14th homework; JAVA, Academic year 2014./2015.; FER

Introduction

For this homework you will integrate the structured web-application which is described in document "`java_tecaj_10_prezentacija_uputa.txt`" which is available in Ferko's repository with a JDBC-based reimplementation of voting application which you have already developed as part of 13th homework. Let this new web application's name be *aplikacija5* so that it will be available using an URL such as http://localhost:8080/aplikacija5. Once you finish the application, you will prepare a ZIP archive of your eclipse project and upload it to Ferko.

Problem 1.

Assume you have on your disposal a database on which you connect using the following URL:
`jdbc:derby://localhost:1527/votingDB;user=ivica;password=ivo`

The exact database name, host, port, user and password will be provided in properties file `dbsettings.properties` which must be directly in `WEB-INF` folder. This file must take the following form (the right side can vary):

```
host=localhost
port=1527
name=votingDB
user=ivica
password=ivo
```

Your application should read this file during connection-pool setup and use provided information; if this file is not present or any of properties is missing, throw an exception which will case the web-application to be stopped (verify this).

Create a basic web-application which has separate presentation+service layer and separate data-access layer, as described in `java_tecaj_10_prezentacija_uputa.txt`. Please note that you will work with different data-model (here we don't write an application for blogs) so you should modifiy DAO interface to best suits your needs. A filter must be responsible for obtaining database connection from pool and for returning it. Connection passing from this filter to the actual JDBC-based DAO implementation must be done through `ThreadLocal` singleton. Initialization of connection-pool and its destroying must be performed in appropriate web-application listener.

Assume you will work with two tables created in this database by the following commands:

```
CREATE TABLE Polls
    (id BIGINT PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
    title VARCHAR(150) NOT NULL,
    message CLOB(2048) NOT NULL
);

CREATE TABLE PollOptions
    (id BIGINT PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
    optionTitle VARCHAR(100) NOT NULL,
    optionLink VARCHAR(150) NOT NULL,
    pollID BIGINT,
    votesCount BIGINT,
    FOREIGN KEY (pollID) REFERENCES Polls(id)
);
```

During the web-application startup, verify if these tables exists in database; if not, send appropriate CREATE statements to create them (*but only if they do not already exists*); if they exist, do not try to delete them. You can google it up ("apache derby create table if not exists").

The first table (**Polls**) models concrete polls. Each row represents a different poll. Each poll will have its own poll ID. For example, to mimic your previous voting-application homework problem where everything was written in files, you would have a single entry in table **Polls** with *ID*=1 (or any other), *Title*="Glasanje za omiljeni bend:" te *Message*="Od sljedećih bendova, koji Vam je bend najdraži? Kliknite na link kako biste glasali!".

If during startup web-application determines that appropriate database tables do not exist, you should create them (as previously described). If you create missing tables (or determine that table `Polls` is empty), you should also populate them with poll data which mimics the data used in homework 13, and some other poll data (so that you end up with two initial polls). Do not assume that the first INSERT will create a record with key 1; always ask for created keys and use this information. To check this behavior, once your code successfully creates the tables (and populates them), stop Tomcat, log to database using ij-console and delete all table contents (first verify how it works when using DELETE statements, and if your application on next start populates everything OK, try DROP TABLE statements and verify that tables are automatically created).

Table **PollOptions** contains options for each defined poll. If we assume that our poll had the ID with value 1, in **PollOptions** table we would have 7 rows with pollID set to 1 (attribute `votesCount` is not shown):

id	OptionTitle	optionLink	pollID
1	The Beatles	http://...	1
2	The Platters	http://...	1
3	The Beach Boys	http://...	1
4	The Four Seasons	http://...	1
5	The Marcels	http://...	1
6	The Everly Brothers	http://...	1
7	The Mamas And The Papas	http://...	1

Problem 2.

Create a servlet which is mapped to `/index.html`. This servlet must obtain a list of defined polls and render it to user as a list of clickable links. When user clicks on a Poll title, the link that will be followed must be `/glasanje?pollID=x` where `x` is selected poll ID.

