



**DUBLIN CITY UNIVERSITY
SCHOOL OF ELECTRONIC ENGINEERING**

**Java Server side development
of a XML based
assessment system**

Slawomir Gruca
August 2002

**MASTER OF ENGINEERING
IN
TELECOMMUNICATIONS**

Supervised by Dr Derek Molloy

Acknowledgements

I would like to thank my supervisor Dr Derek Molly for his guidance, enthusiasm and commitment to this project. While I was working on it, Dr Molloy was always there, ready to offer me his advice and help.

I also wish to express my gratitude to Dr Noel Murphy, the coordinator of the student exchange between DCU and my Polish school – Poznań University of Technology. Despite the fact that Dr Murphy was often busy, he always had time for me and I could always count on his helping hand.

Finally thanks are also due to all DCU staff, who was always friendly, sincere and helpful.

Declaration

I hereby declare that, except where otherwise indicated, this document is entirely my own work and has not been submitted in whole or in part to any other university.

Signed:Date:

Abstract

The general idea of this project is to devise the means, which allow web based on-line testing for distance learning purposes. The goal of this work is not only inventing a way that it may be accomplished but also constructing a working system. The system is supposed to be integrated and fully featured. Moreover, it should be trustworthy and ready to use right away. Another issue is accessibility of the system for end users, which has to be taken into consideration. Ideally, the final product should be flexible, compact, robust and user friendly. Those are contradictory aims and dealing with needs them is not necessarily straightforward.

This project explores several modern technologies. The software is written in Java programming language and uses s new Servlet API. The project's data storage involves SQL database and a young standard for data description called XML. This standard is accompanied by some others, like DTD, XML Schema, XSL, and many of them are tackled and discussed. Finally, a brand new, just emerging technology, namely JAXB, constituting a bridge between XML and Java is an essential part of this project.

The crown of a done work is a compilation of above stated assumptions about the system. The final application is complete and can possibly be used in a real life distance learning system.

Table of Contents

ACKNOWLEDGEMENTS	II
DECLARATION	II
ABSTRACT	III
TABLE OF CONTENTS	IV
TABLE OF FIGURES	VI
1. INTRODUCTION	1
1.1. XML	3
1.1.1. XML applications	5
1.1.2. XML basics	7
1.1.3. XML companion standards.....	9
1.2. JAXB	13
1.2.1. Working with JAXB	14
1.2.2. JAXB Summary.....	17
1.3. SERVLETS	18
1.3.1. Applications of servlets.....	19
1.3.2. Overview of servlets	21
1.3.3. HTTP Support.....	23
1.3.4. Saving client state.....	25
1.3.5. Servlet Summary.....	26
1.4. RELATIONAL DATABASE AND SQL	27
2. THE SYSTEM	28
2.1. THE SYSTEM'S DESCRIPTION	29
2.1.1. The question format.....	29
2.1.2. The system's basics.....	30
2.2. THE IMPLEMENTATION.....	41
2.2.1. The XML test's structure	41
2.2.2. DTD and XML Schema.....	44
2.2.3. JAXB.....	46

2.2.4. XSL	49
2.2.5. The database and SQL.....	52
2.2.6. The servlets.....	55
2.2.7. Other technologies incorporated into the project	65
3. INSTALLATION AND CONFIGURATION	67
3.1. SOFTWARE.....	67
3.1.1. Microsoft Windows98.....	67
3.1.2. Sun Java 2 SDK.....	67
3.1.3. Sun Java Architecture for XML Binding	67
3.1.4. Apache Tomcat web server.....	67
3.1.5. MySQL Database.....	68
3.2. HARDWARE	68
4. CONCLUSIONS.....	69
REFERENCES	71
REFERENCES	71
APPENDIX	73
A.1. ESSENTIAL PROJECT FILES.....	73
A.1.1. XML Schema for the project's XML data format (test.xsd)	73
A.1.2. DTD for the project's XML data format (test.dtd).....	75
A.1.3. JAXB binding schema for the project's XML data format (test.xjs)	76
A.1.4. XSL style sheet for the project's XML data format (test.xsl)	77
A.1.5. CSS style sheet for HTML's display formatting (style.css).....	80
A.1.6. The project's web deployment descriptor (web.inf).....	81
A.2. OTHER FILES	84
A.2.1. XML file containing a sample test (test.xml).....	84
A.2.2. The index page (index.html).....	85
A.2.2. The forbidden page (forbidden.html)	86
A.3. A FLOPPY DISK WITH ALL PROJECT'S FILES	87

Table of Figures

FIGURE 1. JAXB – GENERATION OF CLASSES	14
FIGURE 2. JAXB - UNMARSHALLING	15
FIGURE 3. JAXB - MARSHALLING.....	16
FIGURE 4. LIFECYCLE OF THE SERVLET.....	22
FIGURE 5. DISPATCHING OF HTTP REQUEST	24
FIGURE 6. THE WELCOME PAGE	30
FIGURE 7. THE FORBIDDEN PAGE	30
FIGURE 8. THE LOGIN SUBSYSTEM.....	31
FIGURE 9. THE LOGIN PAGE	32
FIGURE 10. THE LOGOUT/RELOGIN PAGE	32
FIGURE 11. THE MANAGER MENU	32
FIGURE 12. THE MANAGER SUBSYSTEM.....	33
FIGURE 13. THE TEST EDITOR	34
FIGURE 14. THE STRING MATCH QUESTION EDITOR.....	35
FIGURE 15. THE SINGLE AND MULTIPLE CHOICE QUESTION EDITOR	35
FIGURE 16. THE RESULTS CHECKER.....	36
FIGURE 17. THE RESULT CHECKER – AN EXTERNAL EVALUATION TYPE QUESTION	36
FIGURE 18. THE TESTING SUBSYSTEM.....	37
FIGURE 19. THE SAMPLE OF GENERATED TEST’S INTRODUCTORY PART	38
FIGURE 20. THE SAMPLE OF GENERATED SINGLE CHOICE QUESTION FORM (WITH THE NOTICE)	38
FIGURE 21. THE SAMPLE OF GENERATED MULTIPLE CHOICE QUESTION FORM.....	38
FIGURE 22. THE SAMPLE OF GENERATED STRING MATCH QUESTION FORM	39
FIGURE 23. THE SAMPLE OF GENERATED EXTERNAL EVALUATION QUESTION FORM (WITH THE NOTICE).....	39
FIGURE 24. THE SUBMIT AND RESET BUTTONS.....	39
FIGURE 25. REDISPLAYED SAMPLE MULTIPLE CHOICE QUESTION.....	40
FIGURE 26. REDISPLAYED SAMPLE STRING MATCH QUESTION	40
FIGURE 27. THE CONFIRMATION PAGE.....	40
FIGURE 28. THE RESULTS PAGE	40
FIGURE 30. UNFORMATTED HTML PAGE.....	65
FIGURE 31. HTML PAGE FORMATTED BY THE CSS STYLE SHEET.....	65

Chapter 1

1. Introduction

The idea of on-line examination through the Internet is becoming more popular these days, since the global network has become easily accessible by almost everyone. There is plethora of courses available on the web - they are offered by schools, universities and different kinds of companies. Most of those courses are yet limited to only presentation of some information in, hopefully, interactive form. Even this task can sometimes be awkward to be accomplished, if many different courses are to be presented through a single web site. Moreover, users may use different kinds of web browser or may want to ask for notes in some specific format, like for example PDF. In those cases there have to be some means available which help to prepare the notes in a unified format and data prepared in that format should be accessible through different devices. The Y-box developed in DCU is an exemplary solution to those problems. It uses XML for notes storage and makes them available to the most popular web browsers and PDAs.

On-line knowledge testing is a nice form of extending e-learning application. It is a next, natural step after presentation of notes. The idea is relatively simple – a user after reading notes is being presented a form with questions. He/she fills in the form and submits it. The form is then processed and a result of evaluation is put into a database and probably displayed to the user. Although it appears to be easy, in fact many issues arise when one wants to construct such system.

The first problem, which has to be addressed is questions preparation and their storage. A potential tutor, who wants to extend his course notes by a test, has to be given a specification of a test format and what is the most important, the format must be easy to understand and compact but flexible at the same time. Here is where XML comes to rescue. A properly and carefully designed XML structure can be almost a perfect storage for the tests.

The next problem arising is connected to presentation of questions to a user. One can assume that it is done through a web browser, but it does not solve the problem. Still, there is a need for conversion between the format used for storage (XML in the case of this

project) and the format accepted by a web browser which happens to be HTML. This project makes use of XSL for that conversion.

Beside the fact that there is a necessity of dealing with conversion from storage to display format, another conversion is necessary – from storage to application data, to be more exact, to objects in a computer memory. Without that, automatic evaluation of user answers was not possible. A brand new standard JAXB is designed to do that conversion for us. It acts as a link between XML and Java objects. Since JAXB is an essential part of this project, a substantial part of the report is devoted to it.

Another thing that one has to consider is communication between an application and a web client. Also some environment to run the actual application is needed. Java servlets are a very convenient solution that can be used here. They extend functionality of a web server and give elegant solutions to many problems connected to communication between server and client through HTTP interface. Servlets are actually small programs themselves and they work on the server side. The whole system described in this report is based on several servlets which constitute functionality of the final application.

Finally, the system has to cope with different, sometimes strange user's behaviours. It is not supposed to crash even if a user fills in a test form in wrong way. Those issues are discussed in the chapter about the system itself.

All those techniques that are used for this project are relatively new and this is the main reason of the shape of the references. They are mostly links to web resources and those web sites have contributed the most to the project. Frankly speaking, since I have gone through plenty of web pages devoted to XML, servlets and other things I have mentioned before, it is virtually impossible to list all of them.

1.1. XML

XML stands for the eXtensible Markup Language. It is a relatively new markup language, developed by the World Wide Web Consortium (W3C), mainly to overcome limitations in HTML.

HTML is an immensely popular markup language. It is supported by thousands of applications, including browsers, editors, email software, databases, and many more. Originally, the Web was a solution to publish scientific documents. Today it has grown into a full-fledged interactive medium, supporting applications such as online shops, electronic banking, and trading forums. To accommodate this phenomenal popularity, HTML has been extended over the years. Many new tags have been introduced. The first dozen HTML tags eventually have been extended by almost hundred new ones. Furthermore, a large set of supporting technologies also has been introduced, namely: DHTML, JavaScript, Java Applets, CCS, Flash, CGI and more. Some of these technologies were developed by the W3C, whereas others were introduced by vendors.

HTML has ended up becoming a complex language. At almost 100 tags, it is definitively not a small one. The combinations of tags are almost endless and the result of a particular combination of tags might be different from one browser to another. However, despite all these tags already included in HTML, considering electronic commerce applications, even more tags are needed, like tags for product references, prices, name, addresses, and so forth. Streaming needs tags to control the flow of images and sound, search engines need more precise tags for keywords and description, security needs tags for signing.

The list of applications that need new HTML tags is literally endless. However, adding even more tags to an overblown language is hardly a satisfactory solution. It appears that HTML is already on the verge of collapsing under its own weight. Worse, although many applications need more tags, some applications would greatly benefit if there were fewer tags. Machines, which are not powerful as PC, like personal digital assistants, cannot process a complex language like HTML. Another, problem is that, it is often the case, when pages have more markup than content, which make them slow to download and display.

In conclusion, even though HTML is a successful markup language, it has some major shortcomings. XML was developed to address these shortcomings. Moreover, one can say that XML exists because HTML was successful, therefore, XML incorporates many positive features of HTML. XML also exists because HTML could not live up to new demands and breaks new ground where it is appropriate. Anyway, it is difficult to change a successful technology like HTML, so XML has raised some level of controversy.

XML is unlikely to replace HTML in the near future since XML does not threaten the Web but introduces new possibilities. Work is already under way to combine XML and HTML in XHTML, an XML version of HTML. Some of the areas where XML will be useful in the near future include:

- large web sites site maintenance. XML would work behind the scene to simplify the creation of HTML documents,
- exchange of information between organizations,
- databases offloading and reloading,
- electronic commerce applications where different organizations collaborate to serve a customer,
- scientific applications with new markup languages for mathematical and chemical formulas,
- electronic books with new markup languages to express rights and ownership,
- handheld devices and smart phones with new markup languages optimized for these portable devices.

1.1.1. XML applications

XML is not just for web site publishing. This language goes far beyond the scope of it and, in general, its applications are classified as being:

- document applications manipulate information that is primarily intended for human consumption,
- data applications manipulate information that is primarily intended for software consumption.

The difference between the two types of application is a qualitative one. It is the same XML standard, but it serves different goals. This is important because one can reuse tools and experience across a large set of applications. Furthermore, XML integrate in some way, those two exclusive areas. With XML it is possible to create documents which are easily understandable by humans and still useful for automated data processing.

Document applications

The first application of XML is document publishing. Since XML is about structure of the document, it makes the document independent of the delivery medium. That makes possible to edit and maintain documents in XML and use some means to automatically publish them on different kinds of media. The ability to target multiple media is important these days when many publications are available online. Other factor, which makes XML very useful is connected to continuously changing appearance of majority of web sites. When one needs to reformat his site regularly, XML gives a painless way to do that. Finally, some Web sites are optimized for specific viewers, what often leads to development of two or more versions of the same site, which is usually very costly when done manually. For all these reasons, it makes sense to maintain a common version of the documentation in a media-independent format, such as XML, and to automatically convert it into desired format such as HTML, PostScript, PDF, RTF, and so forth.

Data Applications

In the 1986 the Standard Generalized Markup Language (SGML) was published by International Standard Organization as ISO 8879. SGML is based on the early work done by a IBM's employee, Dr. Charles Goldfarb. He was the inventor of the concepts behind SGML.

The two main characteristics of SGML are that its markup:

- describes the document's structure, not the document appearance,
- conforms to a model, which is similar to a database schema, which means that it can be processed by software or stored in a database.

SGML does not standardized structure that every document needs to follow, so it does not define what a title or a paragraph is. In fact, it is unrealistic to assume that a single document structure could satisfy the needs of all authors. The SGML approach is not to impose its own tag set but to propose a language to describe the structure of documents and mark them accordingly. The strength of SGML is that it is a language to describe documents - in many respects similar to programming languages. It is therefore flexible and open to new applications. The document structure is written in a Document Type Definition (DTD). It specifies a set of elements, their relationships, and tags to mark the document.

XML inherits the original goals of SGML, which is one of its predecessors. SGML was created to give document management access to the software tools that had been used to manage data, such as databases. XML goes even further, bringing a publishing kind of distribution to data. This leads to the concept of no difference between documents and applications. Indeed, if the structure of a document can be expressed in XML, so can the structure of a database. XML may then be used to exchange information between parties. One can think of XML web as of a large database on which applications can tap and extract data.

1.1.2. XML basics

XML may be viewed as a standard for exchanging and publishing information in a structured manner. XML is a language used to describe and manipulate structured documents. XML offers the tree structure and, what is important, does not dictate or enforce the specifics of this structure. XML is a flexible mechanism that accommodates the structure of specific applications. It provides a mechanism to encode both the information manipulated by the application and its underlying structure.

The idea behind XML is straightforward. It aims at answering the conflicting demands that arrive at the future of HTML. On one hand, more tags are needed; on the other hand, developers want fewer tags since HTML is already so complex. XML resolves this dilemma, by introducing following two concepts:

- no predefined tags,
- it is strict.

Those concepts are virtually non-existent in HTML.

No Predefined Tags

Since in XML there are no predefined tag, a developer can create the tags that satisfy his/her needs. Those tags can also contain additional attributes. This is the extensibility of XML. It is extensible because it predefines no tags but lets the author create tags that are needed for the application. Although the concept is simple, it opens many questions, mainly:

- how to inform processing software how to interpret tags,
- how to extract information from XML data,
- what kind of software use for processing XML,
- does it simplify the real life task?

Fortunately, there are answer for all those questions. Many standards accompanying XML have been developed and those most important for this project are described in the next section.

