PKSU - Props 2.0

Louise Knudsen, Helena Bach, David Pedersen

25. marts 2014

1 Problem definition

The Royal Danish Theatre's props department uses a locally installed database-system to search through all of the props and productions and their corresponding set-up- and run-lists. The system was developed more than 20 years ago by their own Claus Nepper Fakkenberg. Claus alone holds the responsibility for maintaining the system, as he is the only one who understands the source-code, which will soon be a problem as he is now retiring.
Due to the age of the system, some functionalities are no longer needed, and there are new requirements that are not implemented, most importantly a need to use the system outside of the office.

The problem can be described with the following points:

- The current database system was developed in 1989, and lacks a lot of wanted functionalities.
- The developer, Claus, is the only one who can modify and support the system.
- Claus is retiring.

To analyze the problems defined above and develop a system definition we use the FACTOR Criterion.

1.1 The FACTOR Criterion

Functionality: Keeping track of the props and performances, and help in running these. As well as support the administration.

Application domain: The Royal Danish Theatre.

Conditions: The system should be usable by multiple user at a time, wherever there are access to the internet.

Technology: The system will be developed on standard laptops, and should be usable on standard PC's and tablets and supported by every OS.

Object: Props, pictures and employees in the props department.

Responsibility: Searching and administrative tool.

1.2 System definition

A web-based database-system of The Royal Danish Theatre props department. The system should primarily be a searching tool used to prepare and run performances and search for props in current and previous productions by the assistant stage managers, and secondly be used for budget monitoring, search for supplier information as well as keeping track of all performance information, by the chairman of the props department. The system should be usable for people with greatly variable computer experience.

1.3 Requirements

We have had multiple meetings with the different people who have daily contact with the system to form a idea of the needs and requirements, but only April 9th these will be formalised more specifically, when we kick off the project in collaboration with the client.

1.4 Constraints

The constraints of our project include the security aspect, which the head of the IT-department, Martin Thaarup Larsen, makes sure to handle in a way that makes it possible for our system to be part of the existing login-system. Martin has the responsibility for the media database as well - most likely made accessible to us through a "mirror database".

1.5 Solution domain

The solution domain can be defined as being the props department as well as Cumulus, their media database, which our application will be interfacing with.

1.6 Deliverables

- PHP code
- MySQL database
- Instructions in the form of meetings in addition to papers explaining the system.
- Probably also some support during the initial start up phase.

2 Initial Software Project Management Plan

2.1 Overview of the project

To give a initial overview of the project in terms of planning and managing we have composed the following points.

2.1.1 Project summary

Work product:
- MySQL database
- PHP code for the web-interface.
- Assignments describing the work process

Schedule:
On April 9th we are scheduled to meet with Martin and Mikkel from the theatre, to finalize the requirements elicitation and sign the project agreement. We will thereafter be able to put together a more detailed time-schedule.
On June 23rd we have our final deadline, and will hopefully deliver the final product to the theatre.

Participants:
Developers: David Pedersen, Helena Bach, Louise Knudsen
Project managers: David Pedersen, Helena Bach, Louise Knudsen
Client: The Royal Danish Theatre.

Tasks:
The developers will have a common responsibility to participate in every aspect of the system-development. Of course their will be a natural division of the task in correspondence to each developers skill- and interest-level as described in the skill matrix, which is found in section 5.

2.1.2 Evolution of the plan

All changes to the project management plan must be agreed to by all participants before they are implemented. All changes should be documented in the, to the developers, shared log, and in time become a part of the final SPMP.

2.2 References

All artifacts will conform to the theatres standards.

