Design and implementation of a web-based platform to present designer-portfolios

Bachelor Thesis

University of Applied Sciences Düsseldorf
Department of Media

Degree Program: Media Informatics (B.Sc.)

Author: Jan Monschke (507653)

First Examiner: Prof. Dr.-Ing. MSc Markus Dahm
Second Examiner: Dipl.-Inf. Sebastian Deutsch

Affidavit

Ich, Jan Monschke, geboren am 12.03.1987, versichere, diese Bachelorarbeit selbstständig und lediglich unter Benutzung der angegebenen Quellen und Hilfsmittel verfasst zu haben.

Ich erkläre weiterhin, dass die vorliegende Arbeit noch nicht im Rahmen eines anderen Prüfungsverfahrens eingereicht wurde.

Düsseldorf, den

Jan Monschke

Table of Contents

Abstract

The Internet has become a medium that is used by more and more people on a daily basis. In the past, websites used to be static HTML pages whose only purpose was to provide information to the users. The image of websites has changed a lot recently and websites are now more than just accumulations of information. Websites have become a lot more interactive and content is very often not just created by dedicated authors but by the users themselves.

Although the usage of websites has changed over the time, the development of websites is still similar to the development in the beginning. Most work is done on servers and clients are mainly needed to display HTML. This architecture complicates the development of highly interactive websites.

Single Page Web Apps is a novel architectural style that lets developers write interactive websites in a more integrated and efficient way. Their approach is to assign more duties, which have been typically placed in the backend in traditional websites, to the frontend.

This work will describe an implementation of a website that has been developed as a Single Page Web App to create a unique interactive User Experience and furthermore, it will give an in-depth description and discussion of Single Page Web Apps in general.

Structure

This thesis is divided into two parts. The first part concentrates on the design and implementation of a concrete website and the second part focuses on the description of Single Page Web Apps in general.

The first part starts with an overview of the concept of Salon, the website that has been implemented in the course of this thesis. Afterwards, concrete details of the implementation and the platform's structure are described. In the end, the platform is evaluated.

The second part starts with an overview of Single Page Web Apps in general that also includes a comparison with a traditional website architecture. After that, differences in the structure and in the development process of both approaches are compared. Then the benefits and problems with Single Page Web Apps are discussed and commented with best-practice techniques that have been proven to work well during the development of Salon. The second part closes with a conclusion on the usefulness of Single Page Web Apps.
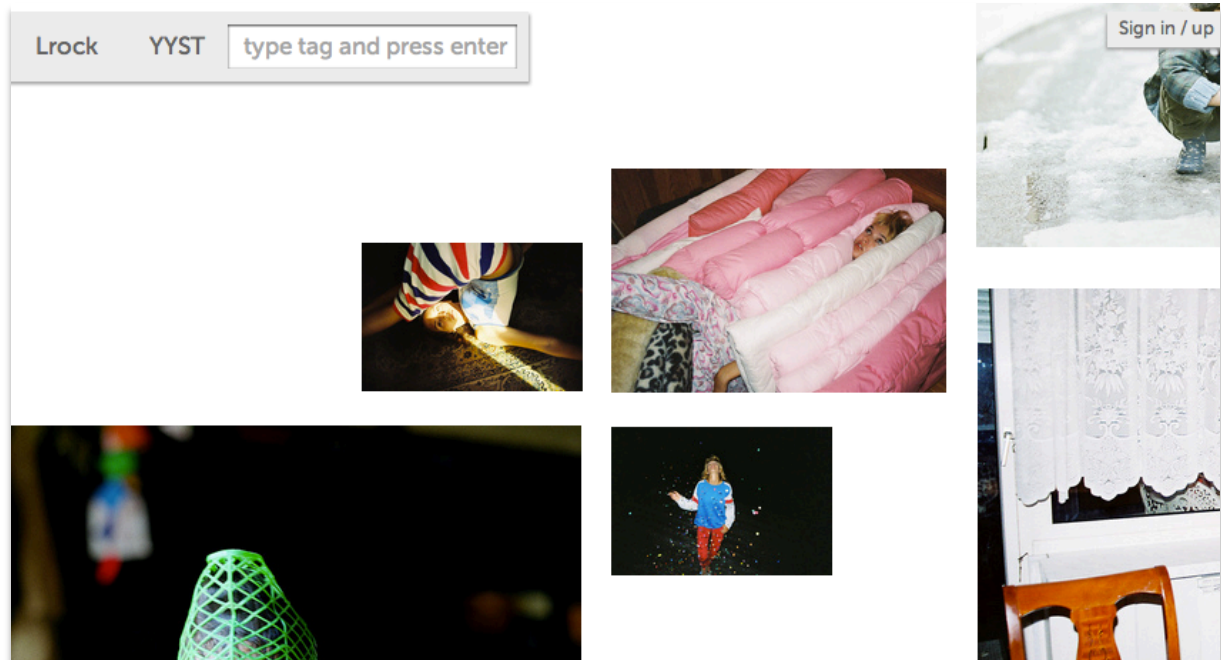
I Salon

1. Overview

Fig. 1: A page called "YYST" by the user "Lrock"

Salon is a web-based platform that allows its users to create pages and to upload images onto these pages for creating an online portfolio of their work. The main target audience is photographers, illustrators and graphic designers. The platform focuses on a unique way in both presenting and arranging sets of images. The user is able to fully control the way the images are presented to the visitors.

Other services, like for example Flickr, do not let users decide about the presentation of their work and only provide simple list views and slide shows. On these websites all albums look the same because the raster is always predefined.

In Salon, all images are placed on a canvas and can freely get dragged around by the user to create innovative and unique arrangements. Also the canvas itself can be moved to focus a certain point of a page. Another feature is that images can link to other pages so that users can create associations between pages or even associations between users. These features give the user another way to express his creativity and make Salon stand out from all competing websites.

2. The idea

Dipl.-Inf. Sebastian Deutsch and Dipl.-Des. Stefan Landrock developed the basic idea behind Salon when they were guest-lecturing courses at the HFG Offenbach. Together

with their students, they built a working prototype of their idea so they could use it for their courses and especially for their presentations. When other universities heard about Salon they were asked if they could host a system for their students, too. But Salon was not built to be deployable for other universities and so they had the idea to rewrite and to extend the features of Salon so that it could easily be set up for other universities.

3. The Platform

This chapter covers the implementation of Salon and gives an overview of the features of Salon and explains why they are important for the platform.

3.1 Technology

The backend of Salon is implemented in Ruby on Rails (short Rails), a web framework written in Ruby[1] and modeled after the MVC software pattern[2] that allows to quickly create solid web applications without having to care about low-level problems like session-handling or database access. The underlying database is MongoDB[3], a document-oriented database system[4] that was chosen because of its flexibility and its very good integration in Rails.

Salon does not make use of the full Rails stack and especially the frontend tool chain is completely ignored because the frontend is designed to work as a Single Page Web App (see II 1.1 "Motivation for Salon").

The communication between the frontend and the backend is realized with a REST[5] interface and all data is being sent in the JSON[6] format, a format that is very easy to use in both JavaScript (frontend) and Ruby (backend).

---

[1] Ruby is a object-oriented programming language with a dynamic type system
[2] [MFPATT2002]
[3] http://www.mongodb.org/
[4] Document-oriented databases are schema-free databases [EHCASS2011]
[5] REST stands for Representational State Transfer and is a software architecture style that is very efficient by making extensive use of HTTP methods [RFARCH2000]
[6] http://en.wikipedia.org/wiki/JavaScript_Object_Notation, retrieved on 04/16/2011
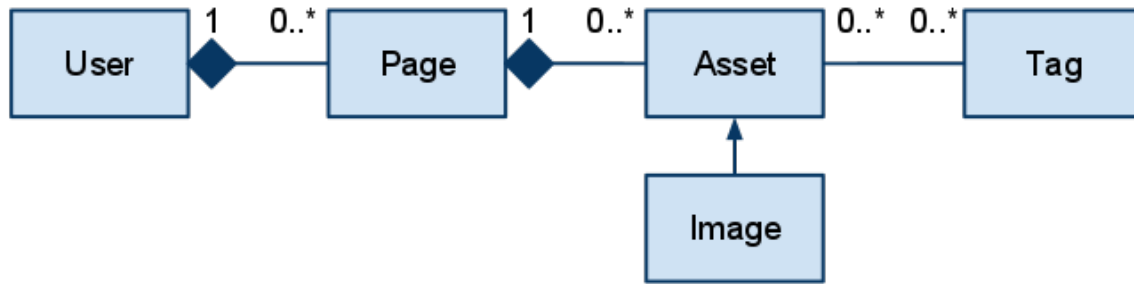
3.2 Data Model

Fig. 2: The underlying data model of Salon. (Properties have not been included in this schema.)

The underlying data model of Salon is shown in Fig. 2. There is the User class that is used for authentication and it has properties like a username and a password.

