A JAVA BASED DEVELOPMENT ENVIRONMENT FOR SPRINGFRAMEWORK

by

Fatih Kasap

B.S., Computer Science Engineering, Marmara University, 2004

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in System and Control Engineering

Boğaziçi University

2008

ACKNOWLEDGEMENTS

I would like to express my great thanks to those who supported me with love, patience, understanding and care. Without their motivating words reflecting their characters the result might be a dream. Sevim Kesimoğlu, Adem Kasap, Dentist Çiğdem Karaman, Computer Engineer Sinan Kasap for being my family and the most powerful supporter for me to complete my thesis. My fiance for a long period during the study and now wife philologist Zuhra Kasap for her stimulating emotional responses to the situation of the thesis, which helped me finish it on time. Computer and System & Control Engineer Serdar Akar for his fellowship, brotherhood, exceptional recommendations and knowledge. Jury members Dr. Tamer Şıkoğlu and Dr. Mehmet Melek for their superior understanding ability. The writers of the books that I read and has left something good and valuable in my mind for their knowledge that they share with others. Finally I want to thank to my thesis supervisor Assoc. Prof. Osman Nuri Darcan for his endless efforts to consummate the thesis document. Love and idealism are two wisdoms behind a support given for a person. Hypocrisy that is in the expectation of a benefit means nothing besides them.

ABSTRACT

A JAVA BASED DEVELOPMENT ENVIRONMENT FOR SPRINGFRAMEWORK

Web application development is one of the most important programming areas today. There are many technologies to provide a better and easier way of programming for web applications in the market. Some of them are open source and free whereas some of them are released by big software vendors such as Microsoft and IBM. The Springframework is an open source alternative in this area. It is a Java-based technology to provide a robust infrastructure to software development based on best practices and accepted standards. This thesis aims to provide a computer program to facilitate Springframework-based web application development by hiding low level tasks from the developer and trying to make it focus on the real purpose of his application. Based on user friendliness and being close to human nature this program provides automatic code and web page generation, database and internationalization support, easy file management.

ÖZET

SPRINGFRAMEWORK İÇİN JAVA TABANLI BİR GELİŞTİRME ORTAMI

Web için uygulama geliştirme bugün en önemli programcılık alanlarından birisini teşkil etmektedir. Web uygulamaları için daha iyi ve kolay programlama yolları sağlamak üzere şu anda sunulmuş ve kullanımda olan bir çok teknoloji mevcuttur. Bunlardan bir kısmı açık kaynak kodlu iken bazıları ise Microsoft ve IBM gibi büyük yazılım sağlayıcılar tarafından pazara sürülmektedir. Springframework bu pazarda açık kaynak kodlu bir alternatif olarak bulunmaktadır. Java tabanlı olan bu teknoloji, en iyi pratikler ve kabul edilmiş standartlar temelinde yazılım geliştirilmesi için sağlam bir alt yapı sunmak üzere tasarlanmıştır. Bu tez, düşük seviye görevleri kullanıcılarından saklayarak ve onları uygulamalarının gerçek amaçlarına odaklandırarak, Springframework üzerinde çalışan web uygulamaları geliştirme işini kolaylaştırmak üzere bir bilgisayar programı ortaya koymayı amaçlamaktadır. Kullanıcı dostu olma ve insan doğasına yakın olma temeline dayanan bu yazılım, otomatik kod ve sayfa üretimi, veritabanı ve uluslararası dil desteği ve kolay dosya yönetimi sağlamaktadır.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS / ABBREVIATIONS

AOP	Aspect Oriented Programming
API	Application Programming Interface
CASE	Computer-Aided Software Engineering
CI	Constructor Injection
CLI	Command Line Interface
CSS	Cascading Style Sheets
DAO	Data Access Object
DI	Dependency Injection
GUI	Graphical User Interface
HTTP	Hyper Text Transfer Protocol
IDE	Integrated Development Environment
IOC	Inversion of Controller
J2EE	Java 2 Enterprise Edition
JDBC	Java Database Connectivity
JDK	Java Development Kit
JIT	Just-In-Time
JSP	Java Server Pages
JVM	Java Virtual Machine
MVC	Model View Control
P2P	Peer-To-Peer
PC	Personal Computer
PDA	Personal Digital Assistant
POJO	Plain Old Java Objects
SGML	Standard Generalized Markup Language
SI	Setter Injection
UML	Unified Modeling Language
WIMP	Window, Icon, Menu, Pointing Device
WWW	World Wide Web
XML	Extensible Markup Language

1. INTRODUCTION

Computer programming is the process of writing, testing, debugging, troubleshooting, and maintaining the source code of computer programs. The purpose of programming is to create a program that exhibits a certain desired behavior [1]. Recent years have witnessed breakthrough developments in computer programming. Especially *Internet* opened new areas. Before, there were *client-server architecture*s but not in the magnitude of Internet's greatness and not being able to answer Internet' needs. This broadening of programming through client-server architectures led to a new field called web development.

Today, web development is a broad term for any activity related to developing a web site for World Wide Web (WWW) or an intranet. It can range from developing the simplest static single page of plain text to the most complex web-based Internet applications, electronic businesses, or social network services.

For developing better Internet applications, there are many attempts to provide a complete, secure, not too complex infrastructure technology. One of these new technologies is the Springframework which is a framework for developing applications based on Java technology, Sun Microsystems' programming platform, securely, rapidly and in a planned way.

The Springframework is a powerful technology. It is a lightweight container, providing centralized, automated configuration and wiring of web application objects [2]. It provides abstraction for database transactions, reduces code necessary to write for error and exception handling. In addition, it integrates with other good technologies such as Hibernate, Toplink and iBatis, thus enabling those proven technologies to be used without much effort and problems. The Springframework supports Aspect Oriented Programming (AOP) which is a new way of programming to modularize the code and separate the concerns of a computer program. These and many other advantages and features of the Springframework make it popular in web development industry. Nonetheless learning and applying it to web development without any tool is a complicated process. To facilitate this

process, Springframework programmers are in need of a tool that helps them to manage their web programs more easily. There are a few examples of such tools already in the market but they are integrated into other huge and heavy development platforms such as NetBeans and Eclipse. An application that can operate standalone does not exist yet. In this thesis document, a computer program developed for the Springframework is reported. This program's purpose is "to be open-source and free from other development environments, and a computer software that facilitates developing Springframework-based web applications".

The computer program has a user-friendly look-and-feel, brings a programming approach that is close to what exists already, is extensible, modular, powerful, bug-free and acceptable among software industry developers. The way that other editors and integrated development environments follow in both look-and-feel and programming approaches are preferred for its programming methodology. Some of its main features are automatic code generation for web pages and Java classes, one-click compilation and deployment of a web application project, adding database support and functionalities to use that database, background management of configuration files of a web application project, easy file addition to and subtraction from a web application project, default internationalization support, advanced code highlighting, simple and friendly look-and-feel.

This thesis document is structured as follows: In section 2, a background survey is presented to the reader to let him have a better idea about what the Springframework is and its underlying technologies are. This section additionally describes which key technologies are used during the thesis development. In section 3, there is a literature overview. In section 4, the resultant program is described in detail in 3 sub sections. A general overview is given first, later graphical aspects are reported and lastly program's code is analysed package by package and important classes are indicated with their functionalities and jobs. In section 5 is a sample application developed step by step using the program. Section 6 is conclusion and is followed by the references.

2. BACKGROUND

The Springframework is based on simplicity and this is the key idea for how it provides its advanced features like integration with so many other technologies and frameworks easily. While achieving that the Springframework makes use of, works with or depends on many trivets such as dependency injection and aspect oriented programming.

This background section will try to give an understanding before going to examine the resultant program. First, graphical user interfaces are described and some example graphical user interfaces are presented because the resultant program has a graphical user interface that provides the interaction between the user and program's functions. In the next subsection, framework concept is mentioned followed by the subsection the Springframework. These sections are included since Springframework is the technology for which the program is written. In this part, a key concept called dependency injection supporting the Springframework technology is introduced. Subsequently, the subsections Java programming language, extensible markup language technology, plain old java objects and beans concept follow. Java is the language with which the Springframework and the thesis program are written. Extensible markup language, plain old java objects and beans are certain technologies that are in relation with the Springframework. Last three subsections mention about model-view-controller architectural software design pattern, unified modeling language and Hypersonic SQL database engine. For the thesis program, model-view-controller is the main focus part of the Springframework. Unified modeling language is a standard accepted in software development process and was used during the development of the thesis program. Lastly Hypersonic SQL is the database engine used to extend the capabilities of a project developed using the thesis program.

2.1 Graphical User Interface

Graphical User Interface (GUI) is a type of user interface that enables human users to interact with a device. GUIs differ from text-based interfaces since they have graphical components like icons, visual indicators or preformed library controls such as a combo box or a text field. Any item on a GUI is presented to the user with text to give a thorough

understanding of why it is there. Items consisting of only text are very rare. The name of GUI usually means a two dimensional interface.

Computer GUIs are manipulated by using cursors directed by a device called mouse and are based on the PARC User Interface which consists of widgets such as windows, menus, radio buttons, check boxes and icons. A widget or control is an element of a GUI that displays an information arrangement changeable by the user, such as a window or a text box [1]. The PARC User Interface brought the pointing device and graphical components to the arena of graphical mediums. In parallel with the technology advancements in computer hardware, GUIs have been also developed and have taken the form now in common use. Figure 2.1 shows the look of the first GUI-centric computer operating model called Star Information System that was developed in 1981 [1].

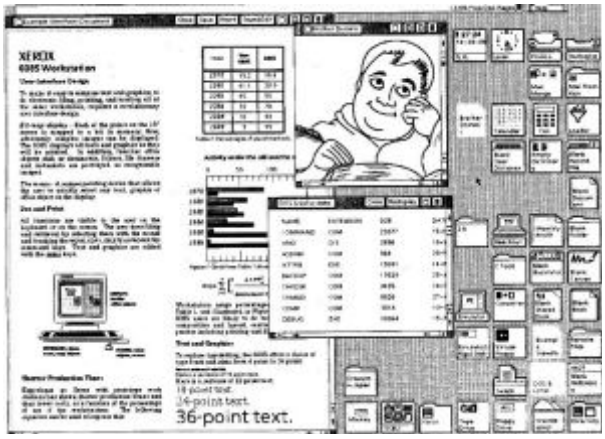


Figure 2.1. The GUI developed in 1981 for the Star Workstation

Today there are many operating systems that are GUI-centric. In spite of their differences, many controls are very similar and have the same job on a GUI. Microsoft Windows, UNIX and Linux, and Macintosh families are some well known examples for these operating systems. They all follow the ideas behind the PARC User Interface.

GUIs were needed because of the difficulties in learning command line interfaces (CLI). CLIs require commands to be typed on the keyboard. With CLIs, it is possible to efficiently and productively complete operations, but this requires a long time of discovering and learning the commands. On the other hand, GUIs present the user with

numerous controls that represent and can trigger some of the system's available commands. Thus, they eliminate the need of learning commands to perform operations.

A GUI uses a combination of technologies and devices to provide a platform the user can interact with, for the tasks of gathering and producing information. For example, window-icon-menu-pointing device (WIMP) style of interaction uses a physical input device to control the position of the cursor and presents information organized in windows and represented with icons. In personal computers (PC), a simulation called desktop environment in which the display represents a desktop upon which documents and folders can be placed. Smaller mobile devices such as personal digital assistants (PDA) and mp3 players use PC elements with different realism and functionality due to constraints in size and available input devices. In recent years, 3D interaction technologies are put through even though they are very premature. Figure 2.2 shows the project looking glass that is a 3D desktop environment of Sun Microsystems. Such a three-dimensional computing environment could be used for collaborative work for instance scientists could study three-dimensional models of molecules in a virtual reality environment, or engineers could work on assembling a three-dimensional model of an airplane.

Figure 2.2. A view of Project Looking Glass

Designing the visual composition and behavior of a GUI is an important part of software application programming. Its main goal is to enhance the efficiency and the

usability of a program. User-friendliness is necessary since it represents a visual language tailored to the tasks a program must perform. In addition, typically a user interacts with information by manipulating visual controls that allow interactions appropriate to the kind of data they hold.

In programs, user interfaces are started to be used extensively especially after windowing modules of operating systems have appeared. Figure 2.3 displays a screen shot of the utility called task manager existing in Microsoft Windows operating systems. There are also many advanced and professional user interfaces used for the purpose of software application development. Delphi for Object Pascal, Visual Studio for .Net and NetBeans for Java are a few well known examples of such computer programs that have sophisticated graphical user interfaces.

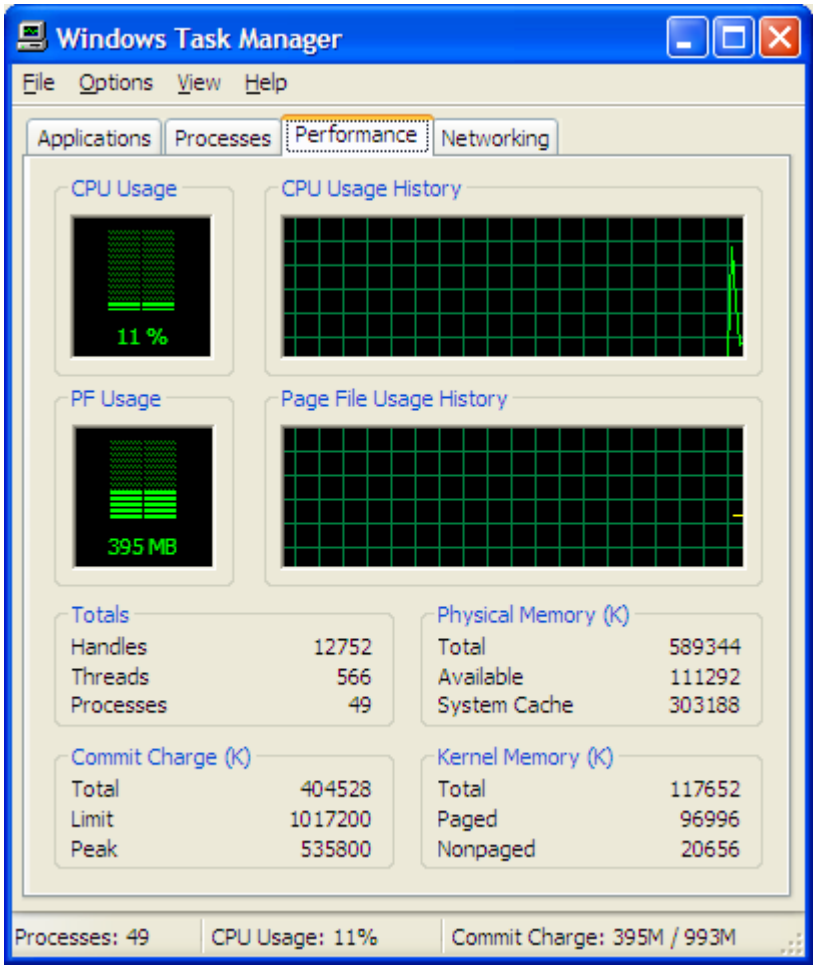


Figure 2.3. Windows Task Manager

The thesis program also uses many GUI components and controls that have become available with the standard version of Java 2 platform. The aim of the thesis was set as the coding of a development environment that will facilitate the programming of a web application that is based on the Springframework MVC. In computer industry, with the achievements in computer programming technology it is important to follow the standards and latest trends since it leads to a more acceptable result among users. Because of that, and to give the most comfortable solution to the users, writing the thesis program as a graphical user interface based application was vital and for that purpose Java Standard Edition libraries were benefited from.

2.2 Frameworks

Since the time when computer programming was started, it has been always an issue to facilitate writing of new software. For small applications like an algorithmic one it is not a very big issue but for bigger or enterprise applications it creates problems. In order to save time and efforts, programmers devised putting through programs that can be used for writing new programs. The main point of this is “code-reuse”. That means enabling reusing codes that are needed not only in one place.

At the start there were procedural languages, and only functions were reused. Later on, as new techniques (such as object oriented programming and aspect oriented programming) arrived it was possible to devise more sophisticated reuse patterns. It can be said that the break-through advancement was object oriented programming technique since it claims to code objects as in real life. The most known example to explain object oriented programming is about vehicles. Everyone understands a common idea when he thinks of an automobile. An automobile has a break pedal, a gas pedal, steering wheel, tires, headlights, signals etc. This is like a pattern. If one thinks of a certain model of an automobile trademark it is still an automobile but has its own different features. This model is an object that is obtained by using the automobile pattern. Likewise everything in the real life can be thought of this approach. Patterns and objects are the key concepts of object oriented programming. Patterns are named as classes in object oriented terminology. By using object oriented programming technique, it is much easier to facilitate programming new software.

When many of these reusable classes for a common target are collected together, they are given special names. Library and framework are two of them. A library is the simpler between these two. In libraries, many classes are put in packages according to their relativity based on what they do. Whatever a programmer needs for the new program, only this class or the total package from the library is imported into the new program. For example, when the coder needs to read files from the hard disk, the coder only puts directives to get the program include the necessary classes to read a file.

A framework has some sort of different features from a library. Outside of computer applications, a framework can be considered as the processes and technologies used to solve a complex issue. It is the skeleton upon which various objects are integrated for a given solution. In software development, however, a framework is a defined support structure in which another software project can be organized and developed.

A framework may include support programs, code libraries, a scripting language, or other software to help develop and get together the different components of a software project [1]. One meaning of this definition is that the programmer has to code the rest of the body of an application but the base classes, namely the skeleton. For example the framework provides a base class for database processes, and developers extend that class to make their database operations. The other mechanisms in the framework take that part of the application (a database class in this case) upon request (in a query for example) by the user of the program (the client) and make it work. This sounds like neither a class library nor a server-like program, but something like a mix of both. Because, it not only provides a library but also mechanisms to make the new application work. That make-it-work issue can be thought as the roads and traffic lights in regular life. The framework is that traffic infrastructure and the program is a vehicle in this structure. It has to follow the rules and travel on the roads. It can travel out of the way but this will not make the life easier, so, no need to do that unless there is another aim.

According to Pree [3], frameworks consist of frozen spots and hot spots. On the one hand, frozen spots define the overall architecture of a software system, that is its basic components and the relationships between them. These remain unchanged (frozen) in any instantiation of the application framework. On the other hand, hot spots represent those

parts where the programmers using the framework add their own code to add the functionality specific to their own project.

Frameworks define the places in the architecture where adaptations for specific functionality should be made - the hot spots. In an object-oriented environment, a framework consists of abstract and concrete classes. Instantiation of such a framework consists of composing and subclassing the existing classes [4]. While developing a concrete software system with a software framework, the hot spots are specialized according to the specific needs and requirements of the system. Software frameworks rely on the Hollywood Principle: "Don't call us, we'll call you." [5]. This means that the user-defined classes (for example, new subclasses), receive messages from the predefined framework classes. These are usually handled by implementing superclass abstract methods.

The most easily recognized advantage of using frameworks is that by bundling a large amount of reusable code into a framework, time is saved for the developer, since he gets rid of the task of rewriting large amounts of standard code for each new application that he develops. Frameworks are designed with the intention to facilitate software development, by allowing designers and programmers to spend more time on meeting software requirements rather than dealing with the more tedious low level details of providing a working system. For example, a team using Apache Struts Framework for the development of a banking web site can focus on how account withdrawals are going to work rather than how to control navigation between pages in a bug-free manner. However, there are common complaints about using frameworks. For example, using frameworks adds to "code bloat" and a result of competing and complementary frameworks is the time spent on learning how to use them.

Nonetheless having a good framework in place allows the developers to spend more time concentrating on the business-specific problem at hand rather than on the plumbing code behind it. In addition, a framework limits the choices during development, so it increases productivity, specifically in big and complex systems.

Today, there are many frameworks outside for different purposes and one of them is

the Springframework. The thesis is based on the subject of the Springframework. Thesis' aim is not writing of a program that does what the Springframework does. Instead, it aims to code a program that will facilitate using the Springframework which in turn eliminates one of the disadvantages of frameworks according to some critics.

2.3 Springframework

The Springframework is an open source application framework for the Java platform. The base of the Springframework was written by Rod Johnson in 2002 based on accepted best practices and available to all application types, not just web applications. It was simpler and more consistent than those developers and companies were used to at that time (Enterprise JavaBeans, Java Servlet API). Rapidly extended and the first release (1.0) was made in March 2004. Now it is in 2.5 version and releases for other programming platforms like .Net Platform were developed as well.

