



Web based database backup with JDBC

Authors:

Tymoteusz Gedliczka

Marcin Grabda

Tomasz Gurgul

Wiktor Kołodziej

Jakub Suder

1 Team

Wiktor Kołodziej team leader, public relations, build administrator

Tomasz Gurgul scribe, code developer, tester

Marcin Grabda code developer, GUI developer, build administrator

Jakub Suder code developer, tester

Tymoteusz Gedliczka code developer, tester

2 Theme of the project

Web based database backup with jdbc.

3 Description of the problem

Let us imagine a computer system that makes use of a database (it may also use different types of databases, for example MySQL, Postgresql, MSSQL). Difficulties appear when there comes a need to create a backup of data and the only technology you can use is JDBC (e.g. we would like to integrate this functionality with an existing, more complex system). There are many applications providing backup facilities, however none of them fulfils our goals (different database types support, use of JDBC).

4 Proposed solution

We have decided to create a system, that would enable its administrator to:

- log on to the system through www interface

The administrative panel that enables you to create a back-up copy of a database will be available only after the authentication. In order to gain access to this panel user will have to enter login and password. When the process of authorisation succeeds you will be allowed to perform administrative activities. SSL tunneling will be used to provide safety for the entered data.
- create configuration file that would include a list of databases with all the necessary information, such as address, login, password, etc. The system has to store information that is necessary to establish connection and then create a back-up of a database:
 - name of database
 - type of database
 - server's address and port
 - login
 - password

After having logged to the system, user has to provide either all of the above data or just part of it using the connection's URL that is characteristic for the database engine. The system stores this data in a form of a configuration file which is used to establish connections. The usage of configuration file has some advantages, such as storing information about all the databases you tend to create backups of. Thanks to this file you do not need to enter the same information every time a back-up is created, all you have to do is select the proper database from the list. The system also takes the possibility of editing this configuration into account.

- connect to database and create a back-up file that can be saved on the local machine

In order to begin the process user selects database and then chooses the option "Backup". System reads the proper data from the configuration file and tries to establish connection with the selected database. If the connection fails, system generates message informing about the probable cause of the failure, for example wrong data, unrecognized type of database inaccessible server.

After the successful connection with database, system begins to create backup file by reading data from database tables. Unless you choose different option, data will be stored in a form of binary file. The back-up file can be saved on the local machine to the directory path that has been stored in the configuration file with all other data. As the need arises, the file can be compressed. You can also choose whether you want to be informed about the fact that backup

has been created. Since databases can be very large, this process can turn out to be time-consuming, so you may decide to use the notification option, then a proper email message will be sent after the backup is complete. The full name of the backup file will be as follows: <number_base_id>_<base_name>_<date>_<format>.<extension>, where:

- number_base_id - base's number taken from the file that contains the list of databases
 - base_name - the name ("title") of the base with slight changes (e.g. "-" substituted for space so the name can be used as a file name)
 - date - backup's creation date in ddmmyyyyhhmm format
 - format - e.g. "binary"
 - extension - zip or gz
- to reconstruct databases by means of the backup file whenever it is necessary
- The second main function of the module is the reconstruction of a database by means of a backup file. You can select the database that needs to be dumped.
- The back-up file is a common SQL script and therefore reconstruction of databases does not cause problems. Provided that the script is coherent, the dump process comes down to deleting all the tables and restoring the data from the script.
- To obtain that goal, administrator chooses the option "Restore" and enters the name of the proper back-up file.
- log off the system

5 User's web interface

JDBDump system is accessed through a WWW interface. As the old saying goes, the picture is worth a thousand words. That's why we want to present the interface by means of the following screenshots:

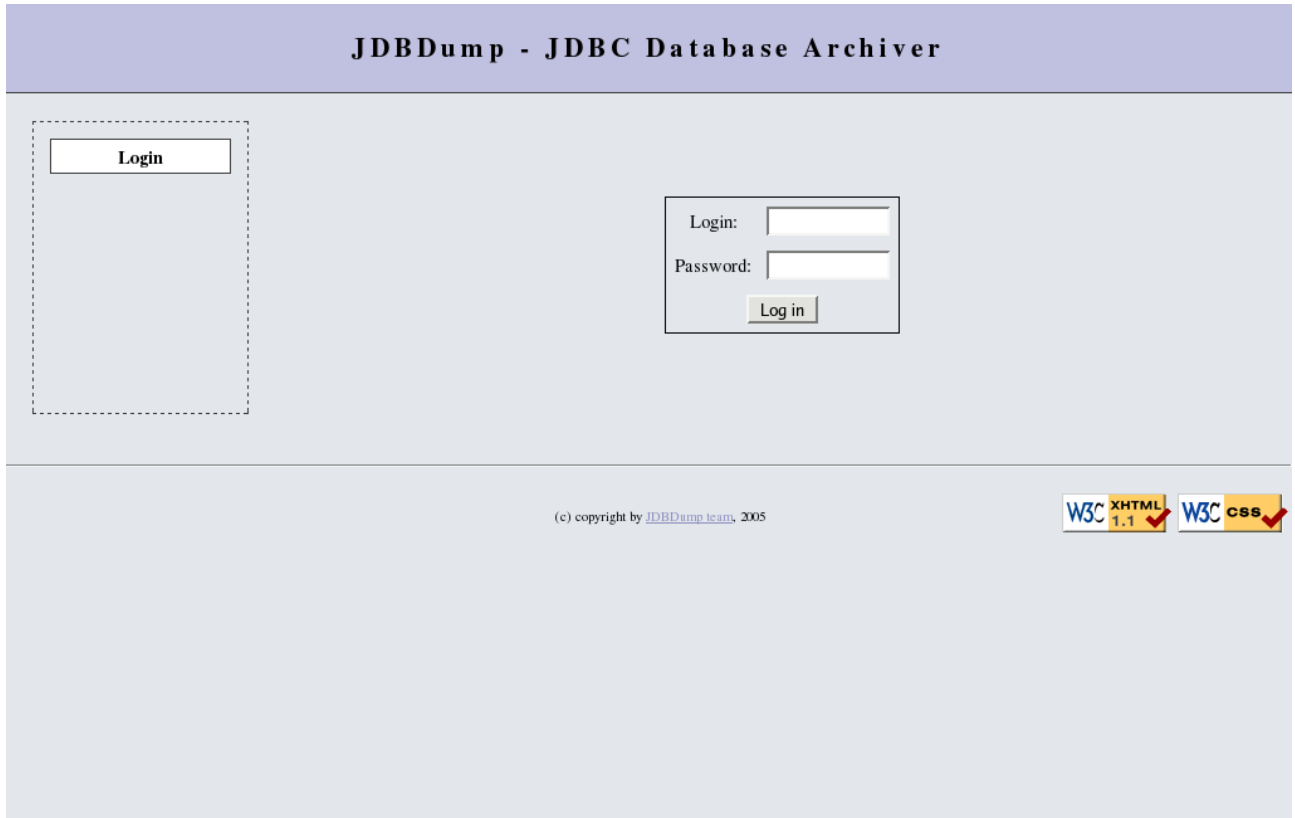


Figure 1: Logging to the system

JDBDump - JDBC Database Archiver

Database list

Add database






Backup

Backup list

Restore

Settings

Logout

Forum phpBB	mysql.agh.edu.pl	 Edit	 Delete
Homepage data	mysql.agh.edu.pl	 Edit	 Delete
Clients database	cassiopea.net.autocom.pl	 Edit	 Delete

(c) copyright by [JDBDump team](#), 2005



Figure 2: The list of databases

JDBDump - JDBC Database Archiver

Database list
Add database
Backup
Backup list
Restore
Settings
Logout

Database title:	<input type="text"/>
<small><i>This name will be used only on the list in JDBDump. Use a descriptive name, e.g. "Main company database".</i></small>	
Connection URL:	<input type="text"/>
<small><i>(optional) You may enter a database-specific connection URL instead of entering server name, port, etc. separately.</i></small>	
Server name or IP:	<input type="text"/>
Server port:	<input type="text"/>
<small><i>Leave empty to use a default port of this DBMS.</i></small>	
Database engine:	<input type="text" value="MySQL"/>
Database name:	<input type="text"/>
<small><i>Name of the database used on the server.</i></small>	
Login:	<input type="text"/>
Password:	<input type="password"/>
Password (again):	<input type="password"/>
<input type="button" value="Add database"/>	