Strict Syntax

One of the strongest sides of HTML is that it has a very forgiving syntax. This means that the syntax does not necessarily have to correspond to the specification. This is great for developers who can be as lazy as they want, and who are not obliged to be very careful about how they prepare the code. This great advantage of HTML is also the major drawback when it comes to data extraction from HTML documents. Basically, such extraction is often impossible to be done automatically by computer software. The other issue is that the browsers are growing in size to cope with imperfect HTML code and because of that fact they are becoming generally slower and more processor-power-consuming, which makes them unsuitable for PDAs.

1.1.3. XML companion standards

The real value of XML is that it is a fully standardized markup language. As a matter of fact, XML is more than a markup language. It is a whole range of tools that may be put to work in application environment. In particular, the W3C has developed a number of standards that complement XML and they are referred to as XML companion standards. Those standards give true strength to XML as they let developers to explore all XML's features. The list of those standards is open as new standards are regularly being introduced. The standards that are essential or connected to this project are listed here.

XML Namespace

XML Namespace is often an overlooked standard, although it is a major one considering its importance. Namespace associates elements with their owners. It lets write an XML document that uses two or more sets of XML tags in modular fashion. This enables extensibility because it means that an organization can add new tags to existing elements and clearly label who is responsible for the extension. This prevents name conflicts and is the only way to enable reuse of standard structures.

DTD

The Document Type Definition (DTD) specification is actually part of the XML specification, rather than a separate entity. On the other hand, it is optional since it is possible to write an XML document without it. A DTD specifies the kinds of tags that can be included in a XML document, and the valid arrangements of those tags. It can be used to verify if a XML structure is valid or not. It allows making sure that the XML structure that is being read (being received from the net) is indeed valid. A DTD can exist at the front of the document, as part of the prolog; it can also exist as a separate entity, or it can be split between the document prolog and one or more additional entities. DTD has some major drawbacks. It is difficult to specify a DTD for a complex document in a way that prevents all invalid combinations and allows all the valid ones. However, while the DTD is the first method defined for specifying valid document structure, there are several newer schema specifications and one of them, XML Schema is described next. This project makes use of DTD by the occasion of JAXB.

XML Schema

A DTD makes it possible to validate the structure of relatively simple XML documents. It cannot restrict the content of elements and it cannot specify complex relationships. Finally, DTD uses syntax which is substantially different from XML, so it cannot be processed with a standard XML parser. XML Schema is one of the proposals that have been made to address shortcomings of a DTD. XML Schema is a large and complex standard that has two parts. One part, Schema for Structures, specifies structure relationships and those relationships can be of any kind. It is a great advantage but is occupied by complex implementation. The other part of XML Schema specifies mechanisms for validating the content of XML elements by specifying a data type for each element and, what is important, those data types may be even pretty sophisticated. Lastly, XML Schema uses XML syntax.

CSS and XSL as Style Sheets

Style sheets play an important role for XML. The XML standard specifies how to identify data, not how to display it. HTML, on the other hand, tells how things should be displayed without identifying what they are. The style sheets complement XML specifying how documents should be rendered on given media. In other words, they specify the transformation from XML to screen, paper, and so forth. XML is supported by Cascading Style Sheet (CSS) and Extensible Stylesheet Language (XSL). CSS is widely implemented, since it has been used for some time as a completion to HTML. XSL, on the other hand, is a newer thing and so is more powerful. Considering this project, XSL is an important part of it, whereas CSS is used but only to format HTML display and not directly XML.

XSL includes two parts, the transformation language (XSLT, described in the next caption) and the part that covers *formatting objects*, also known as *flow objects* (XSL-FO). The transformation language provides elements that define rules for how one XML document is transformed into another XML document. The transformed XML document may use the markup and DTD of the original document, or it may use a completely different set of elements. In particular, it may use the elements defined by the second part of XSL, the formatting objects. XSL-FO gives the ability to define multiple areas within a document and then link them together. When a text stream is directed at the collection, it fills the first area and then flows into the following ones when the first area is filled.

XSLT and XPath

The Extensible Stylesheet Language for Transformations (XSLT) standard is essentially a translation mechanism that allows specifying conversion from an XML tag to another tag or instruction used for information displaying. Different XSL formats can then be used to display the same data in different ways, on different media types, for different uses. The XPath standard is an addressing mechanism that is used within XSLT when constructing transformation instructions, in order to specify the parts of the XML structure to be transformed.

XHTML

The XHTML specification is a way of making XML documents that look and act like HTML documents. This is a linking and presentation standard which aims at preserving the benefits of HTML in the XML arena, and to adding additional functionality, as well. Since an XML document can contain any tags, it can have a set of tags that look like HTML ones. That is the main idea of the XHTML specification. The result of this specification is a document that can be displayed in browsers, still being treated as XML data. The data may not be quite as identifiable as pure XML, but it will be a much easier to manipulate than standard HTML, since XML specifies a good deal more regularity and consistency. In the case of this project the XHTML is not used directly. It is rather a product of XSL transformation of XML data.

SAX

Simple API for XML (SAX) can be thought of as the serial access protocol for XML. This is a fast mechanism, that may be used to read and write XML data when efficiency and small memory requirements are the priorities. It is also called an event-driven protocol, because the technique is to register a handler with a SAX parser, after which the parser invokes user defined callback methods whenever it sees a new XML tag, encounters an error, or, maybe wants to tell something else.

DOM

The Document Object Model (DOM) protocol converts an XML document into a collection of objects in computer memory. One can then manipulate the object model in any way that is needed. This mechanism is also known as the random access protocol, because it allows accessing any part of the data at any time. One can then modify the data, remove it, or insert new data. Although it offers much greater flexibility than SAX, the in-memory data structure slows down the application and so is not suitable for applications that require fast XML-data processing.

JAXB

The Document Object Model provides a lot of power for document-oriented processing, but it does not provide a lot of object-oriented simplification. For data-oriented processing purposes, rather than handling fully-fledged articles, it is more convenient to use a brand new technique, Java Architecture for XML Binding (JAXB). It is totally object-oriented standard, which allows automatic two-way mapping between XML documents and Java objects. JAXB plays so important role in this project that the next entire section is devoted exclusively to it.

1.2. JAXB

JAXB is a technique for simplifying creation and maintenance of XML-enabled Java applications. It provides a schema compiler and a runtime framework to support mapping between XML documents and Java objects. The schema compiler translates XML's DTD into one or more Java classes without necessity of writing complex parsing code. Generated classes contain code to perform error and validity checking of incoming and outgoing XML documents, thereby ensuring that only valid, error-free messages are accepted, processed, and generated by a system. Those classes allow manipulating or even creating XML documents without writing any logic to process XML elements. In other words, the generated code provides an abstraction layer that allows access to the XML data without any specific knowledge about the underlying data structure. Moreover, JAXB is a fast mechanism and its speed matches up the SAX's one. Anyway, as in the case of DOM, processing with JAXB is memory-consuming. A next advantage of JAXB is its extensibility. The generated classes can be extended, like ordinary Java objects, to provide additional application specific functionality. Building a JAXB application is a process that involves several steps which are discussed next.

1.2.1. Working with JAXB

JAXB has potentially wide scope of application, especially when considering data sharing. On the whole, since JAXB provides an easy way to work with data in general, it may be appreciated in many situations. Constructing a JAXB based application contains a few distinct stages.

Create or Obtain a DTD

In order to generate XML processing codes, the JAXB compiler requires a DTD, which defines simple constraints on structure and contents of a XML document. It is assumed that future versions of JAXB will add support for other schema languages, mainly for XML Schema.

Define a Binding Schema

The DTD constraints on structure of a XML document are often not precise enough and the results of the schema compiler processing may be not satisfactory for a developer. That is why a Binding Schema is useful. One can say that it compensates lacks of DTD when necessary. A syntax of Binding Schema in some way resembles shape of the XML Schema.

Generate Processing Codes

Generation of the processing codes is done by the Schema Compiler. As the result one gets Java classes that reflect XML document's structure. The classes are then compiled in a traditional way with one exception only. For obvious reasons, this compilation process needs to have access to the JAXB libraries.

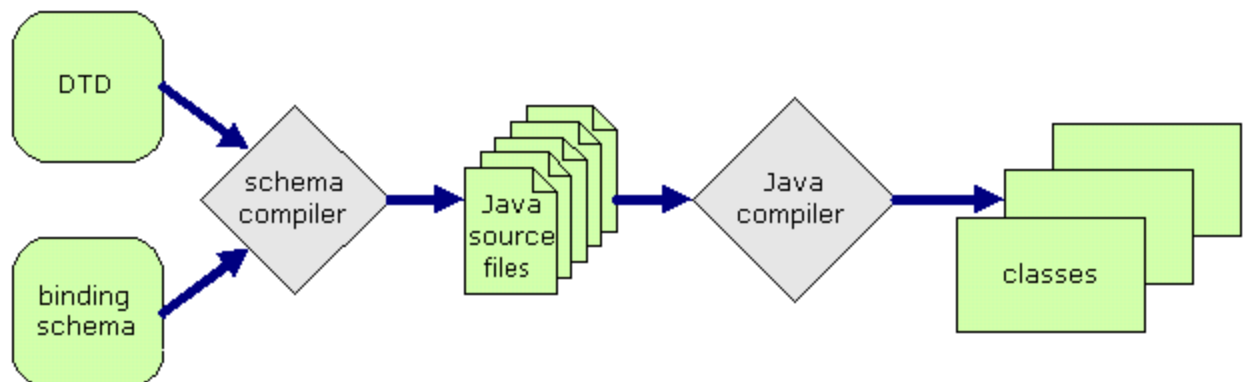


Figure 1. JAXB – generation of classes

Unmarshal from an XML Document

Unmarshaling is a process of populating a JAXB generated class object with a corresponding XML document. To unmarshal a XML document, one has to call the *unmarshal* method in a JAXB generated class object with a parameter that contains contents of a corresponding XML document.

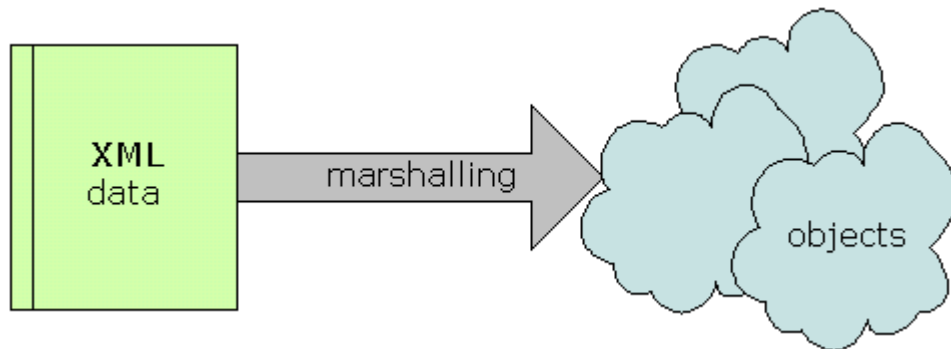


Figure 2. JAXB - unmarshalling

Instantiation

In some cases there is no existing XML document to work with or a new XML document is needed. In such circumstances one simply creates a new instance or instances of generated by Schema Compiler classes.

Working with Data

In addition to the methods of creating new instances of JAXB generated classes, there are also other methods available to work with data. Those methods allow retrieving data and its changing.

Validation

Validation is the process of verifying that the Java object representation of a XML document conforms with the DTD. Validation is required prior to marshalling, if content has been added or modified within the Java object representation. The content validation is a simple method call.

Marshal to a XML document

Marshaling is the process of producing an XML document from Java objects. Marshaling the objects to a XML document follows a similar process to unmarshaling, but this time the *marshal* method takes a parameter that reflects where the constructed XML representation is to be put.

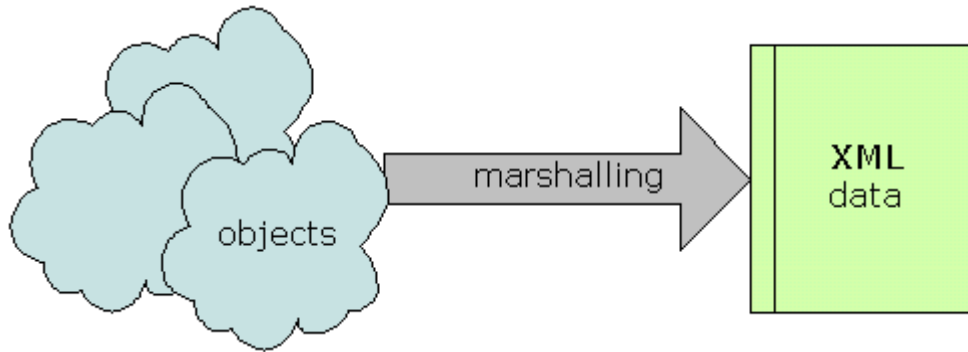


Figure 3. JAXB - marshalling

1.2.2. JAXB Summary

JAXB provides a convenient way of dealing with processing of XML. The most important reason to use JAXB is that JAXB applications are written in the Java programming language and can process XML data. Essentially, JAXB provides a bridge between these two complementary technologies.

JAXB includes a compiler that maps a schema to a set of Java classes. Once having classes, it is possible to build Java object representations of the XML data that follow the rules that the schema defines. Just as an XML document is an instance of a schema, a Java object is an instance of a class. Thus, JAXB allows creating of Java objects at the same conceptual level as the XML data. Representing data in this way allows manipulating it in the same manner as manipulating Java objects, making it easier to create applications processing XML data. Having data in the form of Java objects it is easy to access and process it. In addition, after working with the data, one can write the Java objects to a new XML document. With the easy access to XML data that JAXB provides, the only concern that a developer faces is only using the data, rather than spending time writing code to access it.

1.3. Servlets

Servlets are Java modules that run inside request/response-oriented servers and extend them providing a powerful mechanism for developing server side application. The traditional way of adding functionality to a web server is the Common Gateway Interface (CGI), a language-independent interface that allows a server to start an external process which gets information about a request through environment variables, the command line and its standard input stream and writes response data to its standard output stream. Each request is answered in a separate process by a separate instance of the CGI program or the CGI script. Although CGI played a major role in the explosion of the Internet, its performance, scalability and reusability issues make it less than optimal solutions. Servlets are an effective substitute for CGI scripts as they provide a way to generate dynamic documents that is both easier to write and faster to run. They also address the problem of doing server-side programming with platform-specific APIs since they are developed with the Java Servlet API, which is a standard Java extension.

Servlets have several advantages over CGI. Firstly, a servlet does not run in a separate process. This removes the overhead of creating a new process for each request. Secondly, a servlet stays in memory between requests, whereas a CGI program needs to be loaded and started for each CGI request. Moreover, there is only a single instance of a servlet which answers all requests concurrently. This saves memory and allows easy managing of persistent data. Finally, a servlet can be run by a servlet Engine in a restrictive sandbox which allows secure use of untrusted and potentially harmful servlets.

Servlets are not tied to a specific client-server protocol but they are most commonly used with HTTP. They make use of the Java standard extension classes in the packages *javax.servlet* (the basic Servlet framework) and *javax.servlet.http* (extensions of the servlet framework for servlets that answer HTTP requests). Since servlets are written in the portable Java language and follow a standard framework, they provide a means to create sophisticated server extensions in a server and an operating system independent way.

1.3.1. Applications of servlets

The scope for servlets' uses is really broad. Since a servlet is essentially a Java application, it is capable of doing things that typical Java application can do except for one thing - there is no graphical interface for the servlet.

Middle-Tier Processing

In many systems a middle tier serves as a link between clients and back-end services. Using a middle tier is a way of off-loading both clients and servers. Another advantage of middle tier processing is simply connection management. A set of servlets could handle connections with hundreds of clients, if not thousands, while recycling a pool of expensive connections to database servers. Other middle tier roles include tasks like mapping clients to a redundant servers and supporting different types of clients such as pure HTML and Java enabled clients.

