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

Salon

1. Overview

Salon is a web-based system that allows its users to create pages and to upload images onto these pages. On a first sight this functionality may not look very innovative since there are millions of services on the Internet that allow the user to upload images. But the main improvement that Salon offers that other services don't offer is that registered users are able to fully control the way their images are presented to the visitors of their pages. All images are placed on a canvas and can freely be 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.

2. The idea

Dipl. Inf. Sebastian Deutsch and Dipl. Des. Stefan Landrock developed the basic idea behind Salon when they were given the chance to take over university courses at HFG in 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 completely rewrite and to extend the features of Salon so that it could easily be set up for other universities.

3. Implementation

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 that was chosen because of its flexibility

---

[1] http://rubyonrails.org/
[2] http://betterexplained.com/articles/intermediate-rails-understanding-models-views-and-controllers/
[3] http://www.mongodb.org/

(document-oriented databases are schema free[4]) and its very good integration into Rails.

Salon does not make use of the frontend tools of Rails because the frontend is designed to work as a Single Page Web App (SPWA) and therefore all Rails frontend tools have been replaced with tools that are written in JavaScript so that they could get executed in the browser (see SPWA#intro).

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).

3.2 Data Model
[ADD schema image here]
The underlying data structure of Salon is rather simple. There are users that are used for authentication and have basic properties like a username and a password. Pages are associated to users in a one-to-many relationship, which means that users can have as many pages as they want and each page belongs to only one 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. Asset 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 deriving image from asset is to allow other assets like for example texts in the future (see salon#quovadis) and to provide all derivations with the needed properties to have a valid asset. The image then only needs to save special properties like the image files and its display sizes.

All assets have a list of tags that are associated in a many-to-many relationship which means that a tag can belong to many assets and assets are able to reference many tags. Tags are used in the search (see salon#pages#search).

3.3 Navigation
[ADD screenshot of navi here (initial and completely expanded version)]
The navigation in Salon is designed in a breadcrumb-like style[7]. When first visiting the website, the user only sees a caption saying "Salon" which should tell him that he is

---

[4] http://en.wikipedia.org/wiki/Document-oriented_database
[5] http://de.wikipedia.org/wiki/Representational_State_Transfer
[6] http://en.wikipedia.org/wiki/JavaScript_Object_Notation
[7] http://en.wikipedia.org/wiki/Breadcrumb_(navigation)

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 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, doesn't offer more features 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 (more on search in [x.y]). If a user wants to go back to the index page he can do this via the menu in the top right corner (more on the Menu in [x.y]) at any time. Another element, the name of the current page, is 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, is added when the user navigates to a specific image of a page. 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 don't 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 manually navigate there with the back button.

3.4 Menu

The menu is consistently placed in the top right corner of each page and is by default not expanded so that it doesn't unnecessarily take away screen space. To expand 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.

[add screenshot of menu when 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 (see Fig. XX) will lead the user to the sign-in form and the second point to the sign-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.

[add screenshot of menu when user is logged in]

The caption for the menu when a user is logged is its username. The first entry in the expanded menu 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 (more on that here x.x) and the third one will lead the user to the account page where he can edit details of his account.

The fourth entry is highlighted and it is an interactive entry because when this entry gets clicked it will reveal a simple form that lets the user create a new page right from the menu [show screenshot with the states of the form]. 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.

3.5 Pages

3.5.1 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 (more on search here [x.x]).

3.5.2 Registration and Account

…

3.5.3 User Overview

[add screenshot of a simple user overview]

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 just by dragging them around (more on DnD here [x.x]). 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 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 theses changes will all automatically get saved to the server without the need for the user to initiate the save-process.

To highlight the importance of the images and especially their arrangement there are no further information displayed on top of each image. This is also done so that text elements don't clutter or disturb arrangements that contain a lot of images.

[add screenshot of two hover states (logged-in, not logged-in)]

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 of the page and the number of assets that are contained in this page will then fade in and the image gets a half-lucent overlay to highlight which image currently is being hovered. There is the need for the half-lucent overlay because when many pages are placed in the same page it is hard to find out which of the images just has 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. 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. To drag the whole page the user simply needs to drag the background and all other images will get moved accordingly.

