Dear candidate,

In the following pages you will find a description of the project and the goals you have to achieve. The intention of this “homework” project is for us to understand the way you think when you develop software and to realize how easy you can grasp new technologies.

In order to be able to continue with this project you have to be aware at some level of the following technologies:

* Hibernate
* Spring
* ZK framework (www.zkoss.org)
* Web fundamentals

Most of you are already familiar with Hibernate and Spring so probably the only technology you have not seen until today is the ZK Framework. This is a very simple yet very powerful server-side Java web framework. It is used by many companies to implement and maintain mission critical web applications and the truth is that we love it. The ZK Framework utilizes both MVC and MVVM patterns. We prefer to use almost always the MVVM pattern.

Those who have experience in JavaScript libraries, ZK’s philosophy have many similarities with Angular, Knockout and Ractive.js, where web pages are driven by a data model. The only difference is that ZK runs on the server-side and communicates with the client through transparent ajax calls.

**Notice**:  
You have to follow some simple tutorials and demos from [www.zkoss.org](http://www.zkoss.org/) web site to be able to continue to the project. Please do not waste your time in MVC. Go directly to MVVM demos and samples.

Project Structure

The project is a typical eclipse web project. All libraries needed (Spring, Hibernate and ZK) are inside the **WebContent/WEB-INF/lib** so you don’t have to add any dependencies manually.

**src/cfg** contains the hibernate configuration file and a spring context provider   
**com/personnel/components** contains the domain classes   
**com/personnel/services** contains the service layer classes   
**com/personnel/ui** contains the ZK pages view models

**WebContent/personnel** contains the web pages

There is also the folder **src/servlets** containing just one servlet (Tester.java) which can be used to make some quick tests while developing your application. It is already mapped inside the web.xml with the name ‘**tester-servlet’**.

Database Structure

The project’s database is a very simple and small database with just 5 tables:

* **Employees**   
  This table contains the company’s staff. Each record in this table represents an Employee. We have already mapped the Employee class with this table:  
  /homework/src/com/personnel/components/Employee.java  
  /homework/src/com/personnel/components/Employee.hbm.xml
* **SeminarTypes**   
  The SeminarType represents a type of seminar. We could imagine the SeminarType as a seminar class where each employee may attend. Example of titles of seminar types could be: “Marketing Strategies”, “Accounting Procedures” etc
* **Seminars**   
  Each time an employee attends a specific seminar class (SeminarType) then a new record should be created in this table to keep track of the seminars each employee attends. Notice that “seminarTypeId” and “employeeId” are foreign keys in this table and these two fields can ensure the uniqueness of each record. Each employee may attend a seminar (or else SeminarType) only once.
* **Payrolls**   
  Each record represents the identity of a monthly payroll. For example: “Payroll of June”.
* **EmployeePayrolls**   
  Each record in this table represents the payroll of a specific employee for a given month.

Inside the folder **homework/docs** you will find the **database.schema.png** representing the schema of the database. We have used SQL Server 2008 to build this database but you may use any database you like for this project. In case you use either SQL Server or SQL Server Express (which is free), then hibernate is ready to use. Just change the connection string inside the hibernate.cfg.xml configuration file. The jTDS JDBC driver for SQL Server is also included in the project libraries (WEB-INF/lib).

Inside the folder **docs/database** you will find also the database scripts to create these tables. Note that these scripts are generated automatically from Sql Server Studio so you have to modify them as needed to create the tables in another RDBMS.  
If you want to restore a ready to use database you will find the backup file **homework\_db.bak** inside **docs/database** folder.

Hibernate Mappings

Hibernate mappings are described in separate **hbm.xml** files instead of annotated declarations. We believe that although hibernate annotations look simpler and easier to use, the separate hbm.xml files are much more maintainable in very large projects.

The first thing you will notice is that each table has two keys:

* A primary surrogate key with auto-numbering values generated automatically by the database system (in Sql Server this is the “identity” column).   
  In the Employees class the surrogate key is the EmployeeId.
* A business unique key which should be mapped as “natural-id” and it must be used in equals() and hashcode() methods to ensure proper handling of uniqueness inside collections.  
  In the Employee class the natural business key is the EmployeeCode.  
  Take care that in some tables (like the table Seminars) the uniqueness of the business key may be achieved using more than one column.

We strongly believe that using both a natural business key and a surrogate key for each hibernate entity gives us the power of a simpler and more maintainable database schema while ensuring the correct handling of entities inside java collections.

**Homework**

* **~~We have already mapped the Employee class for you but some fields are missing. Add the rest of the employee fields to the Employee class and Employee.hbm.xml definition.~~**
* **~~Create the mappings for the rest 4 classes. Do not forget to map also the relationships between the tables (e.g. one-to-many, many-to-one etc).~~**

The Service Layer

As we described in a previous page, the folder **/homework/src/com/personnel/services** contains the classes representing the service layer. The services are classes responsible to execute all necessary database queries and apply also any additional business logic.

Each service is a Spring managed bean. In this project all spring beans definitions should be described inside the **/homework/WebContent/WEB-INF/app-context.xml**

This project should have two services:

* The EmployeeService which is responsible to handle queries related to employees and seminars. This service is already described as a spring bean.
* The PayrollService responsible to handle the payroll queries and calculate new payrolls.

As you can see we have implemented already the most basic functionality of employee service.

