using singleton object is that in multithreading environment data races can occur if programmer would

not synchronize code properly. Java delivers a very safe possibility for creating singleton classes by using Enums (enumeration types). The most common reasons for using this design pattern are:

- the need to provide access to common data in the whole application, for example configuration files,
- to load or buffer valuable resources and to optimize the time of such operations,
- to create an instance of application logger, since usually only one instance is needed,
- to manage objects in class which implements factory pattern,
- to lazily create static classes. [5]

4.3 Interfaces

To satisfy the SOLID principles, and in particular the Open-Close principle, the system uses an abstraction in the form of interfaces. Interfaces provide the decoupling of the different parts of the system. Thanks to them, the system is easy to extend. In this project all types of tasks are forced to implement a common interface. This way the application can easily call appropriate method of the specific task. The *Taskable* interface has got one method which returns the type of specific task. This object contains the information about the URL needed for the redirection. The schema can be seen in Figure 4.3.

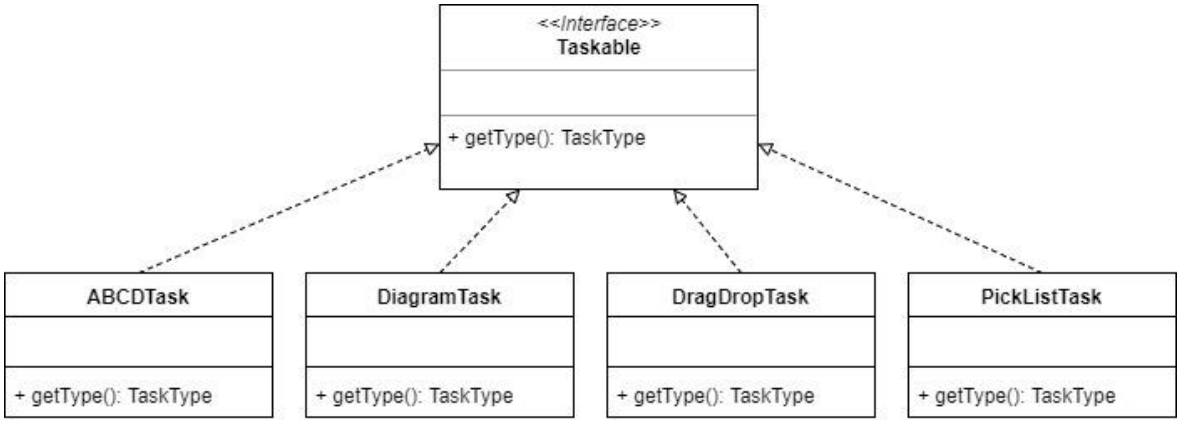


Fig. 4.3. *Taskable* class structure

Apart from the *Taskable* interface, another interface called *BeanTaskable* was used for connecting the managed beans responsible for the specific tasks. The most important method to be implemented is *validate()* which indicates the way in which the correct answer is determined. This is a generic interface. The schema can be seen in Figure 4.4.

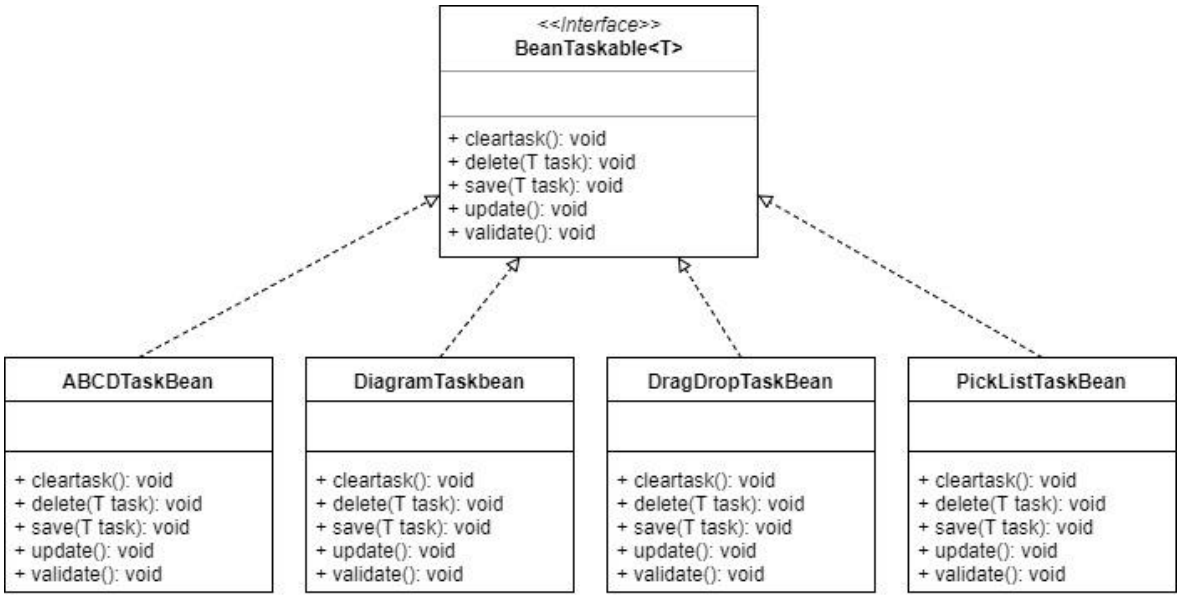


Fig. 4.4. *BeanTaskable* class structure

The last interface used in the project was *TaskRepository* which provides CRUD method declarations for DAO classes. It is also generic. It can be seen in Figure 4.5.

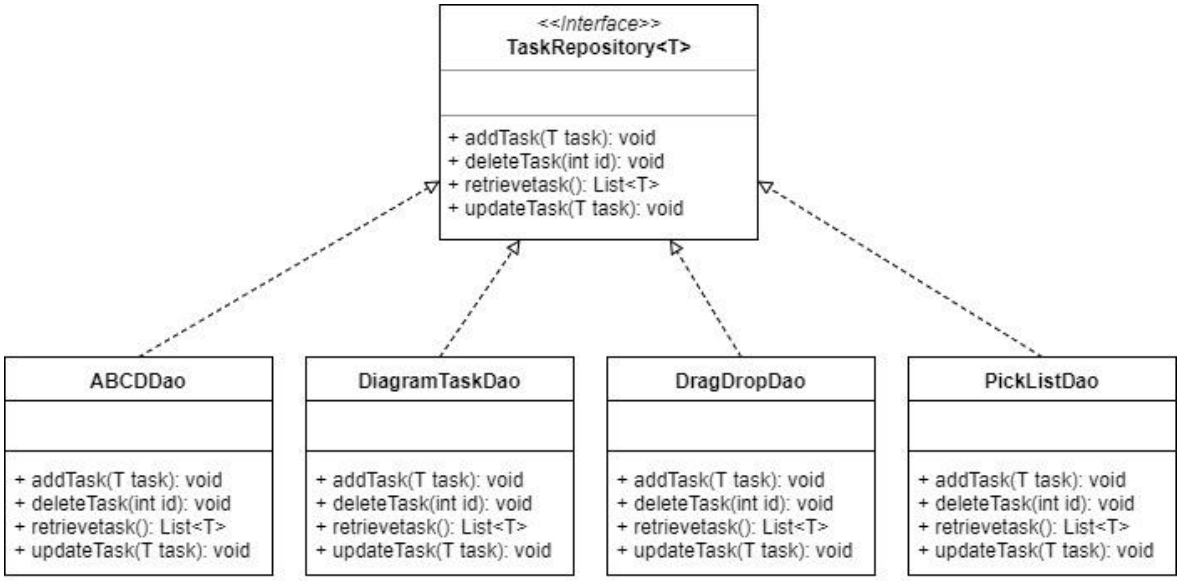


Fig. 4.5. *TaskRepository* class structure

4.4 Communication between front-end and back-end

The method used in application to provide communication between front-end and back-end parts of system is based strictly on MVC assumptions. Java Server Faces framework used for creating the view part of the project, along with *Primefaces* library, handles the communication through Managed Beans. These are Java classes using which there is possibility to manipulate content on *xhtml* pages. To indicate that specific class should be Managed Bean it is necessary to use *@ManagedBean* annotation. In this project Managed Beans with annotation *@ViewScoped* are designated to be the controllers which combine the model and the view with each other. By default, the managed bean can be accessed from the view part using the class name, starting with a lowercase letter, unless the name has been explicitly changed by the developer.

