Gjøvik University College

Software Engineering

Group Project

EpisodeGuide

Victor Rudolfsson - 120912
Tommy B. Ingdal - 120913
Halvor Thoresen - 120915

March 21, 2014

Contents

1 Project Plan

1.1 Goals and frameworks

1.1.1 Background

FrostByte is a company specializing in informative web solutions for everything related to TV-shows. The company has previously had a decent product which covered their basic needs, but has recently been met with increasing competition. Because of this, they've decided to further develop their product and give their existing and new users a completely new product with increased functionality which they hope will give them a competitive edge.

1.1.2 Goals

1.1.2.1 Expected results It's expected that the finished product should be able to:
- register new users
- allow users to stay up-to-date with new and old TV-shows
- suggest new TV-shows based on the users preferences
- distribute e-mail to all users when there are significant news (news letters)
- export TV-schedule to an existing calendar (iOS, Android, WP8)
- offer advertising space to other companies
- allow users to pay for additional functionality

1.1.2.2 Expected impact By developing this system, the company expects the following effects:
- Increased publicity
- By utilizing new technology and functionality, become leading within their niched market
- Increased profits
- Satisfied customers

1.1.3 Frameworks

The project is executed and delivered within the provided framework.

1.1.3.1 Legal framework

The finished system may not violate national or international law. The Data Protection Directive is very relevant here.

1.1.3.2 Technological frameworks

The system should have gone through a thorough security audit before being handed over, to ensure the safety of users' information.
The platform should also be easily managed and maintained.

1.1.3.3 Budget and time frame

FrostByte has stated they want the system up and running within the next 10 months. Furthermore, the project has a budget of 3.5 million NOK.

1.2 Scope

1.2.1 Definitions and limits

FrostByte is a company that currently has a large pool of information about old and current TV-shows, and has for a long time had a web interface to present this information to its users.
As a result of increased competition from other actors on the market, and a strong need for better functionality, a new system titled EpisodeGuide will be developed.

The project will eventually result in a robust system allowing users to register accounts, which will provide them with functionality exclusive to registered users. It'll also be possible to provide users with premium functionality for a small monthly fee.
Administrating the system and distributing new information to the users should

be made simple and smooth, by utilizing new and innovative technology and services.

FrostByte already has a platform / web-interface built on an old but working code base, and they'd like to see some of the old functionality migrated to the new system.
As branches of the platform, mobile applications should be developed to allow for communication with the platforms API. The mobile applications should be released primarily for iOS, Android and Windows Phone.

From a security perspective, because sensitive data such as credit card information and personal user data may be stored, it's important that this data is treated, transmitted and stored in a secure manner. Because of this, FrostByte has entered a partnership with MasterCard/VISA for secure payment solutions.

1.3 Project organization

1.3.1 Resources and responsibilities

The development-company is rather small, with only 7 employees. No previous projects require maintenance, and thus 100% of the available resources are allocated to develop the product for FrostByte. Part of the tasks differ significantly from one another, so to make development as effective as possible over a small group of employees, the workload is split into tasks and distributed over the team.

The development team is split into the following groups:
One project owner / employer
One SCRUM master
Two developers working on backend
Two developers working on solutions for handheld devices
One developer working on design

All employees are experienced and competent within these fields, and workload can thus be delegated rather flexibly if required.

1.3.2 Miscellaneous roles and employment

The development team does not have enough resources to perform thorough testing of the system, and thus an external user group will be established which will be given access to the system whilst it's still under development. This will provide the development team with enough useful feedback and a good overview of bugs and missing features in the platform.
This user group will consist of volunteering invidiuals who've shown an interest in the project through various communication channels. The group will be as diverse as possible, with people of different background and expertise (or lack thereof), which should lead to a large variety of different perspectives and thus, more varied feedback.

Security is an important focus for FrostByte, and an external security audit-team will therefore be brought in to make sure all development is done with security as part of the development life cycle. This will also help improving and maintaining the trust-relation between the company and its users.

1.3.3 Choice of development methodology

1.3.3.1 Daily and weekly SCRUM meetings Every day, the developers will be given an overview of the goals for the day. Problems and challenges can be discussed and space will be given to make adjustments if needed.

Every second week, at the end of each sprint, the developers should engage in a combined sprint review and sprint retrospective meeting, where they shall present a short overview of which tasks have been completed and what problems and challenges have arisen since the beginning of the last sprint.

Because these meetings are common to all developers, time will be allocated at the end of the meeting to allow for a short discussion of the potential issues and challenges.

