**Vacations manager application.**

This application was developed for managing vacations in the small organization. Below you can find short description of its main functionality.

**Application layers description**

Application consists from 4 layers. Below you can find description of each of them.

1. Client layer.

It is written in *GWT*. For vacation’s chart drawing is used *gFlot* library. *Spring Security* is used for implementing securing mechanism (user authentication, concurrent sessions control etc.).

Client layer contains 3 JSP pages:

* login.jsp – this JSP page is used by Spring Security for user’s authentication (username and password check).
* VacationsManager.jsp – this is “main” JSP page to which GWT application is bound. There also some Spring Security tags is used.
* error403.jsp – this JSP page is shown when access for user is denied.

Entry class of our GWT application is *VacationsManager*. There initialization of the client side is started.

For easier development of the client side were developed a set of custom widgets. They are placed in package *com.crediteuropebank.vacationsmanager.client.widget*. You can look in code for more details.

The main unit of the application user interface is form. In application exists 3 forms that are used in the application’s single page: one form for central tab panel (*MainForm* class), another two for approval steps that is waiting for approval (*ApprovalListForm* class) and for remaining vacation days displaying (*RemainingVacationDaysForm* class).

Also for work with tabs was crated special *CustomTabPanel* which holds tabs that implements *CustomTab* interface. This was done for simplifying work with tabs (in particular, update of their content) and for customization tab header. As you can see in this application each tab’s header has update button with icon.

2. Client-Server communication layer.

For client-server communication is used *GWT RPC* that’s why this layer contains remote interfaces and their implementations necessary for RPC calls. This layer doesn’t contain any business logic. It just delegates calls to real service layer (“manager” classes). For linking together *GWT* and *Spring* is used *GWTHandler* library (Links related to this and other technologies you can find at the end of the document).

3. Business logic layer.

This layer can be separated into two sublayers: service layer and domain objects layer. Service layer contains logic of domain objects collaboration and some business logic. This layer consist from “manager” classes. Domain layer is represented by POJO objects that is saved into the DB and represent main application entities.

*Note 1. You should remember that service sublayer of the business logic layer is represented by* ***com.crediteuropebank.vacationsmanager.server.manager*** *package, and client-server communication layer is represented by* ***com.crediteuropebank.vacationsmanager.server.service****. So don’t be confused in future by names of the packages.*

*Note 2. For some domain (POJO) classes (like RemainingVacationDays, UsedVacationDays) there is no “manager” classes. This means that there is no much sense to consider such object separately from another object.*

In server side is used *Spring*. For logging is used *slf4j* framework with *log4j* as concrete implementation. Application flow logging logic mostly is implemented using *Spring AOP*.

4. DAO layer.

This layer contains all logic of work with database. For now all database access is implemented using *Spring JdbcTemplate* which simplified work with direct JDBC.

For controlling concurrent modification in DAO layer is implemented *optimistic locking*. During each CRUD operation application checks version of object (it is included in SQL statement) and if version is old – custom StaleObjectStateException is thrown. For example when we update user such query is executing:

*update users set username=?, fullname=?, role\_id=?, email=?, version=version+1 where id=? and version=? and status=1;*

If this query returns empty result then application compare entity version and version of the record in the DB. And in case when they are different – throws exception.

Optimistic locking prevent user from making an operation with old data and in such way it exclude a possibility when two users make operations one after another and second one rewrite changes made by first one.

Also should be mentioned that some entities *RemainingVacationDays* and *UsedVacationDays* have their separated DAO classes which are sometimes used. But most of all this entities are save/updated/deleted as a part of another entities – *User* and *Vacation* respectively.

**Code self-testing description**

By self testing I mean coverage of the application’s code by unit and integration test. For unit testing are used *JUnit* and *Mockito* frameworks. For integration testing is used *Spring* testing support. Separated source folders were created for each testing approach:

* *test* for unit tests classes;
* *integration\_test* for integration tests classes;

For now only few unit tests were written for application and most of all in educational purposes.

Integration tests cover main functionality of the application but they are written very quickly and that’s why can seem a little bit ugly and don’t follow best practices.

Also you should know that for integration tests is used separated “test” Spring configuration. It is represented by *TestApplicationConfig* configuration class placed in *com.crediteuropebank.vacationsmanager.springconfig* package. As you can see in this case configuration almost completely consists from java configuration. XML configuration file is only used for properties loading.

**Logging description**

In application implemented 2 logging approaches:

1. Application flow logging. In this approach application executes logging of its executing flow. For this purpose it uses *slf4j* framework with *log4j* as concrete implementation. It writes log messages to the console (with log level INFO) and two text files (they are placed in */WEB-INF/log/* folder):

* application.log – in this file all events with INFO log level are written. This log file should be checked first if some problem occurs. Then for more detailed information you should look to the *detailed.log* file.
* detailed.log – in this file all events with TRACE log level are written. This log is necessary for detailed error searching.

This kind of logging is implemented using *Spring AOP*. All logging is executed in *LoggingAspect* aspect class. It is placed in *com.crediteuropebank.vacationsmanager.logging* package and contains 2 around advices.

*Also you should know that now maximum log file size is 10 MB. If it is exceeded then new file will be created (its name will be the same + index number). Maximum number of such files will be 5. After this amount will be exceeded the eldest file will be rewritten. You can change this settings in log4j-myapp.properties file that is placed in the classpath.*

1. DB operations logging. In this approach application executes logging of each DB operation. For each concrete table there is a corresponding log table (for example *USERS* → *USERS\_LOG*) to which application writes information about passed DB operation.

If we want to log DB operations under entity we must annotate this entity using such annotations:

* *@Domain(logTableName, tableName)* – marks domain object class for entity that will be logged to the log table; specify log table and base table.
* *@Column(columnName)* – marks field that should be logged in the table; as argument takes name of the column in DB that corresponds to the field; the name of the column should be the same in basic table and log table.