The Springframework is thought primarily as an alternative and replacement for the Enterprise JavaBean model. By design, the framework offers a lot of freedom to Java developers yet provides well-documented and easy to use solutions for common practices in the industry. One of the design goals of the Springframework is to easily integrate with existing Java 2 Enterprise Edition (J2EE) standards and vendor tools. To a large extent this removes the need to define its features in an official committee-controlled specification document which is criticized by some people.

The Springframework provides solutions to many technical challenges faced by Java developers and organizations planning to create applications based on the Java platform. Because of the sheer vastness of the functionality that is offered, it can be hard to distinguish the major building blocks the framework is composed of. The Springframework is not exclusively linked to the Java Enterprise platform although its far-reaching integration in this area is an important reason for its popularity.

The Springframework is probably best known for offering features required to effectively create complex business applications outside of the programming models that historically have been dominant in the industry. Next to that, it is also credited for

introducing previously unfamiliar functionalities into today's mainstream development practices, even beyond the Java platform. This amounts to a framework that offers a consistent model and makes it applicable to most application types that are created on top of the Java platform today. The Springframework is considered to implement one way of working based on best practices and industry standards and making it available to many domains in Java.

While the core features of the Springframework are usable in any Java application there are many extensions and improvements for building web-based applications on top of the Java Enterprise platform. The Springframework Spring has gained a lot of popularity because of this and is recognized by vendors as a strategically important framework.

The Springframework gains importance in the web development and it gets more attention day by day in the industry. A need to fulfill the requests coming from the web developers about an easier way to build Springframework-based applications have arisen. Seeing this necessity and Springframework's many abilities and superior capabilities, the Springframework was set as the thesis subject. Its MVC part is the main point of the thesis, for which the thesis program was developed.

2.3.1 Modules of the Springframework

The Springframework can be considered as a collection of smaller frameworks or frameworks-in-the-framework. Most of these frameworks are designed to work independently of each other yet provide better functionalities when used together. These frameworks are divided along the building blocks of typical complex applications:

- Inversion of Control container: configuration of application components and life cycle management of Java objects.
- Aspect-oriented programming framework: working with functionalities that cannot be implemented with Java's object-oriented programming capabilities without making sacrifices.
- Data access framework: working with relational database management systems on the Java platform using JDBC and *object-relational mapping* tools providing solutions to

technical challenges that are reusable in a multitude of Java-based environments.

- Transaction management framework: harmonization of various transaction management APIs and configurative transaction management orchestration for Java objects.
- Model-view-controller framework: HTTP and Servlet based framework providing many hooks for extension and customization.
- Remote Access framework: configurative RPC-style export and import of Java objects over computer networks supporting HTTP-based protocols, RMI, CORBA and web services (SOAP).
- Authentication and authorization framework: configurative orchestration of authentication and authorization processes supporting many popular and industry-standard standards, protocols, tools and practices via the Acegi security framework (Java) sub-project.
- Remote Management framework: configurative exposure and management of Java objects for local or remote configuration via JMX.
- Messaging framework: configurative registration of message listener objects for transparent message consumption from message queues via JMS, improvement of message sending over standard JMS APIs.
- Testing framework: support classes for writing unit tests and integration tests.

2.3.2 Springframework API Documentation

The API documentation of the Springframework is reachable through the Springframework's official web site www.springframework.org as the reference manual and has a version for every different release of the Springframework. For Java applications, API documentation is prepared via the tool javadoc.exe located in the bin folder of Java SDK. Using special syntax in front of an element like a method in the code, javadoc tool is made aware and the tool produces the documentation for the signed blocks.

2.3.3 Dependency Injection

Dependency injection or Inversion of Control is a programming architectural model. Dependency injection is a design pattern which can be described as “taking the

creation and initialization of objects from the classes and giving it to a controller system like a framework or container". The objects needed by other objects are created by a control system and injected into requester objects. The concept behind Inversion of Control (IoC) is often expressed in the Hollywood Principle [5]. IoC moves the responsibility for making things happen into the framework, and away from application code.

Dependency injection (DI) is also a way to achieve loose coupling. The technique results in highly testable objects, particularly when applying test-driven development using mock objects. Dependency injection removes explicit dependence on container APIs as well; ordinary Java methods are used to inject dependencies such as collaborating objects or configuration values into application object instances. Where configuration is concerned this means that while in traditional container architectures such as EJB, a component might call the container to say "where's object X, which I need to do my work", with DI, container figures out that the component needs an X object, and provides it to it at runtime. The container does this figuring out based on method signatures (usually JavaBean properties or constructors) and, possibly, configuration data such as XML.

There are two common forms of dependency injection: setter and constructor injections.

2.3.3.1 Constructor Injection. This dependency injection method is achieved using constructors of a class. The attributes of an object of a class are set using constructor parameters, which mean an object is initialized during its creation time. This kind of injection is used by some MVC frameworks such as PicoContainer. An example for constructor injection is illustrated in Figure 2.4.

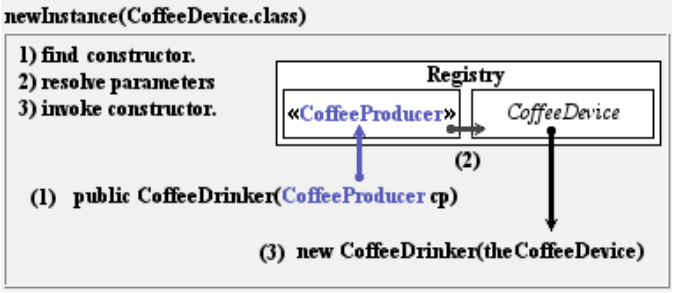


Figure 2.4. An example for constructor injection

Those three steps explain the way constructor injection works;

1. Find the constructor; the manager system will find the constructor of the object requested.
2. Resolve parameters; the container (manager system) has to know what the parameters to be sent to the constructor are.
3. Invoke the constructor.

2.3.3.2 Setter Injection. This injection method is somehow different from the constructor injection. In this injection method, the attributes of an object are set using the methods called setter after it is created. In some programming languages, this kind of attributes having setter methods is called property. This kind of injection is used by the Springframework.

2.4 Java Programming Language

Java is an object-oriented programming language developed by Sun Microsystems in the early 1990s. It is classified as a 3rd generation language after 2nd generation languages. As the trend of recent years, it is open source. Java language derives much of its syntax from C and C++ but has a simpler object model and fewer low-level facilities. In addition, it differs from other programming languages in the way it is compiled and run. Java source files are first compiled in an intermediate form called byte code. Later, at runtime, this byte code is compiled to native computer code or interpreted for execution. However, direct execution of byte code by a Java processor is also possible.

There were five primary goals in the creation of Java language; being object oriented, execution of a Java program on different operating systems without any additional configuration, having built-in support for using computer networks such as Internet, having a design that allows remote procedure calls securely, being easy to use. Java provides all of these functionalities and features nonetheless it suffered from poor performance in its first years. The feature that it can run on multiple operating systems led it to be popular in especially open source software industry. For small companies, it creates cost reductions.

Today, there are many programs and architectures developed using Java; a peer-to-peer (P2P) file sharing program Limewire, Sun's virtualization technology in Solaris 10 operating system, games in hand-held devices such as cell phones. The Springframework is also developed using Java language.

Since the Springframework is based on Java technology, for the thesis program, Java was selected as the development platform. This choice gave a wider work area on the Springframework as well. The existence of plenty of 3rd party libraries for Java, and documentations reachable through Internet provided a more comfortable environment when connected with the fact that Java is open source. In addition, since the projects developed using the thesis program are based on Java and the database chosen for them is also written using Java, the thesis study became a tidier one-technology development experience.

2.5 Extensible Markup Language

The Extensible Markup Language (XML) is a general-purpose, fee-free markup language. It is classified as an extensible language because it allows its users to define their own tags in an XML file. XML is a simplified subset of the Standard Generalized Markup Language (SGML), and is designed to be relatively human-legible.

XML's primary design purpose is to facilitate the sharing of data across different information systems, particularly through Internet by enabling generic SGML to be server, received and processed on the Web in the way that is now possible with HMTL. For example, XML web services have become very common recently. XML web services enable information provider methods to be opened to Internet environment. In addition, by adding semantic constraints, application languages can be implemented in XML.

The thesis program aims to facilitate writing of Springframework-based web application. Springframework's Model-View-Controller (MVC) sub-framework needs certain XML files to manage, configure and run such web applications. The thesis program has to change the content of those XML files according to XML syntax and in the way that the MVC sub framework understands. A file addition, for instance, may affect the content of more than one XML file and may require them to be changed.

2.6 Plain Old Java Objects

POJO is an acronym for Plain Old Java Objects. The name emphasizes that the Java object in question is an ordinary one, not a special object, and in particular not an Enterprise JavaBean (EJB). An EJB is a managed, server-side component for modular construction of enterprise applications. All Java objects are POJO therefore ideally speaking a POJO is a Java object without any runtime dependencies other than to Java Standard Edition classes provided by Sun Microsystems. This feature for Java objects is favored by the idea that “the simpler design the better”. The name POJO thus reminds that simpler designs can be better, rather than incorporating a complicated framework in the architecture of a system without a sufficient reason.

As designs using POJOs become more commonly used, systems that give POJOs some of the functionality used in frameworks have arisen and more choices about which areas of functionality are actually needed have started to appear. Hibernate and the Springframework are two examples of such systems which apply the simplicity of POJOs. These frameworks use POJOs extensively in their implementations for their own purposes. While Hibernate uses them for its object-relational mapping goal, the Springframework uses them in order to make its architectural development infrastructure work.

The Springframework is the framework on which the projects developed using the thesis program are built. The MVC framework in the Springframework uses a standardized version of POJOs called beans in Java terminology. The MVC framework is the actual main focus point of the thesis program. The thesis program must prepare the files for the MVC framework to be able to configure a project.

2.7 Beans Concept

Beans or JavaBeans are classes written in the Java programming language. The specification by Sun Microsystems defines them as "reusable software components that can be manipulated visually in a builder tool". They are used to encapsulate many objects into a single object (the bean), so that the bean can be passed around rather than the individual objects.

In order to function as a JavaBean, an object must obey certain conventions about method naming, construction and behavior. These conventions make it possible to have tools that can use, reuse, replace and connect JavaBeans. These conventions are;

1. The class must have a no-argument public constructor
2. Its properties must be accessible using get, set and other methods (accessor methods) following a standard naming convention. The properties are private and the set and get methods are public. The properties have camel-case and their names' first letters are in small case, while the get and set methods have the names with get and set words followed by the name of the property with camel-case and first letter capitalized
3. The class should be serializable (able to persistently save and restore its state)
4. It should not contain any required event-handling methods

Because these are largely expressed as conventions rather than by implementing interfaces, some developers view JavaBeans as POJOs that follow certain naming conventions.

Since these conventions lead to standardization without a loss in functionality, today, many technologies choose to apply JavaBeans rather than bare POJOs. In that way, a tidier and more understandable structure appears. One of the technologies that use beans is Dependency Injection (DI) that the Springframework includes. The Springframework and DI form one the differentiating features of the Springframework, and it is one of the key technologies behind the Springframework.

The DI and The MVC in the Springframework are cooperated to form a working model. Some XML files required by the MVC in order to configure and manage a web application consist of bean definitions in a way that these definitions can be given to each other. The MVC redirects the requests between the layers and according to these definitions, and the DI creates the requested objects (beans) at runtime and gives them to the requester objects as defined in XML files. This is a fine work and the files must be prepared very carefully in order not to raise an error that can make the web application stop working. The thesis program takes care of the preparation of these files and automatically generates many files and the definitions of these files in the relative XML

files.

2.8 Model View Controller

Model-view-controller (MVC) is an architectural pattern used in software engineering. In complex computer applications, the task of fulfilling user requirements must be divided into smaller and more manageable parts. In MVC approach, a clean separation of business logic, data and presentation logic is achieved. This separation gives flexibility as well as manageability and easier development and evolvement.

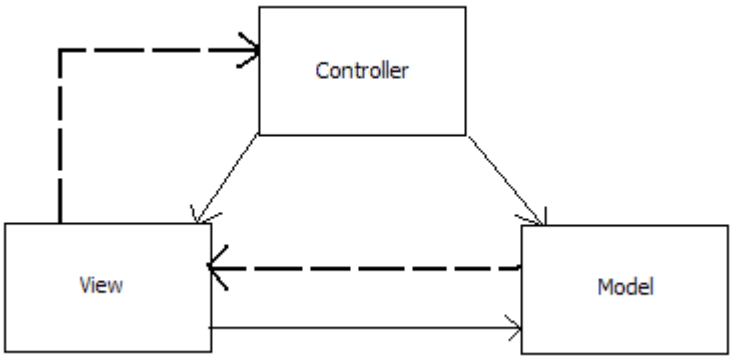


Figure 2.5. A simple diagram depicting the relationship between the Model, View and Controller

Data layer is called model, user interface is view and the layer between these two is control layer. This is not a new approach. It is used in desktop programs and certain application programming interfaces (APIs) such as Swing API of Java. The routes of a request between these layers can be demonstrated as in Figure 2.5. The solid lines indicate a direct association, and the dashed lines indicate an indirect association.

2.8.1 Models

This is the domain-specific representation of the information on which the application operates. In the MVC architecture, the Model represents application data and the business rules that govern access and modification of this data [6]. Today, many applications use a persistent storage mechanism (such as a database) to store data. MVC does not specifically mention about the data access layer, because it is understood to be

underneath or encapsulated by the model. The model put through two functions; informing views about changes of itself and enabling controllers to access application functionality encapsulated by it.

2.8.2 Views

A view element renders the model into a form suitable for interaction, typically a user interface element. Views take the model from the controller and present it to the user according to the logic. In addition, views make some controls on the data sent via a form submission. In web environments, views are programmed by using client side markup languages and scripts like XHTML and VBScript. It is very important to give the same user experience in any web manipulation tool which is usually a web browser. In recent years, new technologies such as AJAX facilitate and eliminate the need for huge data transmissions, making it more close to desktop application logic.

2.8.3 Controllers

Controllers process and respond to events which are typically user actions received through views and may invoke changes on the model. The changes that can be applied to the model come from views. Usually a form and a button to send the data are used to collect data from users. The request comes to the controller class that is responsible for managing that view and by instantiating a new model object the controller calls usually an SQL script to mirror the changes or insert new data. As can be understood, the view and model thus have no information about each other. The bindings are handled by the controller. In Figure 2.6, the logic behind the MVC approach is summarized.

Many times the MVC is implemented by frameworks. One of those implementers is the Springframework. The Springframework has its own MVC implementation but as Springframework's parts are not dependent on each other strictly, it is possible to use other MVC implementations such as Struts and WebWork. The main idea of the thesis is facilitating Springframework MVC-based application development.

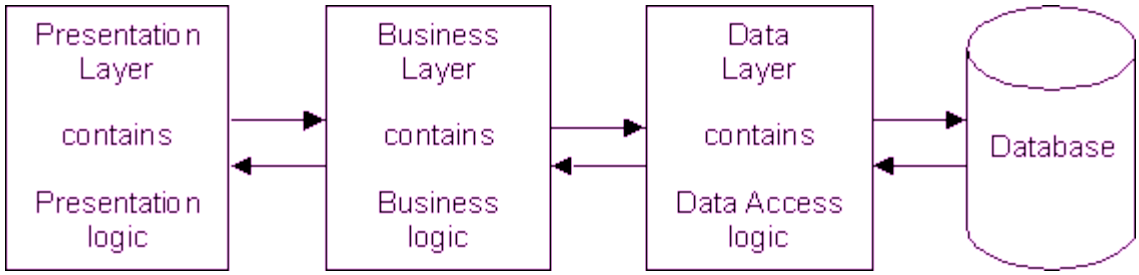


Figure 2.6. The 3-tier structure [7]

2.9 Unified Modeling Language

Unified Modeling Language (UML) is a single standardized model for designing systems. It was accepted by the Object Management Group (OMG) as the standard for modeling object oriented programs. However, it might be used not only for object oriented programming design but also for any kind of systems such as a production line or a machine such as a humanoid robot. It is designed such that it can be used to model every system. Therefore, it is simple and easy to use. UML defines nine types of diagrams to design a system; class (package), object, use case, sequence, collaboration, state chart, activity, component and deployment.

For any system, there exists a design phase. Software projects are not exceptions. Latest trends in software development are following a professional approach in program coding. Especially big magnitude applications require a very carefully planned approach. Standards have paved a long way and the software development process has advanced a lot since it started. UML is a very important part of developing object-oriented software and the software development process [8].

The main aim of the thesis was to write a software program. Without applying the UML phase, it would be harder or impossible to finish the program. Because of that reasons, the UML was used during the development process and the class diagrams for the classes in the thesis application were put through. The software development process is not a process that the steps are completely separated from each other. In every step it is possible to return to the prior steps and change something. Because of that, any class diagrams were not the final and did not have the last shape during the development of the thesis program, which mean the first class diagrams have changed many times. The

resultant class diagrams can be found in Appendix B.

For UML modeling, NetBeans Integrated Development Environment (IDE) was used. NetBeans gives its uses ability to create UML projects beside many kinds of Java projects. A UML project in NetBeans gives the means to form UML diagrams that were defined in UML 2.0 standards, which means NetBeans UML module is fully compatible with the current UML standards. With NetBeans UML Modeling, 8 UML diagrams can be created: activity, class, collaboration, component, deployment, sequence, state-chart and use case diagrams.

One of the most interesting features of NetBeans UML projects is the ability to both forward and reverse engineer. Forward engineering is the normal way in a development process whereas reverse engineering means performing the process in the reserve order, that is, first the program is written and after that, NetBeans can form the UML diagrams from the program source code.

After modeling classes in NetBeans, the process of realizing those classes came onto the scene. UML class diagrams only provide the signatures of classes and methods. Because of that, giving recalling names to them is a vital action during UML modeling. In addition, to make a good programming practice, classes should be implemented in a way providing code reuse as much as possible.

2.9.1 Class Diagrams

Class diagrams identify the class structure of a system, including the properties and methods of each class. Besides, depicted are the various relationships that can exist between classes, such as an inheritance relationship. Class diagrams are one of the most widely used diagrams from the UML specification. Part of the popularity of class diagrams stems from the fact that many Computer-Aided Software Engineering (CASE) tools, such as Rational XDE, will auto-generate code in a variety of languages, including Java, C++, and C#, from these models.

In Figure 2.7 is a sample class diagram. Every box represents a class and lines connecting those boxes represent relationships between those classes. In Figure 2.7, black

filled ended line is an aggregation, empty ended lines are generalizations. The smooth line shows an association. Numbers and asterisk over lines show multiplicity relationships between classes.

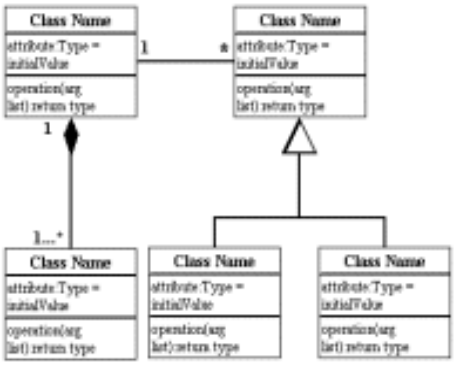


Figure 2.7. A sample class diagram

2.10 HSQL Database Engine

HSQL is a database system written totally in Java programming language. It is distributed as a Java archive (jar) file according to its license and it is an open source system which means its code can be manipulated by anyone who needs additional features or wants to make contribution to the HSQL project. HSQL meets and supports a rich subset of ANSI-92 SQL plus SQL 99 and 2003 enhancements. The SQL-92 standards are supported by a rate of 90% and this is a reason for its common usage in the software industry especially for prerelease purposes. OpenOffice.org 2.0 distribution which is the most widely used open-source office suite software today and has versions for operating systems such as Linux and Unix clones besides Windows chose HSQL for its Microsoft's Access-like database component's background engine.

