Project PM DAT076/DIT126, 2013

General

The workshops have been designed and tested to speed up and ease the learning, now you are on **your own**. Anything can happen!

Extremely important: Start out with a simple (but extensible) design. Apply an iterative/modular work process to add use cases. Testing will save time later on!

Version handling

Git is the only accepted (and mandatory) version handling. Works well with NetBeans.

Project groups

You should form groups with 3-4 members.

- Name the group and send name, person number, email to all and possible phone to someone to course responsible. Also send a link to your Git repository (we really prefer *no* login). **Don't use strange aliases for names. It must be possible to identify the members (or send a translation table).** As a confirmation you will get a group number, keep it and use in all communication!

Reporting

Reporting consist of two parts; a demonstration and the delivery of the sources and the documentation.

Demonstration

You must do a public demonstration of your project (approx. 20 min). The demo includes running of the application and a short technical "walk thought". Use the documentation, see below, as a basis, interesting code snippets are ok. **It's the groups responsibility to fully demonstrate the functionality of the application during the demo.** We will not be able to run it later.

- Give us a list of working use cases before the presentation (to check off).
- Also, before the presentation, handle in the self-evaluation (the same day, self-evaluation form on course page).

Sources and documentation

Don't send anything! We will download all sources and documentation from your Git repository. Again: You must supply us with location (and permissions).

It's the groups responsibility to make it possible to grade the sources in about 2 hours. The documentation, if good, makes it possibility to fulfill this. Bad documentation can impact the grading simple because the time will run out. Keep documentation **short**, focused and in synch with the code. The following is very important:

- Clean the project(s)! Unused things makes us confused and will waste time!
- The documentation should contain the following (the format should be: **pure text or pdf** (UML, pictures, any open format), no Javadoc).
    - Group name and number.
    - Group members (incl. pnumb and mail).
    - General overview over the system.
        - What is this? In which area is the system supposed to be used. What is it supposed to do? Etc.
        - A screen shot of the application.
        - Possible users/roles (admin, others,...) and permissions.
        - A list of fully functional use cases (short description, one sentence/use case).
    - Technical design of the system **in the following order** (UML where appropriate);
        - Physical set up (tiers). Imagine a real deployment of the application (not running on a single computer).
        - Participating software components distributed over the tiers (run time support, applications, middleware, libs, ...). Responsibility for each component. Communication between the components.
        - The modules (packages) of each component and the responsibility for each module. Which specific techniques (AJAX, JSF, JPA, ...) are present in each module (if any).
        - A layered view of the application (GUI, application layer, model, persistence, services). Where does the components/modules fit in.
        - The object oriented model as an UML class diagram, similar to the presented shop model.
        - Classes central to understand the application should have a class comment. Purpose? Responsibility? ...
        - If needed comments for methods or statements (optimal is self explanatory code).
        - If there's anything we should know, add a README file.

Project Types

Any kind of web shop is possible, so are games, blogs, ... use your creativity (possible discuss with assistant).

Grading

Max points for the project is 60 p.

General

- The bigger (more realistic) the better!
- The more technologies the better (an artificial school demand, use in separate parts of application).

Style

Badly organized environment and/or bad style will impact the grading, we don't find/understand...

- **Mandatory**: Must be Maven project
- Follow the code conventions for Java code. Correct use of packages. Good naming etc.
- Good organization of the development environment. Keep things collected/separated (HTML, CSS, java, js, sql, ...) it should be easy to find whatever we are looking for!
- Comment at high level (what is done, not how).

Design/Quality

If you hand in a "big ball of mud" we can't judge the project and you will fail.

- Clean design. Layered.
- System as modular as possible using interfaces between important parts.
- No hard coded data.
- Nothing in any way duplicated.
- Minimize dependencies.
- Use design patterns where appropriate (Facade, PRG-pattern,...).
- Encapsulate as much as possible.
- Use Findbugs, STAN or similar to improve quality (we will...)

Testing

- JUnit, QJunit tests for modules will increase the grading.

Technical points

Client parts

- **Mandatory**: (X)HTML, CSS and some JavaScript

Databases

- **Mandatory:** You must use a small database. Database design is not reviewed.
- **Mandatory:** Dumping string to the web is ok if appropriate, but somewhere in the application you must have an ORM layer.
- **Forbidden:** Native SQL

Example points

- 0-29p: A few ugly HTML/JSF pages. No real OO-model. Primitive persistence handling (no ORM). Little functionality. Sloppy design, environment and style. Bad or no documentation (or a lot of auto generated). This will fail.
- 30-39p: A basic web application using one of the presented approaches. A small but correct OO-model. Fair persistence handling using ORM. Basic, usable GUI with some kind of style. A number of use cases implemented. Some features; confirm via mail, possible file upload, ... Design, testing and documentation ok.
- 40-49p: A more realistic web application using at least two of the presented approaches in a meaningful way. More advanced persistence handling. More advanced GUI. A number of use cases implemented. Possible features like above. Possible mashup. Clean design, high test coverage. Good documentation.
- 50-60p: Like the above but even more realistic and advanced. Using all of the presented approaches in a meaningful way.