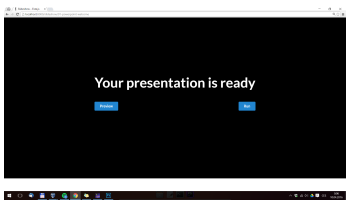
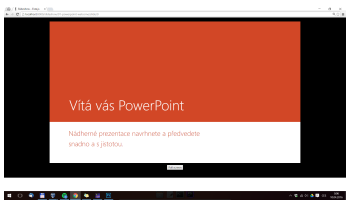
BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2016
http://excel.fit.vutbr.cz

PPTX to HTML Conversion

Hynek Vilímek*





Abstract

PowerPoint is an excellent tool for creating presentations and people are accustomed to use it. Its only handicap is that it is not installed everywhere and it exists in numerous versions. But there is an application that is installed almost everywhere and that application is the web browser. This work aims to create the PowerPoint presentation viewer for the web browser. With the internet as the environment, it may have a wide range of applications from the content sharing point of view. The solution is the web application that allows to upload the PowerPoint file and then the application displays the content of the file. The application also offers functionality such as the navigation between slides and the full-screen mode. The rendered slides in the web browser are very similar to the slides in the PowerPoint. It clearly does not support advanced features, but it supports displaying text, pictures, video and audio. Further, it supports basic styling options such as colours, margins, position and line height. This work shows current possibilities of the web environment and the web development. Moreover, this application may be a core of some start-up project.

Keywords: PowerPoint converter — PowerPoint to HTML — PowerPoint presentation viewer

Supplementary Material: Demonstration Video

*xvilim04@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Introduction

My work allows people to view the PowerPoint presentation on each computer, which has a newer web browser version. This can fulfil the need of portability and flexibility, because no one will be forced to check if the PowerPoint is installed and in what version. The application also gives the presentations the share-ability. Each presentation is possible to address by URL and the same is true for each slide.

The solution consists of three parts. The first part extracts the information from the presentation file. This results in the part with a clearly specified input and output. The input is the presentation file and the output is the internal representation of the presentation. This can be easily evaluated by the unit tests. The second part interprets the internal representation of the presentation and displays it as PowerPoint. This part results in the application that takes the internal representation and visually interprets it. The precision of the interpretation can be evaluated by the comparison of the reference image generated by the PowerPoint and the image of the actual result. The third part is the part of the application that surrounds the second part and adds features like a full-screen mode, navigation and asynchronous data loading. This can be also evaluated by the unit tests.

The existing solutions do not offer the same level of user experience as my application. The first found solution is the web application called *Zamzar*. It offers a PowerPoint to HTML conversion, but it generates a

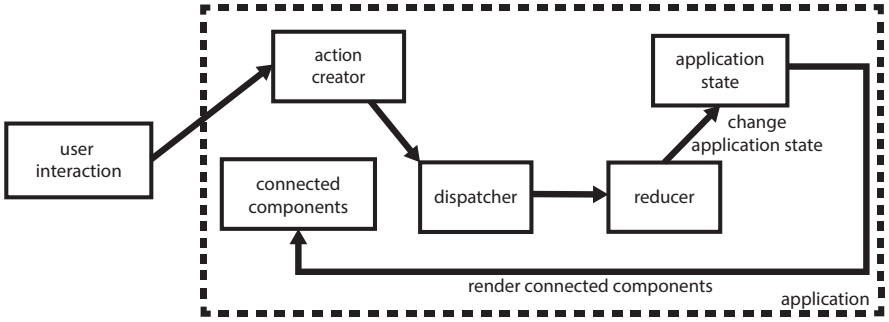


Figure 1. Processing the synchronous action through the application

zip archive, which is delivered to the previously filled email. The main difference is in the result of the conversion. It converts slides into images and places text over it. It does not support video, audio and the method of conversion is not suitable for display scaling. This service fulfils the function of partly viewing the content of the presentation file, but clearly does not fulfil the function to view it in the presentation mode. The second solution is from Microsoft itself and it is offered in the package called Office 365. But it also renders slides into images and uses SVG technology to set the correct position. The result of this application is better, but it still renders text into image, which is not optimal solution. In addition, this is a paid solution. The third solution is from Google and it is implemented as part of the Google Drive and Google Slides. These applications are not usable, because they have for example problems with fonts and margins. They clearly do not aim to the same use cases, but they aim to fulfil the need of a simple viewer for the PowerPoint presentations and they also allow to create new ones.