Proxy Servers

Servlets can act as proxies for applets. This can be important since Java security mechanisms allow applets only to make connections back to the server from which they were loaded. If an applet needs a connection to a server located on a different machine, a servlet can make this connection on behalf of the applet.

Protocol Support

The Servlet API provides a tight link between a server and servlets. This allows servlets adding new protocol support to a server. Essentially, any protocol that follows a request/response model can be implemented by a servlet. This could include SMTP, POP, FTP.

Processing data posted over HTTP

This includes processing data from HTML forms. A servlet can be part of an order-entry and processing system, working with product and inventory databases, and perhaps an on-line payment system.

Providing dynamic content

A servlet may return as the result of some processing a HTML page or another data stream.

Managing state information on top of the stateless HTTP

A servlet can keep state information when client needs it. Considering an online shopping system, which manages shopping carts for many concurrent customers, a servlet may map every request to the right customer.

Forwarding requests

Servlets can forward requests to other servers and servlets. Thus, servlets can be used to balance load among servers that mirror the same content, and to partition a single logical service over several servers.

Allowing collaboration between people

A servlet can handle multiple requests concurrently, and can synchronize those requests. This allows servlets to support systems such as on-line conferencing.

Being a community of active agents

A servlet give a way for distributed processing. It is possible to define active agents that share work among themselves. Each agent is a servlet, and the agents could pass data between each other.

1.3.2. Overview of servlets

The central abstraction in the Servlet API is the *Servlet* interface. All servlets implement this interface, either directly or by extending a class that implements it such as *HttpServlet*. This interface provide methods that manage the servlet and its communication with clients.

A servlet is initialized by a server's servlet engine which instantiates a servlet and perhaps other classes which are referenced by the servlet. Then it calls the servlet's *init* method. The servlet performs setup procedures within this method and store the *ServletConfig* object. This object can be accessed later by calling the *getServletConfig* method and it contains parameters of the servlet and a reference to the *ServletContext*. The *init* method is called only once during the servlet's lifecycle. It does not need to be thread-safe since other servlet's methods have to wait until this method has finished its job.

After a loading and an initialization the servlet is able to handle client requests. The *service* method is called for every request to the Servlet. The method is called concurrently so it should be implemented in a thread-safe manner. If that is not possible a servlet may implement *SingleThreadModel* which makes the *service* method not be called concurrently. The *service* method receives the *ServletRequest* and *ServletResponse* objects. The *ServletRequest* class encapsulates the communication from the client to the server. It allows the servlet accessing information such as the names of the parameters passed in by the client, the protocol being used by the client, and the names of the remote host that made the request and the server that received it. It also provides the servlet with access to the input stream, through which the servlet may get data from the client. *HttpServletRequest* is a subclass *ServletRequest* and contains methods for accessing HTTP-specific header information. The *ServletResponse* class encapsulates the communication from the servlet back to the client. It gives the servlet methods for replying to the client. It allows the servlet to set the content length and mime type of the reply, and provides an output stream and a writer through which the servlet can send the reply data. Subclass of the *ServletResponse* class, *HttpServletResponse* contains methods that allow the servlet to manipulate HTTP-specific header information.

The *destroy* method is called when a servlet is to be unloaded. It usually happens after all service calls have been completed, or when a server-specific number of seconds have passed, whichever comes first. If the servlet handles any long-running operations, the *service* methods might still be running when the server calls the *destroy* method, so this method has to be thread-safe. All resources which were allocated in *init* should be released in *destroy* one. This method is guaranteed to be called only once during the servlet's lifecycle.

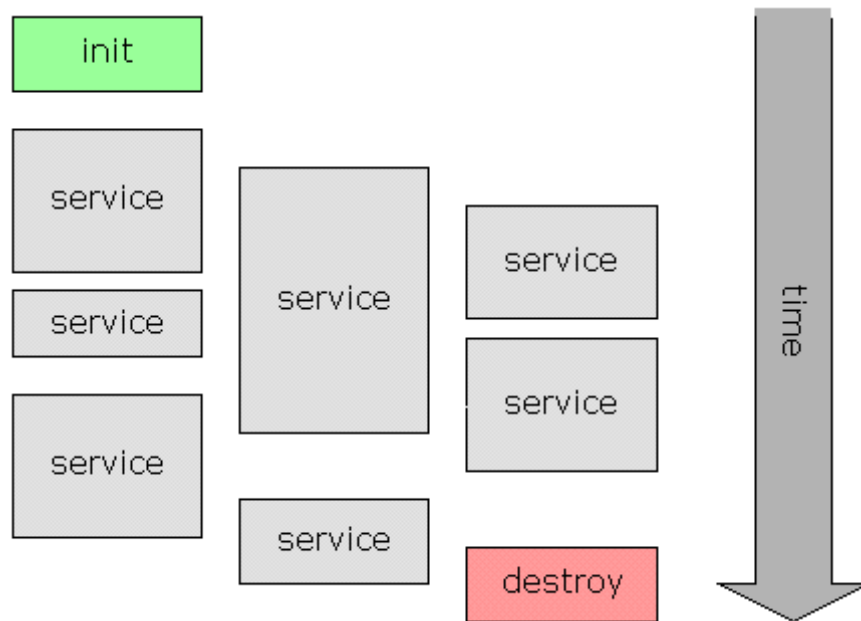


Figure 4. Lifecycle of the servlet

1.3.3. HTTP Support

Most of servlets use the HTTP protocol for communication purposes. Support for handling the HTTP protocol is provided in a separate package, *javax.servlet.http*. HTTP stands for the Hyper Text Transfer Protocol. It defines a protocol used by web browsers and servers to communicate with each other. This protocol defines a set of text-based request messages called *HTTP methods*. They include GET, HEAD, POST, PUT, DELETE, TRACE, CONNECT and OPTIONS. The first three are the most important and they are discussed.

The GET Method

The HTTP GET method requests information from a web server. This information could be a file, output from a device on the server, or output from a program, in particular a servlet. Parameters that may be passed to the servlet by this method are a part of the URL. Anyway, the GET request has an important limitation since most web servers limit how much data can be passed as part of the URL. If more than a few hundred bytes have to be sent to the server, the HTTP POST method should be used instead. It is important that the handling of a GET method is expected to be safe and idempotent. This means that a GET method will not cause any side effects, such as changing some data on the server side, and that it can be executed repeatedly without penalty. Finally, a server sends a HTTP response message back.

The HEAD Method

The HEAD method is very similar to the HTTP GET method. Its request looks the same but the server only returns the header information. This method is used to check parameters of a document prior to downloading. Those parameters include last-modified date, size and type of the document, and information about the server type. The HEAD method is expected to be safe and idempotent.

The POST Method

The POST request allows sending data to the server. It is especially useful when there is a need to send more information than the GET request allows. Since the POST method passes all of its parameter data in an input stream, not as a part of URL, there is no special limit on a data size. The POST method is not expected to be safe nor idempotent, so that it can perform modification of some persistent data, and it is not required to be repeatable.

The *HttpServlet* class dispatches a request to different Java methods for different HTTP request methods. These Java methods include *doGet*, *doHead*, *doDelete*, *doOptions*, *doPost* and *doTrace*. The class also detects which methods are overridden in a subclass and can report back to a client on the capabilities of the server

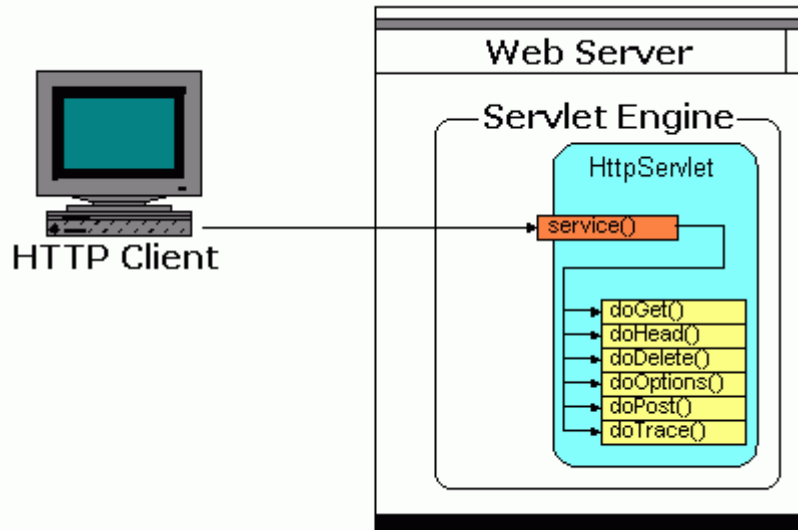


Figure 5. Dispatching of HTTP request

The *doOptions* and *doTrace* methods have suitable default implementations and are usually not overridden. The HEAD method is by default executed by calling *doGet* and ignoring any output that is written by this method. Default implementations of *doGet*, *doPut*, *doPost* and *doDelete* return a *Bad Request* HTTP error. A subclass of *HttpServlet* may override one or more of these methods to provide the desired implementation. Usually *doGet* and *doPost* are overridden.

The *doGet* method is usually supposed to read input parameters, set response headers and then write the response data. The *doPost* method should be overridden when one needs to process an HTML form posting or to handle a large amount of data being sent by a client. It should also be applied, when there is necessity for processing that has side effects.

The HTTP processing methods are passed two parameters, the *HttpServletRequest* object and the *HttpServletResponse* object. The *HttpServletRequest* class has several convenience methods to help parse the request. Anyway, the request can still be parsed separately by simply reading the text of the request.

1.3.4. Saving client state

The Servlet API provides two ways to track client state across connections of the stateless HTTP. In order to do that it uses cookies and session tracking.

Cookies

A cookie is a named piece of data maintained by a browser, typically for session management and/or for storing a small amount of any information associated with the user. Since HTTP connections are stateless, one can use a cookie to store persistent information across multiple HTTP connections but not across multiple browser sessions.

To save cookie information one needs to create an instance of the *Cookie* object, set the response content type to the *HttpServletResponse* response, add the cookie to the response, and then send the output. All cookie data are strings. There is a necessity of conversion information of different type to a String object. By default, the cookie lives for the life of the browser session. It is possible to enable a cookie to live longer or to delete the cookie.

It is impossible to request for a specific cookie. One must ask for all cookies and find the specific one. Since it is possible that multiple cookies could have the same name, just finding the first setting is not always sufficient and there might be a need for checking all cookies anyway to avoid ambiguity.

Session Tracking

Session tracking is a mechanism that servlets use to maintain state about a session, which is a continuous connection originating from the same browser over a fixed period of time. The tracking is normally done through the implicit use of browser cookies, which simply store the session ID. In the case when cookies are disabled, the session ID has to be encoded in the URL string. This method, called URL rewriting, is a less elegant solution because to maintain the session all HTML pages, which are sent, have to be created dynamically.

The Session object is a part of *HttpSession* class and can be obtained by calling *getSession* method. Once having access to an *HttpSession*, one can maintain within it a collection of key-value-paired information. It is possible to store any kind of Java objects inside the session and this is the immensely useful advantage of the session mechanism.

1.3.5. Servlet Summary

Java servlets are server-side Java programs that can generate content or do some processing in response to a client request in much the same way as CGI programs do. Servlets can be thought of as applets that run on the server side without a user interface. They are invoked through an URL invocation.

Servlets are a powerful mechanism for a Java programmer to gain access an object-oriented abstraction of HTTP. Servlets are portable across web servers, are simple to design and implement, have tight integration with the web server, service requests in an efficient manner and may be run inside safe sandbox.

Although HTTP is a stateless protocol, *HttpServlet* provide the means to address the web state problem. Having implemented a session tracking mechanism, servlets are almost perfect to implement applications, which require user identification, like e-commerce applications.

1.4. Relational Database and SQL

SQL stands for Structured Query Language. SQL has its origins in the relational model of data created in the seventies by E. F. Codd of IBM. Over time other companies were involved in relational database research and development, and eventually Oracle became the first generally available relational database. Today there are dozens of relational database products, like the MySQL database system, which is used for the purposes of this project. SQL has been standardized and it has become the standard for data retrieval from relational databases. There is no alternative for the SQL is today in a databases world.

The basic component of a relational database is the table. The table is comprised of rows and rows are made up of columns. Finally, columns contain data. Columns have to contain scalar or atomic data values, that is data with one value only. They may store data of different type, but one column's data has to be of the same kind.

A table inside a database can contain entries called foreign keys, which are links to other tables. This provides great flexibility and is a major advantage of relational databases.

SQL itself is a language that allows talking to a database by using standardized queries. Those queries may ask for data or make changes inside the database. They can create new tables, delete existing ones or put some data into rows. Basically, SQL is a language of databases.

A database plays an important role for this project, but since the SQL queries used are basic ones and the database structure is simple, details will be discussed later.

Chapter 2

2. The system

Every application is about data and its processing. The system being described here is no exception to the rule. It is composed of data structures and algorithms responsible for their processing. Anyway, it is difficult to talk only about one regardless of the other, but it is very convenient to do so, considering description clarity.

The chapter is divided into two parts. Firstly, the basics of the system are described. Then, the second section deals with actual implementation. Its first subsection is mostly about data and touches up the XML's part of the project. In contrast, the sixth part, about servlets¹, is devoted mostly to algorithms. The middle subsections tackle XML companion standards applied in this project (XSL, XML Schema, DTD) and JAXB. The last subsection discusses some other techniques that are used within the application.

¹ The code of the servlets is composed of about 3000 lines and because of its complexity, this report deals only with its most essential fragments. It would be virtually impossible to describe the code line by line, and if it was done, the report would become unnecessarily large. Even though the report only tackles the most vital system details, it has ended up being less than compact. Furthermore, the report may appear to be superficial, but in fact, the author of it just had to skip many issues for the sake of clarity. For the same reason, the report is about the description of the system only, omitting the developing process itself. Since the software development process is awkward to describe, especially, when considering more complicated application, it is more reasonable to concentrate on the results than put too much effort on describing dilemmas of writing the code.

2.1. The system's description

2.1.1. The question format

Prior to constructing the actual application, several issues had to be addressed. One of them was the question format. The HTML language imposes some constraints on the format of allowed form types, which can be displayed by a web browser. Eventually, as the result of trade-off between the available HTML language's elements and possible useful types of questions, the system implements four kinds of questions.

Single choice question

A question with only one correct answer. It may include "don't know" answer when needed.

Multiple choice question

A question with one or more correct answers within it. It may include "don't know" answer when needed.

String match question

A question that expects a single word as its answer. A user may decide not to answer by leaving the enter field blank.

Question for external evaluation

A question where user enters a few sentences as his/her answer. Such answer is not evaluated automatically but is just kept in the database and is expected to be read and evaluated by the tutor himself/herself.

2.1.2. The system's basics

The system is composed of three major parts. The first one is responsible for an authentication of system's users; the next one provides system's configuration centre and the third part displays tests to the system's users, checks validity of users' answers, displays results for an anonymous user and sends users' answers to the database. For testing purposes, a simple HTML welcome page was created to provide access to all parts of the system.

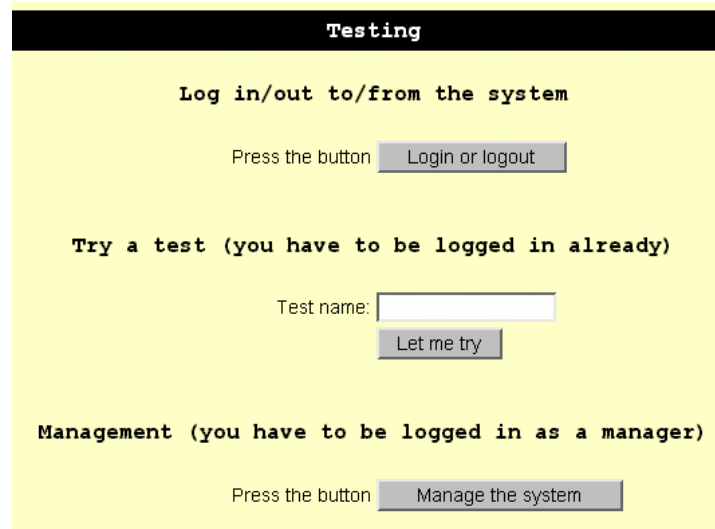


Figure 6. The welcome page

To handle situations when user tries to get access to unavailable for him/her resources and to cope with unauthorised access to the system in general, another, a very simple page, called the forbidden page, was prepared.

