INSTITUT FÜR INFORMATIK

der Ludwig-Maximilians-Universität München

Marking semantically relevant regions on slides in Backstage

Stefan Faßrainer

Bachelorarbeit

Aufgabensteller	Prof. Dr. François Bry
Betreuer	Prof. Dr. François Bry, Alexander Pohl
Abgabe am	30. März 2015

Acknowledgements

First and foremost, I would like to thank Prof. Dr. François Bry for supervising my work for this thesis and giving me advice how to improve the quality of the thesis.

Furthermore, I would like to thank Alexander Pohl. He was always available when I needed any kind of advice, motivation, inspiration and support. I would also thank him for the great and inspiring meetings we had that produced many amazing new approaches for this thesis.

Last but not least, I would like to thank my girlfriend Marie Anne being so patient and motivating during my work on this thesis.

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Hilfsmittel verwendet habe.

Grafing, den 30. März 2015 Stefan Faßrainer

Abstract

Backstage is a backchannel designed to support active participation of students in large-class lectures. *Backstage* offers the students the possibility to post comments referring to, and questions about, the slides of the lecture and it offers the professor the possibility to run interactive quizzes during the lectures, thus collecting data on the students' learning.

During the last years more and more laptops have been replaced by tablets and smartphones in the lecture theatre, whereas, however, the use of *Backstage* with a smartphone, especially because of the small screen, is not optimal. The object of this thesis is to mark the slides' contents into separate components by using so-called *filter boxes* so as to make *Backstage* well useable from small-screen devices like a smartphone as well as without displaying the slides.

This thesis first introduces the notion of *filter boxes,* addresses how *filter boxes* can be specified while producing slides and how *filter boxes* can be used for delineating portions of slides. Afterwards, this thesis introduces a self-programmed prototypic software, and shows, how *filter boxes*, defined by rectangle annotations, can be sort out of slides and provided to *Backstage*.

Zusammenfassung

Backstage ist ein Backchannel, der dazu entworfen wurde, die aktive Teilnahme der Studenten in Vorlesungen mit einer großen Teilnehmerzahl zu fördern. *Backstage* bietet den Studenten die Möglichkeit Kommentare zu und Fragen über die Folien der Vorlesung zu stellen und es bietet dem Professor die Möglichkeit während der Vorlesung interaktive Quizze zu starten, um dadurch Daten zum Wissensstand der Studenten zu sammeln.

Während der letzten Jahre wurden immer mehr Laptops von Tablets und Smartphones in den Vorlesungssälen abgelöst, wobei jedoch gerade die Nutzung von *Backstage* mit einem Smartphone, vor allem aufgrund der geringen Bildschirmgröße, nicht optimal ist. Der Zweck dieser Arbeit ist es Inhalte der Folien zu markieren, um sie mit Hilfe von sogenannten *Filterboxen* in einzelne Bestandteile zu zerlegen, so dass *Backstage* sowohl mit Geräten mit kleiner Bildschirmgröße, wie einem Smartphone, als auch ohne die Anzeige der Folien gut benutzbar ist.

Diese Arbeit führt zunächst den Begriff der *Filterboxen* ein, geht darauf ein, wie *Filterboxen* angegeben werden können, während Folien erzeugt werden und wie *Filterboxen* dazu genutzt werden können, um Teile von Folien zu skizzieren. Anschließend führt diese Arbeit eine selbstprogrammierte prototypische Software ein, und zeigt, wie *Filterboxen*, definiert durch rechteckige Annotationen, aus Folien ausgelesen und *Backstage* zur Verfügung gestellt werden können.

Contents

1 Introduction 1

1.1 Introducing Backstage 2

1.2 Reconstructing the relevant parts of a slide 3

2 Introducing *filter boxes* 3

2.1 Location filter 5

2.2 Message filter 6

2.3 Objectives of this thesis 7

2.4 Using *Backstage* with smartphones and (semi-)automatic posting of annotations 7

3 Alternative representations of slides in *Backstage* 12

3.1 HTML slides in *Backstage* 12

3.1.1 Introducing WYSIWYG editors 14

3.1.2 Create Slides and *filter boxes* in LaTeX 15

3.2 PDF slides in *Backstage* 16

3.2.1 Create *filter boxes* in Microsoft PowerPoint 17

3.2.2 Create *filter boxes* in PDF files 19

4 Conclusion 23

5 Towards an implementation 26

5.1 Choice of an approach 26

5.2 Technical decisions 26

5.2.1 Programming language 26

5.2.2 PDF library 27

5.2.3 Client – server communication 27

5.2.4 Backstage – *Box extractor* communication 28

6 Implementation 29

6.1 Client 29

6.2 The Server 33

7 Conclusion and Future Work 34

Bibliography 37

Appendix 39

1 Introduction

Attending large classes with several tens to hundreds of students is most likely an experience one makes in her academic life, at least in Europe. In fact, teaching and learning in large-class lectures has become daily practice in European universities for decades, especially in introductory courses [1]–[3]. Large classes are one of the many consequences of chronically underfinanced public universities and also of a steadily increasing number of enrolments.

Such large numbers of students attending a lecture makes it difficult for the lecturer and for the attendees to achieve an active partaking of the lecture's attendance [4]. Furthermore, the lecturer hardly receives feedback from the attendees during the lecture. For several reasons, attendees hardly pose questions when the audience is large: Common perceptions are that, in large audiences, personal questions are socially inappropriate and that, in a large audience, one should not draw the audience's attention to oneself. As a consequence and independently from content and pedagogy, lectures with large attendances turn out to be very different from lectures with small attendances - and much less effective.

The backchannel *Backstage* has been developed as a means to lift to larger attendances the kind of student-to-lecturer and student-to-student interactions that are possible, and common, in small attendance lectures. The core function of Backstage is to make it possible to ask questions and make comments which can be read and answered both, during the lecture or afterwards, depending on one's availability or working style. Backstage further supports quizzes and polls.

The goal of *Backstage* is thus to foster interactivity and activity of students so as to improve learning during a lecture. The communication on the backchannel is independent from the lecture’s discourse, the frontchannel, so that it does not interrupt the lecturer’s speech. Furthermore the physical distance between the participants of the communication becomes irrelevant. Finally the backchannel should help students to resolve comprehension issues, possibly in collaboration with other students, so that they do not get lost in the lecture. However, the lecturer has the possibility to supervise the students’ communication, immediately or later, so that he can reply to the students, correct wrong answers from other students and detect comprehension issues early.

1.1 Introducing Backstage

As mentioned above, this thesis deals with the backchannel *Backstage* in which backchannel communication is used to collaboratively annotate a lecturer's slides [5]–[11]. The lecturer's slides can be uploaded to and presented with *Backstage*. Besides interactivity fostered by the communication on the backchannel, the lecturer is able to instantly test the understanding of the students by running quizzes during a lecture.

In *Backstage*, the slides are the central artefact for in-class communication. Communication on *Backstage* is realized as collaborative annotating of slides. Therefore, participants place annotations on slides during the lecture to associate backchannel comments to specific slide locations. The placing of annotations on a slide provides a reference between the annotation and a port of the lecture's content.

