Rent It

Second-Year Project,
Bachelor in Software Development,
IT University of Copenhagen

Group 12
Jakob Melnyk, jmel@itu.dk
Frederik Lysgaard, frly@itu.dk
Ulrik Flænø Damm, ulfd@itu.dk
Niklas Hansen, nikl@itu.dk
Jacob Claudius Grooss, jcgr@itu.dk

April 26th, 2012

Contents

1 Preface

This report is the result of a project on the bachelor in Software Development at the IT-University of Copenhagen spanning from the 2nd of February 2012 to the 23rd of May 2012.
The project was given on the second semester of the bachelor and corresponds to 15 ECTS-credits.

The project ("Software Development in Large Teams with International Collaboration") is described as centered on developing and implementing a rental service for digital media. The is goal is to develop a client-server solution backed by a relational database system.
Students work in the teams of four to six people and collaborate and negotiate with students from Singapore Management University (SMU) in English.
This specific project was done by five ITU students who collaborated with three SMU students.

The challenges of this projects were:

- Dealing with cultural gaps and time differences in communication with team members from Singapore.
- Sharing a balanced amount of information to have successful collaboration.
- Reaching agreements that both ITU and SMU students found satisfactory.
- Conducting constructive feedback on fellow students work.
- Constructing a client/server solution in C#.

Whenever we reference the Bibliography (found on page 13), we use the signature [1] to indicate such. [1] refers to the link to a blog describing "The Way of Testivus".

1.1 Version control repository and Scrum tracker

We use Git (at GitHub) as our version control system. The repository is public and can be found at

- github.com/itu-bswu/RentIt

In addition, we use a free tool called Pivotal Tracker to track our user stories for the Scrum[1] development model we have chosen to use. This tool can be found at

- pivotaltracker.com/projects/492063

1.2 Dictionary

This list describes some of the more uncommon terms we use in this report.

BugJail If there is ever a situation where a teammember has five issues attached to them, they are placed in "BugJail" and are not allowed to work on new code until they have cleared some of the issues.

Feature Freeze We have set our feature freeze date as May 8th. After May 8th, we are not allowed to add new features to our system.

Code Freeze Our code freeze date is May 15th. After this date, we are not allowed to add code to our project (unless extremely important).

[1] Further detailed in our **??** chapter on **??**.

2 Project overview

In this chapter we discuss our take on the project "Software Development in Large Teams with International Collaboration", what we feel are the important parts of the project, the must-haves of the final product and the assumptions we make going into our requirement specification.

2.1 Problem analysis

Not too many years ago, media rental of physical media was a lucrative business to be in[1]. The last couple of years have been hard on companies making their business in physical media rental[3]. This is, at least partially, due to the increasing popularity of companies like Netflix[4] making it easier to rent media digitally, thus enabling users to do it from home and not spend time going to the actual shops.

To create a media rental service that would be seen as interesting (if not competitive), it has to be:

- Easy to use[2]
- Price competitive[3]
- (Optionally) Offer an expanded array of services compared to other services.

A wide array of media rental services already exist for books, movies/films, music and other media, so there are many sources to draw inspiration from. In addition, media rental services do not necessarily have to be run by private companies. Some institutions (like libraries[4]) offer similar services for citizens.
A service does not necessarily have to focus on one kind of media (like Netflix), as evidenced by Apple's iTunes Store[7].

In addition to subjects concerning normal users of the service, the project description also mentions administrators. Administrators can upload, delete and edit movie information on the service. Because these types of administrator users provide content, we have decided to refer to them as `Content Providers`. We make this distinction because we have a user type we called `Admins`[5].

2.2 Assumptions and decisions

In order to narrow down the focus and requirements for our system, we make some assumptions and decisions in addition to the points raised in the problem analysis.

2.2.1 Choosing a service type

As described in the problem analysis, the existing types of media rental services can be narrowed down to a) free public library rental type and b) paid media rental. They can be very similar (depending on development choices) and both types present some security issues (user information, credit cards, etc.).

[1] Blockbuster LLC[2] is an example of a successful company in the media rental industry.
[2] Piracy is a major concern, and if the service does not provide something that is (at least) just as easy to use, people would rather be inclined to download illegally rather than pay for content from a service[5].
[3] Rarely a problem with piracy, but if one services provides the same amount of media, support and access, price is certainly a factor.
[4] Roskilde Bibliotek is an example of a danish library providing similar functionality[6].
[5] `Admins` are explained in further detail in our Design chapter (page 8).

We decided to develop a paid media rental system, as we felt it had more options (such as payment models) in terms of functionality that could be implemented.