My solution is the web application and it consists of the server side part and the client side part. The server side part is responsible for extracting the information from the uploaded file and for their transformation to the internal representation. The client side part visually displays this representation and offers some functionality of the presentation mode from the PowerPoint. It supports timed transitions, full-screen mode with content scaling, navigation between slides and it partly supports an animated transition.

The achieved results are very clear because they can be very easily measured and tested. The accuracy of displaying the slide is very high. The full-screen scaling is also working. Each presentation and each slide has their own URL and that gives many options to transform this application into a start-up. It can be for example the application that allows to remotely control the presentation by smart-phone or the application that allows to distribute the content in the network. The client side application is so called single page application and therefore it delivers great user experience, because there is no need to reload whole application during the navigation process.

2. Theoretical background

2.1 ReactJS Components as the Key element of Building the Client-side Part

ReactJS components can be imagined as simple functions that take parameters called props and state and their output is a HTML mark-up. ReactJS automatically keeps the interface up-to-date when the data changes. This might normally lead to the unnecessary redrawing and to the performace issues, but ReactJS solves this with the mock DOM. Mock DOM is the internal representation of the current DOM. When the redrawing is going to happened, ReactJS takes the current and new mock DOM, compares them and computes the most efficient DOM mutation [1].

2.2 Handling the Application state with ReactJS and Redux libraries

The application state is the core of the client part of the application. The application state is only one and is stored in an object tree within one *Store*. *Store* is an object that allows manipulation with the state. The state is read-only and the only way of mutating the state is to emit an action. An action is a simple object that describes what happened. The state mutating is performed by reducers, functions that take the previous state and action and return the next state. Reducers have to be subscribed to *Store* to process dispatched actions. The way of processing a synchronous action is shown in Figure 1.

It all starts with an interaction. This can be a user interaction such as a click or it can even be a timed interaction. In the callback, which is hooked to the interaction, is called the action creator. The result of this call is an action. This action is dispatched by the dispatcher. The store automatically passes the action to the reducers. All this results in the new application

state and may lead to redrawing of connected components. While the component is connecting to the *Store*, it may specify which part of the application state affects the component [2].

This procedure is not applicable to an asynchronous action. This can be supported by an additional library called Redux thunk. This library allows action creators to return not only an action object, but also a function. The whole asynchronous process can be built from three synchronous actions inside this function. The first action describes the action that starts the asynchronous action. The second one is fired when the asynchronous action is successful and the third one is fired when the asynchronous action failed [3].

2.3 Presentational and Container Components

This division of ReactJS components is convenient for the cooperation with Redux library and also with other libraries that handle the application state. The summary of their features is shown in Table 1.

This approach is good for the following reasons. Firstly, it offers better re-usability, because the presentational components have no application logic. That allows to have more container components with different logic for one presentational component. It also offers a better separation of concerns. It separates application logic from UI components, which is better for testing the components [4].

2.4 Higher-order Components as the Way of the Composing Components together

This principle allows to share functionality that somehow interacts with life-cycle hooks across many components. It is based on the idea of composing functionalities together. It is a function that takes one parameter, which is the inner component, and the result is a new component, which implements new functionality and renders the inner component. This approach allows to apply a higher-order principle to any component many times [5].

2.5 Routing in the Application with React-router library

React-router library keeps the UI synchronized with the URL. The key object from this library is the Route object and it specifies which component has to be displayed according to the URL. If the URL has some parameters, then this parameters are passed to the component. Other parts of the library allow for example to programmatically move the application to the different URL, to create links to URLs, handle the web browser history and allow server-rendering [6].