In the project, the Managed Beans are used in the vast majority of views. By using them, the application can retrieve input data provided by the user during the game, validate answer correctness and render the view or redirect to another resource. In order to have the possibility to access the fields of the bean from the view part, it is necessary to provide getters, setters and other public methods.

4.5 SQL database system

The database server used in the project was MySQL. Taking into account that the application is intended to provide the player with several types of exercises and keep the result of game, it was necessary to use fast and reliable database engine such as MySQL. The database consists of six independent tables. There are four tables representing one specific type of task. Every such table contains information necessary for the application to render the questions and answers and to evaluate the answers in order to score points. The other two tables are used for keeping players' data and users which are able to use administration panel. The schema generated by MySQL Workbench is presented in Figure 4.6.

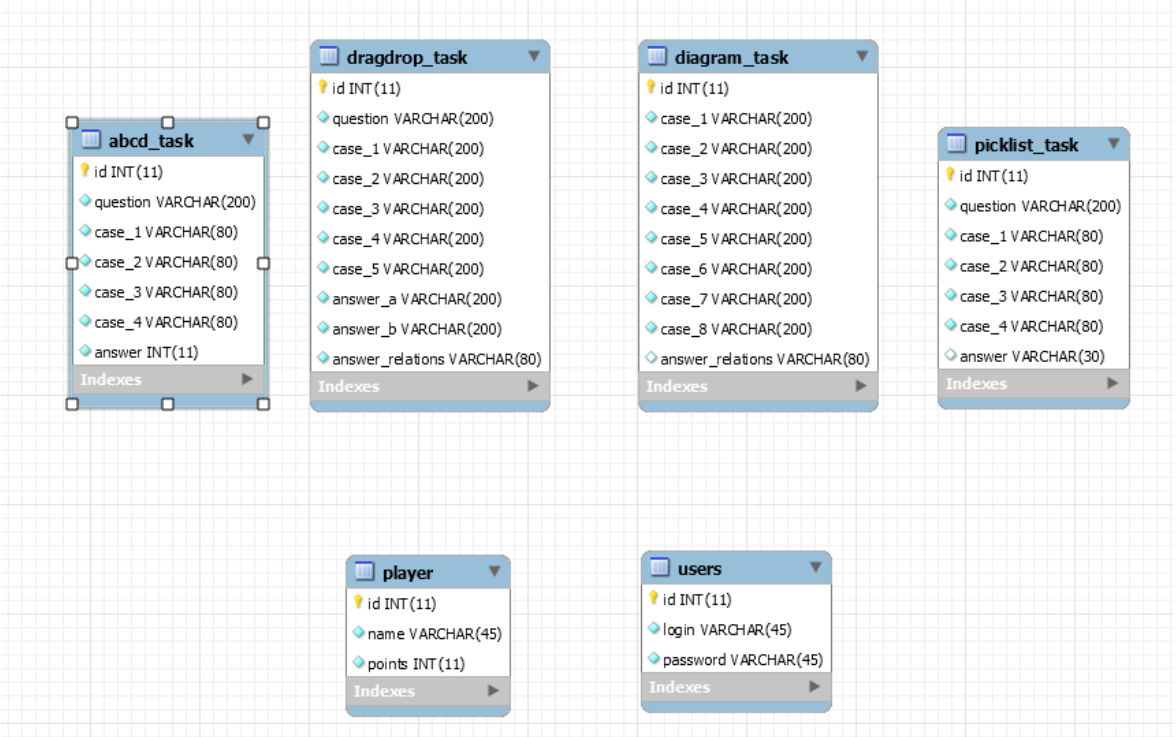


Fig. 4.6. Database structure

5. Implementation

In this chapter selected implementation issues are being discussed.

5.1 Structure of game flow

An ordering algorithm is needed to make the game possible. At the beginning, the user provides the nick, the session is created and a list of tasks is drawn at random from the database. Then if there are tasks to be played the application is getting the first of them, removes it from the list and renders the specific task on the screen. Then the player chooses an answer, which is later validated and the points are awarded accordingly. This process continues until all tasks are removed from list. Clearing the list of tasks ends the game, which results in clearing the game flag, and the session. The result of the game is saved in the database and the hall of fame page is being rendered. This process is shown in Figure 5.1.

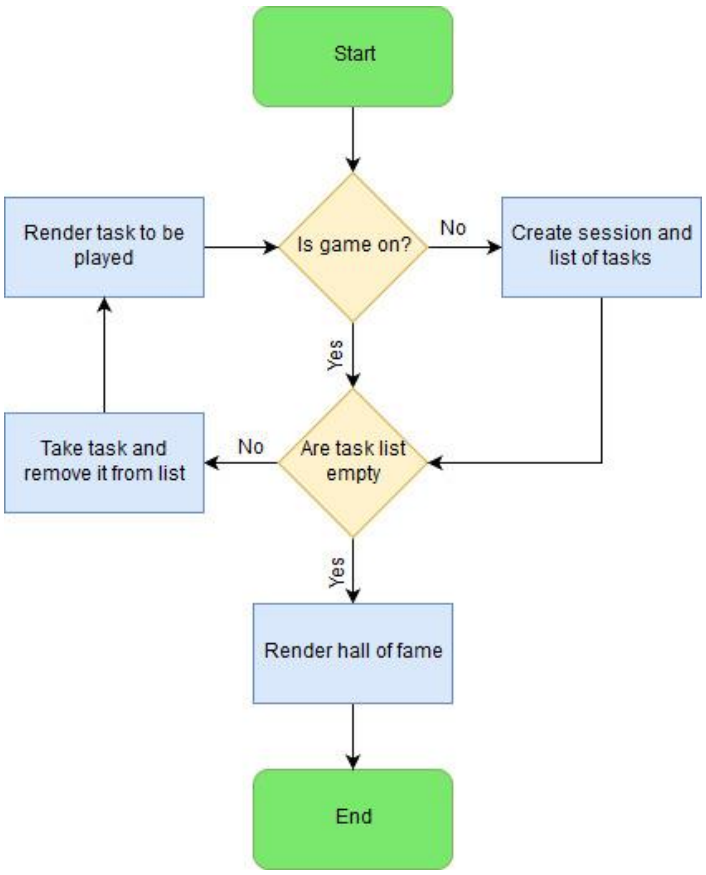


Fig. 5.1. Game flow block diagram

5.2 Security

The specification of servlets delivers default capabilities to prevent specially crafted HTTP requests to get into secured parts. To prevent unauthorized access programmatically, the developer has to create a class which implements *Filter* interface. A filter is an object which performs filtering of the requests to a resource (a servlet or static content). To make individual rules the programmer has to override the *doFilter* method and specify the filtering rules inside this method. Filters were also used for redirecting the player back to game after leaving the browser card. In Listing 5.1 is presented body of override *doFilter* method.