Figure 1 shows the screen of a student using *Backstage*. In the middle (Fig. 1 mark 2) there is a navigation bar for the slides and the current slide is displayed. On the slide there are some annotations (Fig. 1 mark 3) and on the left side the content of the annotations can be seen (Fig. 1 mark 1). Furthermore there are some general information on the right side (Fig. 1 mark 4).

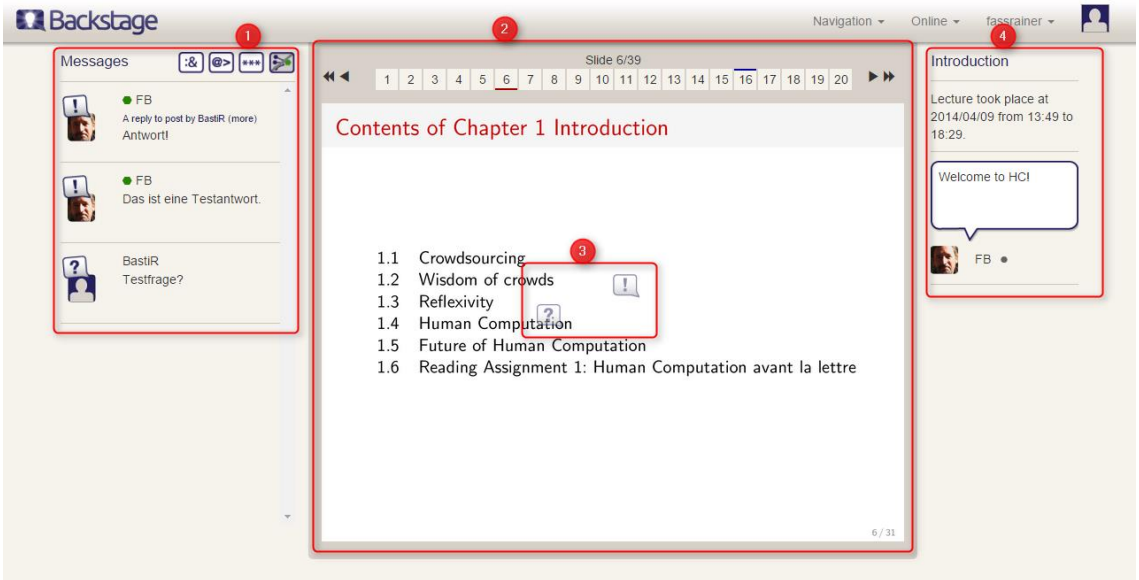


Fig. 1 - *Backstage* from a student's point of view. There are the slides (mark 2), some annotations (mark 3), their content (mark 1) and some general information (mark 4)

In the current version of *Backstage* the slides are transformed from a PDF format into raster graphics, so that standard HTML can be used to display the slides in *Backstage*.

1.2 Reconstructing the relevant parts of a slide

With the assumption that annotations are placed on the content they refer to, like pictures, proofs or theorems, it would be good to know which parts of the slides contain such relevant content. Semantically relevant parts might be given in documents created with semantic markup, but during the transformation into plain raster graphics, however, structural information gets lost [7]. Therefore the goal is to recreate relevant parts of a slide with different approaches, e.g. by *filter boxes* [12].

2 Introducing *filter boxes*

Substantial conceptual work on *filter boxes* has already been done in former theses [7], [12]. In [7], rectangles demarcate relevant parts of a slide that are referred to *filter boxes*. To be consistent with [7], the areas of interest defined by the drawn rectangles are also called *filter boxes* in my thesis. Whereas [12] elaborates on the conceptual foundations of *filter boxes*, my thesis deals with the automatic generation and human creation of *filter boxes*. More precisely this thesis investigates the following questions: how is a *filter box* created? What might a *filter box* contain? How is the content of the *filter box* provided to *Backstage*? And how is the *filter box* shown on a slide?

Backstage, in particular the approach to annotate slides, has foremost been developed for large-screen devices such as PCs, laptops and large tablets. In its current version, *Backstage* is difficult to use with small-screen devices such as smartphones or small tablets. Indeed, slides displayed on small screens are hardly readable. Furthermore, icons that are not placed at the correct location on the slide might distract other students. Section 2.4 shows examples of using *Backstage*, discusses what problems arise in adapting *Backstage* to small-screen devices and proposes an adaption of *Backstage*, but a final solution for these problems will have to be found in another thesis.

For using *filter boxes*, the slide is seen as a composition of several sections divided by structural elements of the document. For example, structural elements can be the location of images on a slide (Fig. 2-a). These structural elements can quite easily be recognized by the user, but it is hardly possible for the computer to process them, because raster graphics, to which slides are converted, do not provide such information. The goal is to reconstruct important parts of some structural elements of the document and provide them to *Backstage*. In Figure 2-a, the headline, the list of bullet points and the image is seen as important structural elements. However, the slide can be divided into more structural elements if necessary, for example, each bullet point or the grey part of the image can be a separate structural element.

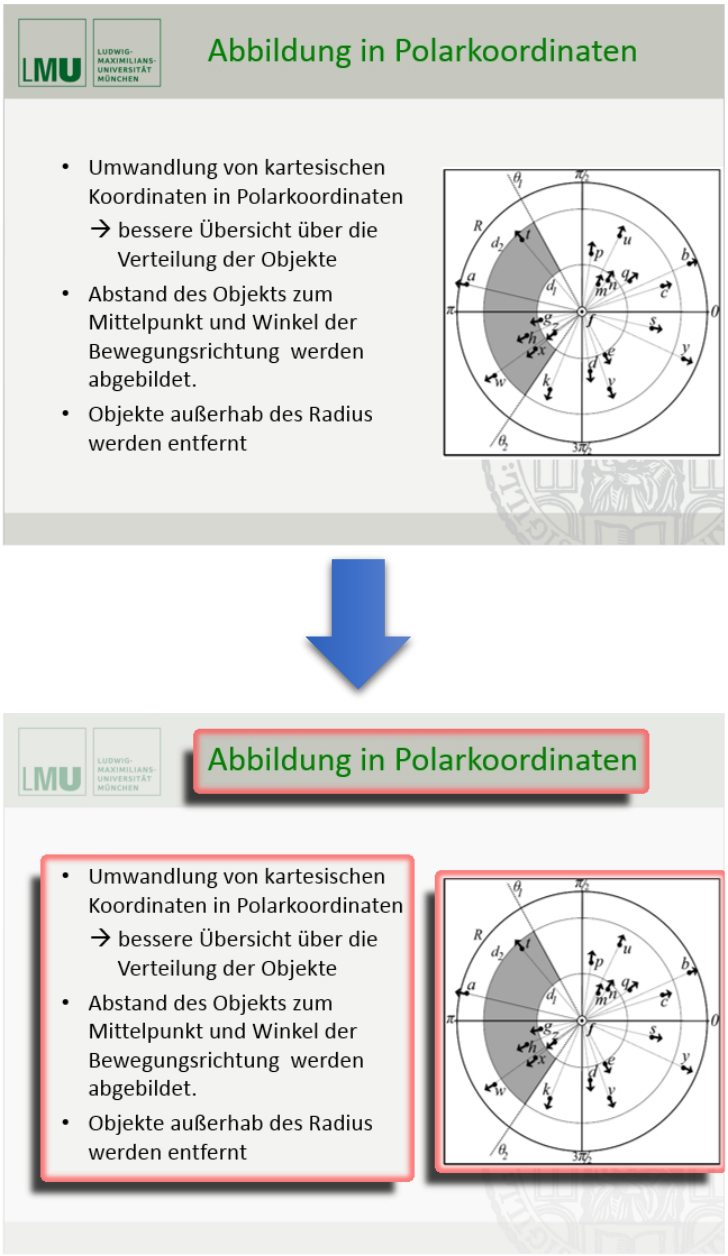


Fig. 2-a - A slide as a composition of several sections divided by some structural elements of the slide.