2.3 Definitions

The Royal Danish Theatre uses a lot of different terms in their work, and some of these should also be used in the future database system. The terms we have been presented to so far are listed with a short description below:
Every production has a unique 8-digit number, containing of 4 random digits followed by a dash and premiere-year for this particular production. (There can be multiple productions of, for instance the same play, which then have different premiere-years)
If a production is *SKILT*, it means that it has been discarded and the props are back in the storage room.
If a production is *I REPERTOIRE*, it is being played in the current season and the props are in use.
If the production is *I CONTAINER* or similar, the production is not yet *SKILT* nor is it currently being played, and the props are therefore occupied but not in use.

2.4 Project Organization

2.4.1 External interfaces

The props database and appertaining web-interface will be managed and developed by David Pedersen, Helena Bach and Louise Knudsen.
The photographs of the props and productions, along with the login system and server connection will be handled by Martin.
All managers will be in continuous contact with the chairman of the props department, Mikkel Rasmus Theut, as well as Martin.

2.5 Managerial process plans

2.5.1 Start-up plan

The total developments time is estimated to be 15 week, counting from 13/03/2014, where first meeting with the client were held.
All necessary hardware are available, as well are the software since MySQL and PHP both are open-source.

2.5.2 Work plan

Week 1-5:

- Initial meeting with Mikkel.
- More technically oriented meeting with Martin where Mikkel were also present.
- Meeting with head of the furniture subdivision. Prepare the requirements elicitation and the Project Agreement.

- Meeting with Martin and Mikkel - signing of the requirements elicitation and the Project Agreement, and project kick-off!

Week 6-7:

- Focus on making the SQL-script.

Week 8-11:

- Finish SQL.
- Start design of the web-application.
- Begin PHP-coding and testing.

Week 12-15:

- Finish PHP-coding and testing.
- Final documentation.

2.5.3 Risk management plan

The risk factors and the tracking mechanisms are as follows.

As one of the goals of the project is to achieve more functionality than the existing system, there should in the testing-process be compared results between the two.

There should in the designing-process be a great amount of communication with the client to ensure an as user-friendly interface as possible, as the system mainly will be used by people with limited computer-experience.

There is a slim chance of hardware failure, in which case each developer is responsible for there own computer, and the client holds the responsibility for the server.

Each developer is responsible for continuous testing of their assigned subsystem(s) and jointly cross-testing between these.

2.6 Techinical process plans

2.7 Supporting process plans

2.7.1 Configuration management plan

Git and GitHub will be used in all aspects of the project to ensure version control.

2.8 Additional plans

2.8.1 Presentation

In connection with the delivery of the final product, the system and its functionalities will be a presented to the chairman of the props department, Mikkel.
As one of the purposes of developing the new system, is to make it possible for the head of the IT-department, Martin, to keep the system up to date himself, we will in addition to the presentation, be going through code with him.

2.8.2 Support

During the initialisation phase support will be offered free of charge.

3 Initial software architecture

3.1 Design goals

First of we would like to note that this architecture is in a very early phase and will most likely change as we move forward in the process.
Given the circumstances that this application will be built and deployed under (namely that we might not be the ones who end up maintaining it) it is important for the design of the system to be well documented and well structured.

We have the following overall design goals for the system:

- The system should be factored into meaning full subsystems each with a well defined responsibility.
- The system should make use of both unit and integration level tests as this will allow for easier refactoring and maintenance of the system. This is especially important since we might not be the ones who will be responsible for maintaining the system.

3.2 Initial system design

3.2.1 Subsystem decomposition

Given that our application will most likely be a website backed by a relation database there will be three primary layers to it:

1. The database.
2. The domain model.
3. The web front end.

Each layer will have their own responsibility.

3.2.2 The database

Here all data will be stored. It will most likely be some sort of relation database.

3.2.3 The domain model

In web terms this will be the back end of the website. This is where domain concepts such as props and performances are given attributes and methods that given them behavior and allow them to interact with each other and the user. This is also where important features of the application will be modeled such as how a search can be performed on the database.