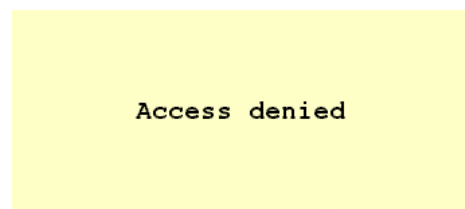


Figure 7. The forbidden page

The login subsystem

The core part of this subsystem is the *Login* servlet. It talks to the database and creates the session object when necessary. The session object contains the user's name and the user's group name.

A user is redirected to the login subsystem when it has not got assigned the session object. The *Login* servlet may redirect the user back to the welcome page if the login attempt is successful or to the forbidden page, if he/her has not granted access to the system.

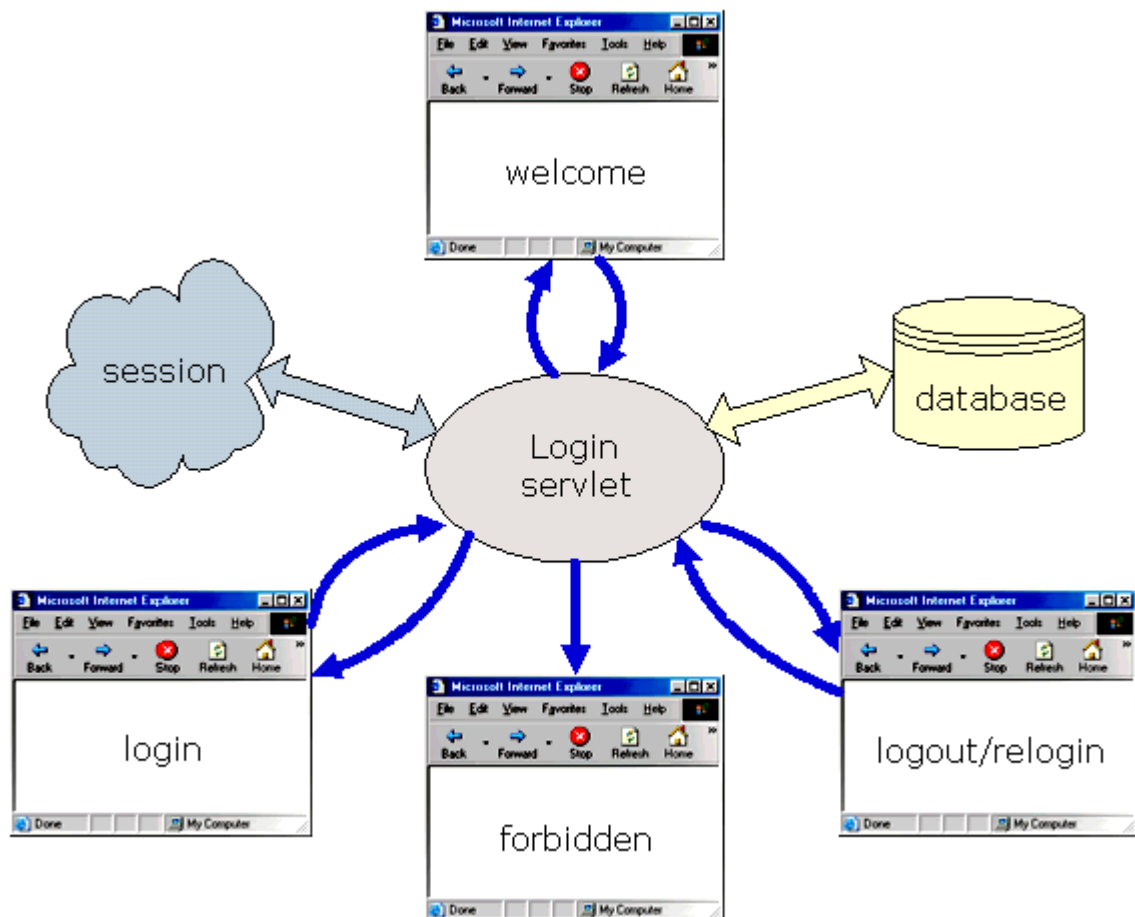


Figure 8. The login subsystem

When user try to login to the system for the first time, he/she is displayed the login page. A database registered user fills in the proper fields with his/her login and password and then submits them. An anonymous user can also login typing *anonymous* as the login and the password.

Login to the system

User name:

Password:

Figure 9. The login page

The login subsystem can be called when the user has been associated the session object already. In this case he/she is displayed the logout/relogin page.

You are already logged in, student

Log out of the system

Just press the button

Login as a different user

User name:

Password:

Figure 10. The logout/relogin page

The manager subsystem

The manager subsystem is the centre of the application. Only registered user of the “manager” user’s group can access the manager. Calling the *Manager* servlet without any parameters makes it displaying the manager menu.

Welcome to the manager page, manager!

Choose an option

Create a new test

Edit existing tests

Delete a test

Deploy a test

Undeploy a test

Check results

Leave the manager

Figure 11. The manager menu

An unauthorised user is redirected to the forbidden page when he/she tries to call the manager. A user who is not logged in the system, is redirected to the login subsystem.

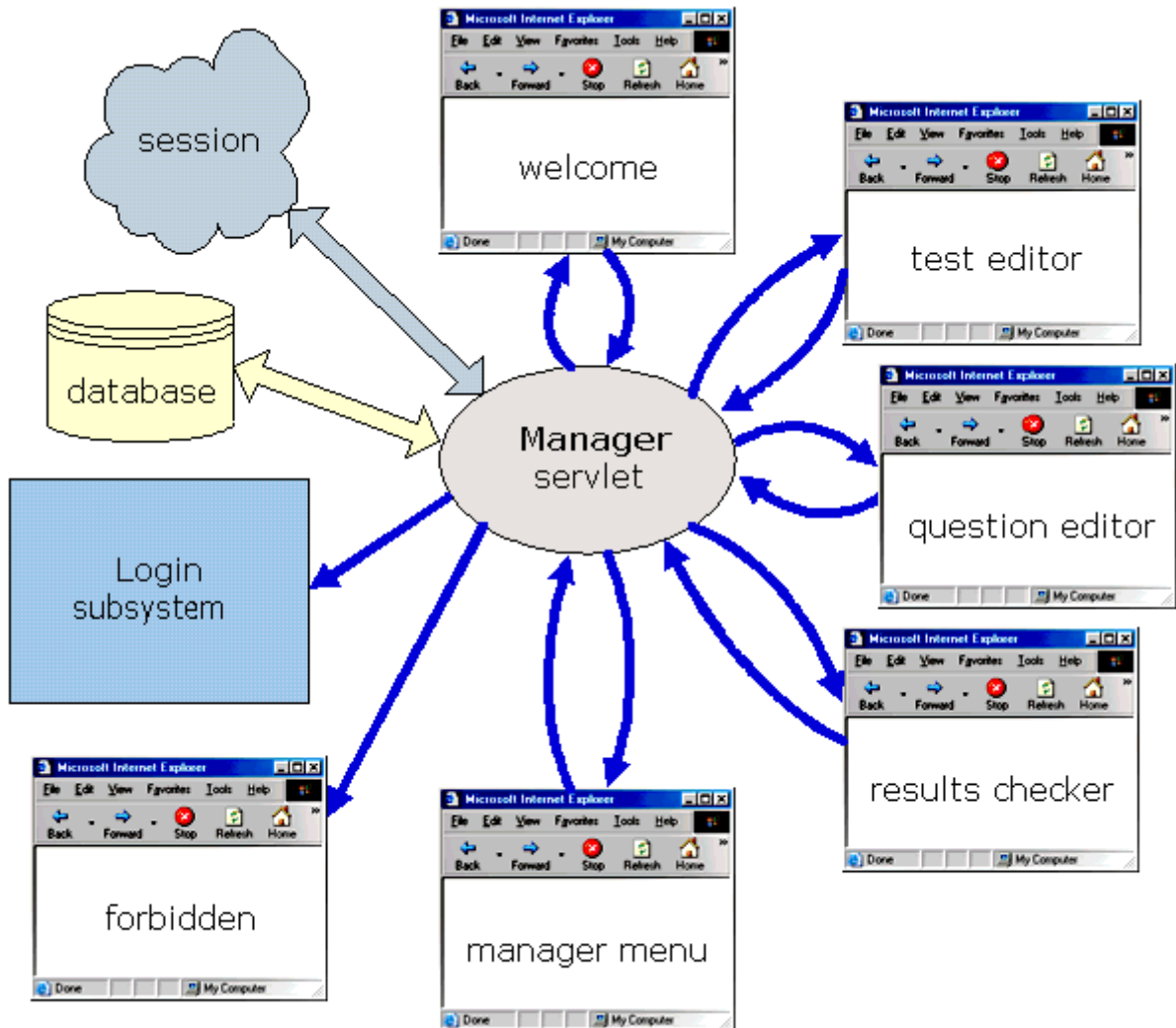


Figure 12. The manager subsystem

The authorised user is able to do several things. First of all, he/she can create a new or edit an existing test. The tests available for editing are only those, which have not been deployed yet. In the case of attempt to edit a deployed test, the user is redirected to the forbidden page. The test editor page allows changing the author, date, title, description, notice and warning of the test. It also allows creation of new questions or editing the existing ones. There are three kinds of the question editor screens depending on the question type.

Test Editor

Title:

Auhor:

Date:

Description:

Notice:

Warning:

Questions:

- 1. A botanical challenge
- 2. A difficult question
- 3. A difficult task
- 4. An easy mission here

Add a new question:

Figure 13. The test editor

Question Editor

Title:

Description:

Notice:

Warning:

Answer: Suggestion:

Figure 2.9. The external evaluation question editor

Question Editor

Title:

Value:

Description:

Notice:

Warning:

Answer: Expected: Suggestion:

Figure 14. The string match question editor

Question Editor

Title:

Value:

Description:

Notice:

Warning:

Answers:

1. True | False | Don't know
 Checked

2. True | False | Don't know
 Checked

Add a new answer:

Figure 15. The single and multiple choice question editor

Apart from creation and editing of tests, there are other options available in the manager menu. A non-deployed test may be deleted – physically removed from the file system. The test which is not deployed, may be also made available for use; in other words, it may be deployed. Deployment involves creation of a database table named after the name of the XML file containing the test. A test may be also un-deployed and in that case the table with its results is dropped.

The most important part of the system is, by far, the result checker. It gives access to system’s users’ results. Basically, it reads users’ answers from the database, evaluates them, calculates statistics and then displays everything in the web browser window. If the test contains questions for external evaluation and if such questions have been answered, a buttons appears in the question’s column; clicking such button opens a window with the answer.

Participant's name	Question no.1	Question no.2	Question no.3	Question no.4	Correct answers	Wrong answers	No answers	Overall result	With penalty
anonymous	1	1,4			2	0	1	2.0	2.0
anonymous	2	5	larch		1	1	1	2.0	1.0
anonymous	1	2,4	oak	Check it	1	2	0	1.0	-1.0666666
anonymous	2	2,4			0	2	1	0.0	-1.0666667
anonymous	1	2,4			1	1	1	1.0	0.93333334
student	1	1,4	larch	Check it	3	0	0	4.0	4.0
Statistics	67% 33% 0%	33% 50% 17%	33% 17% 50%		44%	33%	22%	1.6666666	0.9666667

Go back

Figure 16. The results checker

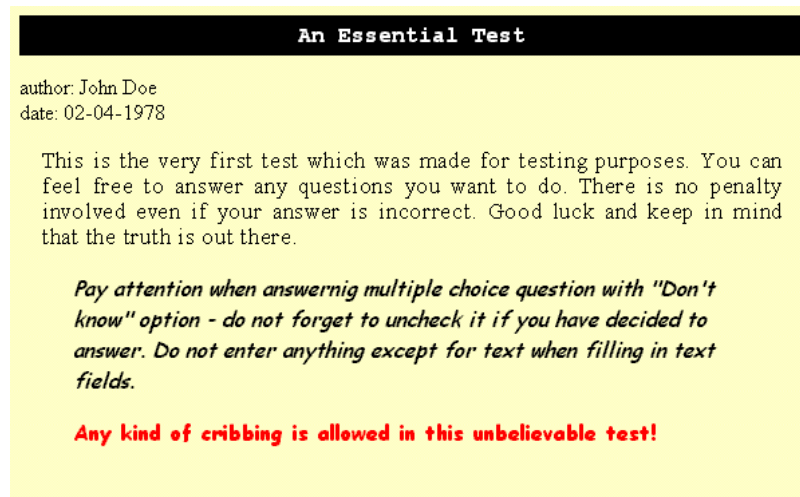
Participant's name	Question no.1	Question no.2	Question no.3	Question no.4	Correct answers	Wrong answers
anonymous	1	1,4			2	0
anonymous	2	5	larch		1	1
anonymous	1	2,4	oak		1	2
anonymous	2	2,4			0	2
anonymous	1	2,4			1	1
student	1	1,4	larch		3	0
Statistics	67					33%

student - Netscape 6

only right answers

Figure 17. The result checker – an external evaluation type question

The testing subsystem is composed of three servlets. The *Presenter* is responsible for XML processing including unmarshalling XML document into Java objects. These objects are then put into the session object and used by the other servlets in the later processing. The transformation from XML to HTML is based on the XSL style sheet. As the result of this transformation the system's user is presented the HTML page built up of the test's introductory part and a sequence of forms followed by the submit and reset buttons.



An Essential Test

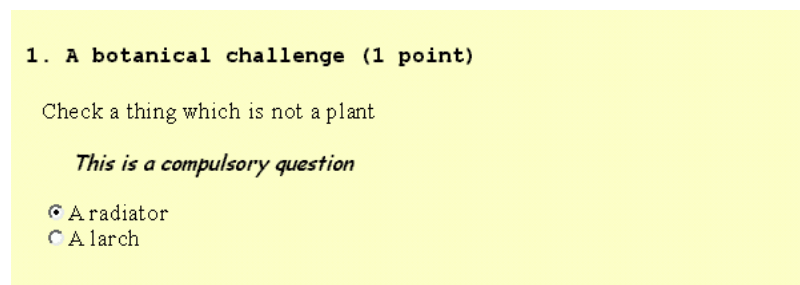
author: John Doe
date: 02-04-1978

This is the very first test which was made for testing purposes. You can feel free to answer any questions you want to do. There is no penalty involved even if your answer is incorrect. Good luck and keep in mind that the truth is out there.

Pay attention when answering multiple choice question with "Don't know" option - do not forget to uncheck it if you have decided to answer. Do not enter anything except for text when filling in text fields.

Any kind of cribbing is allowed in this unbelievable test!

Figure 19. The sample of generated test's introductory part



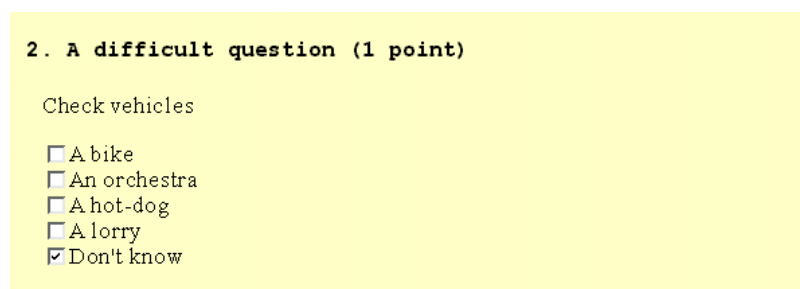
1. A botanical challenge (1 point)

Check a thing which is not a plant

This is a compulsory question

A radiator
 A larch

Figure 20. The sample of generated single choice question form (with the notice)



2. A difficult question (1 point)

Check vehicles

A bike
 An orchestra
 A hot-dog
 A lorry
 Don't know

Figure 21. The sample of generated multiple choice question form

3. A difficult task (2.0 points)

Type a word which is abnormal

No need to hurry!