Figure 3: Adding database



Figure 4: Creating back-up

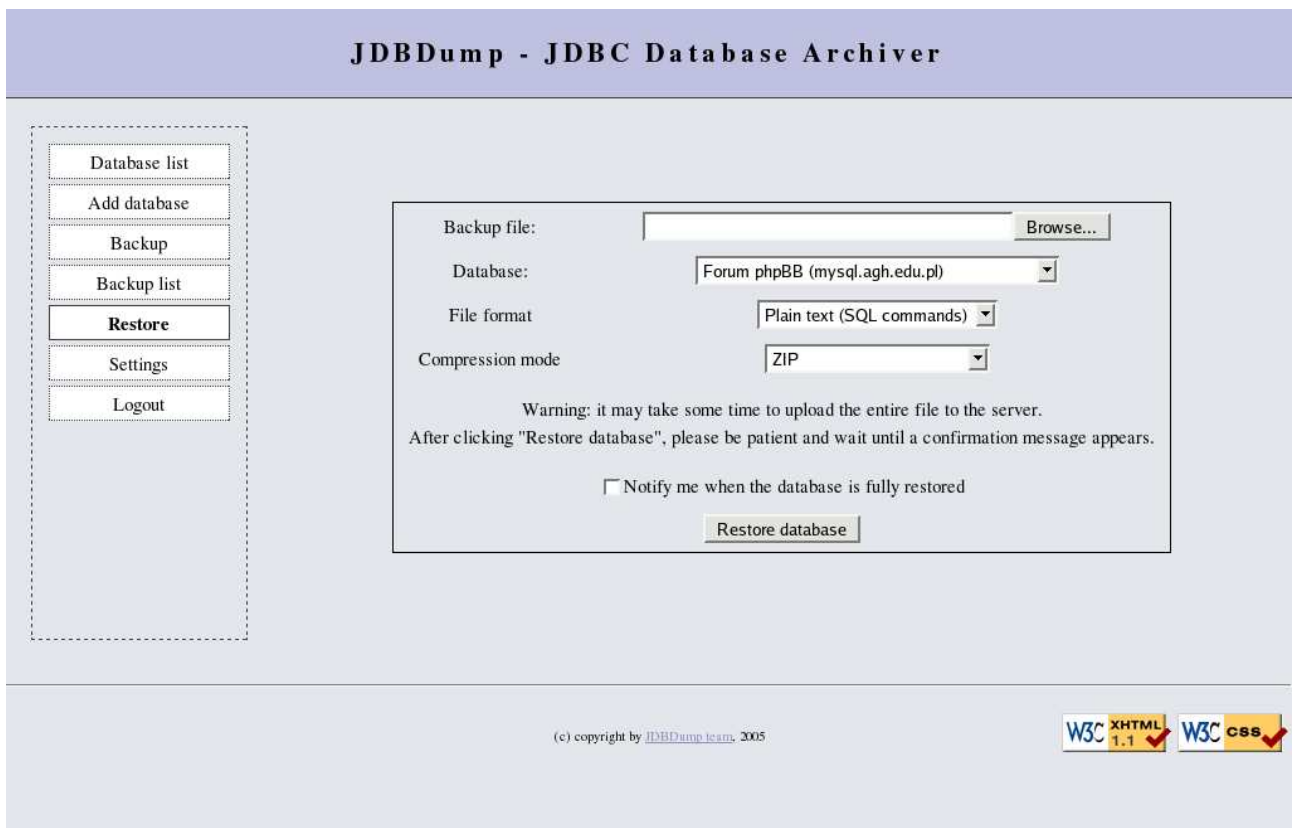


Figure 5: Database dumping

6 Possible problems with implementation and decisions

- saved data format

In order to make the reconstruction of a database possible, we are going to use our own binary format. However it is not the only option. We have decided to use highly flexible approach and let our system handle other data formats when an appropriate plugin is added. When the proper plugin is loaded, the system will be able to create the image of a database in a form of a text file as standard SQL commands.

- file compression (optional)

Databases tend to store huge amount of data, lots of megabytes let's say. As a result, the output back-up file will be probably quite large, regardless of the data format chosen by the user. Downloading such large files is very uncomfortable, especially when your internet connection is not very fast. If you consider the content of a back-up, it is easy to notice that it mostly contains repeatable strings that make it perfect to compress. Java makes files compression very easy. Since the compression functions are available through the standard java package `java.util.zip` you do not need any external libraries. There are two compression algorithms: one uses ZIP format, the other - GZIP. To make the behaviour of the system more flexible, we have decided to let the user chose which of these methods he prefers. User can also reject this offer and create an uncompressed file if its size is not too big. Distinguishing between zip and gzip formats has been caused by different preferences of Windows and linux administrators: those working under Windows would probably prefer *.zip file whereas those working under linux would probably choose *.gz.

- protection of back-up file and connection against the unauthorized access - data encryption

The output file will be optionally encrypted. However the easiest way to obtain safety will be the usage of the system in conjunction with the ssl tunnel. This method is highly recommended.

- selection of DBMS our application works with

Database servers use many different relational database management systems. Although all of them implement common SQL standard, there are some slight differences among the implementation of some details and the possibilities each one provides, for example the most popular database management MySQL had not been able to work with views, storage procedures and triggers till last October. Therefore, one cannot write one universal module that would download data from all database types. We can of course restrict to some ranges of elements common for all databases, but this approach would result in data incompleteness and make such dump completely useless. We can also choose just a few database types, we plan to work with and then make an individual approach towards each of them. We have decided to create a part of the system responsible for downloading data so as to minimize the implementation time and include as many datatypes as possible. To achieve that point, a special group of classes will be used:

- one main class, which will implement all possible functions in the most general way
- several inherited classes that will make some queries in a way suitable for a given database type, for example GeneralConnection class and MySqlConnection, PostgreSQLconnection subclasses

Owing to this solution, functions identical for all databases can be implemented just once in the main class, whereas all other specific functions can be implemented in subclasses. User can also make advantage of this main class when he wants to receive a backup of a database our system does not support. However we can not guarantee the success of such an operation.

In order to give users the possibility to expand the number of system's database types, we are going to make the system create the list of classes that deal with databases dynamically by searching through the appropriate package to find classes which inherit from the main connection class. As a result, you will not need to modify original *.jar files. All you need to do is provide your own jar package with your own classes. The system should detect them.

- version of java: 1.5, because it provides many additions which make coding more efficient and comfortable (like generics and enhanced for loop („foreach” loop))
- the use of JNDI to connect with databases

JNDI takes care of database connections, the application connects to databases by taking a DataSource from JNDI

- the order of actions during system reconstruction - avoidance of conflicts

At the beginning system creates the structure of tables, then fills the tables with data, and in the last step it adds constraints.

- limited JDBC capabilities

In order to recognize tables' names in a database we are going to use so called metadata (DatabaseMetaData class).

7 Diagrams

This documentation involves the following diagrams: class diagram, use-case diagram and sequence diagrams. To achieve better legibility, they have been placed in the supplement to this document.

8 Design patterns used in the project

8.1 Singleton

Singleton means that you want one and only one instance of an object within one program. It is used when creating more than one instance of a given class is inappropriate.

List of classes that implement this pattern:

- Configuration - the class that keeps the configuration, we need only one such object
- DatabaseConnectionFactory - the class that produces connection with the given type of datatype on request.

8.2 MVC - Model-View-Controller

One of the first patterns that were described [Krasner and Proper, 1988], the idea of this pattern is to separate user's interface from data model. Struts plays this role in our project.

8.3 Factory

A simple class that returns an instance of one of several possible classes depending on the data provided to it. This pattern is implemented by DatabaseConnectionFactory - a class that produces connection with the given type of database.