Listing 5.1. Overridden *doFilter* method body

```
1             //Receiving quizBean if session exists
2             if (ses != null)
3             {
4                 quizBean = (QuizBean) ses.getAttribute("quizBean");
5             }
6
7             /*Check if game takes place and player session
8               exists or user want to redirect into welcome page
9               before game end */
10            if ((ses != null && ses.getAttribute("player") != null)
11               && (reqURI.contentEquals("/")
12               || reqURI.contains("/public/index.xhtml"))
13               && quizBean.getPresentTask() != null)
14            {
15                resp.sendRedirect(quizBean.getPresentTask()
16                       .getType().getURL());
17            }
18            /* Check if user want to go into game resources
19                but game not take place at the moment */
20            else if (reqURI.contains("/game/")
21                      && (ses == null
22                       || ses.getAttribute("player") == null
23                       || quizBean.getPresentTask() == null))
24            {
25                resp.sendRedirect(reqt.getContextPath()
26                + "/public/index.xhtml");
27            }
28            /* If user want to login or start game or redirect to
29               public resources let redirection arrive */
30            else if (reqURI.contains("/login.xhtml")
31                    || (ses != null
32                    && ses.getAttribute("username") != null)
33                    || reqURI.contains("/public/")
34                    || reqURI.contains("javax.faces.resource")
35                    || reqURI.contentEquals("/")
36                    || reqURI.contains("/game/"))
37            {
38                chain.doFilter(request, response);
39            }
```

```
40              /* In other case send to lign page */
41              else
42              {
43                  resp.sendRedirect(reqt.getContextPath()
44                 + "/secure/login.xhtml");
45              }
```

It can be noticed from the code shown in Listing 5.1, that there are several conditional instructions which check what the current user wants to do and react properly according to his privileges. By keeping the information about the current task, the application is able to redirect the player back to the game. The server keeps the information about URL (Uniform Resource Locator) and sessions. Every type of task has its own URL indicator by which server can tell where to redirect the user.

5.3 Graphical user interface

Modern web application created in the spirit of MVC architectural pattern should have clear and reusable front-end code. The *Primefaces* library, used in the project, provides not only hundreds of components but also improvements into classic HTML coding.

As mentioned earlier the project provides administration panel. In this element there exists dialog which is used for creating or updating chosen item. In order not to repeat the same code just for changing the values to be listed it had to be used some clever solution. The dialog source code is the same for creating or updating item except rendering specific button. Application orients which button should be delivered by internal data in managed bean. In moment when the user try to add new task chosen task object is empty, so there are not values to be updated and button for adding is rendered as in line number 6. In the reverse situation there is chosen task so text boxes are filled and button for update is rendered as in line number 7. Fragment of source code can be shown in Listing 5.2.

Listing 5.2. Implementation of dialog of single-choice question

```
1         <p:dialog header="" widgetVar="dlg" dynamic="true" id="dialog">
2             <h:form>
3                 <p:growl id="message" showDetail="true" />
4                 <p:panelGrid columns="2">
5                    /* Tasks' input fields  */
6                     <p:commandButton rendered="#{aBCDTaskBean.getTask().id ==
null}" /* Save */ />
7                     <p:commandButton rendered="#{aBCDTaskBean.getTask().id !=
null}" /* Update */ />
```

```
8                </p:panelGrid>
9             </h:form>
10         </p:dialog>
```

Application contains few interesting *Primefaces* components worth mentioning.

First one is radio button panel in which there is possibility to make module with many possibilities to choose. As can be noticed every radio button has its own value as in lines number 3-6. This individual value was used later to verify answer. There is also provided converting this values directly to integer (line number 2) and ajax listener used for receiving user answer (line number 7). Exemplary implementation is shown in Listing 5.3.

Listing 5.3. Implementation of radio buttons component

```
1 <p:selectOneRadio id="answer" value="#{aBCDTaskBean.answer}" columns="1"
layout="grid">
2   <f:convertNumber integerOnly="true" />
3   <f:selectItem itemLabel="#{aBCDTaskBean.task.case1}" itemValue="1" />
4   <f:selectItem itemLabel="#{aBCDTaskBean.task.case2}" itemValue="2" />
5   <f:selectItem itemLabel="#{aBCDTaskBean.task.case3}" itemValue="3" />
6   <f:selectItem itemLabel="#{aBCDTaskBean.task.case4}" itemValue="4"/>
7   <p:ajax listener="#{aBCDTaskBean.setAnswer(aBCDTaskBean.answer)}" />
8 </p:selectOneRadio>
```

Next component is diagram which provides possibility for creating source and target boxes and connection nodes. The implementation can be seen in Listing 5.4.

Listing 5.4. Implementation of dialog component

```
1 <p:diagram value="#{diagramTaskBean.model}" style="height:600px"
styleClass="ui-widget-content" var="el" widgetVar="diagramVar"/>
```

Third is *PickList* which is dual list input component which provides possibility to transferring objects from left to right and backwards. It also works by using *AJAX* requests to change data in the moment of user choose. Exemplary implementation is shown in Listing 5.5.

Listing 5.5. Implementation of *PickList* component

```
1 <p:pickList id="pickList" value="#{pickListTaskBean.tasks}" var="tasks"
style="margin: 0 auto" itemLabel="#{tasks}" itemValue="#{tasks}" >
2   <p:ajax event="transfer" listener="#{pickListTaskBean.onTransfer}"
update="message" />
```

```
3   <p:ajax event="select" listener="#{pickListTaskBean.onSelect}"
update="message" />
4   <p:ajax event="reorder" listener="#{pickListTaskBean.onReorder}"
update="message" />
5   <p:ajax event="unselect" listener="#{pickListTaskBean.onUnselect}"
update="message" />
6 </p:pickList>
```

Last one is *DragDrop* component which provides capability to take and put elements from place to place. To implement this functionality it was necessary to create handler in JavaScript and specify draggable and droppable element. The implementation can be seen in Listing 5.6.

Listing 5.6. Implementation of *DragDrop* component

```
1 <script type="text/javascript">
2   function handleDrop(event, ui) {
3      var droppedTask = ui.draggable;
4      droppedTask.fadeOut('fast');
5             }
6
7 <p:draggable for="pnl" revert="true" handle=".ui-panel-titlebar" stack=".ui-
panel"/>
8
9 <p:droppable for="selectedTasks" tolerance="touch" activeStyleClass="ui-
state-highlight" datasource="availableTasks" onDrop="handleDrop" id="dropA" >
10   <p:ajax listener="#{dragDropTaskBean.onTaskDropAnswerA}" update="dropArea
availableTasks" />
11 </p:droppable>
```

5.4 Communication between front-end and back-end implementation

To make a connection between front-end and back-end there were used mentioned earlier JSF framework. Further there is description of implementation such a connection.

In Listing 5.7 could be noticed fragments of source code responsible for single-choice answer task. First 3 lines mean that this class is *ManagedBean* so it can works with UI component, *ViewScoped* what means that it exists only during one particular view. This class also has to implement Serializable interface to let servlet container share the bean with other resources. Next can be noticed annotation *@ManagedProper*ty. It is used for injecting other beans. In this specific case the class is receiving the *QuizBean* instance which controls the flow of a game. In 15 line there can be seen annotation *@PostConstruct* which ensure that method below it would be called just after managed bean is instantiated, but before the bean is placed in scope. In 25 line can be noticed *validate()* method which

call can be seen in Listing 5.8. We can see how easy is to connect back-end and front-end by using *JSF* framework.