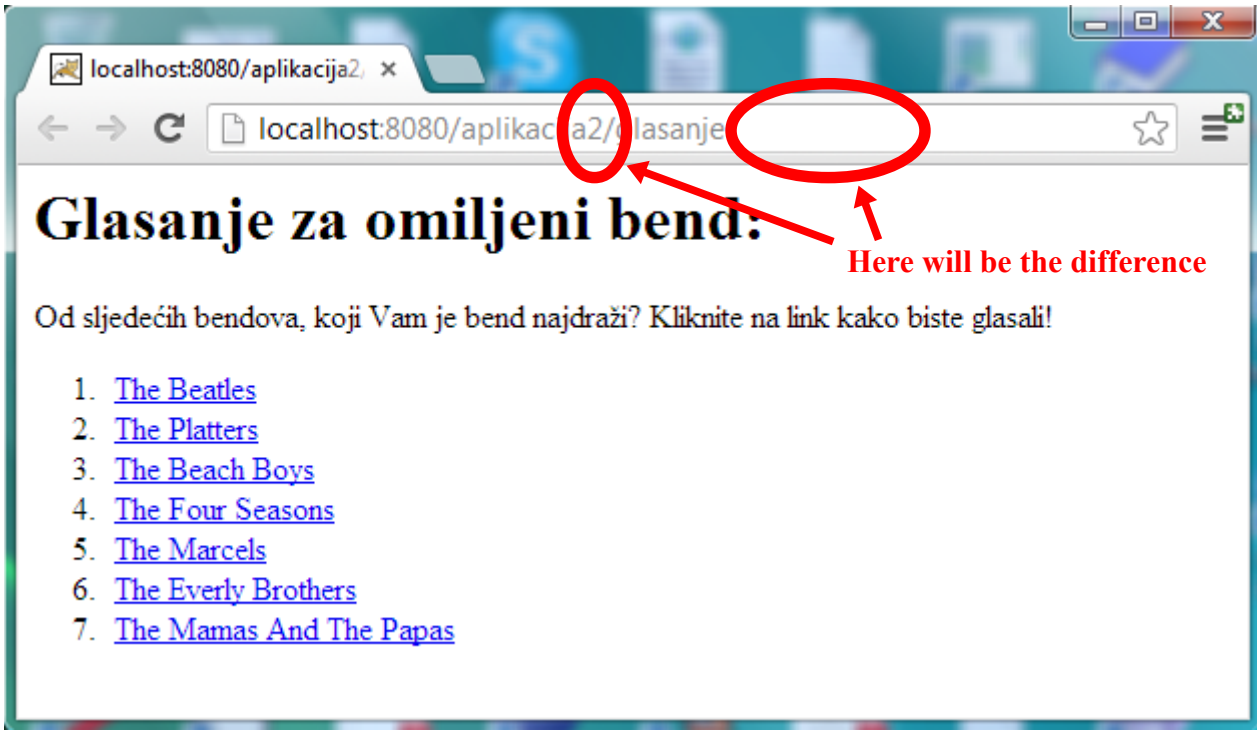
Problem 3.