3.5.3 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. As 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.

[add screenshot of images with special icons]

Besides that, there are subtle changes to some of the images. They have special icons that should indicate that they don't link to the image page but to an external page (see

Fig.YY e.g. http://google.com) or to an another page of this user (e.g. test). (More on cross-references of images in Image#edit[])

Also there is another caption right underneath the navigation that allows the user to quickly jump to edit form of this page.

[add screenshot of upload progress]

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 will open up that shows the user how many files are left to get uploaded and the images will after the upload directly get added to the page so that the user can work with them on the page right away.

3.5.4 Page Edit Form
[add screenshot of 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[8] 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 to edit and delete 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 automatically changes.

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.

Other than on the overview pages the user here has to manually save changes with the buttons that are placed directly under the navigation. There also is 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.

3.5.5 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 images' center lies on top of the pages' center. Like on the other pages, the image here can also get dragged around which is handy for images that are bigger than the browser screen so users can see the rest of each image by dragging it around. The position of an image is not saved to the

---

[8] http://en.wikipedia.org/wiki/Slug_(web_publishing)

server because the main focus on this page should not lie on a specific arrangement but on 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 quickly get an overview over all images.

3.5.6 Image Edit Form

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

Tags can simply get added to the image 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.

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.

[add mini screenshot for each state]

"Detail" 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 image#show for more info on that).

"Page" 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 to show a development of a task where each page shows one state.

"URL" means that the image points to a 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).

3.5.7 Page Index

The page index is a list of all the pages the current user owns and it is the page a user is forwarded to after the login.

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 well as the name of each page. Furthermore, the page offers the user the ability to quickly delete pages and the ability to directly go to the edit form for each page.

In comparison to the user overview where the user also is able to delete and edit pages, 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 but 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.

3.5.7 Search

[add screenshot of drop down menu]

The search field is located inside the navigation 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 drop-down. By hitting the "enter" key or by choosing one of the items from the list the user gets redirected to the search result page. Here 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 prof wants to find all images and the associated pages that this students have put online for an exercise. Students could tag their images with a specific tag so that the prof 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.

3.6 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, mootools and script.acoul.us). 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. Also when using one of the libraries mentioned above one had to include the whole library into the project although only the Drag&Drop functionality was needed. This would add an enormous extra load time especially for users on mobile devices.

Because of these facts I decided to write an own Drag&Drop implementation that supported webkit-mobile browsers as well as desktop browsers. There are basically two ways implement a Drag&Drop System with the given DOM-Events in JavaScript.

Global

This implementation is called to global method because the drag-handler, a component that receives all events and 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 (touchstart on webkit-mobile) 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 or touchend 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 browsers and also on mobile browsers it has some downsides when it comes to touch-device users. When letting iPad users drag elements around a test page they were all confused that they could only drag one element at a time. Also the drag-handler didn't 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

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 are taken to fire drag events. This means that the drag-handler detects drag events from mousemove/touchmove-events that have been fired on this element. The movement delta is calculated not with one global last-position 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 lost track of the mousemove-event and the element would stop moving although the mouse was 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 solution 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 a feedback that they can drop elements on this element e.g. by increasing its size or by changing its color. If the user drops an element the underlying "droppable"-element will receive the 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 Profs 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 whether it should be allowed to the remixer to delete images from the original author.

Overall this feature could boost interaction between users and could be an element that is fun to use.

More Assets

Currently there is only one asset type that can get added to pages: the image. But the backend design allows to easily add other sorts of assets 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 navigation 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.

[Add animation graphic here]

Or a more complex animation: 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 get 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 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).

Layout Generator
…

Search for page titles
…

5 Evaluation
…

Single Page Web Apps

1. Introduction

1.1 Motivation for Salon

In the beginning Salon was a standard Ruby on 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[9] a JavaScript library that gives you Models, Views and Controllers and lets you write event-driven frontend-code. Quickly that lead 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 looked the same as the ruby template. Also parts of the Model have been rewritten logic to enable an easier communication with the backend. More and more of the application logic moved to the client-side and so I decided to rewrite Salon as a Single

---

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

Page Web App because I didn't want to have to maintain application logic on the backend and on the frontend at the same time and because I wanted to profit form the other advantages that Single Page Web Apps offer over normal websites.