Listing 5.7. Single-choice answer bean class implementation

```
1 @ManagedBean
2 @ViewScoped
3 public class ABCDTaskBean implements Serializable, BeanTaskable<ABCDTask>
4 {
5 ...
6     /**
7      * Variable of session bean to control game
8      */
9     @ManagedProperty("#{quizBean}")
10     private QuizBean quizBean;
11 ...
12  /**
13      * Init method
14      */
15     @PostConstruct
16     public void init()
17     {
18         task = (ABCDTask) quizBean.getPresentTask();
19     }
20 ...
21     /**
22      * Method validation of user choose
23     */
24     @Override
25     public void validate()
26     {
27         if (answer == 0)
28         {
29             addMessage("Error!", "Choose one of options");
30         }
31         else
32         {
33             if (task.getAnswer() == this.answer)
34             {
35                 quizBean.setPoints(quizBean.getPoints() + 1);
36             }
37
38             quizBean.game();
39         }
40     }
41 ....
42 }
```

Listing 5.8. Implementation of calling validate function

```
1 <p:commandButton value="Submit" action="#{aBCDTaskBean.validate}"
update="message :taskForm"/>
```

6. External specification

In this chapter the organization of the application will be introduced. Firstly there will be described the system requirements and installation process of application. Next, all types of users which can exist in the application will be shown. Later, the scope of their responsibilities will be discussed. Finally, there would be provided two simple execution paths of the application. The first one is the player's game scenario and the second one shows how the administration panel works.

6.1 System requirements

Due to the fact that the application is developed in Java programming language, it can be used on every system, for which the JVM is provided. It is provided for a large number of devices and operating systems, including Windows, Linux and Mac OS. The project is developed in *Java EE 7* so its system requirements are the most important ones. [11] The recommended browser is Firefox. In table 6.1 there are presented official system requirements delivered by Oracle for Windows and Linux.

Table 6.1

System requirements for *Java EE 7 SDK*

	Windows platforms	Non-Windows platforms
Minimum memory	1 GB	1GB
Recommended memory	2 GB	1GB
Minimum disk space	250 MB	250 MB
Recommended disk space	500 MB	500 MB

The second thing strictly related to *Java EE 7 SDK is Java JRE 8*. The system requirements for it for Windows are for both 32 and 64-bit Platforms the same:

- minimum CPU – Pentium 2 266 MHz,
- minimum disk space – 126 MB,
- minimum memory – 128 MB. [12]

The third thing needed is the *GlassFish 4.1* application server, for which the system requirements are:

- JDK Version – 7 Update 65, or 8 Update 20 or later,
- required Disk Space Full Profiles – 250 MB,
- required Disk Space Web Profiles – 175 MB. [13]

The last element which should be installed is *MySQL 5.7*. Unfortunately appropriate data regarding system requirements are not available.

6.2 Installation process

To install *JRE 8* it is necessary to download the installation package for the specific operating system from the official Oracle site (https://www.oracle.com) and run it with administrator privileges. Then, by following the instructions displayed by the installer, the JRE can be successfully installed.

For Windows the installation process is simply, since everything that the developer has to do is to download the executable installer from the official site mentioned earlier and double click it to start the installation process.

To install *GlassFish 4.1* using Self-Extracting File under Linux OS the user has to make 2 steps. [14]

Grant execute permissions, if it is necessary:

```
chmod +x ./self-extracting-file-name
```

Then, file has to be executed:

```
sh ./self-extracting-file-name
```

To install *MySQL 5.7* under Windows OS it is necessary to download the executable file from the official site (https://www.mysql.com/) and run it. [15]

On Linux, the MySQL database management system can be installed from the MySQL APT Repository.

Firstly the developer has to download release package for Linux distribution from the official site mentioned earlier and install it by issuing the command given here.

```
sudo dpkg -i /PATH/version-specific-package-name.deb
```

Then it is necessary to update package information from repository.

```
sudo apt-get update
```

Finally the developer can install MySQL server as follows.

```
sudo apt-get install mysql-server
```

There would be provided instruction how to create database.

To install the *MySQL Workbench 6.3* there is only necessary to visit home MySQL home page mentioned earlier and download program. If MySQL server installed earlier is running there is possibility to add schema. The server should be run at 3306 port and named it “diabetes_carrier”.

The username which would has access to this database must has username such as “root” and password such as “toor”. It is worth mentioning that MySQL server can be run as Windows Service what provides possibility to starts and stops automatically.

To import an SQL file with database schema there can be used command from MySQL directory. The SQL file has to be in the same directory as MySQL executable file. The SQL file with cloned schema is attached to CDROM.

```
mysql -u root -p toor diabetes_carrier < diabetes_carrier.sql
```

The default login and password for administrator panel is “admin:admin”. It can be manually changed in database. If there is working GlassFish and MySQL server there is possibility to deploy provided *war* file of application. It is necessary to go into GlassFish server /bin directory and run listed command.

To run program deploy is needed, “war-name” is the name of application war file.

```
asadmin deploy war-name
```

Than we can access application resources by typing following URL in the browser.

```
http://localhost:8080/
```

6.3 User types

There are two different types of users defined in the project. The first one is a player. Because of the fact, that the application is hosted online, which means that everyone can come in to it and start the game, the only requirement for the player is to provide a unique nickname. The nickname is later (i.e. after the game) saved in the table of results and if the player was good enough to beat other opponents he would appear at the hall of fame, which is the list of 10 best players. Figure 6.1 depicts the starting screen of the application in which the player is supposed to provide the nickname.

Fig. 6.1 Player's set nick/welcome page

The second type of user is an administrator, who has control on every existing task in application. To enter the administrator page it is necessary to click "Admin panel" button visible at welcome page. As the result the program will automatically move to the login page. This part of execution can be also called at the time when the unauthorized access attempt takes place. If user would try to access one of administrator's panel page and the session would not exists the application should redirect into this login page. This page is presented in the Figure 6.2.

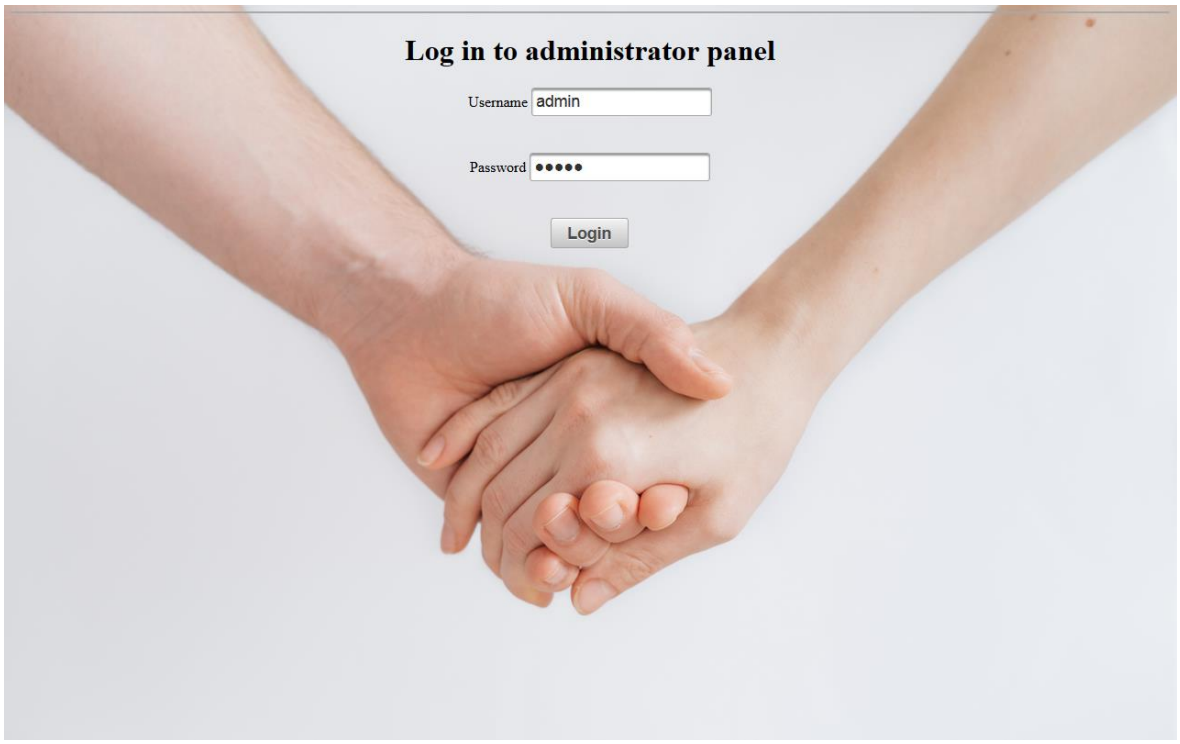


Fig. 6.2. Administrator login panel

6.4 Player's game scenario

As shown earlier the welcome page asks the user to set his nick which is used to create an appropriate entry in the database and the hall of fame. After starting the game the user has 12 exercise to solve. In the following figures, the types of tasks described previously are shown, together with the hall of fame view.