There are two kinds of filters. The first kind is a location filter. The second is a message filter.

2.1 Location filter

The basic idea of the location filter is that annotations are related to special areas on the slide and that the most intuitive and easiest way to define these areas is to draw a rectangle around the areas. So location filters denote rectangular demarcations semantically relevant regions of a slide. The concept of rectangular *filter boxes* is rather versatile in that it is to a large extent format-agnostic. For graphical slides, rectangular *filter boxes* can be drawn by using appropriate editors, e.g., PDF editors for PDF slides. If the markup of slides is available, *filter boxes* can be indicated using special markup and the position can be extracted from the rendered documents. We assume that the content of all annotations in this area is related to the content of the area itself. By selecting the *filter box*, only annotations in this area are shown to provide a better overview (Fig. 2-b).

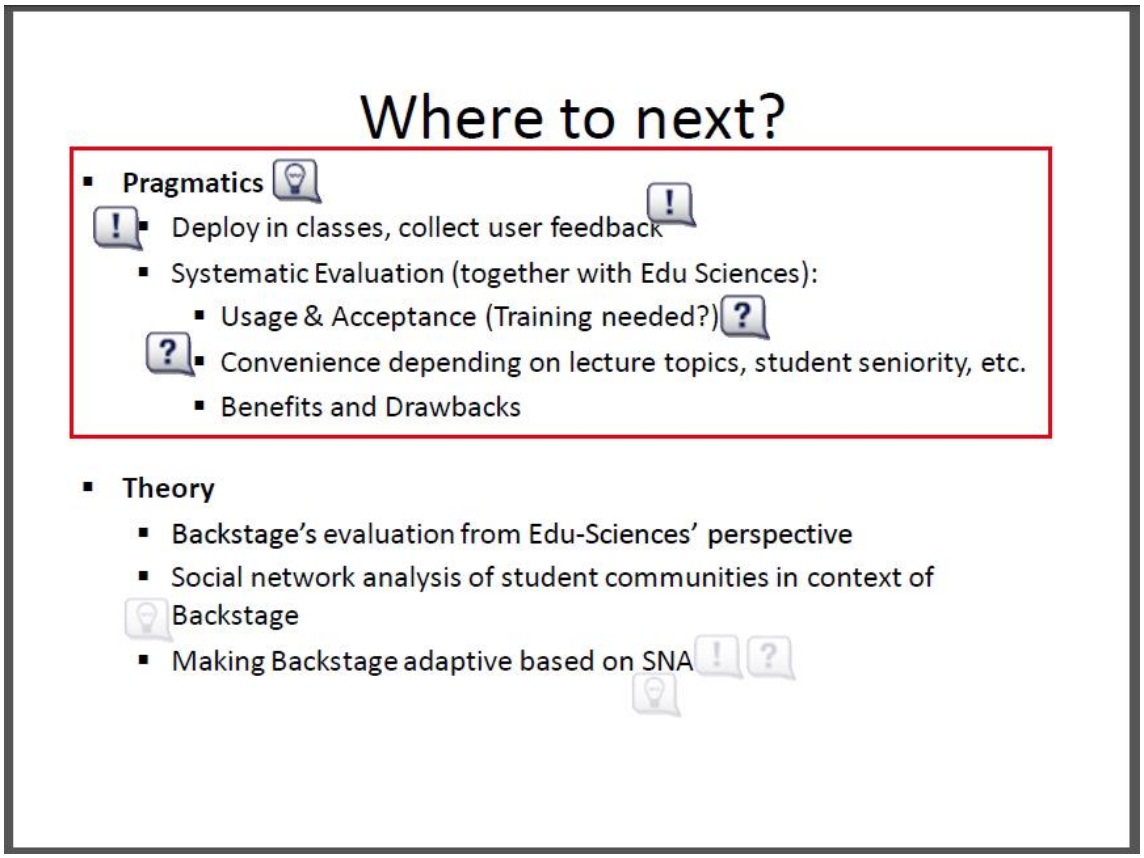


Fig. 2-b - Image of a slide with a location filter selected. Annotations outside of the area of the *filter box* have semi-transparent icons.

2.2 Message filter

The second kind of *filter boxes* relates to messages. This message filter allows the users to filter the comments according to different conditions (Fig. 2-c). Such conditions can be keywords, message types, the length of the message, some kind of relevance score and the author's username.

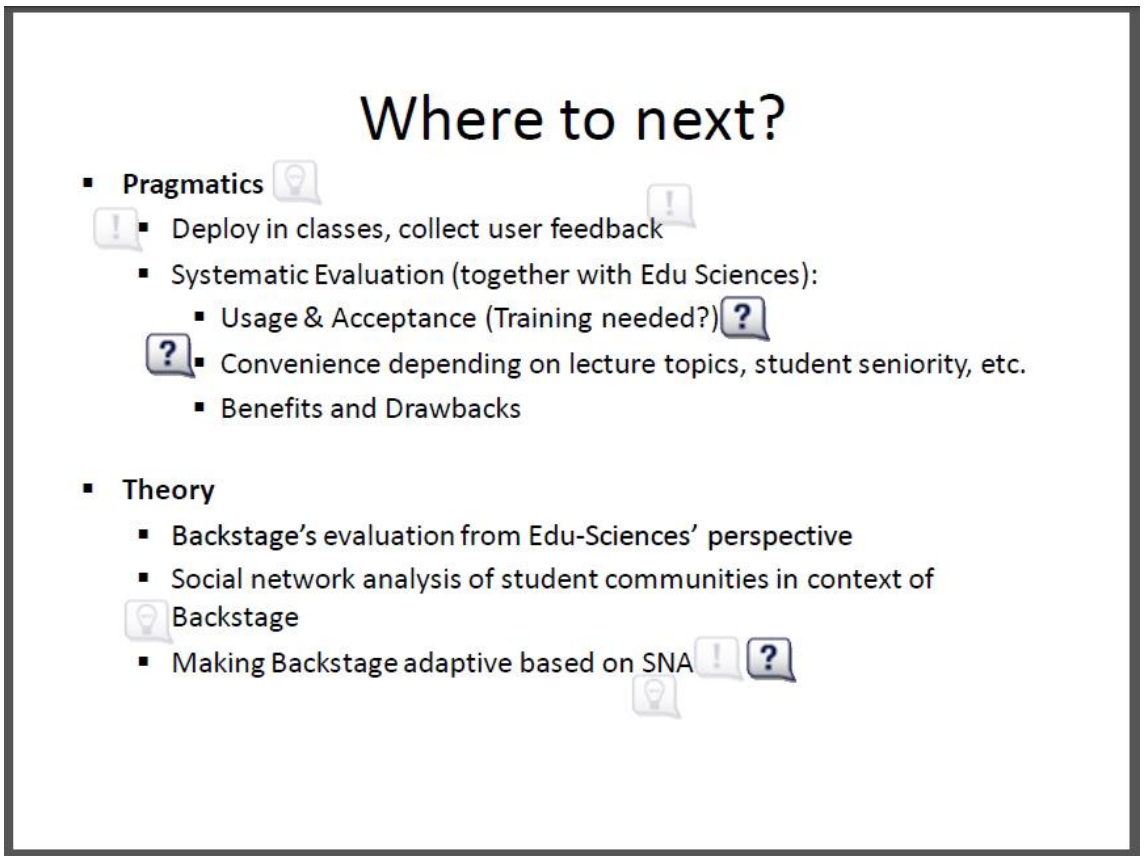


Fig. 2-c - Image of a slide with a message filter selected to show only annotations with a question mark as icon. Annotations with other icons are shown as semi-transparent icons.