The Page class is associated to the User class in a one-to-many relationship, which means that instances of User can reference many instances of Page and each instance of Page belongs to only one instance of User. Pages have properties like a title, a cover image and a publish state. Each Page has a list of Assets that are also associated in a one-to-many relationship so that each Asset can be associated to one Page.

The Asset class is the parent class for Image and it stores properties like a title, a link-to location and a position on the canvas.

The reason for inheriting the Image class from the Asset class is to allow other Assets, like for example texts, in the future (see I 4. "Quo vadis Salon?") and to provide all instances of Image with the needed properties that make up a valid Asset. The Image class then only needs to save properties like the image files and its display sizes.

All Asset instances have a list of Tags that are associated in a many-to-many relationship which means that an instance of Tag can belong to many instances of Asset and each instance of Asset is able to reference many instances of Tag. The Tag class is used in the search (see I 3.3.9 "Search") and only stores a title and the associations.

3.3 Pages
This chapter describes the most important pages of Salon from a user's perspective and additionally explains the concepts that formed the basis for the decisions why things have been modeled the way they are now.

3.3.1 Index
The index page has, as well as other elements in Salon, two states that depend on the login state of the user. If the user is not logged, in the index page displays a text that

invites the user to register an account at Salon and a link to the "about"-page so that new users quickly get an idea about what Salon is and how they can use it.

If the user is logged in, the text on the index page welcomes the user and a list of recently created and edited pages is shown at the bottom of the page.

The index page does not have much functionality since the discovery of pages and images is realized with the search field that is located in the navigation (see I 3.3.9 "Search").

3.3.2 Registration and Account

In Salon users have to sign-up with a Username, an E-Mail address and a password. To complete a registration, the user receives an e-mail with a confirmation link. This step is needed to confirm that a user registered with a valid e-mail address. After clicking the link in the mail, the system redirects the user to an empty page called "untitled page" and a message is shown that the account now has successfully been confirmed. Since this is the first time the user uses Salon, another message on the page tells him that he now is able to add images to the page by dragging them onto the window. Also a link to the "about"-page is shown so that new users can get a quick overview on the features of Salon.

To edit account details, like the password or the e-mail address, users can use the "Account" page (link in the menu). This page also allows the user to delete the account.

3.3.3 User Overview

Fig. 3: Overview of the user "Lrock"

In the user overview all (published and not hidden) pages of a user are displayed on a canvas. Since pages can have a cover image, on this page only the cover images are shown. If a page does not have a cover image, a default picture will get displayed instead. The user is able to arrange all images simply by dragging them around. The positions are being saved to the server so that this page will look the same for all visitors and just as the user wants it to look like. Visitors themselves can also drag the images around and can create a new layout but the position will not get saved to the server since only the owner has the right to decide how his pages look like.

When dragging an image, the image will get populated to the top of all other images so that users can easily create nice effects with occluding images. All these changes will automatically get saved to the server without the need for the user to initiate the save-process.
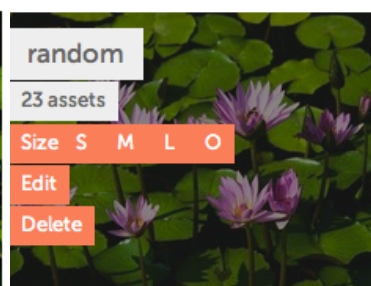

Fig. 4: Image (not hovered)   Fig. 5: Hovered image with extra info

To highlight the importance of the images and especially their arrangement, there are no further information displayed on top of each image (see Fig.4). This also helps to make pages with a lot

of images look not too cluttered.

Further information for a page is displayed on top of the images when a user hovers (or taps) over one of the images. The name and the number of included assets will then fade in and the image gets a half-lucent overlay to highlight which image currently is being hovered (see Fig. 5). There is the need for the half-lucent overlay because when many pages are placed in the same place, it is hard to find out which of the images has just been hovered. Clicking one of the images or its captions will navigate the user to the overview of the page.

When the current user is logged in, there will also be additional controls displayed on top of each hovered image (Fig. 5: The orange captions). First there is the control to set the size of the image that lets the user choose between four different size options. Then there is a link to the edit page of the current page that allows a user to quickly edit the page and there is a link to delete the current page. All delete operations in Salon trigger a prompt before actually deleting an element to prevent accidental deletions. Furthermore, the user is not only able to drag each image around but also the whole page which allows to choose a special "starting" point of the canvas that the visitor sees when he first comes to the page. If the user wants to drag the whole page he simply needs to drag the background and all other images will get moved accordingly.

3.3.4 Page Overview

At first sight the page overview looks similar to the user overview. The images can freely get dragged around and the title of each image is displayed when the image is hovered. Logged in users also have the ability to directly edit or delete images with the additional captions here. Like in the user overview, the user is navigated to the image page when he clicks the image or one of the captions on the image.
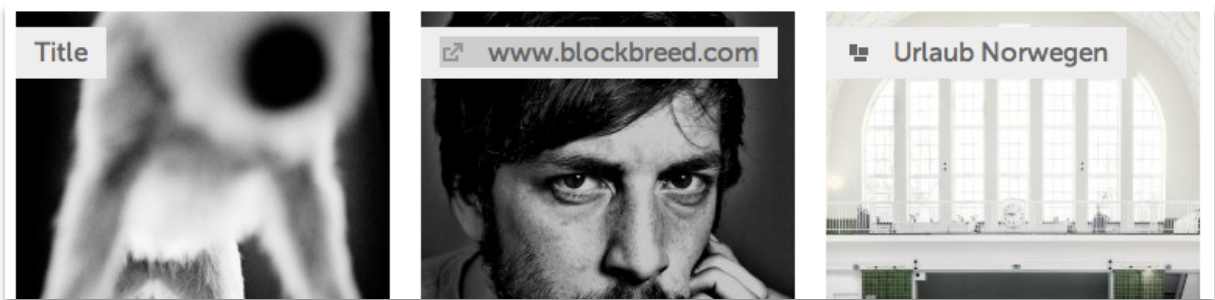

Fig. 6: The three link modes of an image (overview, external link, page of the user)

Besides that, there are subtle changes to some of the images (see Fig. 6). They have special icons that should indicate that they do not link to the image page but to an external page or to another page of this user. (see I 3.3.7 "Image Edit Form")

Also there is another caption right underneath the navigation that allows the user to quickly jump to the edit form of this page.
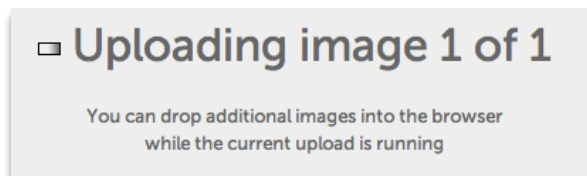
Fig. 7: Upload progress indicator

Another additional feature is the ability to directly upload pictures to the page by simply dragging picture files from the file system onto the page. A progress dialog (Fig. 7) will open up that shows the user how many files are left for upload. The uploaded images will get directly added to the page so that the user can work with them on the page right away.

3.3.5 Page Edit Form

Fig. 8: The page edit form

On this page the user is able to edit several aspects of a page like its title or its description. Changing the title of a page also leads to the creation of a new URL-slug[7] for this page, so that the URL and the title of a page always correspond. Underneath the normal form, there is a listing of all assets that are associated to this page. When hovering one of these images, new controls for editing and deleting the image fade in. Also another option fades in that lets the user set this image as the cover image. When this one gets clicked, the image in the normal form changes automatically.

---

[7] A part of a URL that identifies a page and is human-readable
http://en.wikipedia.org/wiki/Slug_(web_publishing), retrieved on 04/16/2011

The user is furthermore able to add new images on this page directly by dragging them somewhere onto the page or by opening up the file dialog with the "Add asset" button. The uploaded images will then automatically appear in the asset list.

In contrast to the overview pages the user here has to save changes manually with the buttons that are placed directly under the navigation. There is also a button to cancel the edit form that will remove all changes the user has made and will redirect the user to the page. The third button deletes the page.

3.3.6 Image Overview

On this page, the current image is shown in the original size as the user uploaded it. The image is centered horizontally and vertically so that the center of the image lies on top of the center of the pages. Like on the other pages, the image can also get dragged around which is handy for images that are bigger than the browser screen so that users can see the rest of each image by dragging it around. The position of an image is not saved to the server because the main focus on this page is not a specific arrangement but the image itself.

By pressing the right- or left key, the user can navigate through the rest of the images of the current page to get a quick overview on all images.

3.3.7 Image Edit Form

Like in the page edit form (Fig. 8), the image edit form first shows the image in order to make it clear to the user which one of the images he is currently editing. There are basic input fields for changing the title (which will also change the URL), the description and there are fields to add Copyright and source information to the image which are important if the user adds an image that has been taken from another website.