The task presented in Figure 6.3 provides the possibility to choose one and only one answer. It is implemented by simple radio buttons.

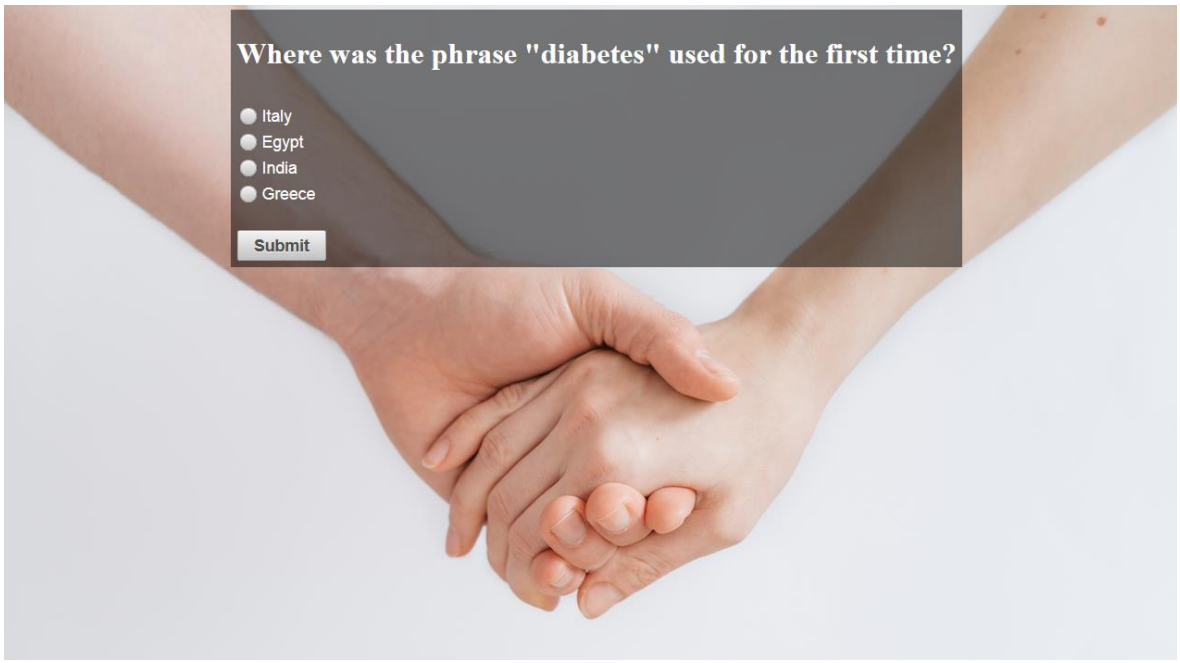


Fig. 6.3. Single-choice question task type

The task shown in Figure 6.4 provides the possibility to connect the elements in many-to-many way. There is also the possibility that one question can have multiple answers. The player may not connect any of elements because there can be case in which there are not related answers with specific questions.

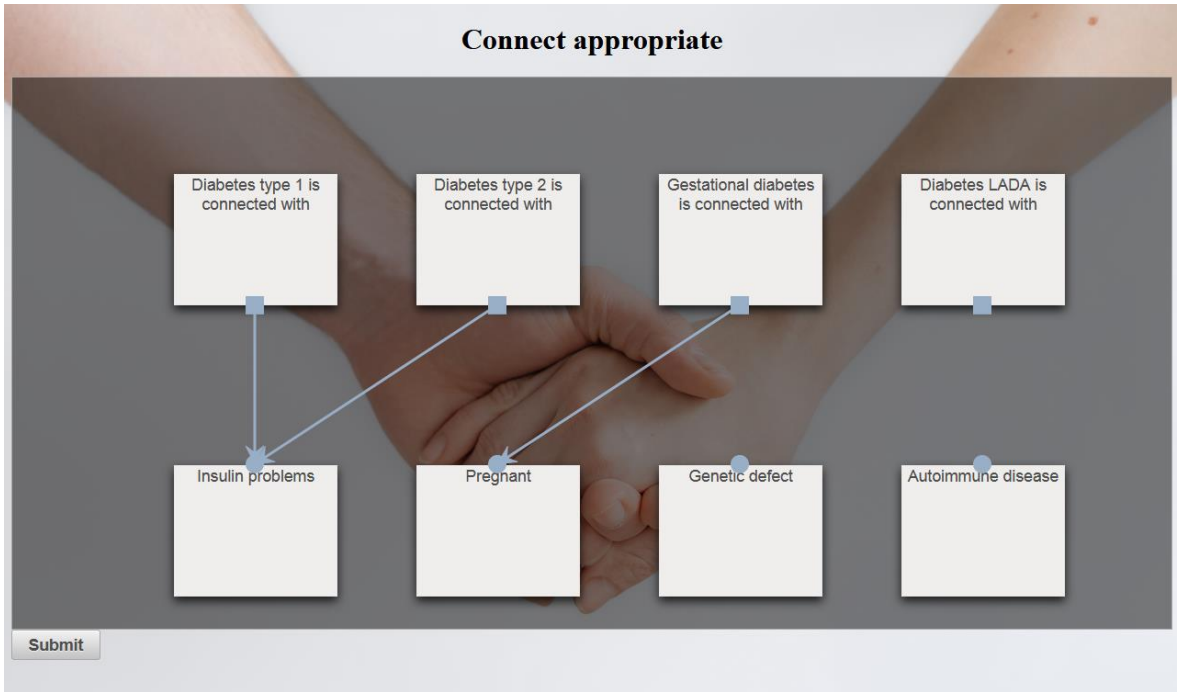


Fig. 6.4. Diagram task type

The task shown in Figure 6.5 provides the possibility to move elements from left to right and backwards. Player can choose one or multiple options and move selected ones to the right by first button seemed in the middle of two boxes. The second button provides possibility to move all elements located in the left side to right side. Two last buttons delivers the same functionality but in the opposite direction. To submit answer player has to click corresponding button under the boxes.

Fig. 6.5. PickList task type

The task shown in Figure 6.6 provides the possibility to drag and drop answers to the chosen container. Player can move answers to chosen boxes and also return them back by using “Return” button. There is possibility that none of the answers should be moved to other containers.

Fig. 6.6. DragDrop task type

In the Figure 6.7 the hall of fame containing the list of ten best players is presented. The data for the view is loaded from the database. The other elements shown in the figure are the current player's game result and the button encouraging to start the game again.

Fig. 6.7. Hall of fame page

6.5 Administration panel

The second functionality of the application is the administration panel. As mentioned earlier, it provides CRUD operations, so it makes it possible to modify every type of task. These functionalities can be achieved after login into panel.

