Krusty-JS

Thesis

Project: Krusty-JS
Student: David Ammeraal
Studentnr: 1520387
Email: david.ammeraal@student.hu.nl
Supervisor HU: Michiel Borkent
Company: Noterik B.V.
Supervisor Company: Konstantin Radoslavov
Version: 0.8
Date: 3-06-2012

Preface

At the end of 2011 it was time for me to finish my studies at the University of Applied Sciences of Utrecht. At the end of the studies a student has to prove to the HU that he is capable and that he can apply all that he has learned during his studies at HU. This document will describe what I have done during my time at Noterik.

During the summer of 2011 I had finished most of my subjects so I set out to find a suitable assignment. Thanks to the help of a friend I was quick in finding a company that possibly had an assignment available for me.

I was introduced to Noterik B.V. (described as Noterik from now on) in November of 2011 through Jechiam Gural, one of the owners of Noterik. I was redirected to Rutger Rozendaal. We organized a meeting and it was not long before we came to a definition of an assignment that I would be able to do at Noterik in the period of January 2012 until June 2012.

The time spend during my time at Noterik was thoroughly enjoyed. It pleases me to know that they took my project seriously, and that they are actually planning to implement the product in their system, and although it was hard work to finish everything on time, it satisfies me knowing that the product I created during that time will actually be used. The product is actually already being used in the LinkedTV project described in the Plan of Approach, a project in which I am actively involved now.

I would like to thank all my colleagues at Noterik for the excellent guidance and help they provided me with, and for allowing me so much freedom in the development of my product. If not for their active involvement in my project, the project would not have been such a success.

I would also like to thank Michiel Borkent for his guidance during my period at Noterik, he was actively involved in the composition of the document.

Management summary

Goals of the project

Noterik currently employs a Flash application called Krusty to present the videos stored in their platform. Flash technology is a degenerating technology, mainly because of lack of support of the standard on IOS devices such as the iPhone and the iPad. Other than that, Adobe has recently announced that it will not provide further updates for the Flash player on mobile devices. Noterik desires to reach a wide spectrum of users, including IOS and mobile device users. So therefore, the functionality had to be transferred.

The goals of the project were to do research on how the functionality contained within the Krusty application could be transferred to a HTML5/JavaScript environment. After this research was done the functionality contained within the Krusty application had to be transferred to this new environment.

Instructing party

Noterik is the instructing party. Noterik is a small company situated at the Prins Hendrikkade 120 in Amsterdam, The Netherlands. Konstantin Radoslavov was the supervisor of the author during the time of the project. Noterik focuses on delivering online video services to mainly non-profit sector companies.

Environment in which the project was done

The project was mainly done at the office of Noterik in Amsterdam. The author was part of the development team of Noterik. The author mainly worked solitary on the assignment, but had to actively communicate with colleagues in order to implement the functionality in a correct way.

Project roles and responsibilities

Student:	David Ammeraal
Supervisor HU:	Michiel Borkent
Company:	Noterik B.V.
Supervisor Company:	Konstantin Radoslavov
Instructing Party:	Noterik

Activities

The project was initiated by making a Plan of Approach. After receiving a reviewed version of a concept of the Plan of Approach, a final version was send to the HU. After this the Research Phase was initiated. It began with making a research plan. The following research question was established:

"How can the functionality encapsulated within the Krusty-JS player be transferred to the HTML5 environment?"

The research question was split up into sub questions. Each sub question enveloped a certain functionality that had to be transferred. After that the research was initiated, each sub question was answered in order to be able to come up for a solution of the main research question.

After the research was done the design of the application was initiated. A functional design was made. The functional design contained the functional and non-functional requirements of the application, from which use-cases were extracted. After the functional design was made, it was time to create a technical design.

The technical design contained descriptions of how the functionality could be implemented by HTML5 and JavaScript technology.

After the functional and technical designs were created it was time to start developing the application. In order to this in a structured way, SCRUM was employed. SCRUM is an iterative and incremental development technique. A product backlog was constructed containing user-, and technical stories extracted from the functional and technical design. Three sprints were defined. Sprints are time boxes with a start time and a fixed end time. Each sprint enveloped an array of stories that had to be developed within the sprint. After each sprint a demo had to be given about the technology that was developed within that sprint.

Results

The research was done successfully. Almost all functionalities could be transferred to the new HTML5 environment. However, because the browsers differently interpret a lot of the HTML5 API, it might be difficult to implement all the functionality in all the browsers.

After completing all sprints, almost all functionality was implemented. However, towards the end of the project, it was decided that the displaying of Events within the application would have to be constructed in a completely different way. Therefore, this functionality did not make it to the latest version of KrustyJS. Also full screen support did not make it into the final version of the application.

Conclusions

HTML5 has proven to be a serious contender for the Flash technology. Most of the functionality was implemented. The functionality that was not implemented will be implemented at a later time. There are, however, some things that a developer should keep in the back of his/hers head when developing an application that uses the HTML5 video element:

-No single video codec works on all browsers
-There is no widely supported streaming platform for HTML5
-HTML5 is very volatile environment that is constantly changing
-No DRM or other security available for copyrighted material.

Looking forward

HTML5 is adopted progressively more every day. Big data providers such as Youtube and Vimeo are currently also in the progress of implementing HTML5 in their play out engines. In order for Noterik to keep up with its competition it is important that following should be done:

-Integrate HTML5 play out in all of its projects that aim to reach an audience which employ a wide arrange of devices.
-Convert the existing Montage Tool to HTML5
-Keep an eye open for a widely adopted HTML5 streaming platform
-Keep up to date concerning the current issues with securing copyrighted material with HTML5

KrustyJS should serve as a start-point for more complex implementations of the HTML5 API.

Table of contents

1. Introduction

This document describes what was done during the project that was performed by the author for his graduation project at Noterik B.V. It will start with a brief management summary. In here a quick summary will be given regarding the entire project.

Following that a brief version of the plan of approach will be described. The Plan of approach contained the planning, requirements and goals of the project.

After this it will be described how the preliminary research was performed. The main research question will be described, and how this question got divided into smaller sub questions. After that it well be described how the research was performed and what the results and conclusions were.

The conclusions of the research served as an entry point to the development of the actual application. The functional and non-functional designs of the system will be shortly described according to the requirements that were defined in the functional design.

Following this the actual realisation of the product will be illustrated. This will be done according to technical examples and diagrams as well as textual specification. The development of the basic requirements that were defined within the Plan of Approach will be described.

After this a summary will be given about the decisions that were made during the time of the project. The reason for every decision will be elaborated.

Following that a summary will be given regarding the conclusion that were drawn during the project, and what proposition there are for possible further projects.

After having read this document the reader should have clear understanding of what was done during the duration of the project.

1.1 Definitions

Item	Definition
HTML5	HTML5 is the fifth revision of the HTML standard. It is a mark-up language for structuring and presenting content on the internet. This fifth revision implements many new features of which the most important (in the context of this document) is the video element.
JavaScript	*'JavaScript (sometimes abbreviated JS) is a prototype-based scripting language that is dynamic, weakly typed, general purpose programming language and has first class functions. It is a multi-paradigm language, supporting object-oriented, imperative and functional programming styles. '* Source**:** http://en.wikipedia.org/wiki/Javascript JavaScript is implemented in all major browsers. Many HTML elements implement JavaScript API's with which you can manipulate them.
JQuery	JQuery is a JavaScript library. Browsers differently interpret JavaScript. This means developers will have to make many different versions for their applications if they want to support a wide spectrum of browsers. JQuery attempts to provide a uniform API for all browsers, so that developers that use JQuery don't have to take into account the differences among browsers.

	http://jquery.org
Backbone	Backbone is a JavaScript framework, which is loosely based on the Model-View-Controller pattern. It implements a lot of basic functionality to help with Object Oriented programming that can be a bit of a hassle in JavaScript sometimes. http://documentcloud.github.com/backbone/
RequireJS	RequireJS is a JavaScript library. It can inject dependencies into encapsulated pieces of code (modules). It can do this asynchronously or synchronously. Also comes in useful for loading templates. It also supplies the developer with an optimizing tool, which compiles all the dependencies into a single heavily optimized file to improve loading times. http://requirejs.org/
ISO-9126	ISO/IEC 9126 Is an international standard for the evaluation of the quality of software. http://en.wikipedia.org/wiki/ISO/IEC_9126
REST	Representational state transfer (REST) is a style of software architecture. It is used for distributed systems. REST is replacing standards such as SOAP and WSDL at a fast rate. This is because the style is simpler. For a more detailed description see: http://en.wikipedia.org/wiki/Representational_state_transfer
RESTful	When an application conforms to the constraints described by REST it can be considered RESTful.
MoSCoW Method	This is a method for attaching priorities to function requirements. It is used to come to an agreement with the stakeholders of the product. Each of the letters of the word (except the o's) stands for a priority level. M stands for Must-Have S stands for Should-Have C stands for Could-Have W stands for Would-Have For a more detailed description of the method see: http://en.wikipedia.org/wiki/Moscow_Method
SCRUM	*SCRUM is an iterative and incremental agile software development method for managing software projects and product or application development.* Source: http://en.wikipedia.org/wiki/SCRUM
AJAX	Stands for Asynchronous JavaScript And XML. It is a term to describe the asynchronous data retrieval by HTTP with JavaScript.

2. Plan of approach

2.1 Context of project

Noterik

Noterik is a small company with currently eight employees. The office is situated in the centre of Amsterdam on the Prins Hendrikkade 120. Noterik was established in 1996 and focuses mainly on delivering online video services. Noteriks target clients are mainly in the non-profit sector. The European Union and the municipality of Amersfoort are examples of the target groups Noterik delivers its services to.

Online video services

Noterik facilitates the storage as well as the streaming of mainly video over the Internet. The videos stored in the Springfield platform, which was developed by Noterik, contain more than just simple moving images. Instead users can actively edit the videos stored in the platform with an annotation/montage tool.

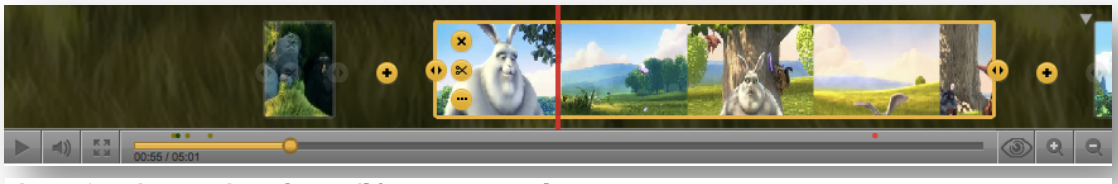


Figure 1 An impression of Noterik's Montage Tool

With this tool users can do all sorts of video editing like splitting, deleting and adding parts of video real time on the Internet. Users can also add annotations to the video on a specific point in the timeline, which will be represented as artefacts overlaying the video. In this way it is possible to attach metadata and content to parts of the video. Because of the addition of this rich data to the videos, Noterik prefers to call these videos 'Presentations'.

Figure 2 An impression of an annotation being shown.

Internal Organization

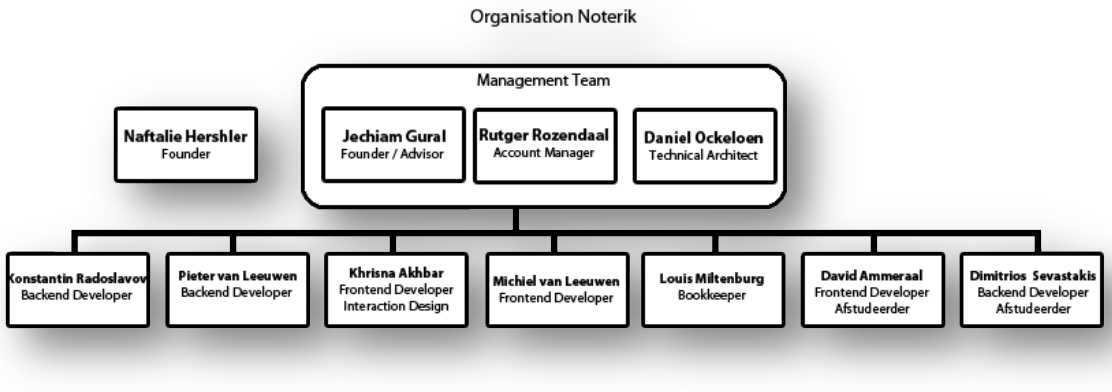


Figure 3. An organization chart of Noterik.

Jechiam Gural and Naftalie Hershler are the original founders of Noterik. Rutger has the function of Account Manager and communicates with clients. Daniel Ockeloen is the main architect of Noterik's software solutions. Together they form the Management Team that is responsible for the business strategy.

Konstantin Radoslavov and Pieter van Leeuwen are the main backend developers. They are responsible for maintaining and developing functionality for the streaming platform and filesystem.

Khrisna Akhbar and Michiel van Leeuwen are the main frontend developers. They are responsible for maintaining and developing new functionality for the presentation tier of Noterik's platform. The author (David Ammeraal) mostly worked solitary. However much of the works done in the frontend so therefore a lot of communication had to happen mostly with the other Front End developers. Sometimes communication with the Back End programmers was necessary, in order to make KrustyJS cooperate optimally with the Backend components. Daniel Ockeloen was invoked in the development and architecture in order make sure that KrustyJS fitted inside the existing architecture.

2.2 Definition of the problem

Noterik currently delivers its video services to its clients with a custom built video player based on Adobe Flash technology. More functionality is continuously added to the player to keep up with the growing demands of customers. Adobe Flash however, is a technology that is gradually being used less and less. Adobe has recently announced that it will not develop further updates for the Adobe Flash platform for mobile devices. Devices based on the iOS operating system by Apple (such as the iPad, iPod and iPhone) do not support Flash. Users of these devices are a potential target group for Noterik's video services and at the moment they can't be reached, and this group is growing quite rapidly. Therefore the functionality currently encapsulated in this player should be transferred to a new environment.

2.3 Definition of the assignment

Research has to be done on how the functionality currently contained within the Krusty player application can be transferred to a HTML5 environment. Krusty is only responsible for the playback of presentations.

Before starting with the main portion of the project, research was done on why the functionality should be transferred to a HTML5 environment. The results of this can be seen in Attachment 3.

Transferring all functionality currently contained found within the Krusty player to the HTML5 environment would be outside of the scope of the project. After contemplating with Noterik we came to the conclusion that the functionality, which should at least be researched, is as follows:

Basic play out

Basic Play out means being able to play, pause and stop a movie stored in the Springfield platform. Scrubbing (navigating through the timeline of a video), volume control and muting should also be available.

Video quality selection

It should be possible to select different qualities for a video. For example: 180p, 360p, 720p and 1080p.

Showing events

It should be possible to show annotations overlaying the video as described in chapter 2.3.

Playlist support

Presentations within the Springfield platform are playlists of items (from a single media source) that should be played in a chronological order. The items are played as if it is a single video, while actually the presentation is a concatenation of temporal selections from a single or multiple media sources. The timeline at the bottom of the player should display the length of the entire presentation. Switching between items should be seamless, and the user should experience the presentation as if it was a single video.

Dock functionality

In the original Flash player there is a button with which the user can open a dock. The dock shows several options with which a user can do the following:

-View info of video
-Share video with others
-Log in to the system
-Tag parts of the video

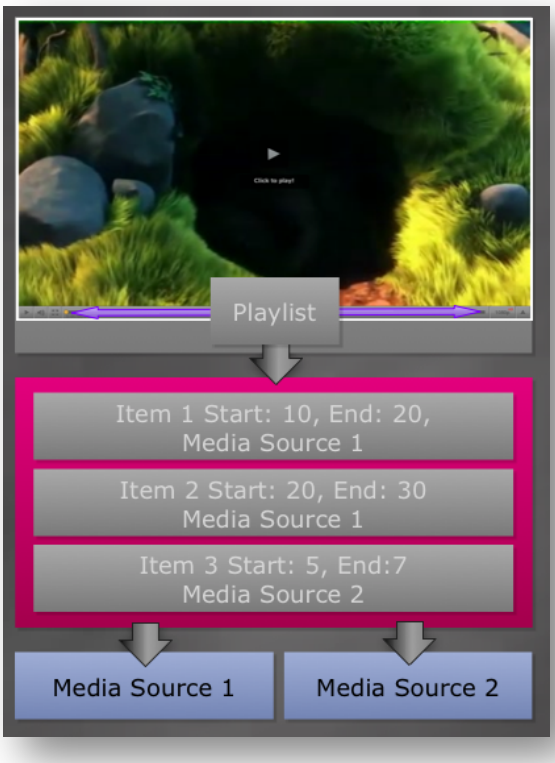


Figure 4. A depiction of how the playlist is build.

However after contemplating with Noterik it has been decided that dock functionalities will not be contained within the HTML5 player for now. Therefore these functionalities fall out of the scope of this project.

After researching the possibilities I should report my findings to Noterik. Following that I should make a Functional Design and a Technical Design.

After that the development of the new player should be initiated. The new player should operate on top of the existing Springfield platform. If needed, certain changes can be made to the backend.

2.3 Products to be delivered

To Noterik

- Plan of Approach
- Research Results
- Functional Design
- Technical Design
- Final Prototype
- Test Document

To the University of Applied Sciences

- Plan of Approach
- Thesis
- Presentation

Personal Documents

- Research Planning
- Product Backlog
- Sprint Backlog

2.4 Planning

SCRUM

It was decided that the development of the application should be cut into three sprints. These sprints are a component of SCRUM method. Each sprint is about three weeks long, and within every Sprint certain functionality will have to be implemented. SCRUM is an iterative and incremental development method for software. A more detailed description of SCRUM will be given in chapter 6.

In the original planning described in the Plan of Approach it was decided that the realisation of the Events should be done in Sprint 2. However after contemplating with colleagues we came to the conclusion that the Events should be moved to Sprint 3 and the Playlist and Quality selection should moved to Sprint 2. This is because Events relies on the Playlist and Quality selection to functional.

Product	**Period**
Concept Plan of Approach	17/01/2012 - 20/01/2012
Research Plan	23/01/2012 - 27/01/2012
Research Results	30/01/2012 - 10/02/2012
Functional Design	13/02/2012 - 16/02/2012
Technical Design	17/02/2012 - 23/02/2012
Sprint 1 (Basic Play out)	27/02/2012 - 16/03/2012
Sprint 2 Quality, Playlist)	19/03/2012 - 06/04/2012
Sprint 3 (Events)	09/04/2012 - 27/04/2012
Presentation/Thesis	07/05/2012 - 27/05/2012

2.5 Organization and responsibilities

David Ammeraal	Student
Michiel Borkent	Supervisor University of Applied Sciences
Konstantin Radoslavov	Supervisor Noterik

3. Research

In order to implement all the functionality that is required, research had to be done. HTML5 is a very volatile standard that is constantly changing. Almost every browser differently interprets HTML5. This means that there are subtle and not so subtle differences between browsers, which a developer will have to take into account before building something. Besides this, the documentation for the JavaScript API, which is implemented in the new HTML5 elements, is severely lacking. So one can not just jump in as opposed to other environments which are clearly defined and interpreted such as Java or C#.

3.1 Research question

A research question had to be defined in order to know what I am going to research. The following question was formulated:

"How can the functionality encapsulated within the Krusty-JS player be transferred to the HTML5 environment?"

Bounds

HTML5 covers a lot of ground. Certain limitations have to be implemented in the research to keep it feasible in the limited amount of time available.

The functionality was limited to the items that are defined in chapter 2.3.

Also the following browsers will have to be taken into account when researching the functionality:

-Firefox
-Safari (Mac OSX and IOS)
-Chrome
-Internet Explorer

All of which are the latest versions.

3.2 Splitting the research question and planning the research

The research question that was established had a wide scope. In order to split this into chunks that could be researched once a time, sub questions had to be established. I established the sub questions and attached a predicted duration to it. The research had to planned over the duration of two weeks. The duration of each sub question was predicted according to the knowledge that was already available of the subject at hand.

Sub questions:
-Does HTML5 support video playback? (30/01)
-Which codecs do the browsers support? (31/01)
-What delivery protocols do the browsers support? (1/02)
-Can simple play out of a video (pause, play, stop, scrub, volume control) be obtained? (2/02 - 4/02)
-Does HTML5 support jumping to time offsets in a (playing) video? (5/02)
-Can the quality of a video be changed in a (playing) video? (6/02)
-Can events attached to the timeline of a video be triggered and displayed? (7/02 - 10/2)

3.3 Executing the research

Every day planned for the research was used. Some items proved to be a bit more complicated as initially predicted, such as the streaming platform, and the quality selection sub questions. In the next chapter a brief version of the results of the research will be described.

HTML5 is such a new standard that sadly there is very little literature on the subject, and the literature that is available is out-dated very quickly because of the current volatile state of the standard. A lot of research had to be done on the Internet by looking at examples.

In order to prove that most of the functionality is actually implementable several prototypes were made implementing a specific sort of functionality relevant to the specific sub question.

3.4 Results

In order to answer the main research question a solution to every sub question was formulated. In each of the following paragraphs a brief description of each solution will be described.

Does HTML5 support video playback?

Yes.

HTML5 implements the new video element. This element allows the developer to add a video element to a HTML page. This can be done as follows:

```
</head>
<body style="overflow: hidden">
    <div>
        <div id="video">
            <video controls="controls" height="320" width="640">
                <source src="http://linkedtv.devel.noterik.com/demo/media/bunny.ogg" type="video/ogg"></source>
            </video>
        </div>
    </div>
</body>
```

Figure 5. Video element in HTML

A source can be defined as a child element of the video element. Multiple sources can be defined. However the way these sources are interpreted differ among browsers. Most browsers will change to a different source in case the first one cannot be found. However, Safari for the iPad for example will not.

Certain parameters can be passed to the element such as "controls", "height" and "width". [1]

When implementing the element without any parameters the default behaviour is different among browsers. Most browsers will show the first frame of the video as an indication that a video element is implemented in the page. Right-clicking the element will provide the user with a context menu in which there is a play option.

What codecs do the browsers support?

This is something that is continuously changing. A video codec is a piece of software that allows a raw video stream to be encoded, compressed and decoded. Raw video provides the highest quality, but the size can be enormous. The storage and bandwidth available to the average user do not allow for the storing and streaming of raw video files. Therefore codecs exist. Codecs encode, decode and compress a raw video file. In this way the file is easier to transfer if the recourses are limited, such as an Internet connection.

Most modern browsers at the time of writing support the video element. However the sources specified have to be encoded in a codec that is supported by the browser. Below is a small explanation of the two most used codecs at the moment of writing of this document.

MPEG4/H.264

Most of the files stored in the h.264 format have the .mp4 or .m4v extension. The MPEG4 container is based on Apple's older QuickTime container (.mov). It is the most used codec for

[1] http://dev.w3.org/html5/html-author/ - the-video-element

videos on the Internet at the moment of writing. It is efficient in the way it encodes its data and thus produces files of relative smaller size, and produces higher quality video pictures in comparison with the other codecs. It is however patent-encumbered, and thus requires vendors and commercial users to pay royalties for the products that use this technology.

WebM

WebM is an open file standard for video files, developed by Google and Microsoft (although Microsoft actually supports the H.264 format in most of its products). Google currently uses it in its Chrome browser together with H.264. However Google has mentioned that it will drop support for H.264 in the future. This announcement was made public over a year ago, and there is no certainty if Google will actually follow up on this announcement.

At this moment the following schema is correct:

Browser/Device	Video Formats	Audio Formats	Multiple Sources
Chrome	MP4, WebM	AAC, MP3, Vorbis	✓
Firefox	WebM	Vorbis	✓
Internet Explorer	MP4	AAC, MP3	✓
Safari	MP4	AAC, MP3	✓
iOS	MP4	AAC, MP3	✓
Android	MP4	AAC, MP3	✓
Opera	WebM	Vorbis	✓

Figure 6. A Schema indicating the current state of browser codec support.

Source: http://www.longtailvideo.com/html5/

For a developer, this adds an extra layer of complexity, as both codecs will have to be served in order to reach a large audience. There is no certainty as to which will win this "codec war", but recently Mozilla (a big player) has announced that it will build in support for the H.264 codec in new releases of Firefox. So it seems H.264 is "winning" at this moment.