Tags can simply get added to the image (Fig. 9) by typing a tag into the tag field and pressing the return-, the space- or the comma-key. To delete a tag, the user simply needs to press the "X"-symbol next to each tag.
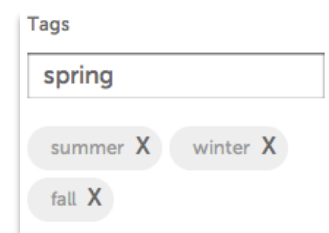

Fig. 9: Tag form

Fig. 10: Image links to the detail

Fig. 11: Image links to a page
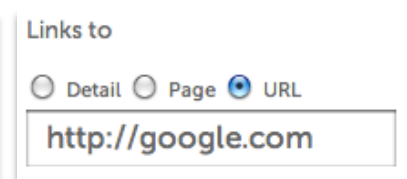
Fig. 12: Image links to an external URL

A special feature of images is that they can link to three different locations and the user is able to choose a different link for each image. The three radio buttons underneath the label "Links to" specify which location an image points to.

"Detail" (Fig. 10) means that the image links to the detail page of an image where the image is shown in original size, which is the default link mode of an image (see I 3.3.6 "Image Overview").

"Page" (Fig. 11) means that the image points to another page of the current user. A drop-down menu is shown where the user can choose the page. This allows the user to create connections between Pages and gives users another way to experiment with the website and to express their ideas. Users could create linked-lists of pages that are in some way connected to each other or they could link pages for showing a development of a task where each page shows one state in the process.

"URL" (Fig. 12) means that the image points to an HTTP URL which could be an external URL, like an entry in the Wikipedia, or it could be another URL from within the Salon website. A scenario could be that users form a group and therefore they create another user. This user then has a page called "Team" where there's an image for each user that links to the user's overview.

Like in the page edit form, all changes that are made in this form need to get confirmed ("Save"-button) and can get discarded ("Cancel"-button).

3.3.8 Page Index

The page index is a list of all the pages the current user owns. Users will get forwarded to this page after the log in.

Fig. 13: A page index containing two pages

Each entry in the list stands for one of the user's pages and gives the user an overview on some facts about this page, like the current cover image and the number of images stored in it, as w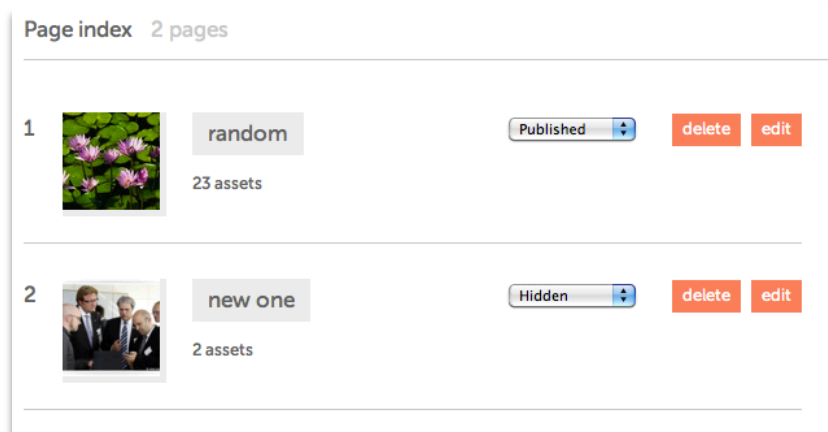ell as the name of each page. Furthermore, the page offers the user the ability to quickly delete pages and the ability to go directly to the edit form for each page.

In contrast to the user overview, this page also shows all pages that have a status of "hidden" or "not published".

Pages basically can have three states:

1) "published": This page is shown in the user's page overview.

2) "hidden": This page is invisible and not accessible for other users.

3) "not published": This page will not be shown in the user's overview but it is accessible for other users that know the URL.

The "not published"-state is useful when a user is currently working on a page and he wants to show the page to others to get a feedback but he does not want to have this page appear in his public overview. To change the publish state, a user simply has to choose the new state from the drop-down list and the page automatically gets updated.

3.3.9 Search
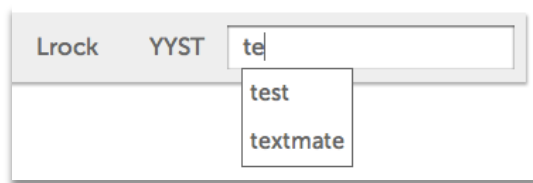
Fig. 14: The auto-complete search field

The search field is located inside the navigation (Fig. 14) to allow the user to quickly search for images on any page. When typing into the field, the system automatically starts a search for matching tags and provides the user with a drop-down menu. By hitting the "enter" key or by choosing one of the items from the list, the user gets redirected to the search result page.

There, all images that match the search term will get displayed and a click on them will navigate the user to the corresponding page.

A scenario for the search could be that a Professor wants to find all images and the associated pages that his students have put online for an exercise. Students could tag their images with a specific tag so that the professor can find them by searching for it. Also a tag search can be used by students to get inspiration by searching for images on a specific topic.

3.4 Visual Design

The visual design has been developed together with Dipl.-Des. Stefan Landrock, Mathias Baer and Dipl.-Inf. Sebastian Deutsch and mainly adopts the style of the Salon prototype. The main requirement for the visual design was to use as less screen space as possible to not distract the user from the custom design of the pages. Therefore, only the navigation, including the search field, and the menu caption are permanently visible.

For the same reason the design itself is also very minimalistic and does not use images and is designed with CSS only. To highlight the elements that belong to the Salon platform (because users could loose track of that on pages with many images), all elements have been decorated with a drop shadow and are always on top of all other elements on a page. In this way, they also remain accessible at any time.

To make it clear to users, which elements on the page can only be seen by users that have the right to edit parts of the page, these elements are highlighted in a red-orange color. This also facilitates to distinguish which visual elements serve to modify the pages. The highlight-color is for example applied to elements like the "edit"-buttons or the "image-size"-widget.

Page owners would normally always see these elements and they would never be able to see their pages as they appear to normal users. For that reason, a "Hide admin"-button has been introduced and is placed next to the menu caption on overview pages. When the user clicks this button it will hide all admin elements so that page owners can get an impression of what their page looks like to normal users.

3.5 Navigation concept
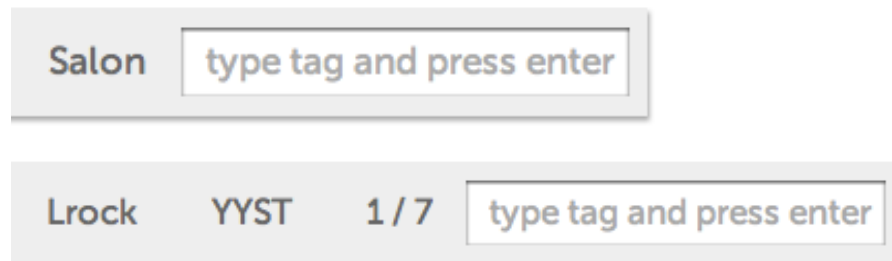
Fig. 15: The navigation of Salon. Above on the index page, underneath on an image page.

The navigation (Fig. 15) in Salon is designed in a breadcrumb-like style[8]. On the index page, the user only sees a caption saying "Salon" which should tell him that he is currently in the most top level of the website. The second element in the navigation is the search bar. Then, when the user goes to the overview of a user, the caption "Salon" is replaced by the name of the user that this page belongs to. Removing the "Salon"-caption should emphasize that Salon is about the users and about the work that they want to present and that it is not about the platform itself. Normally the first element in a breadcrumb navigation brings the user back to the index page, but this is not a scenario that applies to Salon because the index page, intentionally, does not offer more features

---

[8] http://en.wikipedia.org/wiki/Breadcrumb_(navigation), retrieved on 04/16/2011

than any other page in Salon but the list of recently created and edited pages. Searching for images is possible from any page through the search field that also resides in the navigation (see I 3.3.9 "Search"). When a user wants to go back to the index page he can do this via the menu in the top right corner (see I 3.6 "Menu") at any time.

Another element (the name of the current page) gets added to the navigation when the user navigates to a page of a user, and another one (the position of the current image in this set of images) gets added when the user navigates to a specific image. The breadcrumb navigation helps the user to keep track of certain information like the owner of the current page and the page an image belongs to. In that way, these relations do not need to be displayed on every image or every page, which leads to a cleaner and lighter interface. Also the navigation helps a user to quickly jump back to a user's overview without having to navigate there manually with the back button.

3.6 Menu

The menu is consistently placed in the top right corner of each page and is by default not expanded so that it does not take away screen space. For expanding the menu, the user simply needs to hover over it with the mouse or tap it on the screen (on touch-based devices).

There are two states for the menu: a) The user is logged in; b) The user is not logged in.

When the user is not logged in, the menu will have the caption "Sign in/up" which stands for the two most important options that are displayed in the expanded menu. The first point in the menu (Fig. 16) will lead the user to the sign-in form and the second point to the sign-

Fig. 16: Expanded menu when the user is not logged in.

up form. The third point will lead to the "about"-page, that explains the concept of Salon, and the fourth point will lead the user back to the index page.