To execute DB operation logging special *DBLogger* class is used in DAO layer. It has a set of methods to log a DB operation. For more details about using this class look at the code and comments in it.

**Validation logic description**

For validating entered values of the entities properties uses special custom class *ServerValidationUtil*. In this class input object is validated using *javax.validation.Validator* with *HibernateValidator* as concrete implementation. For correct work of the validation you should annotate fields of the entity that you want to check with JSR-303 annotations (*@NotNull*, *@Min*, *@Size* and etc.).

If some validation rules fail then *CustomValidationException* is thrown which holds the list of error messages. In the client side it will be shown to client.

Also for make some validation we need to hit the database. In such case this validation will be done in “manager” classes (for example look to *VacationManager* class).

**Password securing**

For purposes of security password is stored in DB in encoded way. For encoding password is used one way SHA1 encryption. Encoding is implemented using *Spring Security* support. For more details look at the *spring-security.xml* file and *JdbcUserDAO* class.

**Server exception handling on the client side**

For handling server exceptions returned as failure from RPC in one place was implemented special custom implementation of the *AsyncCallback* interface that is named *CustomAsyncCallback*. This custom implementation is used everywhere in the client side. In this implementation we separate custom exceptions from java exception and show corresponding message window depending on the exception type.

We use two custom exception classes to inform client side about some problem:

* *CustomMessageException* – this exception is thrown when we want to inform user about some failure or problem using single string;
* *CustomValidationException* – this exception subclasses previous one and is thrown when we want to inform user about the list of problems using the list of strings; such functionality is useful during validation.

**Transaction support**

All CRUD operations in application is executing inside transactions. When any error occurs (checked or unchecked) then transaction is rolled back so DB will never be in the inconsistent state.

Transaction together with optimistic locking gives a comprehensive control over concurrent user access to the same resources. If some problem occurs then all changes will be rolled back and user will be informed that he has stale data and he need to refresh browser.

**Caching support**

Caching in application is implemented using *Spring* cache support and *EHCache* as cache provider. Cache is used only in DAO layer for most of select operations (*@Cacheable* annotation). Each of the CRUD operations clears cache to exclude wrong return value (*@CacheEvict* annotation).

Settings for the *EHCache* cache provider are placed in the file *ehcache.xml* that is located in the classpath. The most important cache property among others set in this file is *copyOnRead="true"*. This means that each call to the cache will return copy of the cached object, not real object. This is very important in concurrent environment. Don’t change it to false! Better switch off cache at all.

If you want to turn off caching you can simply comment next line in *applicationContext.xml* file:

*<cache:annotation-driven/>*

**Application properties**

All of the application properties contains in special \*.properties files placed in the classpath. Below you can find brief description of them:

* db.properties – contains all settings necessary for connecting to the database;
* log4j-myapp.properties – contains settings for proper work of the log4j.
* mail.properties – contains all settings necessary for sending mails.
* vacations.properties – contains business logic properties (for now only max days ahead for taking vacation).

*Notify that I didn’t specify correct mail sending properties. For proper work of the mail sending you should create special user for application. Then you need to define his username and password in mail.properties file (another properties have already set). Application will use this credentials for connecting to our SMTP server and sending mails. Until you do this mails will not be sent and error message will be written to log. But this will not affect on the application flow.*

*Also it is desirable that you specify technical user’s email in mail.properties so application can send mails from his name. For now I put there my work email.*

**Program business logic description:**

Below you can find description of the business logic of the main operations under domain (POJO) entities. In this part will be described only CRUD operations. In select operations there is no much business logic and you can find it’s description in comments for methods. By business logic I mean the sequence of steps that is required to do to execute operation. Most of it contains in “manager” classes.

**Holiday days operations logic:**

*For HolidayDays entity object ServerValidationUtil and JSR-303 validation is not used.*

* Create holiday days operation:

1. Validate entered date range for holiday days record.
2. Call corresponding DAO layer’s method to save entity.

* Update holiday days operation:

1. Validate entered date range for holiday days record.
2. Call corresponding DAO layer’s method to update entity.

* Delete holiday days operation:

1. Simply call corresponding DAO layer’s method to delete entity.

**Role operations logic:**

* Create role operation:

1. Validate entered role parameters using *ServerValidationUtil*.
2. Call corresponding DAO layer’s method to save entity.

* Update role operation:

1. Validate entered role parameters using *ServerValidationUtil*.
2. Call corresponding DAO layer’s method to update entity.

* Delete role operation:

1. Check that specified role doesn’t have child roles. If it has then child roles should be deleted first.
2. Check that there is no users with specified role. If such users exist then they should be deleted first.
3. Check that there is no approval steps for which specified role is approval role. But at this stage there should not be such approval steps so this will be application error.
4. Call corresponding DAO layer’s method to delete entity.

**User operations logic:**

* Create user operation:

1. Validate entered user parameters using *ServerValidationUtil*.
2. Validate entered password value. (We can’t validate it as other parameters because of encoding)
3. Call corresponding DAO layer’s method to save entity.

* Update user operation:

1. Validate entered user parameters using *ServerValidationUtil*.
2. Call corresponding DAO layer’s method to update entity.

* Delete user operation:

1. Select all vacations for specified user. Delete these vacations and related approval steps one by one.
2. Delete all approval steps for which user's role is used as approver role. If related to approval step vacation has state JUST\_OPENED or WAITING then before deleting approval step it will have been rejected together with vacation. All the approval steps of the related vacation is changed to REJECTED. This step is necessary to exclude situation when there is no users with required role to approve vacation.

*This functionality can be improved in future by adding additional check on how many users with specified role left and delete approval steps if amount of user is less then 2.*

1. Delete approval steps for which specified user defined as approver. If some of the ApprovalSteps have ACTIVE state then their corresponding vacations are rejected before deleting.
2. Delete specified user from the deputies list of all vacation.
3. Call corresponding DAO layer’s method to delete entity.