1.2 Comparison

Single Page Web Apps gained a lot of attention with JavaScript becoming more and more important in the web development tool chain. The AJAX technology is a main reason for this development because on-site DOM manipulation could only be done with JavaScript in the most browsers. Single Page Web Apps take this approach to a next level by shifting a lot of traditional backend functions to the frontend. In the following I will point out the main differences between the traditional (MVC-based) Web App system and Single Page Web Apps by analyzing a typical request flow in both systems.

1.2.1 Traditional
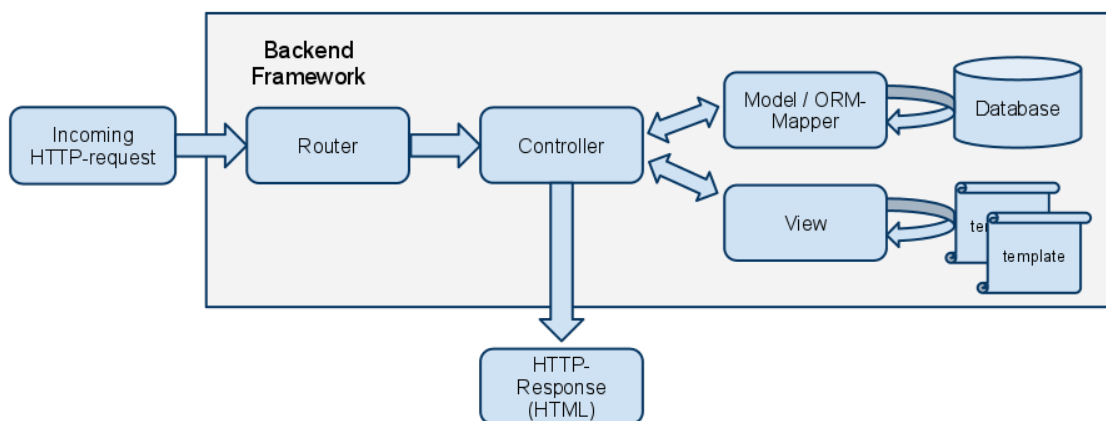
A HTTP request is matched to the corresponding controller by a router. This controller then triggers the Model-layer to retrieve the necessary data for the request from the underlying database. When the data has been successfully fetched, the controller triggers the View-layer to render the data into the requested View. This data is then being transferred to the browser of the user and the current DOM is replaced with the transferred HTML page.

1.2.2 Single Page
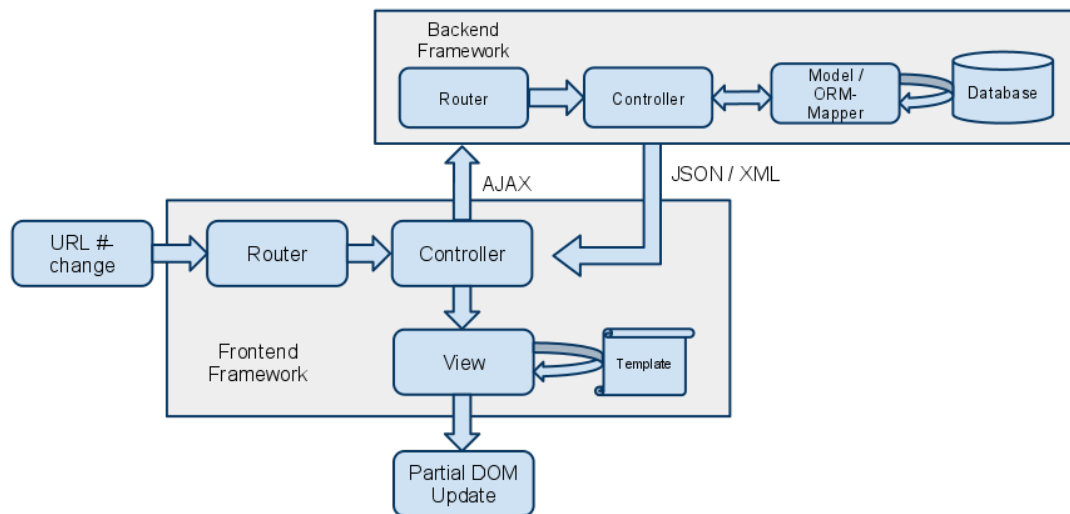
The single-page request-flow is the same as the traditional request-flow until it comes to the rendering of views. Instead of letting the server render a complete new layout and transfer it to the client, the fetched data is serialized into a transport format (JSON, XML...) and the client takes care of rendering the part of the DOM that has changed.
The main differentiation between the two systems is the initial request to the server. In the traditional system you would generate a normal HTML layout and hyperlinks on that page would send GET requests to the server which then would cause a rerendering of the whole page.
In single page web apps the initial request delivers the complete web app and not just a snapshot of it. When the app is initialized a fronted-router takes care of rendering the correct JavaScript view. All requests (e.g. links clicked) will then automatically be passed to the frontend-controller that connects JavaScript Models and JavaScript Views.

