# Internet Application for On-line Testing and Examination

Dumitru Dan BURDESCU, Cristian MIHAESCU
Faculty of Automation, Computers and Electronics, University of Craiova

*The paper presents an Internet application for on-line testing and examination. The students may sustain on-line tests or examinations at any matter as many times as they want. Besides the operations presented professor may modify the content of the course regarding the number of chapters, their content or the maximum number of seconds that a student may use to give an answer. The professor may add, delete or modify questions.*
***Keywords:*** *Internet, Java, database, template, classes, servlet.*

## Introduction

This paper presents the structure and facilities of an Internet application that ensures on-line testing and examination for students of a university. The goal of the application is to provide students and professors an integrated environment for sustaining tests and exams.

Within the application the main roles are: secretary, professor and student. The secretary manages all data concerning professors, students or matters of study. The secretary has the goal to create the general infrastructure in which students and professors will work. Each professor manages one or more matters so that students may sustain self-testing or examination. Under these circumstances for a matter the professor has the possibility to create as many chapters as he needs. For each chapter the professor manages all questions that will be part of testing or examination procedures. There are two types of questions: questions used for testing and questions used for examination procedure. Because some chapters are more difficult than others, the professor may set the maximum number of seconds that a student may use to give an answer. For each chapter there may be specified the number of randomly extracted questions for a student that wants to take a test from that chapter. In the same way the professor creates the structure of examination regarding the number of exam questions randomly extracted from each chapter.

Besides the operations presented professor may modify the content of the course regarding the number of chapters, their content or the maximum number of seconds that a student may use to give an answer. The professor may add, delete or modify questions.

From the point of view of the student, one may sustain on-line tests or examinations at any matter as many times as he wants. All he has to do is to select the matter and than the chapters from which he wants to take the test. Taking exams is accomplished in a more strict way regarding examination periods specified by the secretaries.

Both during testing and examination questions are presented one at a time. Besides the text of the question and possible answers other information such as elapsed time and question number is presented. The secretary, the professor that manages that matter or by the student himself, may consult all tests or examinations sustained by a student. Sustaining exams is accomplished in a similar way to testing. The only difference is that the student does not select the chapters because the questions are randomly extracted from all chapters according to professor's specifications.

## The Structure of the Application

In order to achieve best quality and security of application the users have been differentiated in three categories (roles). From this point of view the application interacts with students, professors and secretaries. Each role has its own set of allowed actions that may be performed. Users that prove to have a certain role may execute only the actions assigned to that role. Each user must authenti-

cate supplying a username and a password before executing actions. After a user authenticates the application determines his role and creates the set of actions that user may run. The set of actions corresponding to a certain role is automatically loaded from a properties configuration file. In order to increase security the sets of actions corresponding to a role are disjoint, so that a user authenticated to a role may not execute actions that are characteristics to other role (e.g.: a user with student role may not add, delete or update a question because these actions may be exclusively performed by users that have professor role).

At lowest level the application manages a relational database. The main relations present in the database are presented in the following lines.

➢ *USER (id,username,password,nume,prenume)* – this relation manages registered user. The most important fields from this relation are:

- *username* – username of the user;
- *password* – password of the user;
- *nume* – second name of the user;
- *prenume* – first name of the user;

➢ *ROLE (id,name)* – this relation manages the roles that the application recognizes. The roles are managed automatically through a configuration file of the application.

➢ *USERROLE (userid,releid)* – this relation gathers information regarding the role that a certain user has. When a new user is added in USER relation, the role of that user is specified through a new entry in *USERROLE* relation.