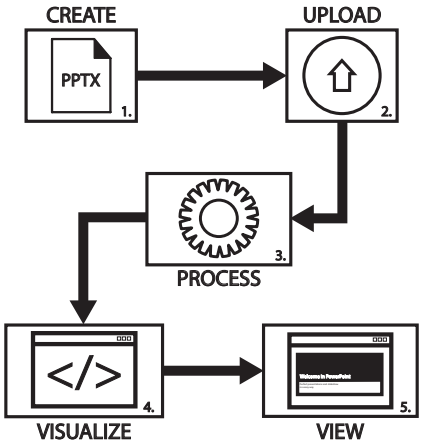


Figure 2. The general use case of the application

3. Application Design

There are several ideas behind the application design. The first one is that the client has to be as minimal as possible. This is advantageous for many reasons. The main ones are that it reduces the requirements for the processing power on the client and exposes only the necessary parts of the application on the client. The second one is to have a user-friendly web interface. This means that there has to be an option to address the presentation by its name and also slides by some number. This is convenient because it allows to share the presentations through the internet and this extends the usability of the application to more use cases. The last one is that the web client has to be a single page application. The reason is to have the best possible user experience from the application.

The application design aims to fulfil the process behind the use case of running the PowerPoint presentation in the web browser. This process is shown in Figure 2. It consists of uploading the presentation file to the server, analysing the file and converting the presentation into the internal representation, sending this representation to the client side application, visualizing this representation and reacting to the user actions such as moving between slides and viewing the presentation in the full screen mode.

3.1 The Server side Part of the Application

The primary functionality of the server is to transform the presentation file to the internal representation. It consists of several tasks such as parsing the presentation files, extracting the relevant information and transforming it into the internal representation. The format of the presentation file specifies standard ECMA-376 [1]. The parsing part contains numerous file readings and

[1] http://www.ecma-international.org/publications/standards/Ecma-376.htm

Table 1. Features overview of the presentational and container components

	Presentational components	Container components
Purpose	How things look (markup, styles)	How things work (data fetching, state updates)
Aware of redux	No	Yes
To read data	Read data from props	Subscribe to Redux state
To change data	Invoke callbacks from props	Dispatch Redux actions
Are written	By hand	Generated by React Redux or written by hand

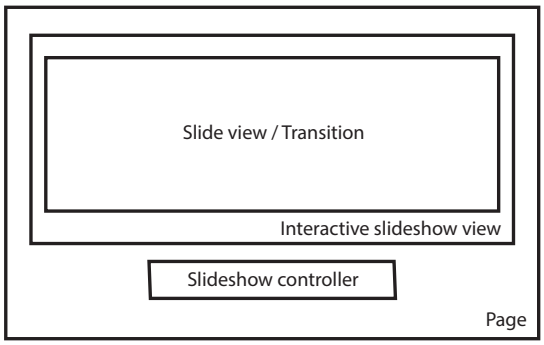


Figure 3. Overview of the components composition

therefore it is convenient to read them asynchronously. It also has to follow the standard specification. This makes the implementation more difficult, but on the other hand it is easily testable by unit tests. The next part is the transformation. This part is important, because the specifications of the objects in the data from the parsing part are distributed into numerous modules. This part transforms the data so that each object holds its own complete specification.

3.2 The Components in the Client side Part

The overview of the components is shown in Figure 3. The root component is named *Page*. It is simple component, which holds the other components together. This component is addressed from the *Route* component from react-router library and accepts parameters from the URL as they were set in the *Route* component. The last responsibility of this component is that it specifies the data used for server rendering. Every action that should be fired during the server rendering, has to be assigned to the component's field called *fetchActions*. This approach is used because the action is asynchronous. The server has to wait with rendering the static mark-up for the response from this action.

The following component is the *Slideshow controller*. Due to its simplicity, it reacts to the application state changes and also defines its visual appearance. It contains one button, which switches the presentation to the full-screen mode.

The next component is called *Interactive slideshow view*. It is the container component for the presentational component called *Slideshow view* with the full-screen and the navigation functionality. The *Slideshow view* component is the presentational component and it is responsible only for displaying the presentation. The full-screen and the navigation functionality are implemented by the higher-order components principle and they are also the presentational components. This separates the application logic from displaying logic and on top of that also helps re-usability.

The *Interactive slideshow view* component may contain either the *Slide view* component or the *Transition* component. The *Slide view* component contains the logic that displays one slide and the *Transition* component contains the logic of performing the transition between the slides. The *Transition* component internally consists of the components performing the animation and two *Slide view* components. The animation components are from the library called Velocity-react[2].