3.2.4 The web front end

This layer will be a graphical representation of the data and relationships between the different data points built by the back end. This is also the part of application that the end user will interact with.
Since we are building a web application this layer will be built using the standard set of web technologies, HTML, CSS, and JavaScript.
Each of these three subsystems will each be composed of their own smaller subsystems that each have a well defined purpose and responsibility.

3.2.5 MVC

For decomposing the back and front end and the relationships between them we have chosen to use the MVC design pattern. This lets us split the system up into small manageable parts with a clear purpose and responsibility. This will make the code more maintainable and more suitable to change. Depending on the domain your application lives in the separation and responsibilities of the models, controllers, and views might be slightly different. Here are some initial thoughts about what each subsystem will be responsible for.

3.2.6 Model

The model layer will contain code that models the entities in the domain. For our application there will most likely be a props models, a performance/play model and others.
This is also the layer that is responsible for communicating with the database and managing the persistence of the objects.

3.2.7 View

The view layer will contain the graphical representation of the model layer that the user can view and interact with.

3.2.8 Controller

The controller layer is responsible to managing the interactions that the user makes with the model layer. A user will be performing actions like navigating between pages, submitting forms, searching for props, and so on. It will be the controllers responsibility to map these actions into the model layer and construct new views for the user. It essentially acts as glue code between the user and the domain/model layer.

3.2.9 Dependencies

The MVC design pattern is very useful for managing and limiting dependencies between the different subsystems. An important part of MVC is that the model layer and view layer should be as little dependent on each other as possible. This will make it easier to change each part independently.
It is however not possible to make the controllers free of dependencies since they sit in between the views and the model. Therefor they are naturally dependent on both the views and the model. However MVC helps with this by providing each subsystem with single responsibility thus making each of them smaller and more well defined.
The system will also have to interact with other external systems. This will primarily be Cumulus for linking a prop in our system with pictures in the media database.

3.3 Initial subsystem components

3.3.1 Views

At this early point it is hard to say which views the application will contain but there will likely be views related to the following areas:

- View that lists props. This could be a listing of all props in the database but more likely a list filtered based on some search criteria.
- View for doing a search. This view would contain a form with fields for filtering through the props.
- View the shows a specific prop. There will most likely be a view that shows the information available about a single prop and its relationships.
- Administrative views. These would be views that let administrators perform usual operations on the data (create new, update existing, and delete existing).

3.3.2 Models

Given the views listed above our model layer will most likely include:

- User model. If the application ends up containing authentication and users with different roles then there will also be a model responsible for that behavior.

- Prop model. Given that this application is about props there will most certainly be a prop model that describes the properties and behavior of props.
- Performance model. If props need to be related to performances then there will also be a performance model.

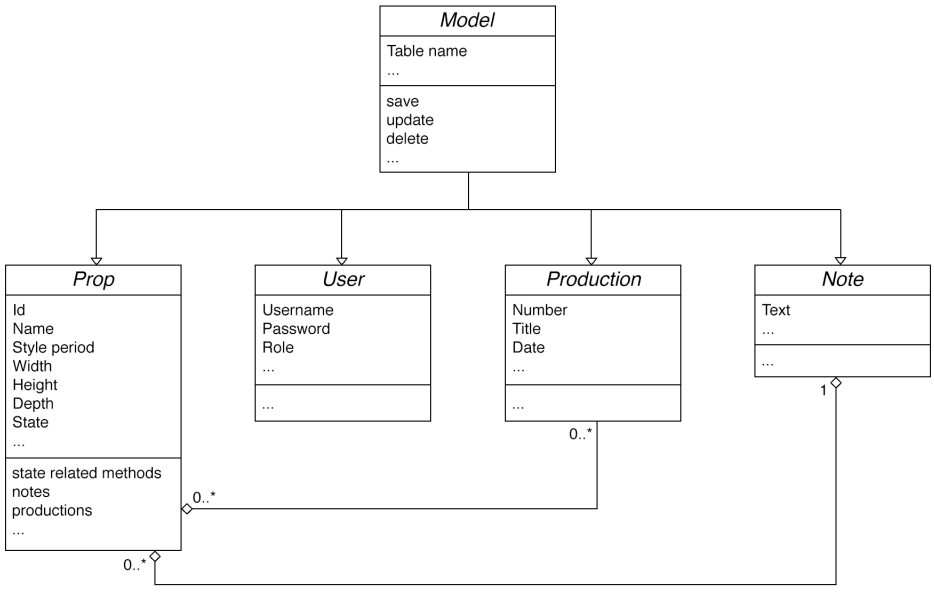
Given that it should be possible to persist the model objects in the database there will need to a subsystem that handles that. It also makes sense for this system to be responsible for building model objects from the data in the database. Exactly how this will be modeled is too early to say but given that this is behavior is shared between all the models it might be modeled with inheritance or object composition.