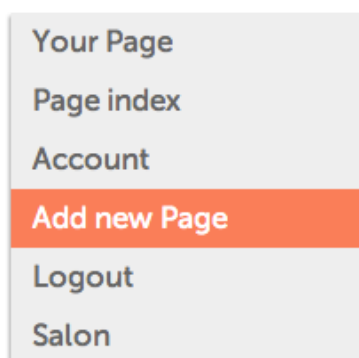Fig. 17: Expanded menu when the user logged in.

The caption for the menu when a user is logged in, is the user's username. The first entry in the expanded menu (Fig. 17) now leads the user to his own overview page, which allows the user to quickly jump there from any other page. The second entry will lead the user to the page index and the third one will lead him to the account page where he

17

can edit details of his account.

The fourth entry is highlighted and it is an interactive entry because when this entry gets clicked (Fig. 18), it will reveal a simple form that lets the user create a new page right from the menu. After submitting the form, the user will get forwarded to the newly created page. By placing the form inside the menu, there is no need to create an own page just for adding a new page and the user is able to create pages no matter on what page he is currently on. He only needs to be logged in.
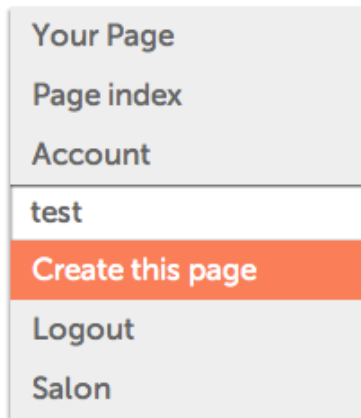
The fifth entry triggers a log-out and a redirect back to the index where also the last entry is leading to.

Fig. 18: Creating a new page
from the menu.

3.7 Drag and Drop

As described before, the Drag&Drop-Feature is one of the most important distinguishing features of Salon. Therefore, there was the need of a good Drag&Drop-implementation in JavaScript. Most major JavaScript libraries offer Drag&Drop-plugins today and in the beginning of development the most prominent libraries have been tried out (namely jQueryUI[9], mootools[10] and script.acoul.us[11]). They all worked great and were very feature-rich including UI-Widgets and many abstractions like automatically sortable tables, but they all lacked support for mobile browsers which is an essential feature-requirement for Salon because it should be usable on all iOs devices, especially on the iPad. Also, when using one of the libraries mentioned above, the whole library had to be included into the project although only the Drag&Drop functionality was needed. This would have added an enormous extra load time, especially for users on mobile devices. Because of these facts I decided to write my own Drag&Drop implementation that supported webkit-mobile browsers as well as desktop browsers.

There are basically two ways to implement a Drag&Drop System with the given DOM[12]-Events in JavaScript:

---

[9] http://jqueryui.com/
[10] http://mootools.net/
[11] http://script.aculo.us/
[12] DOM stands for Document Object Model and is an interface to all nodes of a HTML document - http://www.w3.org/DOM/, retrieved on 04/16/2011

Global System

This implementation is called the global system because the drag-handler, a component that receives all DOM-events and then maps them to drag-events, stores only one DOM element at a time and associates all "global" DOM events with this element. The drag-handler starts when the mousedown-event or the touchstart-event (on touch devices) is fired on an element with the css class "draggable". This element is then saved as the global drag-target together with its current position. All mousemove-/touchmove events that are then fired on the document initiate a movement-delta calculation and a custom drag-event that is fired on the current drag-target.

Drag-events will get fired until a mouseup-event or touchend-event has been fired, which means that the user has stopped dragging an element. This invokes a dragend-event being fired on the drag-target.

Although this method works perfectly on desktop- and also on mobile browsers, it has some disadvantages when it comes to touch-device users. When letting iPad users drag elements around on a test page, they were all confused that they could only drag one element at a time. Also the drag-handler did not work well when multiple touchstart-events were fired. The fact that element-movement is detected by move-events that are fired on the document only allows to track one finger at the same time. Also the iPad users were not only confused but they also thought that the app was not working properly.

Local System

To allow multiple elements to get dragged at the same time, there was the need to not only associate DOM-events to one single element. Each draggable element now needed its own drag-handler and the global mousemove-events could not be used anymore. Instead of the global events in this system, the local move-events of an element are taken to fire drag events, which is why this implementation is called the local system. This means that the drag-handler detects drag events from mousemove/touchmove-events that have been fired on this element. The movement delta is not calculated with one global last-position object but with a last-position object that is stored for each drag-target.

A problem with this technique is that it does not work well on desktop browsers. When moving the mouse very fast, the drag-target looses track of the mousemove-event and the element would stop moving although the mouse is still in movement. Somehow this

problem did not appear on touch-devices so that this technique could still be used on touch devices.

The final implementation of the Drag&Drop-system for Salon uses both techniques and switches to the global system on desktop browsers and to the local system on touch-devices.

By design, the system itself does not alter the positions of the images itself. To make the system as decoupled as possible, this functionality has been delegated to event receivers that then can decide on their own in what way they want to move the elements on the screen (e.g. top/left CSS-attributes or negative margins). This makes it possible to create elements that can only get dragged on one axis (horizontal / vertical) or only in a certain range on the screen.

Drop

Although this functionality is not (yet) used in Salon, the Drag&Drop-system also supports the events "drop" and "drag-over". To let an element receive these events, it only needs to have the CSS class "droppable" assigned. When a drag event is fired, the system looks for elements that can receive a "drag-over" event by matching the current position with the positions of all droppable elements.

This event is useful to give users the feedback that they can drop elements on a specific element e.g. by increasing its size or by changing its color when the user drags over it. When the user drops an element, the underlying "droppable"-element will receive a drop event that includes the current drag-target.

4. Quo vadis Salon?

The development of Salon should not stop after this thesis and there are various additional features planned for the future:

Remix-me

The "Remix-me" feature would allow a user to clone an existing page from another user to then edit it as if it was one of his pages. These remixes would then get listed on the original page and a caption would get added to pages that are remixes so that the original authors would always be mentioned. This feature should be an optional feature for pages and should need to get activated in the page edit form for each page to preserve copyrights of the original content.

A scenario could be that Professors create pages to specific topics and then make an exercise in class that students should remix the current page and add their own ideas to the page. This raises the question how ownership shall be regulated within the remix-system. Should remixers be allowed to delete images from the original page in their remix or should they only be allowed to add images?

Overall this feature could easily boost interaction between users and the virality of the platform.

More Assets

Currently there is only one asset type that can get added to pages: the image. But the backend architecture allows to easily add other sorts of assets like for example texts or sounds. Especially the combination of text assets and image assets could lead to a lot of interesting pages. Texts could link to other pages or they could serve as a description for images.

Animations

To improve the user experience, more animations could get added to Salon. For example, when navigating through the images of a page with the keys, the images could slide-in from the side instead of just suddenly appearing after a key has been pressed.
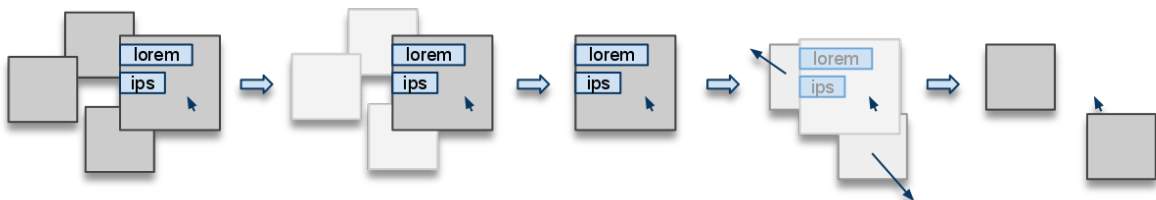
Fig. 19: A proposal for a more complex animation for a page change.

Or a more complex animation (Fig. 19): When selecting a page on a user overview, all other images could fade out. After all other images have been faded out and the new data for the page overview has been fetched, the new images for the page are placed behind the current image. They would then simultaneously fade in and get smoothly moved to their positions while the cover image fades out.

An advantage that comes with this animation is that users are able to see where all the images are placed, even those that would not be visible from the beginning. But this animation could distract the user's attention from what is really important on each page: the images and their arrangement.

Zoom

Pages with a lot of images tend to look cluttered and then images are often dragged out of the visible area of the canvas. Sometimes this leads to several images getting hidden by accident so that users will not notice that there are more images on the page. A zoom feature that lets users zoom in and out of overview pages would help to give users an overview of pages with a lot of content and it could also be another element for users to experiment with (e.g. by hiding images that can only be seen when a user completely zooms out of the page).

Grid Generator

Sometimes it is not necessary to align images on a page in very creative way when a user only wants to simply upload images and to show them to someone else. Currently to align them properly in a grid, a user has to manually drag the images around. The more images there are, the more time is needed to create a nice grid and very often images are not aligned 100% correct because it is hard to align everything manually. To allow the user to simply create grids that are perfectly aligned, a grid generator component could get added to Salon that is visible on overview pages. A user would only have to specify an amount of columns and a padding and the generator would then align the images automatically.

