Contents

Global productivity through low coupling and high flexibility

Building a movie rental system in a global team, while minimizing the need for detailed specifications and agreements through modern development principles

1 Abstract WRITE LATER

A short abstract, describing our report contents and conclusion

2 Contents WRITE WHEN FORMATTING

What is the content of this report?

3 Introduction

This report is a part of the course BNDN-2013, at the It-University of Copenhagen (ITU). The project is a joint effort between 5 students at the ITU and 4 students in a related group at Singapore Management University (SMU).

The aim of the project for the ITU group, is to develop and deploy a movie rental web service, that the partner team at SMU can client. The SMU team will client the web service and develop a graphical user interface (GUI), so that the web service and the GUI together, form a user friendly movie rental platform. The ITU group should also develop their own client, in order to demonstrate the web service produced.

An important aspect of the project is the successful collaboration between the two groups, that are not only culturally different, but also split by 7 time zones. Our project focuses on how we can use standardisation and technology to minimize the collaborative problems, and ensure a smooth execution of the joint project.

3.1 Problem

We often see how joint development projects with global collaboration end up as projects that uses more time on working out specifications and agreeing on very detailed design issues when these matters aren't actually important from an overall perspective. The endless discussions are often not about improving the user experience or the features of the software, but an effort to try and make the rugged endpoints of tailored software come together and have the exact fit that is needed for the various parts of the project to be regarded as successful.

Our project will focus on developing a generic web service, that leaves a lot of choices open to the client developers. This approach should minimize the need for very detailed feature specification discussions that are probably best handled in the client developer team. To further improve collaboration, we will use standardised, intuitive and well documented REST web-services. We believe these to be more flexible and client-developer-friendly, than the more tightly coupled SOAP approach. By combining a generic web service and a standardised interface, we hope to minimize the need for detailed feature specification and instead focus on establishing common principles. In short, we want to invest our time in establishing a common framework of practices that allow for a high degree of autonomy in the groups, as well as a more effective development process. The key principles, that we think will support our ambition, are low coupling between the server and the client and a high degree of flexibility in the web service produced.

Besides explaining the considerations and decisions in designing a movie rental web service, our project will examine how the use of these principles for low coupling and high flexibility affects the collaboration and end product.

3.2 Problem statement

"Can the establishment of a common practice of using low coupling, high flexibility development principles improve collaboration efficiency in a distributed project?"

- Implement a movie rental web service and 2 clients. One in a joint effort with SMU and one on your own.
- Describe the implementation and collaboration process, and describe the considerations and decisions with regards to the technical implementation of web service and client.
- Analyze how the use of a generic web service and a REST interface with standardised and well documented methods has affected the group collaboration and the final products of the BNDN project.
- Discuss which advantages and disadvantages the low coupling, high flexibility collaboration strategy implies, and how this could be improved in future projects.

3.3 Scientific method

With regards to method, the project primarily tries to prove or falsify our hypothesis, about being able to achieve a better product, through introducing a standardized and flexible development paradigms. This is done through the project as an experiment, with a qualitative analysis of both the process and the final product.

The scope and situation of the project, along with the fact that we only analyze one project, makes it harder for us to generalize and make overall conclusion. But our analysis of this particular project, should help us better understand the dynamics of international collaboration, and the impacts our strategy will have on these. Hence, our project is a qualitative study of the effects of applying certain development strategies in a international and distributed software development project.

3.4 Source code

Source code is included on the enclosed CD, and can also be found (including revision history) on GitHub at: https://github.com/BergarSimonsen/BNDN-Project

4 Design and development

This section describes the design and development decisions in our project, with a particular focus on the modeling and architectural decisions, that made it possible to create a flexible web service, that allows the client to shape the end product in many different ways.

4.1 Domain description

A domain description, that describes the use of a web service by developers, rather than the users working routines is rather unusual. This description, however, helps to clarify the intentions of the project, and the produced web service, so we decided to include it anyway.

Developing a custom movie-rental-system can be a challenge. Developers struggle to get the data-model right, and the handling of video files, payments and access controls can be a great challenge. The developers however want to be able to develop a truly custom app, where they, themselves, can control the way content behave and customize all sorts of features to fit their own great ideas.

This yields for a generic and customizable web service, but we can still identify some key concepts, that are important in this kind of service, and that identifies the system as a movie-rental-platform.

First of all, it's about the movies. The client developer wants to be able to upload movies of different formats, and to tag these files with all sorts of data and metadata. It is also important that the system keeps track of who uploaded the file, and when. It would also be of great use, if the system could automatically register statistic data on movie views, etc.

Another important aspect of the movie rental service is the rating and commenting system. End users can help each other get a better experience with the app, by letting each know which movies are worth watching by giving ratings and reviews to the movie watched.

The client developer also needs a way of collecting payments. Payments can be made both on individual movie basis, or with a subscription. In relation to the payments, the ability to control which movies can be watched by which users at a given point in time is important.

4.2 Web service requirements and analysis

The project came with a set of mandatory requirements for the web service, as stated in the project description:

- The system must consist of a web service and a client.
- Users must be able to log in to the system.
- Logged in users must be able to rent and watch media.
- Users can register or create a new account, read (or display), update and delete (CRUD) account information.
- Admin can add or create new product, read (or display), update and delete (CRUD) product information. Negotiate the product information. The display of the product should provide information such as the thumbnail image, category (for search), and analytic information (recent searches). It should support at least 2 media type (mpeg and avi).

- Users can search for the product. Negotiate what information is needed for the search.
- Users can rent and download product (without payment).
- You can assume that there is already a player/viewer for the downloaded medias. Eg. Windows media player or VLC.
- Most of the data should be stored in the server side.