* Update user’s password operation:

*We can change password only for logged in user. Also you should know that password is saved to the DB encoded using sha1 encoding.*

1. Check that old password of the logged in user is the same as typed one.
2. Call corresponding DAO layer method to update old password on new one.

**Vacation operations logic:**

* Create vacation operation:

1. Check that vacations state is JUST\_OPENED.
2. Check that deputies list is not empty.
3. Validate entered role parameters using *ServerValidationUtil*.
4. Validate vacations period:

* Check that start date >= current date.
* Check that end date <= current date + 90 days (because user cannot take vacation more then 90 days ahead).
* Check that user doesn’t have another vacation at the same period.
* Check that no one of the deputies has vacation at the same period.
* Check if user will not be deputy for another vacations in the same period.
* Check that half of all users with vacation owner’s role will stay at their work places during this vacation.

1. Check that user still have enough vacation days for current vacation.
2. Call corresponding DAO layer’s method to save Vacation entity.
3. Deduct vacation days that is necessary for vacation from user's remaining vacation days.
4. Save changes in user’s remaining vacation days to the DB using corresponding DAO.
5. Create approval flow for vacation (for more details look to the create approval flow operation for ApprovalManager).

* Update vacation operation:

*This operation only possible if vacation was rejected.*

1. Validate entered role parameters using *ServerValidationUtil*.
2. Check that vacation’s state is REJECTED.
3. Validate vacations period:

* Check that start date >= current date.
* Check that end date <= current date + 90 days (because user cannot take vacation more then 90 days ahead).
* Check that user doesn’t have another vacation at the same period.
* Check that no one of the deputies has vacation at the same period.
* Check if user will not be deputy for another vacations in the same period.
* Check that half of all users with vacation owner’s role will stay at their work places during this vacation.

1. Recalculate remaining vacation days for vacations owner.
2. Update remaining vacation days for user using corresponding DAO class.
3. Set state JUST\_OPENED to vacations object.
4. Call corresponding DAO layer’s method to update vacation entity.
5. Delete old approval flow for vacation (for more details look to the delete approval flow operation for ApprovalManager).
6. Create new approval flow for vacation (for more details look to the create approval flow operation for ApprovalManager).

* Delete vacation operation:

1. Check whether vacations state is APPROVED and its start date < then today. If this check fails then user cannot delete vacation.
2. Delete approval flow for vacation (for more details look to the delete approval flow operation for ApprovalManager).
3. Call corresponding DAO layer’s method to delete vacation entity.
4. Recalculate user remaining vacation days.
5. Update user remaining vacation days in DB using corresponding DAO class.

**Approval Step operations logic:**

* Create approval flow operation:

1. Create the list of approval steps for vacation. First we add to the list approval steps for deputies (each deputy should approve vacation). Then we add to the list approval steps for parent roles (one person that has such role should approve vacation). Each of the approval steps has WAITING state.
2. Save approval steps list to the DB using special method of the DAO class that executes batch insert.
3. Fetch first approval step, set his state to ACTIVE and update it’s state using special method in DAO class. After this send email message to this approver.

* Delete approval flow operation:

1. Get active approval step. If it exists then send email about approval step deletion to it.
2. Get the list of all approval steps.
3. Delete the list of all approval steps as batch operation using special method corresponding DAO class.

* Approve (approval step) operation:

1. Check that vacation’s (to which approval step belongs) start date is not lower then current date. This means that we can approve only vacation that doesn’t start.
2. Set approval step state to APPROVED and update it in DB using corresponding DAO class.
3. If this is a first approval step in the flow and vacation’s state is JUST\_OPENED then change the state of vacation to IN\_PROGRESS.
4. Check is there exist another approval steps.

If there exists another approval steps then make next approval step active (change it’s state to ACTIVE and update its record in DB using corresponding DAO class

If another approval step doesn’t exist then change vacation’s state to APPROVED and update its record in DB using corresponding DAO class. After this send email message to the vacation’s owner to notify him that his vacation was approved.

* Reject (approval step) operation:

1. Set approval step state to REJECTED and update it in DB using corresponding DAO class.
2. Update vacation’s state to REJECTED in DB using corresponding DAO class.
3. Send email message to vacation’s owner to notify him that his vacation was rejected.

**Approval flow generation logic**

Approval flow consists from approval steps and is generated during vacation creation. And this process depends on the list of deputies for vacation and the presence of the parent role for the role that has owner of the vacation. Below you can find description of the approval flow logic.