Search for page titles

The search currently only works for tags, but this is a functionality that is likely to get extended in the future to also support the search for page- and image titles. This makes it possible to find pages directly and not just by searching for tags that images in a specific page may have been tagged with. Students could then name their pages according to the exercises of Professors so that Professors can find these pages easier. This also leads to a needed restructure of the search result page that then should also show resulting pages.

Page Customizations

In order to give users more control over the look of their pages, more possibilities to customize pages could get added to Salon. This especially means to allow users to insert custom CSS snippets to each page. In this way users could change colors, hide certain elements or remove all elements that are created by Salon (e.g. the navigation or the menu) to create a completely unique design. Additionally, users may be allowed to insert custom JavaScript code so that for example all templates could get overwritten or

custom animations could get added to a page. Allowing users to add JavaScript to their pages has many security implications that should get examined in detail before this feature will be added to Salon.

5. Evaluation

In conclusion it can be said that the implementation of Salon went very well and that all features that were available in the prototype are also available in this implementation.

This implementation contains even more features than the prototype.
Each page and each image now has a dedicated form to edit its properties, images can have several link targets and the whole website has been implemented as a SPWA which brings a lot advantages in User Experience and makes Salon stand out from other portfolio platforms in the market.

Also an own Drag&Drop-system has been written to better match the requirements of Salon because it is more lightweight and it supports (multi) touch-devices.
The clean and well-structured code base makes it easy to make changes to the system when universities or customers want to change its behavior or when they want to remove a certain feature. Since Salon is a SPWA with a Rails backend it is also very easy to deploy instances of it to the Internet or a university intranet.

The overall User Experience and the unique Drag&Drop-portfolio concept make Salon a website that is fun to use both for users that create content and for users that browse the page.

II Single Page Web Apps

1. Introduction

Single Page Web Apps gained a lot of attention with JavaScript becoming more and more important in the web development tool chain. One of the main reasons for this development is AJAX[13], a technology to asynchronously load content from the server in JavaScript. Most modern web pages use this technology to make their pages more interactive. But by using AJAX as an addition to the normal backend framework a lot of duplicated code is created and the whole website becomes inconsistent because sometimes the page is rendered in the backend and sometimes parts of the page are rendered in the frontend.

Single Page Web Apps are the consistent refinement of the AJAX approach because they shift a lot of traditional backend functions to the frontend so that the rendering is consequently done in the frontend and no duplicate code is being created. This leads to browsers becoming fat clients[14] in contrast to the traditional thin client[15] design in traditional websites. The backend is only used as a web service that serves data in special transport formats which makes the communication a lot more lightweight because the transport format is much more efficient than the transport of complete HTML pages.

1.1 Motivation for Salon

In the beginning, Salon was a standard Ruby on Rails (Rails) application. All views were rendered on the server and a lot of JavaScript code was needed to make the UI as flexible as it is now. The JavaScript code was structured with the help of Backbone.js[16], a JavaScript library that provides frontend developers with Models, Views and Controllers and makes frontend code event-driven. From time to time the frontend code became more important and that quickly led to duplicated code that needed to get implemented in the backend language and in the frontend language. An example: In order to dynamically create images in the page overview, a JavaScript template was used that

---

[13] AJAX stands for "Asynchronous JavaScript and XML"

[14] Fat client means that most business logic is executed on the client rather than on the server and the server is only needed for data retrieval.
http://en.wikipedia.org/wiki/Fat_client, retrieved on 04/16/2011

[15] Thin client means the opposite of fat client: Business logic is executed on the server and the client is only used for presentation and relies heavily on the server.
http://en.wikipedia.org/wiki/Thin_client, retrieved on 04/16/2011

[16] http://documentcloud.github.com/backbone/

looked the same as the ruby template and needed to get maintained in both places. Also parts of the model-logic have been rewritten to enable an easier communication with the backend. More and more of the application-logic moved to the client-side and the code became disordered and it was unclear which parts of the application-logic were implemented in the frontend and which parts in the backend.

To restructure and to cleanup the code-base, Salon has then been rewritten as a Single Page Web App (SPWA). In addition to the code cleanup, this also had some other positive affects on Salon, which are described in II 3. "Benefits".

1.2 Comparison
In the following, the main differences between the traditional (MVC-based) Web App system and the Single Page Web App system will be pointed out by looking in detail at the typical request flow in both systems.

1.2.1 Traditional
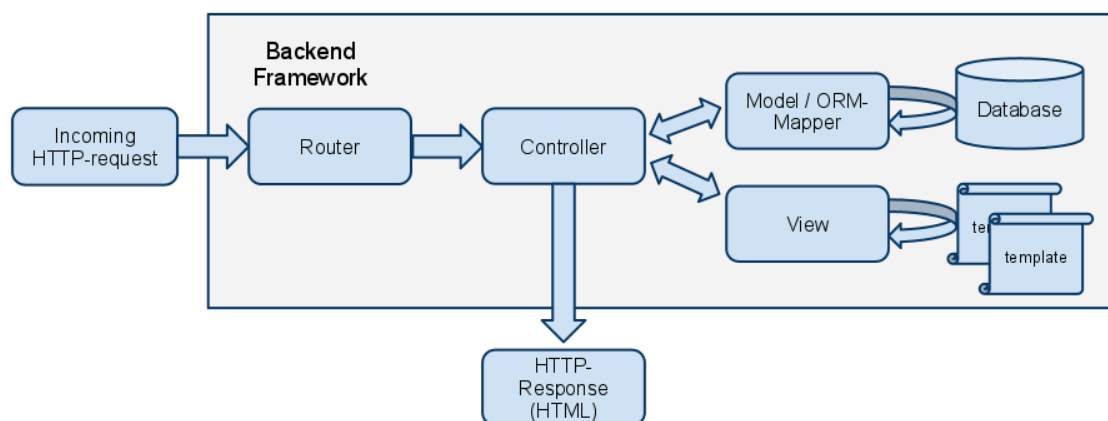
Fig. 20: Traditional request flow with a MVC backend

When a new page is requested, the user's browser sends an HTTP request to the server. This request is then matched to the corresponding controller by a router. The controller then triggers the model-layer to retrieve the necessary data for the request from the underlying database. When the data has been successfully fetched, the controller triggers the view-layer to render the data into the requested view. Views may be constructed from several sub-views that are rendered into one layout file. The rendered layout is then transferred back to the browser.

The browser completely removes the old DOM and re-renders a new DOM from the layout that just has been transferred. This leads to the typical blank screen in the browser when a new page is being requested.

1.2.2 Single Page
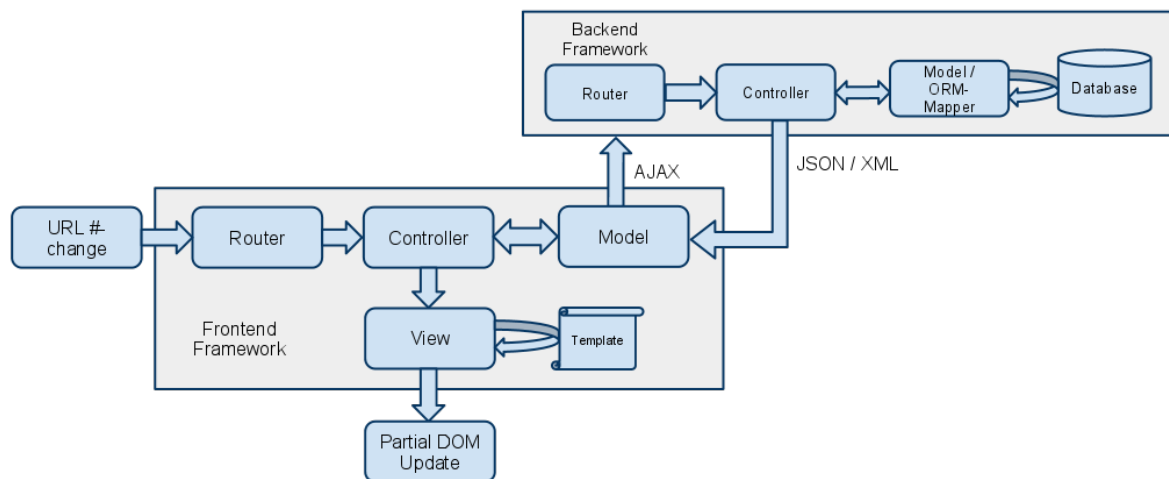
Fig. 21: Request flow in a Single Page Web App with a MVC backend and a MVC frontend

The Single Page request-flow is initialized by a "#-change" of the current URL which gets invoked when the user clicks a link, just as in the traditional approach. But in this case, since only the "#"-part of the URL gets changed, the browser does not create a new HTTP request (see II 2.1 "URLs"). The "#"-change triggers a client-side Router to call a certain Controller function. In this Controller function the necessary Model instances are created, which then automatically make AJAX requests to the backend to fetch the data. In the backend the Single Page request-flow is the same as the traditional request-flow but this time the backend system won't render a HTML view from the data. Instead of letting the server render a complete new layout and transfer it to the client, the data gets serialized into a transport format like for example JSON (see II 3.1 "Speed / Efficiency"). When the data arrives in the frontend framework, the model notifies the controller that the data is available. The controller then takes care of rendering the correct view from the data. Now, instead of replacing the whole DOM, the generated view replaces only a part of the page (e.g. the content panel). The whole request is executed asynchronously, which prevents the screen from turning blank when the new content is being loaded and makes the page reload a lot smoother.