3.3.3 Controllers

Specifically which controllers the system will include depend on which views we want and how they are grouped together. However since this application is primarily about searching there will most likely be a controller that is responsible for receiving a search query, then translating that to a query the model layer can perform and then rendering a view containing the results to the user.

3.4 Diagrams

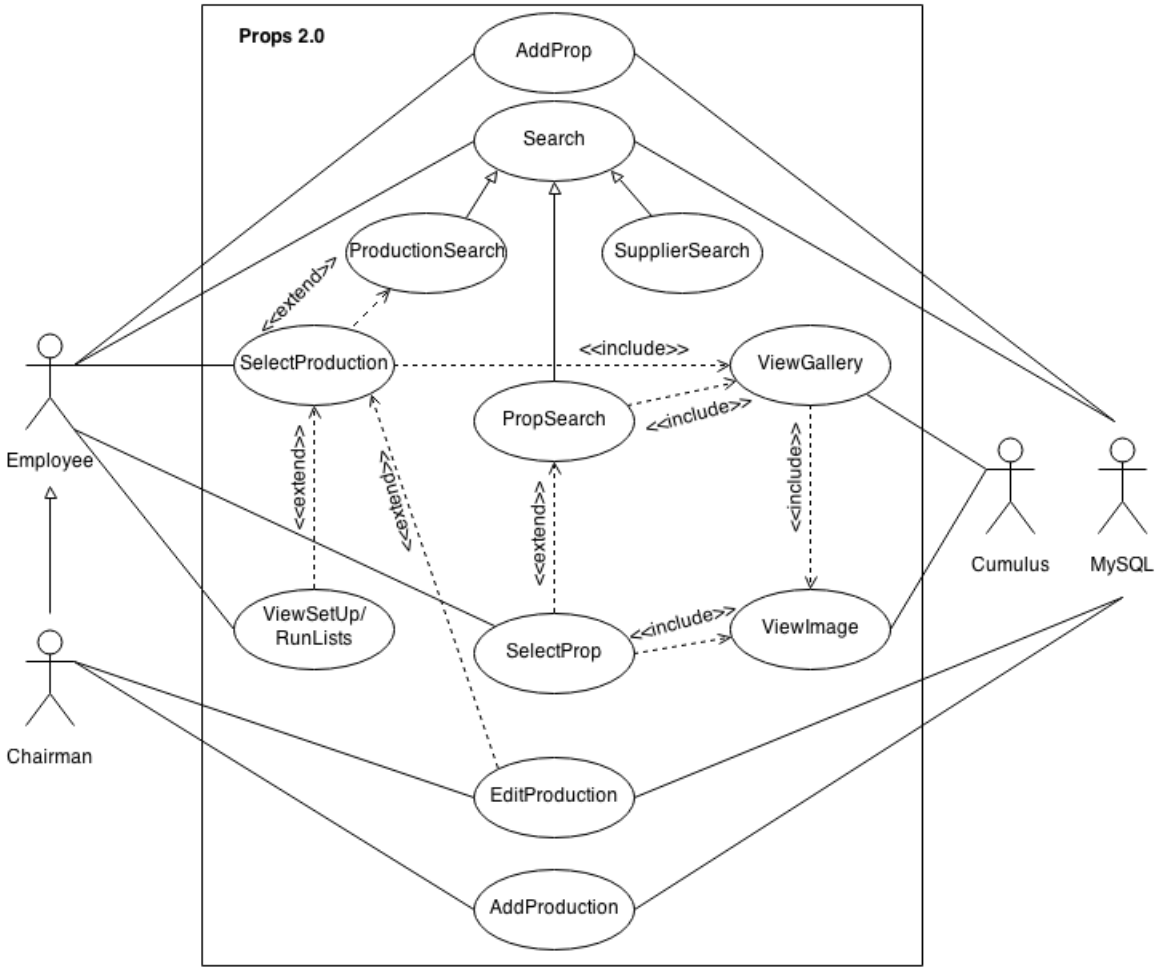
Here is a class diagram of the model layer as we imagine it might look.