1.3.3.2 Meetings with product owner Every other week, a meeting between the product owner and the development team will be held to ensure an active dialog. During this meeting it will be possible to suggest adjustments and changes of the product, and for the product owner to be given a brief report of the current development state.

1.4 Planning, monitoring and reporting

1.4.1 Discussion of overall development method

The development company consists of six skilled people with broad expertise within web-technology and development. The project consists of a variety of different development methods and the complexity is rather high; thus it is expected that development will take around 10 months to reach completion.
With such a system, it's highly plausible the product owner change their mind regularly as a result of feedback from users. The project builds on an existing system and existing users, and thus is not considered a high risk factor.

SCRUM has been chosen as the development methodology, mainly because a sprint time of 14 days will help in providing incremental revisions that can be tested by both product owner and user testing group. This is especially important because of the risk of changes being made to the project. The project requirements may be modified according to product owner's and users' feedback, which is why it's important to continuously review previously completed functionality to improve or make changes to the code. The system is developed by a relatively small team, something which requires each employee to complete their respective tasks within the given time frame. With SCRUM sprints it will be simpler to measure success and workload to make sure a consistent and stable workflow is maintained.

1.5 Risk analysis

To ease the risk assessment, a simplified scale for probability and impact has been established. Figure 1 shows the different levels of impact and probability.

Probability Impact

4 Highly likely 4 Catastrophical

3 Very likely 3 Critical

2 Likely 2 Danger zone

1 Not likely 1 Minimal Figur 1 - , Sannsynlighet og konsekvens tabell

The level of risk is calculated by multiplying these factors, probability and impact, with one another. After calculating the risk, a number between 1-16 indicates the risk level, where 1 is the lowest possible risk and 16 is the highest possible risk. To more easily see what is an actual risk, we have defined an acceptable risk level of 4. Everything with a risk level below 4 is considered acceptable risk. Fig. 2 gives an overview of the possible risk values given by the product of probability and impact. Green indicates acceptable risk level, whereas red indicates unacceptable level of risk.

Lite farlig Farlig Kritisk Katastrofalt Lite sannsynlig 1 2 3 4 Sannsynlig 2 4 6 8 Meget sannsynlig 3 6 9 12 Svært Sannsynlig 4 8 12 16 Fig. 2 - Overview of risk levels

Identify and Analyze project risks Description Probability Impact Risk level Loss of source code 1 4 4 Project exceeds budget 2 3 6 Sudden change of requirement specification 3 2 6 Project exceeds timeframe 2 2 4 Project cancellation 1 4 4 Loss of communication between dev. team and users 4 2 8 Fig. 3 - Risk table

1.5.1 Plan for managing the most critical risks

1.5.1.1 Project exceeds budget Risk can be reduced by performing a monthly review of expenses, and by establishing a financial budget for the development timeframe.

1.5.1.2 Sudden modification of requirement specification By maintaining a good and active dialogue between product owner and development team, preferably once every other week, this risk can be reduced. Additionally, a thorough meeting will be held between product owner and development team led by the scrum master before the first sprint starts, to make sure all requirements and requests have been presented and considered..

1.5.1.3 Loss of communication between development team and users

This risk can be reduced by actively utilizing the user group. As soon as new functionality is complete it should be published for testing by the user group, as to allow them to continuously provide constructive feedback on the latest implemented features.

1.6 Implementation

1.6.1 Decision deadlines

01. mar - Will the project be developed?
15. mai - Use of technology

1.6.2 Milestones

25. feb - Project plan finalized.
21. mar - Requirement specification finalized.
25. apr - Design-plan finalized.
15. may - Complete project report done.
15. dec - Evaluation report done.
15. dec - System fully developed.