HSQL has 3 different working modes each of which has its advantages for the purpose of its selection;

2.10.1 Server Mode

The highest management and accessibility are provided via server modes. The data is persistent and secure as much as the computer provides. The database engine runs

in a Java Virtual Machine (JVM) and listens for connections from programs on the same computer or other computers on the network. Several different programs can connect to the server and retrieve or update the information. Clients can connect to the server using the HSQLDB JDBC driver. The maximum database count is set to 10 in most cases.

2.10.2 In-Process (Standalone) Mode

In this mode, the database engine runs as part of an application in the same Java Virtual Machine. For most applications this mode can be faster, as the data is not converted and sent over the network. The main drawback is that it is not possible by default to connect to the database from outside an application [9]. In that mode, the external tools can not reach the database and check the data there. In version 1.8.0 of HSQL, it is possible to run a server instance in a thread from the same JVM and this drawback is eliminated.

2.10.3 Memory-Only Mode

It is possible to run HSQLDB only in the random access memory of a computer. Thus no information is written to disk. This mode can be preferred where internal processing of data is needed.

For any web application, it is important to record information such as registration or settings information. A web application's pages provide access to the database where that information is stored in a secure manner. By adding database support as a feature for web applications written in the thesis application, a comfortable playground was achieved. In addition, that HSQL is Java based added a tidier infrastructure for development phase.

3. LITERATURE OVERVIEW

The Springframework is not a very new object in web development. But considering other technologies in use, it can be said that it is relatively a new one. However, in an environment that appreciates and responses good movements rapidly it was expected that more resources were present. Especially the lack of the studies in academic level about the Springframework was disappointing. This is not a lack for only the Springframework but also for all family of web development frameworks. Therefore, this thesis' importance in the field of web development gets more obvious.

This section summarizes the articles and technical papers benefited from during the thesis analysis and development.

Since the first version of Johnson's [10] article that was published in October, 2003, the Springframework has steadily grown in popularity. It has progressed through version 1.0 final to the present 2.5, and has been adopted in a wide range of industries and projects. Nonetheless its oldness this article protects its applicability and power. Because it was written by the prime mover and developer of the Springframework. In this article, Johnson explains what Springframework set out to achieve, and how he believed it could help software coders to develop J2EE applications.

This article was the entry point for those who want to make any kind of study about the Springframework. It discusses everything that a new Spring developer or a decision maker will benefit from. The difference between the Springframework and others is an important factor on choosing it among others and this article defines its superiorities and counts its advantages and architectural benefits in brief detail. According to the article Springframework's aim is “Spring is essentially a technology dedicated to enabling you to build applications using POJOs. This desirable goal requires a sophisticated framework, which conceals much complexity from the developer”. What the Springframework does and how it achieves that are must-be-known topics after the goal. It provides enough code examples to make the issues digested well like inversion of control, JDBC (Java Database Connectivity) abstraction and data access exception hierarchy, object - relational mapping

integration and MVC. For a thorough understanding this article should have been revised.

Risberg's [11] article is a step-by-step account of how to develop a web application from scratch using the Spring Framework. It is divided into a number of parts. They can be read in sequence or skipped one if the reader is already familiar with the concepts covered in it. This article is related to what we aimed in the meaning that what we wanted to achieve must be known well. This article paves the steps one by one and provides an understandable way to those who do not anyhow know about using the Springframework. The article is divided into four chapters and each chapter is about using one other base structural important trivet. It adds a new feature in every chapter and develops a simple web site. The article is fictioned for an environment of the simplest form. It does not use any editor or development tool. Classes are compiled and deployed to the server using Ant build tool and its script is formed step by step as the program is written. Thus this article is a very helpful document for those who are unfamiliar with the basic development methods of a Java application as well. The understanding of the behind the scenes structure is helpful even while using an integrated development environment and many integrated development environments actually use such tools without making the developer know about it.

Arthur's and Azadegan's [12] paper briefly describes the Springframework's underlying architecture and presents a case study using Spring. It teaches that the Springframework increases productivity by decreasing complexity. Facilitation capabilities of application development using the Springframework were learned from this article which had been answers to many infrastructural questions lying behind Springframework's success. Inversion of control which provides many features via XML files to a Springframework based application is summarized here. How the Springframework could be superior over EJB technology is a very important point described here. In addition, Springframework's MVC layer implementation is studied in the document which is actually one of the main subjects of the thesis.

In developing the program, it was not aimed a limiting magnitude for an application that could be implemented via our tool. For that reason, it was important to learn about what might be done using SIDE. The question what if an enterprise application is set out to

be implemented on SIDE architecture was meaningful for us. The partner study of Oktay, Gülbağcı and Sarıöz [13] describes issues and handicaps faced while developing an enterprise scale application, thus it served our needs. The architectural issues section of the paper was the most helpful part while trying to find answers to the questions in hand.

Making a choice among frameworks in use to develop a web application on one or to make a study about one requires careful and in-depth research before starting working. Denoncourt's [14] article that compares framework to some extent so that it was helpful in the research. It summarizes the experiences gained. In this article we saw that Springframework has many advantages over its rivals in the same domain.

4. DESIGN

Computer engineering industry is an evolving sector. There is always a great deal of circulation of technologies and human resources. Working with either accepted and widely used technologies and standards or rarely used and new ones are possible. A newly defined technology can be popular in a short time or may diminish without any acceptance. The Springframework is one of those which succeeded to widen its usage with its capabilities and evolving nature. Therefore, its flexibility and broad usage are the key factors for selecting the Springframework as the thesis subject.

The power of the Springframework has protected its value in the industry since its first release. Nonetheless those achievements and successes, only a few examples to facilitate the Springframework based application development appeared. In addition, none of these solutions were standalone. They worked under a huge integrated development environment like Eclipse and Netbeans, and sometimes provided very small facilitations to it for instance only forming a base file structure. Figure 4.1 shows the complex user interface of Netbeans IDE while editing a file.

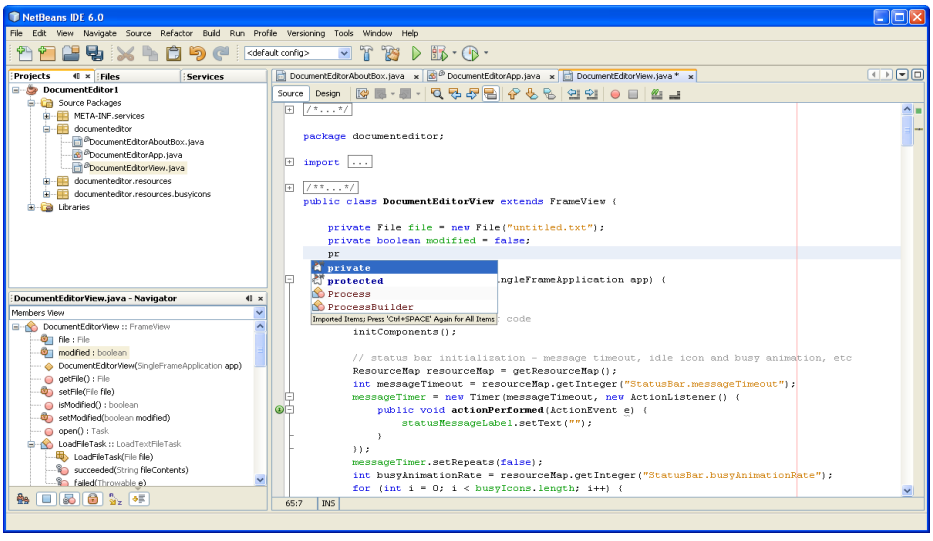


Figure 4.1. A view of NetBeans IDE

All those meant that the thesis program would be not only an open source and an academic level solution but also probably the first example of standalone solutions for the

Springframework.

4.1 General Overview

The resultant program is a development environment that is managed and used totally in a graphical interaction way. Everything that the development environment provides to the user is performed by interacting with the graphical user interface using mouse movements and actions. File edition, project creation and configuration of Java path are a few examples. The development environment stores and presents the files in a project to the user in a way that is comfortable for the user. A different folder structure from the one that is present in runtime is kept because of that purpose. This is a mean to provide an easier way of control for a project to be managed and developed by the user in the development environment.

Configuration of the development environment is easy and simple. The development environment requests only three configuration properties which can be set using the related graphical tool.

The program is designed to build up a Springframework MVC based web application from scratch. A project that is created using the program is capable of running on a Java web server from the creation time. There is no need to perform any extra configuration to make it runnable. The project is also given the ability of internationalization which can be further configured during the development of it. The project has language support for Turkish and English at the creation time.

Another feature that is given to the project is a welcome page. index.htm is the browser calling address for the welcome page. That page shows a welcome message in English which can be changed to Turkish during the usage. In addition, an error page that is directed to in case of an error and shows the error message is put into the project.

In addition to those features, many library files are given in advance to the project so that there will be no need of extra search and effort to add these library files. Some important library files are log4j-1.2.14.jar for logging support, hibernate3.jar for database

mappings, jstl.jar for tag library support. All the files are viewable through the tree component on the left side of the user interface. They are represented with their names and with a special icon for its type. These different icons give a classification to the user while searching for an item on the tree component.

The development environment enables a user to make all the necessary operations through its own user interface. Any file that is needed to be edited can be opened by double clicking on its representation on the tree component on the left side of the user interface. The same icon that is used on the tree component is used on the left upper side of the window opened for the file in hand.

A file is opened in the desktop panel and its text is highlighted according to its type. XML, Java, properties, Jsp and CSS files are highlighted differently. This gives the user better understanding and reading ability for the file currently in editing. A file currently edited can be printed, edited, saved or content cleared. Also on the text of the file, using certain menu operations, search, copy, cut, paste, replace and select all operations can be performed. In addition to these, for a Jsp file, HTML tag insertions are possible.

Many files may be opened at the same time and all are stored on the desktop panel on the right side of the user interface. A file that is already open is not opened again in a following request. Instead, the representation of the file on the desktop panel is gained focus. While editing a file all other files are in the background. They can be minimized not to disturb the view. Also every file has the ability to be maximized to give a larger working area to the user.

All files but vital ones for a project to be able to run can be deleted as well. The vital files are applicationContext.xml, dispatcher-servlet.xml, web.xml and context.xml. For the Springframework to configure and manage an application, they are the files that must exist.

Among all file types in a project, Java files have special meanings. Because, compared to other files in a project they have different features. One of them is their absolute necessity of compilation. The development environment groups Java files in 3

categories. That grouping is for better and easier management. These groups are beans, controllers and entities. Beans are bare Java classes that are created for a certain goal such as for a mathematical operation like addition or subtraction. Controllers are different from beans in their responsibilities and their meaning for the Springframework. They are a vital part of a Springframework MVC application. They are responsible for controlling a Jsp page and returning necessary and demanded information and behaviors. Because of those, they are separated from other Java files in a project. The last group is entities. Entities are for database operations. They are the model part of MVC and are necessary to store and transfer data between MVC layers. Because of their responsibilities they are treated differently by the development environment.

This differentiation among Java files in a project leads to give the user the ability to add those different Java files in different ways. In that way, the user realizes that distinction according to the development environment's view.

An ability that the development environment gives to the user is adding database support automatically in case of demand. HSQL database engine is the selected product for that purpose. A database added to a project can be added tables after that. To add tables to a database, it is a must for a project to have at least one entity file. This entity file is used to create the table in the database. Thus data access object implementation is achieved without a loss in the chain of that implementation pattern. Java objects are used to perform data manipulations.

A project that is wanted to be run on the web server must be compiled in case it has any Java files and deployed. If there is any error or warning returned from the selected Java compiler, it is shown on a text box in a window. If there is no selected Java compiler or Tomcat web server, the development environment will give error in the related operation.

4.2 Graphical Overview

The user interface is separated into 4 parts. These parts are main menu, project tree, tool bar and the desktop panel. Figure 4.2 show the graphical user interface with all parts.

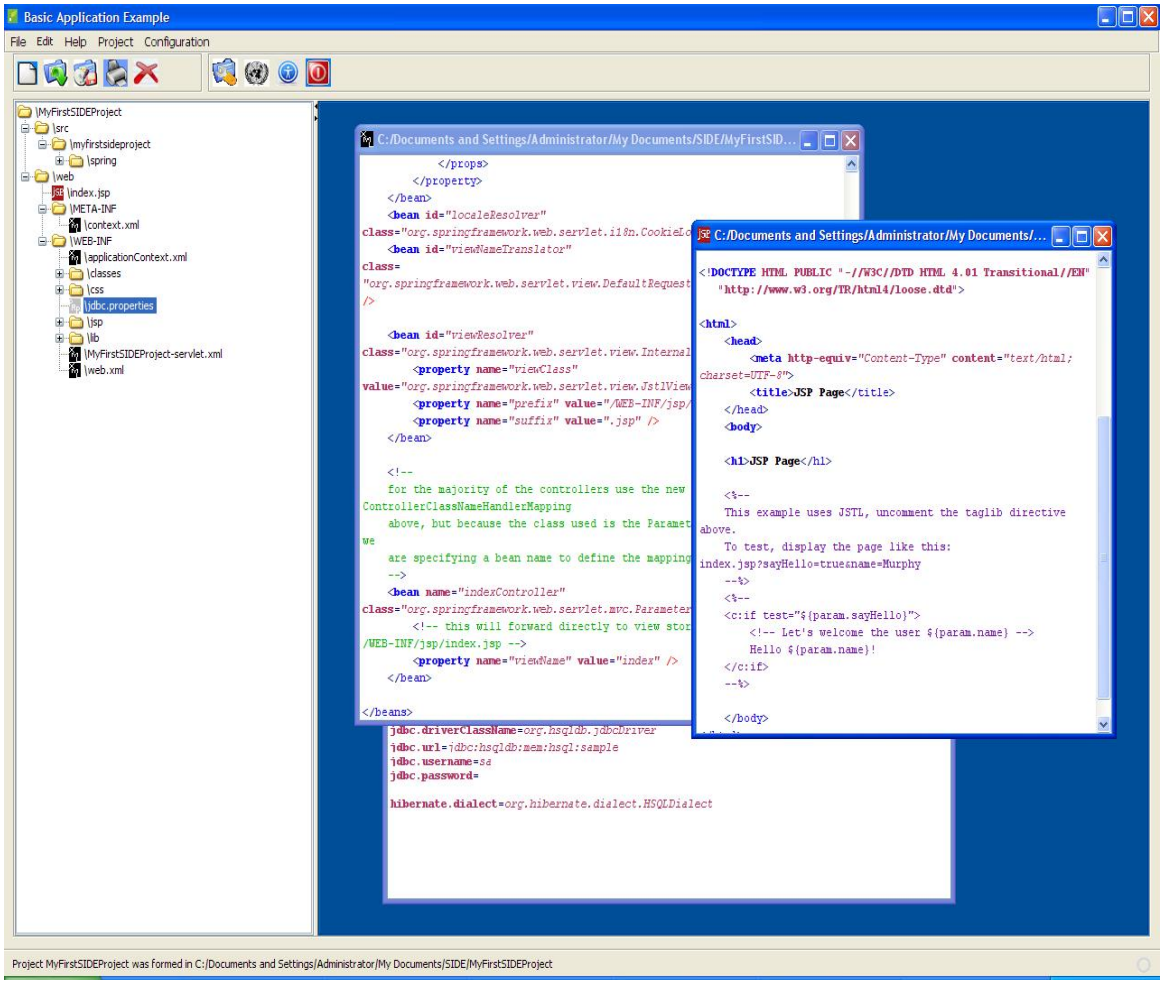


Figure 4.2. The graphical user interface of the thesis program

4.2.1 Main Menu

Main menu is located on the top of the graphical user interface. Almost all the operations can be performed using the menu items in that menu. In addition, many operations may only be performed using this menu.

The first sub menu is *File*. A project is created, opened, closed using the buttons under the file menu. As can be seen in Figure 4.3, there are also exit, save and recent projects items. Save button will save a file currently being edited and exit will close the currently opened project and quit the thesis program. Recent projects item hides recently opened 3 projects according to their load time.

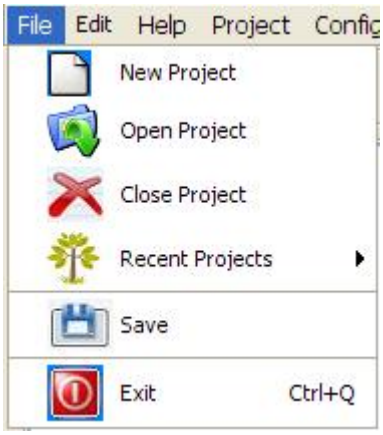


Figure 4.3. The file menu in open mode

Second menu in the order is edit menu which includes operations to perform actions on a file currently in edit mode. These actions are the actions that are used to change the text content of a file. In Figure 4.4, the edit menu is shown as the buttons on it are visible. After edit menu is help menu which only includes the button to show the about dialog that displays the information about the thesis program. Help menu is displayed in Figure 4.5.

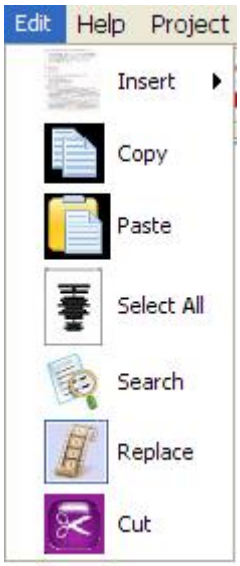


Figure 4.4. The edit menu in open mode

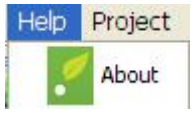


Figure 4.5. The help menu in open mode

Project menu houses the actions which are very important steps in the development of a project developed in the thesis program. Compilation and deployment are two of them. Compilation action compiles all the Java files in a project and deployment action deploys the files to the selected Tomcat web server in a directory structure that is demanded by the Tomcat and the Springframework. This file structure is different from the one that is present during the development phase in the thesis program. Other tasks housed by the project menu are for managing controller Java files, internationalization properties files and style sheet CSS files in a project. Besides, an action gives the user ability to add an external file to the project. The file is added into the folder according to its type. For instance, a CSS file is directly copied to the css folder. Database and database table creations are performed via the buttons in this menu as well. Figure 4.6 shows the project menu as it is open.

Figure 4.6. The project menu in open mode

Last menu is configuration menu. There are three actions in that menu. These actions are used to configure the development environment so that it can perform the demanded tasks by the user. Java compiler's path, Tomcat web server's home directory and the directory where the projects of the user are stored are set using the actions in the configuration menu. In Figure 4.7, the order of the buttons that control these actions can be seen.

Figure 4.7. The configuration menu in open mode

4.2.2 Project Tree

Project tree is the place where the files in a project that is currently evolved by the user in the development environment are presented in a friendly manner. It is placed on the left side of the user interface and may be made invisible using the small triangle on its right edge. Project tree is a javax.swing.JTree component. A JTree has the ability to be configured according to the specific needs. In the thesis program, it is used to display the file structure of a project. This file structure is rendered in the project tree according to the file types. As can be seen in Figure 4.8, all file types have different icons. That way provides easier management of the files to the user. Figure 4.8 also shows how the file structure of a project that is not edited yet seems. Every file added to the project is also added to the project tree.

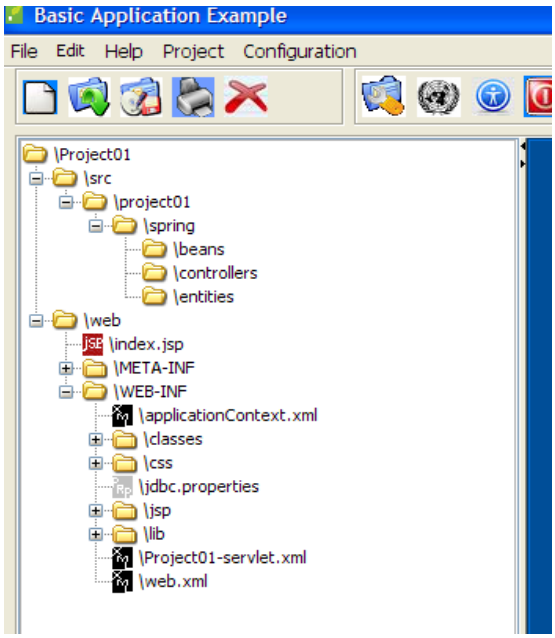


Figure 4.8 Project tree's look of a project that is created newly

Figure 4.9 shows the context menu of the project tree. A context menu is a pop up menu that is made visible with the operation of a right mouse click. This menu houses the actions necessary to add new files to a project besides to open and delete the files currently in the project. A file must be selected with the left mouse button click in order to open or delete it. Else, there will be a warning message and a beep sound signal indicating that there is no file selected currently. Other capability given to the user with this menu is reloading the project tree file structure to refresh its look.