The *Slide view* component overview is shown in Figure 4. It consists of many *Shape view* components and many *Media view* components. The *Shape view* represents a positioned rectangular object. It may display shape related styles such as a background colour. The *Shape view* may contain the *Text view* component. This component displays text with all the margins and indentations between text runs and paragraphs. The *Media view* component is very similar to the *Shape view* component. It is also a positioned rectangular object. The difference is that it displays media files such as images, video and audio. The video and audio files are not delivered with the internal representation, but they are delivered on a demand from player.

3.3 Automated Integration Testing and Comparing with the Reference Images

The automated testing is the key aspect of continual developing because they can validate certain use cases. Tests can check if a change is actually the improvement or the deterioration and do it in bigger scope. This may not sometimes be faster than the developer, but it is repeatable. The key part of the application is visualizing the slide and therefore it is the first part that

[2] https://github.com/twitter-fabric/velocity-react

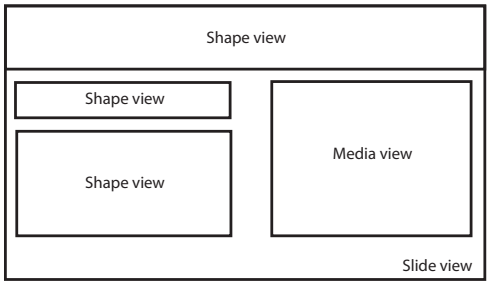


Figure 4. The *Slide view* component displaying sample data

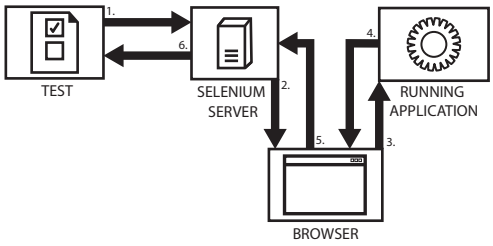


Figure 5. The process of the automated testing

should be tested. That was achieved with Webdriver.IO library[3]. The scheme of how it works is shown in Figure 5.

The running test is sending orders to the selenium server, which fulfils them through the web browser. The orders allow to request a screen-shot of some part of the web application. This part can be defined by a CSS selector. These screenshots can be compared and determine, how much they are identical. This purpose fulfils the library called WebdriverCSS.

4. Experiments and Implementation

The implementation process and experimenting with the results are the key parts of this work. The experiments were done in parallel with the implementation process. This speeds up the implementation itself and it also gives earlier feed-back to the current state of the application.

4.1 The Tools used in the Implementation Process

The whole implementation is written in JavaScript. This is natural for the client side of the web application, but thanks to NodeJS technology, it is also possible in the server side. It allows sharing the code in between and allows to run the complete application as a standalone application in the browser. The application uses new features that were introduced in ECMAScript 2015 standard, such as arrow functions, function generators, classes and promises. Because this standard is not fully supported by some browsers and the old ones does not support them at all, there is need to use some compiler. The used compiler is named Babel. The implementation is based on the library called Este, which unites some useful libraries together.

[3] http://webdriver.io/

4.2 Comparing the Reference Images with the Results

The results of the application are shown in Figure 6. The images in the first column are the reference images generated by PowerPoint. The images in the second column are images from the application and the images in the third column are showing the difference between them. As can be seen, there are some positional errors with the text. This is due to difference in the PowerPoint and the web browser with handling the fonts. The measured percentage difference is lower then 5% for all the images. The smallest difference is 1.5% and the highest is 4.5%.

5. Conclusions

This paper shows the application that allows to view PowerPoint presentations in the web browser. This allows to view and share the content of the presentation through the web environment.

The precision of the conversion is the key aspect of this application. If the slides follow basic rules and do not contain special effects, then the 5% difference between the original slide and the actual slide is easily achievable. In addition the application also contains the development tools to iteratively increase the level of the precision.

This work shows how modern web applications are built. The web environment is very dynamic and there are numerous tools to choose from and to use. This work shows one approach how the tools can be used and combined together. The application can also be the core of a start-up project, because the application extends the ability of editing the content in the web environment from the PowerPoint.

Except the application, the work introduces one approach of how to build modern web applications. This can be used by web developers to inspire and use some tools or procedures in their project. The idea of displaying the PowerPoint presentation in the web browser also deserves to extend in some start-up project.