Timeframe

2 Requirement Specification

2.1 Use Case Diagram

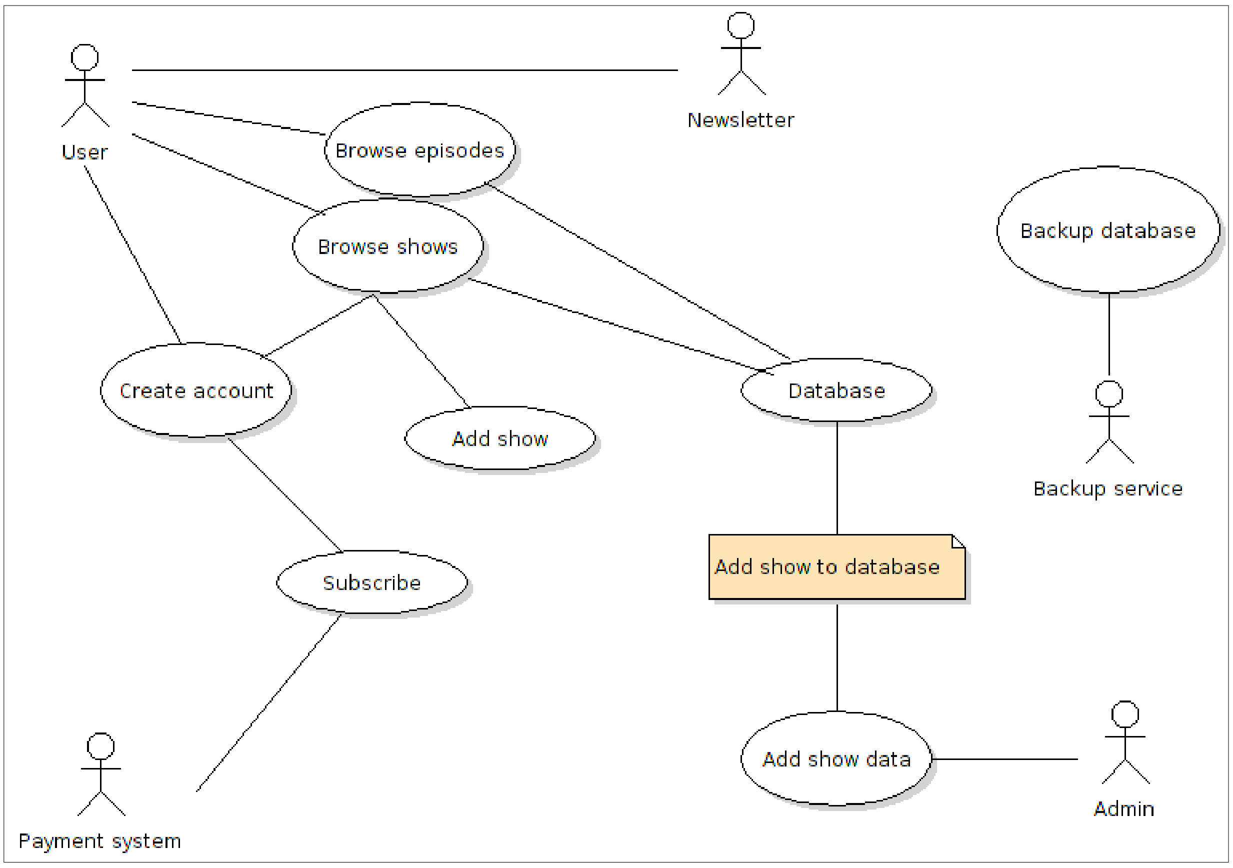

Figure 1: Use Case Diagram

2.1.1 Comments on Use Case Diagram

In the use case diagram we have five entities which make up the flow of the application. The user, and also the main entity, is the one that interacts with most of the parts of the application (exluding: backup system and administration). The administrator is the entity that adds data to the database (ie. new shows and/or episodes) or sends out newsletter.

In our model we have one external entity; the payment system. This system allows the users to upgrade their account and pay their subscription fee directly via the web application. The application communicates with the payment backend (ie. Visa/Mastercard) over a secure channel.
The backup system, as of today, only have one job; on regular intervals, make a backup of all the user accounts and associated data.

2.2 High level Use Case Descriptions

Use Case	Add show data
Entity	Admin
Goal	Keep the database up to date
Description	The administrator make sure the database is up to date with all the newest show and episode data.

Use Case	Backup
Entity	Backup, Admin
Goal	Make a copy of the database and keep the database safe.
Description	The database will make a backup on regular intervals, but the administrator can initiate a backup process whenever he wants.

Use Case	Newsletter
Entity	Admin
Goal	Sends newsletters to user.
Description	The administrator can, through the administration page, send newsletter/emails to his users.

Use Case	Subscribe
Entity	User
Goal	Subscribe to a member-only service.
Description	After a user has created an account, it is possible to subscribe to member-only services. The user will be prompted to pay a subscription fee which will be handled by Visa/Mastercard. All the data transfered between the client/server is encrypted (uses SSL/TLS).

Use Case	Browse
Entity	User
Goal	View information about tv shows.
Description	A registered user, as well as a non-registerered user, can use the application to find information about shows and episodes by searching the database.