What delivery protocols are supported by HTML5

This is also something that differs among browsers. There is no standard streaming protocol such as RTMP for Flash. Most browsers only support progressive download as delivery protocol. However Safari on Mac OSX and IOS do support Apple (Cupertino) Live Streaming.

Progressive download

This is the standard way a video can displayed in HTML5. The source file is defined in the source element of the video element. The browsers requests the file over port 80 at the source specified with a HTTP request.

It will do this with an initial HTTP range request as can be seen in figure 7. This requests the length of the content, as indicated by the "range" variable.

Name	Value
User-Agent	Mozilla/5.0 (Macintosh; Intel Mac OS X 10.7; rv:12.0) Gecko/20100101 Firefox/12.0
Range	bytes=0-
Host	linkedtv.devel.noterik.com
Accept	video/webm,video/ogg,video/*;q=0.9,application/ogg;q=0.7,audio/*;q=0.6,*/*;q=0.5
Accept-Language	en-us,en;q=0.5
Connection	keep-alive

Figure 7. HTTP Range request to the server.

If the webserver has implemented the range parameter it will return the length of the content as such:

Name	Value
Date	Tue, 15 May 2012 14:14:36 GMT
Server	Apache/2.2.8 (Ubuntu) PHP/5.2.4-2ubuntu5 with Suhosin-Patch proxy_html/...
Content-Type	application/ogg
Connection	close
Last-Modified	Tue, 08 May 2012 10:03:16 GMT
ETag	"34005-bbc6f72-4bf837ecfa100"
Content-Range	bytes 0-196898673/196898674
Content-Length	196898674
Accept-Ranges	bytes

Figure 8. HTTP Response from Server containing the content length.

After that ranges of bytes will be requested constantly to buffer the video. In this way only the data, which is needed for playback of the video, is buffered. The video is saved in the cache of the browser and the source location is open and can be downloaded by anyone.

Cupertino Streaming

Safari on Mac OSX and IOS do support the Apple (Cupertino) Live Streaming protocol. This streaming protocol streams its contents over port 80 (the standard http port). The video stream can obtained when requesting a .m3u8 playlist file. The playlist file is obtained through a HTTP request as seen in figure 9.

Name	Value
User-Agent	AppleCoreMedia/1.0.0.11C74 (Macintosh; U; Intel Mac OS X 10_7_2; en_us)
Connection	keep-alive
Host	stream11.noterik.com
Accept	*/*
X-Playback-Sessio...	3FEC11B3-CEA9-4BBE-8B5B-65F2910C20FB
Accept-Encoding	gzip

Figure 9. Requesting a playlist.m3u8

After that a series of requests is initialized for handshaking purposes (such as determining a session id). After that a new playlist.m3u8 file is requested containing a description of the chunks of the video file. The browsers will only stream the chunks needed for the playback of the video. The source of the video however is not known.

Can simple play out of a video (pause, play, stop, scrub, volume control) be obtained?

Yes. The HTML5 element implements several functions that allow for basic play out. Pausing, Playing, Stopping, Scrubbing (or seeking) and volume control are all implemented by default. JavaScript can invoke the functions. Almost all major browsers support the controls argument which displays the video element with a basic set of controls as displayed in figure 10.

These controls use the same JavaScript functions as implemented by the HTML5 video element.

Figure 10 The Basic set of Default HTML5 Video Controls

Does HTML5 support jumping to time offsets in a (playing) video?

Yes. The HTML5 currently has one way of requesting video data from a certain timestamp, and another standard, which is under development.

The 'currentTime' property

The HTML5 video element implements the currentTime property. When setting this property as such:

```
element.currentTime = 5.2;
```

the video will jump to the specified position. It will not load the data up to the position specified. This can be done before playback or during playback of the video file. In this way only the data that is needed for the playback of the current position up until the end will be loaded. All major browsers including Safari for iPad support the currentTime property.

The MediaFragments URI standard

Currently W3C is working on a new standard that supports a new URI standard. With the new URI it will be possible to select temporal and spatial data from a media resource with a URI like these:

```
http://www.example.com/example.ogv#t=10,20h
```

This URI selects the temporal range of 10 to 20 seconds from the example.ogv media resource.

The standard is still very much a work in progress and support differ across browsers. However the developments are very interesting in the context of this project, and keeping an eye on the progress of the standard is paramount.

A full specification of the standard can be found in the footnote of this page. [2]

[2] http://www.w3.org/2008/WebVideo/Fragments/WD-media-fragments-spec

Can the quality of a video be changed in a (playing) video?

Yes and no. Flash has the capability of Adaptive Streaming, which detects the CPU and bandwidth capabilities of the end-user. With this technique it is possible to change the quality of the stream while playing if the end-user is incapable of playing back the video smoothly. Adaptive Streaming is also supported by the HTTP Cupertino Streaming platform, but the support of this platform among browsers is limited.

The other way to switch the quality of the video is by requesting another media resource with a different bitrate. So multiple versions of one video resource will have to be made available. The user can then select in which bitrate the video is to be shown. This however, requires interaction from the end-user, were instead the Adaptive Streaming technique does this automatically.

Can events attached to the timeline of a video be triggered and displayed?

Yes this can be done. The HTML5 video element triggers time update events. The interval of these events can be changed. Most browsers trigger a time update every ~250 microseconds by default. These events are triggered when the video is playing. They can be caught by attaching an event-listener to the 'ontimeupdate' event on the video element in JavaScript:

```
element.addEventListener('ontimeupdate', function(event){
        //Implement logic here.
});
```

The function declaration after the comma is the delegate function, which will be called on a time update. One could check the current time of the video element and check if an event is to be shown on that time.

An overview of all the events that are triggered by the video element, see the footnote of this page.[3]

Any JavaScript logic can be encapsulated by the delegate function, one could for example do:

```
element.addEventListener('ontimeupdate', function(event){
        console.log('This is a temporal based event!')
});
```

And a dialog will be displayed to the user every time an event is handled. You can also manipulate DOM elements in the current page with it like this.

```
element.addEventListener('ontimeupdate', function(event){
        document.getElementById('dialog-box').innerHTML = 'This is a temporal based event!');
});
```

[3] http://www.w3.org/2010/05/video/mediaevents.html

3.5 Conclusions

It is clear that the HTML5 video element has powerful features, which make it very suitable for the functionality that should be implemented in the context of this project. Below is a short summary of the needed functionality and in what way what functionality can be accomplished by HTML5.

Functionality	**Solution**
Basic play out	This can be accomplished by implementing the HTML5 video element API. Streaming can be supported for IOS devices, but progressive download should have priority. The videos in the file system will have to be encoded to both MP4 and WebM to be able to reach all browsers. Adaptive Streaming should not be a priority for now.
Video quality selection	This can be accomplished by making several copies of the same video file with different bitrates. These copies should be made available to the application by means of a web service or something similar. The user should be able to select in which quality the presentation is to be played. Adaptive Streaming is not implementable yet for most browsers.
Triggering and displaying events	By catching the temporal events triggered by the video element it is possible to trigger specific functionality with JavaScript. In this way the entire webpage can be manipulated, and it should be possible to display artefacts on the page at the time an event is triggered.
Playlist support	The HTML5 video element API implements the currentTime function. With this function it is possible to edit the current time position within the video. As the Playlist can exist of several items extracted from a single media source, it should be possible to set the currentTime property to the position at which a Playlist item is to start within the media source and detect when the item has ended by catching the time-update events. When the end is detected the currentTime should be set to the start of the next item.

4. Functional design

In order to clarify between Noterik and me what functionality should be implemented, and how the user should interact with the application a Functional Design was made. The Functional Design is attached to this document as Attachment 4. A short description of the Functional Design will now be defined.

4.1 Functional requirements

The functional requirements were extracted from the requirements that Noterik demanded of the new player. The Requirements are described in chapter 2. An importance rating was attached to each functional requirement according to the MoSCoW method (see chapter 1.2). The following functional requirements were extracted:

ID	DESCRIPTION	PRIORITY
Embed in page	Embed the player into a .html file with an IFRAME, and provide it with a location to request the presentation from.	MUST-HAVE
Embed with dimensions	It should be possible to provide the player with a dimensions parameter, so that the player will set the dimensions of the player and video to the dimensions provided.	SHOULD-HAVE
Show screenshot	Show a screenshot for the video after initializing the player.	MUST-HAVE
Play	Play the presentation from the start or from the point where it was paused. It should be able to play a presentation consisting of a playlist of several video items.	MUST-HAVE
Pause	Pause the presentation from playing.	MUST-HAVE
Timeline	Show a timeline in which you can see the progress of the video playing. The timeline should span all the playlist items.	MUST-HAVE
Scrubbing	Move the box on the timeline to a place where you want the presentation to start playing from.	MUST-HAVE
Show screens while scrubbing	Show a screenshot of the presentation at the time where the scrubber is placed on the timeline.	SHOULD-HAVE
Mute presentation	Mute the volume of audio in the presentation.	MUST-HAVE
Control volume	Control the audio (louder or quieter) of the presentation.	MUST-HAVE
Set to full screen	Make the presentation player full screen.	COULD-HAVE
Show timer	Show a timer in which you can see the progress of the video and the duration of the presentation.	MUST-HAVE
Change quality	Select and change the quality of the presentation. The qualities that can be selected are 180p, 360p, 720p and 1080p.	MUST-HAVE
Show event	Show an event planned for a certain time period in a video.	SHOULD-HAVE

4.2 Non-Functional requirements

The non-functional requirements were established according to the ISO-9126 standard. ISO-9126 defines six categories of quality characteristics, each with a specific set of sub characteristics. For a full overview of the characteristics see chapter 2.2 of Attachment 4.

For each sub characteristic an importance rating of 1 to 5 was defined, 1 being of very low importance and 5 being of utmost importance. A motivation for each importance rating of each sub characteristic was also defined.

The most important characteristics were as follows:

Characteristic	Motivation
Interoperability	Krusty-JS will be build on top of the Springfield stack. It means that it will have to communicate with REST to the Springfield stack. This is of utmost importance because all the functionality implemented in the Krusty player is heavily reliant on data stored within the Springfield platform. This data can only be reached through REST.
Understandability	All the elements with which the user has to interact have to be understandable. This is because the users that are watching the video will usually not be an expert user. This is a trivial, but very important subject nevertheless.
Analysability	This application will serve as a base for more advanced versions. To supply a stable and good base for the next version, it is important that the code is very analysable.
Co-existence	The application will have to work in a webpage where other applications might be active. All the logic of the application should be encapsulated in its own namespace, and it should not interfere with other applications in any way.
Replaceability	Some browsers might not be able to support all of the Krusty-JS functionality. In case this happens, a fall back mechanic must be in place to fall back to the Flash version. Also removing this component from the Springfield Platform should not have any influence on the other components of the Springfield platform.

4.3 Data model

The presentation contains a single playlist. This playlist contains the video items needed.

The playlist will calculate the total duration of itself by using the length of the videos. A playlist might also contain events that should be triggered at certain times.

The video contains at least one RawVideo. The RawVideo is a reference to a media source. A video contains several RawVideos because each RawVideo is a different quality of the same Video.

The Presentation was separated from the Playlist because the Presentation could contain other objects than just Playlists, such as metadata that is not relevant to the playlist.

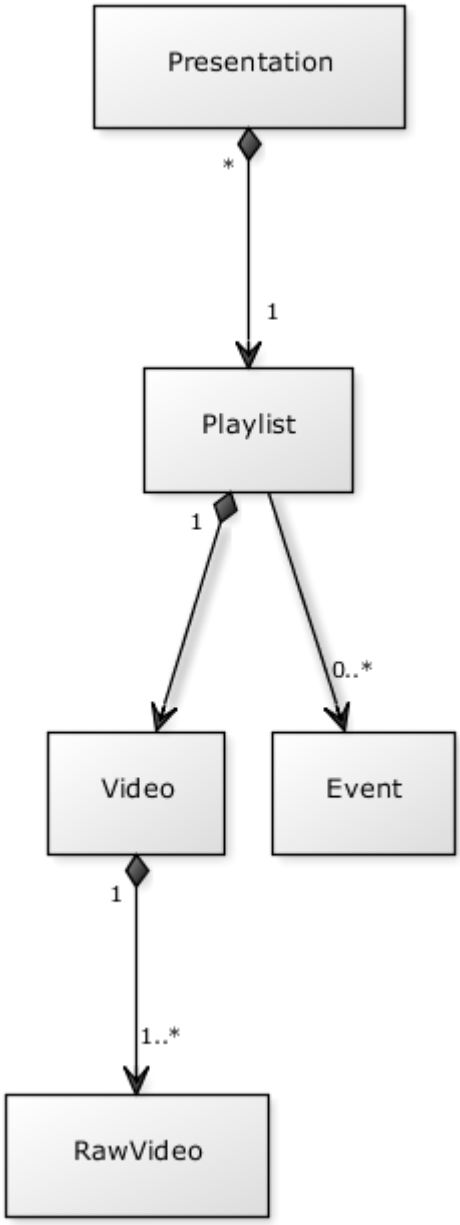
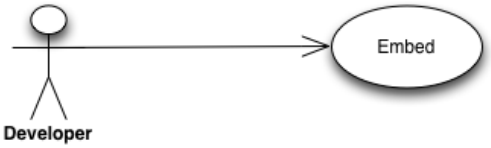


Figure 11. The data model of KrustyJS

4.4 Use cases



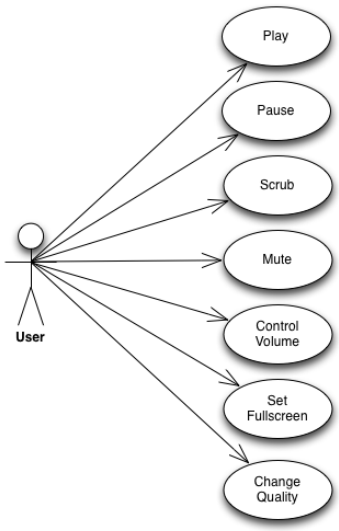


Figure 13. Use-cases.

There are two main actors in the context of the application. First the developer that is going to embed the Krusty-JS into an HTML5 page. And second the user that is going to interact with the player. The usecases were extracted from the functional requirements.

4.5 Screen designs

Basic Layout

This is the basic layout of the Krusty-JS presentation player. This is a mock up. The end result should look exactly like the Flash version.

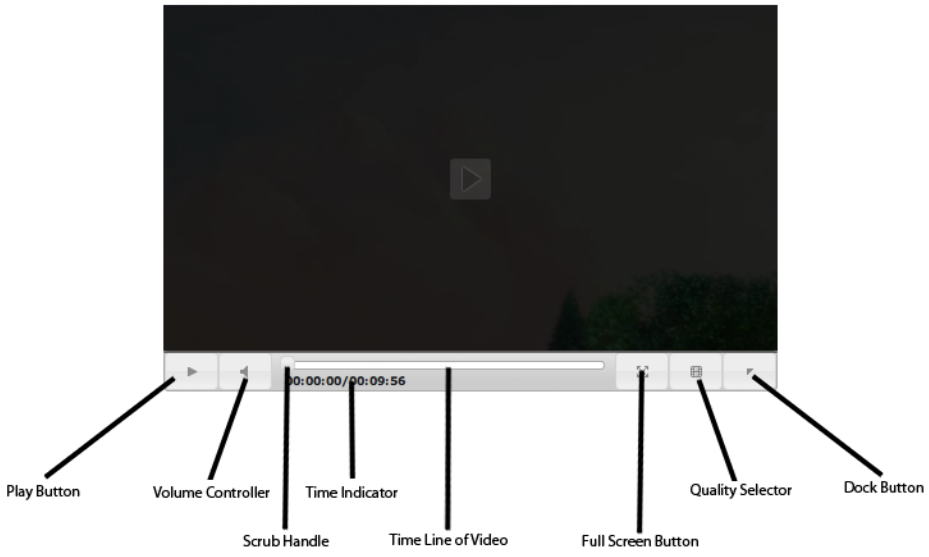
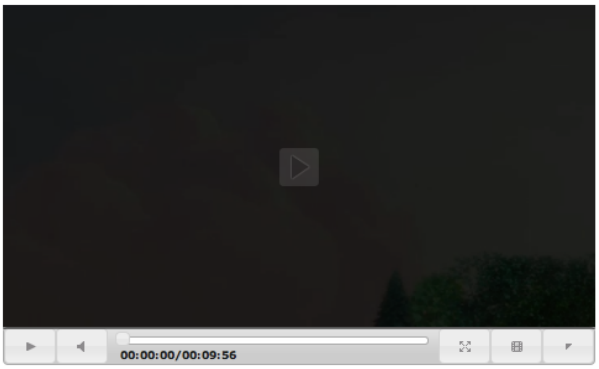


Figure 12. Basic layout of the player.

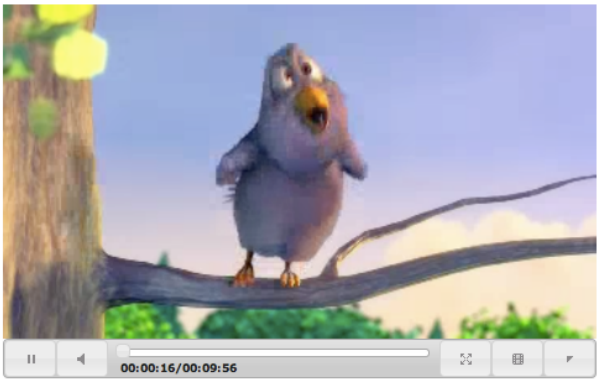
Loading presentation

The player is initialized, but all buttons are disabled, a message is shown to the user to inform him that the presentation is being loaded.



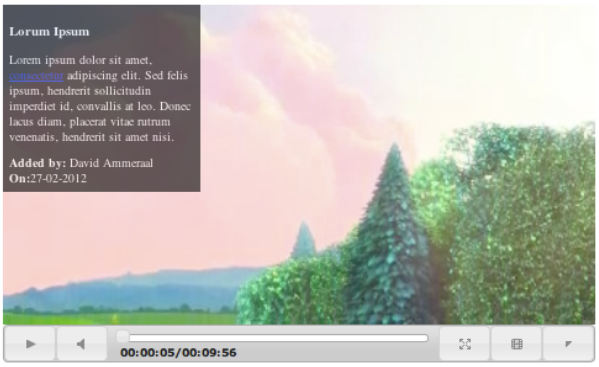
Done loading

This is the screen after the first video of the presentation is loaded. The duration of the video is added and all buttons are available. Also a play button is added to the overlay to indicate that video is ready to start playing.



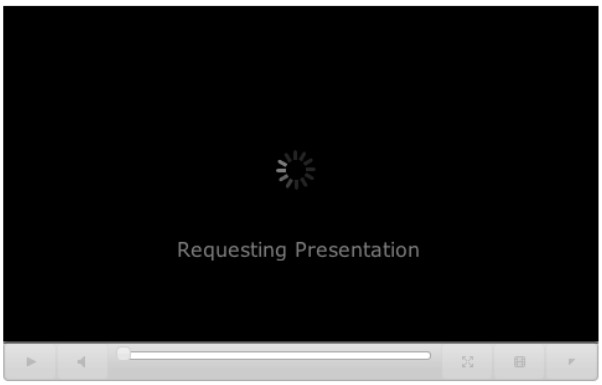
Video playing

This is the screen when a video is playing, all buttons are available for use, and the progress indicator is updated every second to indicate at what time the video is. The play button is changed to a pause button.



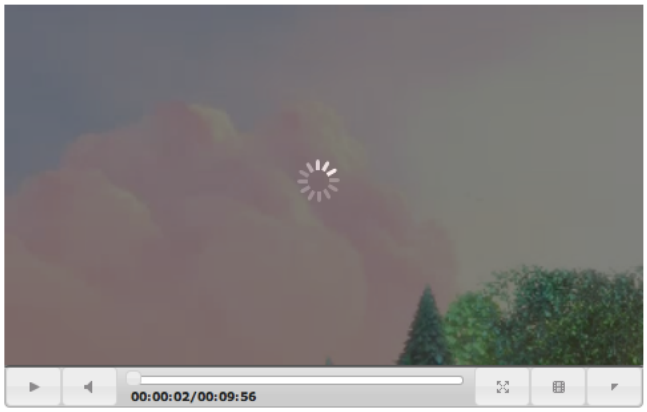
Showing event

The event will overlay the video.



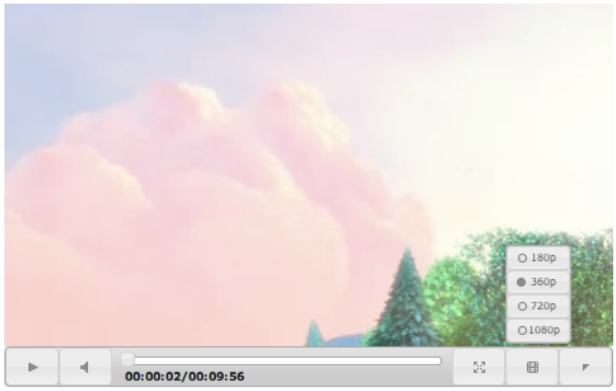
Buffering

When the buffer of video has run out, the user will be informed that the video will have to buffer before playing again.



Quality selection

The quality currently playing is blacked out.



5. Technical design

5.1 Current situation

A diagram of the Springfield architecture can be seen in figure 13. At the bottom there is the hardware layer. It exists of a mix of Windows and Linux (Ubuntu) servers collocated at the xs4all datacentre in Amsterdam. On top of the hardware layer there is the operating system layer. Currently there are only Windows and Linux servers, but Max OSX servers could be introduced in the future.

On top of the operating system the services operate. This diagram does not show the dividing of the software logic in layers. But in the context of this project this is not important. Each service is a black box (which means that knowledge of the internal workings is not necessary), which provides an interface to certain functionality.

On top of this, there is the Firewall Proxy layer named Bart. This is a facade. When the frontend sends a request to Bart, Bart decides to which service the request should be sent. This provides another layer of abstraction. One does not need to know which services are available, a request could just be sent to Bart, and Bart will know which service is responsible for handling the request, and will return the results of the request to the Front End. Only the interface to Bart has to be known by the Front End. Bart can be reached through HTTP requests.

In the Front End layer Krusty operates. This is the layer that is responsible for presenting the presentation to the end user. It presents the data with Flash technology. It requests its data by sending HTTP requests to Bart.

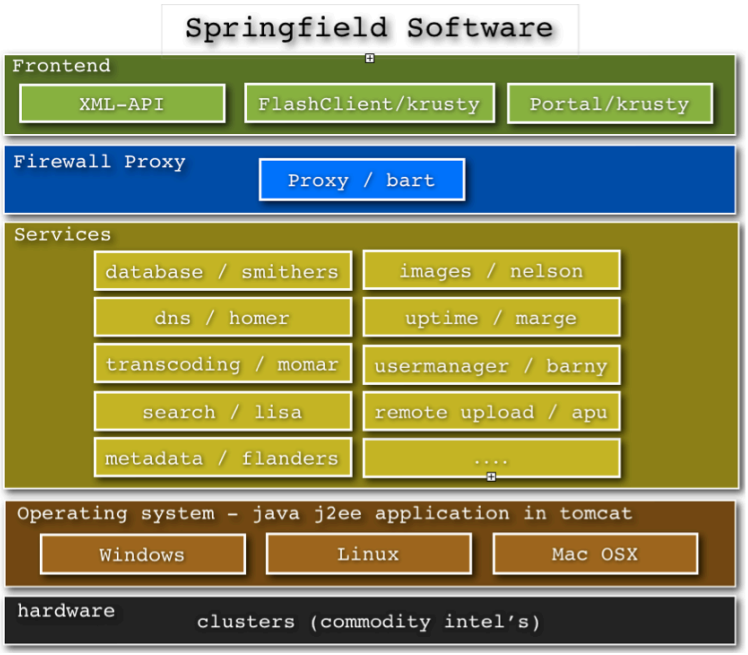


Figure 13. A graphical representation of the Springfield platform.

5.2 Application architecture

Krusty-JS is going to be operational in the context of the Springfield platform. That means that it is already going to be part of an existing architecture. However since the functionality that is going to be encapsulated by Krusty-JS is quite widespread, an architecture was determined for Krusty-JS.