2 Differences

2.1 URLs
Since browsers automatically handle hyperlinks with a GET-request, the URLs in single page web apps look different to normal URLs. They make use of the #-symbol that originally was used as an anchor to an element with an ID 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 viewport automatically jumps to the element with the given ID if there is one. [mention push state]
To prevent the get-request the client side router listens to changes in the URL, especially for changes on the part after the # and then triggers a handler for this url-partial. This

also makes all URLs bookmarkable since the router will render the corresponding views to each URL-partial no matter what page you're coming from.

A typical URL would look like this: http://mydomain.tld/#/username/page_slug.

2.2 Rendering

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

A templating engine pre-compiles your views into functions or string-concatenations so that the backend can execute them faster and doesn't need to interpret them at runtime. Typical templating systems for the backend are ERB[10], Haml and Mustache.

In Single Page Web Apps you don't use a templating system in the backend because you don't want to transfer HTML to the client. Only raw data is transferred to the clients. This data mostly doesn't need to be rendered by a templating system as most backend frameworks offer a way to very fast serialize data into a transport format like JSON or XML.

For the same reasons as mentioned above (readability, maintainability), a templating system is a must to have on the frontend side. There are several implementations of the most used templating systems in JavaScript and they all can compete in manners of speed and flexibility with their backend implementations. In case of Eco, a templating system that mimics ERB and is implemented in CoffeeScript, you can even take existing ERB templates and use them on the frontend without needing to change them.

2.3 Authentication

Authentication is something that still has to be done on the server-side. [mention why, sessions and stuff] But enabling authentication in your web app is a not so trivial task. State-of-the-art authentication systems like Devise are developed to get as easily integrated into your web page as possible. Therefore they offer view-partials for all authentication actions (sign up, log in etc.) that you can integrate in your layout files and they will work out of the box. But you can't use these views in a single page web app and you have to rewrite them and the corresponding controllers to enable authentication via

---

[10] http://en.wikipedia.org/wiki/ERuby

AJAX. Rewriting most of the controller code can take a long time and one should, before starting to develop, very well decide on the authentication system one is going to use. If there's no good authentication solution available one could also hand over all authentication actions to the server and let it render the forms. In this way you can use all authentication systems in the market and you don't have to worry about AJAX authentication. The only problem with this solution is that you have to also provide a server-side layout to let your authentication pages look like the rest of your application. But the effort in maintaining a second layout file is nothing compared to rewriting the controllers especially when you need to upgrade the authentication system and there were changes that make your controllers malfunction.