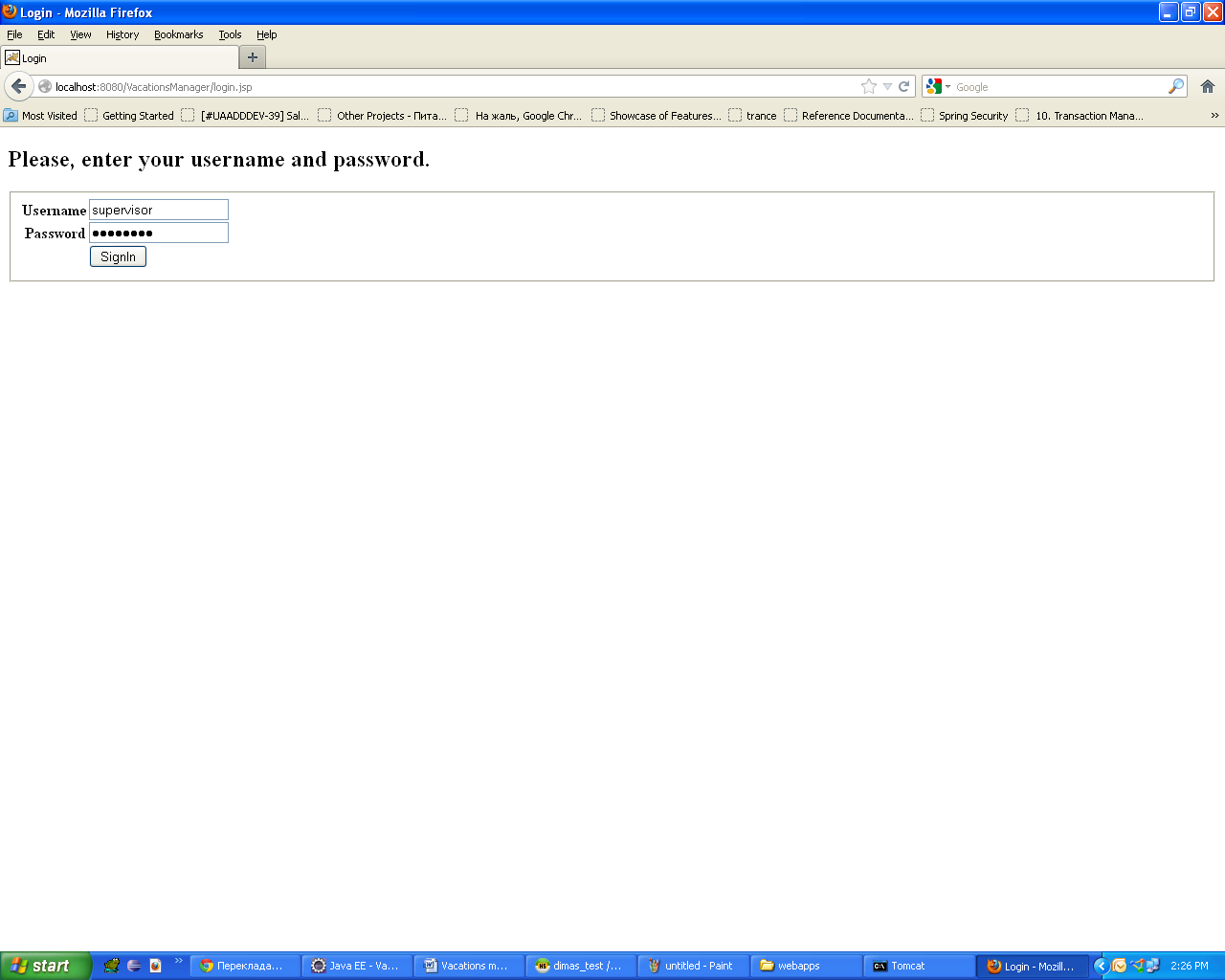
1. First vacation should be approved by all deputies so we generate approval steps for each of deputies and place them in the top of the list. For such approval steps in the DB column APPROVER\_ID will contain the id of the deputy and column ROLE\_ID will be empty (NULL).
2. Next after all deputies vacation should be approved by users who have a role that is parent towards to the vacation owner’s role. So application analyzes the chain of the parent/child roles and generates approval steps for them one by one from lower element to higher. For such approval step in the DB column APPROVER\_ID will be empty (NULL) and column ROLE\_ID will contain the id of the role. Someone with this role should approve your vacation.

Let’s consider an example to clarify all this things. Imagine that we have one deputy for vacation and vacation owner’s role is DEVELOPER. Whole roles chain is DEVELOPER → SUPERVISOR → DEPARTMENT\_HEAD. After creating vacation for such condition we will have next approval flow (in order):

1. Deputy should approve vacation.
2. Some user with SUPERVISOR role should approve vacation.
3. Some user with DEPARTMENT\_HEAD role should approve vacation.

**Vacation’s manager user’s guide**

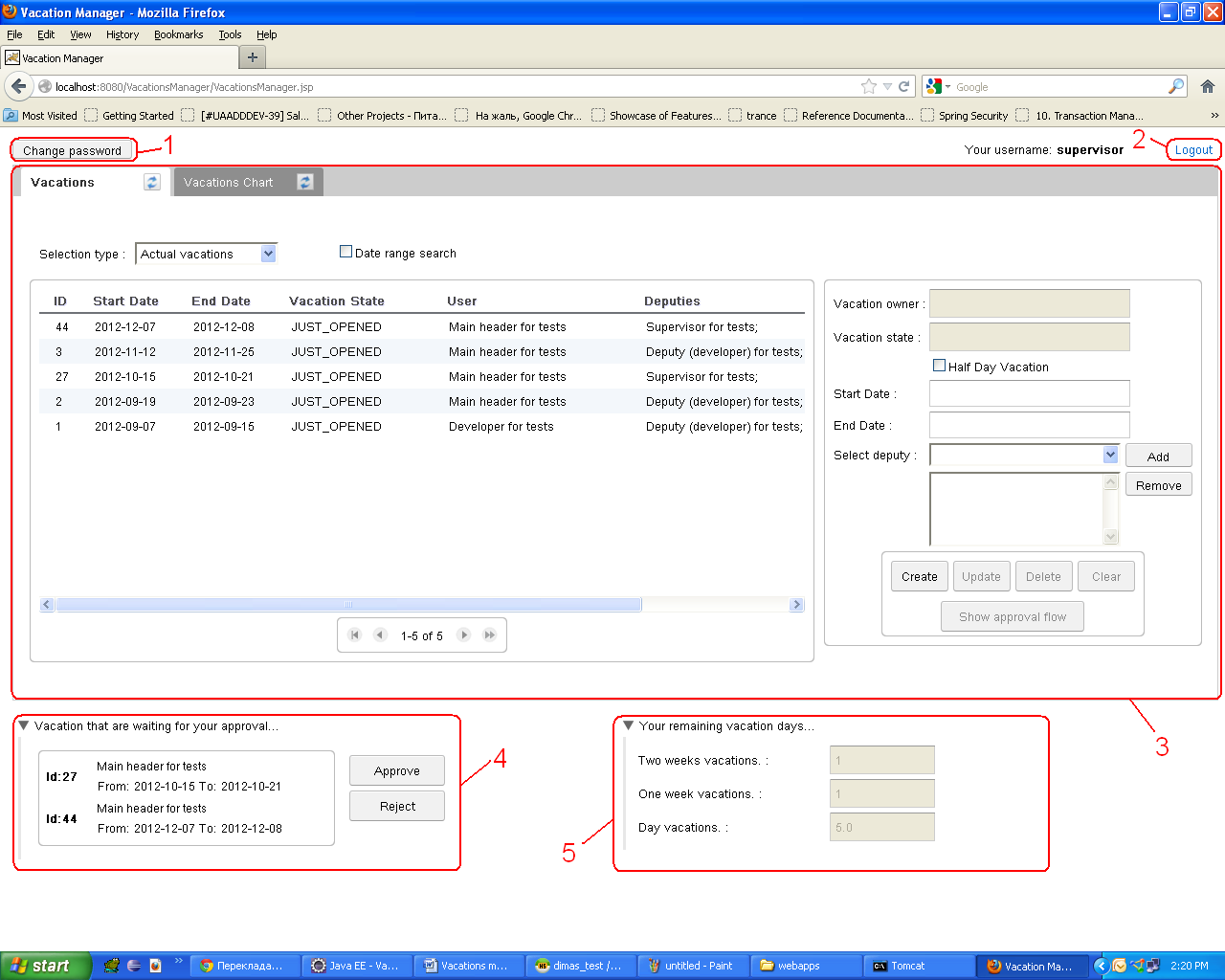
When you try to enter the application then first you will see such page where you will be asked to enter login and password:



Pic. 1. Login page.

*Note. Application will control concurrent access of the users with the same username and allow user to login only once. This means that two peoples cannot connect with the same login at the same moment.*

After user have already successfully logged in he will see the main page of the application (look at picture 2). If he write wrong credentials then error message will be shown.



Pic. 2. Application main page

This main window consists from such parts:

1. Change password button in the top left corner. After pressing this button you will see dialog where you can change your password.
2. Logout link in the top right corner gives you a possibility to execute log out from application.
3. Tab panel in the center of the browser window. Its functionality described below.
4. Drop down list in the left bottom corner that holds the list of vacations that is waiting for current user approval. In this part you can select the vacation from the list that you want to approve/reject and press corresponding button.
5. Drop down list in the right bottom corner where vacation days for currently logged in user are displayed. User cannot change anything in this part. It uses only for information purposes.

Almost all functionality of the application contains in tab panel. The number of tabs differs depending on privileges bound to user’s role. Two first tabs (“Vacations” and “Vacations Chart”) are shown to all users. Another three tabs appears only when users with Admin privilege bound to their roles enter the application.

*Note that each tab header has update button in the right part. This button forces tab to update its content. Also each tab will be updated when it is opened.*

Here is more detailed description of the each tab:

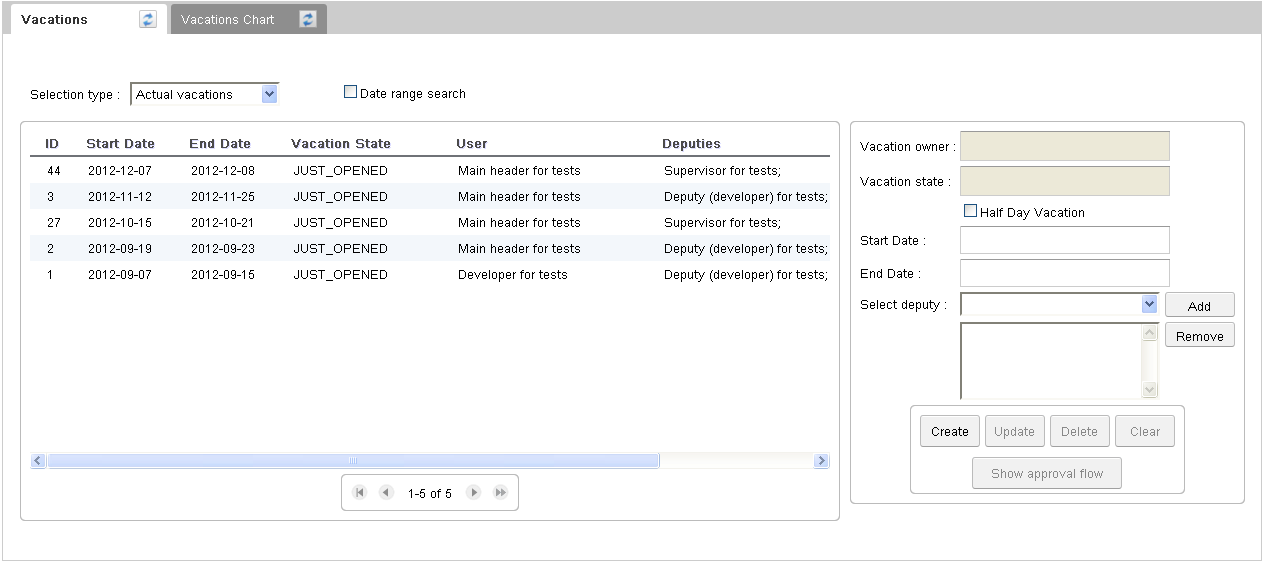
1. Vacations tab.

In this tab you can find table with vacations list displayed using one of the selection types. On current moment such types exist:

* Actual vacations – vacations which end date >= then today.
* All vacations – all vacations from DB.
* Rejected vacations – the list of rejected vacations.

This tab is also using for creating new vacation or updating/deleting existed vacations.