2.3 Objectives of this thesis

In this thesis the location filter approach from [7] is used to mark some relevant parts of a slide by surrounding them graphically with rectangles. Moreover, it might be useful that both, the lecturer and the students, are able to define special areas on the slides, as long as there is some kind of quality management. Quality management, in this context, means that the defined areas should be at the correct place. For instance, if the special area should refer to a proof on a slide, this area has to contain the proof. This might be an issue, because the students might not have fully understood a slide at the time they define a special area, so that this area might be at the wrong place or refers to the wrong content [7].

This thesis deals with the reconstruction of important logical parts as *filter boxes* and these *filter boxes* can be given to *Backstage* as additional information about a slide. Moreover, further uses of *filter boxes* not considered in [12] are considered in this Thesis. It seems reasonable to allow attaching supplementary material to *filter boxes*, so that the students get more information about the part of the slide and the lecturer is able to publish material or information through the slide.

Furthermore, in this thesis, a proof of concept is developed focusing on the basic process of extracting and creating *filter boxes*. This thesis does not focus on a prefect usability of the approach, but discusses usability-related issues and suggests improvements.

2.4 Using *Backstage* with smartphones and (semi-)automatic posting of annotations

Filter boxes make it possible to annotate a slide using a screen on which not all, but only part of the slide is displayed (Fig. 2-d). The user is able to navigate through the slides by swiping left and right and parts of a slide can be selected by a dropdown menu. As mentioned in Section 2.3, the *filter box* is able to contain supplementary material and information. In this context, the *filter box* contains a link, several tags and an image as attachment. The kind of supplementary material and information a *filter box* can contain is described in Section 3.2.2.2.

Fig. 2-d - Example representation of a *filter box* that contains supplementary material and information on a small screen device

Annotations are shown and added as in the normal representation of the slide by placing an icon on the image. In this case, icons can of course only be placed within the image of the *filter box*.

Another way of interaction between the slides and users would be completely without the image of the slide. Only the information of the *filter box's* description are shown to the user (Fig. 2-e). Similar conceptual work has been done in [13].

Fig. 2-e - Example representation of a *filter box* on a small screen device without displaying an image

According to the model proposed in [12], each *filter box* has a unique name. By uniquely naming each *filter box* on a slide, students using small screen devices can navigate through the *filter boxes* by choosing them from a dropdown menu referring to the name of the *filter box*. This requires that the students are able to infer the content of the *filter box* by its name. After choosing a *filter box*, the content of the *filter box* and the referred annotations are shown as a list. New annotations can be added by clicking on the icons at the bottom. The position of the icon of an annotation on the slide is no longer determined by the user, but *Backstage*. Referring an annotation to a *filter box* of a slide instead of a location, however, requires the *filter box* reference to be resolved to a location on the slide (semi-)automatically.

This (semi-)automatic positioning of annotations brings along some questions:

A first question is, if annotations that have been placed by *Backstage* should have another look than annotations that have been placed by the students so that it is possible to distinguish them? For example, a different colour or icon (Fig. 2-f). The student's annotation in Figure 2-f (blue icon) most likely refers to the second bullet point, whereas the annotation placed by *Backstage* (green icon) refers to the whole *filter box* meaning the content of the annotation may refer to any of the bullet points. This leads to the following question: Is it important where the annotations are placed

in the *filter box* or can they be placed randomly? It might be useful to carry out a study to determine if people notice different adjustment of the annotations.

If the location is important, the third question is: What is the best location for an annotation? Annotations can be placed inside the *filter box*, e.g. in the middle of the *filter box* (Fig. 2-g, green annotations) or at the border of the *filter box* (Fig. 2-g, orange annotations). The disadvantage of placing annotations inside the *filter box* is, that the icon of the annotation might overlap the content of the *filter box*. However, the disadvantage of placing the annotations at the border of the *filter box* is, that it might not be clear to which *filter box* the annotation refers. Therefore the annotations should be connected to the *filter box* with a link or the annotations of a *filter box* are only shown if the *filter box* is selected or the mouse pointer hovers over the area of the *filter box* (Fig. 2-h).

It might be useful to give the lecturer the possibility to define a second area (Fig. 2-I, the blue rectangle), that determines, where the annotations that belong to the *filter box* should be located. If the lecturer does not define a second area the *filter box's* dimension is used as location for annotations.

Finally, there is the problem that annotations only refer to a *filter box*, so that it is not possible to add an annotation to a specific part of a *filter box* with this (semi-)automatic positioning. For example, the user does not have the possibility to add an annotation to a specific circular sector of the image in the right slide of Figure 2-h. The annotation will just be added at the border around the image. This leads to the issue that the users are not able to reconstruct the relation between the annotation and the location of the icon on the slide, so the basic approach of annotating slides with icons is questionably for the (semi-)automatic posting of annotations.

Filter boxes can also be used as a filter of the backchannel discourse. By selecting a *filter box*, the time-line is filtered such that only comments belonging to the selected *filter box* are shown [7]. Furthermore the standard view of the slide is able to provide more background information, because it is possible to provide supplementary material, a *filter box* can contain, to the users, if the mouse pointer is in the rectangle of the *filter box* or the *filter box* is clicked. The kind of supplementary material a *filter box* can contain is described in Section 3.2.2.2.

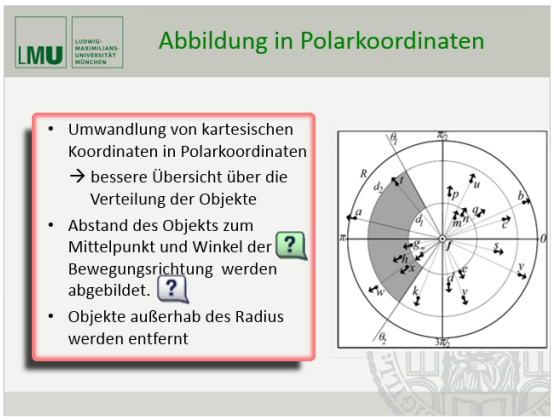


Fig. 2-f - An annotation placed by a student (blue icon) and an annotation placed by *Backstage* (green icon).