8.4 Builder

A Builder is an object which assembles a number of other objects (such as GUI widgets) in a way determined by the data provided to it, to form one bigger object (such as a panel or a window in GUI). In our project, we can say that the DatabaseConnection class is a Builder, because it builds a Dump object out of many objects like Table or Column, in various ways depending on the type of the database it connects to.

8.5 Façade

The idea of Façade is to hide the complex part of a system or a framework and make it easier to use. In our case this means hiding the steps of creating and reconstructing a database in methods like dump() or restore() in DatabaseConnector class. Other classes e.g. those handling the GUI don't have

to handle the dump objects or database connection manually (they don't have to access JDBC classes directly at all).

8.6 Observer

Observer implements an interface with methods to notify itself about some activities. We use observer pattern to notify administrator when backup is ready.

8.7 Proxy

Proxy is an object that act as an object that is being downloaded over network for a long time. Proxy supplies information about this object at once. We will use proxy pattern while dump is being done, because this will be usually very long operation.

8.8 Strategy

The Strategy pattern is used when we want to be able to do the same action in a few slightly different ways, like save a file using one of several formats or calculate a value using various algorithms. In Jdbdump, a strategy is used to select the preferred format of the dump file and the preferred compression algorithm (Gzip, Zip or none).

8.9 Template

Template is basically what an abstract class in Java is for - to define a method in a parent class, but leave it for deriving classes to implement them fully. In our project this pattern is present in the hierarchy of DatabaseConnector and other specific Connectors.

9 Technologies used

9.1 sourceforge.net

We have decided to run our project on sourceforge.net. This service gives you such facilities as www account, mailing list, cvs account or bug reporting system. Sourceforge is famous for thousands of great projects created by people from all over the world. We hope our project will join those that turned out to be successful. Our project's name on sourceforge is jdbdump.

9.2 Maven

We have used Maven to generate the structure of source, tests and documentation directories and to generate our project's website. The address of this website is:

<http://jdbdump.sourceforge.net>

9.3 CVS

We have also benefited from CVS that is available on sourceforge. The current state of the repository can be seen here:

<http://cvs.sourceforge.net/viewcvs.py/jdbdump/>

9.4 UML tools

We have used two tools to make our diagrams:

- Poseidon - it works on different platforms (Java code), but on the other hand uses lots of memory. By means of Poseidon we have created class and use-case diagrams. Unfortunately, the program has not been able to save our sequence diagrams, so we had to find another tool.

<http://gentleware.com/>

- Enterprise Architect - a very fast tool that works with windows and linux Wine emulator. By means of EA we have created sequence diagrams.

<http://www.sparxsystems.com.au/>

10 Requirements

10.1 Functional requirements

1. All interaction with the system is done through a web interface.
2. System grants the user access to the system if a correct login and password is entered.
3. On the „database list” page, system presents a list of all configured databases.
4. The list of databases, as well as the user’s login, password and other data, are read from a configuration file.
5. User can add a new database to the list.
6. User can select a database engine from a list of all available connectors while adding a new database.
7. User can edit data of a previously entered database.
8. User can remove a database from the list.
9. User can order the system to do a backup of a database selected from the list of all configured databases.
10. User may specify the format of the backup file and used compression method.
11. The system downloads the entire database structure and data and stores it in a file of the selected type in a designated directory on the server.
12. A notification e-mail is sent to the user when the backup file is ready for downloading, if the user requests notification earlier.
13. User can view a list of all backup files stored on the server; files can be downloaded or deleted from the list.
14. User can order the system to restore a selected database from a backup file, either stored on the server, or uploaded from a local computer.
15. The system recreates the database using the supplied file.
16. A notification e-mail is sent to the user when the database is fully restored, if the user requests notification earlier.
17. The system allows the user to change their e-mail address used for notification.

18. The system allows the user to change their password.
19. The system allows the user to change the path of the directory on the server, in which backup files are stored.
20. System's functionality may be extended later by providing additional custom connector classes for new database engines.

10.2 Security requirements

1. System denies access if login or password doesn't match.
2. System properly closes session after user's logout, so that he's not logged in anymore
3. Configuration file is not available via www to other users
4. Dump files are not available directly via url
5. System provides secure authentication

10.3 User interface requirements

1. Interface provides capability to log into the system only users with permissions to make backup copy
2. Interface informs user about unsuccessful login attempt
3. Interface lists databases available to dump
4. Interface provides capability to add database to list by providing defined set of parameters
5. Interface enables user to start backup process
6. User interface shows time of backup for every dump file
7. Interface lists available dump files
8. User interface allows deleting dump files
9. Interface enables user to start restore process
10. Administrator is allowed to specify system well known settings
11. Interface provides capability to log out from the system when user is logged in

12. System lists available types of databases to choose from in combobox
13. System allows editing information about available databases
14. Interface allows removing databases from the list
15. Interface lists available types of output files to choose from in combobox
16. Interface lists available modes of compression to choose from in combobox
17. Interface provides capability to upload local dump of database to restore from
18. Interface allows user changing password in a secure way
19. User interface shows warning information in case of making a backup copy and in case of restoring a dump
20. Interface shows a logo on top of the page
21. Interface shows copyright information on the bottom of the page

10.4 Implementation requirements

1. Project uses CVS system for distributed development.
2. Project is maintained using Maven and its structure is compatible with the tool.
3. Eclipse is used by project developers.
4. FindBugs eclipse plugin (<http://findbugs.sourceforge.net/>) is used to detect possible developers' mistakes like null problems, open streams, etc.
5. Checkstyle software (<http://checkstyle.sourceforge.net/>) is used to keep one style of coding amongst all the developers.

11 Testing

11.1 Approach

Testing is the next very important part of software development. That is why, we decided to invest time and:

- brainstorm what to test (this has been done on a piece of paper)

- divide tests into categories
- write a priority list of tests in each category and describe them for future implementation
- implement the most important ones

There are bottom-up and top-down approaches. Bottom-up is when the application is tested from the very tiny pieces of software, so the tests are written as simple bricks, and if they are passed, it implicates that the whole application works. Top-down is different - here we perform tests of the final product and if they are not passed, we write more detailed tests.

11.2 Test categories

11.2.1 JUnit testing

To make our product more reliable, and to decrease the debugging cost during development we decided to make use of JUnit framework. JUnit allows developer to describe simple tests that have to be passed by stable version of software.

These tests are similar to the ones that every developer makes in his code during debugging, but what makes JUnit a very useful and time-saving tool is that once tests are described they can be repeated automatically after every change to the code without any effort from the developer. This is the most important thing about JUnit tests, and because it helps to discover bugs almost at the time of their creation - this is the time-saving power of JUnit.

11.2.2 Dbunit testing

DbUnit is a useful and powerful tool for simplifying unit testing of database operations. It extends JUnit framework, that was described in the previous point. With DbUnit, a database can be seeded with a desired data set before a test; moreover, at the completion of the test, the database can be placed back into its pre-test state. DbUnit has a few extremely useful features for unit testing of database operations, for instance:

- a very simple XML-based mechanism for loading test data
- a framework which simplifies operations for each stage in the the life cycle of individual database tests
- an equally simple mechanism for exporting existing test data into the XML format for subsequent use in automated tests
- methods for comparing data, between flat files, queries and database tables

11.2.3 HTTPUnit and/or HTMLUnit testing

Because our application is web-based, the natural way to test it's behaviour is to emulate browser. There are two interesting test suits, that we decide to use - HttpUnit and HtmlUnit.

HttpUnit emulates the relevant portions of browser behavior, including form submission, JavaScript, basic http authentication, cookies and automatic page redirection, and allows Java test code to examine returned pages either as text, an XML DOM, or containers of forms, tables, and links. When combined with a framework such as JUnit, it is fairly easy to write tests that very quickly verify the functioning of a web site.

HtmlUnit is a java unit testing framework for testing web based applications. It is similar in concept to httpunit but is very different in implementation. HttpUnit models the http protocol so you deal with request and response objects. HtmlUnit on the other hand, models the returned document so that you deal with pages and forms and tables.

We will implement some tests using HtmlUnit and some using HttpUnit because different approach in implementation of the two may make a difference in test results.

11.2.4 Manual testing

Generally manual testing is no testing, however there are some cases when it is just easier to check once before project release if anything unexpected happens.