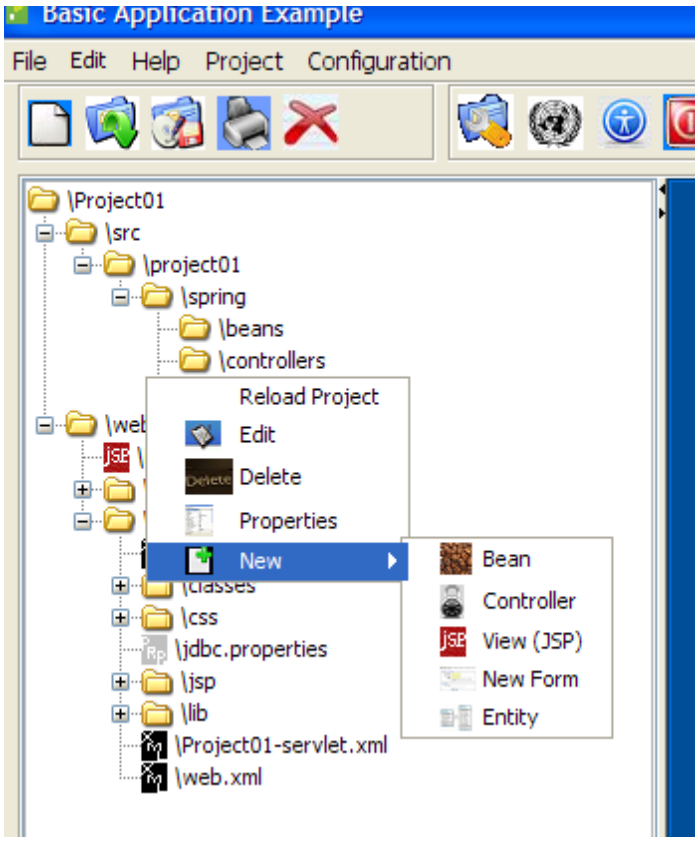


Figure 4.9 Project tree's context menu

4.2.3 Tool bar

The tool bar is a javax.swing.JToolbar component and is placed under the main menu. All the buttons on the tool bar are shown without any text. Instead, they are displayed with an image that indicates their meanings and a tool tip that get visible when pointed on one of them for 2 seconds. The tool bar has nine buttons for the nine actions it stores.

The tool bar of the user interface of the thesis program keeps two types of actions and they are grouped accordingly. That grouping can be seen in Figure 4.10. The left group keeps file edition, and project creation and opening actions whereas the right group holds the managerial actions for the thesis program and the projects.

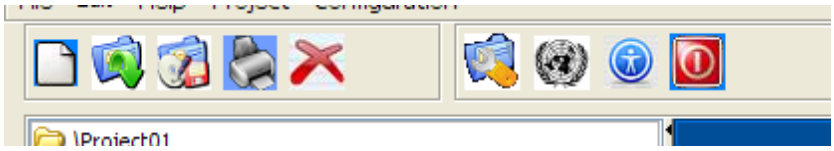


Figure 4.10 The tool bar

Using the actions on the tool bar a user can open a project, save an edited file, print an open file, clear the content of an open file, manage the internationalization files in the currently loaded project, configure the settings of the thesis program and exit the thesis program.

4.2.4 Desktop Panel

The desktop panel is where the files in a project are in open mode and ready to be edited. It stands on the right down side of the user interface and covers the largest surface area of it. It has a blue background in order to give a user-friendly aura.

The desktop panel is a javax.swing.JDesktopPane component and it has the ability to house windows. This makes it a suitable place for storing edition windows inside of it. Any file that is double-clicked on the project tree is opened inside a window here. Minimization, maximization and restoration of a maximized or minimized window are possible options for the windows in the desktop panel. Many files may be opened at the same time and a file is symbolized with an icon according to its type. Figure 4.11 shows the different file types residing in the desktop panel at the same time. As can be seen they have different icons on the upper left corner and have different highlighting. Counting from the left, they are a Java file, a Jsp file, a properties file and an XML file.

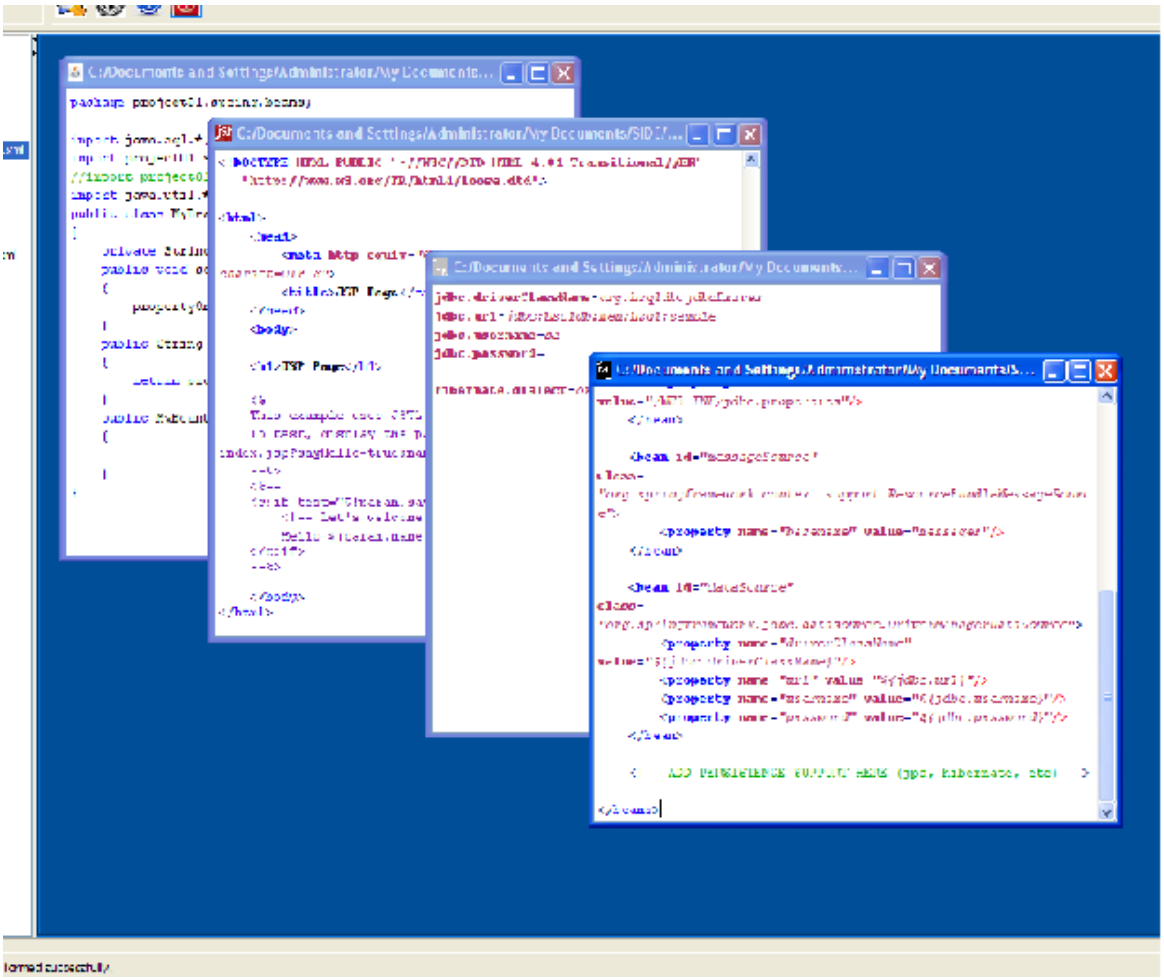


Figure 4.11 The desktop panel

4.3 Package Overview

Below the packages in the thesis project and important classes in those packages are summarized briefly.

4.3.1. Package side

side is the name of the package where many user interfaces including the main window GUI are in. When the *jar* file of the program is double-clicked the java environment looks into the file MANIFEST.MF file in META-INF folder and if there is main-class attribute it executes that main class in the jar archive. In SIDE project, side.SideView.java is the main class called in that manner when SIDE.java is double-clicked.

4.3.1.1 SideView. This is the main graphical user interface. All the operations are performed using this graphical user interface's services presented to the user and other wizards and dialogs are reached via this user interface. Main menu on the top, a JTree component where the current project files are loaded, pop up menu in the JTree, JDesktopPane where text based editing of files are done on the right and a status bar where messages related to the operations he performed on the right are the main components of SideView main screen. Main operations of SideView including their main menu names are as follows;

Table 4.1. File Menu functions

Function #	Function Definition
1	Creating a new SIDE project
2	Opening an existing SIDE project
3	Closing the currently loaded project
4	Exiting SIDE
5	Saving the current file
6	Opening recent project

Table 4.2. Edit Menu functions

Function #	Function Definition
1	Insert any HTML tag if the current file is a Jsp file
2	Copying the selected text
3	Pasting the text in the clipboard
4	Selecting all the text in the currently opened file
5	Searching a text
6	Replacing the selected text with the text in the clipboard
7	Cutting the selected text in the currently opened file

Table 4.3. Configuration Menu functions

Function #	Function Definition
1	Configuring Java compiler home directory
2	Configuring Tomcat Application Server home directory
3	Configuring where SIDE projects are stored

Table 4.4. Help Menu functions

Function #	Function Definition
1	About box

Table 4.5. Project Menu functions

Function #	Function Definition
1	Compile the currently opened project
2	Deploy the currently opened project's files
3	Binding controllers and Jsp pages automatically
4	Managing internationalization files
5	Adding a Css file
6	Importing any external file according to its extension
7	Adding a database
8	Adding a table to a database
9	Adding a bean file
10	Adding a controller file
11	Adding an entity file
12	Adding a Jsp file
13	Adding a form into a Jsp file

Table 4.6. 1st Toolbar functions

Function #	Function Definition
1	Creating a new SIDE project
2	Opening an existing SIDE project
3	Saving the currently file
4	Printing the current file
5	Clearing the content of the current file

Table 4.7. 2nd Toolbar functions

Function #	Function Definition
1	Configurations
2	Adding a language support
3	Exiting the IDE

Table 4.8. Project JTree Pop Up Menu functions

Function #	Function Definition
1	On double-click open the clicked file
2	Adding a bean file
3	Adding a controller file
4	Adding an entity file
5	Adding a Jsp file
6	Adding a form into a Jsp file
7	Delete the selected file
8	Open the selected file

Table 4.9. Project JTree functions

Function #	Function Definition
1	On double-click open the clicked file

4.3.1.2 NewEntityClass. This wizard helps to create an entity file and add it to the currently loaded project. Only the names and the types of the properties and the name of the entity class are given and the related code is generated automatically.

4.3.1.3 NewFormDialog. This wizard helps to create a form page and its underlying controller and validator classes. In addition, into the dispatcher servlet, it adds necessary code to bind these classes and the Jsp page with the entity class that is going to back the form. It realizes user choices about HTML controls in the page and entity class' properties validations in the validator.

4.3.1.4 DatabaseTableAndManipulatorBeanFormer. This wizard helps to create search and edit pages for database tables. The reference object to use in that operation is the entity class which is the counterpart of the database table in question. This wizard's search page has the ability to delete and find requested records and edit page has the ability to update the record.

4.3.1.5 SpringEditControllerFormer**.** This wizard heps to add a controller with the selected type to the current project. There are 13 different controller types in Springframework web mvc. It gives information about the selected controller type to the user. Three of them

(MultiActionController, SimpleFormController and AbstractController) are the implemented and mostly used ones.

4.3.1.6 ManageLanguages. This wizard helps to manage currently supported languages in the project. It is possible to delete a language, and open a language file to edit by using this wizard.

4.3.2. Package side.utils

All infrastructural classes are stored in the package side.utils. Utils is a short for utility. A utility class is a class that defines a set of methods that perform common, often re-used functions. Most utility classes define these common methods under static scope. Examples of utility classes include java.util.Collections which provides several utility methods such as sorting on objects that implement a java.util.collection.Collection.

4.3.2.1 Project. This class is one of the most important classes in SIDE's side.utils package. This class is capable of holding a SIDE project object. It defines the methods in order all the way to manipulate and manage a project. For that reason in future developments for SIDE and additions to it should include this class in its implementations. Project class has two constructors, one takes no arguments and the other one takes a String storing the absolute project path in the hard drive. If no-argument constructor is called, to complete the initialization phase, using setter methods the necessary basic fields must be set. Setter and getter methods in the class provide security and easier maintainability with the logic of beans. In Java contrary to C#, Delphi and many other contemporary languages there are no properties built-in feature. Therefore this is provided via using private fields and public setter and getter methods. I have kept to this convention during the implementation of almost all classes.

4.3.2.2 GeneralFile. Another important class in side.utils package is GeneralFile. This class is the base class of all the file management classes in SIDE. A GeneralFile instance is responsible for the creation, deletion, copying, reading, writing and proving information about the file it handles. On top of these methods, a sub class for example JavaFile.java adds new functionalities according to its needs. For instance a JavaFile object will need to

insert a Method object to it when demanded or will give information about its package.

4.3.2.3 SimpleFormControllerFile. This class is very important in writing a Springframework-based project when using SIDE for that process. Because it manages a class file which extends org.springframework.web.servlet. mvc.SimpleFormController class. A SimpleFormController handles requests coming from a user form submission. According to the receiving information and object it performs developer's needs. onSubmit method is the key procedure in this meaning. Other important controller implementation classes are MultiActionControllerFile, AbstractControllerFile, ControllerFile (the base class of all controller files) and AbstractCommandControllerFile since these are important for Springframework. All controller files are responsible for creating dispatcher servlet presentation for them.

4.3.2.4 EntityFile. This class holds an entity file in a side project. An entity class is the key class in database manipulations and form management. That means it is a data carriage container between a HTML requests and databases. In java convention an entity class is a totally bean class. Because it has only private properties and public methods to get and set those properties. In addition, there are code generating methods in this class which outputs database manipulation methods, table creation scripts, jsp forms and controller classes for themselves.

4.3.2.5 JavaFile. This class manages any Java class file and provides operations basically. EntityFile, BeanFile are subclasses of JavaFile. Some important methods in JavaFile are addMethod, insertProperty and copyMe.

4.3.2.6 Method. A method object is the whole representation of a method including all of its features. Annotation, return type, access level, name, parameters, thrown exceptions and the content are all represented by this class. Its toString method overrides superclass' method to give out its String representation.

4.3.2.7 DispatcherServletXML. This class represents a dispatcher servlet in a Springframework project. A dispatcher servlet xml file configures and informs Springframework about the beans and their settings for a specific extension like .htm or

.do. Springframework uses those files for its key feature namely dependency injection. Beans configured here are created by Springframework and made ready to use.

4.3.2.8 JSPFile. This is the class behind Jsp files in a SIDE project. Besides that it has methods to add forms to itself for a specified bean.

4.3.3. Package side.resources

This package holds properties file needed for the initialization of SIDE during the run. In addition, some pictures such as splash screen image and about dialog's images are stored here.

4.3.4. Package resources.files

This package contains files needed by SIDEView to configure itself during the run. Some application settings like JDK and Tomcat home directories, recent projects opened are recorded into the files raising here. In addition, the archive file which is used to form the base file structure of a Springframework project is stored here as basics.zip.

4.3.5. Package resources.icons

As the name suggests, this package was formed to hold the icons making the user experience with the interface of the application.

5. SAMPLE APPLICATION

For demonstrative purposes a sample application was created using development environment program. The sample application is named as SchoolDataManager. It aims to simply manage a university's database using Springframework's capabilities and Tomcat Application Server. Almost 95% of the code of the SchoolDataManager application was formed by development environment automatically and the whole database and all database manipulation codes were formed 100% automatically by development environment.

5.1 Database Design

As mentioned above SchoolDataManager is for managing departments, courses, students and stuff relationships in a university. For that reason there must be a persistent storage that will always remember the connections. For that reason, first step is designing the database. It is designed in Database Architect 1.9.0. The whole design is shown in Figure 5.1. The database of SchoolDataManager has seven database tables and has the name School. The tables and their fields are as follows:

- Courses: Responsible for storing data about courses within departments. DepartmentId shows the course's possession. CourseId, CourseName and DepartmentId are the fields in this table.
- Departments: Departments (divisions of faculties) are stored in this table with name. DepartmentId and DepartmentName are the fields in this table.
- Students: Stores data of students in specific departments. One student can only have one department and because of that the DepartmentId field is included here. Other information stored is student's age and name. StudentId, StudentName, Age, DepartmentId are the fields in this table.
- StudentsCourses: Relates a student and a course in normalized form. A student can have many courses and a course can have many students denoting a many-to-many relationship. StudentId and CourseId are the fields in this table.
- Stuffs: Stores data of stuff in specific departments. One stuf can only have one

department and because of that the DepartmentId field is included here. Other information stored is stuff's age, name and TitleId which is a foreign key to Titles table. StuffId, StuffName, StuffAge, TitleId and DepartmentId are the fields in this table.

- StuffsCourses: Relates stuff and a course in normalized form. Stuff can have many courses and a course can only have one stuff denoting a one-to-many relationship. StuffId and CourseId are the fields in this table.
- Titles: Stores titles of stuff in departments. TitleId and TitleName are the fields in this table.

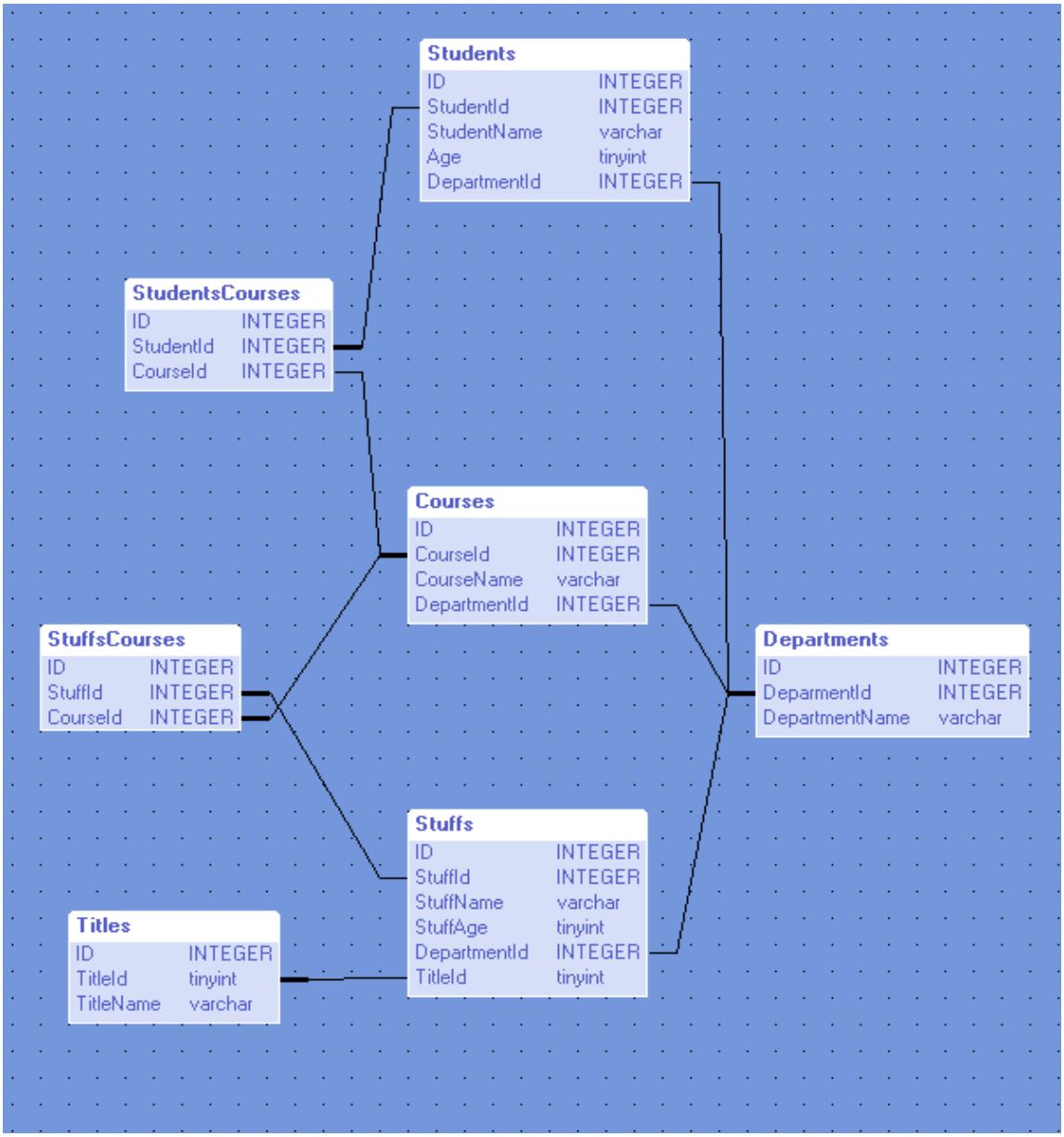


Figure 5.1. Design of the database using Database Architect 1.9.0

5.2 Tasks Before The Design

5.2.1 Configuration

In order one of our development environment instance to carry out its tasks it must be configured. The settings button on the main interface must be worked for that purpose. This will show the Configuration Settings dialog to the user. There are three settings that can be set: JDK Home Directory, Tomcat Home Directory and Project Files. Choosing one of them and hitting the change button will display a select file dialog for the purpose. For each setting only directories can be selected. If a wrong directory is selected for JDK or Tomcat the dialog will reject it. As can be seen in Figure 5.2 the dialog lets the user to configure all settings in one screen. The settings done using this dialog might performed in any time during a project development and can be changed according to the request.