In Figure 6.8 can be seen administration panel where the appropriate task editor can be chosen. In the remainder of this chapter, an execution path for the single-choice task is shown.

Fig. 6.8. Administrator panel home page

The Figure 6.9 presents the table of all tasks loaded from the database. There are columns with question, answer cases and the correct answer. As it can be noticed there exists convenient way of updating, deleting and creating tasks and coming back by using the buttons shown. As mentioned at the beginning not to repeat the code, the dialog window for updating and creating the task is the same except the data in input boxes and chosen buttons so only the update operation will be shown.

Id	Question	Case 1	Case 2	Case 3	Case 4	Answer	Update task	Delete task
1	What's elevated in the blood that causes diabetes?	Oxygen	White blood cells	Glucose	Fat	3	Update	Delete
2	Lack of what substance can lead to diabetes?	Insulin	Iron	Vitamin B12	Folate	1	Update	Delete
3	About what percentage of humanity are diabetics?	8.5	22.5	0.5	4	1	Update	Delete
4	What organ is diabetes related to?	Liver	Heart	Pancreas	Brain	3	Update	Delete
5	Where was the phrase "diabetes" used for the first time?	Italy	Egypt	India	Greece	4	Update	Delete

Fig. 6.9. Single-choice task administrator panel

As it can be seen on the Figure 6.10 there is a popup dialog which provides the possibility to change the data of the specific task. After clicking the update button, the administrator is getting feedback whether the operation was successful or not. There can be noticed also two additional buttons. First is “Delete” which provides possibility to remove object from database and in consequence from presented table. Last button is “Back to admin menu” which simply redirecting user back to administrator panel.

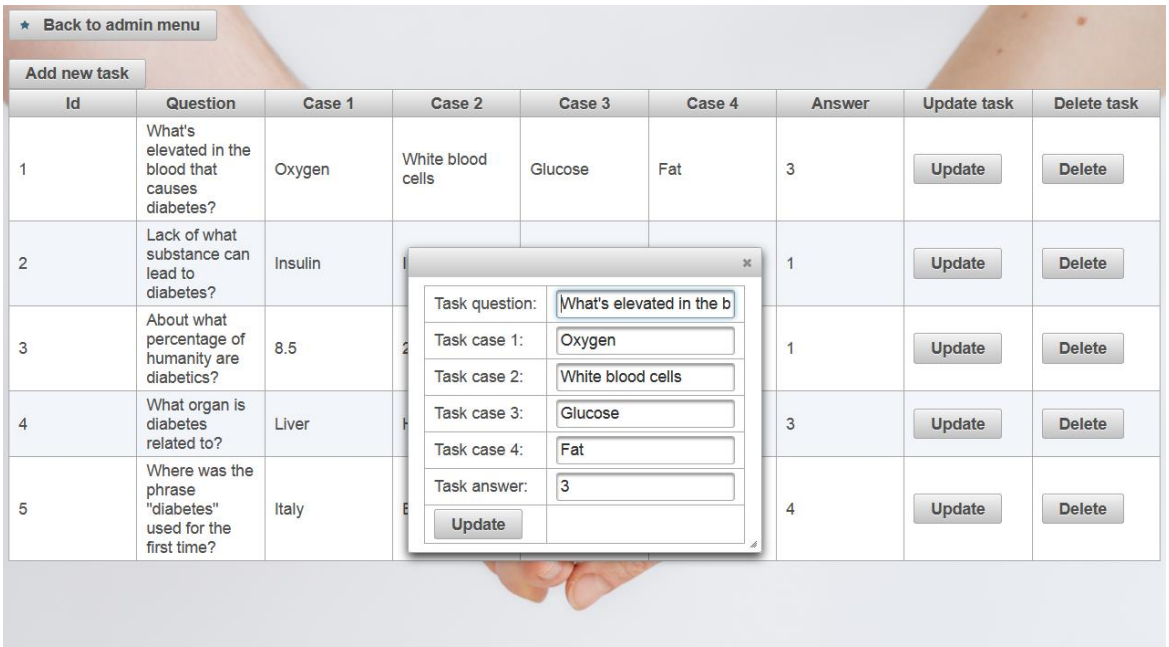


Fig. 6.10. Single-choice task updating

7. Verification

This chapter presents the applied testing methodology and selected test cases. Along with the evolution of software engineering, various techniques of testing the applications have been developed. Activity such as testing is one of most important issues because it allows to provide the software which simultaneously meets the functional and non-functional requirements and makes the programs reliable and maintainable.

7.1 Methodology of testing

Along with developing the application the method of testing has been converted into such technique as agile software development. It provides iterative and incremental development methods. As the application was expanding, the new functions had to be tested along with the previously developed ones, to assure that the integration between the features works correctly. The main way of checking integration was to add new feature in flow of application, then after deploying updated software it was necessary to run it and check in sequence if there is access from administrator panel to new task. Afterwards there were conducted tests of CRUD operations. From the administrator panel there were added new tasks which could be next updated. After this operations the unnecessary one were removed.

The most important issue was testing the operations related to the database. Taking into account that data contained in tables are in huge amount of cases necessary for the operation or even whole system it is very important to make sure that they are reliable. To prevent invalid SQL queries they were always checked in MySQL Workbench before passing into the source code. The example of this can be selecting first ten best players. There are some differences between SQL and JPA Query Language. The source code used in application is shown in Listing 7.1.

Listing 7.1. JPA Query Language example

```
1 Query query = session
2 .createQuery("select p from Player p ORDER BY p.points desc")
3 .setMaxResults(10);
```

As can be seen in Line 2 there is query similar to SQL one but referencing directly to *Player* entity class. Also in Line 3 can be seen method which cut result list to first ten records.

Corresponding SQL query which were used to test behavior of it can be seen here.

```
SELECT * FROM diabetes_carrier.player ORDER BY points DESC LIMIT 10;
```

Another very important thing to test was the business logic such as the verification if the user entered correct input and if there were no problems with rendering views. It was also necessary to limit the possibilities to cheat or receiving points which do no not belong to player. In relation with adding new types of tasks to the project there were necessity to check if their validation methods works correctly. To achieve such conclusions every new type of task was checking in debugging mode. There is possibility to see what current values of variables are. It also lets to test if the main flow of application is connected properly with specific types of tasks.

The next problem which should be looked closer at is the integration of the front-end and the back-end. JSF provides convenient methods to bring together components mentioned above. It was necessary to test if every button and action which is linked with logic works properly. In Table 7.1 are presented expected results of click.

Table 7.1

Buttons' expected results

Button value	Expected result
About	Redirect to page with links to literature used during developing content of tasks
Admin panel	Redirect to administrator panel login page
Start game	Check if user provided nick and start new game
Submit	Call validation method which would or not add point to player's result
Play again	Redirect to welcome page in order to start game again
Login	Call validation method and redirect into

	administrator panel if success
ABCD task editor	Redirect to specific task editor
Diagram task editor	
PickList task editor	
DragDrop task editor	
Add new task	Render dialog which lets to add new task
Update	Render dialog which lets to update new task
Delete	Remove specific task from database
Back to admin menu	Redirect back to administrator menu

The primary technique used to validate if the application works properly were the manual acceptance tests. The main assumption of such tests is to check whether the program works correctly by introducing some input and verifying if the received answer is consistent with the expected one.

7.2 Test cases

By expanding earlier thought the main goal of test cases was to validate if the application fulfills all assumed functionalities. Taking into account the fact that the majority of application is connected with user interaction, the verifications are directed into behaviors such like this. The application is constructed in such a way that if the administrator or player provides incorrect data the error message should appear.