Another difference in the request flow is that in SPWAs the initial request (e.g. "www.mydomain.tld") delivers a complete web app, with all the application-logic in it, and not just a snapshot of it because all the logic has to get transferred to the client.

2. Differences

The new structure in the request-flow, which has been pointed out in the previous chapter, leads to many differences in the development process and the general structure of websites. These differences are described in this chapter.

2.1 URLs

Since browsers automatically handle normal hyperlinks with a GET-request, the URLs in SPWAs look different to normal URLs. They make use of the "#"-symbol that originally was used as an anchor to an element in a HTML page. This is needed on pages like Wikipedia where you have long text articles on one page and you want to point the user to a specific paragraph. The browser's viewport automatically jumps to the element with the given ID (after the "#" symbol).

To prevent a GET-request by the browser, a client-side router only listens to changes in the URL after the "#"-symbol. Changing the "#"-part of the URL works just the same as changing a normal URL. Users only need to click a link with a different "#"-URL and the URL gets updated. Changes to the "#"-part of the URL are being propagated with a "hashchange"-event[17] which unfortunately is not supported by all browsers. In browsers that do not support the "hashchange"-event, client-side routers permanently poll the state of the URL to detect changes. When a change has been detected, the router typically triggers a handler (Controller functions according to Fig. 20) for the new URL-partial. So when the URL before was "mydomain.tld/#index" and it has been changed to "mydomain.tld/#about", the URL-router will trigger a handler that has registered for the "about" partial. This also makes all URLs bookmarkable because the router will render the corresponding views to each URL-partial no matter what page has been displayed before. Initial requests that do not contain a "#"-fragment normally get mapped to an "index" partial and the handler gets called after the page has been loaded although no change is registered.

In HTML5 a "pushState" method has been introduced into the History object[18] that allows developers to completely take control of the browser's behavior when a normal URL changes, and not just the "#"-part. This will make it possible to write SPWA URLs in the same way as normal URLs without a "#"-part in it that may confuse users. But since

---

[17] http://www.whatwg.org/specs/web-apps/current-work/multipage/history.html#event-hashchange, retrieved on 04/16/2011
[18] http://www.whatwg.org/specs/web-apps/current-work/multipage/history.html#dom-history-pushstate, retrieved on 04/16/2011

this method has just recently been introduced into the standard, not all users have switched to browsers that support this method and it may take a while until a big enough percentage of users have switched to compatible browsers, so that for now the "pushState" method cannot be used for SPWA URLs.

2.2 Rendering

A common technique to render views in the backend is to use an abstraction layer called "templating-engine". These engines allow to write views in a mostly HTML-like syntax to improve readability and maintainability over manual string-concatenations in the backend language. Also the syntax makes it easy for designers to create and edit templates on their own, rather than having a backend developer who always needs to implement all changes.

A templating-engine pre-compiles views into functions or string-concatenations, so that the backend can execute them faster and does not need to interpret them at runtime. The pre-compiled functions are mostly automatically generated and developers generally do not need to work in these files.

In Single Page Web Apps the templating-engine resides on the client-side (see Fig. 21). In order to reduce bandwidth, the backend only sends raw data to the clients (in a transportation format like JSON) that can very rapidly get serialized.

The template-engine in the frontend then renders HTML snippets from the data and inserts them into the DOM.

There are several JavaScript implementations of the most used backend templating-engines and they can all compete with their backend implementations when it comes to speed and flexibility. Designers do not need to change their workflow because they still can create templates in the same way as before, assuming that the same syntax can be used in the frontend templating-engine.

2.3 Authentication

State-of-the-art authentication systems are designed to get as easily integrated into a website as possible. Therefore they automatically create views for all authentication actions (sign-up, log-in etc.) that then can get integrated in layout files.

But since in SPWAs the backend views cannot be used, the authentication systems have to get adjusted to support the authentication over AJAX requests. The complexity of these adjustments depends on the modularity and openness of the authentication-systems.

If it is not possible to make changes to the authentication system to support AJAX authentication, for example because it's a closed-source project, the complete authentication part could get moved back to the server-side. This means that all forms that are used for authentication could get served as normal websites instead of client-side views. In this way, all authentication systems in the market could get used.

But a problem of this solution is that there has to be an implementation of the website's layout on the server-side and on the client-side so that all authentication pages look like the rest of the website. This means that layout files have to get maintained in both systems.

2.4 Internationalization

By moving all views to the frontend, the internationalization[19] (i18n) logic has to be moved to the frontend as well. I18n systems in modern web application are very well integrated in the view layer because that's where their functionality is primarily needed. But since in SPWAs all views are rendered in the frontend, these systems can no longer be used.

There are various i18n systems implemented in JavaScript and they have already proven to work well but they all have a very different API to normal i18n systems so that developers would need to first learn a new API before they could use it. For this reason, Salon uses an own i18n system that adopts the API of the Rails i18n system and is therefore easy to use for developers that have already worked with Rails or other backend frameworks.

One disadvantage of client-side i18n systems is that all languages have to get loaded in the initial page request to have a smooth and quick language-change. When it is not important to have a new language to load immediately, the translations files could also be loaded when they are needed. Mostly the language is not changed often by a user, which means that loading the language files only when they are needed seems like a valid method.

2.5 Notifications

Many backend frameworks give developers a simple way to display so called flash messages, which are short messages that are displayed once a request has been finished and the page has been rendered, like "Successfully deleted this item".

---

[19] Internationalization means to provide a website in more than one language and let users decide which language they prefer.

The purpose of these messages is to give the user a feedback on his actions and to remind him on what he just did on the previous page and whether this action has been applied successfully or whether an error occurred during the processing. A typical backend framework provides these messages to the view layer where the messages normally are being rendered into a DOM element.

This technique does not work for SPWAs since the backend does not render parts of the view so there is the need for a new notification system in SPWAs.

A simple component has been implemented for Salon to provide Salon with a notification system that is also easy-to-use in other SPWA projects. The system adds the flash messages to the JSON response on the server-side and a client-side notification
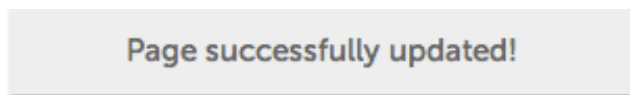

Fig. 22: A notification by the Salon notification system

component, which listens for all incoming AJAX responses, parses the message and displays it at the top of the page. Notifications fade out automatically after some time so they do not get in the way.

Notifications and status indicators are very important for SPWAs and other AJAX-heavy websites because there is no reload of the page that tells the user that something is happening on the page. AJAX requests may take a long time so that users should also get an immediate visual feedback to show the user that the website is working on his input. And since in most cases only parts of the website change, there should be notifications that tell the user exactly what just has happened because the user may not notice minimal changes in the page layout.

2.6 Forms

One of the things that enormously speed up the development when working with a backend framework are form-helpers. They allow developers to rapidly create forms for CRUD-operations[20] on models without having to write much code. Again, these helpers do not work in SPWAs because they generate forms in the backend and create HTML that is used in backend templating-engines.

Writing all forms without such helpers would lead to a lot of duplicated code that is needed for the AJAX requests and the error handling and it is also very time consuming having to write forms from scratch each time. To avoid the unnecessary time consumption and the duplicate code, a client-side form-helper has been developed

---

[20] CRUD stands for Create, Retrieve, Update, and Delete and describes basic operations on Model instances

during the development of Salon. The component works well together with the model classes that have been provided by the already mentioned Backbone.js and automatically takes care of the server communication and validations. The form-helper is additionally capable of pre-filling forms with a model's attributes, it provides hooks to allow developers to override the default behavior (e.g. AJAX calls) and it automatically displays error messages when a user has entered wrong values.

3. Benefits

3.1 Speed / Efficiency

The most important benefit of SPWAs is that they'll speed up a website's performance. They will even make the client-server communication more efficient. These benefits are caused by sending less data to the clients (no HTML, just transport format) and by the less amount of time that is needed by the server to render complex views, because the rendering has been moved to the client-side.

Efficiency is very important, for example, when a lot of clients connect to a website via slower networks or when a website server will have to handle a lot requests per second. The faster a website reacts on user input or the faster it loads, the better will be its user experience.

There are a lot of studies that have investigated the impact of a website's speed on its user experience and they all support the thesis mentioned above. For example, in 2009, Forrester Consulting conducted a study to investigate the behavior of online shoppers[21]. They found out that a page should not take longer than 2 seconds to load or otherwise the user becomes unsatisfied and eventually will stop using the online shop or the user may even switch to another competitor. 52% of the interviewees mentioned in the poll that page speed is one of the most important features for a good online shop.