Figure 22. The sample of generated string match question form (with the notice and the preset suggestion)

4. An easy mission here (2.0 points)

Just type a few words

Please keep your answer short but clear

Figure 23. The sample of generated external evaluation question form (with the notice)

Submit Test Reset Form

Figure 24. The submit and reset buttons

The *Checker* servlets process the response generated by the HTML test form filled in by a user. It marks user's answers inside the test object, which is kept in the session, so that the answers are available to the next servlet in the processing chain. The *Checker* servlet also looks after validity of the user's answers. If the given answer is valid the *Acceptor* servlet is invoked. In the case of the invalid answer, the question form is redisplayed to the user with the main test's warning and warnings of proper questions being set up. The answer is considered to be invalid when:

- no answers have been checked in single or multiple choice question,
- “don't know” answer was checked along with other answers,
- illegal characters have been found within filled in text fields (those illegal characters are ['] and ["] – they may potentially disturb communication with the database) .

The mentioned illegal characters are converted to [#], when the test form is redisplayed.

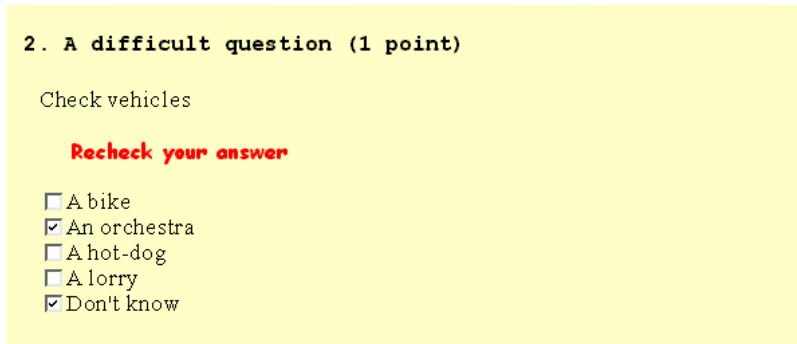


Figure 25. Redisplayed sample multiple choice question

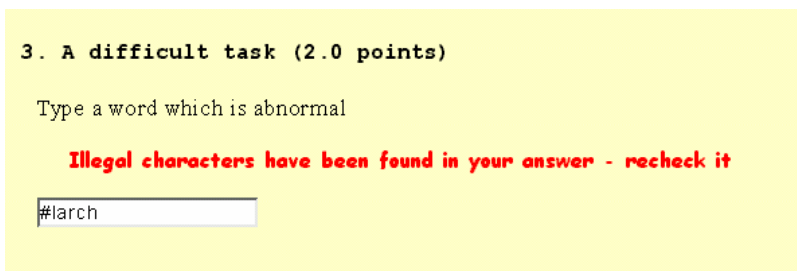


Figure 26. Redisplayed sample string match question

The *Acceptor* servlet is the last one in the chain. Its first task is to save user’s answers in the database’s table. Then depending on the user’s group, it shows the confirmation page or the results page for the anonymous type of user.

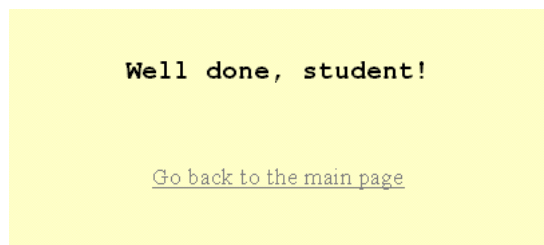


Figure 27. The confirmation page

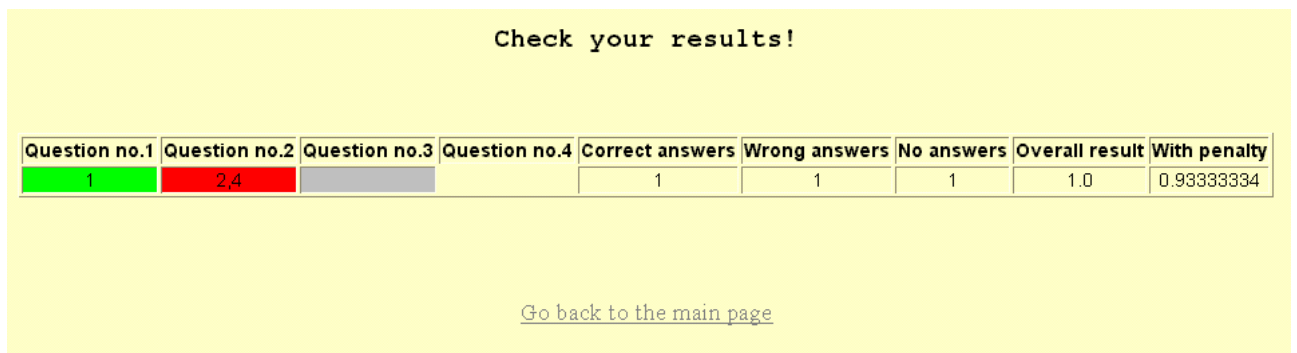


Figure 28. The results page

2.2. The implementation

2.2.1. The XML test's structure

The chosen format for tests' storage is XML. The main reason behind this decision is that the XML is both very convenient for data storage and its processing (XML companion standards provide the means of doing it).

XML structure is defined by tags, as in the case of HTML. Tags can also contain attributes, which are additional information included as part of the tag itself, within the tag's angle brackets.

The vital issue, which one encounters when designing an XML structure, is whether to model a given data item as an element or as an attribute of an existing element. After many hours of working with the XML test data and as the result of decided question types and peculiarities of following JAXB processing, the final structure for test storage seems to be the one presented below.

```
<test name="value">
  <author>value</author>
  <date>value</date>
  <title>value</title>
  <description>value</description>
  <notice>value</notice>
  <warning>value</warning>
  <question type="value" value="value">
    <title>value</title>
    <description>value</description>
    <notice>value</notice>
    <warning>value</warning>
    <answer contents="value" display="value">value</answer>
    ...
  </question>
  ...
</test>
```

The figure on the next page presents graphical representation of the structure excluding the attributes.

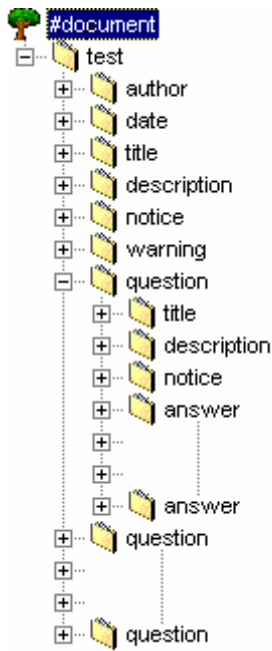


Figure 29. The test structure

Considering the test structure, a few things need to be explained. First of all the elements themselves. Although their structure is well shown by the above figure, there some other issues left and those are discussed by the table below

tag's name	subelement of	mandatory	multiple instances	type	description of element's content
test	-	yes	no	text	-
date	test	yes	no	text	date of test's creation /modification
title	test	yes	no	text	test's title
description	test	no	no	text	test's description
notice	test	no	no	text	test's notice
warning	test	no	no	text	test's warning
question	test	yes	allowed	-	-
title	question	yes	no	text	question's title
description	question	no	no	text	question's description
notice	question	no	no	text	question's notice
answer	question	yes	allowed	text	for single /multiple choice question contains what is displayed as a possible answer

Some of the elements contain attributes. They may be optional or obligatory and they might have default values. The peculiarities of the attributes are exposed in the next table.

tag's name	attribute	of	type	obligatory	default value	description
test	name	-	text	no	-	keeps test's name when test is processed
question	type	-	enumeration (singleChoice multipleChoice stringMatch externalEvaluation)	yes	-	indicates type of the question
	value	-	float	no	1.0	how much is the question worth
answer	contents	single/multiple choice question	text (true false dontKnow)	no	false	indicates if answer is correct, wrong or of "don't know" type
		string match question	text	yes	-	contains the right answer
		external evaluation question	-	-	-	-
	display	single/multiple choice question	text (checked)	no	-	if is set makes the answer to be checked when the HTML form is displayed
		string match question	text	no	-	may contain suggestion
		external evaluation question	text	no	-	may contain suggestion

2.2.2. DTD and XML Schema

A XML structure may be accompanied by DTD or XML Schema to impose some constraints on it. Although this project's application does not involve directly none of these two standards, the DTD (see Appendix A.1.2) is needed for the JAXB part of the project. Nevertheless XML Schema (see Appendix A.1.1) is not necessary at all for the time being, it has been worked out, since it is very likely that future versions of JAXB will use it.

Referencing

Just for integrity of this report, the following line shows how DTD can be bound to the sample XML document; this line is supposed to be placed within the prolog.

```
<!-- <!DOCTYPE test SYSTEM "http://[...]/test.dtd"> -->
```

The XML Schema declaration is placed as an attribute of the *test* tag.

```
<test xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="http://[...]/test.xsd">
```

DTD versus XML Schema

A comparison between the two standards on the base of fragments that describe the same part of XML structure, demonstrates some important differences between them. The DTD for *question* element (including its attributes) is shown next.

```
<!ELEMENT question (title, description?, notice?, warning?, answer+) >
<!ATTLIST question
      type (singleChoice | multipleChoice | stringMatch |
           externalEvaluation) #REQUIRED
      value CDATA "1.0" >
```

The corresponding XML Schema description is more complicated but is more precise by the way; writing it is much more time-consuming.

```
<xsd:complexType name="questionType">
  <xsd:sequence>
    <xsd:element name="title" type="titleType"/>
    <xsd:element name="description" type="descriptionType" minOccurs="0"/>
    <xsd:element name="notice" type="noticeType" minOccurs="0"/>
    <xsd:element name="warning" type="warningType" minOccurs="0"/>
    <xsd:element name="answer" type="answerType" maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="type" type="typeAttribute" use="required"/>
  <xsd:attribute name="value" type="valueAttribute" use="optional"
    default="1.0"/>
</xsd:complexType>
```

```
<xsd:simpleType name="typeAttribute">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="singleChoice"/>
    <xsd:enumeration value="multipleChoice"/>
    <xsd:enumeration value="stringMatch"/>
    <xsd:enumeration value="externalEvaluation"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="valueAttribute">
  <xsd:restriction base="xsd:decimal"/>
</xsd:simpleType>
```

First of all, XML Schema allows precise definition of type of elements' content and their attributes. While the DTD tells only that *value* attribute is character data, the schema confines its values to the *decimal* type only. Secondly, XML schema provides exact control over how many occurrences of an element are allowed.

2.2.3. JAXB

One might say that the project is JAXB based. Basically, JAXB allows mapping between XML documents and Java objects. A part of JAXB is the schema compiler which translates XML document structure (DTD precisely) into one or more Java classes.

The Binding Schema

Since DTD's constraints on structure of a XML document are often ambiguous, there is a need for the binding schema. The schema devised for the test's structure (see Appendix A.1.3) is a simple one. It completes the DTD by specifying that the *type* attribute of the *question* element is an enumeration containing four elements.

```
<element name="question" type="class" >
  <attribute name="type" convert="questionType" />
  ...
</element>
<enumeration name="questionType" members="singleChoice multipleChoice
  stringMatch externalEvaluation"/>
```

Thanks to this declaration, as the result of generation of the processing codes, another, *questionType*, Java class is created; some changes are also made within generated *Question* class. The project's binding schema also tells the schema compiler that the *value* attribute is to be converted to the *float* type.

```
<element name="question" type="class" >
  ...
  <attribute name="value" convert="float"/>
</element>
```

Generate Processing Codes

It is done by simple invocation of the schema compiler.

```
C:\Develop>java com.sun.tools.xjc.Main test.dtd test.xjs
.\Answer.java
.\Question.java
.\Test.java
.\questionType.java
```

Considering the *Question* class for example, it has got generated internal variables that reflect elements and attributes of the *question* structure.

```
private questionType _Type;
private float _Value;
private boolean isDefaulted_Value = true;
private final static float DEFAULT_VALUE = Float.parseFloat("1.0");
private String _Title;
private String _Description;
private String _Notice;
private String _Warning;
private List _Answer = PredicatedLists.createInvalidating(this,
    new AnswerPredicate(), new ArrayList());
```

Elements and attributes are usually converted into *String* objects. In this case, there are a few exceptions. The *value* attribute has been transformed into the *float* type as the result of the declaration within the binding schema. Moreover, it has assigned default value taken from the DTD. Since the *answer* element may appear more than once within the question, it is represented as a list of objects of the *Answer* class, which is also created during compilation.

Generated Java classes need to be compiled by the standard Java compiler in order to obtain actual Java classes, which can be used then.

Working with generated classes

Once having the classes one can do several things with them. XML document may be decomposed into objects and this process is called unmarshalling. When XML file needs to be unmarshalled, the *unmarshalFile* method is invoked.

```
private Test unmarshalFile (FileInputStream data)
    throws UnmarshalException
{
    Test test = new Test();
    test = test.unmarshal(data);
    return test;
}
```

Sometimes there is a necessity for creation of a new element. In such circumstances a new instance of needed class is created, by just calling its constructor.

Having document as Java objects one can manipulate its elements and their attributes by simply calling proper methods, that have been automatically generated during compilation. Those methods allow *setting* values or *getting* them. If the elements are kept on the list, as it is with the *answer* objects kept within the *question* one, the Java language itself provides convenient ways of dealing with that.

```
for(ListIterator j = question.getAnswer().listIterator(); j.hasNext();)
{
    Answer answer = (Answer)j.next();
    ...
}
```

If one is to get the first element, another Java syntax is used.

```
Answer answer = (Answer) (question.getAnswer().get(0));
```

Before marshalling objects into XML file, the Java object representation has to be verified if conforming with the DTD. It is performed just by calling *validate* method of test object (assuming validation of the test structure).

Marshalling to a XML document is again a trivial task. For the project purposes there is *saveTest* method prepared that takes care about that.

```
private void saveTest(Test test)
    throws IOException
{
    String testName = test.getName();
    String docsPath = getServletContext().getRealPath("/docs");
    File file = new File(docsPath + "/" + testName + ".xml");
    FileOutputStream fileOutputStream = new FileOutputStream(file);
    test.marshal(fileOutputStream);
    fileOutputStream.close();
    return;
}
```

2.2.4. XSL

The XSL style sheet (see Appendix A.1.4) is used for transformation from XML data representation of the test into its displayable HTML form. In fact, since XML has strict syntax, the result of XSLT is in XHTML format, rather than HTML.