2.2.2 Choosing a media type

At first we wanted to make a streaming service for TV shows. This could involve paying for a single episode of a TV show or for a full season. After doing some research on what streaming would involve (compared to just downloading and saving a file), we changed the way we let users access our content. Instead of doing the streaming version, we decided to just let users download movie files and store them on their system.

We also changed our intended media type. While we felt it could have been more interesting to do TV shows (compared to other types of media), we decided to pick a slightly less complex system and instead focus on designing the service to offer movie rentals. Designing the service to handle movies instead of TV shows would not make them vastly different, so we decided to focus on movies.

2.2.3 Digital Rights Management

Digital Rights Management (DRM) is an issue we will most likely run into. While we may limit how long users have active rentals on the service, there is a technical challenge in making sure users cannot view the downloaded files after rentals have expired. We do not consider DRM functionality core in the service, but we do have it as an optional goal for our service.

2.2.4 Author rights

Author rights is another concept to consider. When we give `Content Providers` the rights to upload movie files, they may be able to abuse this by uploading files they do not have author rights to. We do not consider this a central focus point in our system, but in order to ensure a great quality service in a broader perspective, some sort of validation of uploaded material should be considered.

3 Requirements

This chapter describes the requirements (and optional features) for our system and project. We have translated the required features and some of the optional features into use cases. We create tests[1] and workflows for our system from the use cases, so that we can document and make sure that our system fulfills our requirements. The use cases are described on page 6.

3.1 Required system features

These required features

-

3.2 Optional system features

3.3 Additional project requirements

3.4 Use cases for the system

User management

-

Browsing media

-

Media rental

-

Content management

-

[1] Further described in our Testing chapter on page 11.

4 Cooperation

4.1 Cooperation

4.2 ITU group structure

4.2.1 Meetings

4.3 SMU cooperation

4.3.1 Meetings

4.3.2 Conflicts

4.3.3 What we could have done differently

5 Design

5.1 Database

5.1.1 Analysis

Decisions

ER-model

SMU involvement

5.1.2 Tables

5.1.3 Entity Framework

5.2 Service

5.2.1 Analysis

5.2.2 Architecture

5.2.3 Interface

SMU involvement

5.3 Client

5.3.1 Analysis

5.3.2 Architecture

5.3.3 Graphical User Interface

Usability

6 Implementation

6.1 Service

6.1.1 Architecture implementation

Issues, workarounds and fixes

6.1.2 Error handling

6.2 Client

6.2.1 Architecture implementation

Issues, workarounds and fixes

6.2.2 Error handling

7 Manual

7.1 Client

7.1.1 Navigating the client

7.2 Service

8 Testing

8.1 Strategy

8.1.1 Test types

Scenario-level tests

Service-level tests

Graphical interface tests

8.1.2 Regression tests

8.2 Test results

8.2.1 Code coverage

8.3 Reflection on test strategy

9 Conclusion and reflection

9.1 Issues and potential fixes

9.1.1 Issues we would prioritise

Bibliography

[1] Blog on "The Way of Testivus": http://www.artima.com/weblogs/viewpost.jsp?thread=203994 (Checked on 22nd of April, 2012)

[2] Wikipedia entry on Blockbuster LLC: http://en.wikipedia.org/wiki/Blockbuster_LLC (Checked on 9th of May, 2012)

[3] Blockbuster revenue loss: http://www.thewrap.com/movies/column-post/blockbuster-announces-q2-results-gets-debt-extension-20105 (Checked on 9th of May, 2012)

[4] Wikipedia entry on Netflix: http://en.wikipedia.org/wiki/Netflix (Checked on 9th of May, 2012)

[5] Interview with Gabe Newell (co-founder and managing director of Valve Cooperation): http://www.tcs.cam.ac.uk/story_type/site_trail_story/interview-gabe-newell/ (Checked on 9th of May, 2012)

[6] Roskilde library website: http://www.roskildebib.dk/ (Checked on 9th of May, 2012)

[7] Wikipedia entry on ITunes store: http://en.wikipedia.org/wiki/ITunes_Store (Checked on 9th of May, 2012)

A Written Review

A.1 Layout

Even though the first draft is only ten pages, it feels odd not to have a table of contents (ToC). A ToC would have made it easier to navigate the document.
Section/subsection titles were not very distinct from the standard text. Either some form of numbers or letters could be used to show that a new section begins. Italics could also be considered.
The description of the data model and the web-service felt quite clustered and had no real distinction between when one ended and the other began. Again sectioning could alleviate this issue.
Bullet points (or some other "fancy" representation) of the methods in the web-service description would have been good. It was not very clear what was a method and what was not.
The communication section could have been improved by splitting it into subsections. There are three or four subjects discussed in the section and each of them could have had their own subsection.