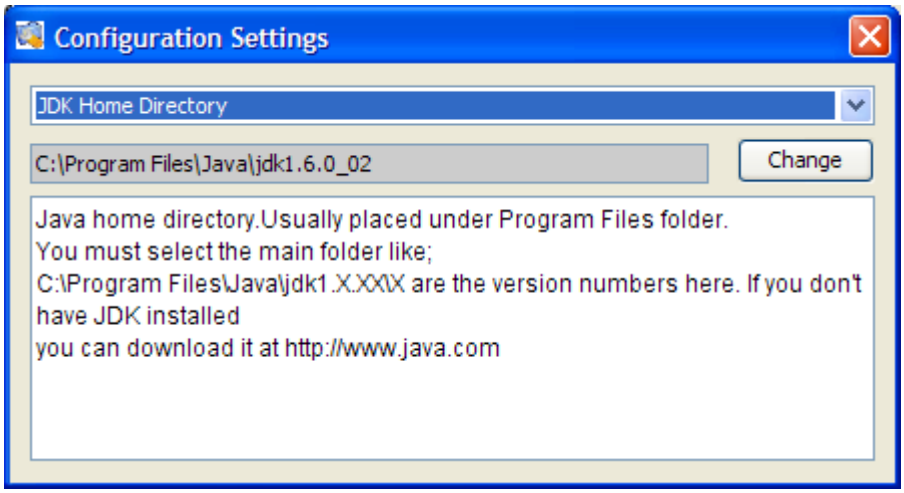


Figure 5.2. Configuration Settings dialog

5.2.2 Project Creation

To start working, a project should be loaded into the development environment. This can be done either opening an existing project or creating a new one from scratch. An existing project can be opened by selecting its directory in the open file dialog shown when open project menu item or open project toolbox item is activated. A project can be created from scratch via hitting the new project menu item under the file menu. This will show an input dialog waiting for user input for new project's name as in Figure 5.3.

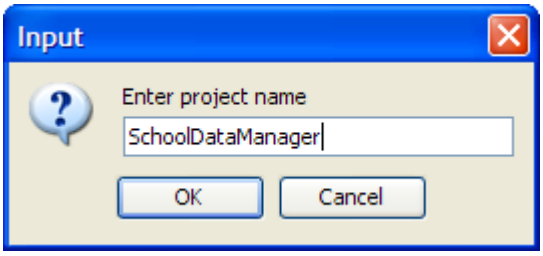


Figure 5.3. Creating the project SchoolDataManager

5.3 Application Development Steps

5.3.1 Creating Entity Classes

To effectively manipulate database persistence, the application needs both database side and application side storage. Application side storage is consisted of entity classes. These entity classes' instances are used during runtime to manipulate the pulled data from the database. Database side storage is database tables.

In order to create database tables entity classes are needed. Entity classes are their counter parts. To add an entity class to the project either right-click on the project tree or use Project menu and select "add entity" button. This will show a wizard that enables the user to enter properties of the entity class. A property is the counter part of database table fields and has the similar data type. For example, a varchar field in database is a string property in an entity class. Properties must obey the bean logic and should have names starting with lower case letter. After adding as necessary as properties to the entity class and entering a suitable name for it, finishing the wizard will end up with an appropriate entity class.

5.3.2 Creating Database

To form database tables from entity classes it is necessary to have a database. To add database support, the add database button under project menu must be used. After giving the desired name, the database support having HSQL engine behind will be created and be ready to be added database tables. After adding a database to the project, the add database button is disabled and the add table button is enabled.

5.3.3 Creating Database Tables

Creating a database table for an entity bean is a one-step process but it gives out many outputs. To form a table, the add table button under project menu must be used. This button is disabled before adding database persistence support.

The click event of the button will show up the dialog that is carrying entity classes in its combo box. As shown in Figure 5.4, after selecting the desired entity class and hitting the generate button will create 4 outputs:

- The database table
- A search page in jsp format with the name "*EntitysName*SearchPage.jsp" in the directory project_home/web/WEB-INF/jsp
- A controller for the search page in the directory project_home/src/project_name/spring/controllers with the name "Search*EntitysName*Controller.java"
- A database helper bean class in the directory project_home/src/project_name/spring/beans with the name "*EntitysName*DBSearchHelper.java"
- A delete operation controller class in the directory project_home/src/project_name/spring/controllers with the name "Delete*EntitysName*.java"
- An edit page in the directory project_home/web/WEB-INF/jsp with the name "Edit*EntitysName*.jsp"
- A controller for the edit page in the directory project_home/src/project_name/spring/controllers with the name "Edit*EntitysName*Controller.java"

For each of our entity classes we repeat this procedure and get helper and manipulator classes and views. These classes with the forms and databases form the backbone of an application. After each operation the dispatcher servlet is changed in order to make the Springframework handle bindings between the controllers and pages.

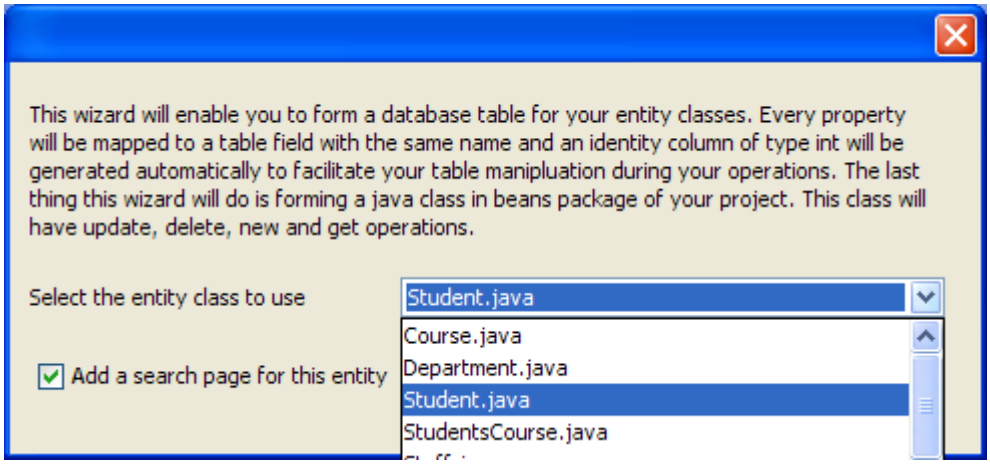


Figure 5.4. Forming the database table for the student entity class

5.3.4 Creating Information Recording Pages

Having the infrastructure classes and databases tables, the step to accept user input and record it to the respective tables in the database may be considered. Pointing the add form button either in the project menu or in the pop up menu will show form dialog.

Form dialog is a 5 step wizard. Each step is positioned on a tab control on the dialog.

5.3.4.1 Step 1. In this step, Jsp pages' details are demanded from the user. Either a page name is entered or an existing page's name is selected from the upper combo box. This page is where the form will be put. Another page is selected in the lower combo box. This page is where the user will be redirected when the data is recorded successfully. If no page is selected the user will be redirected to the entry page of the application. Figure 5.5 shows this step while entering the name of the successful operation page.

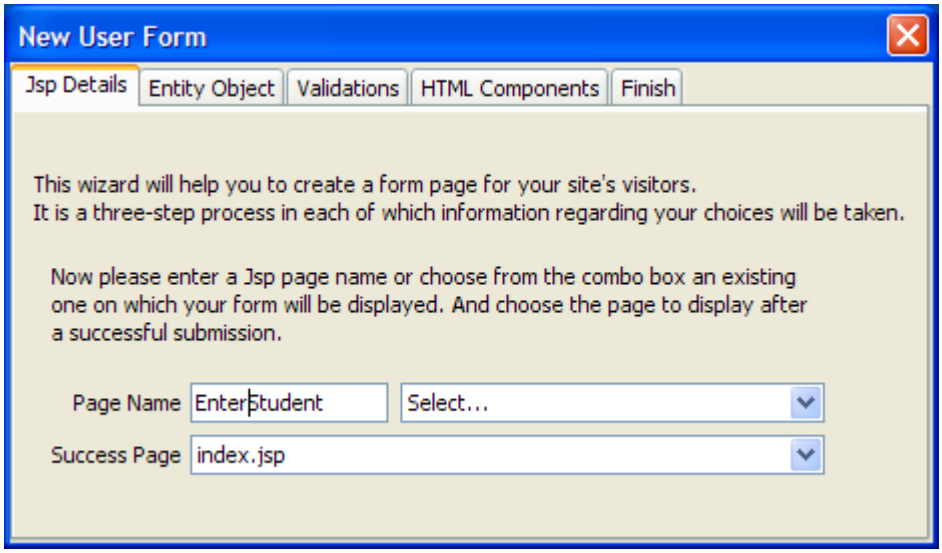


Figure 5.5. Jsp view settings while forming student recording page

5.3.4.3 Step 2. In this step, the entity class for which the form will be prepared is selected from the combo box on the tab. Any selection change here will reset the settings in 4th and 5th steps. Figure 5.6 shows this step while selecting an entity class found in the project.

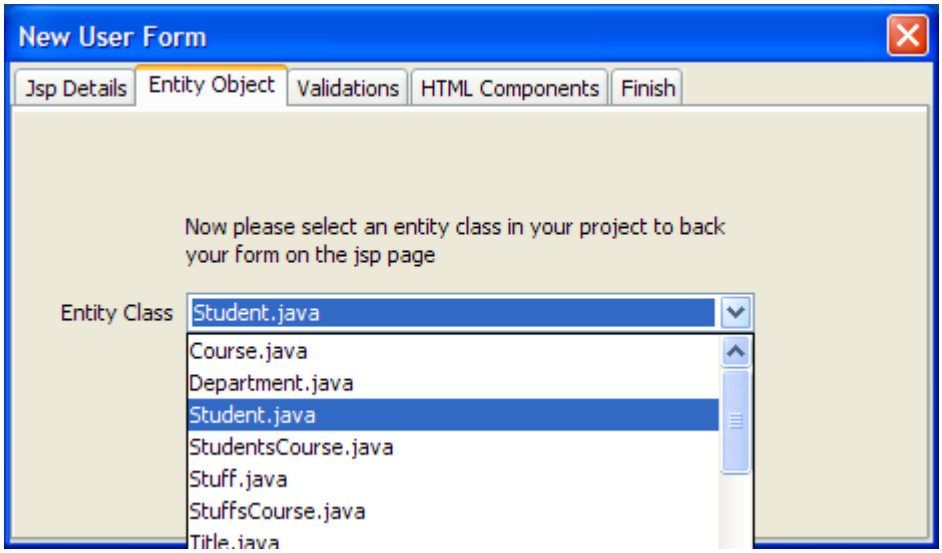


Figure 5.6. Selecting entity class while forming student recording page.

5.3.4.3 Step 3. In this step validations to be done when the form is submitted are set by the user according to his purpose. Validations are done before the request drops to the control and if the page is not valid it is redirected to the requesting page, in this case the form page.

A property is selected in the most upper combo box. For String properties there are 6 settings that can be done. Those settings control the equality, starting character sequence, ending character sequence, any character sequence and the length of the entered value. For integer fields there is only one setting which controls the values equality or inequality according to a preferred value. Update button must be clicked for every property that is set. Figure 5.7 shows this step while selecting a property from the combo box and a value entered with the minimum sign.

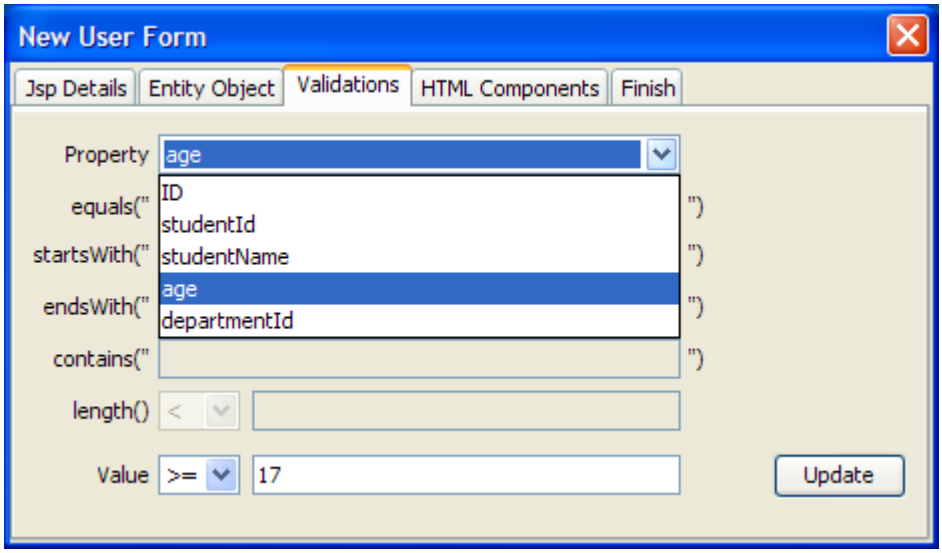


Figure 5.7. Entering validations while forming student recording page.

5.3.4.4 Step 4. In this step the HTML form components for the properties are set. HTML controls are radio button, check box, select, text box, password and text area. For select and radio button a comma separated list can be entered into the box below. After each arrangement the Update Map button must be pushed to record the preferences.

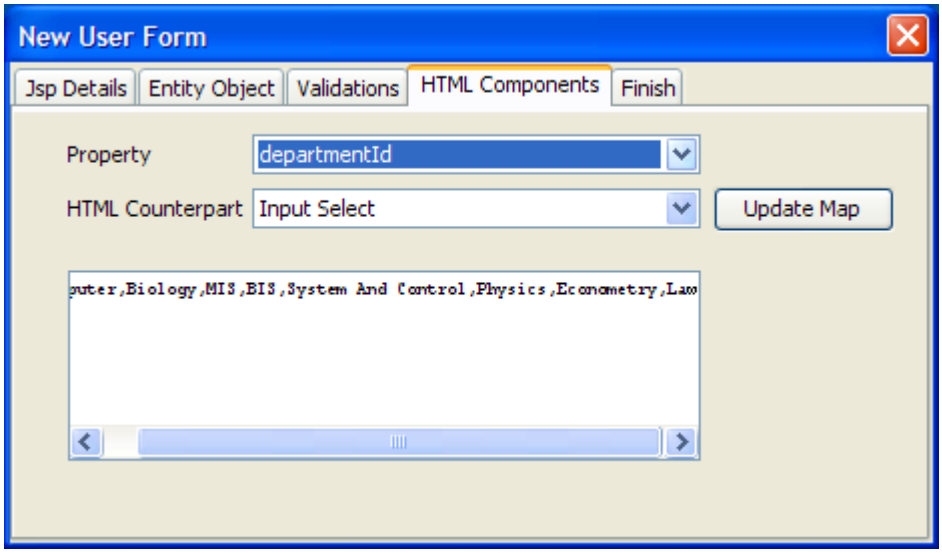


Figure 5.8. Setting up HTML components to be shown on the student recording page while forming it.

5.3.4.5 Step 5. This is the last step of the wizard. In the combo box on the tab the Data Record option can be selected to automate a registration purposes form's data recording. Finish button will finish the operation and ask if the settings are to be approved. Figure 5.9 and 5.10 illustrates these steps in order.

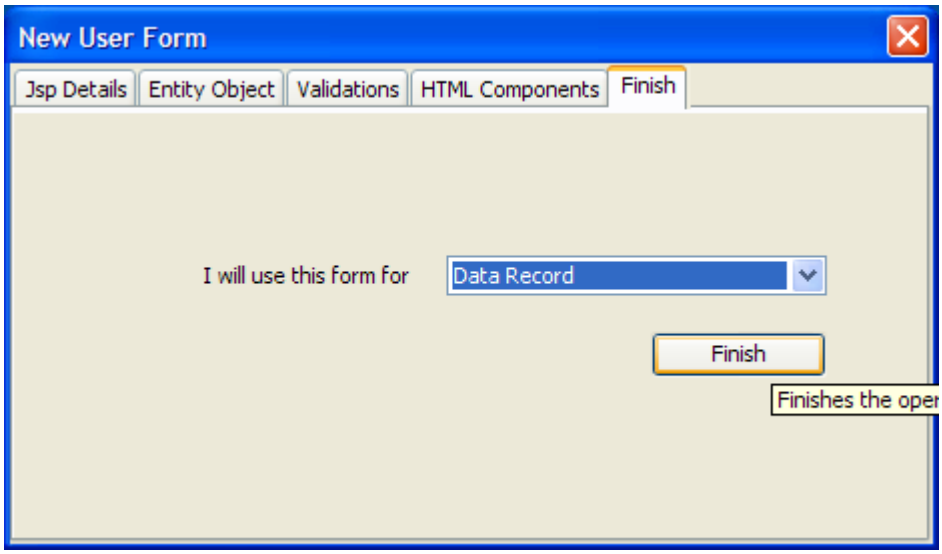


Figure 5.9. Finishing the student recording page creation.

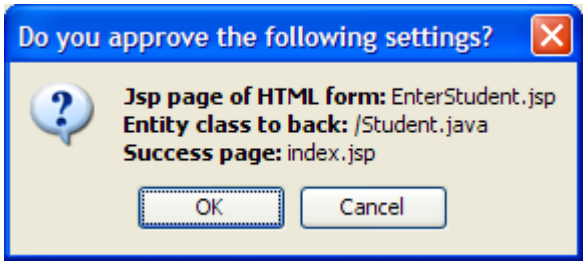


Figure 5.10. Approving the settings.

This wizard will give 4 output files;

- *EntityName*DBHelper.java class in the directory project_home/src/projectname/spring/beans. This bean enables the user to add, delete, update or get a record.
- A jsp page in the directory project_home/web/WEB-INF/jsp with the name entered in the first step. This is the page where the form is placed. If a page is selected from the combo box in the first step there is no new page and the form is placed in the selected view.
- *PageName*Validator.java in the folder project_home/src/projectname/spring/beans. This bean is responsible for controlling the values submitted in the page according to the settings done in the step 4.
- *PageName*Controller.java in the folder project_home/src/projectname/spring/controllers. This class controls the page. It handles redirections, database recording after a valid form submission, error reporting and creating an entity object with the submitted information.

In addition, after this operation the dispatcher servlet is evolved to include the newly coming pages and controllers. The binding is done.

5.3.5 Compiling The Project

To compile the project, the compile project menu item under project menu must be clicked. Figure 5.11 shows how to perform this operation. The development environment will compile each java file under beans, controllers and entities folder with the java compiler selected in the configuration dialog. If there is no error the project is compiled in class files else the errors and warning are shown to the user inside text box on a pop up

box. These errors and warnings are java compiler's output.