➢ *MATERII (id,sectieid,an,sem,denmaterie,profid,stare,tipcalcul)* – this relation manages subject matters to which students may sustain tests or examinations. The most important fields from this relation are:

- *sectieid* – identifies the department that where that subject matter is studied;
- *an* – the year in which the subject matter is studied;
- *sem* - the term in which the subject matter is studied;
- *denmaterie* – name of the subject matter;
- *profid* – identifies the professor that manages the subject matter;
- *stare* – status of subject matter (A – activated, D - deactivated);
- *tipcalcul* – identifies the formula through which the results are obtained;

➢ *CAPITOLE (id, materieid, dencapitol, secondsperq, qfromexam, qfromtest, stare)* – this relation manages the chapters that belong to a subject matter. The most important fields from this relation are:

- *materieid* – identifies the subject matter to which the chapter belongs;
- *dencapitol* – name of the chapter;
- *secondsperq* – number of seconds necessary to answer to a question from the chapter.
- *qfromexam* – number of randomly extracted exam questions from the chapter when an examination is needed;
- *qfromtest* – number of randomly extracted test questions from the chapter when testing is needed;
- *stare* – chapter status (A – activated, D - deactivated);

➢ *INTREBARI (id,capid,text,corectans,visibleans,stareAD,stareTE)* – this relation gathers all questions. The most important fields from this relation are:

- *capid* – chapter to which questions belongs;
- *text* – text of the questions;
- *corectans* – the correct answer of the question;
- *visibleans* – visible options for student;
- *stareAD* – status of question (A – activated, D - deactivated);
- *stareTE* - status of question (T – test question, E – exam question);

➢ *REZULTATETESTE (id, userid, materieid, datatest, result, tipcalcul, chapters)* – this relation gathers the results to all tests sustained by students. The most important fields from this relation are:

- *userid* – identifies the user (student) that sustained the test;
- *materieid* – identifies the subject matter to which the student was tested;
- *datatest* – date when the test was sustained;

- *result* - the result of the test;
- *tipcalcul* – identifies the formula through which the result was obtained;
- *chapters* – chapters from which the test was sustained;

➢ *INTREBARITESTE (id, rezulttestid, raspunsprof, raspunsstud)* – this relation gathers questions that were used for testing. Professors or students may use the data from this relation in order to recreate the tests that were sustained. The most important fields from this relation are:

- *rezulttestid* – identifies test to which the question belong ;
- *text* – test of the question;
- *raspunsprof* – professor's answers;
- *raspunsstud* – student's answer;

➢ *REZULTATEEXAMENE (id, userid, materieid, dataexam, result, tipcalcul)* – this relation gathers the results to all exams sustained by students. The most important fields from this relation are:

- *userid* – identifies the user (student) that sustained the exam;
- *materieid* – identifies the subject matter to which the student sustained examination;
- *datatest* – data of examination;
- *result* - the result of the examination;
- *tipcalcul* - identifies the formula through which the result was obtained;

➢ *INTREBARIEXAMENE (id, rezultexamid, raspunsprof, raspunsstud)* – this relation gathers questions that were used for examinations. Professors or students may use the data from this relation in order to recreate the exams that were sustained. The most important fields from this relation are:

- *rezultexamid* – identifies exam to which the question belong;
- *text* – text of question;
- *raspunsprof* – professor's answers;
- *raspunsstud* – student's answer;

Registered users that are allowed to execute actions according to their role manage all data from the database exclusively on-line. Users that have professor role may add, delete or modify records from CAPITOLE and INTREBARI relations. When a student wants to sustain a test or an exam, the application creates the test following professor's specifications. When testing procedure finishes, the application ads records into *REZULTATETESTE, INTREBARITESTE, REZULTATEEXAMENE, and INTREBARIEXAMENE* relations. Over the records from these relations no user can perform such as update or delete. Professors or students may only retrieve information. For example, a student may view only the results of his own tests and a professor may view only results of tests sustained at subject matters that he manages.

The *INTREBARITESTE* and *INTREBARIEXAMENE* relations allow professor and student to view all questions that were randomly extracted for a test or an exam.

In the next figure it is presented the general functional schema of the application.