*Note. Only rejected vacations can be updated (such vacations should have status REJECTED). In all other cases “Update” button will be disabled.*



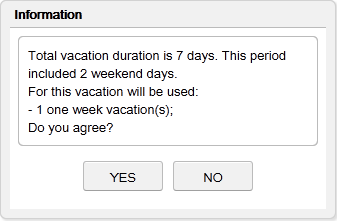
Pic. 3. Vacations list tab

You can take a take vacation for some period (from one day minimum to two weeks maximum) or for half of a day:

* for creating a vacation for a day or more you need just to select start and end date (end date should be greater or equals to start date);
* for creating half day vacation user should choose “Half Day Vacation” checkbox and specify only start date (“End Date” field should be disabled);

Also for taking a vacation you should specify at least one deputy obligatory.

When you press on “Create” or “Update” button then program gets yours remaining vacation days and calculate possible variant how you can use your two weeks, one week and day vacations. After calculation next window is shown to user:

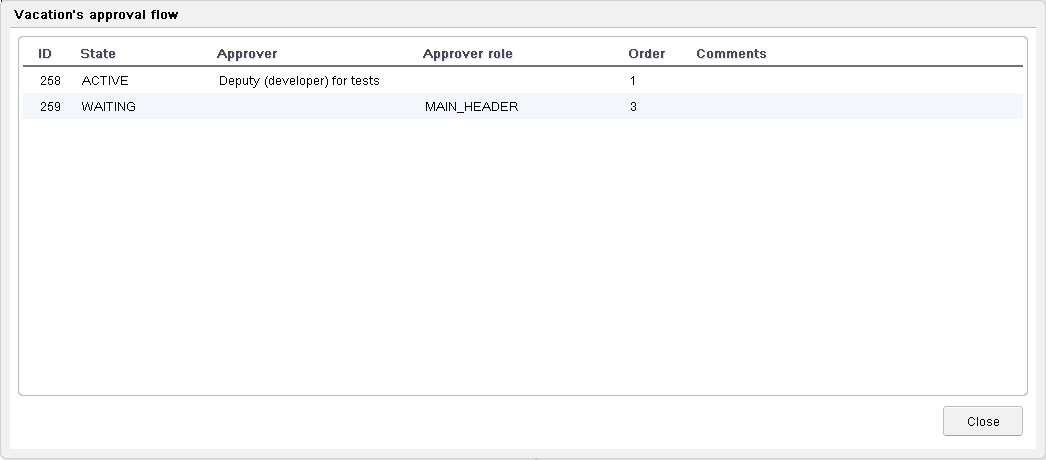


Pic. 4. Used vacations days confirmation message

In this window is displayed total vacation duration in days, weekend and holiday days suring this period and possible usage of your left vacation days. You should accept this message if you agree with used vacation days (only after this actual vacation saving is starting) or decline it and change date interval.

*During calculation of the used vacation days program will automatically try to exclude weekend and holiday days from calculation.*

Also user can see on this screen “Show approval flow” button. After pressing on it window with all approval flow steps will be shown. There you can find who should approve vacation next (approval step with ACTIVE state), who have already approved vacation (approval step(s) with APPROVED state), who should approve vacation (approval step(s) with WAITING state) or who reject vacation (approval step with REJECTED state). Also there you can find the reason of rejection in comments.



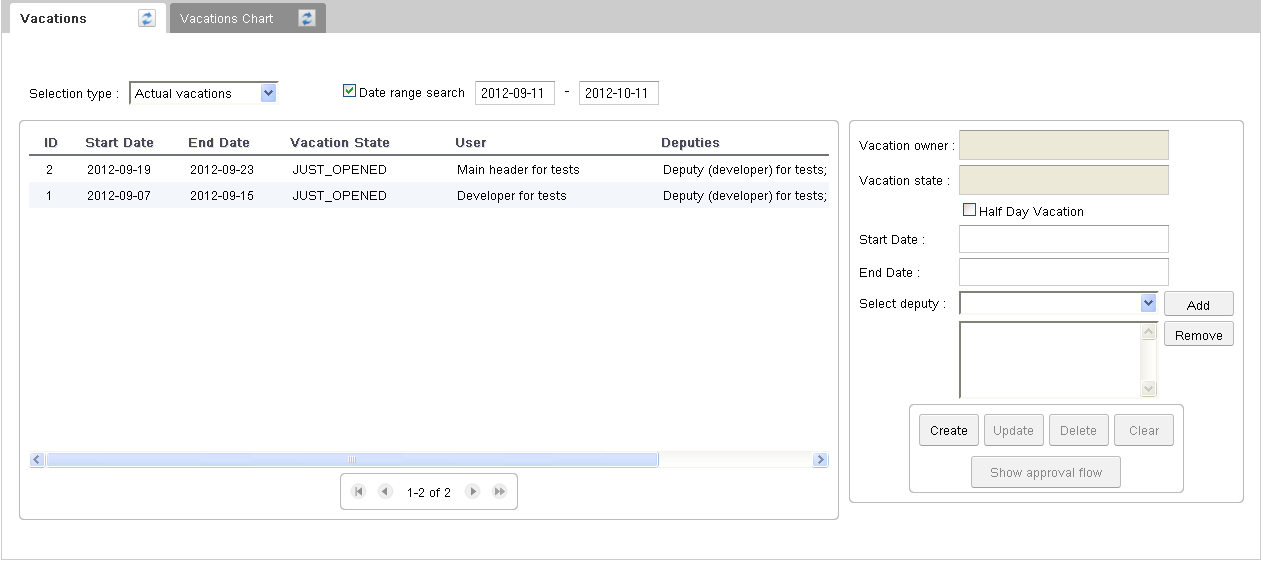
Pic. 5. Approval flow window

If you need you can limit search by date range. This option is available for all search types. The only limitation is that when you search for actual vacations with specified date range your start date of period should be greater or equals then today.

For enabling search by dates range you should select appropriate check box (look at pic. 6). After this start and end date fields will become visible. Default values of the start and end date will be current date. For refreshing table content you can use refresh button in the tab header or choose another selection type combobox.

*I specially didn’t create separate button near date fields because it will only duplicate the functionality of the refresh button in the tab’s header.*

For disabling search by dates range you should just unselect “Date range search” checkbox.