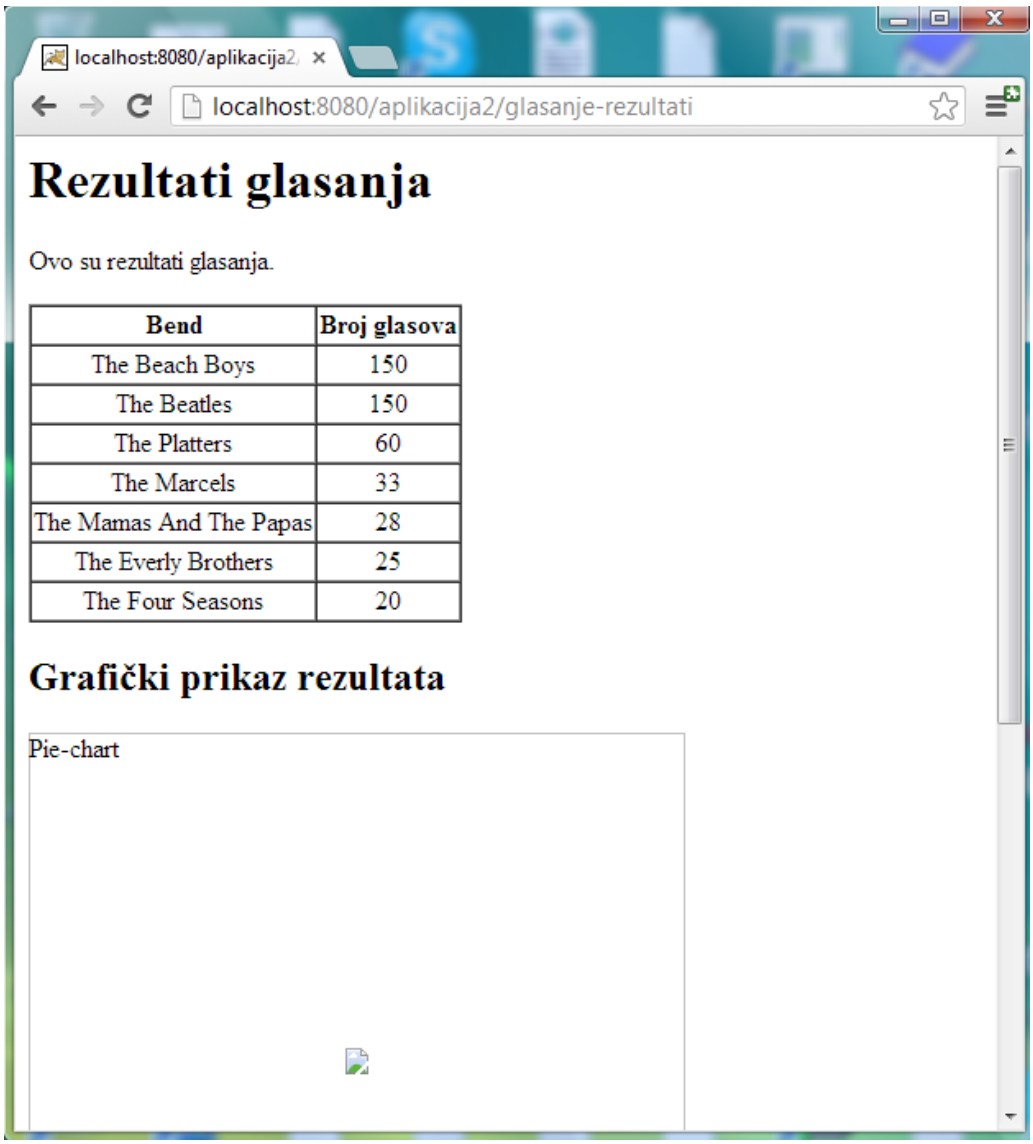
Modify all of remaining servlets you developed in problem 7 of your old homework so that they work with the poll and the poll options defined in database instead of in text files. This includes already mentioned servlet `/glasanje` which has to get the `pollID` identifier in each request so that it knows which poll to offer.