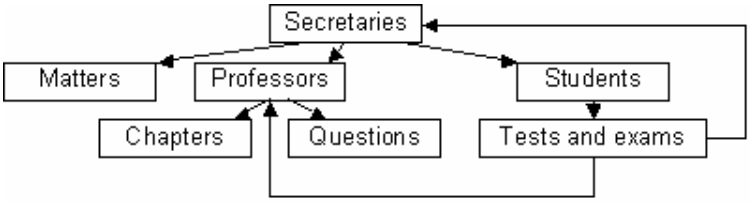


**Fig. 1.** General functional schema of the application

A professor performs maters, chapters, and questions management with the help of template mechanism. A professor has at its disposal the following templates.

- A template in which are presented the subject matters that he manages. In this template, for each matter the professor has two options: to manage the content of the matter (chapters, questions, etc.) and to manage the students that may sustain tests and examinations to those matters.
- A template for managing a matter. With the help of this template professor may add/delete or activate/deactivate a chapter.

From this template professor has access to:

- Templates used for managing questions for testing or examination;
- Templates used for specifying the number of questions that will be randomly selected when a student sustain a test or an exam;
- Templates used for specifying the number of seconds necessary for a student to answer to a question from a specific chapter;

- A template that presents the list of questions used for testing or examination from a certain chapter. From this template a professor may add a new question, edit or delete an existing question.
- A template that presents the list of students that may sustain tests or examinations to a certain matter. From this template professor has access to:

- Template for visualization of tests sustained by a student;
- Template for visualization of exams sustained by a student;

- A template used for managing the content of a question. The content of a question may be modified or previewed. Within the content images (of type .jpg or .gif) may be inserted in order to give the possibility to create questions that contain equations, special chapters, etc.

In the next figure it is presented the template used by professors for subject matters administration.

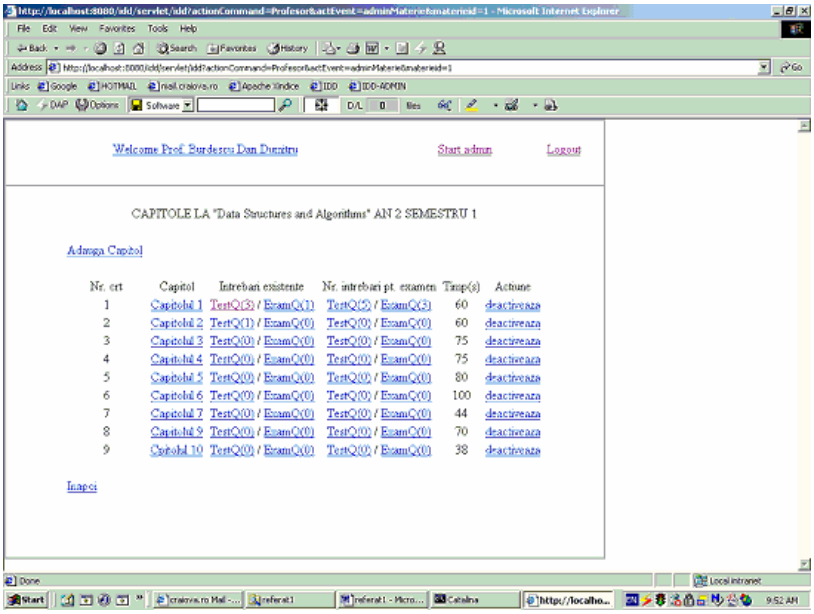


**Fig. 2.** The template for matter subjects administration

From the point of view of the professor, another important template is that one used for adding a question. In the next figure it is presented the template used by professor to add a new question. The template used for editing an existing question is similar.

When editing a question the professor must specify the correct answers and the visible options that the student may select when answering a question. The question may be previewed in such a manner that the professor may see exactly what the student will see when sustaining the test.

A student manages his account and sustains tests with the help of template mechanism. A student has at its disposal the following templates.