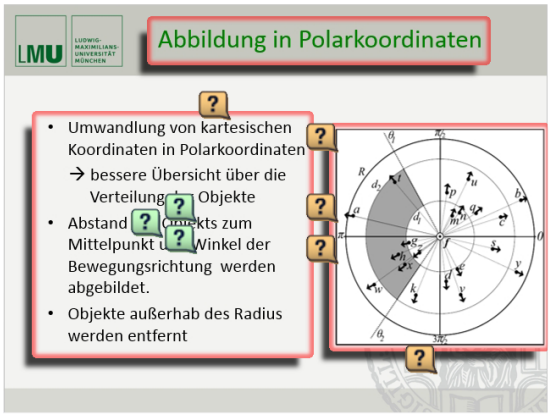
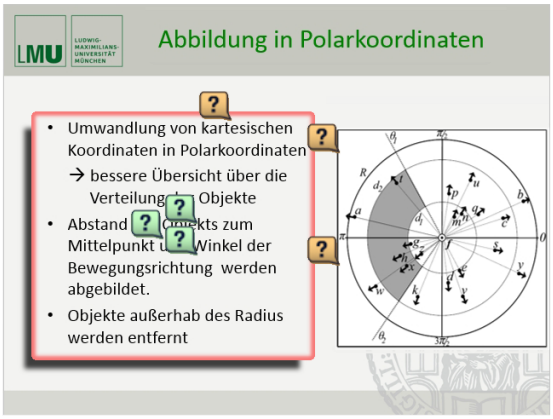


Fig. 2-g - Different locations for placing annotations automatically



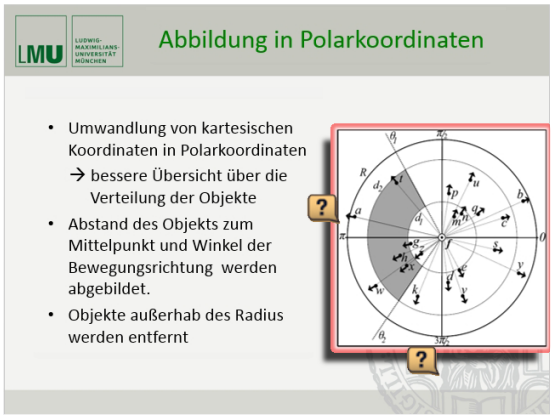


Fig. 2-h - Only annotations referring to a selected *filter box* are shown

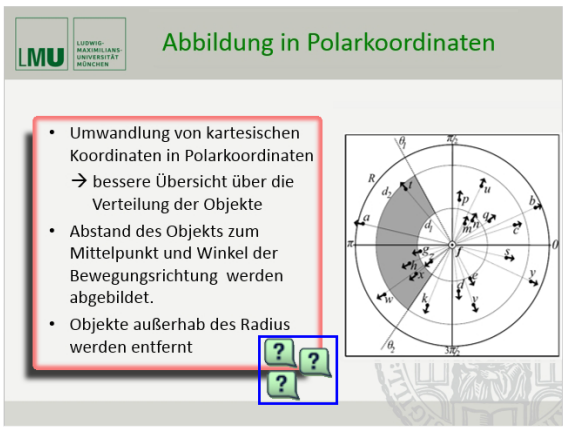


Fig. 2-i - A second area (the blue rectangle) that defines where the annotations should be located

3 Alternative representations of slides in *Backstage*

How to create *filter boxes* on slides and ways to extract *filter boxes* from slides largely depends on the slide format, e.g. PDF. This chapter investigates several slide formats and the implications of using these formats for the *filter box* approach. Four formats are investigated: HTML, LaTeX, Microsoft PowerPoint and PDF slides rasterized to graphics (the currently used format on *Backstage*).

Due to the great interdependence between the realizations of *filter boxes* and lecture slides, a reasonable approach is to reconsider how lecture slides can be represented and how it affects *filter boxes*.

3.1 HTML slides in *Backstage*

This chapter introduces a whole new system for slides in *Backstage.* Slides should no longer be PDF files or graphics, but HTML code specifying the slide and the *filter boxes*.

The HTML code can be created as usual or a LaTeX document can be transformed into HTML code. HTML and LaTeX have the same characteristic that the code has to be rendered first to determine the positions of the elements in the visual representation.

Although *Backstage* uses HTML to represent the views, the slides are embedded into HTML as raster graphics in the current version of *Backstage*. Alternatively, however, it seems reasonable to create and embed HTML slides.

The advantage of using HTML for slides, and hence *filter boxes*, is its natural fit for *Backstage*. All other formats require an external typesetting process and a transformation in a format suitable for Backstage (e.g., LaTeX slides need to be converted to PDF, which needs to be converted in a HTML compatible format such as raster graphics).

In HTML the position of any element in the shown website can be received by using the JavaScript method *getBoundingClientRect()*[1], that is supported by the latest versions of common browsers and the method is specified in a CSS Object Model[2] by the World Wide Web Consortium[3]. The method allows to define the position of the graphical representation of the *filter box* so that annotations can be associated with a *filter box* by their location on the slide.

[1] https://developer.mozilla.org/en-US/docs/Web/API/Element.getBoundingClientRect
[2] http://www.w3.org/DOM/
[3] http://www.w3.org/

Many so-called WYSIWYG[4] editors ("What you see is what you get") exist and are used, because they are the most common and user-friendly type of editors. User-friendly in this context means that the user creates the content similar to well-known word processors like Microsoft Office or Open Office by writing, highlighting or formatting text or changing layouts. Furthermore, most of the common weblogs use integrated WYSIWYG editors, e.g. Blogger[5], tumblr[6] or WordPress[7]. These editors translate the formatted text to HTML and CSS. However, it must meet several criteria to be suitable for creating slides and *filter boxes*:

1. The editor should be extensible and allow a natural integration of *filter boxes* in its command palette.
2. Moreover the editor should not require any active connection to an external server, because *Backstage* should remain independent of external resources. For example, it should is possible to send LaTeX formulas to an external provider and receive a HTML version[8] or an image[9] of the formulas. External Resources can cause problems, e.g. how to handle data, how to guarantee stable availability of the resource or what to do if the resource consumption is no longer for free of costs.
3. The editor's use should be free of costs and best also open source to make own modifications, like a method to define which part of the slide is a *filter box*. Moreover the licence of the editor must allow the integration into *Backstage.*
4. The editor should support the upload of multimedia content into *Backstage* and it should be able to process it.
5. The editor should support the display and the creation of mathematic formulas on the slides.
6. The editor should be able to read slides that have already been created by the lecturer. That means all usual file formats the slides might have should be readable by the editor, so that the slides can be imported and the lecturers can continue working on their slides without creating the slides again or making any additional changes.

[4] http://de.wikipedia.org/w/index.php?title=WYSIWYG&oldid=129201344
[5] http://www.blogger.com
[6] http://www.tumblr.com
[7] http://www.wordpress.com
[8] http://eric.chopin.pagesperso-orange.fr/latex/latex4web.htm
[9] http://www.codecogs.com/latex/eqneditor.php

3.1.1 Introducing WYSIWYG editors

The following table gives an overview which of the five widespread WYSIWYG editors selected for this thesis satisfy the requirements above. TinyMCE[10], CKEditor[11] and NicEdit[12] are the most used editors with big and active communities. Furthermore, TinyMCE is the default editor of WordPress[13]. Aloha Editor[14] and openWYSIWYG[15] are chosen to see if there is any difference to the three major editors above.