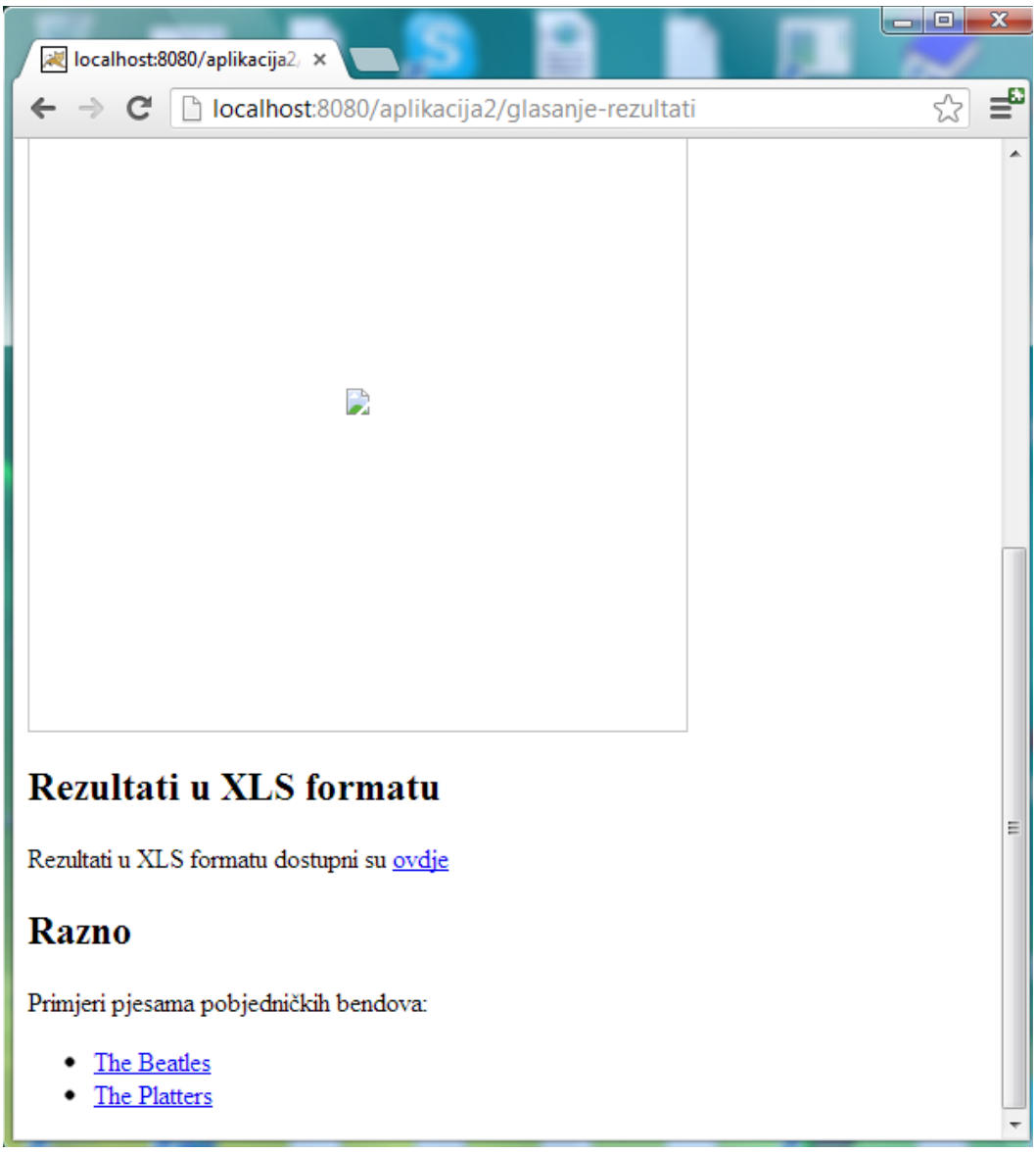
Once completed, the application screenshots should look the same as in old homework (screenshots are repeated here). The only difference will be in URL that will contain `/aplikacija5` and additional parameter that defines a concrete poll we work with. For recording the number of obtained votes use attribute

`votesCount` of table **PollOptions**.

Here are the screenshots.




localhost:8080/aplikacija2/
localhost:8080/aplikacija2/glasanje-rezultati
Rezultati glasanja
Ovo su rezultati glasanja.
Bend
Broj glasova
The Beach Boys
150
The Beatles
150
The Platters
60
The Marcels
33
The Mamas And The Papas
28
The Everly Brothers
25
The Four Seasons
20
Grafički prikaz rezultata
Pie-chart



The generation of XLS document and graphics must also work as expected.

Please note. You can consult with your peers and exchange ideas about this homework *before* you start actual coding. Once you open you IDE and start coding, consultations with others (except with me) will be regarded as cheating. You can not use any of preexisting code or libraries for this homework (whether it is yours old code or someones else), unless it is one of the libraries or your old homework I explicitly mentioned in previous problems. Exception is the testing: if you wish to use some library which will make testing simpler, you can use it. Document your code!

In order to solve this homework, create a blank Eclipse Java Project and write your code inside. Once you are done, export project as a ZIP archive and upload this archive on Ferko before the deadline. Do not forget to lock your upload or upload will not be accepted.

Equip the project with appropriate `build.xml`. You must add `war` target that will automatically create complete WAR file.

You are required to create at least one unit test (for whatever you wish).

Before uploading, please make double sure that a working WAR can be build from console by `ant`. Please take special care not to embed any absolute paths in your code or in scripts – different users will have tomcat installed at different places. Your project name must be `HW14-`*yourJMBAG*.

The deadline for uploading and locking this homework is June, 20th 2015. at 07:00 AM (morning!).