Pic. 6. Vacations list tab with enabled search by date range

1. Vacations chart tab

In this tab you can find chart where displayed all vacations for all users as a plot. So each one can visually see how vacations will be distributed and when it is better to go to vacation.

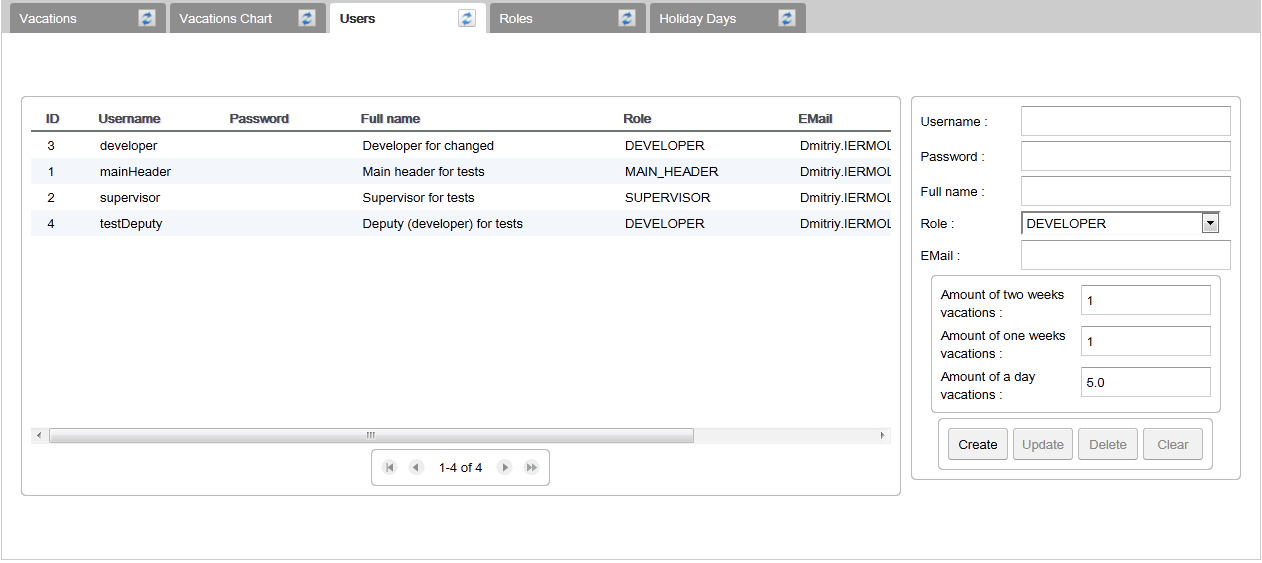


Pic. 7. Vacations chart tab

1. Users tab

*Only users whose roles have administrator privileges have access to this tab.*

In this tab you can view all existed users, their vacations days and also you can create new user or update/delete existed one. Remaining vacation days info also can be updated in this tab too.



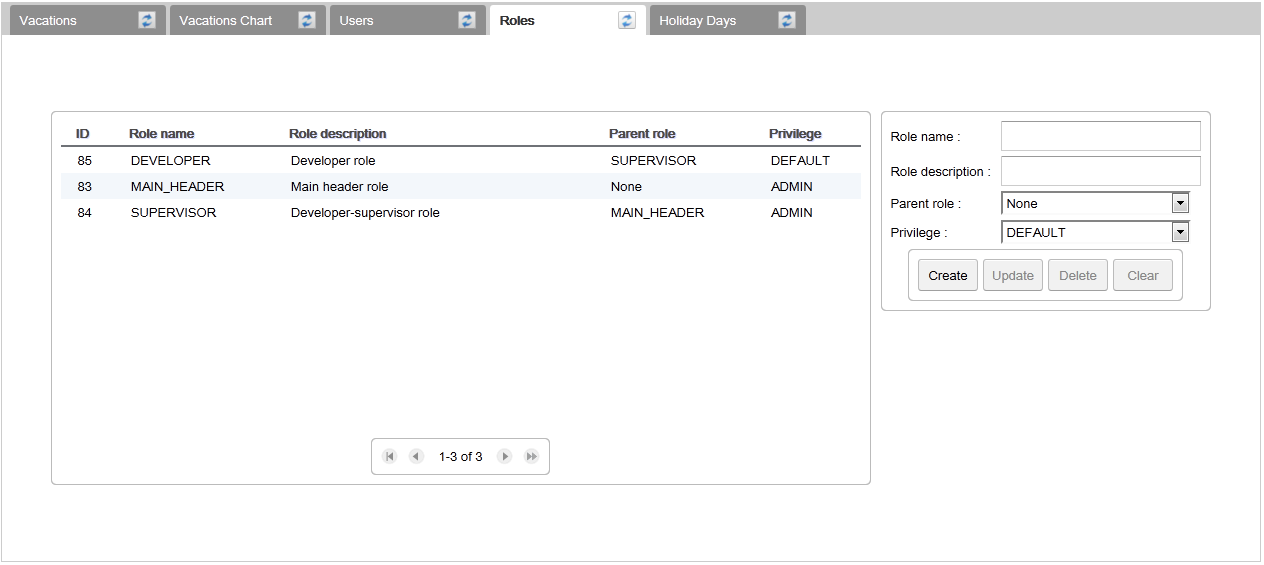
Pic. 8. Users tab

1. Roles tab.

*Only users whose roles have administrator privileges have access to this tab.*

In this tab you can view the list of all roles and add new one or update/delete existed.