	TinyMCE	CKEditor	NicEdit	Aloha Editor	openWYSIWYG
Possibility to define a *filter box*	Yes	Yes	Yes	Yes	No
No dependencies from external resources	No (Plugin[16])	Yes	No (Plugin[17])	Yes	Yes
Software License	Yes (LGPL[18])	Yes (GPL[19], LGPL, MPL[20])	Yes (MIT[21])	Yes (GPL)	Yes (LGPL)
Multimedia content (movies, music, images)	Yes (Plugin[22])	Yes (Plugin[23])	No	No	No
Handle mathematic formulas	Yes	Yes (Plugin[24])	Yes	No	No
Import existing slides	Yes (Plugin [25]not free)	Yes (Plugin[26])	No	No	No

Table 1 - Overview of the features of WYSIWYG editors

[10] http://www.tinymce.com/
[11] http://ckeditor.com/
[12] http://nicedit.com/
[13] https://wordpress.org/
[14] http://www.aloha-editor.org/
[15] http://www.openwebware.com/
[16] http://moonwave99.github.io/TinyMCELatexPlugin/
[17] http://www.wiris.com/en/editor
[18] http://www.gnu.org/licenses/lgpl.html
[19] http://www.gnu.org/licenses/gpl.html
[20] http://www.mozilla.org/MPL/
[21] http://opensource.org/licenses/MIT
[22] http://www.moxiemanager.com/
[23] http://ckeditor.com/addon/filebrowser
[24] http://ckeditor.com/addon/mathedit
[25] http://groupdocs.com/marketplace/plugins/viewer/tinymce
[26] http://code.google.com/p/mollify/wiki/FileViewerEditorPlugin

Considering Table 1, CKEditor is the only editor that fulfils all the criteria. TinyMCE and NicEdit have plugins to handle mathematic formulas written in LaTeX, but they depend on external resources to convert the LaTeX formula into a graphical representation. The plugin to handle mathematic formulas for NicEdit is not for free. To solve the problem of the dependency on external resources and the missing plugin for multimedia content for NicEdit, it is possible to create own plugins and integrate them in the editor. Creating own plugins is also possible for Aloha Editor, but not for openWYSIWYG. The plugin *Viewer Plugin for TinyMCE* created by GroupDocs can render many formats, like DOC, DOCX, XLS and PDF. It is possible to annotate imported documents, but there is no support to modify the content of imported documents. The *FileViewerEditorPlugin* for CKEditor can render document formats like PDF and DOC, but again, it is not possible to edit the content of the files. As a consequence, the last criterion, enabling reuse of existing slides in PDF is difficult to meet because there is no currently existing way to transform a PDF document into a HTML code.

Finally, the table shows that CKEditor is the only one that fulfils all the prerequisites, so this editor is the first choice for a use in *Backstage*.

3.1.2 Create Slides and *filter boxes* in LaTeX

LaTeX[27] is a mixture of specific and generalized markup designed for the creation of scientific documents. As a consequence, LaTeX code is generally structured by using special commands and the type of the command is responsible for the rendering. For example, the title of a document is defined by *\title{The title}*[28]. This *title* command causes the visual representation of the text "The title" to be shown as defined in the related .sty file of the document.

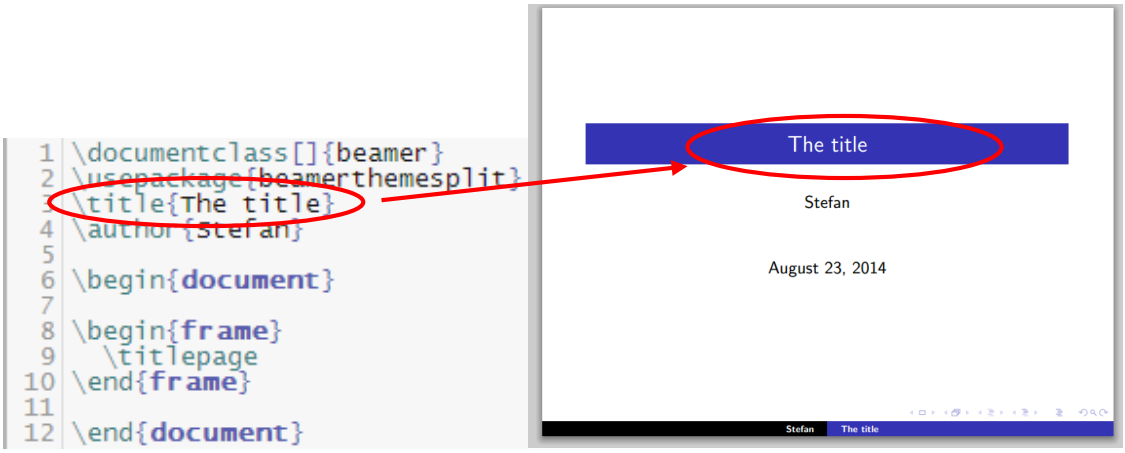


Fig. 3-a - Graphical representation of LaTeX code

[27] http://latex-project.org/intro.html
[28] http://www.weinelt.de/latex/title.html

In LaTeX custom commands can be defined. In particular, *filter boxes* can be represented in LaTeX by using a custom *filter box* command, e.g. *\box{…}*, or by creating a new LaTeX environment. This command or environment contains all the necessary information about the content of the *filter box* inside. It is also possible to convert LaTeX formatted text into normal HTML code so that the slides and *filter boxes* can be shown to the students.

The position of a *filter box* in LaTeX on a slide can be saved by using the *\zsavepos {⟨refname⟩}* command from the zref package [14]. This command provides the absolute position of any element in the graphical representation of the LaTeX document. As the slides are converted into HTML code, it is more important to get the position of a *filter box* on a slide while showing the HTML slide to the students. As already mentioned in Section 3.1, in HTML the position of any element can be received by using the JavaScript method *getBoundingClientRect()*.

3.2 PDF slides in *Backstage*

Creating *filter boxes* on slides gives a lecturer an additional opportunity to provide supplementary material, e.g. a link for a definition, an example for a mathematic formula, a photo of some notices on the blackboard or a program code as an example. This also offers the possibility of media sharing[29]. Allowing attachments to *filter boxes* conceptually makes it possible to also allow student to attach supplementary material to *filter boxes*. Therefore, important parts on a slide can be defined and related to attachments, links, a name, images, tags and videos. These elements are given to *Backstage* and presented to the students depending on which device they use to connect to *Backstage*. Important aspects that must be respected is the compatibility to the existing *Backstage*, so that the *filter boxes* can easily be integrated, and a user friendly handling to achieve a high acceptability from the users.

[29] http://de.wikipedia.org/w/index.php?title=Media_Sharing&oldid=122241417

3.2.1 Create *filter boxes* in Microsoft PowerPoint

In Microsoft PowerPoint text fields are used to create content on a slide. Assuming that related content belongs to the same text field, a text field can be seen as *filter box*.