The style sheet

An exemplary fragment of the sample XML file, is listed next.

```
<question type="singleChoice" value="1.0">
  <title>A botanical challenge</title>
  <description>Check a thing which is not a plant</description>
  <notice>This is a compulsory question</notice>
  <answer contents="true">A radiator</answer>
  <answer>A larch</answer>
</question>
```

The XSLT results in XHTML code given below.

```
<h3>1. A botanical challenge (1 point)</h3>
<p>Check a thing which is not a plant</p>
<blockquote>
  <i>This is a compulsory question</i>
</blockquote>
<p>
  <input value="1" name="q1" type="radio">A radiator<br>
  <input value="2" name="q1" type="radio">A larch<br>
</p>
<br>
```

As it can be seen, the transformation itself is not straightforward. The HTML tags are not just the XML's one but of different names. There are much more going on than only conversion of tags' names. The part of the XSL sheet which handles conversion of a question is presented next.

```
<xsl:template match="question">
  <xsl:variable name="questionPosition" select="position()"/>
  <hr/>
  <h3>
    <xsl:number value="$questionPosition" format="1. "/>
    <xsl:value-of select="title"/>
    <xsl:call-template name="points">
      <xsl:with-param name="value" select="@value"/>
    </xsl:call-template>
  </h3>
  <xsl:apply-templates select="description | notice | warning"/>
  <p>
    <xsl:apply-templates select="answer">
      <xsl:with-param name="questionNumber" select="$questionPosition"/>
    </xsl:apply-templates>
  </p>
  <br/>
</xsl:template>
```


The second line creates a variable that keeps question number within the test. The variable is then used to number question (the format chosen is of the XSL type “1. “) and when *answer* template is being called. At some stage the *points* template is called. This template is responsible for displaying how much is question worth.

```
<xsl:template name="points">
  <xsl:param name="value" select="UNDEFINED"/>
  <xsl:choose>
    <xsl:when test="$value>='2.0'">
      (<xsl:value-of select="$value"/> points)
    </xsl:when>
    <xsl:otherwise> (1 point)</xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Finally, there are four types of the *answer* template. One of them, used when a single choice question is being displayed is given next.

```
<xsl:template
  match="answer[parent::node()/attribute::type='singleChoice']"
  <xsl:param name="questionNumber" select="UNDEFINED"/>
  <xsl:choose>
    <xsl:when test="attribute::display='checked'">
      <input type="radio" name="q{$questionNumber}" value="{position()}"
        checked=""/><xsl:value-of select="."/><br/>
    </xsl:when>
    <xsl:otherwise>
      <input type="radio" name="q{$questionNumber}"
        value="{position()}"/><xsl:value-of select="."/><br/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

One can see the use of the XPath language when the template is being matched to the question type and later to determine answer’s position within the question element.

Exhaustive description of the XSL sheet could possibly take ten more pages, but for the sake of report’s compactness, only two more things are mentioned.

Since the final result of the XSL transformation is supposed to be HTML, the sheet has to inform the XSL processor about that.

```
<xsl:output method="html" encoding="ISO-8859-1"/>
```

The *encoding* attribute sets the character encoding for the output method.

In order to produce valid HTML page, when the root of the XML document is being processed, the skeleton of the HTML page is generated.

```

<xsl:template match="/">
  <html>
    <head>
      <title><xsl:value-of select="test/title"/></title>
      <link rel="StyleSheet" href="conf/style.css" type="text/css"
        media="screen"/>
    </head>
    <body>
      <xsl:apply-templates select="test"/>
    </body>
  </html>
</xsl:template>

```

The Java code

The XSLT is implemented by the *performXSLT* method. This method is used within the *Presenter* and *Checker* servlets.

```

private void performXSLT (InputStream data, InputStream style,
                          Writer result)
  throws TransformerConfigurationException, TransformerException
{
  StreamSource dataSource = new StreamSource(data);
  StreamSource styleSource = new StreamSource(style);
  StreamResult transformResult = new StreamResult(result);

  TransformerFactory tFactory = TransformerFactory.newInstance();
  Transformer transformer = tFactory.newTransformer(styleSource);

  transformer.transform(dataSource, transformResult);
}

```

2.2.5. The database and SQL

The tables

The system uses a set of tables. An obligatory table is the one named “systemusers”. It keeps users’ names, passwords and information about groups that users belong to. A user can follow into two categories of groups – the “manager” group and the rest of users. The manager kind of user has an access to the manager subsystem while other users can access only the testing part (and of course the login subsystem).

The table below shows the structure of the table.

```
mysql> describe systemusers;
+-----+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| user  | varchar(30)   |      |     |         |       |
| password | varchar(30)   |      |     |         |       |
| usergroup | varchar(30)   |      |     |         |       |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.27 sec)
```

For the testing purposes some basic table was created and two users were set .

```
mysql> select * from systemusers;
+-----+-----+-----+
| user  | password | usergroup |
+-----+-----+-----+
| manager | manager  | manager   |
| student | student  | student   |
+-----+-----+-----+
2 rows in set (0.06 sec)
```

The deployment of a test involves creation of a database table. The table presented below was created during deployment of the sample test (see Appendix A.2.1). As it can be seen users’ answers are stored as strings of a maximum length of 255 characters, which is specified by the *varchar(255)* type. The exception is the question for external evaluation and such question’s answer is saved in the column of the text type, which allows 65535 characters. The table’s name corresponds to the name of the XML file with the test.

```
mysql> describe test;
+-----+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| user  | varchar(30)   |      |     |         |       |
| q1    | varchar(255)  |      |     |         |       |
| q2    | varchar(255)  |      |     |         |       |
| q3    | varchar(255)  |      |     |         |       |
| q4    | text          |      |     |         |       |
+-----+-----+-----+-----+-----+-----+
5 rows in set (0.06 sec)
```

Some results stored in the table are presented below. One can see that in the case of single and multiple choice questions, the atomic answers are separated by [#].

```
mysql> select * from test;
+-----+-----+-----+-----+-----+
| user      | q1  | q2   | q3   | q4   |
+-----+-----+-----+-----+-----+
| student   | 1#  | 1#4# | larch | The answer the student |
| anonymous  | 1#  | 1#3# | oak   | The answer of someone  |
+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

The SQL queries

The SQL queries in this project are simple and are about five things, namely:

- listing existing tables;

```
SHOW TABLES;
```

- creation of a table;

an exemplary query for the table of the sample test (see Appendix A.2.1)

```
CREATE TABLE test (
    user VARCHAR(30) NOT NULL,
    q1 VARCHAR(255) NOT NULL,
    q2 VARCHAR(255) NOT NULL,
    q3 VARCHAR(255) NOT NULL,
    q4 TEXT NOT NULL);
```

- deletion of a table;

```
DROP TABLE [table_name];
```

- inserting data into a row of the table;

```
INSERT INTO [table_name] VALUES ('[user_name]', '[q1_answer]',
    '[q2_answer]', ..., '[qN_answer]');
```

- extracting data from table;

```
SELECT password,usergroup FROM systemusers
    WHERE user='[user_name]';
```

or

```
SELECT * from [table_name];
```

The Java code

The Java code responsible for connection to the database usually looks like this, presented below.

```
Connection conn = DriverManager.getConnection(dbURL, dbUser,
                                             dbPassword);
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery("[SQL query]");

while (rs.next())
{
    ...
}

rs.close();
stmt.close();
conn.close();
```

In the case of creation or deletion of a table, the *execute* method of *Statement* object is invoked; for inserting new data into table *executeUpdate* method is used.

2.2.6. The servlets

The system is made up of several servlets, which share application's functionality. Essentially, one can distinguish three parts of the system. The first one is responsible for authentication of system's user and this part includes the *Login* servlet. The next part, made up of the *Manager* servlet, serves as system's configuration centre and it is used for creation, deployment and editing of tests. It also allows checking tests' results and obtaining some tests' statistics. The last part containing the *Presenter*, *Checker* and *Acceptor* servlets, works for actual testing – it is responsible for displaying tests to the system's users, for checking validity of users' answers, for displaying results when it is appropriate (for anonymous users) and finally for sending users' answers to the database.

The Login servlet

GET request:

- try to obtain the session object
- try to extract the "ie.dcu.slawek.user" attribute from the session
 - if the attribute exists, display the logout/relogin page
 - if there is no attribute, display login page

POST request:

- try to obtain the session object; if the session has been found, it is invalidated
- look for request's parameter named "logout"; if it exists, redirect user to the index page and quit the servlet
- extract "user" and "password" parameters from the request
- if anonymous user has been detected (user: anonymous, password: anonymous), set the user group as anonymous
- if the user parameter indicates other than anonymous user, check database for password and group of given user
- compare given password with this extracted from the database
- if the passwords are not equal, redirect user to the forbidden page and quit the servlet
- if password matches, create session object and set "ie.dcu.slawek.user" and "ie.dcu.slawek.usergroup" attributes, redirect user to the index page and quit the servlet

The Manager servlet

GET request:

→ invoke the *doPost* method passing it the request and the response objects

POST request:

- try to obtain the session object; if the session has not been found, redirect user to the Login servlet and quit this servlet
- try to get the string with the user's name kept within the "ie.dcu.slawek.user" attribute; if it cannot be done, redirect user to the Login servlet and quit this servlet
- try to get the string with the user's group kept within the "ie.dcu.slawek.usergroup" attribute; if it cannot be done, redirect user to the Login servlet and quit this servlet
- look for following parameters in the request and take proper action if one is found:
 - "newTest" found, take the same steps as for "editTest"
 - "deleteTest" found, delete the test file
 - "editTest" found:
 - look request's parameter named "testName"; if it exists:
 - if the corresponding XML file exists:
 - if the test is deployed, redirect user to the forbidden page and quit the servlet
 - if the test not deployed:
 - perform unmarshalling of XML file with the test
 - set the name attribute of the test object to the name of the file
 - if there is no proper XML file:
 - create a new test object
 - set the test's name to the value of "testName" parameter
 - set the test's author to the value of the "ie.dcu.slawek.user" session attribute
 - set the test's date to actual date
 - set the test's tile to the value of "testName" parameter
 - put the test object onto the session
 - display the test

- if the "testName" parameter does not exist:
 - "printTest" found, display the test
 - "testTitle" found, set the test's title
 - "testAuthor" found, set the test's author
 - "testDate" found, set the test's date
 - "testDescription" found, set the test's description
 - "testNotice" found, set the test's notice
 - "testWarning" found, set the test's warning
 - "deleteQuestion" found, delete the question
 - "editQuestion":
 - extract question's number from "editQuestion" parameter
 - get the questions' list from the test object
 - if the question's number is equal to 0:
 - instantiate a new question
 - get the list of its answers
 - instantiate a new answer
 - put the answer on the answers' list
 - get "questionType" parameter's value and set the question type accordingly
 - set the question's title
 - put the question on the questions' list
 - if number of the question is greater than 0
 - "questionTitle" found, set the question's title
 - "questionValue" found, set the question's value
 - "questionDescription" found, set the question's description
 - "questionNotice" found, set the question's notice
 - "questionWarning" found, set the question's warning
 - "deleteAnswer" found, delete the answer

- "editAnswer":
 - extract answer's number from "editAnswer" parameter
 - get the answer' list from the question object
 - if the answer's number is equal to 0:
 - instantiate a new answer
 - put the answer on the answers' list
 - if number of the answer is greater than 0:
 - set the answer's content accordingly to the value of the "answerContent" parameter
 - set the answer contents accordingly to the value of "answerContents" parameter
 - set the answer's display accordingly to the value of "answerDisplay" parameter; if this parameter is equal to "null" set the answer's display to an empty string
- display the question accordingly to its type
- "saveTest" found:
 - validate the test object
 - marshal the test object to the XML file
 - remove the test object from the session
- "discardTest" found, remove the test object from the session
- "deployTest" found, deploy the test
- "undeployTest" found, undeploy the test
- "checkResults" found, get, evaluate and display the test's results [complex task]
- call the Manager servlet itself without any parameters
- there is no in the request, display the menu screen

The Presenter servlet

GET request:

- look for request's parameter named "testFileName";
- if the parameter does not exist, redirect user to the index page and quit the servlet
- if parameter exists, invoke the *doPost* method passing it the request and the response objects

POST request:

- look for request's parameter named "testFileName"; if the parameter does not exist, redirect user to the index page and quit the servlet
- try to obtain the session object; if there is no session object created redirect user to forbidden page and quit the servlet
- check with the database if the test has been deployed; if not, redirect user to forbidden page and quit the servlet
- perform unmarshalling of XML file with the test
- set the name attribute of the test object to the name of the file
- validate the test object
- put the test object onto the session as *ie.dcu.slawek.test* attribute
- get the writer
- invoke XSL transformation passing it the XML document, XSL style sheet and the writer

The Checker servlet

GET request:

→ redirect user to the forbidden page and quit the servlet

POST request:

→ try to obtain the session object; if there is no session object created, redirect user to forbidden page and quit the servlet

→ try to get the test object kept within the "ie.dcu.slawek.test" session's attribute; if it cannot be done, redirect user to forbidden page and quit the servlet

→ check for answers validity and mark test object to reflect user answers [complex task] by setting the answers' display attributes

→ if answer is valid:

→ update the test object kept in the session

→ invoke the Acceptor servlet and quit this servlet

→ if answer is invalid:

→ marshal the test object to memory

→ get the writer

→ invoke XSL transformation passing it the XML document from the memory, XSL style sheet and the writer

The Acceptor servlet

GET request:

- try to obtain the session object; if there is no session object created, redirect user to forbidden page and quit the servlet
- try to get the string with the user's name kept within the "ie.dcu.slawek.user" attribute; if it cannot be done, redirect user to forbidden page and quit the servlet
- try to get the test object kept within the "ie.dcu.slawek.test" session's attribute; if it cannot be done, redirect user to forbidden page and quit the servlet
- prepare string that contain answers of all the questions [complex task]
- send the answers to the database
- if the "ie.dcu.slawek.user" session's attribute indicates an anonymous user, evaluate results [complex task] and display the results page
- for all other users different than the anonymous one, show the confirmation page

POST request:

- redirect user to the forbidden page and quit the servlet

Answer evaluation algorithm

→ iterate through all question's answers

```
for(ListIterator j = question.getAnswer().listIterator(); j.hasNext();)
{
    Answer answer = (Answer)j.next();
    ...
}
```

→ for each answer check if the user has chosen it

```
if ((new String("checked")).equals(answer.getDisplay()))
    {...}
else
    {...}
```

→ if it is chosen but it is a wrong answer, mark the question as failed

```
if (!(new String("true")).equals(answer.getContents()))
    {answerFailed=true;}
```

→ if chosen and it is a “don't know” type answer, mark the question as unanswerd and keep a record of presence of such answer

```
if ((new String("dontKnow")).equals(answer.getContents()))
{
    dontKnowChosen = true;
    dontKnowCount++;
}
```

→ if answer was not chosen but it is the right one, mark the question as failed

```
if ((new String("true")).equals(answer.getContents()))
    {answerFailed=true;}
```

→ if it is not chosen and it is “don't know” type answer keep a record of presence of such answer

```
if ((new String("dontKnow")).equals(answer.getContents()))
    {dontKnowCount++;}
```

→ if the answer was not marked as failed or unanswerd, it is correct one

Final mark computation algorithm

→ determine numbers of answers excluding “don’t know” type ones

```
answersNo = (question.getAnswer().size()) - dontKnowCount;
```

→ if the user has chosen “don’t know” answer, assign 0 points to the question

```
if (dontKnowChosen)
    mark = 0;
```

→ if the user has given a wrong answer, assign to the mark of this question negative value of the penalty calculated for this question

```
if (answerFailed)
    mark = -penalty;
```

the penalty for a single choice question:

$$penalty = \frac{1}{answers_number - 1}$$

the penalty for a multiple choice question:

$$penalty = \frac{1}{2^{answers_number} - 2}$$

→ if the answer chosen was not of “don’t know” type and was not wrong one, assign one point to the question

→ multiply the number of points by the value of the question to get the final question’s mark

```
mark *= question.getValue();
```

Exceptions handling

The vital issue of every application is its ability to cope with any kind of errors. During development of this application, much effort was devoted to make it stable and robust. Nevertheless, there are situations, when because of the web server or the database failure an error may occur.

Fragments of code, where an error may happen are enclosed within *try* statement scope. If an error occurs, the exception's message is send to the web browser (if that is possible) and to the server's console.