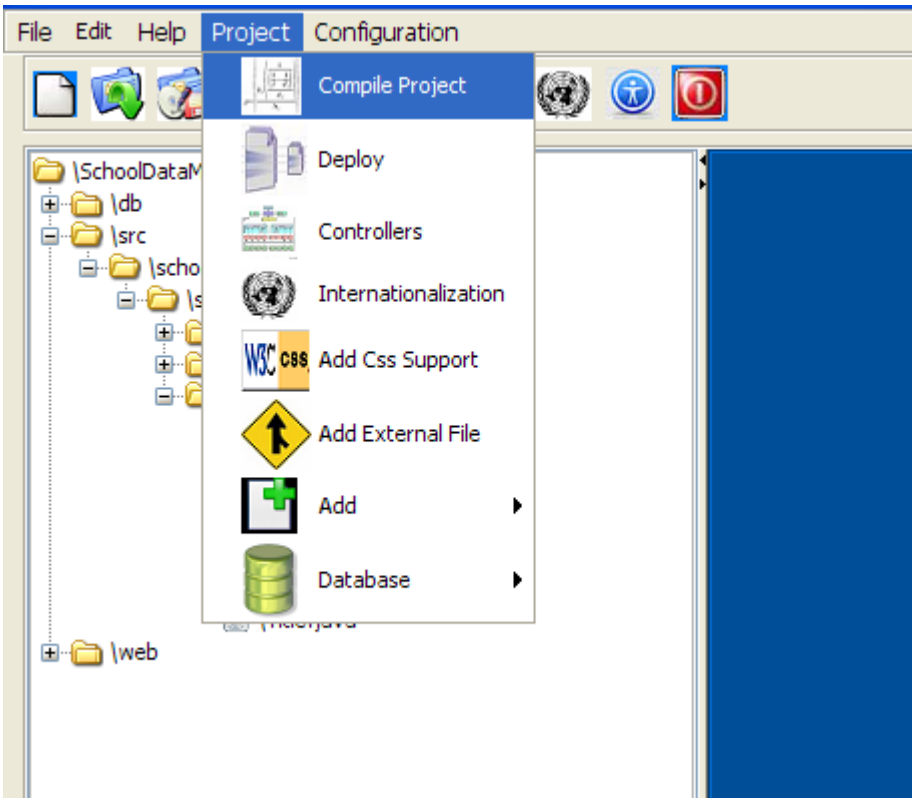


Figure 5.11. Compiling the project.

After the compilation, the project tree looks like as in Figure 5.12. As can be seen, all class files of Java files in the project have been formed in the directory where the corresponding Java class resides. This is also the final look of the project tree.

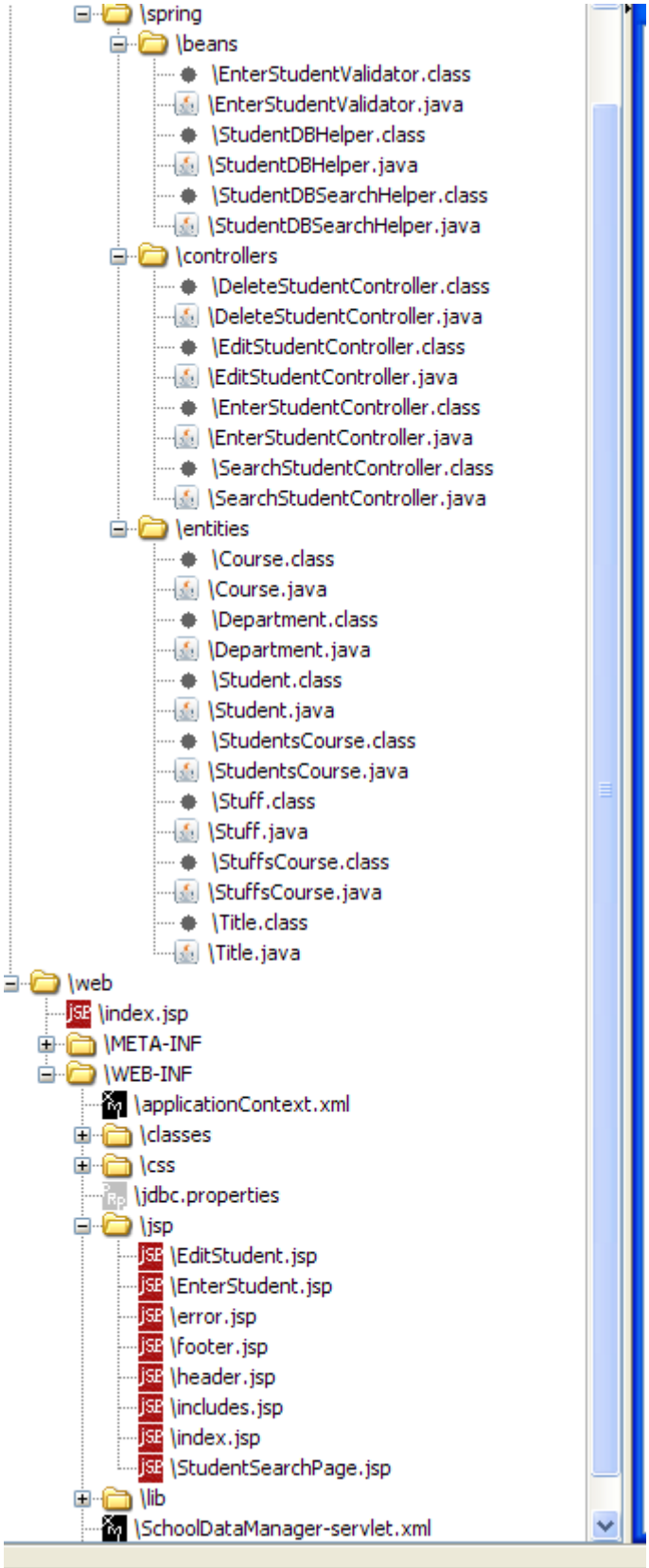


Figure 5.12. The project Tree after forming student entity class' database table, its search page, its edit page, its delete option and its recording page and compiling the java files into classes.

5.3.6 Deployment Of The Project

After successful compilation the project must be deployed into a Jakarta Tomcat instance. This is a one-step process. To deploy the compiled class files, the Deploy menu item under the project menu must be used. If there is no Tomcat home directory selected yet this will end up with a warning requesting a Tomcat configuration else the necessary files will be copied to the Tomcat server selected.

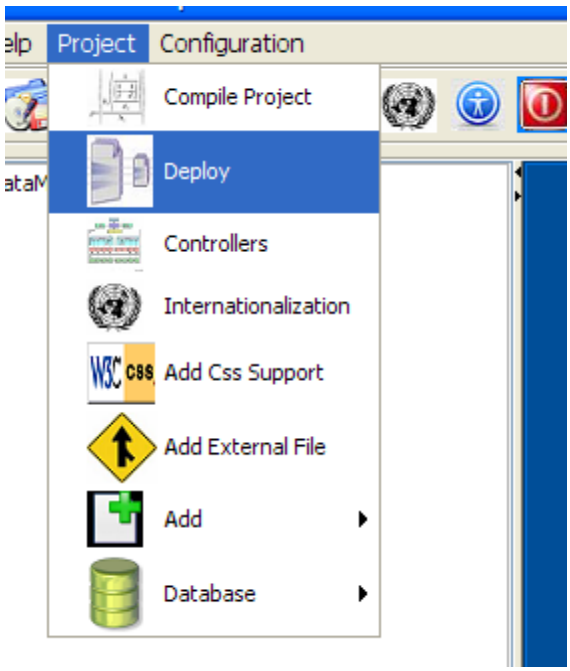


Figure 5.13. Deploying SchoolDataManager into Tomcat 6.0

5.3.7 Running The Project

After the deployment the SchoolDataManager application is ready to use. The EnterStudent page is called via a browser software like Internet Explorer or Mozilla Firefox from the local Tomcat server. The address is changed according to Tomcat's own configuration. Usually Tomcat uses port 8080. Because of that with the context path of SchoolDataManager application the url to be called is http://localhost:8080/SchoolDataManagerApplication/EnterStudent.htm. This will bring the form seen below in Figure 5.14.

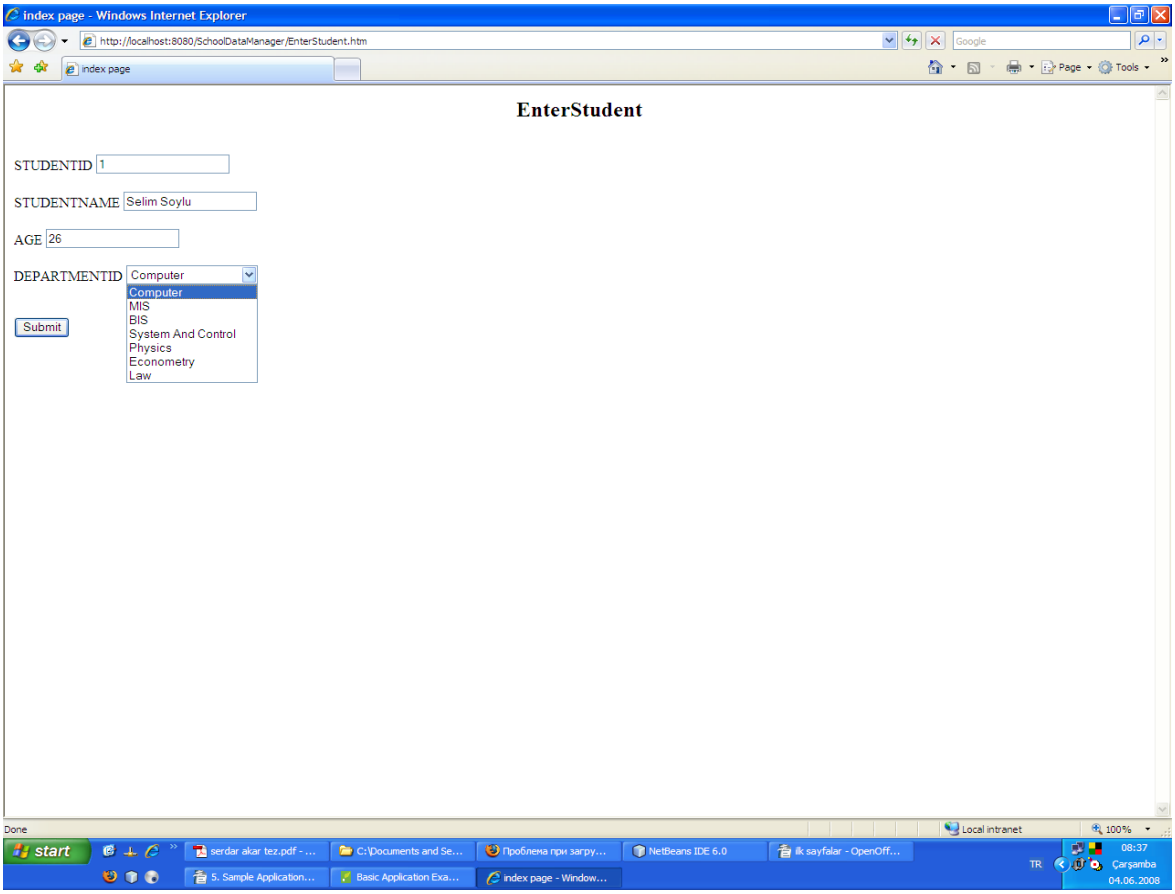


Figure 5.14. Enter student form filled with values

Entering valid values into the text boxes and hitting the submit button will save the student information into the database. After this operation the client will be redirected to the successful operation page which was selected during the addition of this form. In case of an error during database operations the client will be redirected to the error.jsp page. In case of wrong values entered into the field there will be no redirection. Instead, the fields will remain the same and there will be error messages shown under the fields where there was a faulty value entrance. The values chosen for the first record are 1 for StudentId, Selim Soylu for StudentName, 26 for Age and Computer for Department.

After entering values and successfully submitting the form, the search page can be tested. Search page is a form having 3 form controls. A button for submission, a text field for the search value and a select control for selecting the database field to look. Writing 1 into the text box and selecting studentId from the select list will give the search result entered before. The HTML page looks as in Figure 5.15. A line represents a record in the

database that matches the search criteria. The look of the search result is a table with table fields names as headers. There are two links beside every line: Edit and Delete. Delete link will delete the row from the database and edit will redirect the user to the edit page where the values can be changed and updated.

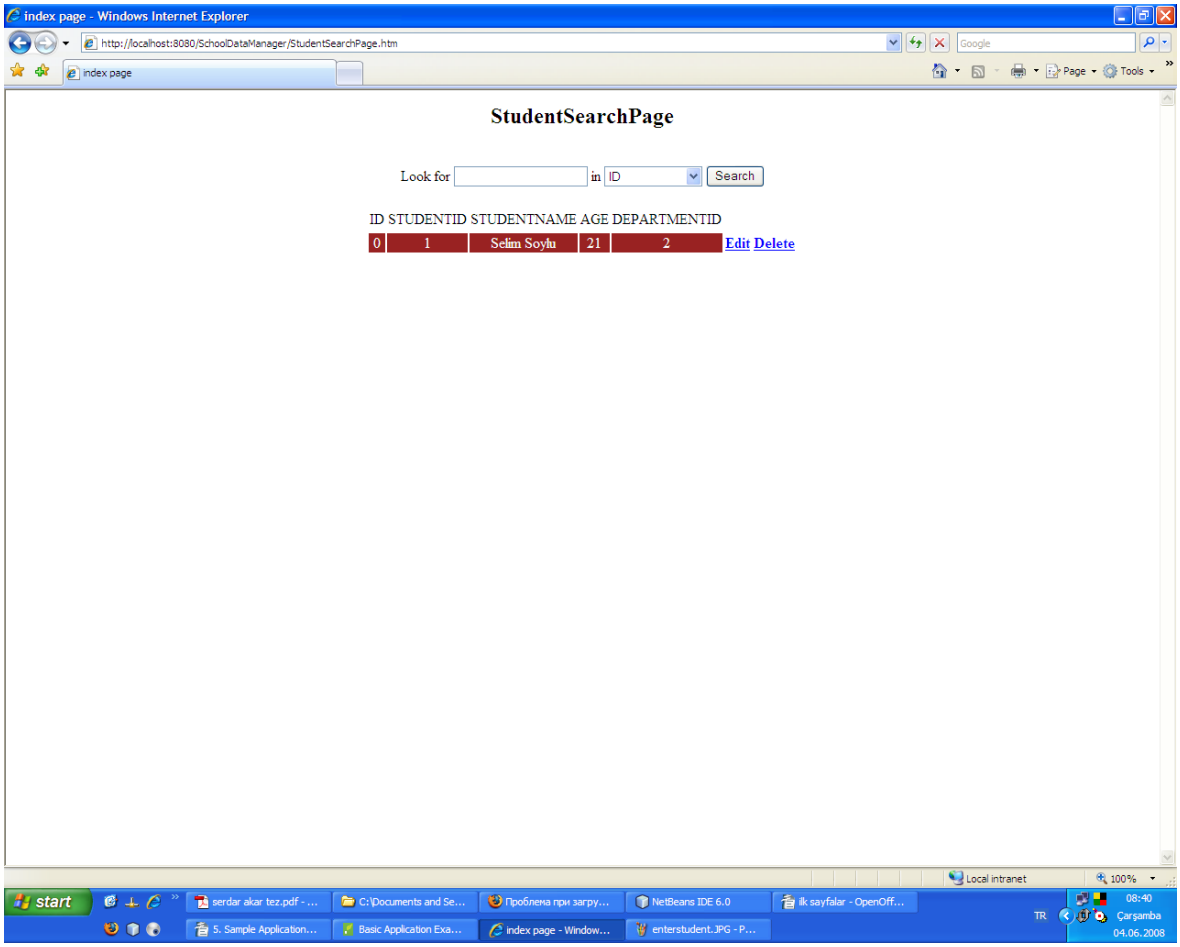


Figure 5.15. Search results in the search page without any styling after looking for the student, studentId 1

In Figure 5.16, editing the record with the studentId 1 is illustrated. This page is brought when hit the edit hyper link on the right of the record resulted after searching in the StudentSearchPage.htm.

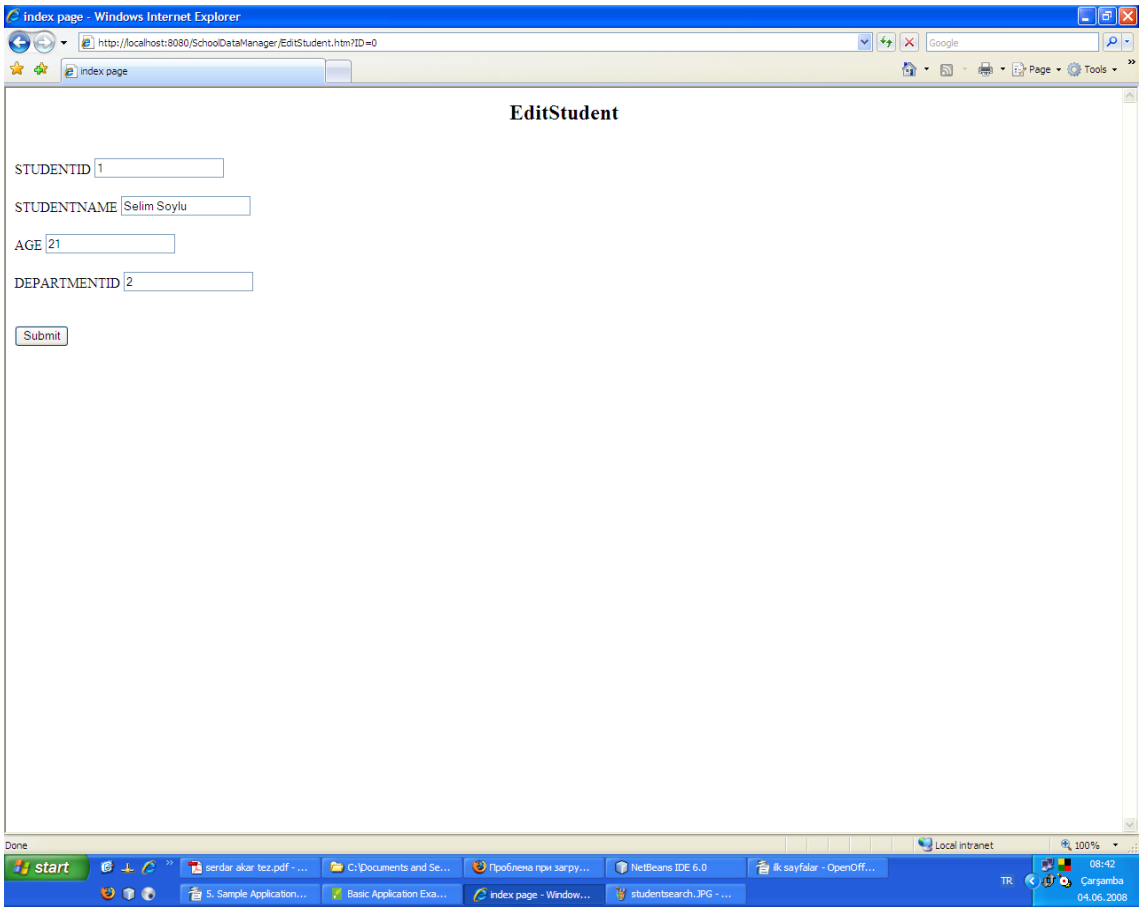


Figure 5.16. Editing the record with the studentId 1 in EditStudent.htm page

6. CONCLUSIONS

The resulting program is a graphical user interface that enables its users to manage and develop their Springframework-based applications. The functions of the program are all used via the items on its GUI. This graphical manipulation technique is for creating a difference from straightforward development and thus facilitating the development process of a Springframework based application.

Besides being a graphical interface and providing a centralized place for a Springframework based application, there are other features that more facilitate developing applications. All the files necessary for an application are stored in a different way than the aimed running application for a more comfortable development phase. And during the development phase many tasks are performed by the program automatically. Some of them are code and JSP page generations, making XML configurations, adding database and database tables, internationalization, compilation, deployment and file additions and subtractions.

Nonetheless having many features that facilitate application development, there are many other features that can be added in the future. Since the thesis is open source, for any Java developer it is possible to extend the program. Some features can be added in the future are wiring views and controllers in a graphical manner, opening many projects in the same session, image import and debugging support.