When Google intentionally slowed down their search results in one of their public experiments[22], they observed a decline of the total number of searches by 0.2% to 0.6%. They found out that the more delay they added to the results, a user would make less searches. When considering the shortness of the delays that Google added to the searches (first 100ms, later up to 400ms), this experiment shows very well how important each millisecond delay can be for the overall user experience on a website.

---

[21] [AKTWOS2009]
[22] [GOSPEE2009]

The simple and efficient design of client-server communication in SPWAs makes them very fast so that the time users have to wait is reduced to a minimum.

3.2 Same Language

Server-side JavaScript has become very popular recently with the development of node.js[23], an event-driven server that allows developers to write all their backend code in JavaScript. Its asynchronous and event-based programming paradigm, I/O operations won't block the server until they're finished and instead an event is fired when data is available, allows the server to handle way more concurrent requests than other (blocking) server technologies. This makes this technology very interesting for high-traffic websites.

Dealing with the same language on both end-points means that developers can share code, for example model-logic, to reduce code duplication and unwanted double-maintenance. This can lead to a faster and more efficient development and developers do not need to change the way they program since it is the same syntax in the frontend and in the backend.

3.3 User Experience

Talking about speed in the context of user-experience means more than just performance of client-server communication[24]. When a website makes heavy calculations for the user, SPWAs won't perform better in calculating the result on the server-side than normal websites. But one weakness of normal websites is that there is no feedback telling the user that a certain operation may take longer. The screen will stay blank or will not change for a long time.

In SPWAs, loading indicators like labels (e.g. "Loading...") or spinning animations are used to give the user an immediate feedback on an action that may take longer. This won't speed up the calculation but it shows the user that the system has registered the input and that the user has to wait. This also prevents users from clicking the same link again, which may even lead to longer response times.

Transitions

Another benefit SPWAs have over normal web pages is that it is possible to have transitions / animations between page changes.

---

[23] http://nodejs.org/
[24] [GOUIME2011]

Transitions and animations are more and more often used in modern web pages to make them feel more dynamic and to make the user have more fun using the page. But a reload on a normal website will break the dynamic impression because the page will simply turn blank on page change until the new page is loaded.

A good example for this behavior is Apple's website. When visiting the index page, only a main focused element, like for example a new iPad, is shown. All other elements like the navigation fade in after a certain delay. But when going to another page, the page turns blank and the new page is being loaded which completely breaks the smooth impression of the index page.

To make the User Experience on a website consistent, page transitions like they are implemented in the Salon overview pages could get added so that the dynamic and interactive impression will not get lost. On Salon overview pages all images fade-out and fade-in when navigating through the different pages. This makes the navigation feel a lot smoother and it also hides loading times (caused both from the AJAX request and the image loading time).

Users normally do not see the transitions actively. It is, when it's done correctly, only a subtle effect. But when transitions are removed from a page, users see that something has changed and that the page does not feel as good as before. Showing users first the transitioning implementation of the overview pages and then an implementation without transitions has proven this. Users thought that the site needed longer to load but they could not tell that the transitions had been removed.

Transitions should not be used too heavily because otherwise the focus in the important parts of a website will get lost and users tend to get annoyed by too long animations.

Sound

Another neat feature that SPWAs offer is that they allow websites to have music playing permanently in the background without stopping when the page changes.

Currently most websites that let users play music suffer from the page-reload problem and do not allow the user to browse the page and listen to the music simultaneously (e.g. http://www.last.fm, http://www.soundcloud.com). Users have to keep at least two tabs/windows of these pages open.

Other websites bypass this problem by opening a dedicated new window only for the music player (e.g. http://www.jamendo.com, http://www.play.fm). Both solutions suffer from the same problem: it is very cumbersome for the user to control the player.

The user has to switch the tab / window or even, when the user has too many tabs / windows open, the user has to search for the player in the browser.

Stopping the player or altering the volume can take quite a while and this delay leads to a bad user experience. In SPWAs, the player can simply get embedded into the page and it will always remain on the same position so that users can easily control it. Of course the player won't stop playing when another page is requested since the reload does not affect the player.
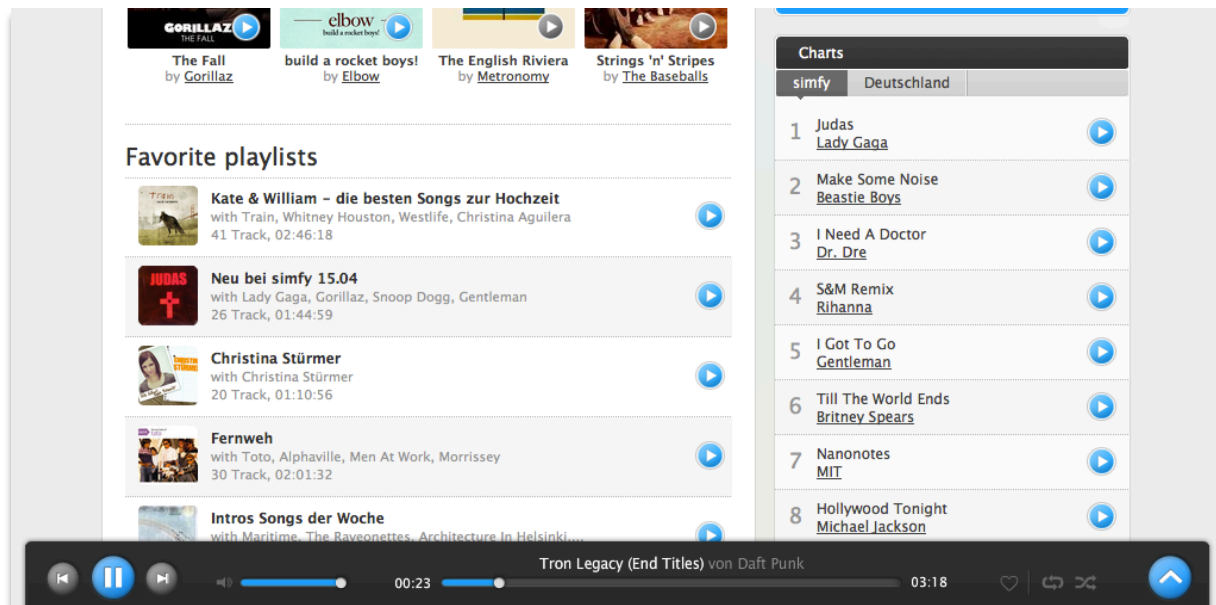

Fig. 23: The music player of www.simfy.de

A good example for the use of SPWAs in a music-context is simfy (see Fig. 23). The player is fixed at the bottom of the page and it remains there when the content of the page changes and the music plays without stopping. In order to avoid that the music player gets in the way while browsing, its size is adjustable.

4. Problems

4.1 Search Engine Optimization

Search Engine Optimization is very important for modern websites in order to get a good ranking in search results from Google or any other search engine.

Search engines build their indexes with so called Web crawlers[25] that process the contents of websites to get an understanding to what topics they are related. Web crawlers automatically follow links on web pages to create relations between websites and to find out the importance of websites by counting the links that link back to a certain page.

---

[25] http://en.wikipedia.org/wiki/Web_crawler, retrieved on 04/16/2011

They are built to crawl rapidly through many websites, which means that the basic crawlers neither load images nor CSS-, nor JavaScript files to load pages faster. This has a negative impact on SPWAs because the content would not get correctly indexed or even not get indexed at all because the client-side JavaScript-based URL-router would not get started when a Web crawler is on the website, since crawlers do not execute JavaScript.

Furthermore, if the start page of a SPWA was generated by the JavaScript templating system, the page wouldn't even get added to any search engine index because the crawler would just see a blank HTML page. This would make SPWAs unusable for projects that need to have a good ranking in search engines.

But there are several ways circumvent this problem.

Google proposed a technique that lets their crawlers index an SPWA[26].

When their crawler finds a URL in the SPWA-typical "#!"-style it will request a special URL on the website's server that should return a HTML snapshot of the requested page. This snapshot should represent the content of the page that should get indexed. As an example, a request to the URL "mydomain.tld/#!/test" would create a Web crawler request to "mydomain.tld/?_escaped_fragment_=test" and the server should respond with the HTML snapshot.

This solution can easily lead to a lot of duplicated code since a router is needed in the backend that has to work exactly the same way as the router in the frontend. Additionally, often view code might get duplicated because templates have to get maintained in the frontend and in the backend.

One thing to keep in mind with this technique is that currently only the Google Web crawler supports the advanced URL scheme and none of the other competitors such as Bing and Yahoo do.

Another technique to circumvent the search engine problem works especially well for community pages where there is a difference between the site a user sees when he is logged in and the site he sees when he is not logged in (Web crawlers normally do not log in to communities). In that case, all public pages could get served from the backend so that Web crawlers can index them easily. Internal pages would then get served as SPWA since these pages should not get indexed anyway. The extra effort that is needed

---

[26] [GOMAKE2011]

for this technique is reasonable since only few pages (e.g. index, about, pricing, contact) need a backend view and most of the client-side code does not need to get duplicated.