11.3 Test scenarios

11.3.1 JUnit tests scenarios

The list of classes that may be tested with JUnit tests:

1. Configuration
2. DatabaseConnector
3. DatabaseConnectorFactory
4. MysqlConnector
5. PostgresqlConnector
6. Dump
7. DumpFileManager
8. Restore

11.3.2 Dbunit testing scenarios

The purpose of this test is to check the state of the database before and after the backup. The test is performed using external tool (Dbunit).

Input data: database image before and after backup

Output: true if images are identical

1. Export a database to a xml file (Dbunit)
2. Dump the same database to our format type file
3. Restore database by means of the file created by dump method
4. Export the restored database to a new xml file (Dbunit)
5. Compare those two xml files (Dbunit) and make sure they are identical

11.3.3 HTTPUnit and/or HTMLUnit testing scenarios

1. Authentication test [AuthenticationTest]

The purpose of this test it to test logging facilities and find possible abuses of this part of the system.

Input data: „login” page, „settings” page

Output: fail if expected behaviour differs

- (a) Login to the system with username and password
- (b) Go to „Settings” page
- (c) Fill in the change password form with random data (and remember it)
- (d) Update settings
- (e) Logout
- (f) Try to log in with the old password
- (g) Log in to the system with remembered username and password
- (h) Go to „Settings” page
- (i) Fill in the change password form with original data
- (j) Update settings
- (k) Logout
- (l) Try to log with the previus password

(m) Log in with current password

2. Database list test [DatabaseListTest]

The purpose of this test it to test whether the same databases are shown in „database list” page and stored in the configuration file and find possible abuses of this part of the system.

Input data: „database list” page, configuration file

Output: fail if expected behaviour differs

- (a) Read „Database list” page on „Database list” page
- (b) Read configuration file on the server
- (c) Check whether they match

3. Add database test [DatabaseAddTest]

The purpose of this test it to test it adding databases to the system works properly and find possible abuses of this part of the system.

Input data: „database list” page, „add database” page

Output: fail if expected behaviour differs

- (a) Read and remember entries on „Database list” page
- (b) Fill in the form on „Add database” page
- (c) Submit the form
- (d) Read „Database list” page again
- (e) Modify data
- (f) Submit the form
- (g) Read „Database list” page again
- (h) Generate random database entries
- (i) Add generated database
- (j) Remove generated database
- (k) Read „Database list” page again
- (l) Expect newly generated page to contain just added entry

4. Download file from the server test [TrivialDownloadTest]

The purpose of this test it to test if backed up database is ready to download and find possible abuses of this part of the system.

Input data: „backup list” page

Output: fail if expected behaviour differs

- (a) Select iteratively database from the database list on the „Backup list”
- (b) Start downloading currently selected database
- (c) If started receiving data, the this part of the test is passed and download stopped
- (d) Logout from the system
- (e) Try to abuse the system and download images using URL
- (f) Try to download configuration file
- (g) If download test is passed and after logout there is impossible to download data, test is passed

5. List plugins test [ListPluginsTest]

The purpose of this test it to test if listed plugins match entries in the configuration file and find possible abuses of this part of the system.

Input data: „backup” page, configuration file

Output: fail if expected behaviour differs

- (a) Read database engine list on „Backup” page
- (b) Read plugin configuration file on the server
- (c) Check whether they match

6. Backup list test [BackupListTest]

The purpose of this test it to test whether backups shown on the page match directory contents and find possible abuses of this part of the system.

Input data: „backup list” page, contents of the backup directory

Output: fail if expected behaviour differs

- (a) Read backup list on the „Backup list” page
- (b) Read backups in the server’s backup directory
- (c) Check whether they match

7. List backup database test [ListBackupDatabaseTest]

The purpose of this test it to test if system properly lists databases stored in the system and find possible abuses of this part of the system.

Input data: „database list” page, „backup” page

Output: fail if expected behaviour differs

- (a) Read database list on the „Database list” page
- (b) Read database selectlist on the „Backup” page

(c) Check whether they match

8. Output file format test [OutputFileFormatTest]

The purpose of this test it to test if created output file format is really like one selected and find possible abuses of this part of the system.

Input data: „backup” page, configuration file

Output: fail if expected behaviour differs

- (a) Read output file format selectlist on the „Backup” page
- (b) Read configuration file for output file formats
- (c) Check whether they match

9. Compression test [CompressionTest]

The purpose of this test it to test if compression of the database works and find possible abuses of this part of the system.

Input data: „backup” page, configuration file

Output: fail if expected behaviour differs

- (a) Select random compression mode from the selectlist on the „Backup” page and remember it
- (b) Backup database
- (c) Use file properties to determine whether the output really is the selected format

10. Delete local backup test [DeleteLocalBackupTest]

The purpose of this test it to test if deletion of the local backup works and find possible abuses of this part of the system.

Input data: „backup list”

Output: fail if expected behaviour differs

- (a) Select random database file from the list on the „Backup list” page and remember it
- (b) Delete it
- (c) Re-read „Backup list” page and check whether the deleted database disappeared

11. Restore options test [RestoreOptionsTest]

The purpose of this test it to test restore facilities and find possible abuses of this part of the system.

Input data: „restore” page

Output: fail if expected behaviour differs

- (a) Read database list and file format from backup selectlists
- (b) Do the same with restore selectlists
- (c) Check whether they match

12. Settings test [SettingsTest]

The purpose of this test is to test settings and find possible abuses of this part of the system.

Input data: „settings” page

Output: fail if expected behaviour differs

- (a) Read backup directory path from the „Settings” page
- (b) Read the same but from configuration file on the server
- (c) Check whether they match

11.3.4 Manual testing scenarios

1. Backup notification test

The purpose of this test is to check whether notification mechanisms work properly.

Input data: „backup database” page

Output: fail if expected behaviour differs

- (a) Type in correctly an email address
- (b) Check „notify me when backup is ready” box
- (c) Click „backup database”
- (d) Check for an email
- (e) Check whether backup is really ready to download

2. Restore notification test

The purpose of this test is to check whether notification mechanisms work properly.

Input: „restore database” page

Output: fail the test if expected behaviour differs

- (a) Type in correctly an email address
- (b) Check „notify me when the database is fully restored” box
- (c) Click „restore database”
- (d) Check for an email

(e) Check whether database is really restored

3. Naughty test

The purpose of this test is to check how the system reacts in some unlikely and pesimistic conditions:

- (a) User deletes a database, then presses reload in the browser, which in effect sends the same delete request once again.
- (b) User orders the system to dump or restore a database, then returns to the menu and restores the same database again. As a result two processes would be inserting the same data at the same time or one would be inserting the new data whereas the other would be reading the mixture of new and old data.
- (c) User has got a database, let's say MS SQL. He inserts the MssqlConnector plugin to the right directory, adds the database to the configuration file and performs the dump operation. One day later a nasty admin removes this plugin. The user comes back, chooses the dump and wants to perform "restore" operation. Of course this operation cannot succeed, because there is no proper connector. The same situation takes place when user wants to create another dump from this database.
- (d) User provides a wrong path to save dump files, for example „C:” would be wrong on a Unix server; or a user provides the name of the directory he cannot access, for example „/root”. Then he tries to dump a database.
- (e) User sends a dump file, which is really not a dump file but for example his favourite music video (he may have click a wrong file in the open file dialog).
- (f) User downloads a dump file from database #1, which is a PostgreSQL database, then tries to put it into database #2 which runs MySQL 4. No matter how hard he tries, he cannot succeed (because of problems with e.g. procedures, triggers, views...)
- (g) User downloads a dump file in a binary/ZIP format, and then tries to insert it to the database as a plain text or GZIP file.