```
try
{
    ...
}
catch (Exception ex)
{
    try
        {response.sendError(response.SC_INTERNAL_SERVER_ERROR,
                            ex.getMessage().toString());}
    catch (Exception exceptionMessageEx)
        {System.err.println(exceptionMessageEx.getMessage().toString());}
    finally
        {System.err.println(ex.getMessage().toString());}
}
```

Writer configuration

Since output of all the project's servlets is dynamically generated and is of HTML type, the response content type is set to *text/html* and the response is marked as not cacheable to proxy servers and clients by setting an HTTP header *pragma: no-cache*.

2.2.7. Other technologies incorporated into the project

CSS

Cascading Style Sheets play a small role in this project. There is one style sheet created (see Appendix A.1.5) and it is responsible for modelling appearance of HTML pages, those static and those created dynamically as well. Results of formatting done by the CSS are shown next.

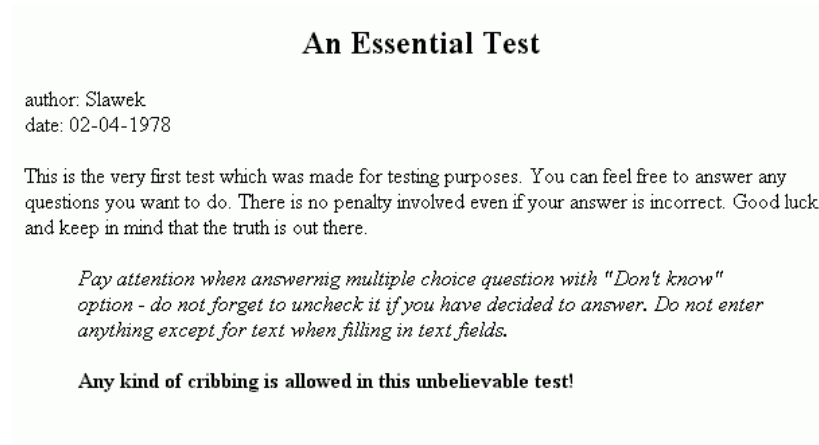


Figure 30. Unformatted HTML page.

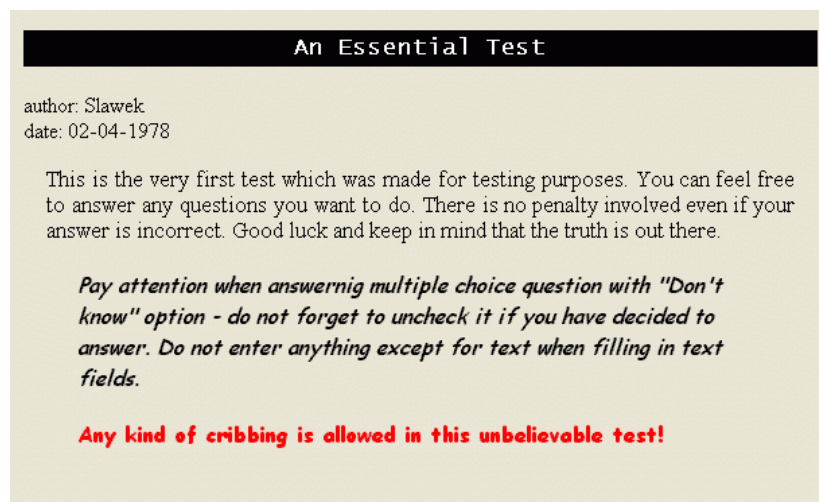


Figure 31. HTML page formatted by the CSS style sheet.

JavaScript

JavaScript is a script language designed specifically for Internet documents. Scripts written in this language can be embedded inside HTML pages. In general, the JavaScript language brings many new elements to the web pages.

The part of the manager subsystem, which is designed for checking tests' results, uses JavaScript to create a new web browser window, when one needs to check an answer of the question for an external evaluation.

```
function winOpen(userName, answer)
{
    msg=open("", "", "toolbar=no,directories=no,
                "menubar=no,scrollbars=yes,width=512,height=200");
    msg.document.write("<HTML><HEAD><TITLE>" + username + "</TITLE></HEAD>");
    msg.document.write("<BODY><P>" + answer + "</P></BODY></HTML>");
}
```

The function presented above is called when a user clicks the button of the HTML form shown below.

```
<FORM>
  <INPUT TYPE=BUTTON VALUE="Check it"
        ONCLICK="winOpen(' [username] ', ' [userAnswer] ') ">
</FORM>
```

Chapter 3

3. Installation and configuration

3.1. Software

3.1.1. Microsoft Windows98

All software run under Microsoft Windows98 operating system.

The *autoexec.bat* file was suitably modified to set necessary paths and environmental variables.

```
PATH C:\J2SDK\BIN;C:\TOMCAT\BIN;C:\JAXB\BIN;C:\WINDOWS;C:\WINDOWS\COMMAND;
SET JAVA_HOME=C:\J2SDK
SET CATALINA_HOME=C:\TOMCAT
SET JAVACMD=C:\J2SDK\BIN\java
SET CLASSPATH=C:\DEVELOP;C:\TOMCAT\COMMON\LIB\SERVLET.JAR;
    C:\MYSQL\JDBC_LIB\MYSQL_~1.JAR;
    C:\JAXB\LIB\JAXB-R~1.JAR;C:\JAXB\LIB\JAXB-X~1.JAR;
```

3.1.2. Sun Java 2 SDK

Java 2 Platform Standard Edition, version 1.4.0_01 has been installed on the machine. The chosen installation directory was *C:\J2SDK* and some changes were made to the *autoexec.bat* file (see section 3.1.1).

3.1.3. Sun Java Architecture for XML Binding

An early-access release of Sun's implementation of JAXB, the Java Architecture for XML Binding, has been installed to the *C:\JAXB* directory. The *autoexec.bat* file was then updated (see section 3.1.1).

3.1.4. Apache Tomcat web server

This project is based on Tomcat 4.0, a server that implements the Servlet 2.3 and JSP 1.2 Specifications from Java Software. The server has been installed into a *C:\TOMCAT* directory.

In order to install tomcat, a Java Development Kit (JDK) release of version 1.2 or later, has to be already installed in the system. An environment variable named *JAVA_HOME* has to be set to the pathname of the directory into which the JDK release has been installed (see section 3.1.1).

Finally, to avoid an “out of environment space” error when running the batch files that start and stop the Tomcat server, in Windows98, the "Initial environment" property of the shortcut to those batch files, needs to set to 4096.

Some changes were made to *server.xml* configuration file in the subdirectory *conf*; the server’s port number has been changed to 80 and automatic reloading of project’s classes has been enabled.

```
<Connector className="org.apache.catalina.connector.http.HttpConnector"
           port="80" minProcessors="5" maxProcessors="75"
           enableLookups="true" redirectPort="8443"
           acceptCount="10" debug="0" connectionTimeout="60000"/>

<!-- Slaweks Application Context -->
<Context path="/testing" docBase="elearn" debug="0" reloadable="true">
  <Logger className="org.apache.catalina.logger.FileLogger"
          prefix="testing_log." suffix=".txt" timestamp="true"/>
</Context>
```

3.1.5. MySQL Database

The system cooperates with the MySQL database software. The choice for this database was made because of several reasons. Firstly, this relational database management system is compact; it takes only around 30 MB of disk space. Secondly, it is Open Source Software, so it can be downloaded and used by everyone without paying anything (except for commercial purposes). Finally, the MySQL software is fast, reliable, and easy to use.

A MySQL database system has been installed to the *C:\MYSQL* directory. Then the *JDBC_LIB* subdirectory was created and a copy of a MySQL JDBC library was placed there. Next, some changes were made to the *autoexec.bat* file (see section 3.1.1). Finally, the *MyODBC* driver has been installed into the operating system.

For the purposes of the project, *elearn* database has been created along with its user servlet and its password *sheriff*. Lastly, the *systemusers* table has been prepared (see section 2.2.5).

3.2. Hardware

The project was implemented on a Dell Latitude CPi laptop equipped with Intel PentiumII processor, 128 MB RAM and 5GB HDD. This computer was fast enough to easily handle Tomcat web server and MySQL database.

Chapter 4

4. Conclusions

The initially stated assumptions for this project have been met. The final application works and, what is most important, is stable and robust. Although the report does not describe directly development steps that were devoted to errors' handling, the final result can be seen in the algorithms' description and in the code itself.

The application was supposed to be flexible and user-friendly as well. The final product may be considered to come up to these assumptions. The part of the system, which is responsible for the managing, is easy to use and equipped with most vital functions.

One of the major project's goals was an investigation of possibilities that XML provides. XML is the format chosen for tests' storage. Eventually, it turned out that it was an excellent choice. XML offers a great flexibility for a developer and allows implementing almost every data structure. The real strength of XML is its companion standards and availability of software libraries with their implementation. The XSL provides a way for conversion XML document into some other document. This project makes use of the XSL transform to produce a HTML page containing questions' forms that are displayed to the end user of the system. Since proper Java libraries have been developed, such conversion is almost straightforward to implement.

The JAXB, although not finally released, does work fine. This technology is truly useful and makes developer's life much easier. The help that JAXB provides for XML processing could not be overestimated. Hopefully, the next release will include support for XML Schema, which theoretically could even further simplify using the XML binding by simplifying Binding Schema for given XML structure (XML Schema is more exact than DTD, so fewer ambiguities needed to be clear up by Binding Schema).

The project is about server-side development. The application consists of several Java servlets. These are programs that extend web server functionality and, since they are normal Java programs, they can potentially do any kind of work.

The servlets of the project perform XML processing and talk to the database. As many other servlets, they use methods of the Servlet API for HTTP requests' processing and session management.

There is a big room for improvement considering code of the servlets. First of all, the Manager servlet may be decomposed into two parts and one of them might be responsible only for tests' editing while the other would handle remaining functionality of the manager. The other improvement, which could simplify the code, would be connected to JAXB. It is possible to extend the derived *Test* class and to move some servlets' methods, especially those responsible for users' answers validation and evaluation, to its body.

The database part of the system may also be improved. The passwords are stored in plaintext in the table. From the security point of view, only passwords' digests, produced by MD5 or another one-way hashing function, should be kept in the database.

The project touches on many modern and interesting technologies and for me, the author, it gave a great opportunity to explore those techniques and to do something useful at the same time.

References

- [1] W3C Recommendation, “Extensible Markup Language (XML) 1.0”, February 1998, <http://www.w3.org/TR/1998/REC-xml-19980210.html>
- [2] W3C Recommendation, “Extensible Stylesheet Language (XSL) 1.0”, October 2001, <http://www.w3.org/TR/2001/REC-xsl-20011015/xslspec.html>
- [3] W3C Recommendation, XSL Transformations (XSLT) 1.0”, November 1999, <http://www.w3.org/TR/1999/REC-xslt-19991116.html>
- [4] “The Java™ Web Services Tutorial”, Sun Microsystems, June 2002, <http://java.sun.com/webservices/downloads/webservices/tutorial.html>,
- [5] Elliotte Rusty Harold, “XML Bible, Gold edition”, November 2001
- [6] Elliotte Rusty Harold and W. Scott Means, “XML in a Nutshell”, January 2001
- [7] Eric Armstrong , “The Java API for Xml Processing (JAXP) Tutorial”, December 2001, <http://>
- [8] “Document Type Definition (DTD) Tutorial”, TheScarms, October 2000, <http://www.thescarms.com/XML/DTDTutorial.asp>
- [9] Miloslav Nic, “DTD Tutorial”, Zvon, <http://teachwww.cogs.susx.ac.uk/vs/it/dtdZvonTut.html>
- [10] “DTD Tutorial”, W3Schools, <http://www.w3schools.com/dtd/default.asp>
- [11] “An XML Schema Tutorial”, TheScarms, October 2000, <http://www.thescarms.com/XML/SchemaTutorial.asp>
- [12] “XML Schema Tutorial”, W3Schools, <http://www.w3schools.com/XMLSchema/default.asp>
- [13] Benoît Marchal, “XML by Example”, Que, 2000
- [14] “Kurs języka XML”, Paweł Stroiński, 2001, <http://www.pabloware.w.pl>
- [15] „Prezentacja XML & XSL”, Bartłomiej Bóbski and Tomasz Zieliński, <http://>
- [16] „The Java™ Architecture for XML Binding User’s Guide”, Sun Microsystems, May 2001
- [17] “Data binding with JAXB”, IBM, <http://www-105.ibm.com/developerWorks>
- [18] Sam Brodtkin, “Use XML data binding to do your laundry”, JavaWorld, December 2001, file:///D:/temp/JAXB/Use_JAXB_to_do_your_laundry/jw-1228-jaxb.zip
- [19] “The Java™ Tutorial, Trail: Servlet”, Cynthia Bloch and Stephanie Bodoff, Sun Microsystems, , <http://java.sun.com/docs/books/tutorial/index.html>

- [20] „Servlet Essential, v.1.3.5“, Stefan Zeiger, <http://www.novocode.com/doc/servlet-essentials>
- [21] “Tutorial on Servlets and JSP”, Marty Hall, 1999, <http://>
- [22] “Fundamentals of Java™ Servlets”, MageLang Institute, <http://www.jguru.com/portal/index.html>
- [23] “Servlet Tutorial”, AcknowledgeTECHNOLOGIES, 2002, <http://www.acknowledge.co.uk/download/>
- [24] “MySQL Reference Manual”, MySQL AB, 2001, <http://www.mysql.com>
- [25] Maydene Fisher, “The Java Tutorial, Trail: JDBC(TM) Database Access”, Sun Microsystems, <http://java.sun.com/docs/books/tutorial/index.html>
- [26] David Cornelius, “Introduction to SQL”, <http://www.corneliusconcepts.com/study/IntroToSQL/>
- [27] Paweł Wimmer, “Kurs HTML”, PC Kurier, 2001, <http://www.pckurier.pl/html/index.htm>

Appendix

A.1. Essential project files

A.1.1. XML Schema for the project's XML data format (test.xsd)

```
<?xml version="1.0" encoding="UTF-8"?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:simpleType name="authorType">
    <xsd:restriction base="xsd:string"/>
  </xsd:simpleType>

  <xsd:simpleType name="dateType">
    <xsd:restriction base="xsd:date"/>
  </xsd:simpleType>

  <xsd:simpleType name="titleType">
    <xsd:restriction base="xsd:string"/>
  </xsd:simpleType>

  <xsd:simpleType name="descriptionType">
    <xsd:restriction base="xsd:string"/>
  </xsd:simpleType>

  <xsd:simpleType name="noticeType">
    <xsd:restriction base="xsd:string"/>
  </xsd:simpleType>

  <xsd:simpleType name="warningType">
    <xsd:restriction base="xsd:string"/>
  </xsd:simpleType>

  <xsd:simpleType name="nameAttribute">
    <xsd:restriction base="xsd:NMTOKEN"/>
  </xsd:simpleType>

  <xsd:simpleType name="contentsAttribute">
    <xsd:restriction base="xsd:string"/>
  </xsd:simpleType>

  <xsd:simpleType name="displayAttribute">
    <xsd:restriction base="xsd:string"/>
  </xsd:simpleType>

  <xsd:simpleType name="typeAttribute">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="singleChoice"/>
      <xsd:enumeration value="multipleChoice"/>
      <xsd:enumeration value="stringMatch"/>
      <xsd:enumeration value="externalEvaluation"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:simpleType name="valueAttribute">
    <xsd:restriction base="xsd:decimal"/>
  </xsd:simpleType>
```



```

<xsd:complexType name="answerType">
  <xsd:simpleContent>
    <xsd:extension base="xsd:string">
      <xsd:attribute name="contents" type="contentsAttribute" use="optional"
default="false"/>
      <xsd:attribute name="display" type="displayAttribute" use="optional"/>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>

<xsd:complexType name="questionType">
  <xsd:sequence>
    <xsd:element name="title" type="titleType"/>
    <xsd:element name="description" type="descriptionType" minOccurs="0"/>
    <xsd:element name="notice" type="noticeType" minOccurs="0"/>
    <xsd:element name="warning" type="warningType" minOccurs="0"/>
    <xsd:element name="answer" type="answerType" maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="type" type="typeAttribute" use="required"/>
  <xsd:attribute name="value" type="valueAttribute" use="optional"
default="1.0"/>
</xsd:complexType>