*Note. Admin privileges (or rights) are bound to role, not to user.*

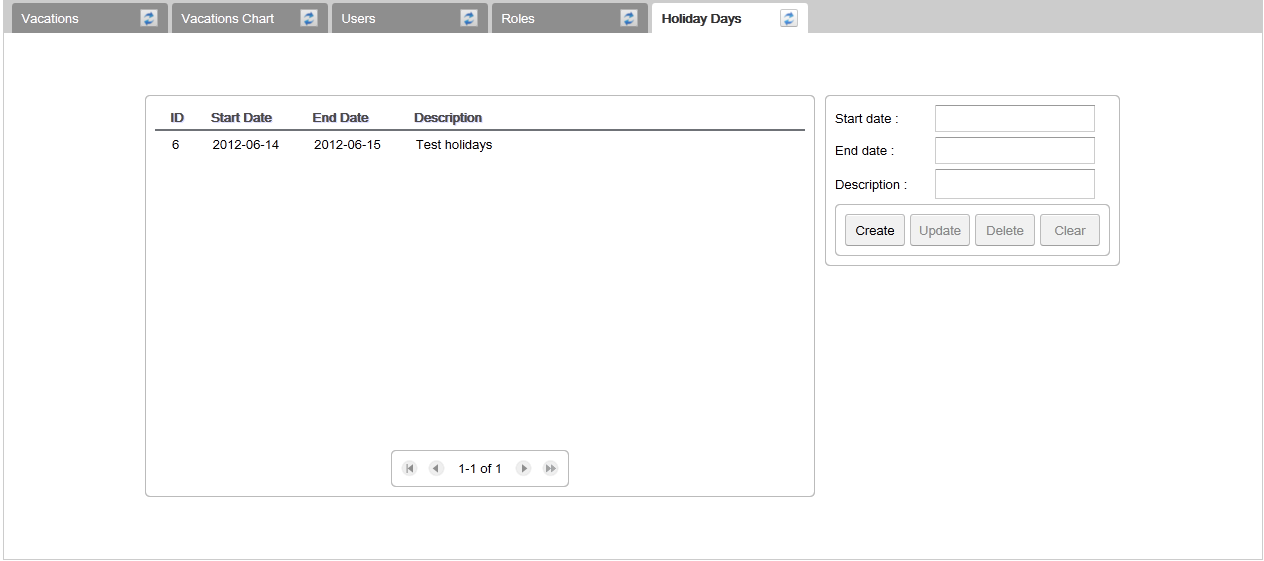


Pic. 9. Roles tab

1. Holiday days tab

*Only users whose roles have administrator privileges have access to this tab.*

In this tab you can view existed holiday days, add new holiday or update/delete existed. This holiday days info will be used during vacation period calculation to exclude such days from vacation duration. But such holiday days should be specified before vacation is taken and will not affect on any existed vacation.



Pic. 10. Holiday days tab

**Some improvements that can be done**

1. Add possibility to build vacations chart on second tab by date range. *As for me this will make tab uglier and that’s why I didn’t implement such functionality. Vacations chart is displayed good enough without such improvements because of limitation on max day ahead for taking vacation. I left commented some code for displaying date range controls in the VacationsChartTab. If you will need to implement this you can uncomment it. For implementation you can look at VacationsListTab because there similar thing have already been done.*
2. Create special tab for editing user’s remaining vacation days.

**How to init database on new computer**

For creating database and all tables you should open and run file *database\_backup.sql* (I tested this with HeidiSQL). It contains scripts for creating database, tables and their relationship. But it doesn’t contain any data. For accessing to the application you should have user and its role to be defined in the DB. Spring security will then compare data entered by user and data, existed in DB. So after creating all tables using script you should create role (with ADMIN privileges) and user for it. You can use such queries to do this:

**insert** **into** roles (id, name, description, privilege, **status**, **version**) **values** (1, 'ADMIN\_ROLE', 'default role', 'ADMIN', 1, 0)

insert into users (id, username, password, fullname, role\_id, email, status, version) values (1, 'admin', '7c222fb2927d828af22f592134e8932480637c0d', 'dummy user', 1, 'dummy\_user@gmail.com', 1, 0)

insert into remaining\_vacation\_days (id, two\_weeks\_vacations, one\_week\_vacations, day\_vacations, status, version) values (1, 1, 1, 5, 1, 0)

After running that queries you will have new user with username “admin” and password “12345678”. Value specified as password in insert query for user is encoded representation of “12345678” string.

*Note 1. After running script file will be created database “vacations” and 16 tables (8 usual tables and 8 log tables).*

*Note 2. After creating custom roles and users from application better to delete this records related to dummy admin user manually from DB. If you will do this from application then this will be mentioned in the log tables and it will only confuse you.*

Queries for deleting dummy user, role, and remaining\_vacation\_days:

delete from remaining\_vacation\_days where id=1

delete from users where id=1

delete from roles where id=1

**Where to find more info about used technologies**

Spring:

1. Official reference documentation for appropriate version: <http://static.springsource.org/spring/docs/3.1.x/spring-framework-reference/html/>.
2. Spring in Action. Third edition. (As I remember I have put this book in the shared folder)

Spring Security:

1. Official reference documentation for appropriate version: <http://static.springsource.org/spring-security/site/docs/3.1.x/reference/springsecurity-single.html>.
2. Spring in Action. Third edition. (As I remember I have put this book in the shared folder)

GWT:

1. Official Google documentation: <https://developers.google.com/web-toolkit/doc/latest/DevGuide>.
2. GWT showcase: <http://gwt.google.com/samples/Showcase/Showcase.html#!CwListBox>

gFlot:

1. There are good description on their official web site: <http://code.google.com/p/gflot/>

GWTHandler (framework for binding Spring and GWT):

1. <http://code.google.com/p/gwt-sl/>
2. <http://krams915.blogspot.com/2011/01/spring-and-gwt-integration-using-maven_08.html>. There is a set of articles on this site so you can look to another too.
3. <https://sites.google.com/site/ggeorgovassilis/usingthegwthandler>

Mockito:

1. They have very good user’s guide on their official site: <http://docs.mockito.googlecode.com/hg/org/mockito/Mockito.html>

For all other used technologies I didn’t look in some concrete place but there is a lot of information about them in the web.