11.4 Requirements and scenarios table

Functional requirement	Coverint scenario (default from HTTP/HTML unit tests)	Testing technique
1	all HTTP and HTML unit tests (1-12)	HTTP Unit / HTML Unit
2	1	HTTP Unit / HTML Unit
3	2	HTTP Unit / HTML Unit
4	2	HTTP Unit / HTML Unit
5	3	HTTP Unit / HTML Unit
6	5	HTTP Unit / HTML Unit
7	3	HTTP Unit / HTML Unit
8	10	HTTP Unit / HTML Unit
9	4, 6	HTTP Unit / HTML Unit
10	8, 9	HTTP Unit / HTML Unit
11	4, Dbunit tests	HTTP Unit / HTML Unit, Dbunit
12	1	Manual
13	6, 10	HTTP Unit / HTML Unit
14	11	HTTP Unit / HTML Unit
15	Dbunit tests	Dbunit
16	2	Manual
17	12	HTTP Unit / HTML Unit
18	12, 1	HTTP Unit / HTML Unit
19	12	HTTP Unit / HTML Unit
20	5	HTTP Unit / HTML Unit

Security requirement	Coverint scenario	Testing technique
1	1	HTTP Unit / HTML Unit
2	1	HTTP Unit / HTML Unit
3	4	HTTP Unit / HTML Unit
4	4	HTTP Unit / HTML Unit
5	1	HTTP Unit / HTML Unit

User interface requirement	Coverint scenario	Testing technique
1	1, 6	HTTP Unit / HTML Unit
2	1	HTTP Unit / HTML Unit
3	2	HTTP Unit / HTML Unit
4	1, 2	HTTP Unit / HTML Unit
5	3	HTTP Unit / HTML Unit
6	5	HTTP Unit / HTML Unit
7	6	HTTP Unit / HTML Unit
8	10	HTTP Unit / HTML Unit
9	11	HTTP Unit / HTML Unit
10	12	HTTP Unit / HTML Unit
11	1	HTTP Unit / HTML Unit
12	5	Manual
13	2, 3	HTTP Unit / HTML Unit
14	2, 3	HTTP Unit / HTML Unit
15	8	HTTP Unit / HTML Unit
16	9	HTTP Unit / HTML Unit
17	11	HTTP Unit / HTML Unit
18	1	HTTP Unit / HTML Unit
19	-	-
20	-	-
21	-	-

11.5 Implementation

We decided to implement only the most important tests at this stage of the development. The chosen tests are as follows:

11.5.1 JUnit tests

1. ConfigurationTest.testAddDatabase() - a JUnit test that tests the use of Configuration.addDatabase() method. It takes the list of defined connections with Configuration.getConnections(), adds an entry with Configuration.addDatabase(), gets the list once more, and compare it with the previous one.
2. ConfigurationTest.testRemoveDatabase() - a JUnit test for Configuration.removeDatabase() method. It works very similar to the Configuration.testAddDatabase() test.
3. ConfigurationTest.testGetInstance() - a JUnit test for checking if the Singleton design pattern is implemented as it should be.

4. `ConfigurationTest.testIO()` - a JUnit test for checking if save/load operations (performed by `Configuration.saveData()`, and `Configuration.restoreData()` respectively) on configuration file works good. The test consists of sequences of save, modify, load, and compare operations on the configuration data.
5. `AuthenticationTest.testWrongPassword()` - a HTTPUnit test that tries to log into the system using wrong password
6. `AuthenticationTest.testEmptyPassword()` - a HTTPUnit test that tries to log into the system without specifying any password
7. `AuthenticationTest.testMessingWithPasswords()` - a HTTPUnit test that tries to log into the system, change current password using Settings interface, tries to log in with the old password, logs in with the new password, change password back to the original one, and tries logging in with old and current password once more.
8. `DatabaseConnectorFactoryTest.testGetInstance()` - JUnit tests whether the Singleton design pattern works properly, that is one and only one instance of `DatabaseConnectorFactory` class exists
9. `DatabaseConnectorFactoryTest.testCreateConnector()`- JUnit tests whether the Factory design pattern works properly; that means `createConnector` method produces the proper `DatabaseConnector` on the basis of user's needs (e.g. `MysqlConnector`, `Postgresql connector`)
10. `DatabaseConnectorFactory.listPlugins()` - JUnit test whether the `listPlugins` method returns classes that extend `DataBaseConnector`
11. `MysqlconnectorTest.testConnect()` - JUnit tests whether the `Connection` object has been created and checks if it is not null type
12. `MysqlconnectorTest.testDisconnect()` - JUnit tests whether the connection has been closed
13. `MysqlconnectorTest.testCreateURL()` - JUnit tests whether the connection URL has been properly created

11.5.2 Dbunit tests

Dbunit test has been implemented in the method `MysqlconnectorTest.testDumpRestore()`, which fulfills the scenario.

11.5.3 HTTPUnit and/or HTMLUnit tests

1. Add database test has been implemented using HTTP unit, which probably is better to this purposes.

- AddDatabaseTest.readList() - reads database list and remembers it
 - AddDatabaseTest.addDatabase() - randomly generates database details and adds it to the system
 - AddDatabaseTest.check() - check whether database was really added
2. Trivial download test has been implemented using HTTP unit, which probably is better to this purposes.
- TrivialDownloadTest.readNext() - reads next database from database backup list
 - TrivialDownloadTest.download() - starts downloading currently selected database
3. Authentication test has been implemented using HTTP unit, because it uses mainly http requests to deal with tested forms and actions. The invoked methods are:
- AuthenticationTest.readConfig() - reads generic username and password from the config file, remembers it and sets active
 - AuthenticationTest.login() - logs in with active username and password
 - AuthenticationTest.changePassword() - changes password of the user to randomly generated one
 - AuthenticationTest.logout() - logs out from the system

12 Implementation

12.1 Requirements

To launch the application (besides a piece of hardware), the following external applications are needed:

- Java version 1.5 (JDK)
- WWW server that serves jsp pages - we are using Apache Tomcat 5 and we recommend it to use with our application, but other tools are welcome (please let us know if you experience any trouble or success running jdbdump on other platforms)

12.2 Safe launching and possible security risks and data loss

There are few security risks when using jdbdump. The following list describes them and suggests a solution:

- Not encrypted username and password when logging into the system.
We strongly suggest using an SSL enabled web server in order not to be sniffed by third party's spy software.
- Other users of the system that jdbdump is being launched on may read jdbdump files.
In this situation we strongly advise to check whether the files created by jdbdump (e.g. backups) are readable only by „tomcat” user (in case you are using Apache Tomcat server).
- Not encrypted database image files transferred from the server to the user's computer.
This might be harder to sniff, because transferred images are stored in a serialized and zipped / gzipped format, but there is still a possibility that someone nasty will log the whole image and then try to find some passwords. That is why we also recommend to download the image file via an SSL channel, which is however pretty slow. In the future versions secure ftp transfers should be implemented. The other solution is to encrypt the image file using e.g. a password algorithm and then use that password to retrieve image.
- Database images are stored temporarily in the server's working directory.
In case the server is tomcat, on every reload or deployment of the project probably all backed up data is lost. It means that users need to download the image file before someone does one of the mentioned things. This is normally not a problem, however we suggest to download the image file just after it is being done.
The best solution would be to create another implementation of DumpFileManager, which would upload files to a remote database as blob objects instead of saving them in the filesystem. Then they would be protected from malicious sysadmins.

12.3 Performance testing

There are two approaches to performance testing, either to use dedicated software or to write own tests.

We have decided to measure the time the console program for dumping mysql databases (mysqldump) consumes and compare it to the dump of the same database set, but created by jdbdump.

Here are the results of mysqldump and our Dump comparison:

SIZE	MYSQLDUMP TIME	JDBDUMP DUMP TIME
4.3MB	1,53 s	6.20 s
7.2 MB	2.5 s	12.6 s
10.3 MB	4.05 s	21.9
15.2 MB	6.7 s	51.75 s
21.7 MB	8.5 s	90.3 s

As we had expected mysqldump turned out to be faster. That is caused by many factors, for example that it is a natively compiled program and Jdbdump runs on a virtual machine, it handles MySQL databases specifically and Jdbdump tries to be universal, and Jdbdump uses JDBC calls which add another layer of complexity.

