Common Expenses

Web Architecture's course project

Daniele Orler

Index

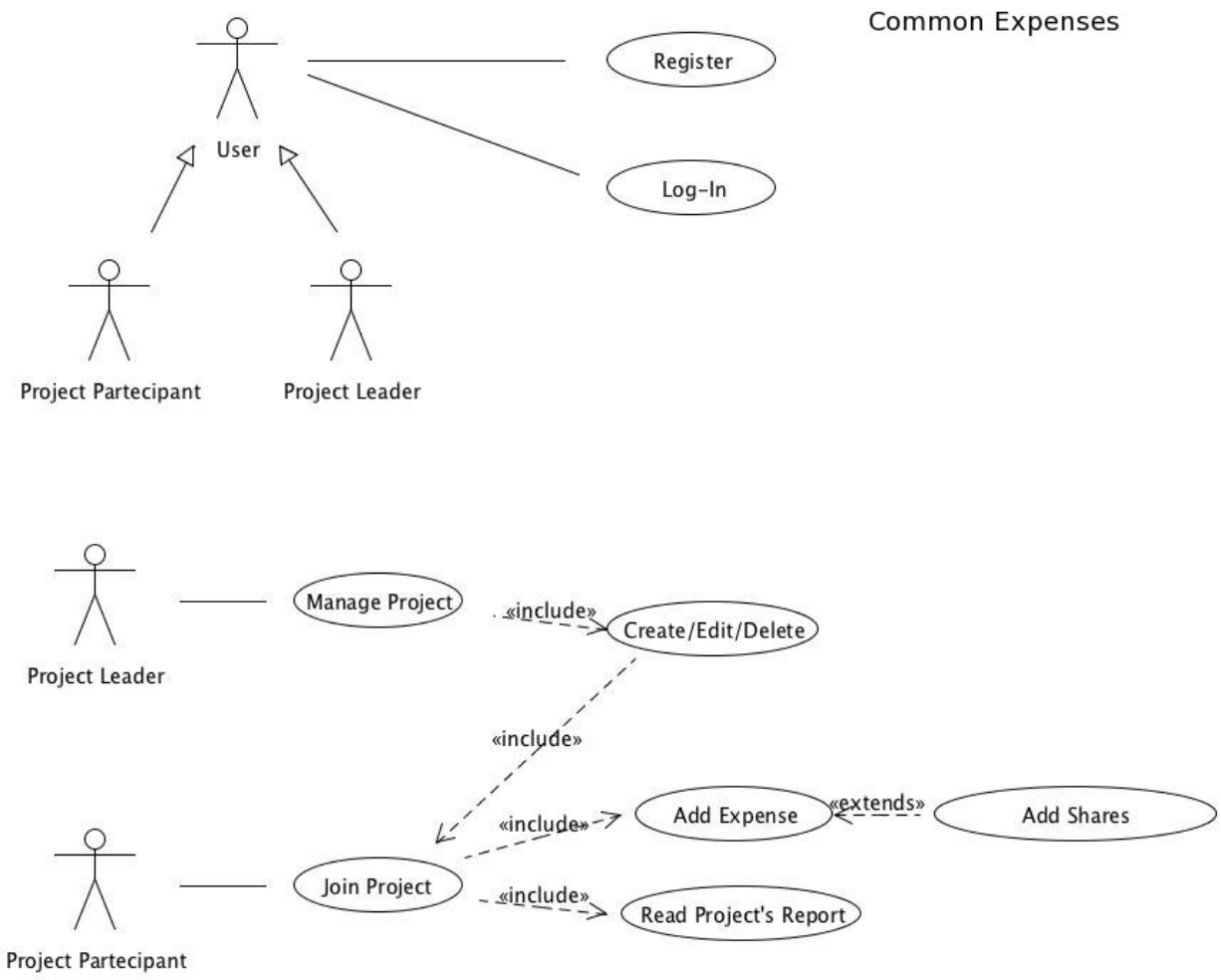
Domain

If you do not live in a big city you problably have to move and live in a flat in order to attend university. Living with other people means, between other things, share the expenses needed to 'run' the flat: bills, accounts, fees etc.
How do you divide the amout? Well it is not so hard I had a big Excel file which does the work.
But this has some problems: everybody needs to ask me to insert expenses or to have the monthly report or …
Wouldn't it be smarter with a web app? Everybody could access the expenses without bother the guy who holds the Excel file.
If we want to generalize the whole thing we can look at the flat rental as a project and the tenants as the project participants sharing the expenses.
That's what i will try to develop: an application which will help keeping track of common expenses on a generic project.