Goals of the architecture

In order to make the application conform to the functional and non-functional requirements defined in chapter 4. Goals to the architecture are defined as follows:

Goals	Non-functional requirements
Should be able to operate on top of the Springfield platform.	**Interoperability**
Does not interfere with the internal working of Springfield, should be a seperate component.	**Interoperability**
Clear seperation of the presentation logic from the model logic, in order to make it possible to make changes to the presentation without interfering with the model logic.	**Replaceability**
In order to increase analysability and testability of the the application, functionality should clearly be encapsulated in correct classes.	**Understandability**

Domain Data Model

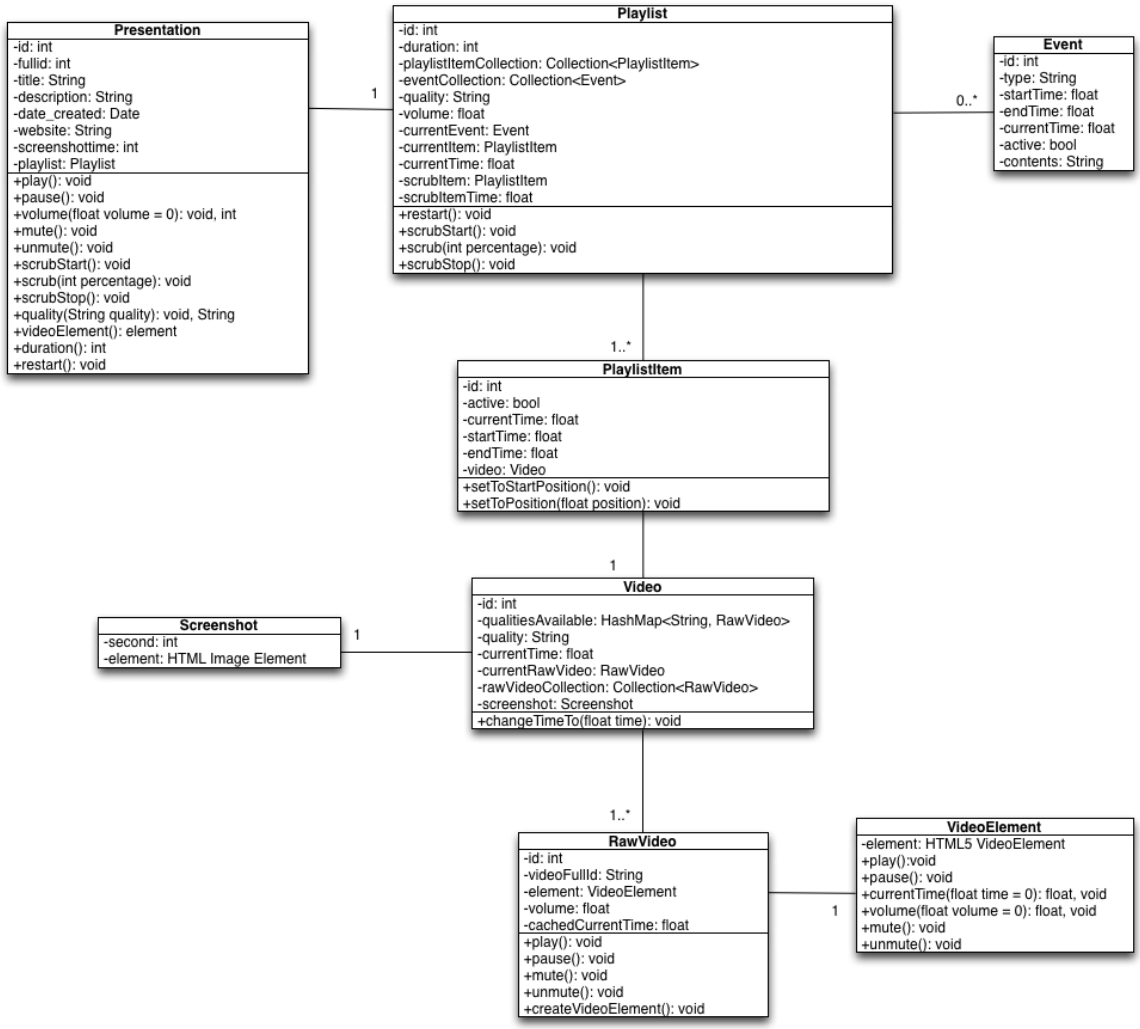


Figure 14. The domain model

Presentation: Contains the metadata of the presentation, things such as the description and the date it was created on. It also contains a reference to the playlist. It also serves as the main entry point for the upper layers. The Presentation class figures out how the domain is constructed and will invoke the necessary functions on the objects of the domain when it receives a request from another object.

Playlist: This model contains a collection of all the PlaylistItems objects in the collection. It also calculates the duration of the presentation. It will keep track of the time the current item is at and will calculate how the actual times of the media source can be calculated to presentation time. It is responsible for switching PlaylistItems after a PlaylistItem has finished playing, or when the user is scrubbing through the presentation. It will keep track of the events and will detect when the events should be triggered. Playlist also maintains the state of the quality and volume across all the playlist items. Thus when switching items, the Playlist will know in what quality and volume the last item was playing and will pass this information to next item in the queue.

Event: Events are attached to the timeline of a Playlist. This object contains the data of the Event, which is necessary for application functionality and presentation. Specific types of Events will have to be abstracted because they will be able to hold a wide spectrum of contents.

PlaylistItem: A PlaylistItem contains things such as the current time the PlaylistItem is at relative from the start time of itself. It will trigger time events containing the current relative time from the start and the absolute time the video is at. Other objects (Playlist) can attach delegate functions to the event. The playlist item will only trigger events if it is active (being played).

Video: The video contains references to RawVideo objects. It maps the RawVideos that are available and attaches a quality rating to them, such as 180p, 360p, 720p and 1080p. It is then easy to select a RawVideo by a specific quality. It also keeps track of which RawVideo is currently playing and it will propagate the events triggered by the current RawVideo and make them available for other objects.

Screenshot: This model contains the information of the location where the screenshots for the current video can be retrieved. When changing the second property, the screenshot object will automatically retrieve the actual screenshot for the given second and put it in the element property. When a screenshot is retrieved it will trigger an event. Other objects can attach delegate functions to this event.

RawVideo: This object contains the information about where the video media source can be retrieved (the HTTP location). When a RawVideo is made active it will construct a VideoElement at real time. Video Elements are not preloaded because this would put a heavy strain on the connection of the client.

VideoElement: This contains a reference to the actual HTML5 video element. It attaches event listeners to the HTML5 video element and make these events available in a uniform way to other objects, which can attach delegate functions to them. Different implementations of the VideoElement will be necessary because the events triggered by the HTML5 video element differ among browsers. VideoElement will be an abstract class, which provides the signatures for the concrete classes.

Layers of the architecture

A three-layer architecture will be implemented with the addition of a utility layer that encapsulates certain functionality such as data validation and date formatting which is used across the entire application.

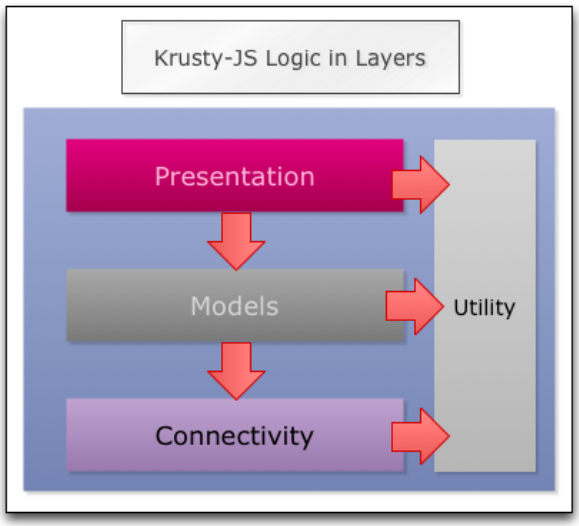


Figure 15. The Architectural Layers

The connectivity layer

This layer is responsible for creating and maintaining a connection to the Springfield platform. It retrieves the data and makes it available for the model layer. The connectivity layer is separated from the Model layer because the models should be able to work with different data sources. Currently, only the connection to Bart should be implemented, but it should be possible to replace this with other data sources in the future.

The models layer

The Models layer is responsible for reading out the domain data from the data source and should parse this to objects. These objects contain the domain data as well as most of the application logic. The objects are capable of observing each other, and change their state according to the state of another object. Because of this, a controller layer was not implemented as a lot of the application logic can be realised by these observing models. There should be a single interface to this layer, so as to keep a clear line of communication between the layers.

The presentation layer

The Presentation layer is responsible for presenting the application to the end-user. It is responsible for implementing the video element. It is also responsible for receiving user-interaction and propagating these actions to the Models layer.

Components of the Application

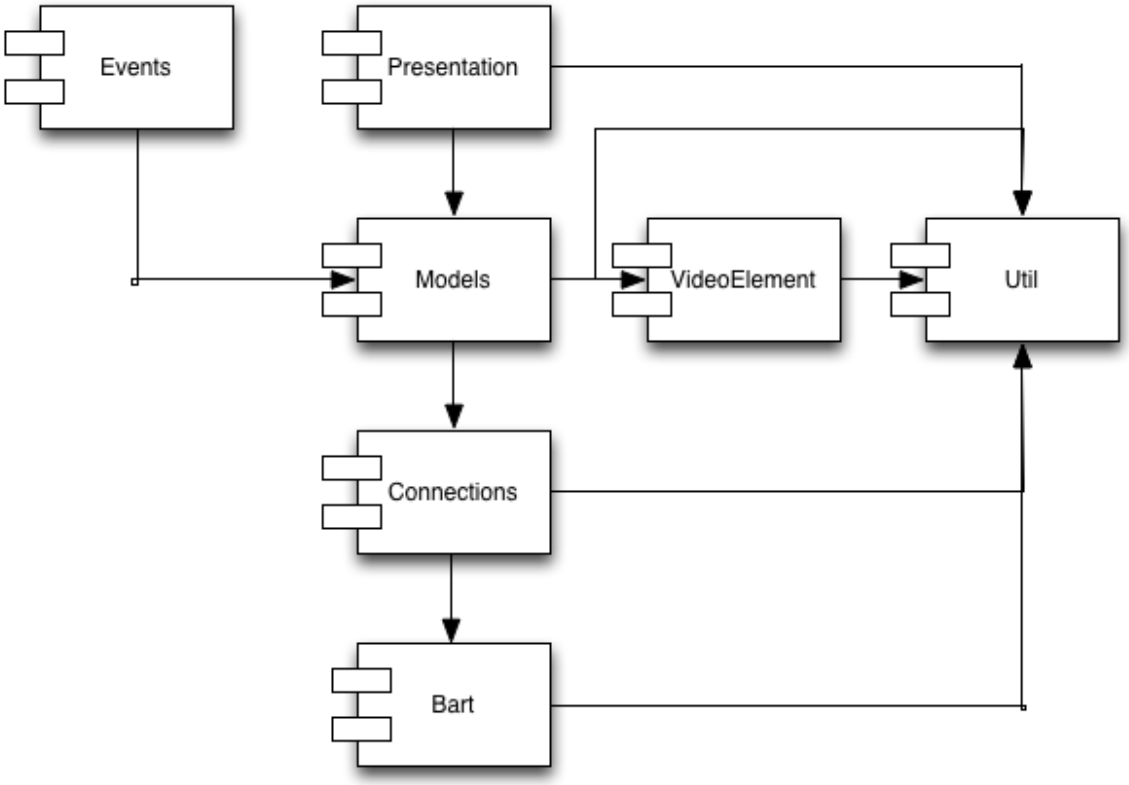
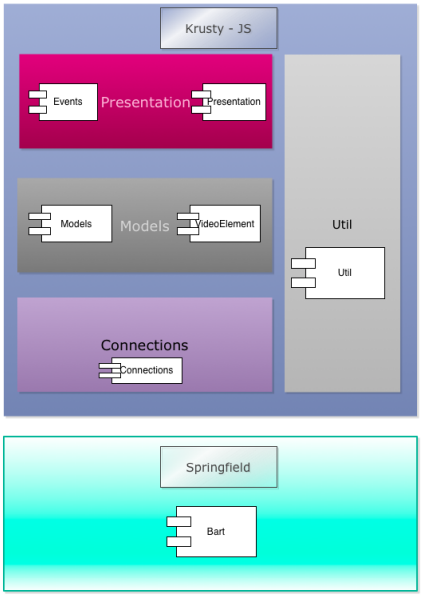


Figure 16 The components of the application.

Connections: This component contains the functionality of connecting to the Bart servers, and retrieving the data sources needed for the models. They communicate with Models through events. When the data has been loaded, Models is informed about this and can start constructing its objects.

Models: This component contains most of the domain and application logic. Other components can communicate with Models by making an instance of the Presentation class (which is located in the Models component). This object implements public functions to manipulate Presentation play out and retrieving metadata. It also triggers Events to which other components can subscribe.

Figure 17. Components in Layers



VideoElement: This component contains the abstraction class of the VideoElement. The HTML5 video element implements an API of functions and events. A lot of these events are interpreted differently across browsers. Therefore it was decided to wrap the element in a wrapper object, which can manipulate the HTML5 video element. The wrapper object implements most of the HTML5 video element functions but the implementations of these functions might differ according to the environment the application is operating in.

Presentation: This component contains the Presentation logic. It displays the video player, screenshots and events. It will also provide controls with which the play out of the presentation can be manipulated.

Events: This component contains an array of events, which can be listened to by other components. Other components trigger a wide arrange of events, which might not all be interesting to other components. Therefore it was decided to implement another component, which listens to all events and normalizes these events. Only events that might be interesting to other components such as 'presentation-playing', 'presentation-stopped', etc., are triggered.

Util: Util contains a range of utility or helper classes used to format and validate data uniformly across the application. It also contains a class for detecting the environment in which the application is running.

5.3 Technical implementations

Loading dependencies

It was decided that RequireJS would be used to structure the application. RequireJS is a JavaScript file and module loader. Modules can be defined and dependencies can be loaded asynchronously. RequireJS provides the user with a single entry point for an application, the so-called 'main' module (equivalent of Main.java). The order in which the dependencies are loaded is not important. RequireJS figures out the relations between dependencies. Modules are defined as such:

```
define([
        "jquery",
        "underscore",
        "backbone",
],
function($, _, Backbone){
        //MODULE CONTENTS HERE
}
```

The define() functions takes two arguments. The first argument contains an array of dependencies; the second argument is a function in which the contents of the module can be defined. Each dependency defined in the array is passed to an argument of the module function. In this way the dependencies are only injected into that specific module and they do not contaminate the common namespace. RequireJS also provides the developer with a optimizing tool. This tool can be used to heavily optimize the code. It merges all the modules into a single file and minimizes the code. In this way, a browser interprets the code faster.

Implementing MVC (Model, View, Controller)

Backbone is a JavaScript Framework. It provides a lightweight implementation of the MVC (Model, View, Controller) presentation. In this way presentation logic can be separated from model logic. This is important if one wants to change the presentation logic without having to change the model logic. It implements a publisher/subscriber pattern in order to communicate changes in the state of an object to other objects. Objects can trigger events and other objects can subscribe delegate functions to handle to these events.

A model could be defined like this:

```
var Robot = Backbone.Model.extend({
          defaults: {
                    name: "Bleepy",
                    description: "Bloop"
          },

          initialize: function(){
                    //Constructor logic goes here.
                    this.on('change:name', function(){
                              console.log(this.get("name"));
```

```
                    });
          },

          sayName: function(){
                    console.log(this.get("name"));
          },

          changeDescription(description){
                    this.set("description", description)
          }
});
```

In this example Robot is a model. It has the properties "name" and "description". Each child of the Backbone.Model class implements the get(attribute) and set(attribute, value) functions. These functions allow retrieval and changing of the properties of the model. The model can listen to changes in it properties, when a property changes it triggers a "change:<property>" event. In this way changes to the state of the model can be communicated to other objects.

A View could be defined as this:

```
var RobotView = Backbone.View.extend({
         robot: new Robot();
         descriptionBox: null;

         initialize: function(){
                  descriptionBox = this.$('#description-box');
                  this.robot.on('change:description', this._descriptionChanged, this)
         },

         _descriptionChanged: function(){
                  this.descriptionBox.html(this.robot.get("description"));
         }
});
```

This view creates a new Robot. Each class that inherits from Backbone.View implements the this.$ function. This is a reference to a JQuery object. JQuery makes it easy to traverse the DOM tree of a HTML page. In this example we want to have a reference to the HTML element with an id of 'description-box'. This could be a div like this:

```
<div id='description-box'>
         <!-
-Description of Robot goes here-->
</div>
```

In order to get a reference to this element you could invoke the following JQuery call:

```
this.$('#description-box');
```

The # symbol means that JQuery should be looking for an element with the id of 'description-box' within the DOM tree.

The constructor of RobotView uses the JQuery object to get a reference to the element. After that the constructor makes the view listen to changes of the description of the Robot. It attaches a delegate function (_descriptionChanged) to the event. And finally passes the context in which the delegate function should be invoked (this).

When the description of the Robot gets changed it triggers the 'change:description' event, and the 'descriptionChanged' function gets invoked. 'descriptionChanged' then changes the contents of the description-box within the view. In this way the views are very loosely coupled to their model. This makes it possible to completely replace the views without having to change the implementation of the models.

Backbone also provides a more familiar environment if the programmer is used to programming in non-prototype based programming languages such as Java.

Backbone was combined with RequireJS in order to encapsulate the functionality as much as possible across the application.

Building a connection to Bart

Bart can be reached through a HTTP request. In earlier versions of the Krusty player, built in Flash, Krusty had to do several asynchronous HTTP requests to Bart in order to get all the data it needed. This proved to cause a lot of overhead because the server had to handle each request, and as the amount of users looking at videos rose, so did the amount of requests.

In order relieve the server it was decided that a single request should be sent to Bart defining which data is necessary for the playback of the presentation (the so-called Quickstart request). Bart then decides where that data should be retrieved and sends it back asynchronously to Krusty. In this way only a single request is necessary to Bart.

Each presentation in the Springfield platform has its own location in the Filesystem that can be reached by BART with a URI through HTTP. This is an example of a presentation that belongs to me:

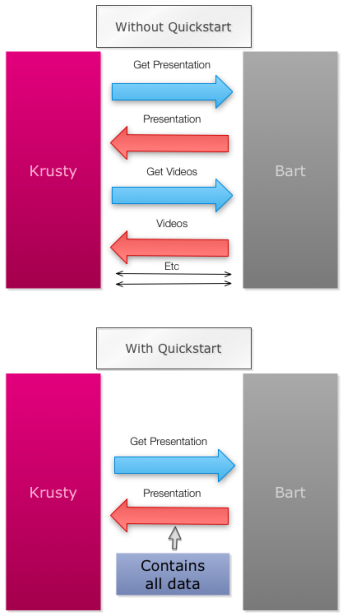


Figure 18. Connecting to Bart.

```
http://bart1.noterik.com/bart/domain/springfieldwebtv/user/david/collection/3/presentation/2
```

The quickstart file can be retrieved by sending an asynchronous (AJAX) POST HTTP request to this location with the following contents:

```
<fsxml mimetype='application/fscommand' id='dynamic'>
        <properties>
                <handler>/dynamic/presentation/playout/flash</handler>
        </properties>
</fsxml>
```

The mimetype declares that this is a file system command. The id='dynamic' means that the id of the presentation should be extracted from the URI. The handler element defines the object that will be able to retrieve all the data needed for play out. In the future an extra handler should be added which retrieves the data, which is necessary for HTML5 playback, but for now the HTML5 player should work with data that is retrieved for the Flash playback.

After sending this request by AJAX the client will have to wait for the Bart to retrieve the data.

Implementing the connection to Bart

In order to retrieve the data an AJAX call had to be made to Bart. After the data was retrieved, the data had to be validated. A class BartConnection is created, which can make an ansynchronous HTTP POST request to a Bart server.

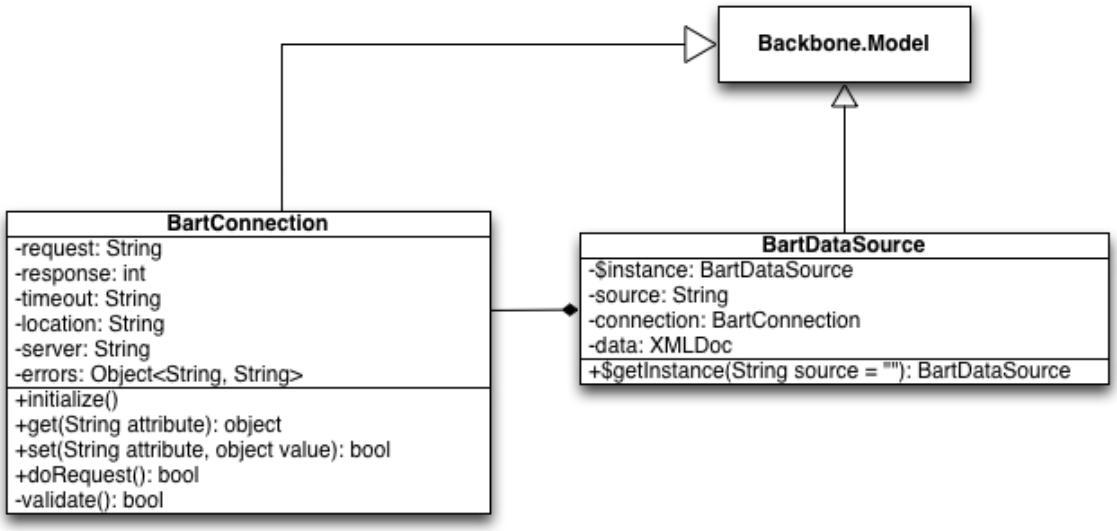


Figure 19. Classes for connecting to Bart.

The class BartDataSource was made to load the data retrieved from BartConnection into an XMLDoc variable, which could be traversed by an XML parsing tool.

The BartDataSource is a Singleton. A Singleton class contains a static instance of itself inside. This instance can then be used by other objects by calling the static method getInstance() while maintaining the same state. This was done because each instance of an object needs a data source without having to re-instantiate the BartDataSource every time, as this would reconnect to Bart every time a model gets instantiated.

Parsing the contents of the Quickstart file retrieved from Bart into the Models

The data, which was stored in the Quickstart XML file has to be mapped to values of the Bacbone models in the application. The behaviour for mapping the xml values to object values has to be identical across all the models, this is because the functionality of retrieving data from that single data source is identical across most of the models.

The Model and Collection classes defined within Backbone.js implement the fetch() function. This function retrieves data and maps the values of the data into the properties of the model. In the default implementation the fetch() function retrieves JSON data from a HTTP location. This means it does a separate HTTP request for every object. However in the context of this project only a single request should be done to the data source in order to relieve the strain on these servers. So the fetch() function will have to be overwritten in custom implementations of the Model and Collection classes which will have to be inherited by the models which need to retrieve their data from the Quickstart file.

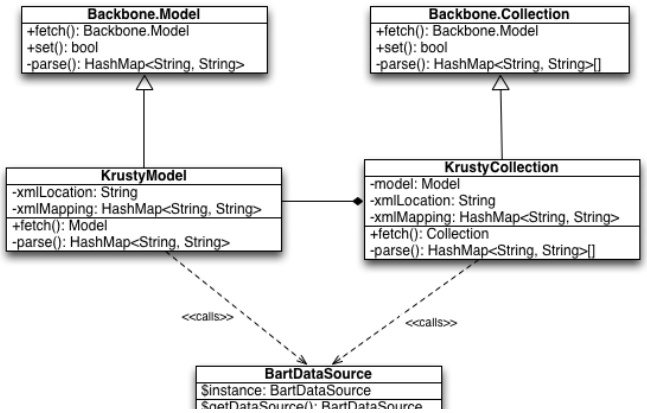


Figure 20. The basic models.

The classes in figure 20 were defined.