2.4 Internationalization

By moving all views to the front-end you also have to move all internationalization (i18n) logic to the front-end. I18n systems in modern web application systems are very well integrated in the View layer because that's where their functionality is mainly needed. But since now in SPWAs all Views are rendered on the front-end we can no longer use the back-end i18n system. There are various i18n implemented in JavaScript but I wanted to have a system that has the same API as the i18n implementation of Ruby on Rails so that all my old templates could be used without having to change them. It should not just have the same API as the RoR i18n system but also it should have the same format for translation files, which in RoR's case is YAML[11] so that I also should not have to rewrite existing translation files.

My implementation of the RoR i18n system supports the normal and the shorthand calls ("translate PARAMS", "t PARAMS") as well as the string access notation for the specific translations (e.g. "user.create.success"). It can be used as a full replacement for the RoR system. Other than the RoR system one needs to specifically import the translation files on the first load, they won't get loaded on default. The source code is licensed under the MIT license and is hosted on Github[12].

2.5 Notifications

Many backend frameworks give developers a simple way to display so called flash messages. Messages that should be displayed once a request has been finished and the page has been rendered, like "Successfully deleted this item". The purpose of these

---

[11] http://en.wikipedia.org/wiki/YAML
[12] https://github.com/janmonschke/International-Coffee

messages is to give the user a feedback to his action because maybe the user just got redirected and the message should remind him that he got redirected because he deleted the page that he was on or maybe the message should show him that the system has successfully finished his task but the page that he is on doesn't give any visual feedback that something changed e.g. "Settings saved successfully". A backend framework would provide these messages to the view layer where the message normally is being rendered into a DOM element in the layout.

This technique doesn't work for SPWAs since the backend doesn't render parts of the view so I wrote an own notification system. The system adds the flash messages to the JSON response and a client-side notification component, that listens for all incoming AJAX responses, parses the message and displays it accordingly (distinguishes between success and error messages). 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 one should always give the user an immediate visual feedback of any kind that the site has registered his action. And since in most cases only parts of the website change there should be notification that tell the user what just has happened because he may not notice minimal changes in the page layout.

2.6 Forms

One of the thing that speeds up the development enormously when working with a backend framework are form helpers. They allow you to rapidly create forms for CRUD operations on models without having to write much code. Again these helpers don't work in SPWAs since they generate forms in the backend and create HTML that is used in backend templating engines.

Writing all forms without these helpers would lead to a lot of duplicated code that is needed for the AJAX requests and the error handling. So one of the most important tools that have been written with the development of Salon is a client-side form-helper that allows developers to quickly create forms that automatically take care of client-server communication. The helper is capable of pre-fill forms with a model's attributes, provides hooks to allow a developer to override the default behavior (e.g. AJAX calls) and automatically displays errors 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 your website performance. Even more: They make the client-server communication more efficient. Speed comes with less data being sent to the clients and less time that is needed by the server to render complex views. Efficiency is very important e.g. when you know that a lot of your clients connect via slower networks or when your 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 is its user experience. There are a lot of studies that investigated the impact of a website's speed to 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[13]. 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 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[14], they observed a decline of the total number of searches by 0.2% to 0.6%. The more delay they added to the results, the lesser searches would be made by a user. By regarding how short delays 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.

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

3.2 Same Language

With SPWAs you get the chance to eventually use the same language in the frontend as you use in the backend: JavaScript.

Server-side JavaScript has become very popular recently with the development of node.js, an event-driven server that allows you to write all your backend code in JavaScript. Its event-based programming paradigm, I/O operations won't block the server until they're finished, instead an event is fired when data is available, allows the

---

[13] http://www.akamai.com/html/about/press/releases/2009/press_091409.html
[14] http://googleresearch.blogspot.com/2009/06/speed-matters.html

server to handle way more concurrent request than other (blocking) server technologies. [add usage statistics for node]

Dealing with the same language on both end-points means that you can share code to reduce code duplication and unwanted double-maintenance.

3.3 User Experience

But talking about speed in the context of user-experience means more than just performance of client-server communication[15]. Say you have a website that does heavy calculations for the user. SPWAs won't perform better in calculations on the server side than normal websites. But one weakness of normal web pages is that there won't be a feedback that tells the user that it takes a longer time to generate the next page other than a long break until the next page has loaded completely. 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.