4 Project Agreement definition

As mentioned in previous sections our final requirement elicitation and project agreement will be formalized on April 9th, at our meeting with Mikkel and Martin. To prepare for the meeting and make an outline for this final definition we have made the following UML use case diagram (See Appendix X for use cases) and functional/nonfuntional requirements overview.

4.1 UML use case diagram



4.2 Nonfunctional Requirements

- The system should be web-based and usable on standard PC's and tablets, and supported by every OS.
- The webapplication should be user friendly and easy to navigate. The most commonly used functionalities, namely the run- and set-up-lists should not be more than 3 clicks

away.

- It should be possible for Martin to maintain and support, and possibly expand the system after product-delivery. Therefore the system will be developed using MySQL and PHP, which is familiar to the XX.
- The system should be able to communicate with the existing media-database Cumulus.
- noget med firewall login og halløj.
- The interface should be in Danish.

4.3 Functional Requirements

- It should be possible to do cross search for props, productions and suppliers.
- It should be possible to add, edit and delete props, productions and suppliers.
- noget med Kongeplanen
- It should be possible to print run- and set-up-lists.
- noget med drag/drop props til run- and set-up-lists.

5 Internal project establishment

To get an overview of the participants skill set and learn who could be responsible for which subsystems we made a skill matrix.

- • = Primary skill ◯ = secondary skill △ = interest.

Tasks/Participant	*David*	*Helena*	*Louise*
Database Design	◯	•	•
Web Programming	•	△	△
Testing	•△		
UI Design	△	△	•△
Project Management		•	◯
Object oriented programming	•△	◯△	◯

When the subsystems get defined, we can now use this matrix to delegate the tasks and responsibilities in the most suitable and efficient way.

6 Appendix A

6.1 Use cases

Use case name	`AddProp`
Participating actors	Initiated by `Employee` Communicates with `MySQL`
Flow of events	1. The `Employee` activates the "Add prop"function. 2. `Props 2.0` responds by presenting a form to the `Employee`. 3. The `Employee` fills out the form with the relevant prop-info, and submits it. 4. `Props 2.0` receives the form, and the MySQL database gets updated. `Props 2.0` displays a confirmation of the update.
Entry condition	The `Employee` must be logged into the Theatre's Wi-Fi
Exit condition	The `Employee` has received a confirmation OR An explanation indicating why the transaction could not be processed.
Quality requirements	Not yet defined

Use case name	`AddProduction`
Participating actors	Initiated by `Chairman` Communicates with `MySQL`
Flow of events	1. The `Chairman` activates the "Add production"function. 2. `Props 2.0` responds by presenting a form to the `Chairman`. 3. The `Chairman` fills out the form with the relevant production-info, and submits it. 4. `Props 2.0` receives the form, and the MySQL database gets updated. `Props 2.0` displays a confirmation of the update.
Entry condition	The `Chairman` must be logged into the Theatre's Wi-Fi
Exit condition	The `Chairman` has received a confirmation OR An explanation indicating why the transaction could not be processed.
Quality requirements	Not yet defined