KrustyModel and KrustyCollection each inherit all of the functions of Backbone.Model and Backbone.Collection. The properties xmlLocation and xmlMapping are added to each of the child class. The xmlLocation property

defines where the data for the model can be found in the BartDataSource and the xmlMapping defines which field belongs to what property. The xmlLocation and xmlMapping are defined in XPath. XPath is a standard for traversing xml files. [4]

The location and the mapping of the values are defined as follows:

```
xmlLocation: //presentation,
xmlMapping: {
        id: '@id',
        fullid: '@fullid',
        title: '/properties/title',
        description: '/properties/description'
}
```

Jath[5]

Jath is a JavaScript library with which XML data can be parsed into JavaScript objects by using XPath to map where the data for a given object can be found.

Jath implements the Jath.parse() function. It can be invoked like this. A mapping of strings to XPath values can be defined like this:

```
mapping: {
        id: '@id',
        name: '/name'
        description: '/description'
}
```

With Jath it is possible to traverse the XML file and parse the values of the certain relevant elements into JavaScript objects.

[4] http://www.w3schools.com/xpath/xpath_syntax.asp

[5] https://github.com/dnewcome/jath

Constructing the GUI

In order to implement the functionalities according to the functional design, a user interface will have to be constructed. The user interface has to look exactly like the Krusty Flash player. The Flash player is styled with a Flash based technology. The HTML5 player has to be styled with HTML and CSS.

The following View classes are defined.

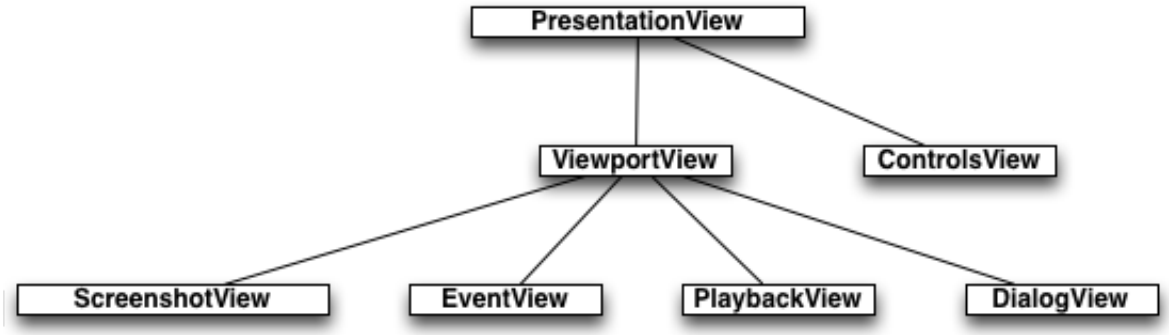


Figure 21. Views of Krusty-JS

PresentationView is the main container of the application. It contains the ViewportView and ControlsView. ControlsView contains the video controls buttons, such as play, pause and volume control. The ViewportView contains four views which all overlay each other. The ScreenshotView displays the screenshot at the start of the video, and also shows the screenshots while a user is scrubbing through a presentation. The EventView shows the events which are triggered. The DialogView is responsible for showing a dialog to the user showing the current state of the presentation (loading, ready to play etc). The PlaybackView is the container of the HTML5 video element, and shows the actual playback of the presentation.

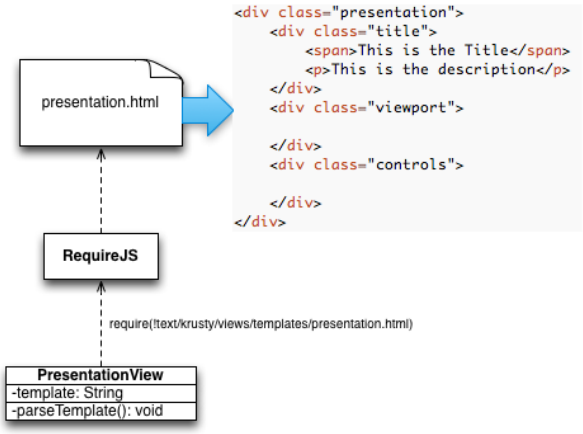


Figure 22. Templating system.

The views each contain a template, which is dynamically loaded by RequireJS. These templates consist of HTML code, which the view can then manipulate. These templates can be style by CSS as they are just plain HTML. This is done to keep HTML and CSS out of the JavaScript view code. This makes it clear how the HTML is composed, as JavaScript does not dynamically generate it.

Each view is responsible for a specific type of functionality. Some are just containers (such as the presentation and viewport view), while other listen to and control the state of the presentation (ControlsView, EventView, PlaybackView and DialogView). The views communicate and listen to the Model layer by an instance of the Presentation class. A single Presentation is instantiated in the Main file. This instance is passed to PresentationView, which then injects this instance across all its children, which in their turn inject the instance to all their children. In this way a single instance of the Presentation is assured, since it is only instantiated once.

Creating a Facade into the Models Layer

A way has to be found to connect the Views to the Models. A class Presentation is defined, which will be the central Facade into the Models layer. Apart from being a Facade this class also contains simple metadata such as the title and description. The Presentation class knows where calls made by the Presentation layer should be redirected to, and it makes events triggered by the Models layer available to other objects. In this way there is a clear communication line between the views and the models. All the views listen to a single object. And if they want to manipulate the models they can do this by calling a function in the Presentation class.

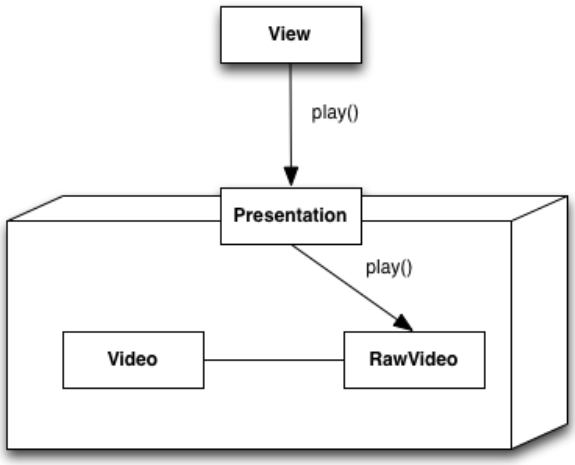


Figure 23. The Presentation Facade.

Codec selection

Video Codec support differs among browsers. In order to assure that a correct video file (with the correct codec) is loaded for every browser, a codec selection mechanism will have to be implemented.

This can be done by building a static Environment class that can detect in what context the application, is running. The Environment class can be used by a class VideoElementFactory. This VideoElementFactory will detect in what environment the application is running, and will create a correct VideoElement, containing the correct codec for the given environment. This VideoElement will also aggregate all the Elements triggered by the VideoSource and will normalize them so that the event triggering behaviour is uniform for each VideoElement.

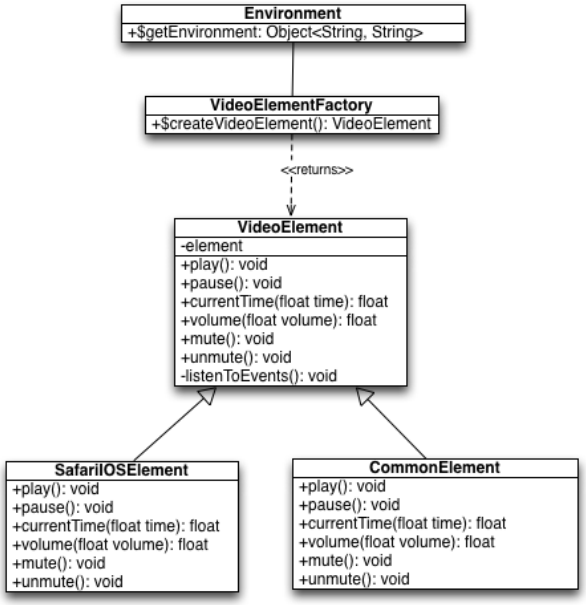


Figure 24. Abstracting the video element.

Playing out the Playlist

The presentations contained within the Springfield platform consist of temporal ranges from a single or multiple video sources. These temporal ranges are defined as PlaylistItems. A PlaylistItem is basically a fragment from a video with a start time and duration. All these PlaylistItems are supposed to be played out in chronological order in such a way that it seems you are actually watching a single video. The playlist items are defined within the quickstart.xml file as can be seen in figure 25.

The videoplaylist element contains video elements. The naming might be quite confusing but actually these are playlist items. They have a start time and a duration. They also have a position which defines in which order the playlist items should be played. The referid refers back to the actual video element that was described earlier.

In order to load these properties, the video elements were parsed into PlaylistItem objects with the method described in chapter 6.3. Each PlaylistItem object also gets a reference to the Video that belongs to them, so that they can listen to time-updates triggered by the VideoElement.

After this had been done an object had to be created that could actually control these objects. The class Playlist was created. The Playlist is responsible for maintaining the state of the current PlaylistItem to the next one, so that things such as the current quality and current volume could be passed over to the next PlaylistItem. This class is also responsible for implementing the scrubbing functionality.

```
<videoplaylist id="1">
  <properties/>
  <video id="2" referid="/domain/springfieldwebtv/user/david/video/5">
    <properties>
      <position>1</position>
      <starttime>40000</starttime>
      <duration>20000</duration>
    </properties>
  </video>
  <video id="4" referid="/domain/springfieldwebtv/user/david/video/5">
    <properties>
      <position>3</position>
      <starttime>80000</starttime>
      <duration>20000</duration>
    </properties>
  </video>
 <video id="5" referid="/domain/springfieldwebtv/user/david/video/5">
    <properties>
      <position>4</position>
      <starttime>20000</starttime>
      <duration>4999</duration>
    </properties>
  </video>
 <video id="7" referid="/domain/springfieldwebtv/user/david/video/5">
    <properties>
      <position>7</position>
      <starttime>50000</starttime>
      <duration>10000</duration>
    </properties>
  </video>
</videoplaylist>
```

Figure 25. Playlist in quickstart.xml

The sequence diagram in figure 26 above describes what happens when the Playlist gets initialized. In order to begin from the start of the presentation it requests the first item from the PlaylistItemCollection. It then calls the selectItem() function, with the first item as the argument. The selectItem function does the following things:

-Maintain the state (quality, volume) of the last playlist item into the new item which is selected and passed as an argument to selectItem().
-Makes the Playlist listen to events triggered by the new item.
-Sets the item to the position passed as an argument, if no position is passed, it will just set the position to the start time of the playlist item.

After the PlaylistItem has started playing it will trigger time-updates every 250ms (default interval). The PlaylistItem triggers two time-update events.

playlist-item-time-update: Contains the time of the playlist relative to the start time of the video. This is the time that is displayed by the slider in the ControlsView.

video-time-update: Contains the time of the video is currently at. This is the time relative of the start time of the video source.

The Playlist listens to these events and according to times of the video and playlist it can calculate the current playlist time. It recalculates the playlist-time every time the PlaylistItem triggers a time-update:

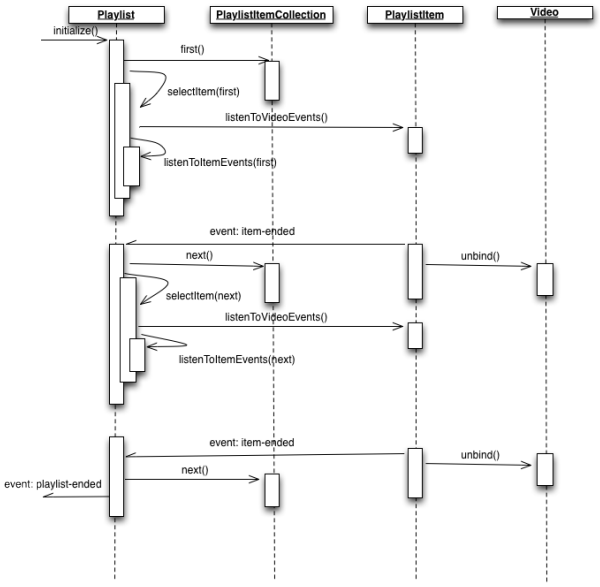


Figure 26. Playlist functionality.

playlist-time-update: Contains the actual playlist time, so the time relative from the start time of the first item of the playlist.

This diagram shows the different time events, which are triggered by the application.

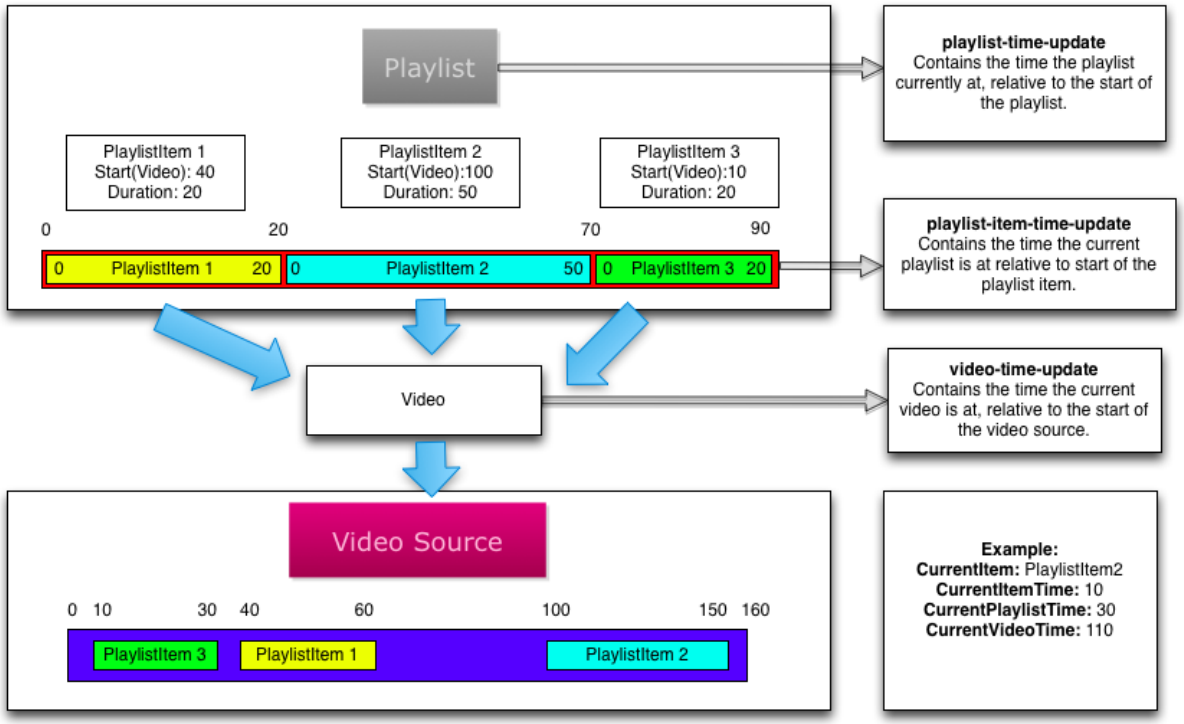


Figure 27. The times triggered by the playlist.

After a PlaylistItem has finished playing, it will trigger the 'item-ended' update. The Playlist listens to the event. When the event is triggered, the Playlist will look if there is a next item available for playout, if so, it will invoke selectItem() again with the next item as an argument. The quality and volume stays the same for the next PlaylistItem and the Presentation just continues playing.

If there is no next item available the Playlist will trigger the 'playlist-ended' event. This will inform the Presentation that the playlist has ended, and that it should inform the views about this. The DialogView will display a 'Play Again' dialog to the user. If the user clicks this, the Playlist once again selects the first item of the collection and starts over.

Scrubbing through the Playlist

The user can drag the handle of the timeslider to a desired position. When the user clicks the handle the slider will trigger the 'onslidestart' event. When user moves the handle an 'onslide' event is triggered periodically. When user releases the handle the 'onslidestop' event is triggered.

While the user is scrubbing to a new position the presentation should be paused. A screenshot should be displayed instead of the presentation, containing an image of the current position the handle is at. This mimics the functionality of fast forwarding through a VHS for example. You can actually see the image of the position you are at. In this way it is clear for the user where he is changing the position to. Figure 28 shows the sequence of scrubbing through a Playlist.

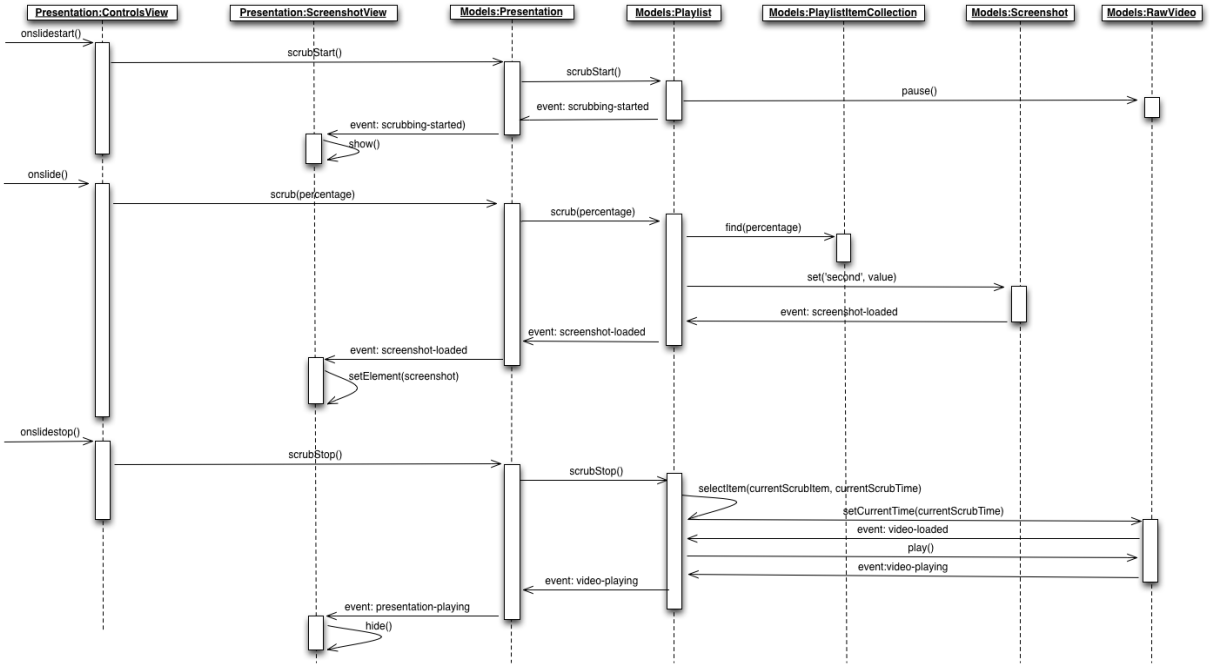


Figure 28. Scrubbing through a playlist.

The user initializes the scrubbing sequence by clicking on the slider handle. The Controls view will then invoke the scrubStart() function in Presentation. The Presentation then invokes the scrubStart function on Playlist which then pauses the current RawVideo. After this it will trigger the 'scrubbing-started' event. ScreenshotView listens to the 'scrubbing-started' event on Presentation. When scrubbing has started it will make itself visible, overlaying the current VideoElement.

When the user actually starts moving the handle about, the scrub() function is invoked on the Presentation with the current percentage of the slider passed as an argument. The Playlist will then figure out which PlaylistItem belongs to which percentage of the Playlist. It will also figure out to what the relative time position of the PlaylistItem should conform to. For example 10% might actually be 15 seconds into the first PlaylistItem. The find() function figures this out. It will set the 'second' property of the Screenshot object to the current video time. The screenshot object will then load the screenshot for the given second, and will trigger the 'screenshot-loaded' event when the screenshot has loaded. The Presentation propegates this event to the views. Every time a screenshot has been loaded, the ScreenshotView will update its internal HTML Image Element.

When the user lets go of the handle the scrubStop() function is invoked on the Presentation. The Presentation calls the scrubStop() function on Playlist. The Playlist will call the selectItem() item function with the current item the user is scrubbing over, along with the relative position of the playlist item. selectItem will invoke the setCurrentTime on the new RawVideo to change the time. Once the RawVideo has loaded enough data it will trigger the 'video-loaded' event. After the 'video-loaded' event has been received by the Playlist, it will call the play() function on the RawVideo. The RawVideo triggers the 'video-playing' event which is propegated to the Presentation. The ScreenshotView listens to the 'video-playing' event and hides itself while the video is playing.

Achieving Quality Selection

The quality of the Presentation has to be able to be changed while playing a video. Because HTML5 lacks the addition of a widely used Streaming platform this could not be done dynamically. So for every video file on the Springfield platform multiple versions are created.

Each version is a different bitrate (or rather quality). They are defined in the quickstart.xml as can be seen in figure 29.

They were already being parsed into objects by the method defined in chapter 6.3. A way had to be found to change the quality (bitrate of current video) of the presentation while a presentation was actually playing.

The sequence diagram in figure 30 shows the sequence of switching a RawVideo while a presentation is playing.

```
<video fullid="/domain/springfieldwebtv/user/david/video/5" id="5">
  <properties/>
  <rawvideo id="1">
    <properties>
      <audiocodec></audiocodec>
      <width>1920</width>
      <audiochannels></audiochannels>
      <metadata_file></metadata_file>
      <videocodec>H264</videocodec>
      <pixelaspect>0.0000</pixelaspect>
      <format>H.264</format>
      <extension>mp4</extension>
      <filesize></filesize>
      <duration>102.54</duration>
      <height>1080</height>
      <samplerate>44100</samplerate>
      <mount>stream11,stream14</mount>
      <videobitrate></videobitrate>
      <audiobitrate></audiobitrate>
      <original></original>
      <framerate></framerate>
      <status></status>
    </properties>
  </rawvideo>
  <rawvideo id="2">
    <!--Contents of second RawVideo here -->
  </rawvideo>
</video>
```

Figure 29. RawVideos in quickstart.xml

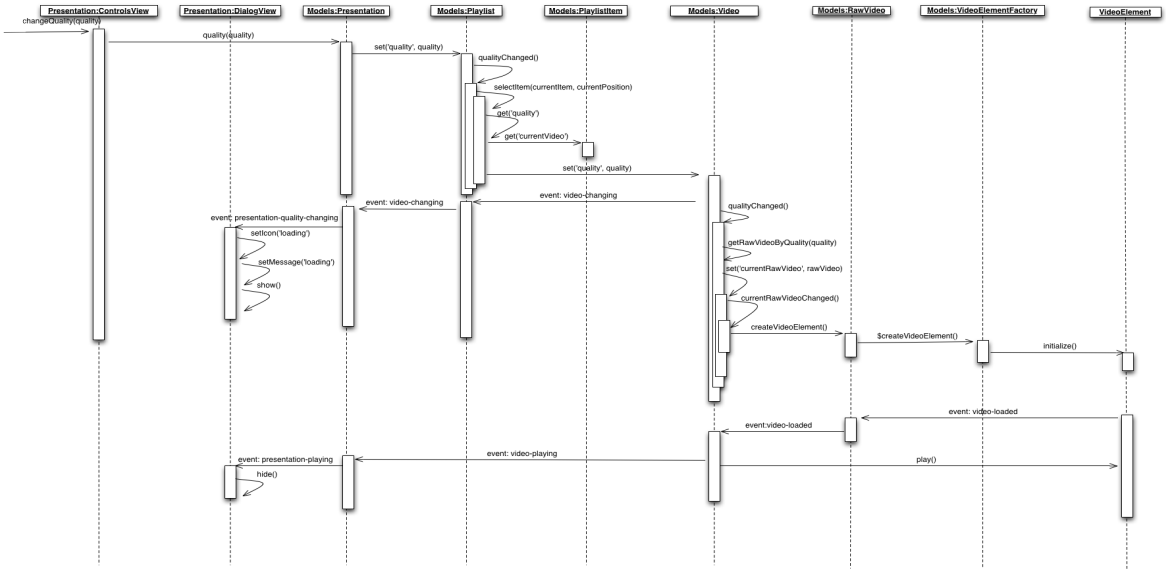


Figure 30. Switching quality.