- A template which contains the main options: changing password, starting a test or an examination or presentation of all study matters;
- A template for changing the password;
- A template with presentation of all study matters;
- A template with presentation of all sustained tests;
- A template where the student may select the chapters from which the questions for testing will be selected;

- A template with subject matters for which the student may sustain examination;
- A template that is used to present a question during testing or examination. In this template the student will check the correct answers;

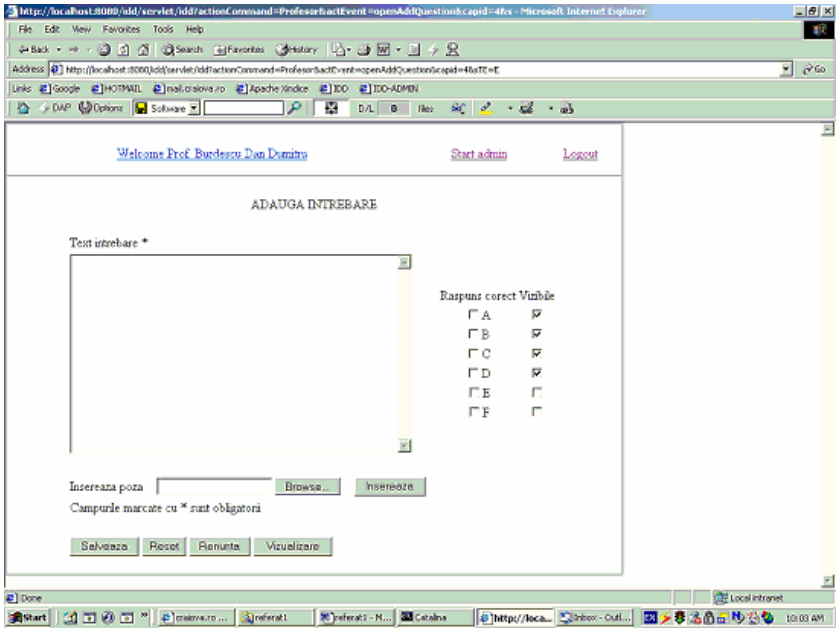


**Fig. 3.** Template used for adding a question

In the next figure it is presented the template used for presentation of a question, where the student must fill the correct answers.

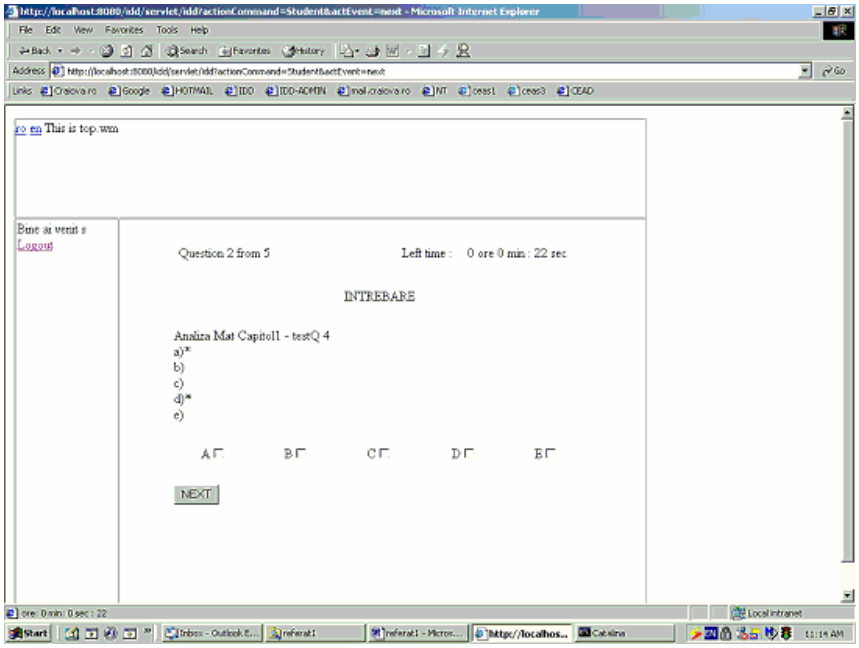


**Fig. 4.** Template for filling the correct answers

All templates are dynamically filled with data.