<xsd:complexType name="testType">
  <xsd:sequence>
    <xsd:element name="author" type="authorType"/>
    <xsd:element name="date" type="dateType"/>
    <xsd:element name="title" type="titleType"/>
    <xsd:element name="description" type="descriptionType" minOccurs="0"/>
    <xsd:element name="notice" type="noticeType" minOccurs="0"/>
    <xsd:element name="warning" type="warningType" minOccurs="0"/>
    <xsd:element name="question" type="questionType"
maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="name" type="nameAttribute" use="optional"/>
</xsd:complexType>

<xsd:element name="test" type="testType"/>

</xsd:schema>

```

A.1.2. DTD for the project's XML data format (test.dtd)

```
<!ELEMENT test (author, date, title, description?, notice?, warning?,
                question+) >
<!ATTLIST test name NMTOKEN #IMPLIED >
<!ELEMENT author (#PCDATA) >
<!ELEMENT date (#PCDATA) >
<!ELEMENT title (#PCDATA) >
<!ELEMENT description (#PCDATA) >
<!ELEMENT notice (#PCDATA) >
<!ELEMENT warning (#PCDATA) >
<!ELEMENT question (title, description?, notice?, warning?, answer+) >
<!ATTLIST question
    type (singleChoice | multipleChoice | stringMatch |
         externalEvaluation) #REQUIRED
    value CDATA "1.0" >
<!ELEMENT answer (#PCDATA) >
<!ATTLIST answer
    contents CDATA #IMPLIED
    display CDATA #IMPLIED >
```

A.1.3. JAXB binding schema for the project's XML data format (test.xjs)

```
<xml-java-binding-schema version="1.0ea">

<element name="test" type="class" root="true" />
<element name="question" type="class" >
  <attribute name="type" convert="questionType" />
  <attribute name="value" convert="float"/>
</element>
<enumeration name="questionType" members="singleChoice multipleChoice
  stringMatch externalEvaluation"/>

</xml-java-binding-schema>
```

A.1.4. XSL style sheet for the project's XML data format (test.xsl)

```
<?xml version="1.0" encoding="UTF-8"?>

<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="html" encoding="ISO-8859-1"/>

  <xsl:template match="/">
    <html>
      <head>
        <title><xsl:value-of select="test/title"/></title>
        <link rel="StyleSheet" href="conf/style.css" type="text/css"
          media="screen"/>
      </head>
      <body>
        <xsl:apply-templates select="test"/>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="test">
    <h2 align="center"><xsl:value-of select="title"/></h2>
    <xsl:text>author: </xsl:text><xsl:value-of select="author"/><br/>
    <xsl:text>date: </xsl:text><xsl:value-of select="date"/>
    <xsl:apply-templates select="description | notice | warning"/>
    <br/>
    <form action="checker" method="Post">
      <xsl:apply-templates select="question"/>
      <hr/><br/>
      <center><table><tr>
        <td><input type="submit" value="Submit Test"/></td>
        <td width="15"></td>
        <td><input type="reset" value="Reset Form"/></td>
      </tr></table></center>
    </form>
    <br/>
  </xsl:template>

  <xsl:template match="question">
    <xsl:variable name="questionPosition" select="position()"/>
    <hr/>
    <h3><xsl:number value="$questionPosition" format="1. "/>
      <xsl:value-of select="title"/>
      <xsl:call-template name="points">
        <xsl:with-param name="value" select="@value"/>
      </xsl:call-template>
    </h3>
    <xsl:apply-templates select="description | notice | warning"/>
    <p>
      <xsl:apply-templates select="answer">
        <xsl:with-param name="questionNumber" select="$questionPosition"/>
      </xsl:apply-templates>
    </p>
    <br/>
  </xsl:template>
```

```

<xsl:template name="points">
  <xsl:param name="value" select="UNDEFINED"/>
  <xsl:choose>
    <xsl:when test="$value>='2.0'">
      (<xsl:value-of select="$value"/> points)</xsl:when>
    <xsl:otherwise> (1 point)</xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template match="description">
  <P><xsl:value-of select="."/></P>
</xsl:template>

<xsl:template match="notice">
  <blockquote>
    <i><xsl:value-of select="."/></i>
  </blockquote>
</xsl:template>

<xsl:template match="warning">
  <blockquote>
    <b><xsl:value-of select="."/></b>
  </blockquote>
</xsl:template>

<xsl:template
  match="answer[parent::node()/attribute::type='singleChoice']">
  <xsl:param name="questionNumber" select="UNDEFINED"/>
  <xsl:choose>
    <xsl:when test="attribute::display='checked'">
      <input type="radio" name="q{$questionNumber}" value="{position()}"
        checked=""/><xsl:value-of select="."/><br/>
    </xsl:when>
    <xsl:otherwise>
      <input type="radio" name="q{$questionNumber}"
        value="{position()}" /><xsl:value-of select="."/><br/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template
  match="answer[parent::node()/attribute::type='multipleChoice']">
  <xsl:param name="questionNumber" select="UNDEFINED"/>
  <xsl:choose>
    <xsl:when test="attribute::display='checked'">
      <input type="checkbox" name="q{$questionNumber}"
        value="{position()}" checked=""/><xsl:value-of select="."/><br/>
    </xsl:when>
    <xsl:otherwise>
      <input type="checkbox" name="q{$questionNumber}"
        value="{position()}" /><xsl:value-of select="."/><br/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template
  match="answer[parent::node()/attribute::type='stringMatch']">
  <xsl:param name="questionNumber" select="UNDEFINED"/>
  <input type="text" name="q{$questionNumber}" size="25"
    maxlength="25" value="{attribute::display}" /><br/>
</xsl:template>

```

```
<xsl:template
  match="answer[parent::node()/attribute::type='externalEvaluation']">
  <xsl:param name="questionNumber" select="UNDEFINED"/>
  <textarea name="q{$questionNumber}" rows="5" cols="80"
    wrap="virtual">
    <xsl:value-of select="attribute::display"/></textarea><br/>
  </xsl:template>
</xsl:stylesheet>
```

A.1.5. CSS style sheet for HTML's display formatting (style.css)

```
BODY {background-color: #EDE7DA}

H1 {font-family: monospace;
    font-size: 20px;
    padding: 6px}
H2 {font-family: monospace;
    font-size: 17px;
    color:white;
    background-color: black;
    padding: 5px}
H3 {font-family: monospace;
    font-size: 17px;
    padding: 5px}

B {color: red}

BLOCKQUOTE {font-family: cursive}

TEXTAREA {font-family: sans-serif;
           font-size: 14px}

TABLE {font-family: sans-serif;
        font-size: 14px}

INPUT {font-family: sans-serif;
        font-size: 14px;}

A:link      {font-family: serif;
             font-size: 17px;
             letter-spacing: 0.75px;
             color: navy}
A:active    {font-family: serif;
             font-size: 17px;
             letter-spacing: 0.75px;
             color: red}
A:visited   {font-family: serif;
             font-size: 17px;
             letter-spacing: 0.75px;
             color: gray}

P {font-family: serif;
   font-size: 17px;
   letter-spacing: 0.75px;
   margin: 17px 17px;
   text-align: justify}
```

A.1.6. The project's web deployment descriptor (web.inf)

```
<!DOCTYPE web-app
  PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
  "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>

  <display-name>Slaweks Application</display-name>
  <description>An e-learning application which deals with on-line testing.
  </description>

  <servlet>
    <servlet-name>login</servlet-name>
    <servlet-class>Login</servlet-class>
    <init-param>
      <param-name>indexURL</param-name>
      <param-value>index.html</param-value>
    </init-param>
    <init-param>
      <param-name>forbiddenURL</param-name>
      <param-value>forbidden.html</param-value>
    </init-param>
    <init-param>
      <param-name>dbDriver</param-name>
      <param-value>org.gjt.mm.mysql.Driver</param-value>
    </init-param>
    <init-param>
      <param-name>dbURL</param-name>
      <param-value>jdbc:mysql://localhost/elearn</param-value>
    </init-param>
    <init-param>
      <param-name>dbUser</param-name>
      <param-value>servlet</param-value>
    </init-param>
    <init-param>
      <param-name>dbPassword</param-name>
      <param-value>sheriff</param-value>
    </init-param>
  </servlet>

  <servlet>
    <servlet-name>presenter</servlet-name>
    <servlet-class>Presenter</servlet-class>
    <init-param>
      <param-name>indexURL</param-name>
      <param-value>index.html</param-value>
    </init-param>
    <init-param>
      <param-name>forbiddenURL</param-name>
      <param-value>forbidden.html</param-value>
    </init-param>
    <init-param>
      <param-name>dbDriver</param-name>
      <param-value>org.gjt.mm.mysql.Driver</param-value>
    </init-param>
    <init-param>
      <param-name>dbURL</param-name>
      <param-value>jdbc:mysql://localhost/elearn</param-value>
    </init-param>
  </servlet>

```



```

<init-param>
  <param-name>dbUser</param-name>
  <param-value>servlet</param-value>
</init-param>
<init-param>
  <param-name>dbPassword</param-name>
  <param-value>sheriff</param-value>
</init-param>
</servlet>

<servlet>
  <servlet-name>checker</servlet-name>
  <servlet-class>Checker</servlet-class>
  <init-param>
    <param-name>forbiddenURL</param-name>
    <param-value>forbidden.html</param-value>
  </init-param>
</servlet>
<servlet>
  <servlet-name>acceptor</servlet-name>
  <servlet-class>Acceptor</servlet-class>
  <init-param>
    <param-name>indexURL</param-name>
    <param-value>index.html</param-value>
  </init-param>
  <init-param>
    <param-name>forbiddenURL</param-name>
    <param-value>forbidden.html</param-value>
  </init-param>
  <init-param>
    <param-name>dbDriver</param-name>
    <param-value>org.gjt.mm.mysql.Driver</param-value>
  </init-param>
  <init-param>
    <param-name>dbURL</param-name>
    <param-value>jdbc:mysql://localhost/learn</param-value>
  </init-param>
  <init-param>
    <param-name>dbUser</param-name>
    <param-value>servlet</param-value>
  </init-param>
  <init-param>
    <param-name>dbPassword</param-name>
    <param-value>sheriff</param-value>
  </init-param>
</servlet>

<servlet>
  <servlet-name>manager</servlet-name>
  <servlet-class>Manager</servlet-class>
  <init-param>
    <param-name>indexURL</param-name>
    <param-value>index.html</param-value>
  </init-param>
  <init-param>
    <param-name>forbiddenURL</param-name>
    <param-value>forbidden.html</param-value>
  </init-param>
  <init-param>
    <param-name>dbDriver</param-name>
    <param-value>org.gjt.mm.mysql.Driver</param-value>
  </init-param>

```

```

<init-param>
  <param-name>dbURL</param-name>
  <param-value>jdbc:mysql://localhost/elearn</param-value>
</init-param>
<init-param>
  <param-name>dbUser</param-name>
  <param-value>servlet</param-value>
</init-param>
<init-param>
  <param-name>dbPassword</param-name>
  <param-value>sheriff</param-value>
</init-param>
</servlet>

<servlet-mapping>
  <servlet-name>login</servlet-name>
  <url-pattern>/login</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>presenter</servlet-name>
  <url-pattern>/presenter</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>checker</servlet-name>
  <url-pattern>/checker</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>acceptor</servlet-name>
  <url-pattern>/acceptor</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>manager</servlet-name>
  <url-pattern>/manager</url-pattern>
</servlet-mapping>
</web-app>

```

A.2. Other files

A.2.1. XML file containing a sample test (test.xml)

```
<?xml version="1.0" encoding="UTF-8"?>

<test name="test">
  <author>John Doe</author>
  <date>02-04-1978</date>
  <title>An Essential Test</title>
  <description>This is the very first test which was made for testing
purposes. You can feel free to answer any questions you want to do. There
is no penalty involved even if your answer is incorrect. Good luck and
keep in mind that the truth is out there.
  </description>
  <notice>Pay attention when answernig multiple choice question with
"Don't know" option - do not forget to uncheck it if you have decided to
answer. Do not enter anything except for text when filling in text fields.
  </notice>
  <warning>Any kind of cribbing is allowed in this unbelievable test!
  </warning>
  <question type="singleChoice" value="1.0">
    <title>A botanical challenge</title>
    <description>Check a thing which is not a plant</description>
    <notice>This is a compulsory question</notice>
    <answer contents="true">A radiator</answer>
    <answer>A larch</answer>
  </question>
  <question type="multipleChoice">
    <title>A difficult question</title>
    <description>Check vehicles</description>
    <answer contents="true">A bike</answer>
    <answer contents="false">An orchestra</answer>
    <answer contents="false">A hot-dog</answer>
    <answer contents="true">A lorry</answer>
    <answer contents="dontKnow" display="checked">Don't know</answer>
  </question>
  <question type="stringMatch" value="2.0">
    <title>A difficult task</title>
    <description>Type a word which is abnormal</description>
    <warning>No need to hurry!</warning>
    <answer contents="larch" display="kind of tree"></answer>
  </question>
  <question type="externalEvaluation" value="2.0">
    <title>An easy mission here</title>
    <description>Just type a few words</description>
    <notice>Please keep your answer short but clear</notice>
    <answer/>
  </question>
</test>
```

A.2.2. The index page (index.html)

```
<HTML>

<HEAD>
<TITLE>Main page of Slaweks application</title>
<LINK REL=StyleSheet HREF="conf/style.css" TYPE="text/css" MEDIA=screen>
</HEAD>

<BODY>

<CENTER>
<H2>Introduction</H2>
</CENTER>
<P>Welcome to the main page of a system designed for interactive testing
for distance learning purposes. The system consists of several parts and
is a fully featured. It deals with creation, storing, presentation and
evaluation of tests. They may be composed of different types of questions
- single choice, multiple choice ones and such those that expect text
answer. All tests are stored in XML files designed in a way that making
them is a trivial task. Moreover there is a tool provided for automatic
generation of XML files. The application cooperates with a MySQL(R)
database which keeps authentication information like user names and
passwords and also information of tests evaluation results for each user.
The results are also processed to give some statistics to the authorised
user, let say to a tutor of a course who have prepared tests.</P>

<CENTER>

<H2>Testing</H2>
<H3>Log in/out to/from the system</H3>
<TABLE ALIGN=CENTER><FORM ACTION="login" METHOD=GET>
<TR><TD WIDTH=50% ALIGN=RIGHT>Press the button</TD>
<TD ALIGN=LEFT><INPUT TYPE=SUBMIT VALUE="Login or logout"></TD></TR>
</FORM></TABLE>
<BR>

<H3>Try a test (you have to be logged in already)</H3>
<TABLE ALIGN=CENTER><FORM ACTION="presenter" METHOD=POST>
<TR><TD WIDTH=50% ALIGN=RIGHT>Test name:</TD>
<TD ALIGN=LEFT><INPUT TYPE=TEXT NAME="testFileName"></TD></TR>
<TR><TD></TD><TD><INPUT TYPE=SUBMIT VALUE="Let me try"></TD></TR>
</FORM></TABLE>
<BR>

<H3>Management (you have to be logged in as a manager)</H3>
<TABLE ALIGN=CENTER><FORM ACTION="manager" METHOD=GET>
<TR><TD WIDTH=50% ALIGN=RIGHT>Press the button</TD>
<TD ALIGN=LEFT><INPUT TYPE=SUBMIT VALUE="Manage the system"></TD></TR>
</FORM></TABLE>
<BR>

</CENTER>

</BODY>

</HTML>
```

A.2.2. The forbidden page (forbidden.html)

```
<HTML>

<HEAD>
<LINK REL=StyleSheet HREF="conf/style.css" TYPE="text/css" MEDIA=screen>
<TITLE>Access Denied Page</TITLE>
</HEAD>

<BODY>
<CENTER>
<BR><BR><BR>
<H1>Access denied</H1>
</CENTER>
</BODY>

</HTML>
```

A.3. A floppy disk with all project's files