Formal:

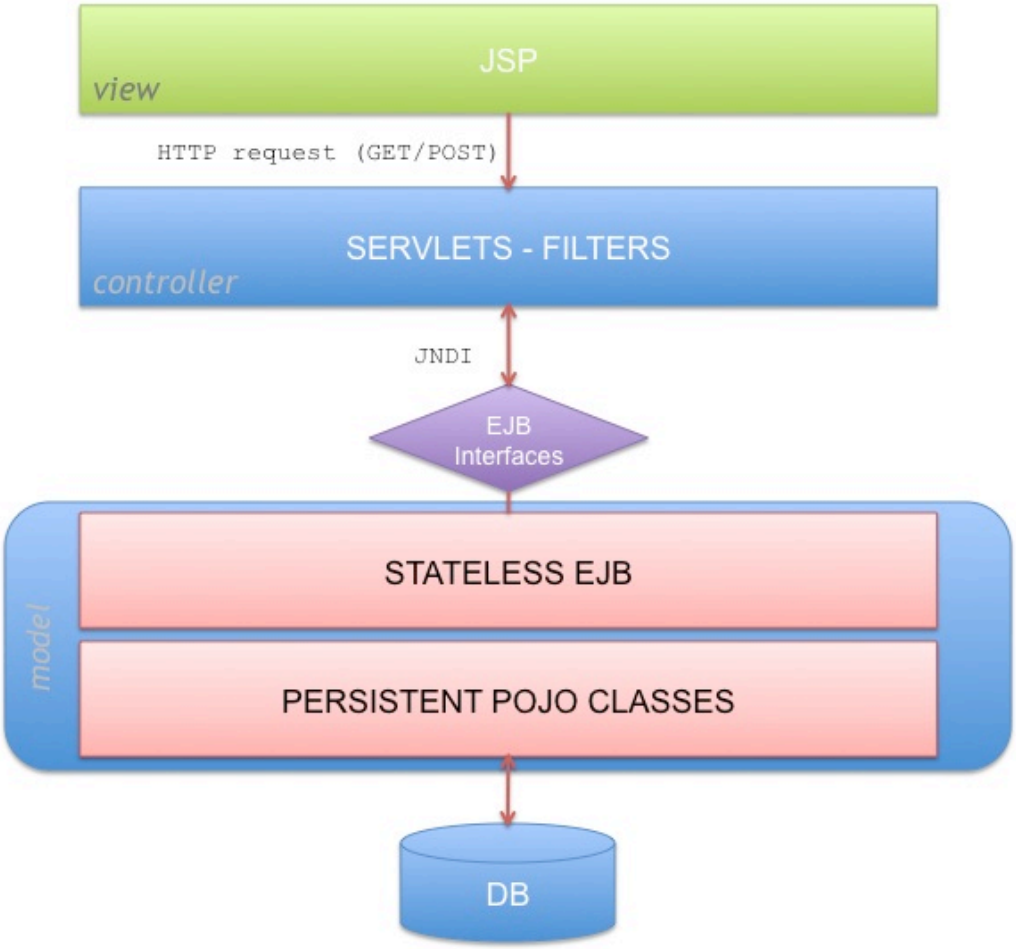
The user is the actor, he can manage the projects, join the projects add expenses to the project and share the expenses between other users.

Here is the use case diagram which explain the domain:



Application Architecture

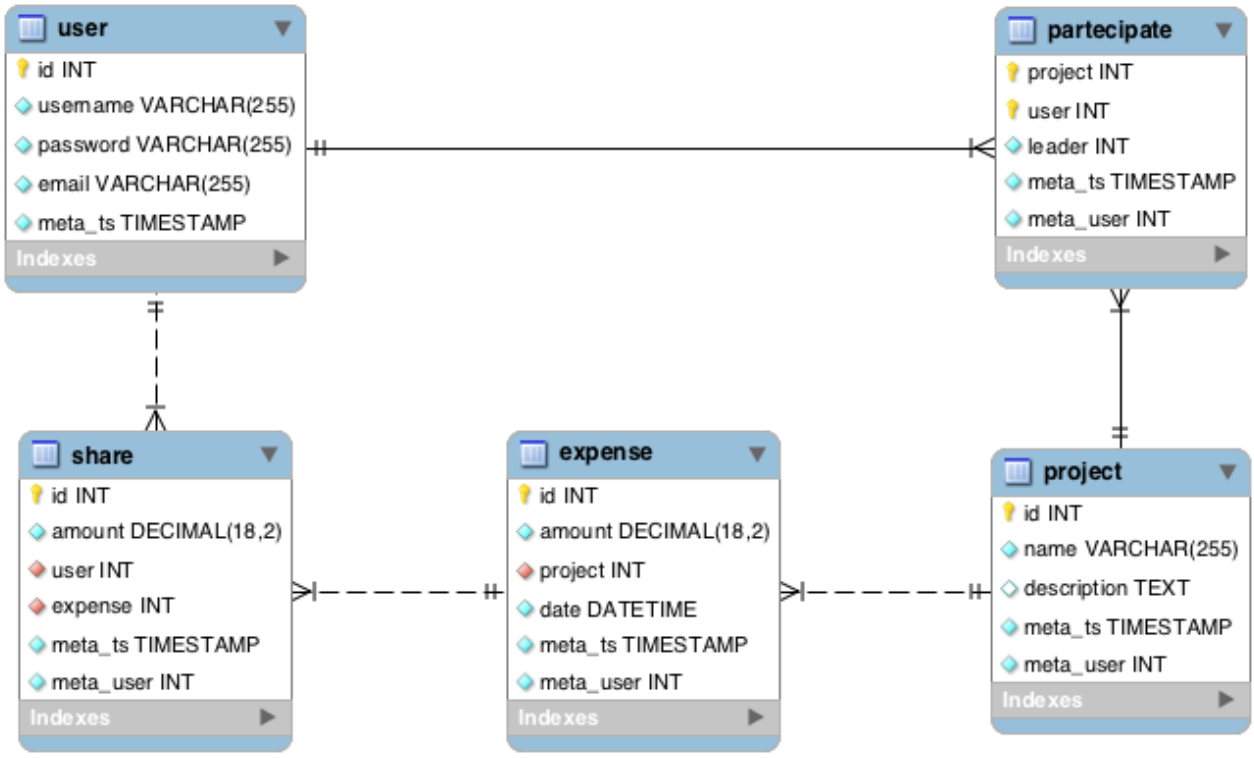
The application try to implement the MVC design pattern.
The persistence layer toghether with the Business Logic layer made up the model of the application.



Model

The persistence layer is implemented with Plain Old Java Classes (Entity classes) annotated with Hibernate's implementation of JPA.
Hibernate maps those classe to the actual tables of the database.

This is the ER schema of the database:



The schema itself should be quite self-explaining anyway here are the main points:

- *meta_ts* and *meta_user* are logging data they represent date and user of tuple creation.
- OneToMany relationship between Project and Expense
- OneToMany relationship between User and Share
- OneToMany relationship between Expense and Share
- ManyToMany relationship between User and Project, there are a join table (Partecipate) for this relationship which hold also the leader of a project.

The Business Logic layer is implemented with Stateless Session Beans (EJB) which manage the Entities. EJB exposes Bean's interfaces in order to let other layers comunicate with the model via JNDI.

Controller

A set of Servlets together with a filter compose the controller layer. The controller retrive data from http requests pass it to the model. Once the model has finished, the result is sent back by the controller. There is a filter which controll that the user is authenticated before he access non public pages.

View

This is what the user actually see. A set of JSP pages with a bit of javascript. The requests are sent to the controller via http post or get request.

Allowed Operation

First of all a visitor to become a user needs to register and the log-in.

Once logged in the user can:

- **Create a new project:** `Your Projects -> Create new Project under the first table`

- **Edit a project he owns (created by him):** `Your Projects -> edit on the row of the project the user wants to edit`

- **Delete a project he owns:** `Your Projects -> delete on the row of the project the user wants to delete`

- **Join an already existing project:** `Your Projects -> join on the row of the project the user wants to join (last table, if there are no rows it means that there are no project to join)`

- **Add an expense to a moine project:** `Your Projects -> add expense on the row of the project the user wants to edit (second table)`

- **Add a share to an existing expense on an existing project:** `Your Projects -> click on the project name the user wants to add the expense to (second table)`

- **Log Out**

How To Run the Project

The project does much of the work on his own.

He just need to find an empty MySQL database.

On src/config/hibernate.cfg.xml you need to set the database configuration, the default database's name is commonexpenses but it can be changed in whatever else.

Once the db is ready lauch the ant script (/build.xml), it will produce a war file under /dist folder. The name of the war file should not be changed.

That war file is ready to be deployed (tested on glassfish 3.1.1)