Use case name	`Search`
Participating actors	Initiated by `Employee` Communicates with `MySQL`
Flow of events	1. The `Employee` activates the "Search"function. 2. `Props 2.0` responds by presenting a form to the `Employee`. 3. The `Employee` fills out the form with the search criteria, and submits it. 4. `Props 2.0` receives the form, the MySQL database returns the matching data and `Props 2.0` displays it
Entry condition	The `Employee` must be logged into the Theatre's Wi-Fi
Exit condition	The `Employee` has received a list of matching data OR A "No match"message
Quality requirements	Not yet defined

We have not yet fully specified the search criteria with client. We know that separate search-functions for productions, props and suppliers are wanted, but until we have the detailed criteria, we will not include these as specific use cases.

Use case name	`SelectProp`
Participating actors	Initiated by `Employee`
Flow of events	1. The `Employee` selects a prop from the results of a premade search 2. `Props 2.0` responds by presenting a more detailed description of the prop to the `Employee`.
Entry condition	This use case extends the `PropSearch` use case
Exit condition	The `Employee` is looking at the prop description
Quality requirements	This use case includes the `ViewImage` use case

Use case name	`SelectProduction`
Participating actors	Initiated by `Employee`
Flow of events	1. The `Employee` selects a production from the results of a premade search 2. `Props 2.0` responds by presenting a more detailed description of the production to the `Employee`.
Entry condition	This use case extends the `ProductionSearch` use case.
Exit condition	The `Employee` is looking at the production information.
Quality requirements	This use case includes the `ViewGallery` use case.

Use case name	`ViewSetUp/RunLists`
Participating actors	Initiated by `Employee`
Flow of events	1. The `Employee` selects the list from the results of a production search 2. `Props 2.0` responds by presenting the list to the `Employee.`
Entry condition	This use case extends the `SelectProduction` use case
Exit condition	The `Employee` is looking at the list
Quality requirements	Not yet defined

Use case name	`EditProduction`
Participating actors	Initiated by `Chairman` Communitates with `MySQL`
Flow of events	1. The `Chairman` activates the "Edit production"function 2. `Props 2.0` responds by presenting a form to the `Chairman` 3. The `Chairman` fills out the form and submit the changes OR The `Chairman` selects the "Delete"option 4. `Props 2.0` receives the form, and the MySQL database gets updated. `Props 2.0` displays a update- OR delete-comfirmation
Entry condition	This use case extends the `SelectProduction` use case
Exit condition	The `Chairman` has received a confirmation OR An explanation indicating why the transaction could not be processed.
Quality requirements	Not yet defined

Use case name	`ViewGallery`
Participating actors	Initiated by `Employee` Communicates with `Cumulus`
Flow of events	1. The `SelectProduction` or `PropSearch` use case gets evoked 2. `Props 2.0` requests `Cumulus` for the image data 3.`Cumulus` returns the matching images 4. `Props 2.0` responds by presenting the images as a gallery
Entry condition	The `Employee` must initiate the `SelectProduction` or `PropSearch` use case to initiate this use case
Exit condition	The `Employee` is looking at the gallery.
Quality requirements	This use case includes the `ViewImage` use case.

Use case name	`ViewImage`
Participating actors	Initiated by `Employee` Communicates with `Cumulus`
Flow of events	1. The `ViewGallery` or `SelectProp` use case gets evoked 2. `Props 2.0` requests `Cumulus` for the image 3. `Cumulus` returns the matching image 4. `Props 2.0` responds by presenting the image to the `Employee`
Entry condition	The `Employee` must initiate the `ViewGallery` or `SelectProp` use case to initiate this use case
Exit condition	The `Employee` is looking at the image.
Quality requirements	Not yet defined