**Technological aspects**

The technology used for implementing the application is based on MVC (Model – View

- Controller) three-tier architecture. In the next figure it is presented the structure of the application corresponding to the MVC model.

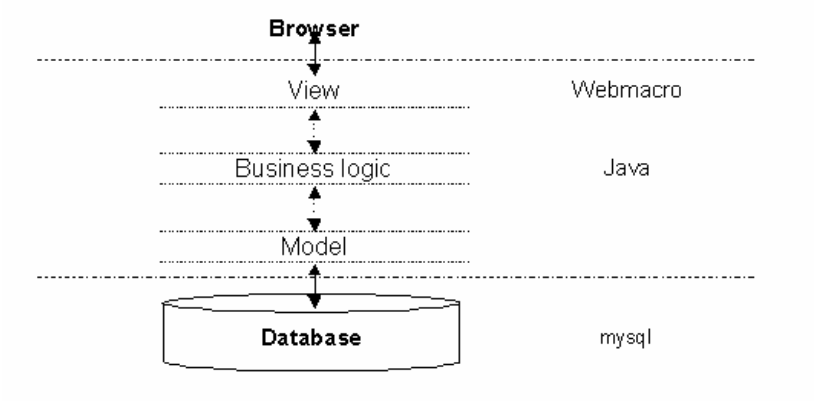


**Fig. 5.** MVC Model

This architecture is highly recommended for dynamic Internet applications that need implementation of medium or high complexity actions. It also clearly separates the logical parts of such an application: the model (database), the business logic and the view (templates). This separation presents major advantages in processes of development, deployment, maintenance or upgrading. All these processes may be accomplished by three groups of programmers each one working at just one level from the MVC architecture. This is possible because MVC model implementation realizes a true separation between tiers.

The DBMS (Database Management System) the application uses MySql. This product has many advantages. One of the most important advantages is that it is *open source*. This fact reduces the cost of the application regarding licenses. The quality of system regarding response time and possibility to process multiple requests simultaneously is proved by numerous applications that use MySql as DBMS. With all this, if the database would become extremely large and response times would become unacceptable large it is highly recommendable using a proprietary DBMS such as Microsoft SQL Server 2000, Oracle, etc. is not a problem. This may be accomplished due to the fact that business logic is implemented in Java and queries sent to DBMS respect SQL standard. Connecting from Java to DBMS is done through JDBC (Java Database Connectivity). This fact allows querying almost any DBMS because for all database servers there were developed appropriate drivers.

Access to database from Java is implemented in such a manner so that more users may use the application at the same time. In order to make this possible, there is implemented a pool of connections from where a connection object may be obtained whenever needed. Connections management is implemented in Java class specially designed for this purpose.

Using Java for implementing business logic level offers more advantages. First of all the application becomes independent from the platform where it runs, whether it is Microsoft Windows, Linux, Unix or Mac. The fact that Java is a pure object oriented programming language represents another advantage. Existence of scalable software architecture ensures a development and maintenance that can be performed extremely efficient. From architecture point of view the business logic level is implemented through a java class hierarchy.

At the lowest level there are classes of type *bean*. Private members of these classes are loaded with data corresponding logically to a record from corresponding table from database. The main *bean* classes are:

- *User* – class that is loaded with user's data;
- *Materie* – class that is loaded with data regarding a subject matter;

- *Capitol* - class that is loaded with data regarding a chapter;
- *Question* - class that is loaded with data regarding a question;
- *Test* - class that is loaded with data regarding extracted questions for testing;
- *Exam* - class that is loaded with data regarding extracted questions for examination;

Classes of type *manager* use classes of type *bean*. *Manager* classes interact directly with database. Each *manager* class implements basic operations like insert, edit, delete. Main *manager* classes are:

- *UserManager* – implements all operations regarding users of application, whether they are professors or students;
- *MateriiManager* – implements all operations regarding subject matters used by application for sustaining a test or an examination;
- *CapitoleManager* – implements all operations regarding chapters used by application;
- *QustionsManager* – implements all operations regarding questions used by application;
- *RezultateManager* – implements all operations regarding results obtained after students sustain tests or exams;

Classes of type *action* use classes of type *manager*. *Action* classes identify and perform the task sent by the user. A*ction* classes allow execution of actions conform to the role of the user. When a user wants to execute an action, his role is identified and after that it is verified that the wanted action is among the actions corresponding to that role.