The user initializes the sequence by changing the quality in the GUI. The ControlsView invokes the quality() function on the Presentation with the selected quality as the argument. The Presentation changes the quality property on the Playlist. The Playlist listens to changes of the property. When the quality changes, the qualityChanged callback is invoked. The qualityChanged function invokes the selectItem() function with the currentItem and the currentTime as arguments, the PlaylistItem and time should remain the same after the switch. It changes the quality of the Video. The Video also listens to changes of the quality property. When it changes it automatically finds out which RawVideo belongs to the quality which was set. The Video unbinds the old RawVideo, so that time updates are not longer received. The Video sets the new RawVideo and invokes the createVideoElement which creates a new HTML5 video element through the VideoElement factory. Once this new VideoElement has loaded enough data it triggers the 'video-loaded' event that is a signal that the video can resume playing from the original position

6. Development

In this chapter a brief description of the development method that was used will be given. After a global description will be given of what was done during each sprint.

6.1 Development Method

In order to develop the application in a structured and progressive manner it was decided chose implement SCRUM. SCRUM is an iterative and incremental development method. This means that several time boxes (or sprints) are defined in which a certain amount of functionality has to be completed. The functionality required is defined in a product backlog. Before each Sprint a Sprint backlog has to be defined consisting of the functionality which has to be completed in that Sprint.

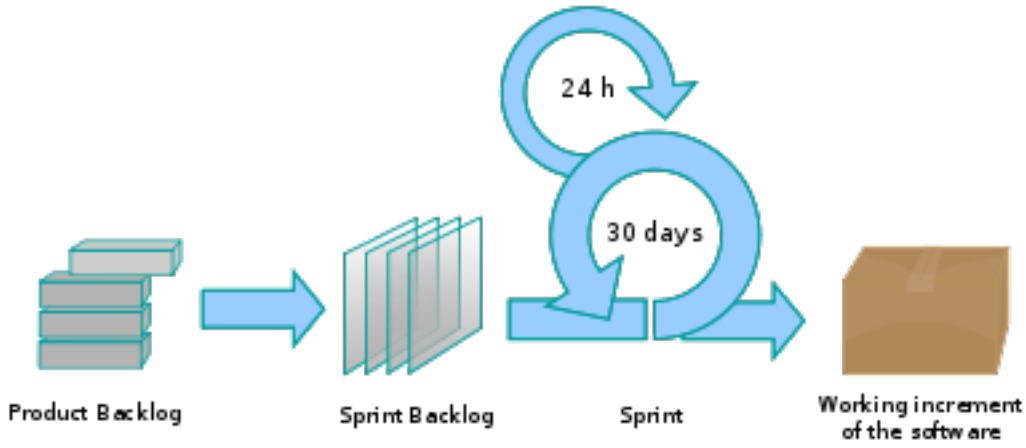


Figure 30. A diagram showing SCRUM in progress. Source: http://upload.wikimedia.org/wikipedia/commons/thumb/5/58/Scrum_process.svg/400px-Scrum_process.svg.png

A product backlog had to be defined containing all the user stories (extracted from functional requirements) and technical stories (extracted from non-functional requirements, and technical requirements). After these had been defined, an importance rating and estimated time until complete had to be attached to each story. I attached the importance rating of each story according to the priority that was defined in the Functional Design (Chapter 4), and according to the reliance of other stories on the specific story. A typical sprint is between 2 and 4 weeks. The sprints were defined as follows:

Sprint 1 (Basic Play out)	27/02/2012 - 16/03/2012
Sprint 2 (Quality, Playlist)	19/03/2012 - 06/04/2012
Sprint 3 (Events)	09/04/2012 - 27/04/2012

At the end of each day I had to evaluate what was done, and what I was going to be doing the next day. In this way there is constantly a deadline to be working to and this helps in the progress, there can be no postponement.

After the end of each Sprint a demo had to be given to Noterik. This demo had to be 'working'. So no static code or mock-ups were allowed.

Of course SCRUM is mostly used in a project environment of three or more members, so not all the facets and roles of the method could be applied. But some of the rules prescribed within SCRUM are still useful within solo development.

Test driven development

In the original Play of Approach it proposed to use Test Driven development. This means that before developing a specific functionality, tests first have to be defined which the developed code will have to complete successfully. In this way the code that is constructed is much more stable and tested before it is implemented.

However, the author had very little experience with this method. Halfway through the first Sprint it was clear that it was delaying the development in such a way that functionality could not be completed before the end of the Sprint. Therefore it was decided to discontinue the test-driven development. Test-driven development could have been very useful though, but in a project with more time available and were the deadlines are not as strict.

7. Motivation choices

7.1 Lightweight implementation of SCRUM

A development method had to be implemented, in order to make sure that progress was made on the development of the product, and to make sure that the final product achieved all the requirements defined in Functional Design.

SCRUM was an ideal candidate, as it requires the developer to divide the functionality into chunks, which can then be developed separately. The Product Backlogs and Sprint Backlogs serve as monitors as to how far the functionality has progressed. The definition of tight time boxes (sprints) makes sure that the developer is always working towards a deadline. It is clear for the developer what has to be done in the limited time available.

SCRUM however, is a method mostly used within a team environment. This means that certain components of the SCRUM method such as the roles and the daily scrums were not implemented into the actual development, as they would just provide too much overhead, as the project was done individually.

7.2 HTML5 + JavaScript

This was the only viable environment into which the application could be developed. A more detailed description as to why can be found in the research that was done prior to the start of the project (Attachment 3).

7.3 RequireJS

A way had to be found to structure the application, without having to litter the main HTML page in which the application is embedded with script includes like this:

```
<script type="javascript/src" src="js/script1.js">
<script type="javascript/src" src="js/script2.js">
<script type="javascript/src" src="js/script3.js">
```

One of the main setbacks of this method is that these scripts might have dependencies to each other. For example script2.js might depend on script1.js. If the scripts were included like this, it would not work. We would have to order the includes like this:

```
<script type="javascript/src" src="js/script2.js">
<script type="javascript/src" src="js/script1.js">
<script type="javascripot/src" src="js/script3.js">
```

This will work. But if the list of includes grows to a larger number such as 20 includes, it can become a big hassle to order all dependencies.

Another big setback of this method is that the scripts are loaded synchronously. This means that the loading of the script will actually block the rendering of the page. For every script a separate request will have to be done. Once the list of includes start to grow this can start to severely influence the load time of the page.

So a way had to be found to asynchronously load dependencies. The dependencies had to defined within each module of code in order to not clutter the main page with includes.

RequireJS had all the functionality that was required. RequireJS provides a method to define modules. The modules contain code that might have dependencies. The dependencies are defined in the decleration of the modules and are loaded asynchronously. For a more detailed description see chapter 6.2. There are other alternatives such as CommonJS. However RequireJS provided the most understandable documentation and syntax (in the humble opinion of the author). RequireJS

also provides a optimization tool which can minify(removing whitespace) and uglify(shorten variable names) code, so that it is interpreted faster by the browsers.

7.4 MVC (Model, View, Controller) pattern

It had to be possible to loosely couple the presentation to the domain and application logic, because the domain and application logic might have to be reused in the Montage Tool, which is to be developed in the future. By separating the functionality, the application becomes more structured. It could of course have been possible to implement all the functionality needed for the application into a single class, but this would have made the structure of the application very chaotic, and very difficult to make quick changes to.

The Model, View and Controller pattern separates the logic as can be seen in figure 31.

Further separation was not needed in the context of this project. As each layer provides an extra layer of complexity, which might influence the performance of the application in a negative way. A very lightweight of the MVC pattern had to be found, in order to keep the application responsive enough.

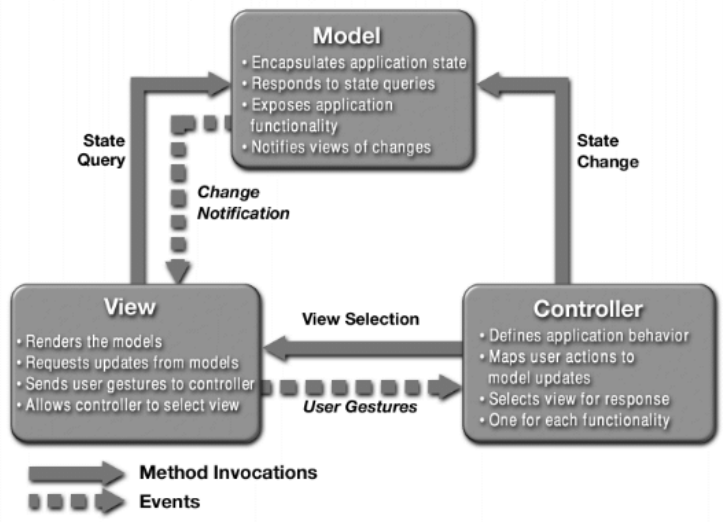


Figure 31. The MVC pattern.

7.5 Backbone

The application had to be build based on the MVC method. Of course it would have been possible to the application from scratch, but this would have proved to take too much time in the context of this project. The quality of this application might have been questionable as well as creating an application based on the MVC pattern can be quite complex. There are however a lot of frameworks available which already provide a tried and tested implementation of the MVC pattern. However in the context of this project it was important that the framework was as lightweight as possible.

Backbone.js is a very lightweight implementation of the MVC pattern. The developer already had experience in programming in Backbone. Because of the limited time available for this project, it was chosen not to look at too many alternatives, as the research and learning of the new framework would have taken too much time.

7.6 Not implementing events

After seeing the capabilities of the HTML5 player during the demo of sprint 2, Noterik came to the conclusion that events should be implemented in completely different way. Instead of the events only being active within the context of KrustyJS, other components should also be capable of listening to the events triggered by the application. In this way it would be possible to change the state of elements also contained by the same page. This however adds a big layer of complexion to the existing code. And this was something that could not be implemented anymore in the limited time that was available. Therefore it was decided that the implementation of the events would be postponed until a later period. This however cancelled the entire purpose of Sprint 3. Instead sprint 3 was used to optimize the existing application.

7.6 Not implementing fullscreen mode

In the original Plan of Approach it was defined that fullscreen support should also be implemented. However after my research it became clear the support of a fullscreen mode still might prove very challenging in order to implement it to all the browsers. Currently a lot of browsers do not support it. Also the fullscreen mode was defined as a 'COULD-HAVE' in the functional design. Therefor it was decided not to be implemented in the current version of KrustyJS.

8. Conclusion and recommendations

8.1 HTML5

It is clear that HTML5 is a serious contender to Flash, certainly in the context of this project. By implementing the API defined within the HTML5 video element almost all the functionality described in the Functional Design could be implemented. Some things such as the Event handling and the full screen support proved to be possible, but not implementable within the time that was defined within the Plan of Approach.

Almost any modern browser, including IOS, supports the HTML5 video element. This means that by using the HTML5 video element API a developer is able to reach a very wide spectrum of users, including the group that could not be reached by Flash. The need for installing a third party plugin in order to play back videos (Flash) is also removed. All the functionality is embedded within the browser.

There are however some things to keep in mind when implementing HTML5 into an online video service platform.

There still is no single codec that is supported by every major browser. The so-called "Codec War" is in still progress, and though H.264 seems the likely winner there is no certainty. Both H.264 and WebM support will have to be implemented in order to reach the widest audience.

Another issue is the lack of a widely supported streaming platform such as Flash currently has. This makes it impossible to implement adaptive streaming, and requires users to change the quality of their videos manually. This is a double-edged blade, because progressive download does not dynamically change the bitrate of the video. Instead, several versions have to be made of the same video in different bitrates.

The lack of a streaming platform also has another major setback. By using progressive download, the sources of the video are retrievable for anyone with some technical knowledge. This means that security is a big issue at this moment.

However HTML5, as said before, is still very much a technology in development, and it likely that these problems will be resolved in the near future.

8.2 Recommendations

In order to reach the widest audience possible Noterik and to keep up with competition, Noterik should try to implement the HTML5 alongside Flash in most of its existing front-end systems.

It means that in all existing projects that aim for a large target group, such as EUScreen and LinkedTV, the playback should be changed to HTML5 (if security is not a priority).

It also means that in the near future the Montage Tool should also be transferred to a HTML5 environment. This project will prove to be a lot more challenging, as the Montage Tool implements a lot more functionality then Krusty.

Noterik should also keep their eyes open for changes in the security management in HTML5. As not all of their clients like their files to be retrievable from the Web.

The MediaFragments standard, currently being developed by the W3C, also implements a lot of technology that might be interesting for Noterik. Therefore Noterik should remain actively involved in the development of this standard.

9. List of Sources

9.1 Literature

Pilgrim, Mark. *HTML5: Up and Running*, Sebastopol: O'Reilly Media, 2010

Flanagan, David. *JavaScript: The Definitive Guide, 6th edition*, Sebastopol: O'Reilly Media, 2011

9.2 Websites

HTML5

Definition	http://en.wikipedia.org/wiki/HTML5
Usage	http://w3schools.com/html5/html5_video.asp
Events	http://w3.org/2010/05/video/mediaevents.html

MediaFragments

Proposal	http://w3.org/TR/media-frags/

Javascript

JavaScript:	http://en.wikipedia.org/wiki/Javascript
JQuery	http://jquery.org
Backbone	http://backbonejs.org
RequireJS	http://requirejs.org
Jath	http://github.com/dnewcome/jath

Attachment 1

Plan of Approach

David Ammeraal
Studentnumber: 1520387,
Noterik B.V.,
University of Applied Sciences Utrecht, Internal
Supervisor: Konstantin Radoslavov, External
Supervisor: Michiel Borkent,
david.ammeraal@student.hu.nl

April 23, 2012

Contents

Chapter 1

Introduction

The past five and a half years I have been been studying informatics at the University of Applied Sciences in Utrecht. After some delays and other diversions the time has finally come to graduate. After having been informed as to how the graduation project should be performed I immediately set out looking for a project. After approaching several companies I had quite a selection of possible projects. An acquaintance advised me to contact Noterik B.V. as well. After a short correspondence by email, Rutger Rozendaal invited me for a meeting to see if we could find a project of which both parties could profit. We found one quickly.

By the end of the summer of 2011 I had finished most subjects of my study and already had most of my ECT's. I was quite quick in finding a project so I started with my assignment earlier. I've been working on my assignment from the 16th of January 2012.

In this document I will elaborate the context in which the assignment will be performed. Afterwards I will determine the goals and products which have to be delivered after having completed this project. I will also elaborate as to how I am going to deliver these goals and products.

Chapter 2

Context in which the project will be performed

2.1 Noterik B.V.

Noterik is a small company with currently eight employees. The office is situated in the centre of Amsterdam on the Prins Hendrikkade 120. Noterik was established in 1996 and focusses mainly on delivering online video services. Noterik's target clients are mainly in the non-profit sector. The European Union and the municipality of Amersfoort are examples of the sort of customers Noterik delivers its services to. An organization chart of Noterik can be found in figure 2.1.

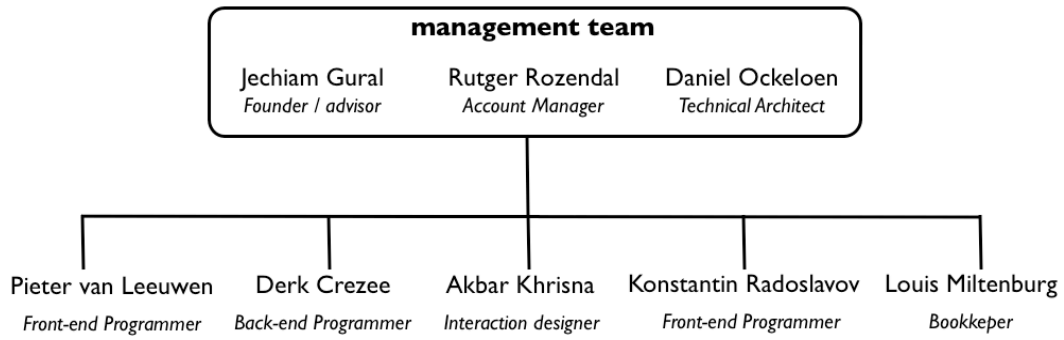


Figure 2.1: An organization chart of Noterik

Because Noterik is such a small company there are no real divisions within the company, however there are different functions. The top tier consists of the management team, this tier consists of the three managers. Jechiam Gural is the founder of company and advices the other managers. Rutger Rozendaal is the account manager and also does a

lot of project management. Daniel Ockeloen is the technical architect of the company and is responsible for the architecture of the software of Noterik. The bottom tier consists of the developers and bookkeeping. Every developer has their specific speciality within the springfield platform such as front-end programming, interaction design and back-end programming. In reality everyone has to be able to understand the basic functionality of every tier of the platform. Louis Miltenburg is responsible for the bookkeeping of Noterik, and answers to the management team.

2.2 Services of Noterik

The video services delivered by Noterik have more functionality than just being capable of watching a video stream. Events can be assigned to periods of time in the timeline of a video. Also users can select certain regions in the canvas of the video and attach extra information to it. In this way users can add extra information to a certain point in time or space of the video. For example: A user can indicate who is currently speaking in a video concerning a meeting of council members of a city. The next time a user watches this video it will be be displayed in the player and it will be clear who is speaking and what his/her function is. This makes a video much more interactive and information concerning the current subject of a video can be easily shown to the user watching the video. Noterik therefore prefers to call the combination of the video stream and event functionalities not a video but rather a presentation.

2.3 Springfield

To accommodate the video services Noterik has developed the Springfield platform. Springfield is a distributed system consisting of several tiers of systems. These tiers can be seen in figure 2.2

At the center of the system is the Services Tier. Each Service is responsible for a separate functionality. The services can be reached by REST by sending a request to HOMER. In the figure Homer is specified as a DNS, this is because HOMER maps the location of the other services. For a more detailed explanation of the REST principle see the article by Michael Jakl[1].

The services communicate with each other through HOMER to get the data which is needed. All these services are hosted on a cluster of Windows servers in the Hardware tier.

Users can communicate with the Services tier either with a browser which sends REST requests to HOMER or with a Flash Client which is called Krusty to communicate and watch the video's stored inside the system. Every request done by Krusty is done in REST.

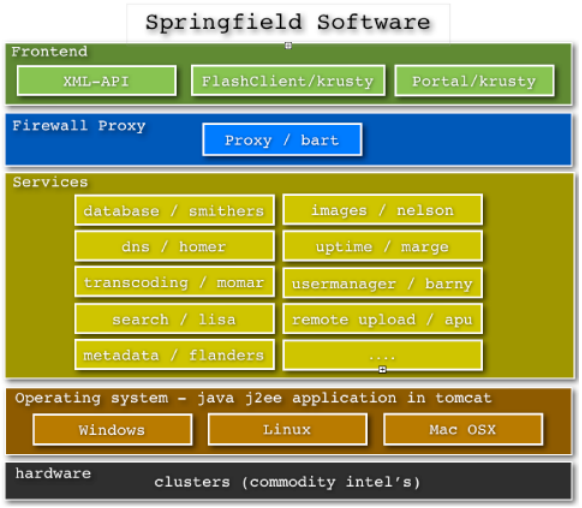


Figure 2.2: A graphical representation of the functionality of Springfield platforms divided by tiers

2.4 LinkedTV

LinkedTV is an initiative by Fraunhofer IAIS. Noterik is an active participant in the development of this system. LinkedTV aims to interconnect video and other media on the Web, and make it watchable on a wide spectrum of systems. A more detailed explanation can be found on LinkedTV's website: http://www.linkedtv.eu

Figure 2.3: An example showing which sort of information should be possible to be tagged and interconnected with LinkedTV.

Networked Media will be a central element of the Next Generation Internet. Online multimedia content is rapidly increasing in scale and ubiquity, yet today it remains largely still unstructured and unconnected from related media of other forms or from other sources.

This cannot be clearer than in the current state of the Digital TV market. The full promise and potential of Web and TV convergence is not reflected in offerings which place the viewer into an Internet closed garden, or expect PC-like browsing on a full screen Web, or offer interesting new functionalities which however lack any relation to the current TV programme.

Our vision of future Television Linked To The Web (LinkedTV) is of a ubiquitously online cloud of Networked Audio-Visual Content decoupled from place, device or source. Accessing audio-visual programming will be TV regardless whether it is seen on a TV set, smartphone, tablet or personal computing device, regardless of whether it is coming from a traditional or new media broadcaster, a Web video portal or a user-sourced media platform.

Television existing in the same ecosystem as the Web means that television content and Web content should and can be seamlessly connected, and browsing TV and Web content should be so smooth and interrelated that in the end even surfing the Web or watching TV will become as meaningless a distinction as whether the film is coming live from your local broadcaster, as VOD from another broadcaster, or from an online video streaming service like Netflix. [2]

Springfield will be used for the storage and showing of the video's. The player which I will construct will be implemented into the context of this system.

Chapter 3

Definition of the problem

Noterik currently delivers its video services to its clients with a custom build video player based on Adobe Flash technology. More functionality is continuously added to the player to keep up with the growing demands of customers. Adobe Flash however is a technology which is gradually being used less and less. Adobe has recently announced that it will not develop further updates for the Adobe Flash platform for mobile devices. Devices based on the iOS operating system by Apple (such as the iPad, iPod and iPhone) do not support Flash. Users of these devices are a potential target group for Noteriks' video services and at the moment they can't be reached, and this group is growing quite rapidly. Therefore the functionality currently encapsulated in this player should be transferred to a HTML5/Javascript environment. I did research for Noterik to establish to which environment we should transfer the functionality. The results of this research can be seen in Attachment 1.

3.1 Desired outcome

The functionality currently contained by the Krusty video player should be transferred to a HTML5/Javascript environment. It should be done in such a way that clients have full functionality of the player in a browser (such as Internet Explorer or Firefox), or with a mobile device such as the iOS or Android phones.

Chapter 4

Goals of the project

This project will serve as my graduation project. After completing this project I will have shown the University of Applied Sciences Utrecht and Noterik that I have all competences needed to attain a Bachelor's degree in ICT.

4.1 Research

The research to which environment the Krusty player should be transferred has already been completed, and can be seen in Attachment 1.

I have been assigned by Noterik to research how the functionality of the Krusty player can be transferred to an alternative environment.

4.2 Player

After publishing the results of this research to Noterik I should commence creating a prototype of a player which can be used on a desktop computer running the Firefox version 9 internet browser and iPad running the latest Safari browser on iOS5.

This player will serve as a potential springboard to more advanced implementations customized for HTML5, iOS and Android.

4.3 Thesis and presentation

A thesis will be written about the entire project. After all the products have been delivered I will present them and the thesis to Noterik and the University of Applied Sciences Utrecht during a final meeting at the University of Applied Sciences Utrecht.

Chapter 5

Scope of the project

Transferring all of the existing functionality of the Krusty video player to HTML5/Javascript would be too much to be able to do for one person in the limited time available. Therefore Noterik has made a selection of the functionality which at least has to be implemented in the prototype. I will list them here. These are just global descriptions, in the functional design I will go into deeper detail. In the functional design I will also specify a priority to these functionalities according to the MoSCoW method.

During the research, it might become clear that more functionality will have to be added, the current planning however allows for some extra functionalities to be added if needed.

5.1 Basic playout

Basic playout means being able to play, pause and stop a movie stored in the Springfield platform. Scrubbing (navigating through the timeline of a video), volume control and muting should also be available.

5.2 Video quality and aspect ratio selection

It should be possible to select a different resolutions for a video. For example HD(1080p) or SD(480p) for a given video. Also different aspect ratio's should be selectable such 16:9 and 4:3.

5.3 Dock functionality

Users should be capable of seeing the dock which gives the following functionality:

View info of the video
: Users should be able to view basic info of the video such as the title, description, and author details.

Share video with others
: Video's should be able to be shared with friends through media like Facebook and Twitter.

Log in to the system
: Users should be able to log in to the Springfield platform using their credentials.

Tag events in the video
: It should be possible to create events in the video and attach them to a certain moment in the timeline of a video.

5.4 Showing events

It should be possible to see events which have been added by other users. They should be visible in a layer above the video.

5.5 Playlist support

It should be possible to add the video to a video playlist. The video's in this playlist should be able to be played in sequence after each other.

Chapter 6

Delivering the products

Several products will have to be delivered. Some for Noterik and some for the University of Applied Sciences.

6.1 Plan of approach

This is the document you are reading now. In this I will explain how I am planning to do the project.

6.2 Researching how the functionality can be transferred

6.2.1 Research planning

In this document I will establish a research question, and how I'm planning to do my research. I will define a research question according to the problem that needs to be solved. I will split the research questions into subquestion and I will try to answer each of these question separately to come a answer for the main question.