4.2 New Tools needed

When working on a normal web application there are tons of frameworks and tools that help throughout the whole development-, deployment- and maintenance process. These tools have been optimized over the past years and developers have learnt how to become most productive with these tools.

Such integrated tools and frameworks do not exist for SPWAs yet. There are tools that fit one specific part of the process like compiling the Views or giving the app a MVC structure, but developers have to connect these tools manually. In the backend, the old tools can still get used but they do not support the client-side development that much. New tools have to get developed so that developers do not have to struggle with their development environment on every new project and can clearly focus on working on the project.

A first step for new tools has been made with brunch[27], a tool chain that combines all the needed technologies for the client-side development into one command line call. This helps very much to speed up the development on SPWAs, but developers still have to develop the back-end with another tool because brunch currently is backend-agnostic and does not provide any backend helpers. To ease development even more, there is definitely the need for tools that support the development in the frontend as well as in the backend at the same time.

4.3 Exposure of Business Logic / Sensitive Data

Moving the complete business-logic to the client-side means that every user who knows how to display the source of a website can easily see how a website / a business works. Modern browsers even further have integrated tools that allow users to deeply inspect a website's code and to especially monitor AJAX requests. The problem that users can cheat on a website's code is not a problem that only occurs in SPWAs, it's generally a problem of every website, but since almost all business-logic resides in the user's browser, for SPWAs this problem has a much bigger impact.

If a website deals with sensitive data (bank accounts, credit card numbers...), developers will need to make sure that none of the code on the front-end exposes security holes that could harm the website's users. Generally all privacy relevant operations should be done

---

[27] https://github.com/brunch/brunch

on the server-side and to secure the client-server communication, HTTPS should be used instead of normal HTTP.

Another important thing is to double-check login-states and admin-rights on the server and do not let only the client-side handle it. Assume the following scenario: An app has a global user object with a Boolean field called "admin". The app displays editable elements according to the value of the admin field in the user object. A user could now simply open up the JavaScript console of his browser and change the value of the admin field to get access to all editable elements.

A way to protect websites from this attack is to add a server-side generated field in the responses that adds information on the rights of the current user. The app then switches the editable elements according to the response rather than according to a front-end object that could have been edited by a user. But still the backend should always double-check the user's rights on incoming requests.

Another method to secure a website's code is to obfuscate it before it gets deployed to the server. In that way, the code is not as readable as before and attackers would need to put a lot more effort in understanding the obfuscated code to harm a website. A nice side effect that usually comes with code-obfuscation in JavaScript, is that the code also gets compressed so that not only the website gets secured but also the load time gets reduced.

4.4 Accessibility

SPWAs only work when clients have JavaScript enabled and this makes them useless for users that either have disabled JavaScript on purpose, e.g. to increase the browser performance, or for users that are browsing at work and that are not allowed to have their browsers execute JavaScript.

According to statistics by Yahoo[28], the percentage of users that have disabled JavaScript is approximately 1%. This number may seem low at first but it means that when a page has one million page-visits a month, ten thousand of them are not able to use the page. There is the possibility to display a message that demands the user to activate JavaScript with the "noscript-tag" in HTML[29], but users that have actively disabled JavaScript did this for a reason and may simply decide not to use the site. Developers have to be aware of the numbers mentioned above when creating JavaScript-heavy applications.

---

[28] [YAHOWM2010]

[29] http://www.w3.org/TR/html4/interact/scripts.html#h-18.3.1, retrieved on 04/16/2011

Another problem with dynamically generated layouts is that it is hard for screen readers to interpret these layouts semantically. Although 98% of screen reader users have JavaScript enabled[30], the screen readers are not able to properly interpret dynamic DOM changes that are used to "switch" pages in SPWAs.

The Web Accessibility Initiative (WAI), an organization that creates recommendations for web developers to make websites more accessible, is aware of the problems described above and created a guideline called "WAI-ARIA"[31] that aims to provide solutions to developers and its publication is expected for the middle of 2011. The current working draft[32] describes a set of additional events that should get fired for example when a DOM element's text-content changes or an element is being hidden. These events are processed by screen readers to create a semantically correct feedback for the user. Maybe SPWAs will become more accessible with the introduction of these new events.

---

[30] [WESCRE2010]

[31] http://www.w3.org/WAI/intro/aria.php, retrieved on 04/16/2011

[32] http://www.w3.org/TR/wai-aria-implementation, retrieved on 04/16/2011

5. Conclusion

SPWAs really can help to make a website feel better and to give users a better experience. They allow a lot of new interaction concepts and more dynamic sites than we have today. AJAX was a first step to make websites feel more fluid but SPWAs bring the whole concept to a next level by giving the ability to get a completely fluid navigation and transition system. Now there is the possibility to create websites that do not look and feel like normal websites and actually are fun to interact with.

The concept has already been taken over by big companies like Google (Google Mail Chat / Client-side routing), Facebook (Facebook Chat / Facebook Messages / Content is replaced inline, no full page reload) and others to make parts of their websites more dynamic and I think that there will be more and more companies that will take over the technique.

In my opinion, the rise in the interest for Node.js will also result in more companies switching to the SPWA-idiom. Not just because of its positive impact on the User Experience but also because of the ability to share code between the client and the server. And since there are (currently) no big web frameworks, like Rails, for Node.js it may be easier and faster for developers to use tools like brunch to program websites with a simple REST-based Node.js backend.

But since SPWAs use a lot of very new technologies, it takes time for developers to learn and adopt all needed tools before they can start developing. Also, as mentioned in II 4.2 "New Tools needed", there are no tools that cover the complete tool chain and several tools have to get connected manually which slows down development enormously. Especially when developers do not have experience in programming with JavaScript, the learning process can take a while because of the asynchronous language parts that are not common in most other backend languages.

Before deciding to develop a website as a SPWA, developers should take account of the negative points mentioned in II 4. Problems and decide whether it's okay to not get ranked high in search engines (without a lot of extra effort) or whether it's okay to expose the business logic to the user. Only if developers do not think that these negative impacts will harm the success of the website, they should decide for a SPWA.

Furthermore, if developers decide to port an already running and established website, they should think twice about switching, because the page may already be well-indexed in the major search engines and all links may lead to a dead-end after the rewrite of the page. This is exactly what happened to gawker.com, a well established "media news and

gossip" blog[33]. The developers of gawker.com rewrote the whole blog as a SPWA in February 2011 and after they had published the new version, all indexed links were broken so that the number of unique visitors dropped by 50%[34] and a lot of users wrote bad reviews about the website.

In conclusion, there is no single answer whether a website should be implemented as a SPWA or as a normal website. It always depends on the website's focus. If the website has to meet the special accessibility requirements, it would be easier to realize the website as a normal website because it is hard to make SPWAs accessible as shown in II 4.4 "Accessibility". But when it is important for a website to have a good User Experience and the User Experience is more important than a good ranking in several search engines, the implementation of the website as a SPWA is the right decision.

---

[33] http://en.wikipedia.org/wiki/Gawker, retrieved on 04/16/2011
[34] http://techcrunch.com/2011/02/17/gawker-redesign, retrieved on 04/16/2011

Literature Index

[AKTWOS2009] - Akamai Technologies, Inc (2009) - http://www.akamai.com/html/about/press/releases/2009/press_091409.html, retrieved on 04/16/2011

[EHCASS2011] - Eben Hewitt (2011) "Cassandra - The Definitive Guide" (first edition), Chapter: "Appendix: The Nonrelational Landscape", O'Reilly Media, Inc.

[GOMAKE2011] – Google, Inc (n.d.) "Making AJAX Applications Crawlable" - http://code.google.com/intl/de/web/ajaxcrawling/docs/getting-started.html, retrieved on 04/16/2011

[GOUIME2011] - Roma Shah of Google, Inc (n.d.) "UI messaging and perceived latency" - http://code.google.com/intl/de/speed/articles/usability-latency.html, retrieved on 04/16/2011

[GOSPEE2009] – Jake Brutlag of Google, Inc (2009) "Speed Matters for Google Web Search" - http://code.google.com/intl/de/speed/files/delayexp.pdf, retrieved on 04/16/2011

[MFPATT2002] - Martin Fowler (2002) "Patterns of Enterprise Application Architecture", Addison-Wesley

[RFARCH2000] - Roy Fielding (2000) "Architectural Styles and the Design of Network-based Software Architectures", Chapter: Representational State Transfer (REST), http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm, retrieved on 04/15/2011

[WESCRE2010] – WebAIM (2010) "Screen Reader User Survey #3 Results" - http://webaim.org/projects/screenreadersurvey3/#javascript, retrieved on 04/15/2011

[YAHOWM2010] – Nicholas C. Zakas of Yahoo! Inc. (2010) "How many users have JavaScript disabled?" - http://developer.yahoo.com/blogs/ydn/posts/2010/10/how-many-users-have-javascript-disabled, retrieved on 04/15/2011