The main *action* classes are:

- *Welcome* – presents the start page for any user before authentication;
- *LoginUser* – authenticates an user (verifies username and passwords), identifies his role and creates the set of permitted actions according to obtained role;
- *Secretary* – executes actions asked by users that have *secretary* role;
- *Profesor* - executes actions asked by users that have *professor* role;
- *Student* - executes actions asked by users that have *student* role;
- *Logout* – executes necessary actions whenever a user ends his session;

Business logic software architecture is coordinated through *servlet* technology. A central servlet receives all requests from the client (browser) and according to parameters that are sent (username, password, action, event, etc.) it runs appropriate java classes producing the response. After execution it is determined the next template which is filled dynamically with data. In the next figure it is presented the Java class hierarchy used for implementation of the business logic of the application.

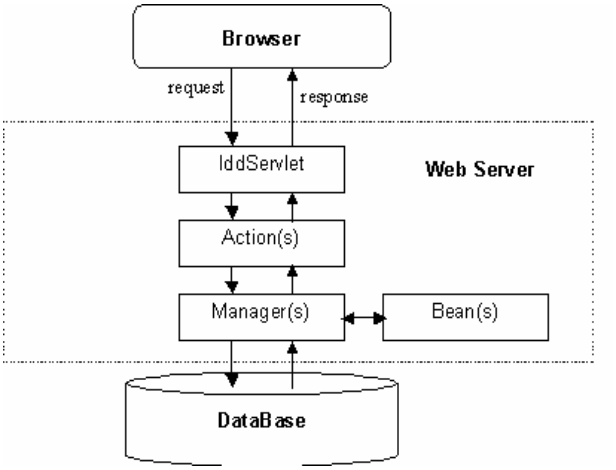


**Fig. 6.** Java class hierarchy

As template mechanism is used *webmacro* technology. This technology allows easy template design, development and maintenance. It is also ensured a good and efficient

separation between business logic model and view model.

**Conclusions**

The application is based exclusively on powerful *open source* technologies. These technologies have been implemented with very good results. Under these circumstances the price of the entire application decreases no licenses being needed.

Security was carefully taken into consideration in the design phase of the application. Generally speaking, security at network level is a primary concern. In fact, the security of the computer host where the application runs is of major importance. The security at application level must be well taken care because many users and especially type of users interact with it. Further development may introduce other type of uses, with permissions. Using different type of users each one of them having the permission to run a specific set of actions presents a fine granularity and good control of what application performs. More than this specifying the rights of the application at table level ensures the security at database level.

Platform independence is one of the main needed approaches regarding this kind of application. This task is fulfilled by using Java at business logic level. More than this, using servlet technology that has become standard the application may run on great majority of web servers. Using Java a high level of scalability is reached speaking about a pure object oriented programming language.

**References**

1. Dumitru Dan Burdescu, Cristian Mihaescu, **"**Portal pentru testare si examinare on-line**"** (Portal for Testing and Examination), National Symposium for Software Technologies for Electronic Platforms in Technical Educational System, Compress, Bucuresti, 2003

2. Cristian Mihaescu, Sorin Scortan - "Securitatea la nivel de aplicatie – analiza comparativa a solutiilor oferite de Sun si Microsoft" (Security at Application Level – comparative analisys for solutions offered by Sun and Microsoft), NetReport, May 2002, Ed. Agora

3. Cristian Mihaescu, Mihai Mocanu – Programare distribuita în Java (Distributed Programming in Java), NetReport, April 2003, Ed. Agora

4. http://java.sun.com

5. http://www.mysql.com

6. http://www.webmacro.org