6.2.2 Research results

The results will consist of several prototypes and documents describing how the functionality of the Krusty player can be transferred to a HTML5 environment. The prototypes will be actual pieces of software which demonstrate a piece of functionality.

6.3 Designing the new player

6.3.1 Functional design

In this document I will list all the functional and non-functional requirements. I will show the functional requirements in the form of use-cases. Also the interaction with the system from the viewpoint of the user will be described. I will attach a priority to each use-case according to the MoSCoW principle. These priorities will be based on how important a certain part of functionally is for Noterik. For example: Basic Playout is of critical importance, because of this, it will be a must-have functionality because other parts of the application rely on this functionality to be operational. The priority rating I attach to a functionality will also be the main influence to the importance rating for that piece of functionality in the Product Backlog.

6.3.2 Technical design

In this document I will elaborate how all the functional and technical requirements should be implemented. It will include a UML class-diagram, and how the application should communicate with the services of the Springfield platform. Also certain things such as the operational context and required software will also be specified. Certain tasks will have to be performed for the application to work (such as making a test environment, constructing stylesheets etc), these tasks will also be added to Product Backlog as a technical task.

6.4 Constructing the new player

I have divided the development time of the application into three sprints. In this way I can cut the project into small pieces which I can work on one at a time.

6.4.1 Product backlog

According to SCRUM this document contains all the functionalities described in the Functional Design with an importance rating. Some tasks In this document I will put all the functional requirements of the application. I will attach an importance rating to every requirement, and I will estimate the time it will take me to build the functional requirement. After each sprint I will show the progress of each functional requirement in this backlog.

This document will also contain technical task extracted from the technical design. I will attach an importance to each technical task. Some tasks will have a very high importance (such as making a test environment), but some will have a lower importance (such as making stylesheets) according to the reliance of the system on these technical tasks.

6.4.2 Sprint backlog

Every sprint I will make a sprint backlog. I will divide the functional requirements selected for creation in the specific spring into smaller subtasks. I will attach a new importance rating and estimated time to every subtask. I will attach a new importance rating according to what is described in the functional design, and how much the parent task relies on this subtask to be operational. After every day of work I will add how far the completion of each task has progressed. The time-estimate will be defined according to how productive I can be in the days to come and how fast I have completed similar tasks in the past.

6.4.3 Prototype

After every sprint I will show a prototype to Noterik. In this way they can see how I am progressing. The prototype has to be a 'working' prototype. That means: it has to be able to do something, like run a video, play a video etc. Just code is not allowed.

6.4.4 Test Document

I will document every test I write for the current functionality I'm working on together with outcome of these tests in this document. Before delivering the final prototype I will conduct tests to check if all the functionality described in the Functional Design works as it should. I will also document the outcome of these tests in the test document.

6.4.5 Final Prototype

After the final sprint the prototype should have all the functionality described in the functional design. I will present this prototype to Noterik and to the University of Applied Sciences Utrecht.

6.4.6 Technical Documentation

In this document I will describe the technical details of the implementation. Things such as system requirements, a final class diagram etc.

Chapter 7

Planning

I have made a planning for the entire duration of the project. I have placed most products to be delivered into this planning. The final Plan of Approach however I have not been able to add because it depends on how fast the University of Applied Sciences Utrecht will take to check it.

7.1 Detailed Planning

January 16 - May 25: Thesis
During the entire duration of the project I will work on my Thesis. Every week I will spend the half of Friday to work on my thesis.

January 16 - January 27: Phase 1 - Orientation

January 16 - January 27: Orientation
During this period I will focus on learning the workflow and culture of Noterik.

January 17 - January 20: Create Concept PoA

January 23 - January 27: Create Research Plan

January 30 - February 10: Phase 2 - Research
During this period I will research how the functionality currently contained within the Krusty player could be transferred to a HTML5 environment. At the end of the phase I will present my results.

February 13 - February 27: Phase 3 - Design

February 13 - February 16: Create Functional Design

February 17 - February 23: Create Technical Design

February 24: Create Product Backlog

February 27 - April 27: Phase 4 - Construction

February 27 - March 16: Sprint 1

February 27: Make Sprint Plan

February 28 - March 16: Develop Basic Playout

February 16: Show prototype with Basic Playout

March 19 - April 6: Sprint 2

March 19: Make Iteration Planning

March 20 - March 31: Develop Displaying of Events

April 2 - April 6 : Develop Adding of Events

April 6: Show prototype in which you can show and add events.

April 9 - April 27: Sprint 3

April 9: Make Sprint Planning

April 10 - April 12: Develop Video Quality Selection

April 13 - April 18: Develop Aspect Ratio Selection

April 19 - April 27: Develop Playlist Support

April 28: Show final Prototype

April 28 - May 6: Vacation

May 7 - May 25: Phase 5 - Evaluation and fine-tuning

May 7 - May 11: Finetuning
During this time I will fix things that are still not functioning as they should.

May 14 - May 19: Make Technical Documentation

May 21 - May 25: Make presentation

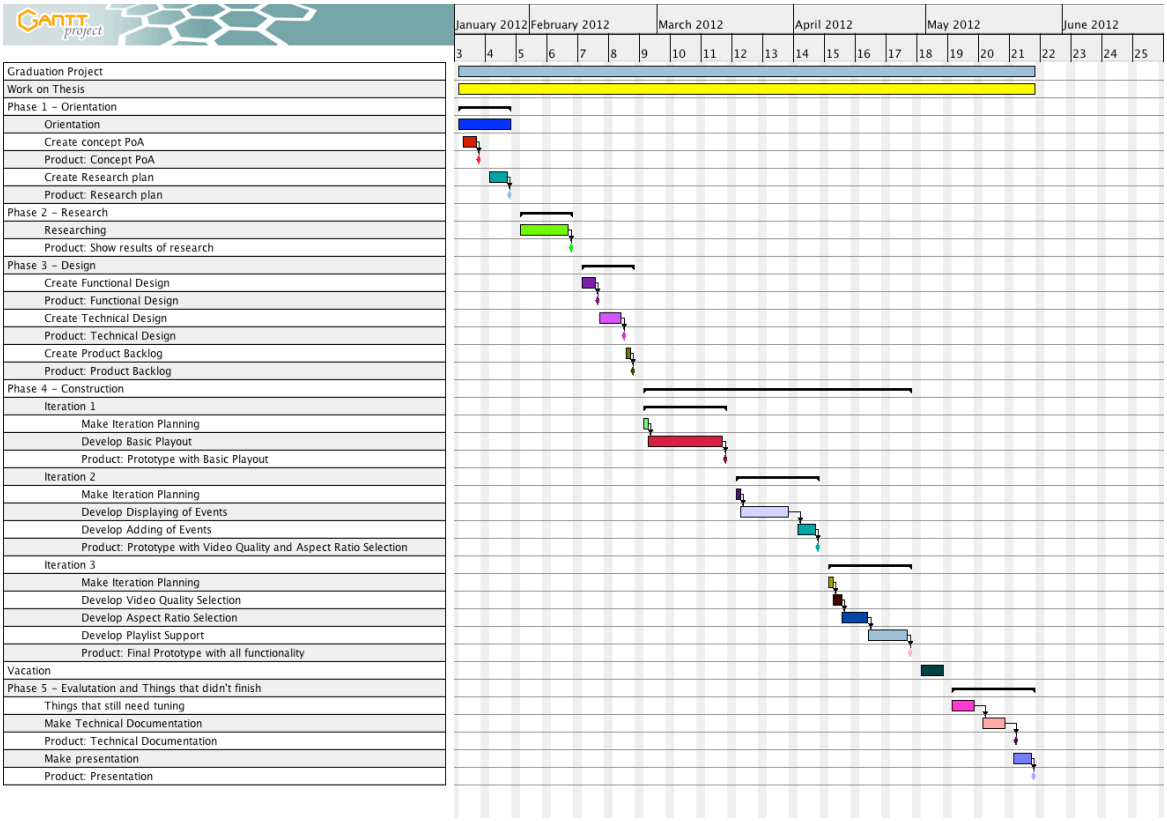


Figure 7.1: A gantt chart of the project planning

Chapter 8

Used Methods

To make sure everything in the project is done in a structured and controlled manner, I will have to apply some methods. In this chapter I will explain some of the methods I will be using during the project.

8.1 Research

I will first establish a concrete research question. After this I will split the research question into subquestions. After this I will set out to try and answer every subquestion. After every subquestion has been answered I should also be able to answer the main research question.

8.2 Project Management

To make sure I bring the project to a successful end, I will use some parts of the SCRUM methodology. SCRUM is about developing in a team, however I will be working on my own. Some things however are still good to use.

I will maintain a product backlog to be able to keep track of the progress of the project. I will also keep strict deadlines on my sprints to make sure that I have a target to work to every time. After every working day I will update the Sprint backlog with the progress I have made on every task. And every morning I will establish what I will have to do in the day to come.

After every sprint I will have to show something that 'works'. This way Noterik also has a good idea of how I am progressing in my work. I will however not be able to have a product owner so I will not be able to implement SCRUM to the fullest. However, I think if I follow these practices I will be able to deliver all the products needed, and the project will have a lot more structure.

8.3 Application Development

I will try to implement Test Driven development. This will mean I will establish unit tests for a certain task in the spring backlog. And then create code that will validate for those tests. This way I'm sure that the code I'm writing will contain the least possible faults, and that extra functionality build upon an existing tested functionality will have a solid base to rely on. Test Driven development however has not been a subject in my curriculum, so I will have to learn this. I will learn this by reading tutorials and articles on the web. If it becomes clear that it takes too much time to write the tests, I might choose to not use it for the rest of the project.

Code will be committed to a code.google.com while I'm developing it locally. After the code qualifies to the tests I will submit it to the cvs server of Noterik. After that it will be committed to the development environment, where it will be tested again. After everything has been tested it will be submitted to the production environment. It is not clear yet who will write these tests as there is no certainty how the new environment can be tested.

Chapter 9

Risk Management

In this chapter I will elaborate the risks that are possible in this project and how I'm planning on handling these risks.

9.1 Defects of systems

I will always maintain a backup of my documents on docs.google.com, code will be committed to a code.google.com as well as to Noterik CVS server. This way I'll always have an up-to-date backup of all my files.

9.2 Certain functionality of the Flash environment cannot be converted to the new player

I'll research all the functionality needed before actually building the player. In the research phase it should be made clear which functionalities will work and which will not.

I'll make prototypes for the functionalities which I think might be hardest to implement, to be able to work around eventual impossibilities.

If however during the sprint it seems that certain functionality cannot be implemented alternatives should be looked for. However these alternatives will fall outside of the scope of the sprint currently being working on, because new time estimates will have to be established for the functionality. In case this happens I will add tasks planned for a later sprint to the current spring and I will start researching alternatives in time that is not being used during the sprint.

9.3 Not all the tasks scheduled for a sprint are finished within the time box of the sprint

I will keep a few hours free in every sprint scheduled for working on things that didn't finish last sprint. However I will continue working on the tasks that are planned for the current sprint.

9.4 Illness

I might have to stop working for medical reasons somewhere during the project. In case this happens I will have to make changes to the planning. Maybe I will have to omit some low priority tasks in the sprint. Also the vacation planned in may will only be used if all products planned to be delivered are still on schedule. I have also included a slack period during phase 5 of about a week. In this period there is time to fine-tune and finish of some tasks that didn't finish within the sprints.

9.5 Supervisor is not available

My supervisor is in Bulgaria most of the time. So he might not be able to attend all the meetings. However channels such as Skype and e-mail should be used as much as possible to maintain the communication between me and my supervisor and the teacher from school.

Chapter 10

Supervisor

At first it seemed that Rutger Rozendaal would be my supervisor. But Rutger is the project manager and account manager and can't really help me when I need help with technical problems. Konstantin Radoslavov however had already done some research on the matter of transferring the functionality of the Krusty player to a new environment and thus it was decided that he would be appointed as my supervisor.

Konstantin mainly works from Bulgaria, but sometimes he is also in the Netherlands. When he's not here, we communicate through Skype and E-Mail.

Konstantin has a Bachelors Degree of Science in Engineering (Electronic) attained at the University of Natal, Durban, South Africa, and also has a Masters Degree of Science in Computing attained at the Griffith College Dublin in Ireland.

Chapter 11

Contact Details

Company
Noterik B.V.
Prins Hendrikkade 120
1001MD Amsterdam
T: 020 - 592 9966
info@noterik.nl

Supervisor
Radoslavov, Konstantin
Developer
k.radoslavov@noterik.nl

Unitversity of Applied Sciences Utrecht
Nijenoord 1,
3552AS Utrecht
T: 088 481 8283
info@hu.nl

University of Applied Sciences Supervisor
Borkent, Michiel T:
088 481 82 83
michiel.borkent@hu.nl

Student
Ammeraal, David
Middenweg 72
1394AL Nederhorst den Berg
T: 06 - 52 57 36 50

Bibliography

[1] Jakl, Michael. Representational State Transfer. http://blog.interlinked.org/static/files/rest.pdf 25 February 2005 Web. 18 January 2012.

[2] About the project. http://www.linkedtv.eu/about-the-project/description/ Web. 19 January 2012.

Attachment 2

Personal evaluation

Student: David Ammeraal
Project: Krusty-JS
Company: Noterik B.V.
Date: 3-06-2012

Introduction

In order to graduate, I had to do a graduation project. This was done in the period between January 2012 and June 2012. This document serves as a personal evaluation of the time that was spent at Noterik. I will evaluate each phase of the project separately. The definition of the phases of the project can be found in the Plan of Approach (Attachment 1, if this document is attached to the thesis). Finally a conclusion will be given regarding the entire period.

Phase 1: Creating the Plan of Approach

I started my time at Noterik on the 16th of January 2012. The usual start date for a graduation project was between the first week of Februrari 2012 until the last week of May, however I had almost finished all my subjects and was able to start earlier. I used the first week of my project to used to the new environment, this was needed because some of Noteriks systems can be quite complex. During this week I also got introduced to everyone in the office. Rutger Rozendaal and Daniel Ockeloen instructed me with a more detailed description of what Noterik wanted from my project. Out of these descriptions I started work on a first concept of the Plan of Approach in the second week of the project. At the end of the first week I had finished my first Plan of Approach. I am quite pleased about the first concept of the Plan of Approach. I thought I had everything planned out quite well. This, however, seemed to be a bit different towards the end of the project.

Phase 2: Research

The second phase of the project went quite well. I constructed a planning of the research in the first two days. In this planning I defined a main research question, and divided this research question into smaller sub questions. During the research each of these questions were answered. Everything went according to planning and I was quite please with the outcome.

Phase 3: Design

After the research was done, it was time to create Functional Design. I started out creating a concept of the Functional Design. It turned out however, that I misinterpreted some of the requirements, such as the playlist support and the dock support. I already sent a version of this Functional Design to my teacher before Noterik actually approved it. So this went a bit wrong. However I edited the Functional Design (which is added to the thesis, as Attachment 4). After this it was complete and correct. These corrections however took a significant amount of time reserved for the phase. In the future, extra days should be planned into a phase for editing and reviewing.

Because of this delay, the technical design could not be constructed during the design phase. So this had to be constructed during the Construction Phase.

Phase 4: Construction

I divided the construction of the KrustyJS player into three increments. I decided to use several parts of SCRUM in order to assure progress. I started out by making a Product backlog. This however proved challenging, as the Technical Design was not finished yet. Instead I decided to

abstract a lot of the technical stories. In this way I could just say: "Construct build environment", instead of "Make Ant build-file". This worked quite well, but because these technical stories were so abstract, these stories had to planned in with a longer duration. Which made the time available during the sprints that much tighter.

After this I started out making a plan for the second Sprint. I extracted user and technical stories from the Product backlog that were relevant to the first sprint, and attached an estimated duration and importance rating to each.

Sprint 1: Basic Play out

I started this sprint out by deciding which IDE I was going to use. This was done quite quickly. I am quite happy with the Aptana development environment for Eclipse and would advice any JavaScript developer to consider using it.

After this it was time to start constructing the player. Everything went quite well, but some things proved to be rather challenging, things such as codec selection for a given browser etc. But I managed to get everything finished before the end of the Sprint. Because this was my first time using SCRUM, I misjudged some estimated durations. But by working extra, I got everything finished on time.

I had to, however, get rid of the test driven development. I feel that it could have been a great addition to SCRUM, and the quality of the final product might have been higher. But it delayed the progress of the first sprint in such a way, that I decided it should not be used anymore.

At the end of the sprint I showed a demo of the player to my colleagues, showing simple HTML5 play out of a video stored in the Noterik Video platform. They were quite pleased with the results, and saw perspective in it.

Sprint 2: Implementing the Playlist functionality

This sprint went quite well. The technical design was finished in the first sprint, so I did not have to abstract the technical stories as much anymore. Also I was more used to the development environment (Javascript, Backbone, RequireJS). Everything went according to plan.

At the end of the Sprint 2, I gave a demo to my colleagues. The Playlist functionality still had some bugs here and there though, that had to be fixed. I did this in the next Sprint. I think that in the future I will add some time for testing at the end of every sprint, so that embarrassing bugs do not show during the demo.

Sprint 3: Implementing Events

I started this Sprint out by making a sprint planning. I was already getting quite used to SCRUM. And I had a good estimation of how each story should be judged. I began the sprint. However at the start of the Sprint, it became clear the events should be implemented in a completely different way. To my disappointment I could not continue implementing the Events as they were defined in my Functional Design. The events had to be completely redesigned, and this had to be done in such a way that they could not have been in implemented in my sprint. Therefore I decided to use the remaining time of the Sprint to optimize the code that was already constructed.

Phase 5: Optimization and thesis

Phase 5 was set aside for optimization of the remaining code, and for finishing my thesis. During this time, I had time to further optimize my code and I got time to implement Progressive Download into KrustyJS. Aside from this a lot of time was used for writing my thesis. I already had made a very global definition of my thesis, but it was not anywhere near finished yet. I finished it during this time.

Conclusions

I consider this project a success. I am quite happy with the way my Plan of Approach turned out. Even though the phases where a bit too short for my taste, I do not think the planning could have been done in a different way.

I am very happy with the results of the research phase. During the research I got a feel of the HTML5 Video Element. The wisdom I gained during this period helped a lot during the construction phase of the product.

I am moderately satisfied with the outcome of the design phase. I think a little more time should have been spent reviewing and editing the functional design, because I could not get the technical design finished before the functional design was approved.

I am satisfied about the way the construction phase turned out. Sprint 1 and 2 went according to plan. And I had the feeling my colleagues were pleased with the results I showed them in the demo's. Sadly the events could not be implemented. But this was not something that I could have predicted.

Overall I am very happy with the outcome of the project, and I feel that Noterik can use the outcomes of this project for more advanced implementations of HTML5.

Attachment 3

Selection of new environment for Krusty functionality

27-02-2012
David Ammeraal
Noterik B.V.
University of Applied Sciences Utrecht
david.ammeraal@student.hu.nl

Abstract

Krusty is Noteriks Flash application with which it displays it presentations to its end-users. Up until recently the Flash environment has been able to provide all the functionality needed. But with rise of IOS by Apple, the market share for Flash on mobile devices has been declining. This is because IOS does not support Flash. Adobe has announced it will discontinue the further development of Flash updates for mobile devices.

To keep up with the demand of end-users, the application will have to be transferred to an environment with which Noterik can reach all its target groups. This environment will have to be able to run on IOS and Android, and be able to implement all the functionality currently encapsulated within the Krusty player.

After comparing all alternatives it has become clear that HTML5 and Javascript should be the environment to which the functionality should be transferred. There are however still risks such as codec and stream incompatibilities between browsers, as well as security issues. All these risks should be held in mind when developing the application in this new environment.

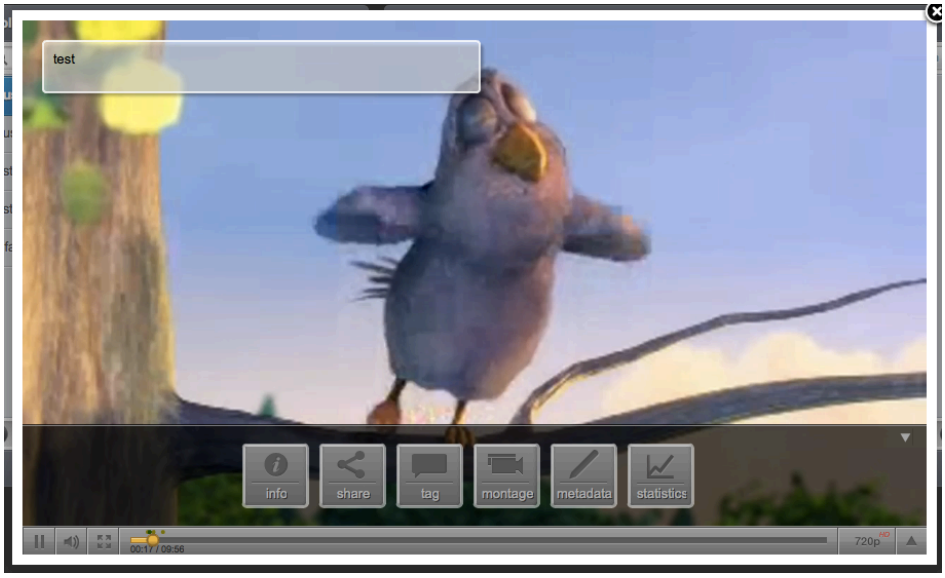
1. The krusty player

Krusty is part of the presentation tier of the Springfield platform, for a more detailed description of the architecture of the Springfield platform see chapter 2.1 of the Springfield Technical documentation.

Krusty encapsulates most of the the presentation logic for the end user of Noterik. It is a Flash application which can display the presentations contained in the Springfield platform. It connects to the Services tier of the Springfield platform through REST. It plays all the video's in the playlist of the presentations and shows events (such as information regarding the current scene, or peercomments) at the right time.

Aside from displaying the presentations for the users, it also allows them to add events at certain timestamps in the video, cut the video into segments, shorten video's and add video's to a presentation.

Apart from all this, new functionality is constantly being added to the application to keep up with demands of the users.

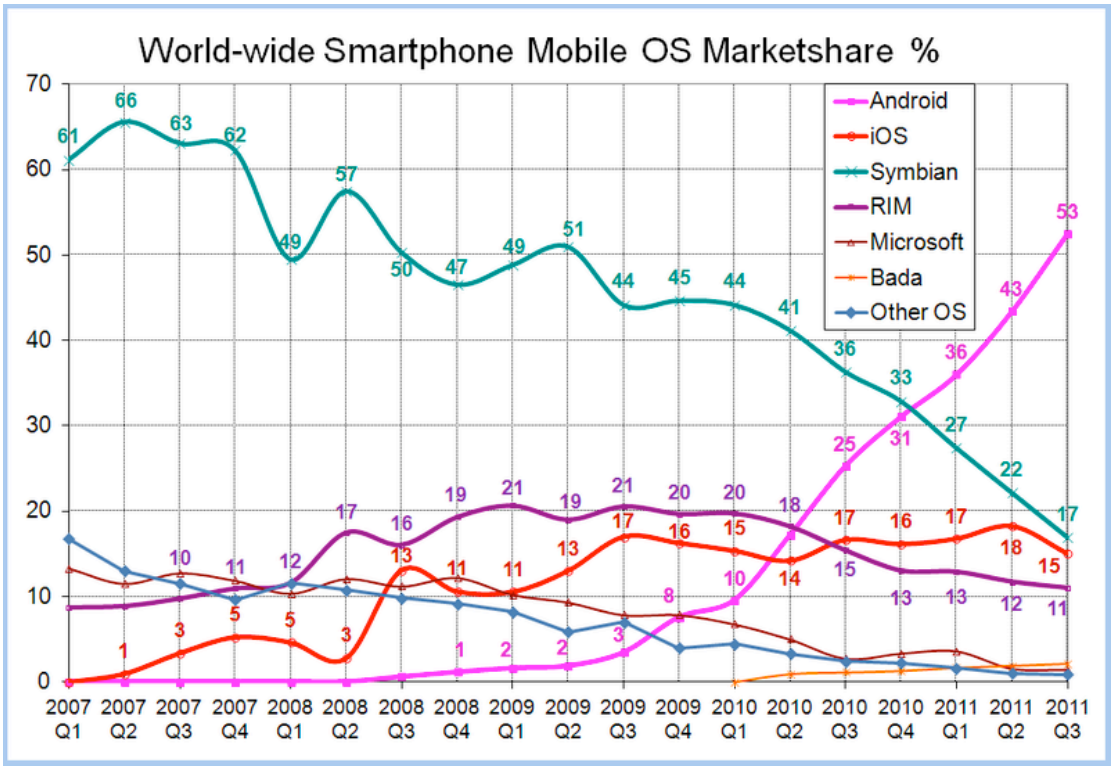


Krusty playing a presentation

2. Reason for migration

Krusty is based on Adobe Flash/ActionScript. Up until recently Flash has fulfilled all the demands of the end-users. Flash runs in the context of the Flash Player which provides a uniform runtime environment across most platforms, thus making development very efficient as code only has to be written once to reach a very wide spectrum of platforms.