REFERENCES

1. Website “http://www.wikipedia.org”

2. Website “http://www.springframework.org”

3. Pree, W. (1994). Meta patterns - a means for capturing the essentials of reusable object-oriented design. in M. Tokoro and R. Pareschi (eds), Springer-Verlag, proceedings of the ECOOP, Bologna, Italy: pp 150-162

4. Tate, Bruce A. and Gehtland, J. (2005). Spring A Developer's Notebook, O'Reilly MediaISBN 9780596009106

5. Buschmann, F. (1996). Pattern-oriented software architecture: a system of patterns. Chichester; New York, Wiley. ISBN 978-0471958697

6. Kassem N. (2000) Designing Enterprise Applications with the Java 2 Platform, Enterprise Edition. California USA, Sun Microsystems. ISBN 0-201-70277-0

7. Website “http://www.marston-home.demon.co.uk/Tony/uniface/3tier”, 3-Tier Overview

8. Web Site “http://atlas.kennesaw.edu/~dbraun/csis4650/A&D/UML_tutorial/”, What is UML

9. Website “http://hsqldb.org/web/hsqlDocsFrame.html”

10. Johnson, Rod (2003). Introduction to Springframework

11. Risberg, Thomas (2003). Developing a Springframework MVC application step by step

12. Arthur, J. and Azadegan S. (2005). Spring Framework for rapid open source J2EE web

application development: A case study

13. Oktay, M., Gülbağcı, A. B., Sarıöz, M. (2007). Architectural, Technological and Performance Issues in Enterprise Applications by [Proceedings of world academy of science, engineering and technology volume 05.21.2007 ISSN 1307-6884]

14. Denoncourt, Don (2005). Crafting an application architecture with Java frameworks. Proceedings of the Sixth International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing and First ACIS International Workshop on Self-Assembling Wireless Networks (SNPD/SAWN'05) 0-7695-2294-7/05

REFERENCES NOT CITED

Larman, C. (2002). Applying UML and patterns: an introduction to object-oriented analysis and design and the unified process. Upper Saddle River, NJ, Prentice Hall PTR.

Knuth, Donald E. (1997). An expression of a computational method in a computer language is called a program. The Art of Computer Programming, Volume 1, 3rd Edition. Boston: Addison-Wesley, pp.5. ISBN 0-201-89683-4.

Bayramlı, B. (2005). Spring Framework

Johnson, Rod (2002)., J2EE Design And Development, Wrox SBN 978-0-7645-4385-2

Walls, Craig (2007). Spring In Action 2nd Edition, Manning Publications ISBN 1-933988-13-4

CGI (Common Gateway Interface). World of Computer Science. Ed. Brigham Narins. (Detroit: Gale, 2002) Science Resource Center.

Website “http://www.computer.org/portal/site/computer” J2EE Development Frameworks

Website “http://www.oxfordreference.com/views/ENTRY.html”, Scripting language; A Dictionary of Computing.

Website “http://www-128.ibm.com/developerworks/java/library/j-aopwork13.html”, Dependency Injection with AspectJ and Spring

Website “http://www.smartdraw.com/tutorials/software/uml/tutorial_01.htm”

Website “http://www.dotnetcoders.com/web/learning/uml/diagrams/statechart.aspx”

Website “http://dev2dev.bea.com/pub/a/2006/10/spring-jdbc-dao.html”, A Primer on Spring's Data Access Object (DAO) Framework

7. APPENDICES

A. SAMPLE APPLICATION CODES

1. SchoolDataManager-servlet.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:aop="http://www.springframework.org/schema/aop"
xmlns:tx="http://www.springframework.org/schema/tx"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-2.0.xsd
http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-2.0.xsd">

   <bean
class="org.springframework.web.servlet.mvc.support.ControllerClassNameHandler
Mapping"/>
   <bean class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor"
id="localeChangeInterceptor">
      <property name="paramName" value="lang"/>
   </bean>
   <bean
class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping"
id="urlMapping">
      <property name="interceptors">
         <list>
            <ref bean="localeChangeInterceptor"/>
         </list>
```

```
    </property>
    <property name="mappings">
      <props>
        <prop key="/index.htm">indexController</prop>
        <prop key="/StudentSearchPage.htm">SearchStudentController</prop>
        <prop key="/EditStudent.htm">EditStudentController</prop>
        <prop key="/DeleteStudent.htm">DeleteStudentController</prop>
        <prop key="/EnterStudent.htm">EnterStudentController</prop>
        <prop key="/TitleSearchPage.htm">SearchTitleController</prop>
        <prop key="/EditTitle.htm">EditTitleController</prop>
        <prop key="/DeleteTitle.htm">DeleteTitleController</prop>
        <prop key="/CourseSearchPage.htm">SearchCourseController</prop>
        <prop key="/EditCourse.htm">EditCourseController</prop>
        <prop key="/DeleteCourse.htm">DeleteCourseController</prop>
        <prop
key="/DepartmentSearchPage.htm">SearchDepartmentController</prop>
        <prop key="/EditDepartment.htm">EditDepartmentController</prop>
        <prop
key="/DeleteDepartment.htm">DeleteDepartmentController</prop>
        <prop key="/StuffSearchPage.htm">SearchStuffController</prop>
        <prop key="/EditStuff.htm">EditStuffController</prop>
        <prop key="/DeleteStuff.htm">DeleteStuffController</prop>
        <prop
key="/StudentsCourseSearchPage.htm">SearchStudentsCourseController</prop>
        <prop
key="/EditStudentsCourse.htm">EditStudentsCourseController</prop>
        <prop
key="/DeleteStudentsCourse.htm">DeleteStudentsCourseController</prop>
        <prop
key="/StuffsCourseSearchPage.htm">SearchStuffsCourseController</prop>
        <prop key="/EditStuffsCourse.htm">EditStuffsCourseController</prop>
        <prop
key="/DeleteStuffsCourse.htm">DeleteStuffsCourseController</prop>
```

```
            <prop key="/EnterCourse.htm">EnterCourseController</prop>
            <prop key="/EnterDepartment.htm">EnterDepartmentController</prop>
            <prop key="/EnterStuff.htm">EnterStuffController</prop>
            <prop key="/EnterTitle.htm">EnterTitleController</prop>
            <prop key="/EnterStuffCourse.htm">EnterStuffCourseController</prop>
        </props>
    </property>
  </bean>
  <bean class="org.springframework.web.servlet.i18n.CookieLocaleResolver"
id="localeResolver"/>
  <bean
class="org.springframework.web.servlet.view.DefaultRequestToViewNameTranslat
or" id="viewNameTranslator"/>
  <bean
class="org.springframework.web.servlet.view.InternalResourceViewResolver"
id="viewResolver">
    <property name="viewClass"
value="org.springframework.web.servlet.view.JstlView"/>
    <property name="prefix" value="/WEB-INF/jsp/"/>
    <property name="suffix" value=".jsp"/>
  </bean>
  <bean
class="org.springframework.web.servlet.mvc.ParameterizableViewController"
name="indexController">
    <!-- this will forward directly to view stored in /WEB-INF/jsp/index.jsp -->
    <property name="viewName" value="index"/>
  </bean>
  <bean class="schooldatamanager.spring.controllers.SearchStudentController"
id="SearchStudentController"/>
  <bean class="schooldatamanager.spring.controllers.DeleteStudentController"
id="DeleteStudentController"/>
  <bean class="schooldatamanager.spring.controllers.EditStudentController"
id="EditStudentController">
```

```
        <property name="formView" value="EditStudent"/>
        <property name="successView" value="Success"/>
        <property name="commandName">
            <value>student</value>
        </property>
        <property name="commandClass">
            <value>schooldatamanager.spring.entities.Student</value>
        </property>
    </bean>
    <bean class="schooldatamanager.spring.controllers.EnterStudentController"
id="EnterStudentController">
        <property name="validator" ref="EnterStudentValidator"/>
        <property name="formView" value="EnterStudent"/>
        <property name="successView" value="index"/>
        <property name="commandName">
            <value>student</value>
        </property>
        <property name="commandClass">
            <value>schooldatamanager.spring.entities.Student</value>
        </property>
    </bean>
    <bean class="schooldatamanager.spring.beans.EnterStudentValidator"
name="EnterStudentValidator"/>
    <bean class="schooldatamanager.spring.controllers.SearchTitleController"
id="SearchTitleController"/>
    <bean class="schooldatamanager.spring.controllers.DeleteTitleController"
id="DeleteTitleController"/>
    <bean class="schooldatamanager.spring.controllers.EditTitleController"
id="EditTitleController">
        <property name="formView" value="EditTitle"/>
        <property name="successView" value="Success"/>
        <property name="commandName">
            <value>title</value>
```

```
        </property>
        <property name="commandClass">
            <value>schooldatamanager.spring.entities.Title</value>
        </property>
    </bean>
    <bean class="schooldatamanager.spring.controllers.SearchCourseController"
id="SearchCourseController"/>
    <bean class="schooldatamanager.spring.controllers.DeleteCourseController"
id="DeleteCourseController"/>
    <bean class="schooldatamanager.spring.controllers.EditCourseController"
id="EditCourseController">
        <property name="formView" value="EditCourse"/>
        <property name="successView" value="Success"/>
        <property name="commandName">
            <value>course</value>
        </property>
        <property name="commandClass">
            <value>schooldatamanager.spring.entities.Course</value>
        </property>
    </bean>
    <bean
class="schooldatamanager.spring.controllers.SearchDepartmentController"
id="SearchDepartmentController"/>
    <bean
class="schooldatamanager.spring.controllers.DeleteDepartmentController"
id="DeleteDepartmentController"/>
    <bean class="schooldatamanager.spring.controllers.EditDepartmentController"
id="EditDepartmentController">
        <property name="formView" value="EditDepartment"/>
        <property name="successView" value="Success"/>
        <property name="commandName">
            <value>department</value>
        </property>
```

```
        <property name="commandClass">
            <value>schooldatamanager.spring.entities.Department</value>
        </property>
    </bean>
    <bean class="schooldatamanager.spring.controllers.SearchStuffController"
id="SearchStuffController"/>
    <bean class="schooldatamanager.spring.controllers.DeleteStuffController"
id="DeleteStuffController"/>
    <bean class="schooldatamanager.spring.controllers.EditStuffController"
id="EditStuffController">
        <property name="formView" value="EditStuff"/>
        <property name="successView" value="Success"/>
        <property name="commandName">
            <value>stuff</value>
        </property>
        <property name="commandClass">
            <value>schooldatamanager.spring.entities.Stuff</value>
        </property>
    </bean>
    <bean
class="schooldatamanager.spring.controllers.SearchStudentsCourseController"
id="SearchStudentsCourseController"/>
    <bean
class="schooldatamanager.spring.controllers.DeleteStudentsCourseController"
id="DeleteStudentsCourseController"/>
    <bean
class="schooldatamanager.spring.controllers.EditStudentsCourseController"
id="EditStudentsCourseController">
        <property name="formView" value="EditStudentsCourse"/>
        <property name="successView" value="Success"/>
        <property name="commandName">
            <value>studentsCourse</value>
        </property>
```

```
        <property name="commandClass">
            <value>schooldatamanager.spring.entities.StudentsCourse</value>
        </property>
    </bean>
    <bean
class="schooldatamanager.spring.controllers.SearchStuffsCourseController"
id="SearchStuffsCourseController"/>
    <bean
class="schooldatamanager.spring.controllers.DeleteStuffsCourseController"
id="DeleteStuffsCourseController"/>
    <bean class="schooldatamanager.spring.controllers.EditStuffsCourseController"
id="EditStuffsCourseController">
        <property name="formView" value="EditStuffsCourse"/>
        <property name="successView" value="Success"/>
        <property name="commandName">
            <value>stuffsCourse</value>
        </property>
        <property name="commandClass">
            <value>schooldatamanager.spring.entities.StuffsCourse</value>
        </property>
    </bean>
    <bean class="schooldatamanager.spring.controllers.EnterStuffController"
id="EnterStuffController">
        <property name="validator" ref="EnterStuffValidator"/>
        <property name="formView" value="EnterStuff"/>
        <property name="successView" value="index"/>
        <property name="commandName">
            <value>stuff</value>
        </property>
        <property name="commandClass">
            <value>schooldatamanager.spring.entities.Stuff</value>
        </property>
    </bean>
```

```
   <bean class="schooldatamanager.spring.beans.EnterStuffValidator"
name="EnterStuffValidator"/>
   <bean class="schooldatamanager.spring.controllers.EnterCourseController"
id="EnterCourseController">
      <property name="validator" ref="EnterCourseValidator"/>
      <property name="formView" value="EnterCourse"/>
      <property name="successView" value="index"/>
      <property name="commandName">
         <value>course</value>
      </property>
      <property name="commandClass">
         <value>schooldatamanager.spring.entities.Course</value>
      </property>
   </bean>
   <bean class="schooldatamanager.spring.beans.EnterCourseValidator"
name="EnterCourseValidator"/>
   <bean class="schooldatamanager.spring.controllers.EnterDepartmentController"
id="EnterDepartmentController">
      <property name="validator" ref="EnterDepartmentValidator"/>
      <property name="formView" value="EnterDepartment"/>
      <property name="successView" value="index"/>
      <property name="commandName">
         <value>department</value>
      </property>
      <property name="commandClass">
         <value>schooldatamanager.spring.entities.Department</value>
      </property>
   </bean>
   <bean class="schooldatamanager.spring.beans.EnterDepartmentValidator"
name="EnterDepartmentValidator"/>
   <bean class="schooldatamanager.spring.controllers.EnterTitleController"
id="EnterTitleController">
      <property name="validator" ref="EnterTitleValidator"/>
```

```
        <property name="formView" value="EnterTitle"/>
        <property name="successView" value="index"/>
        <property name="commandName">
          <value>title</value>
        </property>
        <property name="commandClass">
          <value>schooldatamanager.spring.entities.Title</value>
        </property>
    </bean>
    <bean class="schooldatamanager.spring.beans.EnterTitleValidator"
name="EnterTitleValidator"/>
    <bean class="schooldatamanager.spring.controllers.EnterStuffCourseController"
id="EnterStuffCourseController">
        <property name="validator" ref="EnterStuffCourseValidator"/>
        <property name="formView" value="EnterStuffCourse"/>
        <property name="successView" value="index"/>
        <property name="commandName">
          <value>stuffsCourse</value>
        </property>
        <property name="commandClass">
          <value>schooldatamanager.spring.entities.StuffsCourse</value>
        </property>
    </bean>
    <bean class="schooldatamanager.spring.beans.EnterStuffCourseValidator"
name="EnterStuffCourseValidator"></bean>
</beans>
```

EnterStuff.jsp

```
<%@ include file="/WEB-INF/jsp/includes.jsp" %>
<%@ include file="/WEB-INF/jsp/header.jsp" %>
```

```
<center>
   <h2>EnterStuff</h2>
   <br />
</center>

<%@ include file="/WEB-INF/jsp/footer.jsp" %>
<form name="form1" method="post">
   STUFFID<spring:bind path="stuff.stuffId">
      <input type="text" name="stuffId" value="<c:out value="${status.value}"/>"/
>
         <br/><c:forEach items="${status.errorMessages}" var="error">Error
code: <c:out value="${error}"/><br/></c:forEach><br/>
   </spring:bind>
   STUFFNAME<spring:bind path="stuff.stuffName">
      <input type="text" name="stuffName" value="<c:out value="$
{status.value}"/>"/>
         <br/><c:forEach items="${status.errorMessages}" var="error">Error
code: <c:out value="${error}"/><br/></c:forEach><br/>
   </spring:bind>
   STUFFAGE<spring:bind path="stuff.stuffAge">
      <input type="text" name="stuffAge" value="<c:out value="$
{status.value}"/>"/>
         <br/><c:forEach items="${status.errorMessages}" var="error">Error
code: <c:out value="${error}"/><br/></c:forEach><br/>
   </spring:bind>
   TITLEID<spring:bind path="stuff.titleId">
      <form:select path="titles">
            <form:options items="${titles}" itemValue="titleId"
itemLabel="titleName"/>
         </form:select>
<br/><c:forEach items="${status.errorMessages}" var="error">Error code: <c:out
value="${error}"/><br/></c:forEach><br/>
   </spring:bind>
```

```
    DEPARTMENTID<spring:bind path="stuff.departmentId">
<form:select path="departments">
                <form:options items="${departments}" itemValue="departmentId"
itemLabel="departmentName"/>
            </form:select>
<br/><c:forEach items="${status.errorMessages}" var="error">Error code: <c:out
value="${error}"/><br/></c:forEach><br/>
    </spring:bind>
    <br/>
    <input type="submit" value="Submit" name="Submit"/>
    <spring:hasBindErrors name="errorMessage">
        <font color="red"><c:out value="${status.errorMessage}"/></font>
    </spring:hasBindErrors>
</form>
```

Course.java

```
package schooldatamanager.spring.entities;

public class Course {

    private int ID;

    public void setID(int value) {
        ID = value;
    }

    public int getID() {
        return ID;
    }

    public Course() {
```

```
    }
    private String courseName;

    public void setCourseName(String value) {
        courseName = value;
    }

    public String getCourseName() {
        retu”rn courseName;
    }
    private int departmentId;

    public void setDepartmentId(int value) {
        departmentId = value;
    }

    public int getDepartmentId() {
        return departmentId;
    }
    private int coursetId;

    public void setCoursetId(int value) {
        coursetId = value;
    }

    public int getCoursetId() {
        return coursetId;
    }
}
```

EnterStudentController.java

```
package schooldatamanager.spring.controllers;

import java.util.HashMap;
import java.util.Map;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.SimpleFormController;
import javax.servlet.ServletException;
import org.springframework.validation.BindException;
import org.springframework.web.bind.RequestUtils;
import org.springframework.web.bind.ServletRequestBindingException;
import schooldatamanager.spring.entities.*;
import schooldatamanager.spring.beans.*;

public class EnterStudentController extends SimpleFormController {

   public EnterStudentController() {
      setSessionForm(true);
      setBindOnNewForm(true);
   }
         @Override
         protected ModelAndView showForm(HttpServletRequest request,
                        HttpServletResponse response,
                        BindException errors,
                        Map controlModel)
                  throws Exception
         {
                  HashMap model = new HashMap();
                  try
                  {
                           ArrayList<Department> departments =
DepartmentDBHelper.getDepartments();
```

```
                    ModelAndView mav =  new
ModelAndView(getFormView());
                    mav.addObject("departments", departments);
                    mav.addObject("student", new Student());
                    return mav;
                }
                catch(Exception ex)
                {
                    ModelAndView mav =  new ModelAndView("error");
                    mav.addObject("errorMessage","could not bring
departments.");
                    return mav;
                }
        }
    @Override
    public ModelAndView onSubmit(Object commandObject) {

        Student student = (Student) commandObject;
        int ID = student.getID();
        int studentId = student.getStudentId();
        String studentName = student.getStudentName();
        byte age = student.getAge();
        short departmentId = student.getDepartmentId();

        Map model = new HashMap();
        if (perform(student)) {
            ModelAndView mav = new ModelAndView(getSuccessView());
            mav.addObject("student", student);
            return mav;
        } else {
            ModelAndView mav = new ModelAndView(getFormView());
            mav.addObject("errorMessage", "There is an error.");
            mav.addObject("student", student);
```

```
            return mav;
        }
    }

    public boolean perform(Student student) {

        int ID = student.getID();
        int studentId = student.getStudentId();
        String studentName = student.getStudentName();
        byte age = student.getAge();
        short departmentId = student.getDepartmentId();

        try {
            StudentDBHelper.newStudent(student);
            return true;
        } catch (Exception ex) {
            return false;
        }
    }
}
```

SearchStudentController.java

```
package schooldatamanager.spring.controllers;

import schooldatamanager.spring.beans.*;
import schooldatamanager.spring.entities.*;

import java.util.HashMap;
import java.util.Map;
import java.util.ArrayList;
import java.util.List;
```

```
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.AbstractController;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class SearchStudentController extends AbstractController {

   protected ModelAndView handleRequestInternal(HttpServletRequest request,
HttpServletResponse response) throws Exception {