Animations / Transitions 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. To make the user-experience on a website consistent one could add page transitions like they are implemented in the salon canvas views. All images will fade-out and fade-in when navigating through the different pages and user overviews. This makes the navigation feel a lot smoother and it also hides loading times (both from the server request and each image) from the user.

- mention apple.com: good try, nice fading front page but changing a page leads to staggering impression

---

[15] http://code.google.com/intl/de/speed/articles/usability-latency.html

Sound

Furthermore a not so important but maybe pretty neat feature that SPWAs offer is that they allow you to have music play in the background without stopping when the page changes. Currently most websites that let users play music either suffer from this problem and don't allow the user to simultaneously browse the page and listen to the music they offer (e.g. http://www.last.fm, http://www.soundcloud.com) so that users have to keep at least two tabs/windows of these pages open or websites bypass this problem by opening a dedicated new window only for the 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 to many tabs / windows open, search for the player. Stopping the player or altering the volume can take quite a while and this delay leads to a bad user experience. With SPWAs you can simply embed the player into the page and it will always remain on the same position so that users can easily control it. A good example for the use of SPWAs in a music-context is simfy (http://www.simfy.de)[add screenshot, maybe with comparison to other sites mentioned above]. The player is fixed at the bottom of the page and it remains there when the content of the page changes. To not get in the way while browsing the page, the player has an adjustable size.

4 Problems

4.1 Search Engine Optimization

Search Engine Optimization is very important for modern websites 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[16] 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 lead to a certain page. They are built to rapidly crawl through many websites which means that the basic crawlers neither load images nor CSS files, nor JavaScript files to improve the load time. This has a negative impact on SPWAs because the content would not be 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 they don't run JavaScript. Furthermore if the start

---

[16] http://en.wikipedia.org/wiki/Web_crawler

page of your website is also generated by the JavaScript templating system your page wouldn't even get added to any search engine index because the crawler would just see a blank HTML page. Until Web crawlers will properly run JavaScript, SPWAs will not properly get indexed. This makes them for now unusable for client projects that need to have a good ranking in search engines. But there are ways to go around this problem: Google proposed a technique that let's their crawlers index an SPWA[17]. When their crawler finds a URL with in typical #!-style it will request a special URL on your server that should return a HTML snapshot of the requested page that represents the content to be indexed. So a request to 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 you need to have a router in the backend that needs to work exactly like the one in your front-end to map the URLs that the Web crawler created. Also you might in addition need to duplicate view-code because you often can't use the same view files in the frontend as 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.

Another method to have a SPWA indexed in search engine rankings works especially 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. In that case you could serve all public pages, which might not be as many as internal pages (index, about, pricing, help etc...), from the backend so that they easily can get indexed by Web crawlers because you don't want to have the internal pages to get indexed anyway. The extra effort that is needed for this technique is reasonable since only few pages need a backend view and most of the client side code doesn't need to get duplicated.
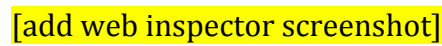
4.2 New Tools needed

When you want to create a normal web application there are tons of frameworks and tools that help you 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.

---

[17] http://code.google.com/intl/de/web/ajaxcrawling/docs/getting-started.html

There are no such integrated tools and frameworks for SPWAs yet. There are tools that fit one specific part of the process like compiling the Views (Eco) or giving you a MVC structure in your app (backbonejs) but as a developer you have to connect these tools manually which can be quite time consuming. For the backend you still can use the old tools but they won't help you very much for your client-side development. New tools have to get developed so you don't have to struggle with your 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[18], a tool-chain that combines all the needed technologies on your client-side into one command line call. This very much helps to speed up the development on your SPWA but you still have to develop your back-end with another tool because brunch currently is backend agnostic and doesn't provide any backend helpers. To ease development even more there is definitely the need for tools that also help with your backend.