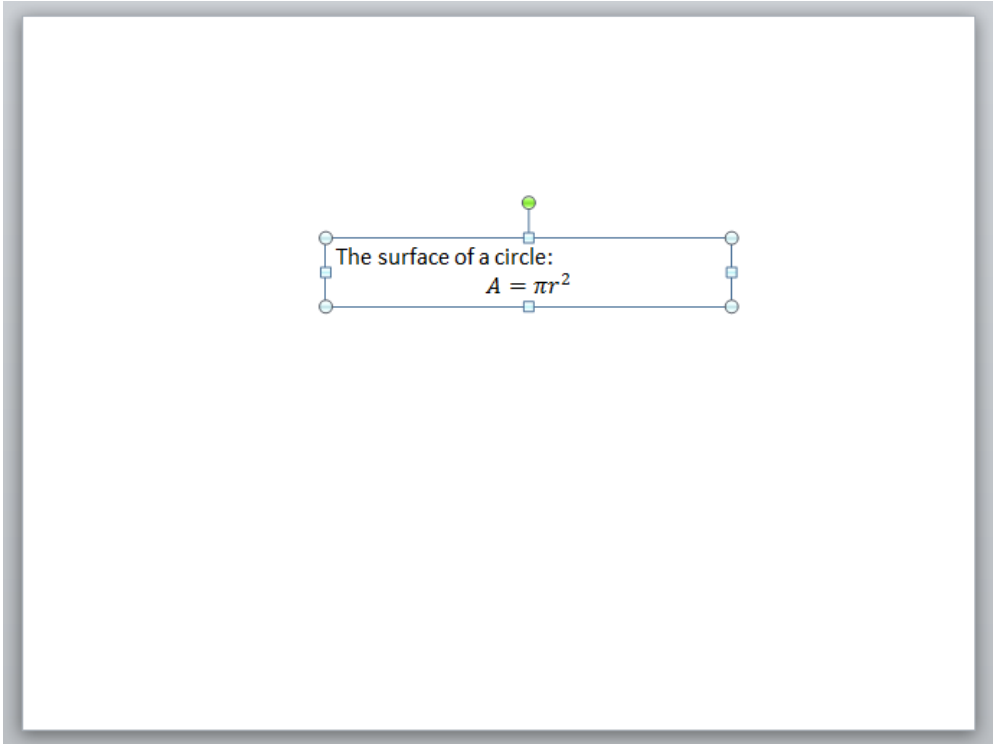


Fig. 3-b - A text field in Microsoft PowerPoint

As mentioned above, a *filter box* should have a name and it should be possible to add further information to the *filter box*. A good place to add this information is the alternate text value of the text field.

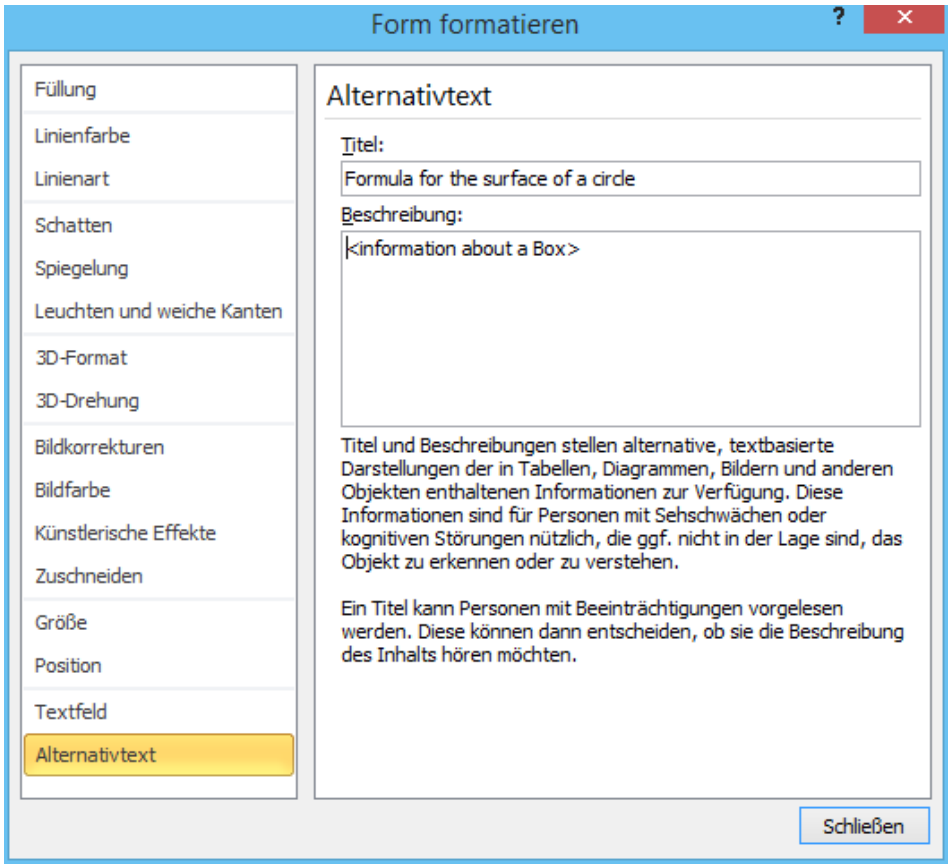


Fig. 3-c - A *filter box* in Microsoft PowerPoint

After adding the additional information to the text fields there are two different ways of extracting the *filter boxes'* information for every text field on the slides.

The first one is, that the lecturer runs a Microsoft Visual Basic script to extract the *filter boxes'* information into another file and the lecturer transforms the slides in a PDF file. After that, she can upload the file with the *filter boxes'* information and the PDF file to *Backstage*.

The second one is, that the slides can be uploaded to *Backstage* as a Microsoft PowerPoint file. In *Backstage,* a Microsoft Visual Basic script can be used again to extract the *filter boxes'* information and after that *Backstage* transforms the Microsoft PowerPoint file in a PDF file. As *Backstage* runs on a Linux distribution and Linux does not support the execution of a Visual Basic Script, a Microsoft Windows server and Microsoft PowerPoint is necessary to run the script. That means purchasing licences for Microsoft Windows and PowerPoint is required.

In both ways the *filter boxes'* information should be stored, together with the coordinates of the text field on the slide, in *Backstage* so that it is still available after the transformation in a PDF file.

3.2.2 Create *filter boxes* in PDF files

The current state is that the lecturer creates the slides with her preferred tool. After that the slides are converted into the PDF format and uploaded to *Backstage*.

3.2.2.1 Workflow of creating and extracting *filter boxes* from PDF slides

Figure 3-d depicts the process for creating and extracting *filter boxes* from PDF slides.