It is a real challenge to make the application, which at the same time works perfectly fine and there is still possibility to make changes which do not destroy the structure of whole project.

This project is written using the software which is open-source which means that this application can be freely shared. The source code of the project would be available on GitHub to distribute to other developers.

It is worth to mention that all unexpected exceptions are caught and handled in a proper way, this means that the errors are logged. It ensures that the server does not crash and still makes the interactions with the users possible. In Figure 7.1 can be seen the dialog which appears when user try to start game without providing name.

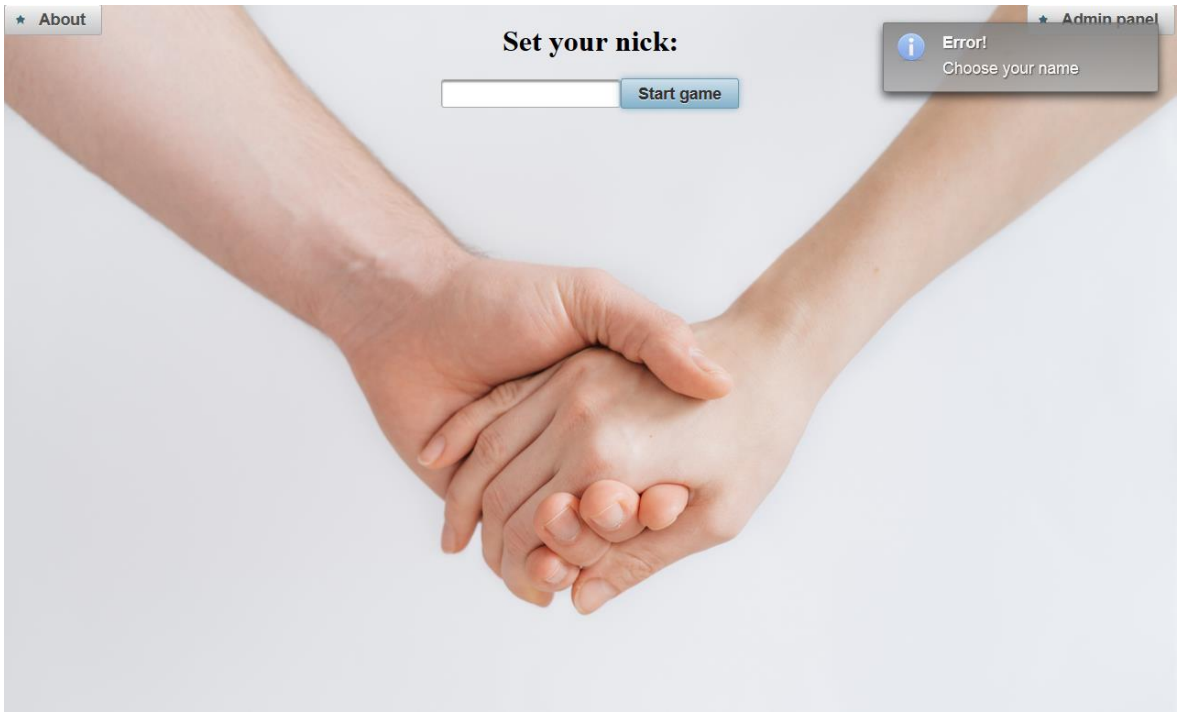
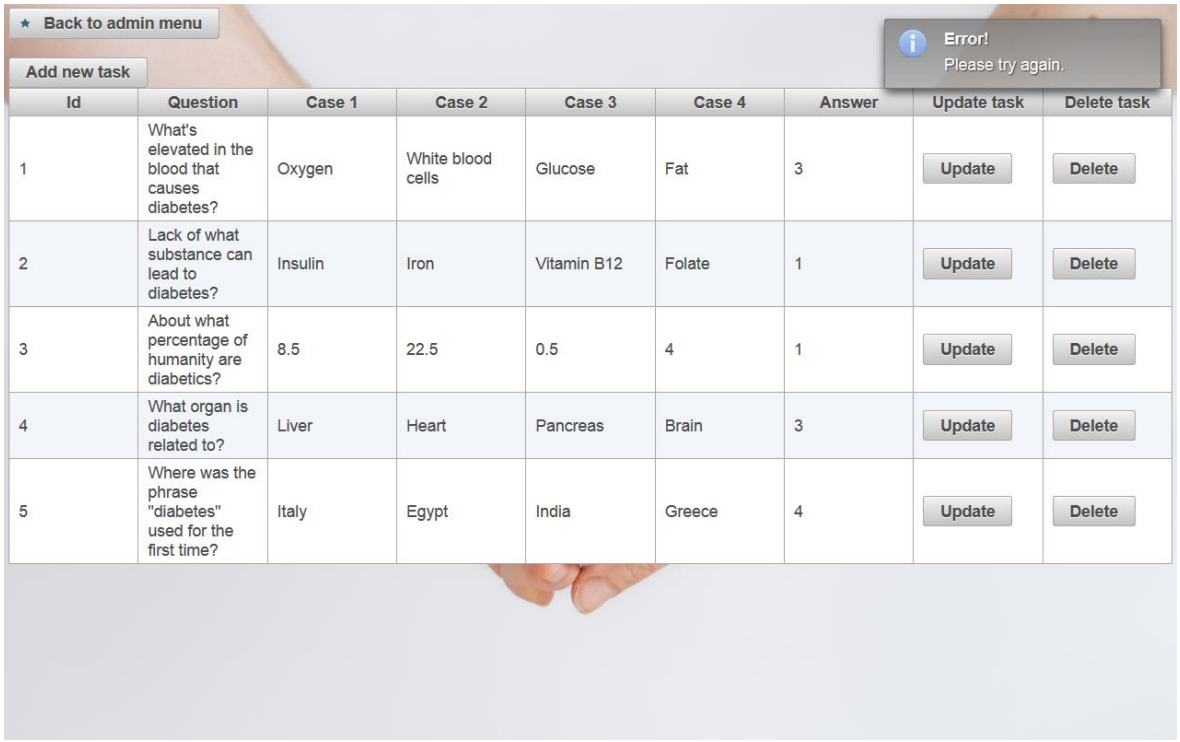


Fig. 7.1. Lack of name error

In Figure 7.2 can be noticed dialog with error when there occurred some problems with CRUD operations.



Id	Question	Case 1	Case 2	Case 3	Case 4	Answer	Update task	Delete task
1	What's elevated in the blood that causes diabetes?	Oxygen	White blood cells	Glucose	Fat	3	Update	Delete
2	Lack of what substance can lead to diabetes?	Insulin	Iron	Vitamin B12	Folate	1	Update	Delete
3	About what percentage of humanity are diabetics?	8.5	22.5	0.5	4	1	Update	Delete
4	What organ is diabetes related to?	Liver	Heart	Pancreas	Brain	3	Update	Delete
5	Where was the phrase "diabetes" used for the first time?	Italy	Egypt	India	Greece	4	Update	Delete

Fig. 7.2. CRUD operation error

In Figure 7.3 can be noticed error dialog which appears when user tries to login to administrator panel with incorrect data.