In order to make our web service as flexible as we wanted it to be, we decided on creating a web service that could support many different, and each more specific and customized media rental systems. This meant building a the service as generic as possible, leaving the more specific requirements to the client.

Since the mandatory requirements state that the system must be a rental system, some sort of payment had to be included. We decided to deal with both pay-per-view and subscriptions, and preferably in a way so that our client-applications would have as much flexibility in creating different product offerings as possible. Building our web service in a generic manner, also implied doing some more generic tagging systems, which allows for some of the mandatory features to be implemented, but also allows for a lot of uses, that are not part of the mandatory project requirements.

Along with our flexible system for media handling, we decided on doing a flexible system for handling permissions. The system should be able to handle permissions on both user and group level, and leave it relatively simple for the developers to utilize the permissions system throughout the code.

An important part of the project was the negotiation of requirements with our corresponding Singapore Group. The Singapore Group however, wasn't so ambitious and didn't have a lot of demands. This meant that our initial suggestion for web service features covered all their needs, rendering the negotiation of requirements rather trivial.

4.3 Limitations and scope

Our project is limited to primarily creating a flexible, functional and generic web service, and not focus as much on the business part. Although there is a subscription/pay-per-view element implemented in the system, the project doesn't focus too much on this part. We give the clients the ability to handle orders in the system, data wise, but we do not support any kind of real payment system or handling of real payment transactions.

Neither do our project consider any kind of media rendering or conversion. The media is served back as uploaded to the system, with no consideration of which media types might be supported in different contexts. The project doesn't deal much with statistics either. Some details can be collected, and we will consider to include some automatic statistics as well, but it's not the primary focus of the project.

The common attribute of all the limitations we have put down on the project, is that they aren't very important for developing a functional system on top of the web service, and that they could relatively easily be added to the system later on. Some aspects, such as real payments, conversion, etc. would be extremely important for using our web service in a real life context. For our project though, the abstractions and limitations on these aspects are completely fine, since we can still test our hypothesis about collaboration in an international, distributed development project, as the missing features mentioned doesn't really affect the way we work with the project.

4.4 Final System requirements

Based on the mandatory requirements, the extra features introduced for this project, the limitations and scope for this project and the use case analysis (see appendix 2), this is our final system requirements for the movie rental web service:

Functional requirements

- The system must consist of a web service and a client.
- Users can create a personal profile on the system.
- Registered users must be able to log into the system
- Users must be able to rent and watch media (download or stream)
- Users must be able to post/edit/delete comments and ratings on media they have watched.
- Users can read comments/ratings from other users.
- User can pay for a specific media or buy a subscription which allows the users to watch all media for a limited time
- User must have the ability to search for media based on tags.
- Users can have a list of their favorite media.
- Users can receive recommendations based on what they have watched in the past.
- Distributors can upload/edit/delete media. This includes media files as well as meta data (information) about the media.
- Distributors can connect tags to their media, making them easier to search for.
- System admin can send private messages to users.
- System admin can give/take away functionality from users.
- System admin can block users.
- System admin can remove comments/ratings that users have posted.
- System admin can remove media.

Non-functional requirements

- All single entity calls must return a response in less than a second (not taking external factors into account, such as bad/slow internet connection etc.)
- Persistence. The system must be able to restore data (user data, media data etc) in case the server breaks down.

4.5 Web service design

This section describes the system design and architecture in our web service. The section will describe our considerations and explain how our system design and architecture, makes the system support our ambitions about functionality and flexibility.

4.5.1 Designing a generic web service

A key part of our project, is to make the web service generic and to make it accommodate for as many client app scenarios and features as possible. That is why we have put great effort into designing the data model and web service in a flexible way, that allows the client developer to make most of the decisions with regards to what data should be available, and how the system should be used in general.

While the flexibility is very useful from a technical point of view, the approach also gives the client developers more responsibility, and requires them to put greater effort into configuring the system to fit their exact needs. Though we tried to make these configurations simple and intuitive, it is clear that our approach leaves a lot more concepts for the client developers to grasp, and a lot more decisions for them to consider.

We did however decide on doing a truly generic and flexible web service. This was both to test our own ability to built such a system, but also to see how well the less technically experienced SMU group of client developers would deal with these concepts.

4.5.2 Using REST as the web service interface

REST is both a set of principles and a protocol, that form a modern standard for web service interfaces together. A REST web service uses standard HTTP-requests, such as GET and POST, and gives standardized responses, typically in JSON or XML, in plain text data without any mandatory overhead for types or similar. This part of the REST philosophy suits our project well, since it's a standardized, accessible and light-weight protocol, that can be accessed from almost all other programming environments and languages.

But REST is more than just a protocol, it also comes with a lot of best practices for what features the web service should have, and how it should make them available. A web service is said to be RESTful when it incorporates these principles. One of the principles, is that the server should be stateless, and all

state information encoded in the requests[1]. This leaves the server without the need to keep track of session states, and allows it instead to focus solely on data manipulation and representation. The stateless servers also allows for easier distribution of computing efforts, and therefore scalability - for example in a cloud environment. Other principles includes a standardized access to entities and loose coupling. This project shall not discuss the deeper aspects of the RESTful principles, but instead refer to *Roy Fiedling's doctoral dissertation*[2] for some interesting thoughts on the principles.

The obvious alternative to using REST would be the more tightly coupled and strongly typed SOAP protocol. This protocol has the advantages of a stronger coupling, more automated type checking and similar features that can be helpful to the client developer, but requires a heavier and more complex protocol. For our project, and our ambition to develop a flexible and loosely coupled web service, the choice of REST as our protocol and the incorporation of RESTfull principles was the obvious choice.

[1] REST is an acronym for representational state transfer
[2] Kilde???