A.2 Content

The ER-model notation is mentioned (and Søren Lauesens book referred to) in the communication section, but not in the data model section. It probably should be, as the data model section lacked an explanation of how to read the ER-models.
The test report section could have been improved by showing (and explaining) a template, then giving an example of an actual report log entry. Another option could be to just explain the report log snippet you have actually included, instead of just leaving it in there with no explanation.
The text has a lot of poorly argued for decisions. One of particular note is "Revised ER_model. If your smart you will notice that there is no distinction between admins and normal users, but just use your imagination and trust us n this one."
Another example of poor argumentation: "The way we have been testing out system is far from the most optimal way, but it is the only way to do it." A tool such as Pex is mentioned, but it would be nice if there were a more detailed explanation as to why your system is designed in such a way that only manual testing is possible. It seems weird because it is possible to make tests that do exactly what you are doing in your system. They can just do it automatically. Spamming could be avoided by just being sensible about how often/when you test.
Grammar and spelling errors: We assume this is because of no proofreading being done, but this should be a priority come the final hand-in. One example is the label for the revised data model as quoted previously. Just that label has two spelling errors.
The use cases could use some elaboration (in terms of describing each use case), but the illustration works quite well, and you could possibly just explain the more complicated ones (what does it mean to update a movie?).
You have a good problem introduction (besides the spelling and grammar problems).
Description is a bit lacking on the data models, but the way you present them is quite good.

B SMU meeting logs

C System Diagrams

C.1 Class diagrams

C.2 Service API

D Test results

D.1 Result table

E GUI images

E.1 Hand-drawn sketches

E.2 GUI prototype

F Original Use Cases

F.1 User account management

- A user is signing up for the service.
- A user log in to the service.
- A user edits his/her personal information.

F.2 Media browsing

- A user is browsing the newest added movies.
- A user is browsing the most downloaded movies.
- A user is browsing movies by genre.
- A user is browsing all movies and sorts them by their name or genre.
- A user is browsing all movies he/her previously rented.
- A user is searching for at movie by its name.

F.3 Media rental

- A user is renting a movie.
- A user is viewing information about a movie.

F.4 Content management

- A content manager uploads a new movie, and enters information about that movie.
- A content manager edits information for a movie.
- A content manager deletes a movie.

F.5 System management (?)

- A system manager browses all the content managers.
- A system manager creates a new content manager.
- A system manager deletes a content manager.
- A system manager browses all the users.
- A system manager deletes a user.

G F# Handins

This chapter of the appendix contains our handins for the F# assignments. To make sure they could fit on the pages, we had to break up some of the lines in the code. We do feel that we have been able to make it look decent, however. The original .fs files have also been included on the DVD including with the report.

G.1 F# Handins - Frederik Lysgaard

G.1.1 HandIn 1

G.1.2 HandIn 2

G.1.3 HandIn 3

G.1.4 HandIn 4 & 5

G.2 F# Handins - Jacob Claudius Grooss

G.2.1 HandIn 1

```
module HandIn1
//Exercise 1
let sqr x = x * x

//Exercise 2
let pow x n = System.Math.Pow(x, n)

//Exercise 3
let dup (s:string) = s + s

//Exercise 4
let rec dupn (s:string, x) = if x = 0 then s else s + dupn(s, x - 1)

//Exercise 5
let timediff (hh1, mm1)(hh2, mm2) = (hh2 * 60 + mm2) - (hh1 * 60 + mm1)

//Exercise 6
let minutes (hh, mm) = timediff (00,00)(hh,mm)
```