Acknowledgements

I would like to thank my supervisor Prof. Ing. Adam Herout, Ph.D. for his help.

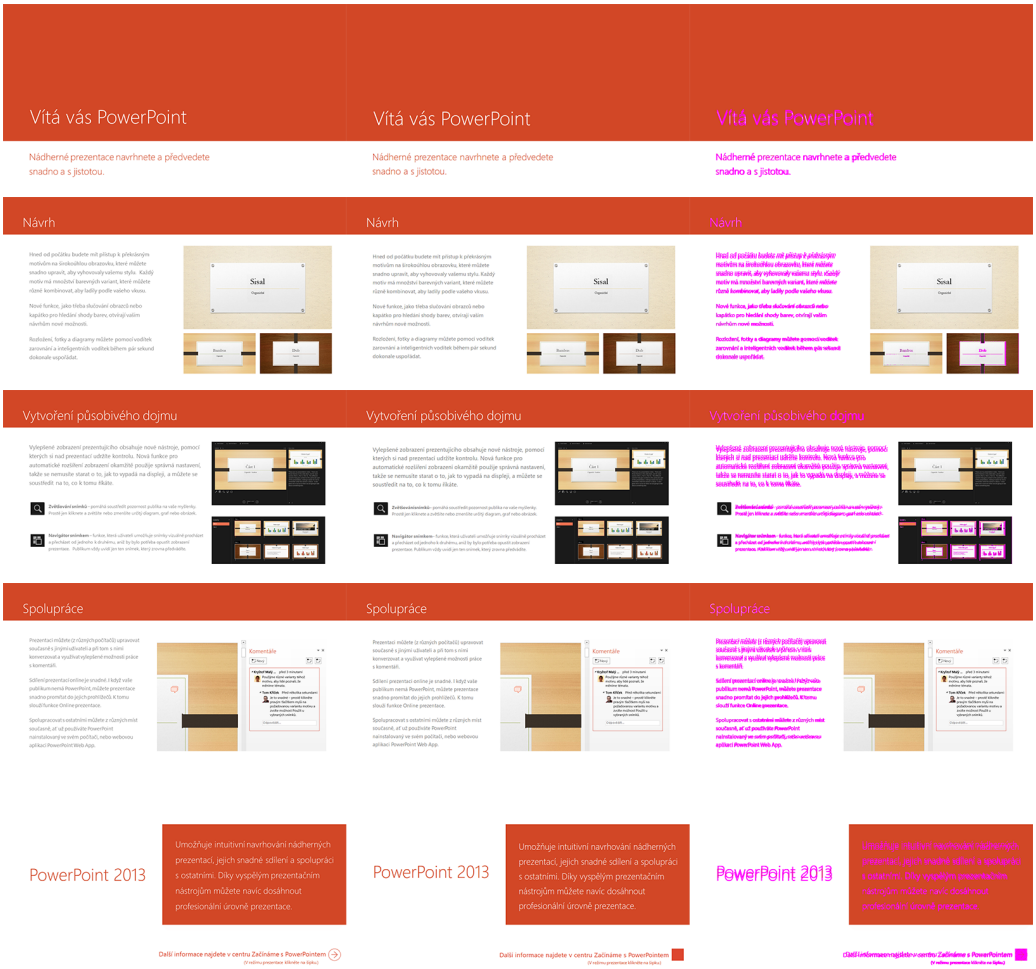


Figure 6. The comparsion of the reference images with the results

References

[1] Facebook Inc. Why react? Web page, 2013. https://facebook.github.io/react/docs/why-react.html.

[2] Dan Abramov. Basics. Web page, 2015. http://redux.js.org/docs/basics/index.html.

[3] Dan Abramov. Async actions. Web page, 2015. http://redux.js.org/docs/advanced/AsyncActions.html.

[4] Dan Abramov. Presentational and container components. blogpost, 2015. https://goo.gl/Cu8fud.

[5] Dan Abramov. Mixins are dead. long live composition. blogpost, 2015. https://goo.gl/GocvhS.

[6] Michael Jackson. Introduction. Web page, 2015. https://github.com/reactjs/react-router/blob/master/docs/Introduction.md.