In january 2007 Apple introduced the iPhone. This is a smartphone which runs on their proprietary operation system IOS. Over the years Apple has introduced more devices which run on IOS, such as the iPad, iPod and Apple TV. At the end of fourth quarter of 2011 IOS had a market share of 15%.



Source: http://en.wikipedia.org/wiki/File:World-Wide-Smartphone-Market-Share.png

As can be seen in the graph IOS is making a steady ascent with a slight decrease in market share by the end of 2011, but with the probable introduction of the new iPad in March 2012. The market share is likely to grow.

IOS does not support Flash, and is not likely to ever do so, as can be understood from this article by Steve Jobs (ex-CEO of Apple)

http://www.apple.com/hotnews/thoughts-on-flash/

However, a growing number of Noteriks customers are starting to use IOS and want to use the functionality currently contained in Krusty on their mobile devices as well. Currently this is not possible because the application will not run on IOS.

Besides the boycott of Flash on IOS, Adobe has recently announced that it will not provide further updates for the Flash platform for mobile devices as announced here:

http://blogs.adobe.com/conversations/2011/11/flash-focus.html

Besides this, Flash is not an open standard. Future operating systems similar to Android and IOS might also choose to not adopt Flash because of this.

From these arguments we can extract the following conclusions as to why the fuctionality currently encapsulated in the Krusty player should be migrated to a new environment:

-IOS-users are a large target group which can not be reached at this moment.
-Flash platform will not be updated for mobile devices, it won't grow with future technical developments.
-Flash is not an open platform.

3. Requirements of the new environment

To achieve all the functionality currently encapsulated in the Krusty player the new environment should achieve the following requirements:

1. Has be able to run on mobile operating systems IOS and Android
2. Has be able to run within the following browsers: Safari 5+, Firefox 10+, Internet Explorer 9+, Google Chrome 17+
3. Has to be able to implement the following functionality of the Krusty player:
    - 3.1 Embedding into HTML page
    - 3.2 Basic Playout (playing, pausing, scrubbing, volume control)
    - 3.3 Showing events (such as peercomments and notes)
    - 3.4 Dock functionality(sharing video with friends, tagging moments etc)
4. Has to be an open standard.
5. Has to run on the client-side
6. Has to be compatible with the current backend structure of Noterik

4. Available alternatives

To migrate the functionality to a new environment, there has to be an understanding of what alternatives are available. Below here is a list of the alternatives that are most suitable to provide the functionality described in chapter 3.

4.1 Alternatives available

4.1 Adobe AIR

Adobe AIR is a cross-platform runtime environment which allows the developer to write code in ActionScript 3.0 which then gets compiled to a Flash and Javascript files. This provides a very flexible development environment because the compiled code will be able to run on a wide spectrum of devices including IOS. However it is not a open standard.

4.2 Apache Pivot

Apache Pivot is an opensource development environment developed by the Apache Software Foundation. Its aim is to allow developers to build installable Internet Applications. It is based on the JavaFX platform, and used to build desktop applications. It is therefore not supported on most mobile browsers.

4.3 HTML5 + Javascript

HTML5 is the latest version of HTML (HyperText Markup Language). This version introduces several new elements such as the <video>, <audio> and <canvas> element. These elements provide a new Javascript API with which you can manipulate these elements. Because HTML is the standard on which all internet browsers are based, there is a wide acceptance of the new version across almost all platforms. The interpretation of the standard however differs across browsers. Browsers on IOS and Android and almost all Desktop operating systems support HTML5 and the new Javascript API.

4.4 JavaFX

JavaFX is a platform developed by the Oracle Corporation used to develop Rich Internet Applications. Developers can code in a language called JavaFX Script which then gets compiled into Java bytecode. It is mostly used in Desktop applications and is not supported on IOS and Android.

4.5 Microsoft Silverlight

Silverlight is a proprietary platform by Microsoft to created Rich Internet Applications. The purposes of this platform are very close to that of Adobe Flash. Silverlight however is not supported on IOS and Android, and it is not an open standard.

4.2 Breakdown of functionality of all alternatives

A cross in a cell means that the OS of that row can fulfill the functionality of that column.

OS	IOS	Android	Basic Functionality	Open Standard	Client-side
Adobe AIR	x	x	x		x

Apache Pivot			x	x	x
HTML5 + Javascript	x	x	x	x	x
JavaFX			x	x	x
Microsoft Silverlight			x		x

5. Environment to be chosen.

As can be seen in the breakdown of the environments in the earlier chapter, there is only one environment that fulfils all the requirements. This is HTML5 + Javascript. The other alternatives either are not a open standard, or are not supported on all operating systems required.

HTML5 is already used in a lot of web applications and media content providers such as YouTube and Vimeo. Apple (which is one of the most influential businesses in the IT branche) has already declared its preference of this standard above Flash.

Even Adobe, which still has the largest market share of online video and RIA applications acknowledges that HTML5 is currently the best technology to develop with on mobile devices, as can be understood from the following citation:

This makes HTML5 the best solution for creating and deploying content in the browser across mobile platforms.[6]

There are however still things which might prove difficult in the implementation to HTML5. Video's can at this moment only be streamed through Apple's HTTP Live Streaming method or by Progressive Download.

HTTP Live Streaming is only supported by Safari and IOS devices.

Progressive Download is supported by all browsers, however it is not secure as the file will get downloaded to the harddisk.

Furthermore, there is no uniform codec for the video which can be used across all browsers, below is breakdown of what codecs are supported by which browsers:

Format	IE	Firefox	Opera	Chrome	Safari
Ogg	No	3.5+	10.5+	5.0+	No
MPEG 4	9.0+	No	No	5.0+	3.0+
WebM	No	4.0+	10.6+	6.0+	No

- Ogg = Ogg files with Theora video codec and Vorbis audio codec
- MPEG4 = MPEG 4 files with H.264 video codec and AAC audio codec
- WebM = WebM files with VP8 video codec and Vorbis audio codec

However, since HTML5 is a technology still in development, a standard will probably arrive once the technology is adopted more. But this is still a risk which should be held in mind.

---

[6] *Flash to Focus on PC Browsing and Mobile Apps; Adobe to More*
http://blogs.adobe.com/conversations/2011/11/flash-focus.html

6. Conclusion

Because HTML5 and Javascript fulfils all the requirements needed, it has been chosen as the environment to which the application should be migrated. There are however certain risks such as codec and stream incompatibility which should be held in mind when developing the application in this new environment.

Attachment 4

Functional Design

Krusty-JS

By order of:
Noterik B.V.
University of Applied Sciences Utrecht

Author:
David Ammeraal

Version:
1.0

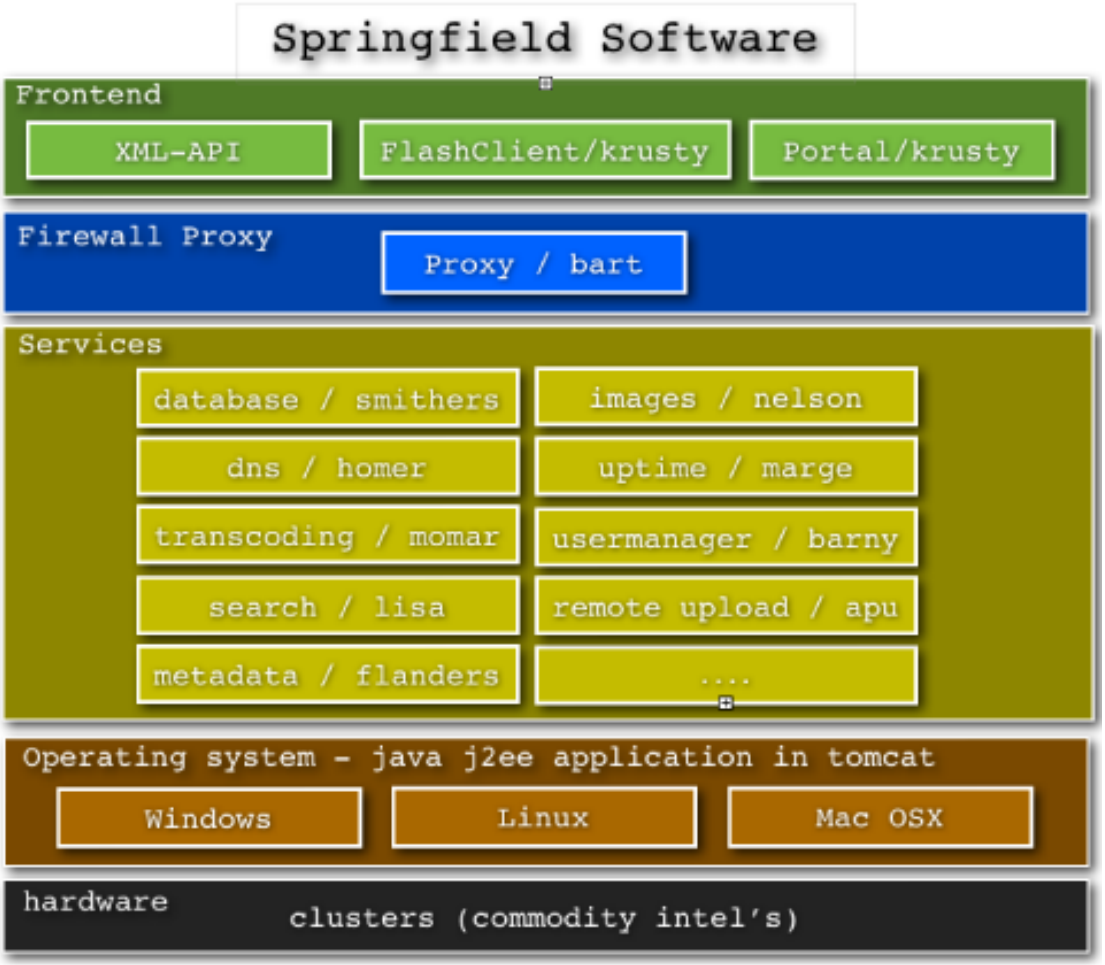
Version date:
23-02-2012

Contents

1. Introduction

1.1 Cause and goal

Noterik is a company which offers online video services. They have developed a platform to be capable of accommodating these services. This platform is called the Springfield platform. The platform consists of several tiers of functionality. One of these tiers is the frontend tier.



Inside the frontend tier is the Krusty player. This is an application in which the Springfield platform presents the video's stored in the operating system. Krusty is written in Adobe Flash.

The decision was made to construct a new Krusty player based on HTML5 and Javascript. This document describes the functionality which is required in the new Krusty player.

1.2 Scope of this document

In this document I will describe the functionality required for Krusty-JS at the end of the Construction phase. The following items will be described:

-Determination functional requirements
-The non-functional requirements
-A UML model describing the domain model of the application
-Determination of use cases
-Layout of the screens

2. Requirements

To be able to reach the wanted level of functionality, Krusty will have to meet some requirements.

2.1 Functional requirements

These are the functional requirements. I will attach a priority according to the MoSCoW model to every specific functional requirement.

ID	DESCRIPTION	PRIORITY
FN_EMBED	Embed the player into a .html file, and provide it with a location to request the presentation from.	MUST-HAVE
FN_EMBED_PARAM_DIMENSIONS	It should be possible to provide the player with a dimensions parameter, so that the player will set the dimensions of the player and video to the dimensions provided.	SHOULD-HAVE
FN_SCREENSHOT	Show a screenshot for the video after initializing the player.	MUST-HAVE
FN_PLAY	Play the presentation from the start or from the point where it was paused. It should be able to play a presentation consisting of a playlist of several video items.	MUST-HAVE
FN_PAUSE	Pause the presentation from playing.	MUST-HAVE
FN_TIMELINE	Show a timeline in which you can see the progress of the video playing. The timeline should span all the playlist items.	MUST-HAVE
FN_SCRUB	Move the box on the timeline to a place where you want the presentation to start playing from.	MUST-HAVE
FN_SCRUB_SCREENSHOT	Show a screenshot of the presentation at the time where the scrubber is placed on the timeline.	SHOULD-HAVE
FN_MUTE	Mute the volume of audio in the presentation.	MUST-HAVE
FN_VOLUME	Control the audio (louder or quieter) of the presentation.	MUST-HAVE
FN_FULLSCREEN	Make the presentation player full screen.	COULD-HAVE
FN_TIMER	Show a timer in which you can see the progress of the video and the duration of the presentation.	MUST-HAVE
FN_QUALITY	Select and change the quality of the presentation. The qualities that can be selected are 180p, 360p, 720p and 1080p.	MUST-HAVE
FN_EVENT	Show an event planned for a certain time period in a video.	SHOULD-HAVE

2.2 Non-functional requirements

The ISO-9126 standard provides a framework for non-functional requirements. I will summarize every non-functional requirement described in the ISO-9126 standard and attach a priority and explanation for this.

In the table below the number 1 represents a very low priority, and the number 5 means the highest priority.

2.2.1 ISO 9126

Requirement	1	2	3	4	5	**Description**
Functionality						
Suitability			x			Krusty-JS will have to perform as it should as described in this document. However some things might not finish in time, as the technology used is quite immature.
Accuracy				x		The functions described in this document should be performed as described.
Interoperability					x	Because Krusty-JS will rely on other systems for its data (the Springfield platform and possibly others), interoperability is very important.
Security	x					Security is at this moment not very important, HTML5 does not do much in means of protecting you data, and it isn't a priority for the project.
Reliability						
Maturity	x					Krusty-JS will not have all the functionality currently contained in the Krusty Flash player. Krusty-JS will serve as a springboard to more advanced versions of the player.
Fault Tolerance			x			Krusty-JS should not crash on every error, it should catch most exceptions and provide the user with information regarding the error.
Recoverability	x					No important data is stored in the instance of a Krusty-JS player. Recoverability therefore is not a priority.
Usability						
Understandability					x	Krusty-JS has to be very understandable. All the buttons have to be clear as to what functionality they provide.
Learnability					x	Krusty-JS has to be very easy to learn, a manual should not be needed.
Operability				x		There shouldn't have to be much interaction with the user.
Attractiveness		x				Krusty-JS should be skinnable through the use of .css files, this is however a task for a interaction designer and not for the developers.
Efficiency						
Time Behaviour				x		The application has to be very responsive and downloading extra resources should be done asynchronously.

Resource Behaviour				x		The footprint of the application should be as small as possible, the source should be optimized and minimized after every build to ensure this.
Maintain-ability						
Analysability					x	The application has to very analysable. This can be achieved by documenting the code thoroughly and showing function calls in the debug console.
Changeability				x		The code shouldn't be hard to change or add extra functionality to. This can be achieved by creating entry points in the code through the use of call-back functions.
Stability		x				The application will not be mature after the project. It will therefore not have to be very stable, although it shouldn't crash at every error.
Testability				x		The application has to be testable. By writing tests before actually creating code, every functionality the of code will be very testable.
Portability						
Adaptability		x				For now Krusty-JS only has to work properly in Safari. Adaptability therefore is not a big priority at this moment.
Installability		x				The installation of the application should not be very complicated. However since the users that are going to install it are mostly developers, the ease of installing krusty-js is not very important.
Co-existence					x	The application will have to run inside an existing web-app without interfering with any of the variables or functionality of another webapp, the scope of events and variables should therefore be kept within the application.
Replaceability					x	If the user for some reason can't use JavaScript or HTML5 or doesn't support the codec in which the video is provided. It should not be possible to fall back to the old Krusty player.

3. Domain model of the application

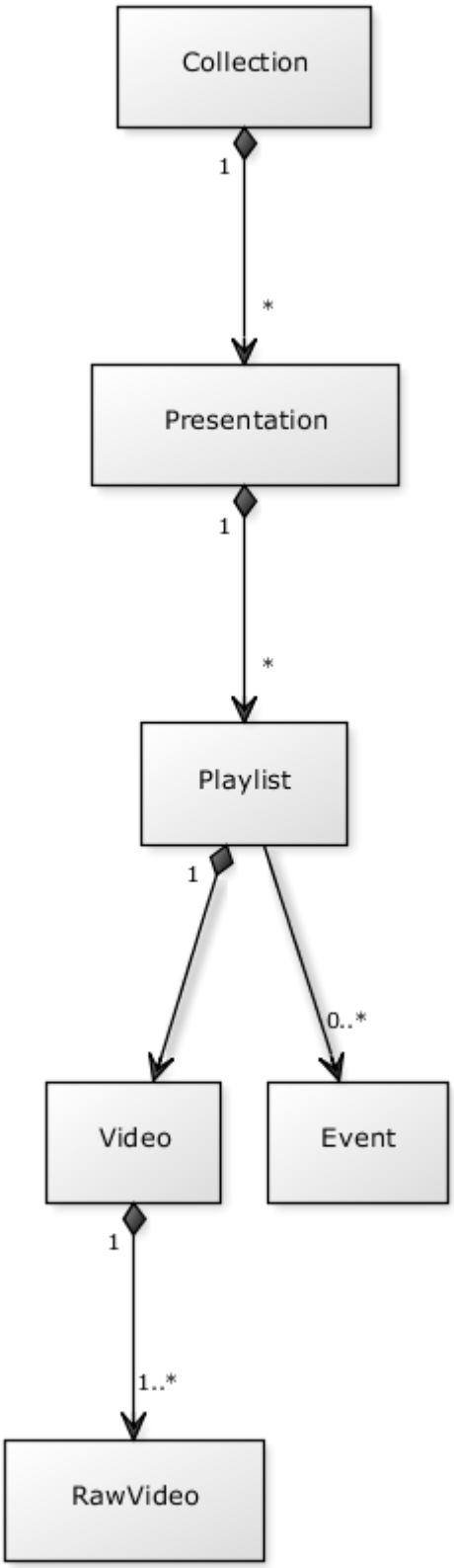


Figure 17 http://yuml.me/diagram/dir:tb/class/%5BCollection%5D++1-%3E*%5BPresentation%5D,%5BPresentation%5D++1-%3E*%5BPlaylist%5D,%5BPlaylist%5D++1-%3E%5BVideo%5D,%5BPlaylist%5D-0..*%3E%5BEvent%5D,%5BVideo%5D++1-%3E1..*%5BRawVideo%5D

4. Use cases

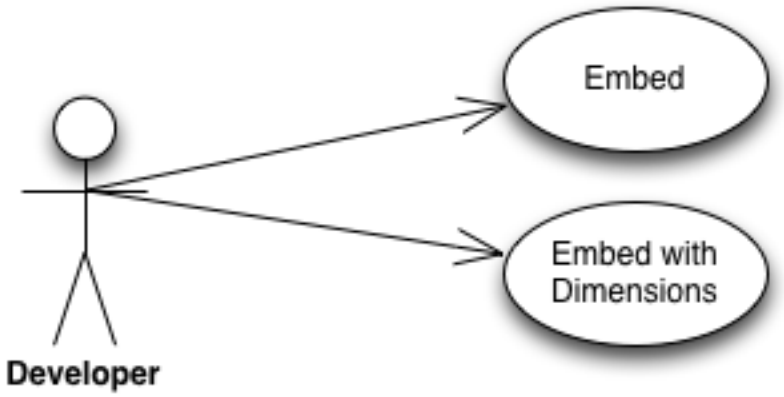


Figure 18. Developer Use cases

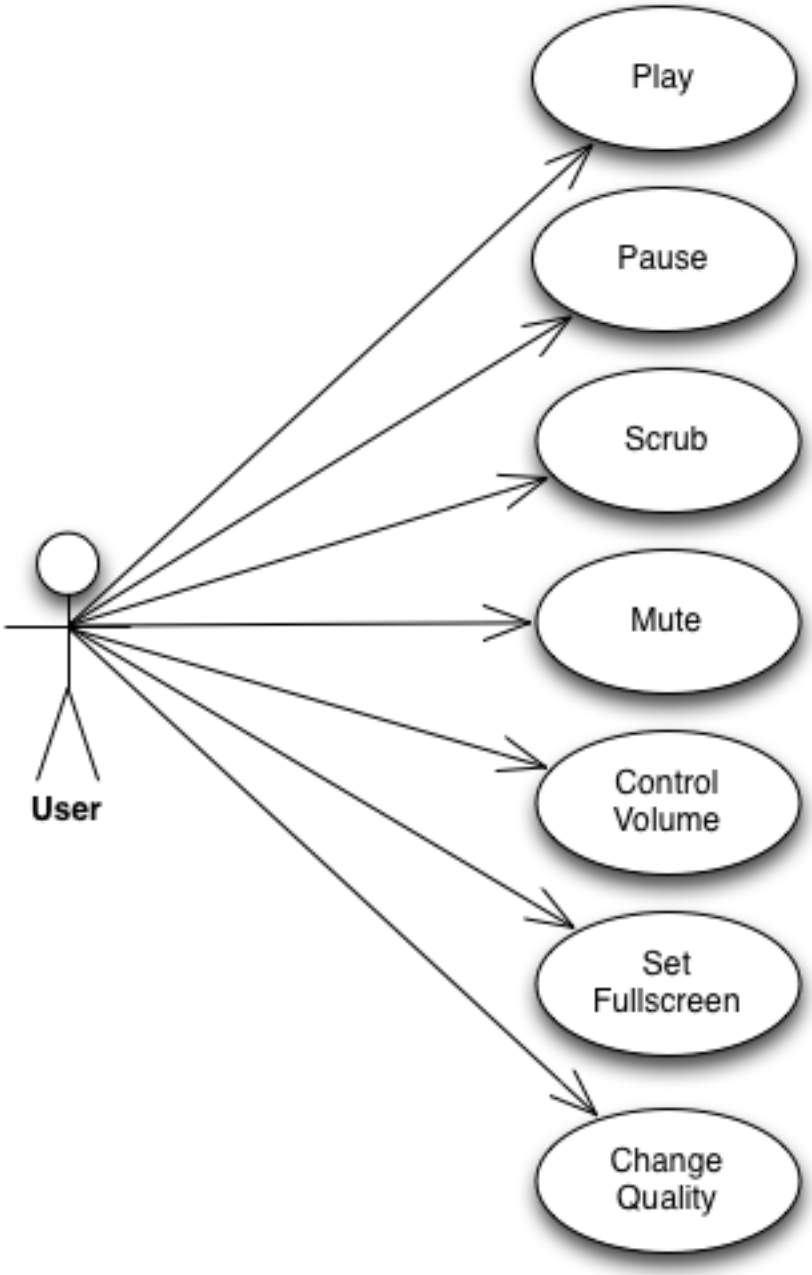


Figure 19. User Use cases

5. Use case specification

5.1. Embed

5.1.1 Brief description

This use case describes what happens when the developer embeds a presentation in an existing .html webpage, without any parameters.

5.1.2 Actors

-Developer

5.1.3 Domain objects involved

-Presentation
-Playlist
-Video

5.1.4 Preconditions

-A HTML page has been created.