G.2.2 HandIn 2

```
//Exercise 7
let rec downTo n = if n < 1 then [] else n :: downTo(n - 1)

let rec downTo2 n =
    match n with
```

```
                | n when n < 1 -> []
        | 1 -> [1]
        | _ -> n :: downTo(n-1)

//Exercise 8
let rec removeEven (xs: int list) =
    match xs with
        | [] -> []
        | [xs] -> [xs]
        | xs :: xy :: rs -> xs :: removeEven(rs)

//Exercise 9
let rec combinePair (xs: int list) =
    match xs with
        | [] -> []
        | [xs] -> []
        | xs :: xy :: rs -> xs :: combinePair(rs)

//Exercise 10
let explode (s:string) =
    s.ToCharArray() |> List.ofArray

let rec explode2 (s:string) =
    match s with
        | s when s.Length < 1 -> []
        | _ -> s.[0] :: explode2 (s.Substring 1)

//Exercise 11
let implode (s:char list) =
    List.foldBack (fun str ch -> string(str) + string(ch)) s ""

let implodeRev (s:char list) =
    List.fold (fun str ch -> string(ch) + string(str)) "" s

//Exercise 12
let toUpper (s:string) =
    implode (List.map (fun x -> System.Char.ToUpper x) (explode s))

let toUpper1 (s:string) =
    explode >> (List.map (System.Char.ToUpper)) >> implode

let toUpper2 (s:string) =
    explode s |> (implode << List.map System.Char.ToUpper)

//Exercise 13
let palindrome (s:string) =
    (explode s |> implodeRev |> toUpper) = toUpper s

//Exercise 14
let rec ack (m, n) =
    match (m, n) with
```

```
        | (m, n) when m < 0 || n < 0 -> failwith "The Ackermann function
                                    is defined for non negative numbers only."
        | (m, n) when m = 0 -> n + 1
        | (m, n) when n = 0 -> ack (m - 1, 1)
        | (m, n) -> ack (m-1, ack(m, n-1))

//Exercise 15
let time f =
    let start = System.DateTime.Now in
    let res = f () in
    let finish = System.DateTime.Now in
    (res, finish - start);

let timeArg1 f a = time(fun () -> f(a))
```

G.2.3 HandIn 3

G.2.4 HandIn 4 & 5

G.3 F# Handins - Jakob Melnyk

G.3.1 HandIn 1

```
module Module1

// Exercise 1
let sqr x = x*x

// Exercise 2
let pow x n = System.Math.Pow(x, n)

// Exercise 3
let dup s : string = s + s

// Exercise 4
let rec dupn (s:string) x =
        if x>=1 then (if x = 1 then s else s + dupn s (x-1)) else ""

// Exercise 5
let timediff (hh1, mm1)(hh2, mm2) = (hh2*60 + mm2)-(hh1*60 + mm1)

// Exercise 6
let minutes (hh, mm) = timediff(00, 00)(hh, mm)
```

G.3.2 HandIn 2

```
module Module2

// Exercise 7
let rec downTo x =
```

```
        if x < 1 then [] else (if x = 1 then [x] else x :: downTo (x - 1))

let rec downTo2 x =
    match x with
    | x when x < 1 -> []
    | 1 -> [1]
    | _ -> x :: downTo2 (x - 1)

// Exercise 8
let rec removeEven (x:int list) =
    match x with
    | [] -> []
    | [xs] -> [xs]
    | xs :: ys :: zs -> xs :: removeEven zs

// Exercise 9
let rec combinePair (x:int list) : (int * int) list =
    match x with
    | [] -> []
    | [xs] -> []
    | xs :: ys :: zs -> (xs, ys) :: combinePair zs

// Exercise 10
let explode (s:string) = List.ofArray (s.ToCharArray())

let rec explode2 (s:string) : char list =
    match s with
    | s when s.Length < 1  -> []
    | _ -> s.[0] :: explode2 (s.Substring 1)

// Exercise 11
let implode (cl:char list) : string =
                List.foldBack (fun elem acc -> string(elem) + string(acc) ) cl ""

let implodeRev (cl:char list) : string =
                List.fold (fun elem acc -> string(acc) + string(elem) ) "" cl

// Exercise 12
let toUpper (s:string) = implode (List.map System.Char.ToUpper (explode s))

let toUpper1 = explode >> List.map System.Char.ToUpper >> implode

let toUpper2 (s:string) = explode s |> (implode << List.map System.Char.ToUpper)

// Exercise 13
let palindrome (s:string) = (explode s |> implodeRev |> toUpper) = toUpper s

// Exercise 14
let rec ack (m, n) =
    match (m, n) with
    | (m, n) when m < 0 || n < 0 -> failwith "The Ackermann function
```

```
                                    is defined for non negative numbers only."
    | (m, n) when m = 0 -> n + 1
    | (m, n) when n = 0 -> ack (m - 1, 1)
    | (m, n) -> ack(m - 1, ack (m, n - 1))

// Exercise 15
let time f =
    let start = System.DateTime.Now in
    let res = f () in
    let finish = System.DateTime.Now in
    (res, finish - start)

let timeArg1 f a = time(fun () -> f(a))
```

G.3.3 HandIn 3

G.3.4 HandIn 4 & 5

G.4 F# Handins - Niklas Hansen

G.4.1 HandIn 1

G.4.2 HandIn 2

G.4.3 HandIn 3

G.4.4 HandIn 4 & 5

G.5 F# Handins - Ulrik Flænø Damm

G.5.1 HandIn 1

G.5.2 HandIn 2

G.5.3 HandIn 3

G.5.4 HandIn 4 & 5