3 Detailed use case

Name: Subscribe	**Actors:** User — Payment system
Pre-condition: User has created account. Connection with payment system is up.	
Post-condition: Payment system will either accept or decline the User information input, and then send the appropriate response to database.	
Trigger: User wants to upgrade the account to a paid subscription.	
Event flow: 1. The user requests the subscription page. 2. User fills out and submits the necessary payment information. 3. The information is encrypted and sent over a secure connection to the payment system. 4. Payment system recieves and decrypts the information. 5. Payment system executes the necesarry verifications and confirmations. 6. The appropriate response is sent back to the user	
Event variation: 1. User has not created an account: (a) The requested subscription page will not be displayed. (b) The user will be sent to the create account page. 2. User submits invalid information that does not fit the input template: (a) The information will not be sent to the payment system. (b) The subscription page will be reloaded and an error message will be displayed. 3. The payment system response is positive and the transaction went without problems: (a) The user's status is updated. 4. The payment system response is negative and the transaction did not go through: (a) An error message will be displayed and the process will be terminated. (b) The subscription page will be reloaded for the user.	

Name: Backup database	**Actors:** Backup service — Admin
Pre-condition: There is an already existing database.	
Post-condition: A full backup of the database is created and stored separatly from the original	
Trigger: The backup serive requests a backup on regular intervals — The admin manually requests a backup.	
Event flow: 1. Either the admin or the backup service requests to run a backup. 2. The backup service checks if any changes had been made since last backup. 3. The backup service checks if there is enough disk space for the backup to be written to disk. 4. The backup service creates an exact copy of current database and stores to the appropriate disk with the current timestamp.	
Event variation: 1. No existing database can be found: (a) The process will be terminated. (b) An error message will be written to the log. 2. No changes has been made: (a) The process will be terminated. 3. Not enough disk space to store backup: (a) The process will be terminated. (b) An error message will be written to the log. 4. Not enough disk space to store backup: (a) The process will be terminated. (b) An error message will be written to the log. (c) An email will be sent to the the appropriate administrator. 5. Backup is interrupted mid-process: (a) The process will be terminated. (b) The files already written to disk will be removed. (c) An error message will be written to the log.	

3.1 Operational Requirements

Usability

- The application should be easy to use, with a smooth learning curve.
- The administrators should not need to re-learn everything. Since we are building this project on top of some existing code, the administrators should be able to do things the way they are used to.
- The layout should be clean and minimalistic.

Performance

- The user should be able to quickly get data from the database (ie. minimal latency).
- The system should be able to handle 20 000 concurrent users with minimal delay.
- When a user searches for something in the database, the delay before getting the results should be no more than 3 seconds.

Reliability

- If a bug is discovered it should be patched as soon as possible. This should not take more than 48 hours from the time of notification.

Environment

- The main programming language will be PHP since the core developers have experience with PHP from other projects.
- This project will also take advantage of a few open source libraries to accomplish the performance and availability goals.

Security

- Personal information should be kept safe and encrypted in the database. No person should be able to access and read this information.
- When a user pays the subscription fee, the data between the client and server should be transfered over a secure channel (eg. SSL/TLS)
- A users payment information (ie. creditcard numbers etc.) should not be saved in the database.

Availability

- The application should have an uptime of at least 99.9% (ie. 24 hours/day, every day of the year)

