

Project PM DAT076/DIT126 2014

General

The workshops have been designed to reduce problems and smoothening the learning curve. Now you are on your own. *Anything can happen!*

Extremely important

- Start out with a simple (but extensible) design (you should have a model before starting out. Start with model early in course).
 - Apply an iterative/modular work process to add use cases.
 - Refactor! Test! Will save time later on!
-

Project goal

Three different approaches to web applications have been presented in a rather basic fashion (and also how to handle persistence). There are much more to say about any of the approaches. During the project the goal is to deepen your understanding of one of the approaches. You should select one of the approaches for your project (request-, service- or component-based approach). If needed you may incorporate other techniques (like client-server or other). Any approach must include persistence handling.

Version handling

Git is the only accepted (and mandatory) version handling for the project.

Project groups

You should form a group with 4 members.

- Name the group and send name and members (pnumb and email to all, possibly phone to someone) and a link to your Git repository to course responsible (we appreciate Git repos without login). *Don't use strange aliases for names*. It must be possible to identify the members (or send a translation table or create a mailmap).
- As a confirmation you will get a group number in return. Use it in any contacts with us.

Reporting

Reporting consist of two parts;

- The group makes a demonstration of the application.
- The cloning of the project from the Git repo (we do). *Everything in project must be in repo at deadline (branch master).*

Demonstration

You must do a public demonstration of your project. The demo includes running of the application and a short technical “walk-through” (slides, use the documentation, see below, as a basis, interesting code snippets 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. Approx. time 10-15 min.

- Right before the presentation you should handle in the self-evaluation. See Course pages > Project.
- Right before the presentation you should handle in a list of working use cases (for us to tick off).

Cloning of project

We must be able to download all sources and documentation from your Git repository. Again: You must supply us with location and rights. We appreciate repos without login.

It's the groups responsibility to make it possible to grade the sources in about 2 hours. The following is very important:

- Clean the project! Unused things makes us confused and will waste time!
- The documentation gives us a possibility to speed up. Bad documentation can impact the grading simple because the time will run out. Keep documentation short and focused. The documentation should contain the following (the format should be: pure text or pdf (UML, pictures, any open format), no Javadoc).
 - Group name.
 - Group members (incl. pnumb and mail).
 - General overview over the system.
 - * What is this (a game, an office application, a shop, ...)? In which area is the system supposed to be used. What is it supposed to do?
 - * Possible users/roles (admin, others,...) and permissions.
 - * A list of fully functional use cases (short description, one sentence).

- Technical design of the system in the following order (UML where appropriate);
 - * The object oriented model as a UML class diagram.
 - * Selected approach (and possibly any additional).
 - * Physical setup (tiers)
 - * Participating software components (applications, middle ware, libs, ...) distributed over the tiers. Responsibility for each component. Communication between the components.
 - * The modules (packages) of each component and the responsibility for each module.
 - * A layered view of the application (model, persistence layer, service layers, control and view). Where does the components/modules fit in.
 - * If there's anything we should know, add a README file.

Project suggestions

Your own suggestions are welcome, discuss with course responsible. If you have no idea; here's a couple (use your imagination to fill in the business processes);

Targeting grade 3: Any kind of web shop or blog.

Targeting grade 4: Contact system for a high school (absence, results...), applications for parents, teachers, admin, mail, more technologies used. More realistic, register users, login/out, ... or

Targeting grade 5: Travel Agency, user app. to browse, search, purchase a travel, admin to update add travels, order handling processing. Integrating external RESTful services, application contacts payment services, airline systems, hotel systems (all simulated) some real time features, fancy GUI or...

Grading

General overview

- The more realistic the better!
- The more functionality the better.
- Clean design is very important.
- Must be a Maven project.
- A database must be used (any relational accepted).
- ORM must be used.

A note on size: Expected code contribution for each member is about 800 SLOC (incl. everything: HTML, CSS, JSP, JavaScript, Java, XML, JSF, ..., and also sensible amount of, comments). If you are below this add a README to justify/explain!

Details

The following inspection points will contribute to the overall grading of the project:

- OO Model: There exists an easily identifiable OO model? Complexity, size and quality of model? Model tested?
- Functionality: Number of working UCs. Average complexity UCs.
- Architecture/Design: The different parts of the application are easily identified. Each part has a well defined responsibility. No duplicates of "anything". MVC style (easy to spot what belongs to different parts/layers). The abstraction layers are easily identifiable. Layers of application separated by interfaces (in particular services).
- Clean Persistence layer. Quality of persistence layer implementation.
- Navigation (easy to trace page flows)? PRG-pattern for writes.
- Validation (simple ... exhaustive, all layers?). Any custom conversion (if needed).
- If request based: Correct use of Front controller pattern.
- If JSF: Correct usage and scopes for beans (backing, control,. request...)
- If REST: Correct usage of URI mappings/resources.
- GUI general technical level of GUI (graphical design not much considered). GUI seems to be modular (composing pages).
- Roles (admin, etc) and authorization (simple in-house...advanced using standard JEE and database or OAuth).
- Resource handling: Well designed systematic (ad hoc ... standard JEE). App using i18n?
- Testing: Testing code separated from application. Back-end (services) are tested? Quality of back-end test? Front-end is tested? Quality of front-end test?
- Circular dependencies (use STAN or similar)? Code quality (use Firebugs or similar)?

- Style: Development organization. Easy to locate anything (separation of Java, JS, test, CSS, HTML, etc.). Java packages, hierarchy, naming. Java code style, naming. Java script style, naming (simple functions ... pseudo classical, unobtrusive). Other code style (HTML, CSS, ...). Other (README files, ...)
- Documentation: Code seems to be self documenting? Any file has a short comment (i.e. what is this?). If class file, comment says: “Responsibility, used by ..., uses ...”, tricky passages (unclear methods) commented (but NO other). Good README files used?