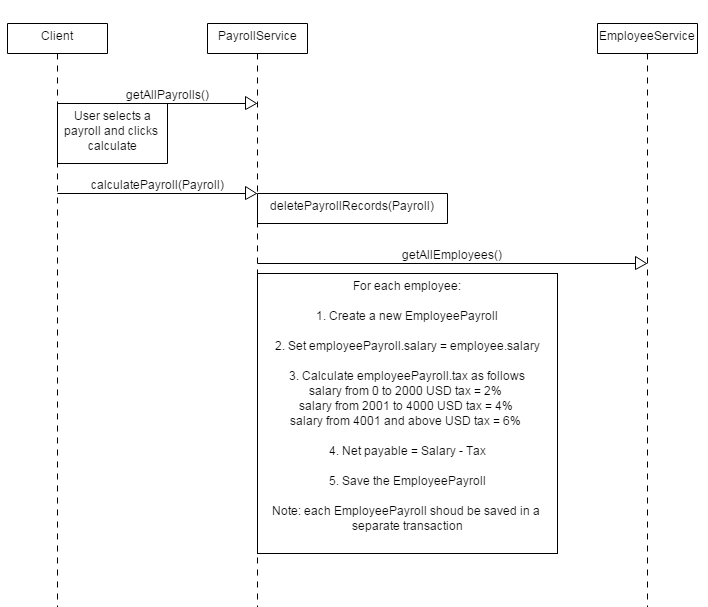
**Homework**

**Add the following methods to the EmployeeService:**

* **~~getAllSeminarTypes()~~** ~~returns a list of all available seminar types.~~
* **~~saveSeminarType(SeminarType seminarType)~~** ~~insert/update a seminar type into the database~~
* **~~deleteSeminartype(SeminarType seminarType)~~** ~~remove a seminar type from the database~~
* **~~getEmployeeSeminars(Integer employeeId)~~** ~~returns a list of all Seminars attended by the given employee~~
* **~~saveEmployeeSeminar(Seminar seminar)~~** ~~insert/update a seminar attendance into the database~~
* **~~deleteEmployeeSeminar(Seminar seminar)~~** ~~remove a seminar attendance from the database~~
* **~~getSeminarTypesForEmployee(Integer employeeId)~~** ~~returns a list of SeminarTypes for which there is no related seminar record into the Seminars table for the given Employee. This method will be useful in case we want to find which seminar types have not been attended by the given employee.~~

**Create the PayrollService class as a spring managed bean and add the following methods:**

* ~~Simple CRUD operations for the tables~~ **~~Payrolls~~** ~~and~~ **~~EmployeePayrolls~~**~~.~~
* **~~getAllPayrolls()~~**  ~~returns all Payroll records ordered by Year/Month descending order.~~
* **~~deletePayrollRecords(Payroll payroll)~~** ~~This method should be responsible to call a stored-procedure to delete all~~ **~~EmployeePayroll~~** ~~records related to the given payroll. This method will be useful making a batch calculation for all employees.~~
* **~~calculatePayroll(Payroll payroll)~~**~~This method should be responsible to create and calculate payroll records for all employees for the given payroll. You may find the sequence diagram below also as a separate file~~  
  **/homework/docs/calculate.payroll.sequence.png**



The Interface

Assuming that you already have a look at ZK web framework, you should know now that each web page consists of two components:

* The web page (ZUL file)
* The MVVM view model class responsible to “drive” this page

We have already implemented for you the employees management list and a simple employee editor. If you have setup your project correctly then you should be able to debug the page at a url similar to this:

**http://localhost:8080/homework/personnel/employees.zul**

The web pages are created inside the folder: **/homework/WebContent/personnel** while the view models are implemented under the package: **/homework/src/com/personnel/ui**

**Homework**

* **~~Add a button “Refresh” in the employees.zul to reload all employees. See if you can move the code which loads the employees list from the init() method into a separate @command method so that to be able to re-use the same method both for initialization and ‘refresh’.~~**
* **~~Add missing text fields in the employee-editor.zul (sex, salary etc)~~**
* **~~Add the “Delete” button into the employee-editor.zul and implement the appropriate command to remove an employee from the database.~~**
* **~~Create the web page seminar-types.zul (similar to employees.zul) to manage the SeminarTypes.~~**
* **~~Create the web page seminar-type-editor.zul (similar to employee-editor.zul) to edit a SeminarType.~~**
* **~~Create the web page employee-seminars.zul to display the seminars attended by a specific employee.~~**
* **~~Create the web page employee-seminar-editor.zul to add or edit a seminar attendance for a specific employee.~~**
* **Create the web page employee-payrolls.zul to display a list of all payrolls for a specific employee.**
* **Add a link inside the <grid> template of the employees.zul to open the web page employee-payrolls.zul**
* **~~Create the web page payrolls.zul to view a list of all payrolls.~~**
* **~~Create the web page payroll-editor.zul to add or edit a payroll issue.~~**
* **~~In the payrolls.zul <grid> add the button “Calculate Payroll” next to each payroll line. The user may click this button to make a batch calculation of a payroll for all employees.~~**

Inside the folder **docs/web\_pages** you will find a set of screenshots of some pages to understand the required layout.

Optional Tasks

If you have the time, you may implement a utility class (Database.java) to handle hibernate common tasks to avoid repeating code in the service methods:

* create a hibernate session if needed
* begin / commit database transactions
* execute queries
* handle hibernate exceptions and rollback transactions if needed

Another thing is the validations. As a developer you know that some fields should have certain values while some others may have special business rules. An employee for example cannot be a valid instance without an employee code, a last name and a first name.  
Which place do you think is the best place to put this code? Inside the interface or inside the service? Or maybe inside the Pojo? Find the most basic validation rules for each entity and implement this code in such a way to have the best re-usability and maintainability.

Finally, we encourage you to design and implement – if you have the time – any additional web page you believe it is required.

We wish you good luck and happy coding!