4.3 Exposure of Business Logic / Sensitive Data

Putting all business logic onto the client-side means that every user that kind of every user who knows how to display the source of a website can easily see how your website / your business works. Modern browsers even further have integrated tools that allow users to deeply inspect the code of a website and especially to monitor AJAX requests [add web inspector screenshot]. That users can cheat on a websites' code is not a problem that only occurs in SPWAs but you should keep in mind that almost all your business logic resides in the user's browser. So when a website deals with sensitive data (bank accounts, credit card numbers...) you need to make sure that none of the code on the front-end exposes security holes that could harm your users. Generally you should still do all privacy relevant operations on the server-side and additionally use HTTPS for the communication.

Also be sure to double check log-in states and admin rights on the server and don't let only the client-side handle it. Assume the following scenario: Your app has global User object with a Boolean field called "admin". Your 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 your website from this attack is to add a server-side generated field in the responses that adds information on rights the

---

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

current user has and switch the editable elements according to the response rather than on some front-end object. You still should double-check the rights in the backend. Another method to secure your code is to obfuscate it before you deploy it to your server. In that way your code is not as readable as before and attackers would need to put a lot more effort in understanding your code to harm your website. A nice side-effect that comes with code obfuscation in JavaScript is that your code also gets compressed so that you not only secure your website but also reduce load time for the user. One can choose from a variety of code-obfuscators in JavaScript: UglifyJS[19], YUI Compressor[20], Google Closure Compiler[21] (also gives advices on how to optimize your code before compressing it).

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 to increase the browser performance or users that are browsing at work that are not allowed to have their browsers to execute JavaScript. According to statistics by Yahoo[22] the percentage of users that have disabled JavaScript is approximately 1%. That number may seem low but if 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[23] but users that have actively disabled JavaScript did this for a reason and may simply decide not to use the site. One has to be aware of these numbers when creating JavaScript-heavy applications.

Another problem with dynamically generated layouts is that it is hard for screen readers to semantically interpret these layouts. Although 75% of screen reader users have JavaScript enabled[24] 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 the web more accessible, is aware of the problems described above and created a guideline

---

[19] http://marijnhaverbeke.nl/uglifyjs
[20] http://developer.yahoo.com/yui/compressor/
[21] http://code.google.com/intl/de/closure/compiler/
[22] http://developer.yahoo.com/blogs/ydn/posts/2010/10/how-many-users-have-javascript-disabled
[23] http://www.w3.org/TR/html4/interact/scripts.html#h-18.3.1
[24] http://webaim.org/projects/screenreadersurvey2/#javascript

called "WAI-ARIA"[25] that should provide solutions to developers and it is expected to get published in the middle of 2011.

5. Conclusion

SPWAs really can help to make a website feel better, to give the user a better experience browsing it. 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. There now is the possibility to create websites that don't look and feel like normal websites and actually are fun to wrk with.

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

In my opinion the rise in the interest for Node.js will also result in more and 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 client and server. And since there are (currently) no big web frameworks like RoR for Node.js developers it may be easier and faster for developers use tools like brunch to program websites with a simple REST-based Node.js backend.

But one also has to admit that SPWA brings a lot of new tools and technologies with it and so it might in a first run not be faster o develop because the programmers need to learn the new tools. Especially when developers don't have experience in programming in JavaScript the learning process can take a while because of the asynchronous parts that you have to deal with in JavaScript.

Before developing a SPWA one should first think about the negative points above and decide if it's okay to not get ranked in search engines currently (without extra effort) or if it's okay to expose the business logic to the user. Only if you don't think that these negative impacts will harm the success of your website you can start building the app. If you decide to port an already running and established website you should think twice about switching because your page may already be well indexed in the major search engines and all links may lead to a dead end after the rewrite of your page (if you don't

---

[25] http://www.w3.org/WAI/intro/aria.php

have a redirect component) like it happened to gawker.com, a well established "media news and gossip" blog[26], that switched the whole blog to a SPWA in February of 2011 and all indexed links were broken so that the number of unique visitors dropped by 50%[27] and a lot of users wrote bad reviews about the page. So before your switch, make sure you don't break indexed URLs.

---

[26] http://en.wikipedia.org/wiki/Gawker
[27] http://techcrunch.com/2011/02/17/gawker-redesign/