Here is a diagram that presents these dependencies:

12.4 Problems encountered during performance test

Performance tests are also a good way to discover some problems or errors that may result from technologies' missing features. We have created a very simple mysql database and inserted 100000000 (100 million) rows into one of its tables. However the dumping process did not succeed. Instead of the backup file we have received the following error: "java.lang.OutOfMemoryError". That shouldn't have happened, because Jdbdump copies tables directly to a file record after record, not loading entire tables into memory.

It turned out that the guilty one was the Mysql server and JDBC driver. In our java application we have used setFetchSize() method to make the program download data piece by piece, that means if you write setFetchSize(100) and want to download data from a table consisting of 1000000 rows, it should download only the first 100 rows at the beginning, and then the next 100 rows when they are accessed, and so on. However, to our surprise it turned out that this feature is available only in MySQL 5.0. So we had to implement it ourselves in our code.

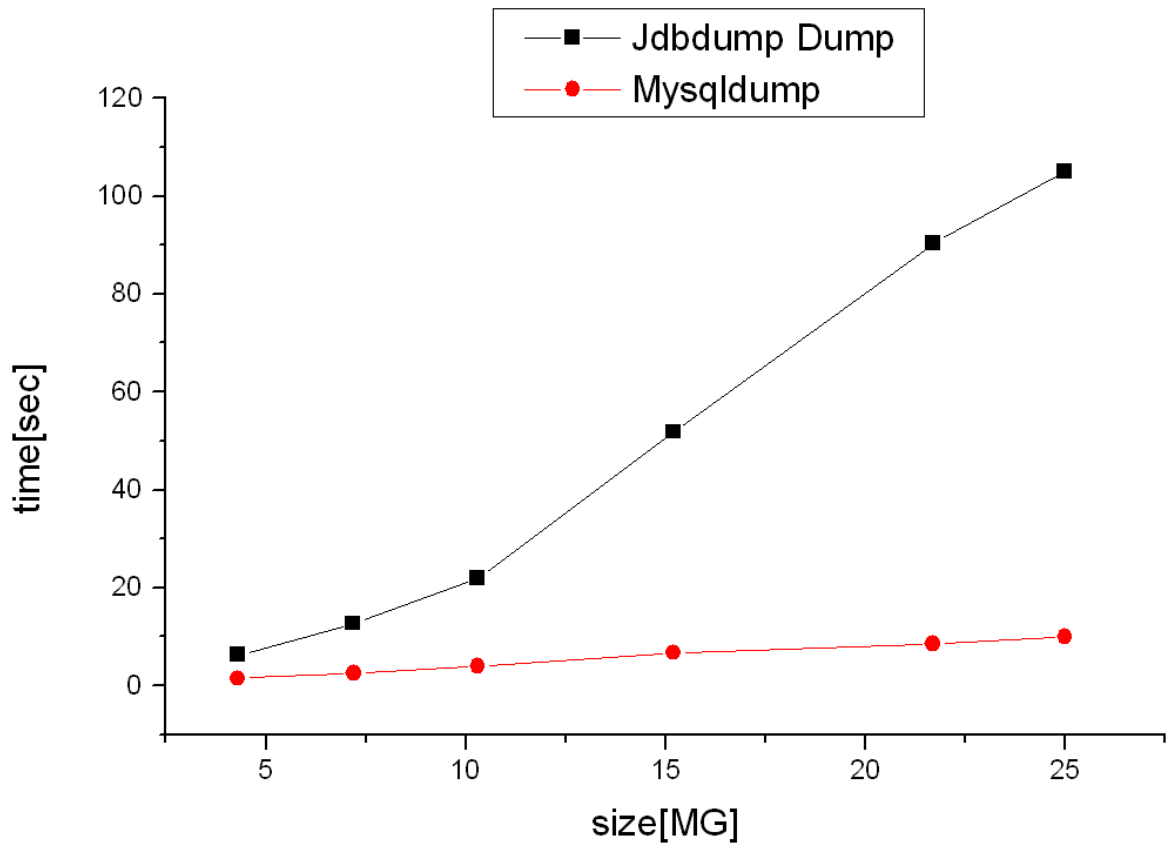


Figure 6: The comparison between mysqldump and jdbdump

12.5 Code and application environment optimization

Sometimes it is more convenient or more efficient to use a Prepared Statement for sending SQL statements to the database. In jdbdump we use Prepared Statements because we deal with many tables and data. The main feature of a Prepared Statement is that it is given an SQL statement when it is created. The advantage to this is that in most cases, this SQL statement will be sent to the DBMS right away, where it will be compiled. As a result, the Prepared Statement object contains not just an SQL statement, but an SQL statement that has been precompiled. This means that when the prepared statement is executed, the DBMS can just run the prepared statement's SQL statement without having to compile it first. This speeds up the dump / restore process.

Currently we use Prepared Statement in the process of uploading records of data from one table during the execution of DatabaseConnector.restore(). An insert statement is prepared each time a new table has to be filled with data, and then the same statement is executed once for each record of data.

12.6 Safe configuration and its impact on performance testing

Generally speaking, the more secure the application is, the slower it is. This (considering jdbdump) might be caused by:

- Slower data transfer because of the use of SSL protocol (this might be actually quite fast if hardware SSL accelerators are used, but who besides very big companies can afford those?)
- If all the security issues from the „wishlist” were implemented, there might be a noticeable slowdown in performance tests due to encrypting of image files
- If we use a remote database or FTP server instead of the local filesystem to store backups and/or configuration, it's obvious that it will take more time to upload and download them than it would take to just save them and load them back.

12.7 Installation and configuration

This web application is provided as a WAR file so it is simple to deploy to a server, but due to J2EE authentication used in project some preparations must be done to properly start the application. Also some properties must be set to suit user requirements.

- For MySQL authentication:
 1. Create a MySQL database named 'authority'. You can use a provided script called 'realm.sql'. You should notice that allowed users are those from group 'systemadmin'
 2. Specify allowed users adding them into the database table
 3. If you use mysql based authentication, then next thing is to copy mysql-connector-java.jar into server/lib directory placed in tomcat directory (TOMCAT_HOME)
 4. Enable authentication based on created database
- Standard authentication:
 1. Add role „systemadmin” into the tomcat-users.xml file
 2. Add a username and password (with „systemadmin” role) to the tomcat-users.xml file
 3. Edit properties files to suit your needs (remember to create the backup directory and set privileges that Tomcat server can use this directory to store backups (default is /var/backups))
 4. You may sometimes need to copy app.properties file into another location (it depends on your's Tomcat configuration)
 5. After that you can deploy the application to the tomcat server using for example manager panel

12.8 User's guide

In order to use Jdbdump application you need to decide whether you prefer web or console application. Both of these have some advantages: web interface is user friendly, self descriptive and easy to use. However some people are used to traditional console commands and therefore we have decided to satisfy their needs and provide a simple console tool.

If you have not used neither of these Jdbdump interfaces before, then look through the following guides to obtain the necessary knowledge.

12.8.1 Web interface

Before making use of any of Jdbdump functionality you have to log to the system. If logging is successful you will find a menu consisting of the following operations:

- Database list
- Add database
- Backup
- Backup list
- Restore
- Settings
- Logout

If your database list is empty you have to add a new database that needs to be backedup. Just click "Add database" and provide all the necessary information (Database title, Server name or IP, server port - optional, database engine, database name, login and password), for example:

When databases are added to the configuration file, you should be able to list them by clicking "Database list". This will display database names and next to each name you will notice "Edit" and "Delete" options. "Edit" enables you to change the information you have provided in the Add database panel, whereas "Delete" removes all the configuration data concerning the given database, for example:

Now let's get to the most important functionality of our JDBC database archiver. Suppose you want to backup a database "IOSR Database", which you have already added to the list of available databases. First of all, choose "backup" option in the menu and then find "IOSR Database" on the list. Choose