      Map myModel = new HashMap();
      String criteria = request.getParameter("criteria");
      List al = new ArrayList();
      try {
         if (criteria != null && !criteria.equals("")) {
            if (criteria.equals("ID")) {
               al =
StudentDBSearchHelper.findByID(Integer.parseInt(request.getParameter("searchstr
ing")));
               myModel.put("list", al);
            } else if (criteria.equals("studentId")) {
               al =
StudentDBSearchHelper.findByStudentId(Integer.parseInt(request.getParameter("s
earchstring")));
               myModel.put("list", al);
            } else if (criteria.equals("studentName")) {
               al =
StudentDBSearchHelper.findByStudentName(request.getParameter("searchstring")
);
               myModel.put("list", al);
            } else if (criteria.equals("age")) {
               al =
StudentDBSearchHelper.findByAge(Byte.parseByte(request.getParameter("searchs
```

```
tring")));
                    myModel.put("list", al);
                } else if (criteria.equals("departmentId")) {
                    al =
StudentDBSearchHelper.findByDepartmentId(Short.parseShort(request.getParamet
er("searchstring")));
                    myModel.put("list", al);
                }
            } else {
            }
        } catch (NumberFormatException nfe) {
            Map err = new HashMap();
            err.put("eo", "Sorry, you have entered an invalid number...");
            return new ModelAndView("error", "err", err);
        } catch (IndexOutOfBoundsException ioobe) {
            Map err = new HashMap();
            err.put("eo", "No such return value...");
            return new ModelAndView("error", "err", err);
        } catch (Exception e) {
            Map err = new HashMap();
            err.put("eo", "Unspecified error...");
            return new ModelAndView("error", "err", err);
        }
        return new ModelAndView("StudentSearchPage", "searchresult", myModel);

    }
}
```

DeleteDepartmentController.java

```
package schooldatamanager.spring.controllers;
```

```
import schooldatamanager.spring.beans.*;
import schooldatamanager.spring.entities.*;

import java.util.HashMap;
import java.util.Map;
import java.util.ArrayList;
import java.util.List;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.AbstractController;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class DeleteDepartmentController extends AbstractController {

   protected ModelAndView handleRequestInternal(HttpServletRequest request,
HttpServletResponse response) throws Exception {

      try {
         String ID = request.getParameter("ID");
         if (ID != null && !ID.equals("")) {
            int id = Integer.parseInt(ID);
            DepartmentDBHelper.deleteDepartment(id);
         } else {
            Map err = new HashMap();
            err.put("eo", "Be sure your came here via a suitable way.");
            return new ModelAndView("error", "err", err);
         }
      } catch (Exception e) {
         Map err = new HashMap();
         err.put("eo", "Database error, please try again later" + e.getMessage());
         return new ModelAndView("error", "err", err);
      }
```

```
        return new ModelAndView("DepartmentSearchPage");
    }
}
```

StuffDBHelper.java

```
package schooldatamanager.spring.beans;

import java.sql.*;
import schooldatamanager.spring.entities.*;
import java.util.*;

public class StuffDBHelper {

    private String propertyOne;

    public void setPropertyOne(String value) {
        propertyOne = value;
    }

    public String getPropertyOne() {
        return propertyOne;
    }

    public StuffDBHelper() {

    }

    public static Statement statementFormer() {

        Connection connection;
        try {
```

```
            Class.forName("org.hsqldb.jdbcDriver");
            connection = DriverManager.getConnection("jdbc:hsqldb:file:C:/Program
Files/Apache Software Foundation/Tomcat
6.0/webapps/SchoolDataManager/db/School", "sa", "");
            return connection.createStatement();
        } catch (Exception ex) {
            return null;
        }
    }

    public static Stuff getStuff(int ID) throws SQLException {

        Statement statement = StuffDBHelper.statementFormer();
        ResultSet myTable = statement.executeQuery("Select * From Stuffs WHERE
ID = " + ID);
        Stuff stuff = new Stuff();
        if (myTable.next()) {
            stuff.setStuffId(myTable.getInt("stuffId"));
            stuff.setStuffName(myTable.getString("stuffName"));
            stuff.setStuffAge(myTable.getByte("stuffAge"));
            stuff.setTitleId(myTable.getByte("titleId"));
            stuff.setDepartmentId(myTable.getInt("departmentId"));
        } else {
            stuff = null;
        }
        statement.execute("SHUTDOWN");
        return stuff;

    }

    public static void deleteStuff(int ID) throws SQLException {

        Statement statement = StuffDBHelper.statementFormer();
```

```
        String deleteQuery = "DELETE FROM Stuffs WHERE ID = " + ID;
        System.out.println(deleteQuery);
        statement.execute(deleteQuery);
        statement.execute("SHUTDOWN");

    }

    public static void newStuff(Stuff stuff) throws SQLException {

        Statement statement = StuffDBHelper.statementFormer();
        String insertQuery = "INSERT INTO Stuffs
(stuffId,stuffName,stuffAge,titleId,departmentId)";
        insertQuery += " VALUES(";
        insertQuery += stuff.getStuffId() + ",";
        insertQuery += "'" + stuff.getStuffName() + "',";
        insertQuery += stuff.getStuffAge() + ",";
        insertQuery += stuff.getTitleId() + ",";
        insertQuery += stuff.getDepartmentId() + ")";
        System.out.println(insertQuery);
        statement.execute(insertQuery);
        statement.execute("SHUTDOWN");

    }

    public static void updateStuff(Stuff stuff) throws SQLException {

        Statement statement = StuffDBHelper.statementFormer();
        String updateQuery = "UPDATE Stuffs SET";
        updateQuery += " stuffId = " + stuff.getStuffId() + ",";
        updateQuery += " stuffName = '" + stuff.getStuffName() + "',";
        updateQuery += " stuffAge = " + stuff.getStuffAge() + ",";
        updateQuery += " titleId = " + stuff.getTitleId() + ",";
        updateQuery += " departmentId = " + stuff.getDepartmentId() + "";
```

```
            updateQuery += " WHERE ID = " + stuff.getID();
            statement.execute(updateQuery);
            statement.execute("SHUTDOWN");

        }
    }
```

StuffCourseDBSearchHelper.java

```
    package schooldatamanager.spring.beans;

    import java.sql.*;
    import schooldatamanager.spring.entities.*;
    import java.util.*;

    public class StuffsCourseDBSearchHelper {

        private String propertyOne;

        public void setPropertyOne(String value) {
            propertyOne = value;
        }

        public String getPropertyOne() {
            return propertyOne;
        }

        public StuffsCourseDBSearchHelper() {

        }
```

```
    public static ArrayList<StuffsCourse> findByID(int ID) throws SQLException {

        Statement statement = StuffsCourseDBSearchHelper.statementFormer();
        ArrayList<StuffsCourse> resultValue = new ArrayList<StuffsCourse>();
        ResultSet myTable = statement.executeQuery("Select * From StuffsCourses
WHERE ID = " + ID);
        while (myTable.next()) {
            StuffsCourse stuffsCourse = new StuffsCourse();
            stuffsCourse.setID(myTable.getInt("ID"));
            stuffsCourse.setStuffId(myTable.getInt("stuffId"));
            stuffsCourse.setCourseId(myTable.getInt("courseId"));
            resultValue.add(stuffsCourse);
        }
        statement.execute("SHUTDOWN");
        return resultValue;

    }

    public static ArrayList<StuffsCourse> findByStuffId(int stuffId) throws
SQLException {

        Statement statement = StuffsCourseDBSearchHelper.statementFormer();
        ArrayList<StuffsCourse> resultValue = new ArrayList<StuffsCourse>();
        ResultSet myTable = statement.executeQuery("Select * From StuffsCourses
WHERE stuffId = " + stuffId);
        while (myTable.next()) {
            StuffsCourse stuffsCourse = new StuffsCourse();
            stuffsCourse.setID(myTable.getInt("ID"));
            stuffsCourse.setStuffId(myTable.getInt("stuffId"));
            stuffsCourse.setCourseId(myTable.getInt("courseId"));
            resultValue.add(stuffsCourse);
        }
        statement.execute("SHUTDOWN");
```

```
        return resultValue;
    }

    public static ArrayList<StuffsCourse> findByCourseId(int courseId) throws
SQLException {

        Statement statement = StuffsCourseDBSearchHelper.statementFormer();
        ArrayList<StuffsCourse> resultValue = new ArrayList<StuffsCourse>();
        ResultSet myTable = statement.executeQuery("Select * From StuffsCourses
WHERE courseId = " + courseId);
        while (myTable.next()) {
            StuffsCourse stuffsCourse = new StuffsCourse();
            stuffsCourse.setID(myTable.getInt("ID"));
            stuffsCourse.setStuffId(myTable.getInt("stuffId"));
            stuffsCourse.setCourseId(myTable.getInt("courseId"));
            resultValue.add(stuffsCourse);
        }
        statement.execute("SHUTDOWN");
        return resultValue;

    }

    public static Statement statementFormer() {

        Connection connection;
        try {
            Class.forName("org.hsqldb.jdbcDriver");
            connection = DriverManager.getConnection("jdbc:hsqldb:file:C:/Program
Files/Apache Software Foundation/Tomcat
6.0/webapps/SchoolDataManager/db/School", "sa", "");
            return connection.createStatement();
        } catch (Exception ex) {
            return null;
```

```
        }
    }
}
```

EnterStudentValidator.java

```
package schooldatamanager.spring.beans;

import org.springframework.validation.Validator;
import org.springframework.validation.ValidationUtils;
import org.springframework.validation.Errors;

import schooldatamanager.spring.entities.Student;
public class EnterStudentValidator implements Validator {

    private static final int MINIMUM_LENGTH = 6;

    public boolean supports(Class clazz) {
        return clazz.equals(Student.class);
    }

    public void validate(Object target, Errors errors) {
        Student student = (Student) target;
        if (!(student.getAge() >= 17)) {
            errors.rejectValue("age", "erros.field.invalidvalue",
                    null, "age is invalid.");
        }
    }
}
```

StuffSearchPage.jsp

```
<%@ include file="/WEB-INF/jsp/includes.jsp" %>
<%@ include file="/WEB-INF/jsp/header.jsp" %>

<center>
   <h2>StuffSearchPage</h2>
   <br />
</center>

<%@ include file="/WEB-INF/jsp/footer.jsp" %>
<center>
   <form name="form1" method="post" action="">
      <table>
         <tr>
            <td>
               Look for
            </td>
            <td>
               <input type="text" name="searchstring">
            </td>
            <td>
               in
            </td>
            <td>
               <select name="criteria" id="criteria">
                  <option value="ID">ID</option>
                  <option value="stuffId">stuffId</option>
                  <option value="stuffName">stuffName</option>
                  <option value="stuffAge">stuffAge</option>
                  <option value="titleId">titleId</option>
                  <option value="departmentId">departmentId</option>
               </select>
```

```
            </td>
            <td>
               <input type="submit" name="Submit" value="Search">
            </td>
         </tr>
      </table>
   </form>
   <table style="background:white">
      <tr>
         <td>ID</td>
         <td>STUFFID</td>
         <td>STUFFNAME</td>
         <td>STUFFAGE</td>
         <td>TITLEID</td>
         <td>DEPARTMENTID</td>
      </tr>       <tr>
      <c:forEach items="${searchresult.list}" var="stuff">
         <tr>
         <td style="background:#992222;color:white">
         <c:out value="${stuff.ID}"/>
         <td style="background:#992222;color:white">
         <c:out value="${stuff.stuffId}"/>
         <td style="background:#992222;color:white">
         <c:out value="${stuff.stuffName}"/>
         <td style="background:#992222;color:white">
         <c:out value="${stuff.stuffAge}"/>
         <td style="background:#992222;color:white">
         <c:out value="${stuff.titleId}"/>
         <td style="background:#992222;color:white">
            <c:out value="${stuff.departmentId}"/>
         </td>
         <td style="background:white">
            <b><a href="EditStuff.htm?ID=${stuff.ID}">Edit</a></b>
```

```
            </td>
            <td style="background:white">
               <b><a href="DeleteStuff.htm?ID=${stuff.ID}">Delete</a></b>
            </td>
            <tr>
            </c:forEach>
         </tr>
      </table>
   </center>
```

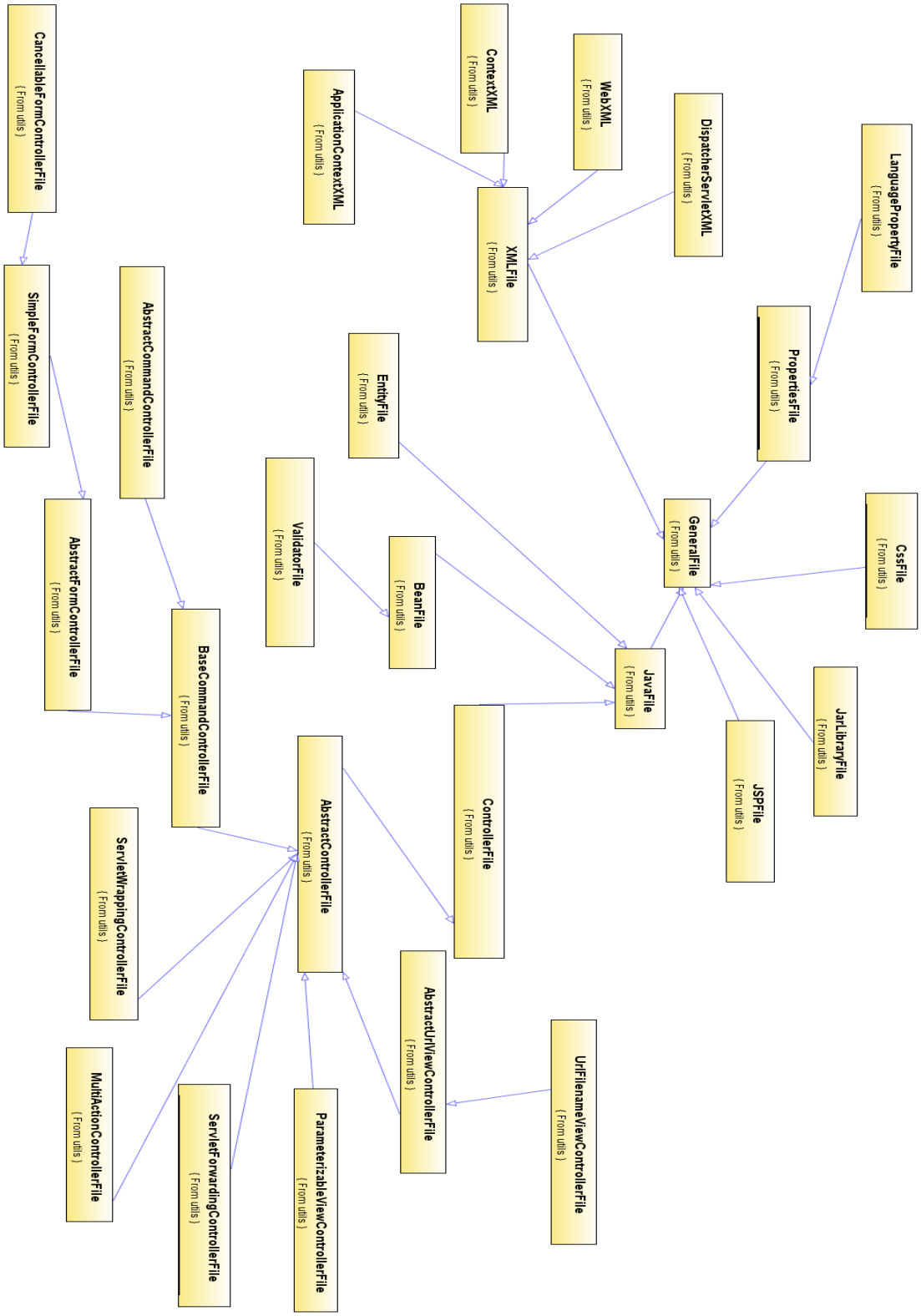
---

error.jsp

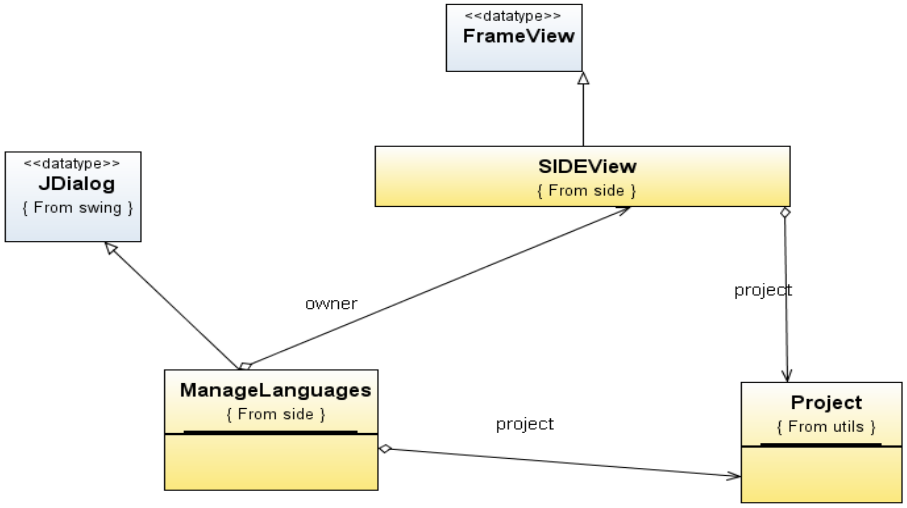
```
   <%@ include file="includes.jsp"%>

   <center>
      <c:out value="${err.eo}" default="Please check your operations. There is an
   error."/>
   </center>
   </body>
   </html>
```

B. UML DIAGRAMS

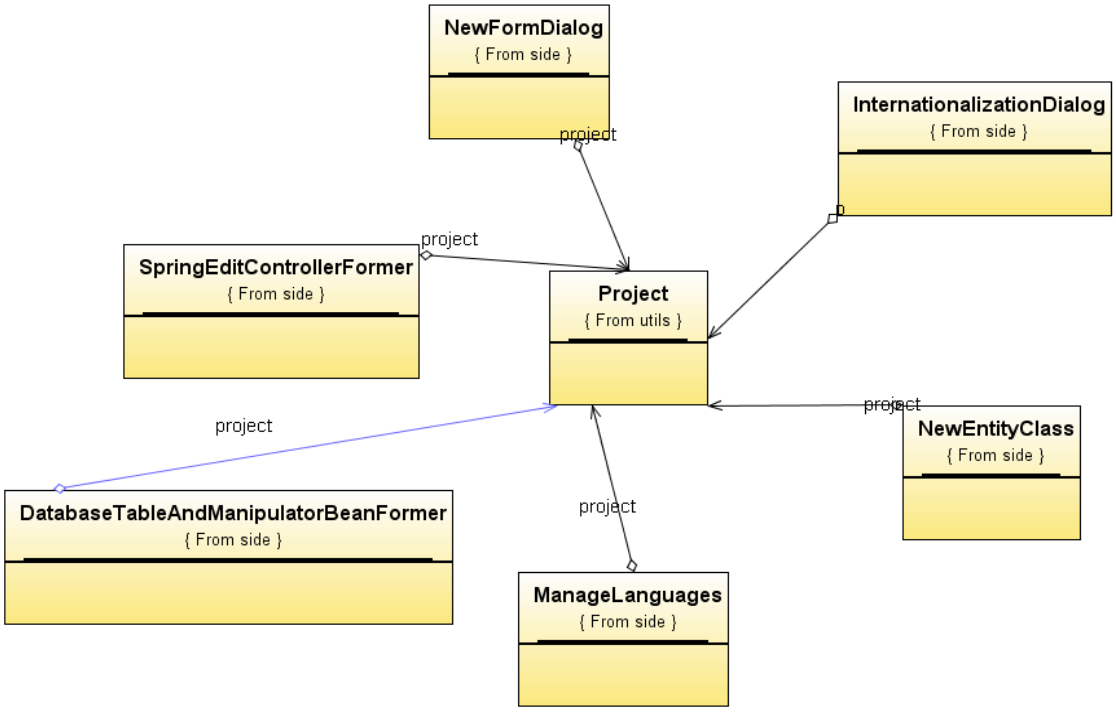
B.1. Class Diagram for File Types in a SIDE Project



B.2. Generalizations between SIDEView, Project and ManageLanguages



B.3. Associations of Graphical Classes



B.4. Aggregation Relationships in Project Class