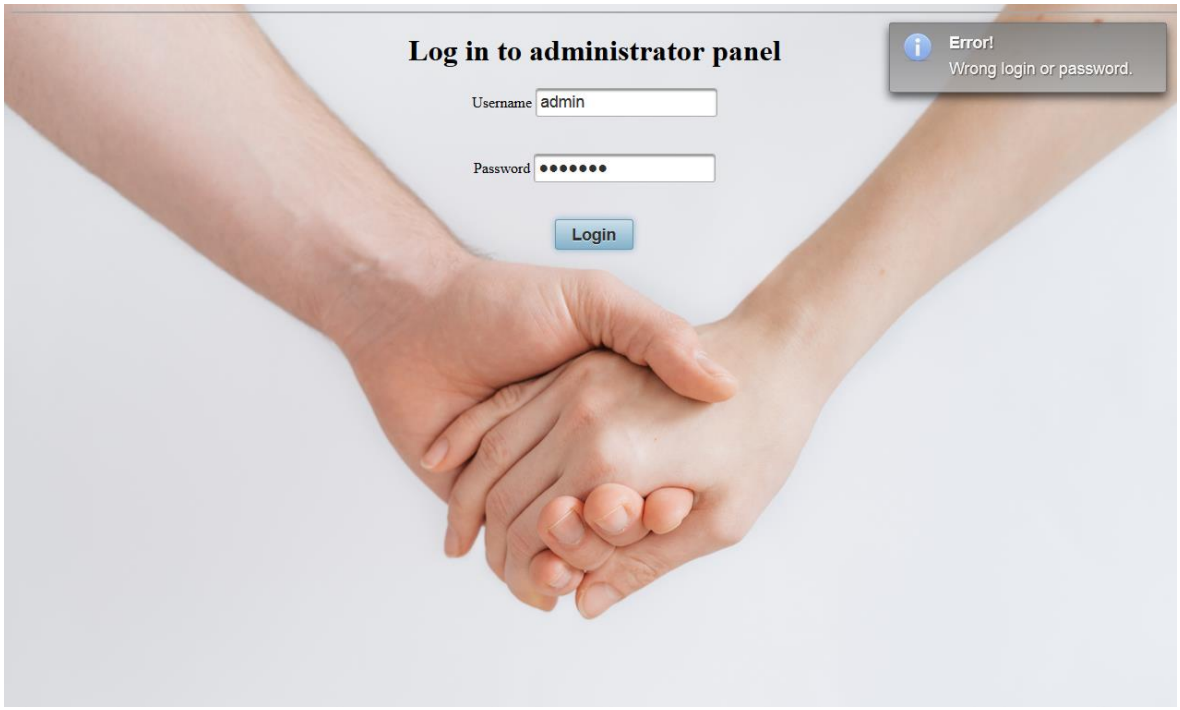


Fig. 7.3. Incorrect login or password error

Figure 7.4 presents error displayed at the time of lack of answer in single-choice task.

Fig. 7.4. Lack of answer in single-choice task.

8. Final remarks

This chapter describes the general results achieved and outlines some of the possible future directions of application development.

8.1 Final project's outcomes

The main goal of the project was fulfilled. The final outcome is an application which satisfies all functional and non-functional requirements assumed at the beginning. The application can be used not only for patients with diagnosed diabetes but also all other people which can be the potential victims of this disease.

Various exercises and tasks can bring closer what this disease is in a fast and pleasant way. For every type of task there are three exercises drawn at random. There is possibility for various interactions with player. The first task is a single-choice question in which only one answer is correct. The second one is diagram in which user get four questions and four answers and has a chance to choose some questions with any other answer. Third task is pick list in which player could freely transfer answers from left side box to right side which contains chosen objects. The last task gives opportunity to drag answers and drop it to appropriate containers. All tasks provides possibility to change chosen answer before submit. The consequence of game is saving of player's result which make opportunity to compete with another users.

The application also makes possibility to use administrator panel for convenient way of modifying existing tasks or adding new ones. It fulfills completely initial functional requirements.

Except for the functional requirements mentioned above, the non-functional requirements are also fulfilled. They include low response times obtained with the help of continuous refactoring and reliability provided by Java programming language. It is also secured in the way that unprivileged users cannot get access to administrator's panel pages. It is achieved by using filter method. Moreover application is providing real-time system and can be used on various operating systems.

8.2 Possible future development

In modern software developing the applications which are able to be extended is a must. In relation with open-source environment which is becoming more and more popular, the projects can be developed by programmers from all over the world.

The most important thing that application can become is using it as template for another types of quizzes. It can be not only directed to make exercises for raising awareness about diabetes but also for other purposes such as teaching about cancer or any other disease. By The administrator panel provides capabilities to reconstruct whole database of tasks,

Thanks to the use of the MVC pattern there is no problem with adding new tasks. It is only necessary to stick to the assumptions of architecture patterns used and implement appropriate interfaces. The administration panel with implemented CRUD operations makes creating new tasks very easy.

The last important thing that can be implemented is a relation of this web application to mobile devices. In times when everyone has his own smartphone which is inseparable part of modern life it is necessary to raise awareness about such a disease as diabetes.

Literature

1. J Evans B., Flanagan D., Java w pigułce. Wydanie VI, przeł Piwko Ł., Helion, Gliwice 2015
2. Bloch J., Java.Efektywne programowanie. Wydanie II, przeł Gonera P., Helion, Gliwice 2009
3. C. Martin R., Czysty kod. Podręcznik dobrego programisty, przeł Gonera P., Helion, Gliwice 2010
4. Gamma E., Helm R., Johnson R., Vlissides J., Wzorce projektowe. Elementy oprogramowania obiektowego wielokrotnego użytku, przeł Walczak T., Helion, Gliwice 2010
5. Thinking in Java. Edycja polska. Wydanie IV, przeł. Szeremiota P., Helion, Gliwice 2006
6. Diabetes. http://www.who.int/diabetes/en/ [access: december 2017]
7. NetBeans IDE 8.2 Release Notes. https://netbeans.org/community/releases/82/relnotes.html#new [access: December 2017]
8. MySQL Workbench Manual. New in MySQL Workbench 6.3. https://dev.mysql.com/doc/workbench/en/wb-what-is-new-63.html [access: December 2017]
9. The Principles of OOD. http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod [access: November 2017]
10. Yener M., Theedom A., Java EE. Zaawansowane wzorce projektowe, przeł. Piwko Ł, Helion, Gliwice 2015
11. Java Platform, Enterprise Edition 7 SDK Update 2 – Release Notes http://www.oracle.com/technetwork/java/javaee/documentation/javaee7sdk-readme-1957703.html [access: February 2016]
12. Java Platform, Standard Edition Installation Guide. https://docs.oracle.com/javase/8/docs/technotes/guides/install/windows_system_requirements.html [access: December 2017]
13. GlassFish Server Open Source Edition. Release Notes. Release 4.1 https://javaee.github.io/glassfish/doc/4.0/release-notes.pdf [access: December 2017]

14. GlassFish Server Open Source Edition. Installation Guide. Release 4.0 https://javaee.github.io/glassfish/doc/4.0/installation-guide.pdf [access: December 2017]
15. MySQL 5.7 Reference Manual. Installing MySQL on Microsoft Windows. https://dev.mysql.com/doc/refman/5.7/en/windows-installation.html [access: December 2017]

Description of the contents of CD-ROM

- */bin* directory which contains war file of application,
- */bin/install* directory which contains installers of programs needed to deploy application such as: *jre-8u152-windows-x64.exe*, *jre-8u152-windows-i586.exe* for installing *JRE 8* for the Windows 64 and 32 bits versions, *glassfish-4.1.zip* for *GlassFish 4.1 Server*, *mysql-installer-web-community-5.7.20.0.exe* which provides possibility to install MySQL components,
- */bin/lib directory* which contains files such as: *javax.faces-2.2.8.jar*, *mysql-connectior-java-5.1.23-bin.jar* and *primefaces-6.1.jar*. First one is for *JSF 2.2*, second for obtaining connection with MySQL server and last one for *Primefaces 6.1*,
- */bin/lib/hibernate* directory which contains libraries required by *Hibernate,*
- */doc* directory which contains thesis in PDF format,
- */javadoc* directory which contains generated *Javadoc* source code documentation,
- */src* directory which contains source code of application.