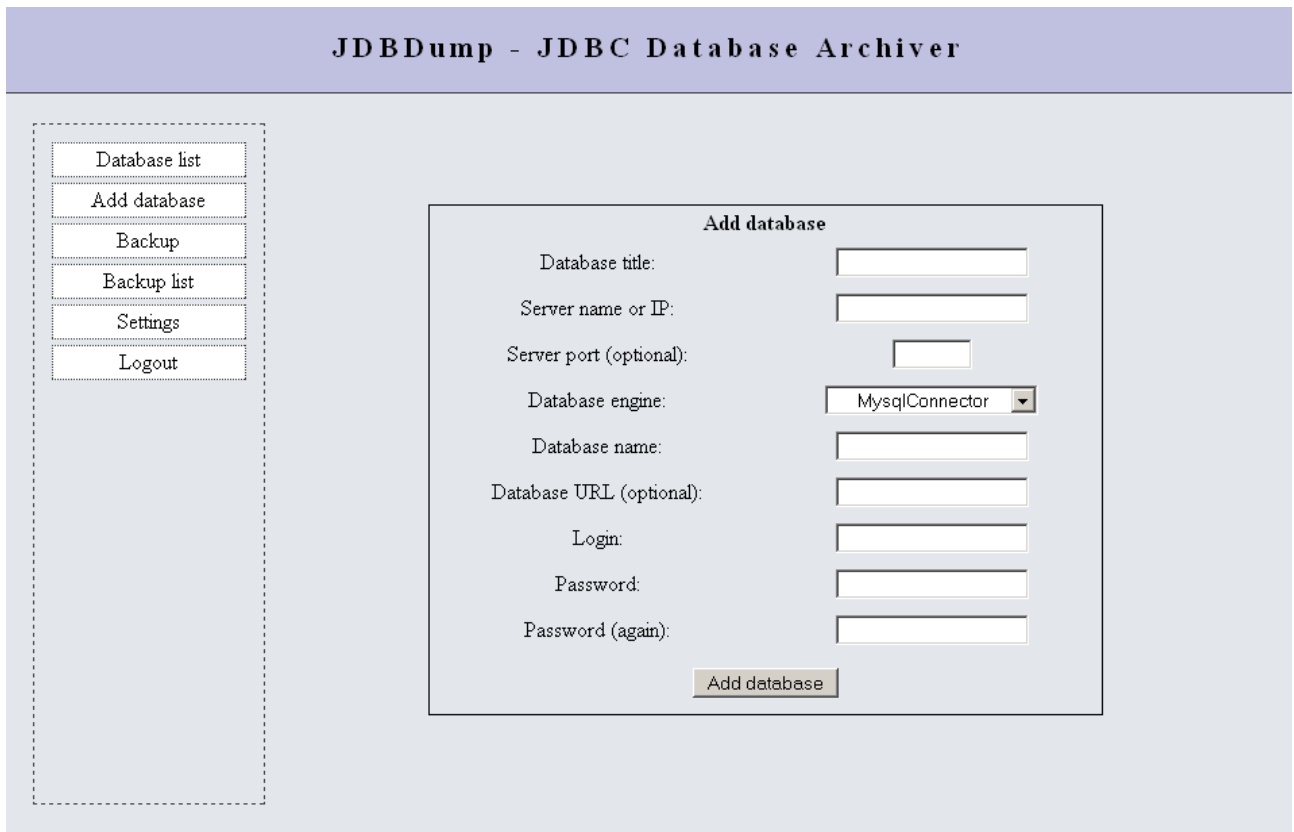


Figure 7: Add database

the compression mode. You can prefer to get Zip or Gzip files. Since the backup process may take some time, you can also select that you want to be informed about the end of the backup. Then accept your choices by clicking "Backup database", for example:

When the backup file is ready you should download it to your local machine - "Backup List" panel. If you have the backup files on your machine you can restore your database with data present in one of those files. This operation is very similar to "Backup", however this time you have to choose the file, by means of which you want to reconstruct your database and in the same way as before choose database name, file format and compression mode. The restoring process may be time consuming as well so it is recommended to click notifying option.

It is also very important to get familiar with the Settings panel, where you can keep your email address, and point the directory where you want all your backup files stored, for example:

JDBDump - JDBC Database Archiver

Database list











Add database

Backup

Backup list

Settings

Logout

Database list			
Database title	Server address		
nie wchodzic tu	localhost	 Edit	 Delete
Moja baza	localhost	 Edit	 Delete
Jsuder2 mysql.agh	mysql.agh.edu.pl	 Edit	 Delete
nie wchodzic kopia	localhost	 Edit	 Delete
Jsuder.agh	mysql.agh.edu.pl	 Edit	 Delete

(c) copyright by [JDBDump team](#), 2005-2006






Figure 8: List databases

12.8.2 Console tool

The console tools works in a slightly different way and involves some more configuration work. Here are the basic steps that will allow you to work with your database.

Firstly, create a file with configuration data, let's call it "mydatabase.conf". This is a java property file and it should look like this:

```
connector=net.sourceforge.jdbdump.connect.connectors.MySqlConnection
url=jdbc:mysql://localhost/cpe
login=mylogin
password=...
compression=gzip
```

If you have it you are almost ready. Open you terminal and write down the following command:

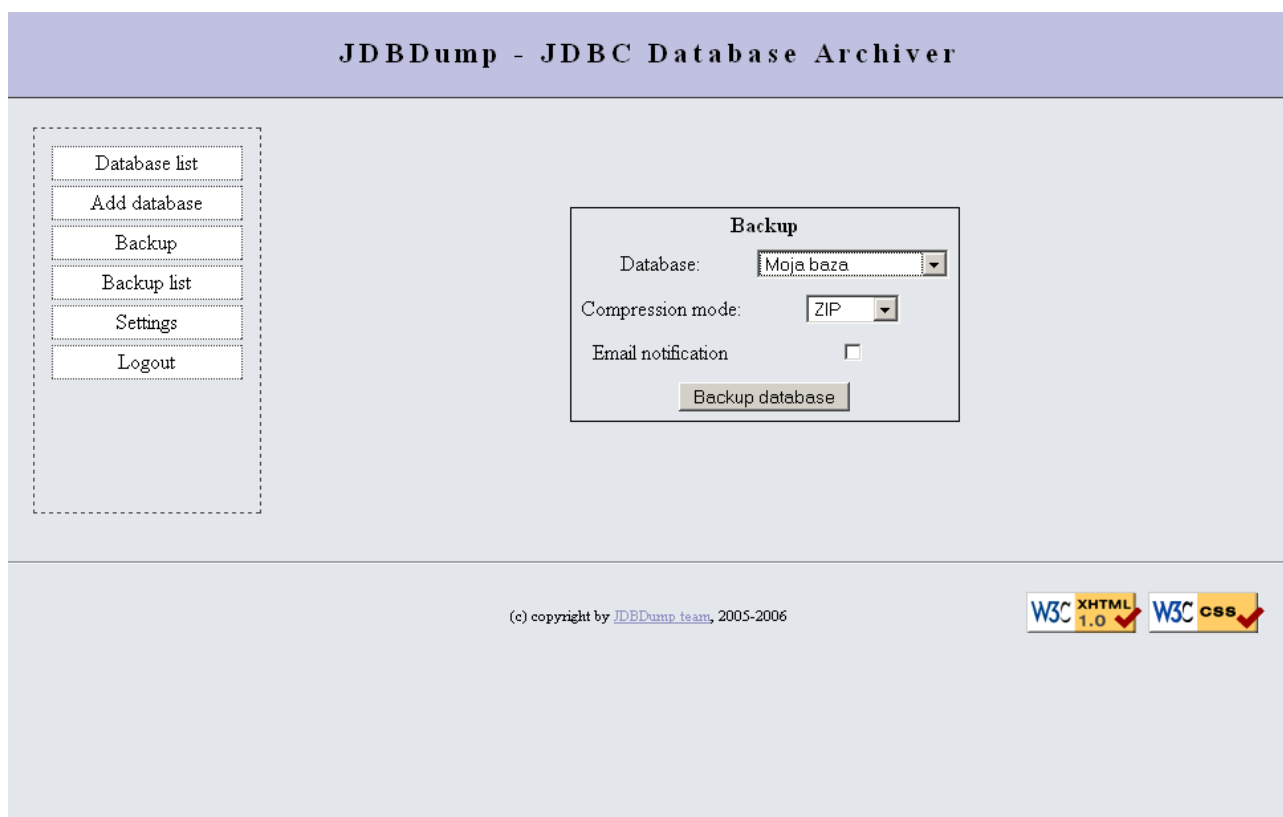


Figure 9: Backup your database

```
~$ java -classpath <path_to_jdbdump> \
net.sourceforge.jdbdump.connect.TestDump dump \
testdump.gz mydatabase.conf
```