5.1.5 Basic flow of events

1. The developer inserts the embed code into a .html document
2. The developer opens the .html document in a web-browser.
3. The system is initialized with the default settings, and the location of the quickstart file provided by the user.
4. The system validates the settings.
5. The player view is initialized with the default dimensions and buttons.
6. A loading icon is shown to the user within the player view, and the buttons are disabled but visible.
7. The system requests the quickstart file containing the presentation information from the remote location.
8. The systems parses all settings for the presentation.
9. After getting the quickstart file the system checks if the codec of the presentation described in the quickstart file is supported by the browser.
10. The system requests the first video of the presentation with the default quality from the remote server.
11. The screenshots of the video for scrubbing are requested.
12. The system starts buffering the video.
13. A buffering icon overlaying a screenshot of the first video in the presentation is shown to the user within the player view.
14. After buffering enough of the file for a successful play through a play button overlaying a screenshot of the first video in the playlist is shown to the user within the player view.
15. The buttons are enabled.
16. Use case successfully completed.

5.1.6 Alternate flow of events

5.1.6.1 The format of the url provided by the user is incorrect

If in step 4 the validation of the url doesn't complete then:

1. A message will be shown within the player view to the user that the url provided for the video is not correct.
2. The use case ends in failure.

5.1.6.2 The remote server is not responding

If in step 7 the remote server containing the quickstart file can not be reached then:

1. A message will be shown within the player view to the user that the server containing the quickstart file can not be reached.
2. Use case ends in failure.

5.1.6.3 The codec of the video on the server is not compatible with the browser being used

If in step 9 the codec of the video isn't compatible with the browser of the user then:

1. A message will be shown within the player view to the user that his browser is not compatible with the codec of the video.
2. The use case ends in failure.

5.1.6.4 The video on the remote server can not be loaded

If in step 10 the server containing the video file can not be reached then:

1. The systems attempts to get the video from the fall back location defined in the quickstart file (not the same as the quickstart fall back)
2. The flow of events continues from step 11.

5.1.6.5 The video on the fall back server can not be loaded

If in step 1 of 5.1.6.4 the video from the fall back server can't be loaded then:

1. A message will be shown within the player to the user that the codec of the video is not compatible with the browser that the user is using.
2. A dialog will be shown within the player view asking the user if he wants to skip to the next video.
3. The use case ends in failure.

5.1.6.6 The screenshots can not be found

If in step 11 screenshot can not be found then:

1. The player will continue without screenshots, scrubbing will have no screenshots
2. The flow of events continues from step 12.

5.1.7 Post-conditions

5.1.7.1 The user can play the presentation.
5.1.7.2 The user has been informed why the presentation could not be loaded.
5.1.7.3 The user has been informed why the video in the playlist could not play, and is asked to skip to the next video in the playlist.

5.2 Embed with Dimensions

5.2.1 Brief description

This use case describes what happens when the developer embeds a presentation in an existing .html webpage, with a parameter defining the maximum width of the player.

5.2.2 Actors

-Developer

5.2.3 Domain objects involved

-Presentation
-Playlist
-Video

5.2.4 Preconditions

-A html page has been created

5.2.5 Basic flow of events

1. The developer inserts the embed code into a .html document.
2. The developer defines the dimensions of the player.
3. The developer opens the .html document in a web browser.
4. The system is initialized with the default settings, and the location of the quickstart file provided by the user.
5. The system validates the settings.
6. The player view is initialized with the defined dimensions.
7. A loading icon is shown to the user within the player view, and the buttons are disabled but visible.
8. The system requests the quickstart file containing the presentation information from the remote location.
9. The systems parses all settings for the presentation.
10. After getting the quickstart file the system checks if the codec of the presentation described in the quickstart file is supported by the browser.
11. The system requests the first video of the presentation with the default quality from the remote server.
12. The aspect ratio of the player is changed to that of the video while maintaining the dimensions of the player.
13. The system starts buffering the video.
14. A buffering icon overlaying a screenshot of the first video in the presentation is shown to the user within the player view.
15. After buffering enough of the file for a successful play through a play button overlaying a screenshot of the first video in the playlist is shown to the user within the player view.
16. The buttons are enabled.
17. Use case successfully completed.

5.2.6 Alternate flow of events

Same alternate flows as in 5.1.6

5.2.7 Post-conditions

Same post conditions as in 5.1.6.1 and 5.1.7.2 with the addition of:
5.2.7.1 The player is shown in the dimensions defined by the user.

5.3 Play presentation

5.3.1 Brief description

This use case describes what happens when the player indicates that he wants the presentation to be played.

5.3.2 Actors

-User

5.3.3 Domain object involved

-Presentation
-Playlist
-Video

5.3.4 Preconditions

-A HTML page has been created.
-The video is embedded in the HTML page.
-The system has already requested the first video from a presentation, and is ready to start playing (done buffering).

5.3.5 Basic flow of events

1. The user indicates that he wants the presentation to be played.
2. The system starts playing the video, while still buffering the video in the background.
3. If there are any events to be shown at any time in the presentation the system will display those at the right time.
4. After done with playing the current video in the playlist, the player automatically skips to the next video in the playlist.
5. After playing the last video in the playlist the player stops displaying the video and displays the screenshot of the first video with a play button overlaying it, ready to start playing again.
6. Use case completed successfully.

5.3.6 Alternate flow of events

5.3.6.1 The internet connection dies or the video can't be buffered anymore.

If in step 2 the video can't be buffered anymore, then:

1. Continue play out from the point where buffering stopped from a fall back location.
2. Continue from step 3.

5.3.6.2 The fall back location also can't be reached.

If in step 1 in 5.3.1 the fall back location also can't be reached, then:

1. Tell the user that the presentation can't be played.
2. Use case ends in failure.

5.3.6.3 The location of the next video in the playlist can't be reached

If in step 4 the location of the next video can't be reached, then:

1. Get the video from a fall back location.
2. Redo step 5 for the next video.

5.3.6.4 Fall back location of the next video also can not be reached

If in step 1 of 5.3.6.3 the fall back location also cannot be reached, then:

1. Inform the user that video could not be loaded, and ask him to skip to the next video in the playlist.
2. Redo step 5 for the next video.

5.3.7 Post-conditions

5.3.7.1 The user has finished watching the presentation and can play it again if he wants to.

5.3.7.2 The user has been informed why the presentation couldn't be played anymore.

5.4 Pause presentation

5.4.1 Pause Presentation

This use case describes what happens when the user presses the pause button while a presentation is playing.

5.4.2 Actors

-User

5.4.3 Domain objects involved

-Presentation
-Playlist
-Video

5.4.4 Preconditions

-A presentation is playing.

5.4.5 Basic flow of events

1. The user indicates that he wants the presentation to be paused.
2. The presentation is paused.
3. There is a button with which the user can start the presentation playing again.
4. If there is any buffering to be done, the system continues buffering.

5.4.6 Alternate flow of events

No alternate flow of events.

5.4.7 Post-conditions

5.4.7.1 The presentation has been paused and the user can start playing again if he wants to.

5.5 Scrub Presentation

5.5.1 Brief description

This use case describes what happens when the user wants the presentation to continue playing from a certain time in the video. He can do this by dragging a handle over the timeline of the presentation and dropping it at the place he wants to continue playing from.

5.5.2 Actors

-User

5.5.3 Domain objects involved

-Presentation
-Video

5.5.4 Preconditions

-A presentation is playing, ready to be played or paused

5.5.5 Basic flow of events

1. The user drags the handle on the timeline.
2. While scrubbing the presentation, the player shows a screenshot of the presentation on the time where the handle is on. On every move of the handle the screenshot changes.
3. The user lets go of the handle on the place where he wants the presentation to continue playing from.
4. The system sets the current time to the time of that of the handle.
5. If the presentation was paused the video remains paused, but the position is changed, if the video was playing the video will continue playing from that time.
6. The use case ends successfully.

5.5.6 Alternate flow of events

5.5.6.1 The screenshots for the video can't be found.

If there are no screenshots available to show while scrubbing the presentation then:

1. No screenshots will be shown.
2. Continues from step 3.

5.5.7 Post-conditions

5.5.7.1 The user can scrub to any point in the video and see screenshots of the video of the time where he is scrubbing.
5.5.7.2 The user can scrub to any point in the video.

5.6 Mute or unmute presentation

5.6.1 Brief description
This usecase describes what happens when the user indicates that he wants the audio of the presentation to be muted.

5.6.2 Actors
-User

5.6.3 Domain objects involved
-Presentation

5.6.4 Preconditions
-A presentation is playing, ready to be played or paused.

5.6.5 Basic flow of events
1. The user indicates that he wants the presentation to be muted.
2. The system provides the user a button with which he can unmute the audio.

5.6.6 Alternate flow of events
No alternate flows.

5.6.7 Post-conditions
5.6.7.1 The user does not hear any audio, and is provided with a button with which he can unmute the audio again.

5.7 Control Volume

5.7.1 Brief description
This use case describes what happens when a user controls the volume of the audio of a presentation.

5.7.2 Actors
-User

5.7.3 Domain object involved
-Presentation

5.7.4 Preconditions
-A presentation is being played, ready to be played or paused.

5.7.5 Basic flow of events
1. The user indicates that he wants the volume of the audio to be lower.
2. The system lowers the volume.

5.7.6 Alternate flow of events

5.7.6.1 Volume button disabled
No alternate flows.

5.7.7 Post-conditions
5.7.7.1 The volume of the audio is changed to what the user indicated.

5.8 Set video to full screen

5.8.1 Brief description
This use case describes what happens when a user indicates that he wants the presentation to be shown in full screen mode.

5.8.2 Actors
-User

5.8.3 Domain objects involved
-Presentation

5.8.4 Preconditions
-A presentation is being played, ready to be played or paused.

5.8.5 Basic flow of events
1. The user indicates that he wants the presentation to be displayed in full screen.
2. The system scales the video to full screen mode. If the aspect ratio doesn't allow the presentation to fully fill the screen, then add black bars to the bottom and top to fill the screen.
3. The use case ends in success.

5.8.6 Alternate flow of events
No alternate flow of events.

5.8.6.1 Full screen button disabled
1. The user can't click the full screen button.
2. The use case ends in failure.

5.8.7 Post-conditions
5.8.7.1 The presentation is displayed in full screen.

5.9 Change quality of video

5.9.1 Brief description
This use case describes what happens when the user indicates that he wants the quality of the video to change.

5.9.2 Actors
-User

5.9.3 Domain objects involved
-Presentation

5.9.4 Preconditions
-A presentation is played, ready to played or paused.

5.9.5 Basic flow of events
1. The user indicates that he wants to change the quality of the video.
2. The system stops the current video and requests the new video.
3. A loading indicator is shown to the user.
4. After buffering enough of the video, the presentation continues playing from the point where the user indicated that he wanted the quality to change.
5. The user sees the presentation in the quality that he indicated.

5.9.6 Alternate flow of events
No alternate flow of events.

5.9.7 Post-conditions
55.9.7.1 The user sees the presentation in the quality that the user selected, and the rest of the videos in the playlist will also be displayed in this quality.

6. Screen designs

6.1 Basic layout

This is the basic layout of the Krusty-JS presentation player. This is a mock up. The end result should look exactly like the Flash version.

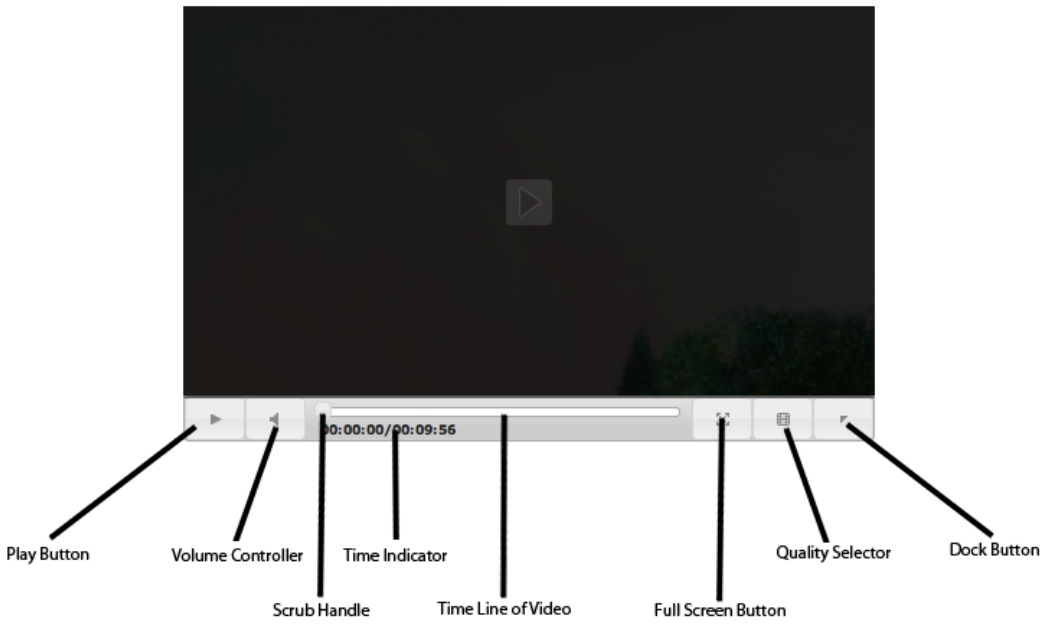
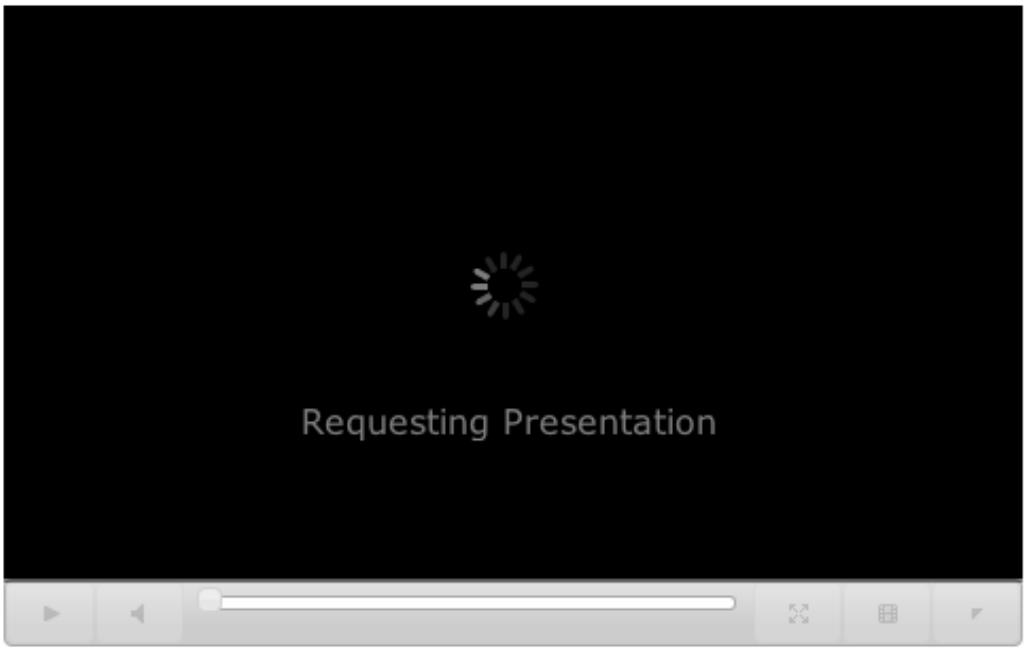


Figure 20 Basic Layout

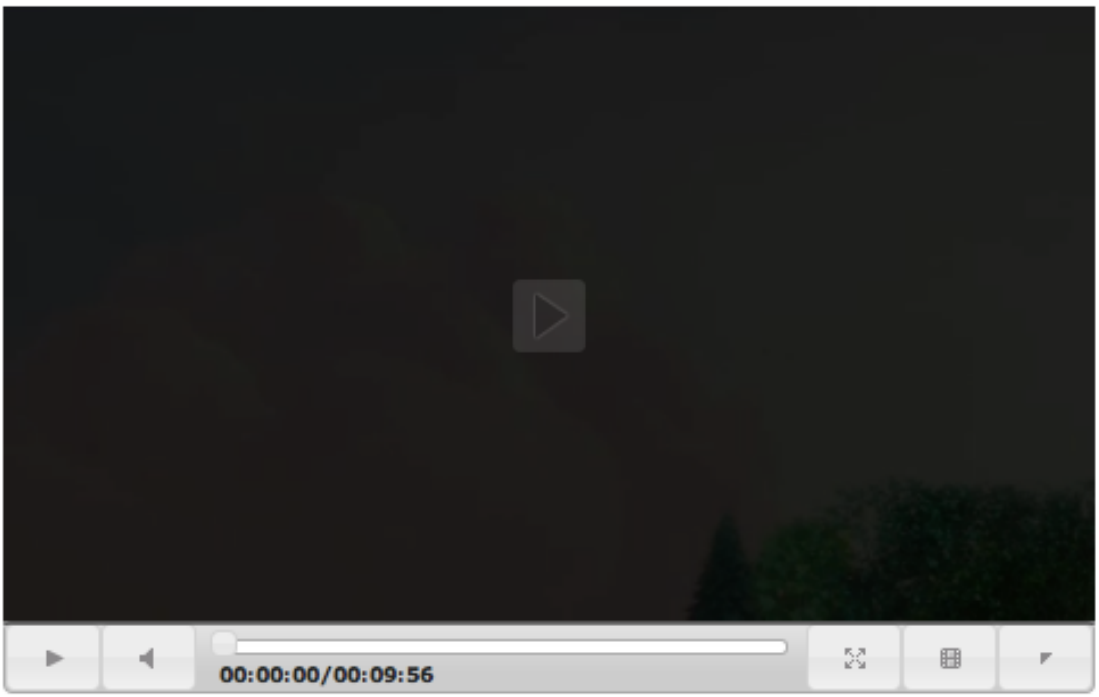
6.2 Loading presentation

The player is initialized, but all buttons are disabled, a message is shown to the user to inform him that the presentation is being loaded.



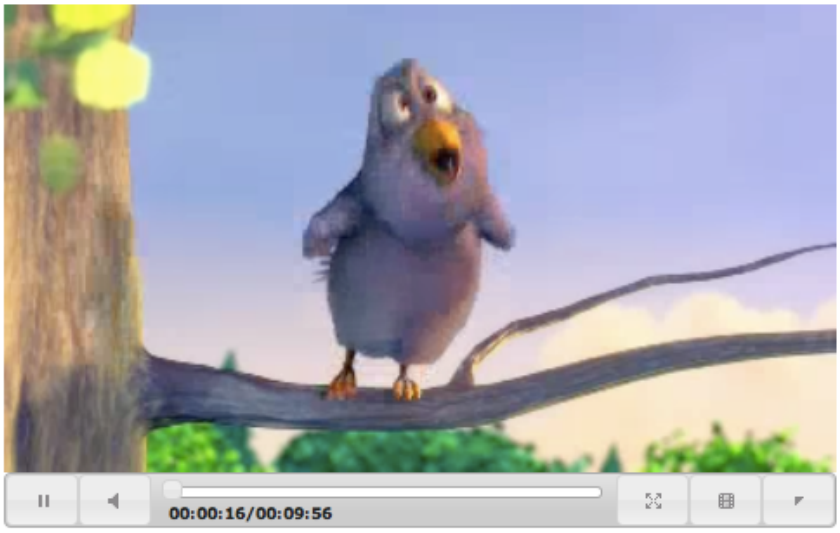
6.3 Done loading

This is the screen after the first video of the presentation is loaded. The duration of the video is added and all the buttons are available. Also a play button is added to the overlay to indicate that video is ready to start playing.



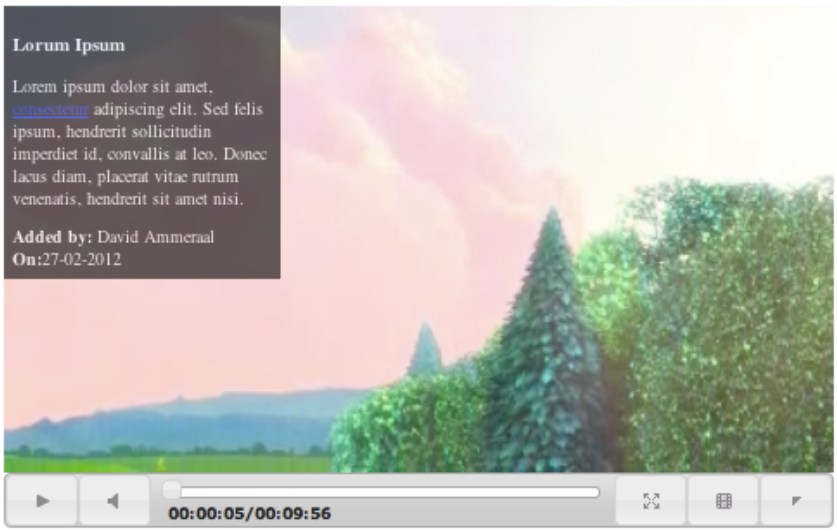
6.4 Video playing

This is the screen when a video is playing, all buttons are available for use, and the progress indicator is updated every second to indicate at what time the video is. The play button is changed to a pause button.



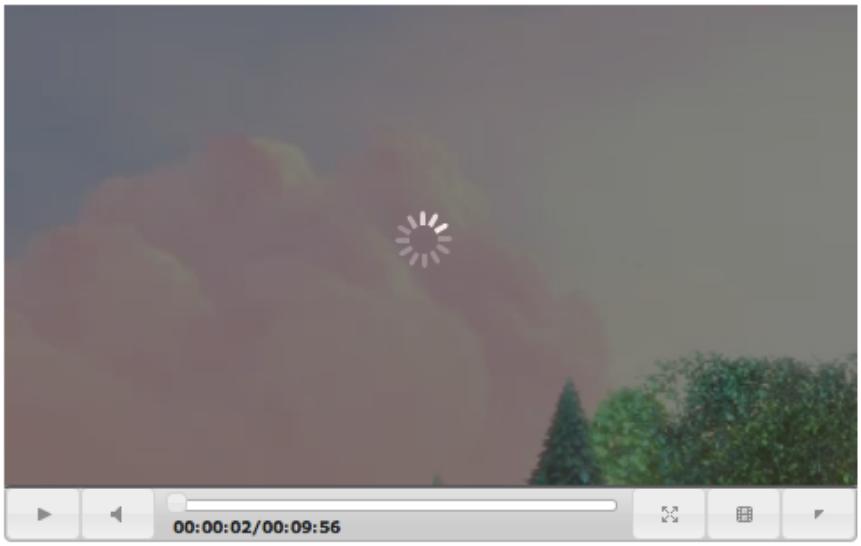
6.5 Showing event

The event wil overlay the video.



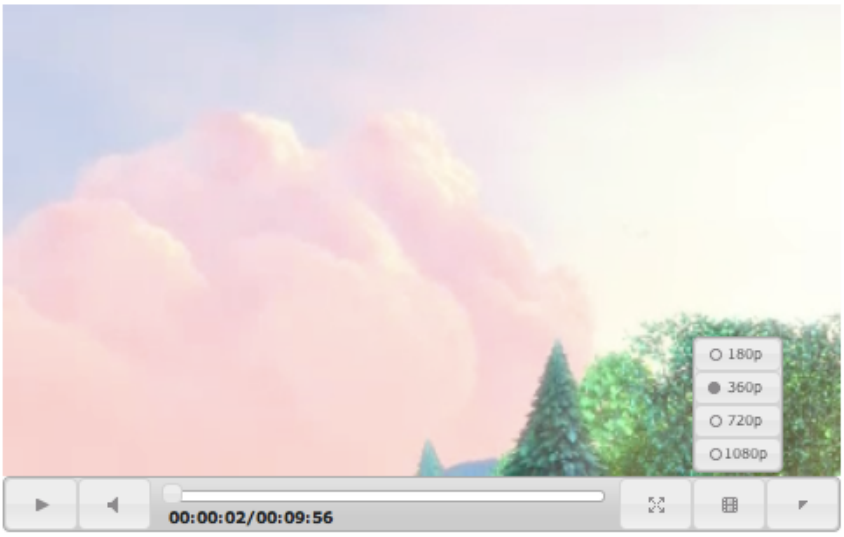
6.6 Buffering

When the buffer of video has run out, the user will be informed that the video wil have to buffer before playing again.



6.7 Quality selection

The quality currently playing is blacked out.



6.8 Show dock

Dock is still empty at the moment of integration, but extra functionality should be added there.