3.2 Conceptual Class Diagram

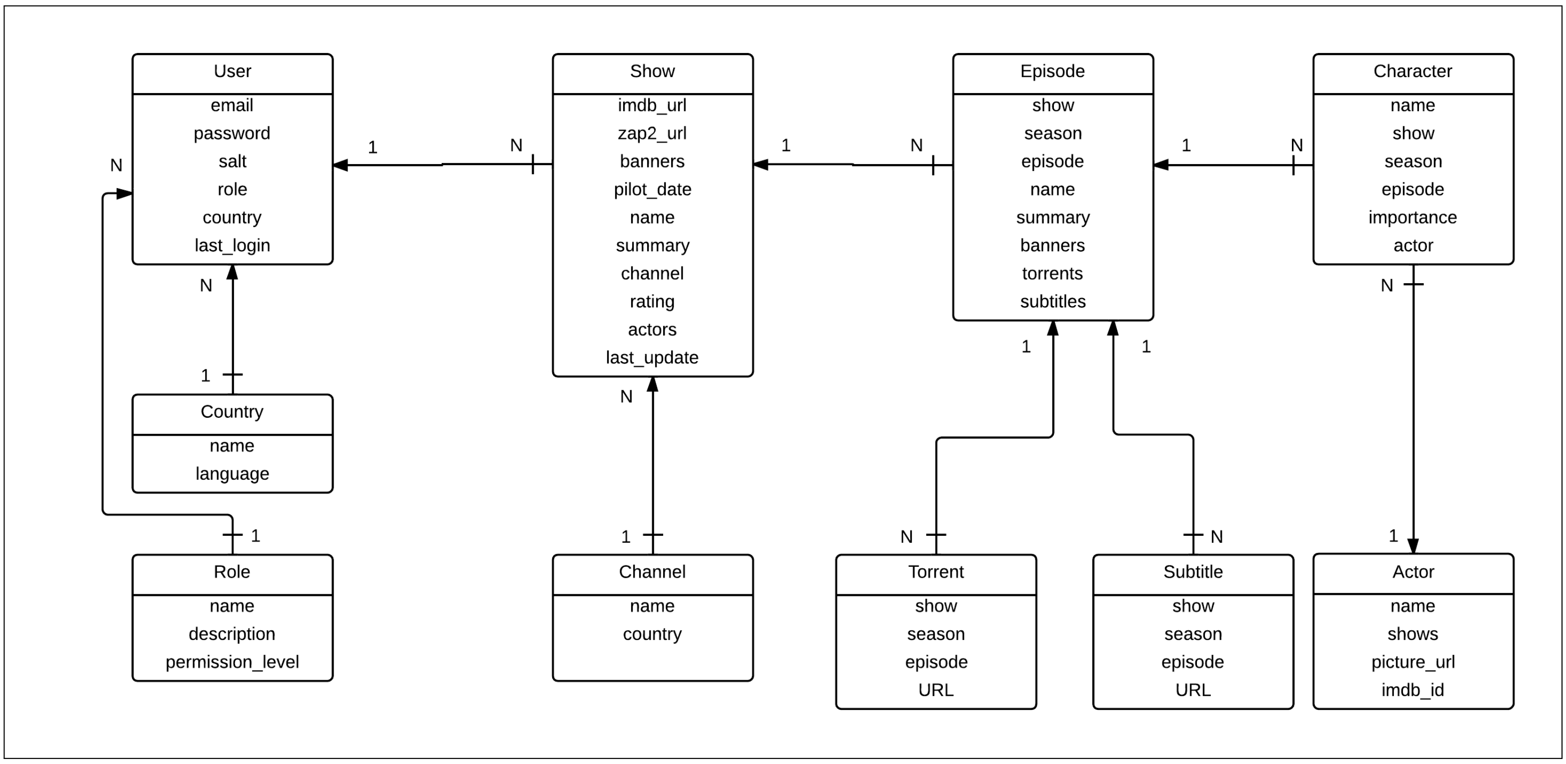

Figure 2: Conceptual Class Diagram

The conceptual class diagram illustrated above shows the most vital objects for the application and their properties and relationship to one another. Abstract classes, 3rd party library classes or classes utilized by others (such as the database handler class) are intentionally omitted here, but it serves to provide an overview of the relationships between objects within the application.

The most important objects here are *User* and *Show*. The main focus of the system should be on *Show*, because this is the object users will interact the most with. Even without creating a user, it should be possible to browse through the list of shows and their episodes.

However, for more tailored functionality, it's of utmost importance that a *User* is able to log in after being created, and that functionality is provided to allow the user to specify what shows he or she watches.

A *User* may also be an administrator, or any other *Role* with special permissions one may want to assign a user, and it should be possible to create and assign new *Roles* at a later time.

Next, when a *Show* has been selected, it's highly important that we're able to present some information about this *Show.* Information that may be of interest, such as what *Channel* it was originally broadcast on, information about each episode, character, or actor, needs to be retrievable from each *Show* object.

It's not unlikely that a *User* would like to filter *Shows* by what channel produced or first broadcast it, and as such, it should be possible to retrieve a list of *Shows* from a *Channel* object.

If a *User* has found a *Show* he or she may want to watch, it's very likely that he or she will begin to look for a way to download or stream it. It's therefore desired that lists of *Torrents* and *Subtitles* can be presented for any given *Episode* of any given *Show.*

Furthermore, when a *User* views an *Episode*, he or she may be interested in knowing what *Characters* exist and maybe more importantly, what *Actors* play in this *Episode*, and therefore it's important that a list of *Characters* (sorted by their *importance*) and what *Actors* plays them can be retrieved and presented to the *User.*