The program takes three parameters:

- the action -"dump", "restore" or "print"
- the name of the backup file
- the name of the configuration file

12.9 Changes in realisation

During the application development we decided to change following things:

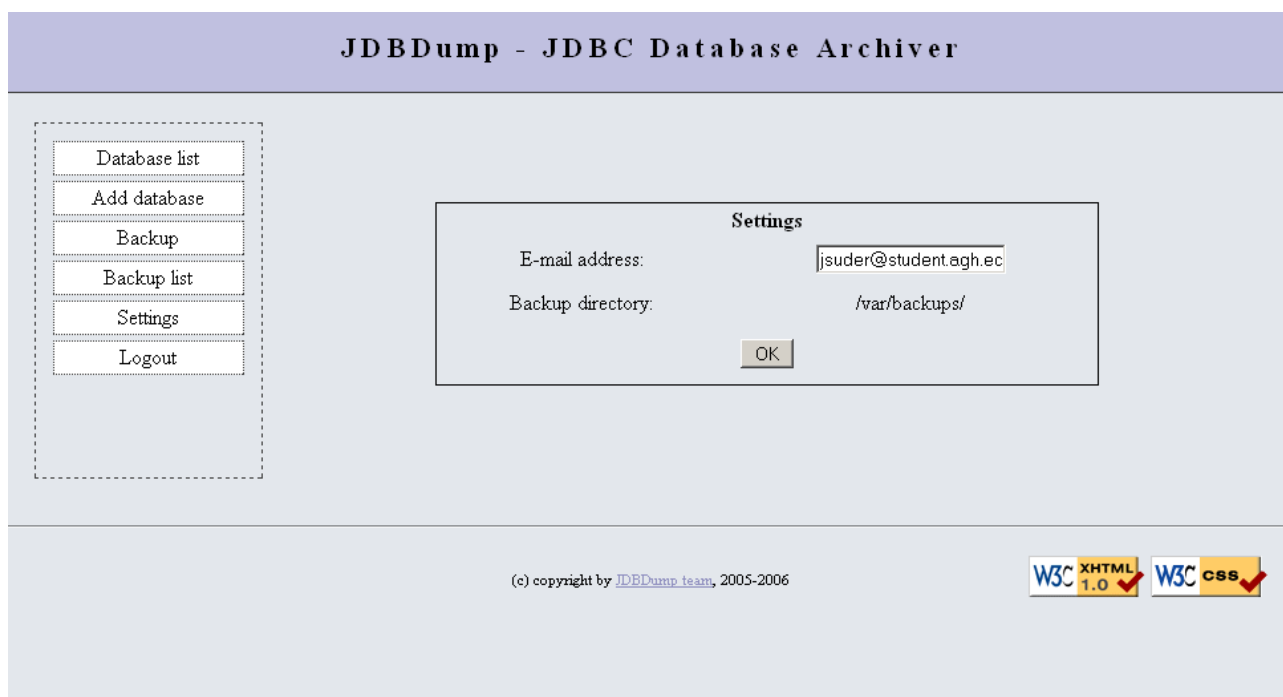


Figure 10: Settings panel

- GeneralConnection has been renamed to DatabaseConnector, *Connection classes to *Connector, and we decided to make DatabaseConnector an abstract class, because there is no such thing like universal connector which backs up any type of database and some of the methods simply had to be done in derived connectors
- When we started implementing restore, it turned out that the database has to be cleared of old tables before the new ones are inserted. That involved not only "DROP TABLE" queries, but also some "ALTER TABLE", because we learned that before the tables are removed, some constraints have to be removed from them. That is because we can't delete a table as long as there is any other table that uses its fields as foreign keys. So first the constraints have to be removed from tables, then the tables themselves, and only then new versions of tables can be added to the database.
- To create the web interface, we decided to use Java Server Faces (MyFaces) instead of Struts.
- We decided to extend the configuration subsystem, using one abstract base class and a few implementing classes, handling different ways of storing the configuration (for example, in local filesystem and on a FTP server). The reason for this is that storing the configuration in local files may not be safe, because they may get lost when the application is redeployed.
- We added a Mailer class using Sun's JavaMail framework to send notification emails.

- During the development we created a TestDump class which at first was meant to test the behavior of „engine” classes while the web GUI was not finished, and later it evolved into a simple but powerful command line client for Jdbdump.
- The methods initializing and downloading the stream of data records from a table in a database have been moved to specific connectors, because it turned out that even these simple commands have to be handled specifically for them (e.g. MySQL requires that table names are given in apostrophes (“) if they contain some special characters, and some databases don’t support setFetchSize() so their connectors have to do it for them).
- We added the Log4J library and replaces System.out.println() calls with Log4J methods.
- At first we couldn’t download a test MySQL database, because it threw SQLExceptions concerning a date in a wrong format. We found later on the net that it was caused by MySQL JDBC driver’s wrong interpretation of date; MySQL server allows storing an empty date (0000-00-00), but the JDBC driver throws an SQLException when it sees one. We learned that a special option „?zeroDateTimeBehavior=convertToNull” had to be added to the connection URL to fix that.

At the moment we did not finish implementing some parts of GUI. These are uoload and download sections and also the behaviour of a browser while the dump is being done. However, this is not a problem, you just only need external file download/upload software like sftp.

12.10 Further development

There are still a lot ideas that may be implemented in the future versions of jdbdump. We are aware of the fact, that some parts of the application are not 100% safe and handy. That is why we have prepared a „wishlist” for further development. Here it goes:

- encrypted dump file
- possibility to automatically send dump file via SSH or SFTP to any defined location
- support other database elements such as stored procedures or triggers
- support less commonly used table data types and options which may appear in table definition
- add a few more connectors for other database engines like Oracle, MS SQL server
- use JNDI data sources to connect to databases
- add more file formats like plain text (currently, only binary/serialized file format is available)
- provide the possibility to convert one type of database into another

- protect against the special situations described earlier in the paragraph named „Naughty tests” (see 11.3.4.3)
- test Jdbdump in environments other than Tomcat
- improve performance

13 Changelog

- 05.11.2005 - Expansion of problem's description.
- 06.11.2005 - Writing down and deciding upon some implementational issues
- 07.11.2005 - User's interface description
- 08.11.2005 - Setting up jdbdump project on sourceforge: jdbdump
- 08.11.2005 - Generating documentation in Maven: Maven
- 14.11.2005 - Creation of class diagram
- 14.11.2005 - Creation of mailing list on sourceforge
- 15.11.2005 - Creation of use-case diagram
- 15.11.2005 - Update on documentation due to the recently created class diagram
- 18.11.2005 - Creation of documentation in pdf version
- 19.11.2005 - Creation of WWW interface model
- 20.11.2005 - Topics added to documentation: description of design patterns and technologies used in the project
- 02.12.2005 - Tests descriptions and tests added
- 12.12.2005 - Additional tests added
- 04.01.2006 - Dump implementation added
- 05.01.2006 - Restore implementation added
- 06.01.2006 - Mysql support added
- 07.01.2006 - Team meeting: 3 hours of heavy disputes about current development. Decision: dumps are stored in the local filesystem, but users are suggested to download the file images
- 07.01.2006 - Some notes on implementation added
- 08.01.2006 - A few bugs discovered
- 08.01.2006 - Security and environment notes added
- 09.01.2006 - „Wishlist” added

- 09.01.2006 - Team meeting: some bugs fixed, performance tests planned
- 10.01.2006 - Memory leak bug discovered in Dump, fixed
- 10.01.2006 - Performance tests performed
- 10.01.2006 - FAQ added
- 11.01.2006 - Team meeting: discussion about integrating every jdbdump module
- 11.01.2006 - Mailer implemented
- 12.01.2006 - A few changes in the documentation, e.g. description of installation and configuration
- 13.01.2006 - Final maven integration, final site generations
- 13.01.2006 - Version 1.0 released :-)