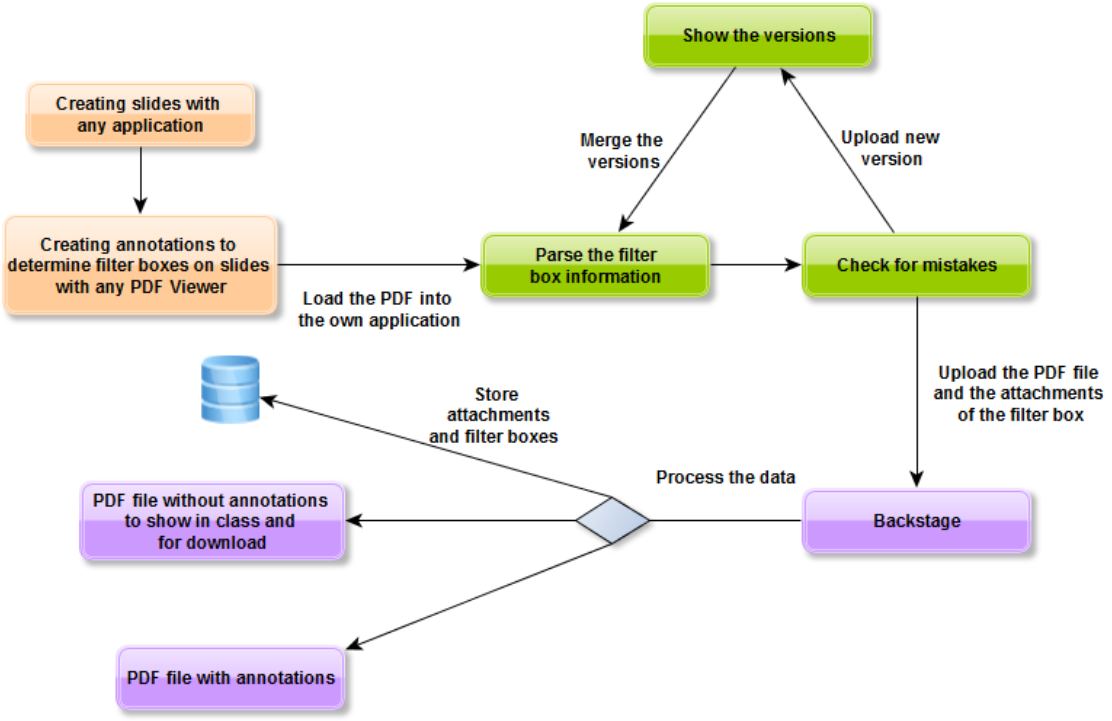


Fig. 3-d - Workflow import slides and parse *filter boxes*

In this workflow there is a new process after converting the slides into PDF format. At this point the lecturer has the possibility to create *filter boxes* by adding rectangular annotations to the slides. For doing that she has just to add square annotations to the slides. This can be done by using almost every PDF editor the lecturer wants to use. Since numerous PDF editors provide the functionalities required for creating rectangular annotations, developing a proprietary solution for *Backstage* is not sensible. This is an advantage for the lecturer, because she does not need to learn how to handle a new PDF Editor and it is also an advantage for *Backstage*, because no own PDF Editor has to be written and maintained.

Figure 3-e gives an example of a square annotation added with Adobe Reader XI[30].

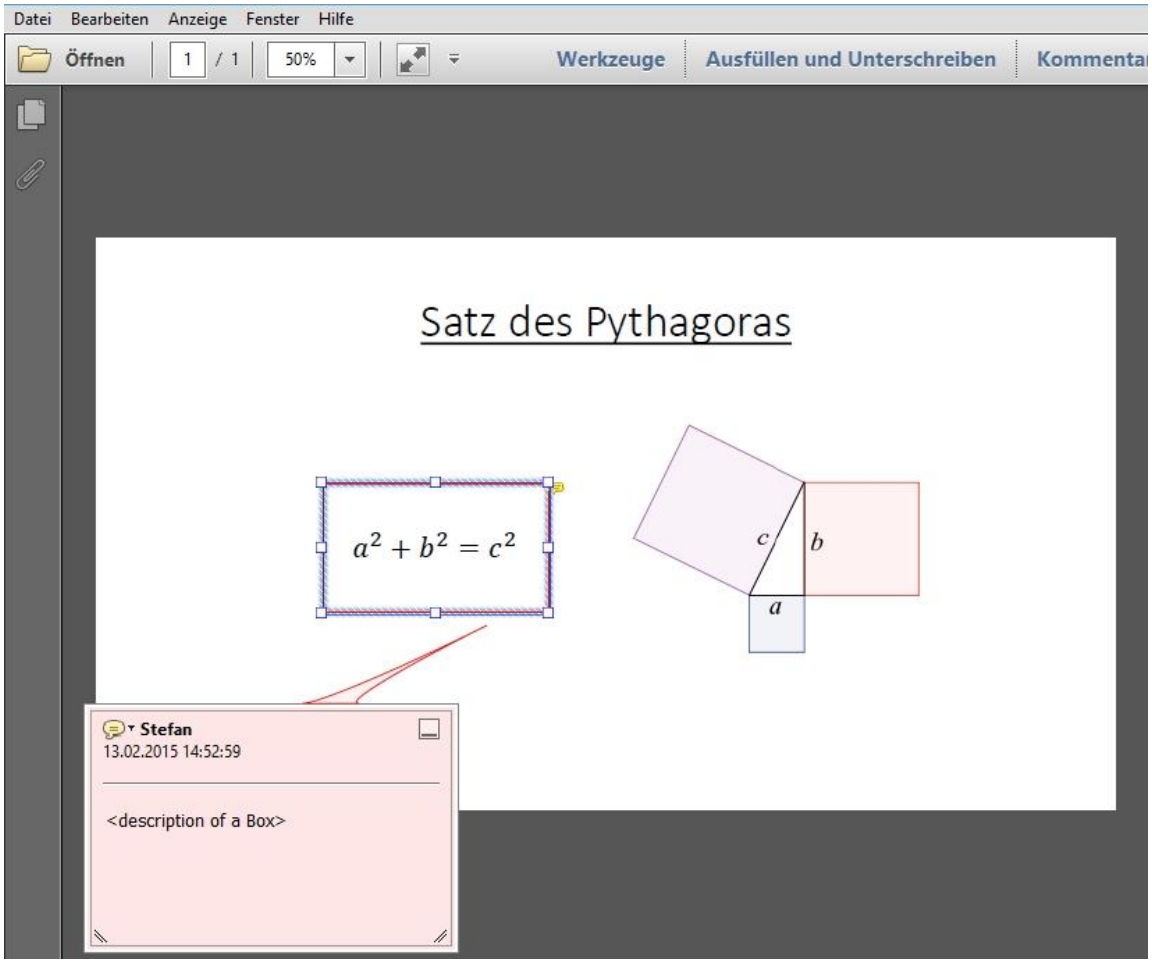


Fig. 3-e - Annotation in Adobe Reader XI

3.2.2.2 Description of the *filter boxes'* content

While creating *filter boxes* several information can be added to a *filter box*. At first the *filter box* needs a name. This should help to get a brief information about the content of the *filter box* and to quickly find a specific *filter box* on a slide. After that there a several optional attributes.

A good thing is to know where you can find further information about the content of the *filter box* or from where a quote is taken. Therefore you can add a link as additional information to a *filter box*.

To support searching for *filter boxes* it is useful to add tags to *filter boxes*. As known from other tag based platforms, e.g. twitter.com, the tags should start with a "#" in front.

[30] http://www.adobe.com/de/products/reader.html

Often there is not enough space on a slide to place all information about a topic on the slide, so that it can help to have the opportunity to attach further files to a *filter box*. Such files can be e.g. image, audio, video or text files.

That means the supported kinds of information are:

- A title tag (a special tag starting with "#t", so that the title of a *filter box* can automatically be extracted)
- One or several further tags (optional, start with "#", may contain whitespace characters)
- One or several comments (optional, start with "--")
- One or several links (optional)
- One or several image, video, text or other attachment files (optional, start with "@")
- One or several notices for the lecturer (optional, start with ":&")

The order of the information does not matter. The types of the information are identified by the special character at the beginning of the line or "http://", "https://" or "www." for links. So the description of a *filter box* looks like this in Adobe Reader XI:

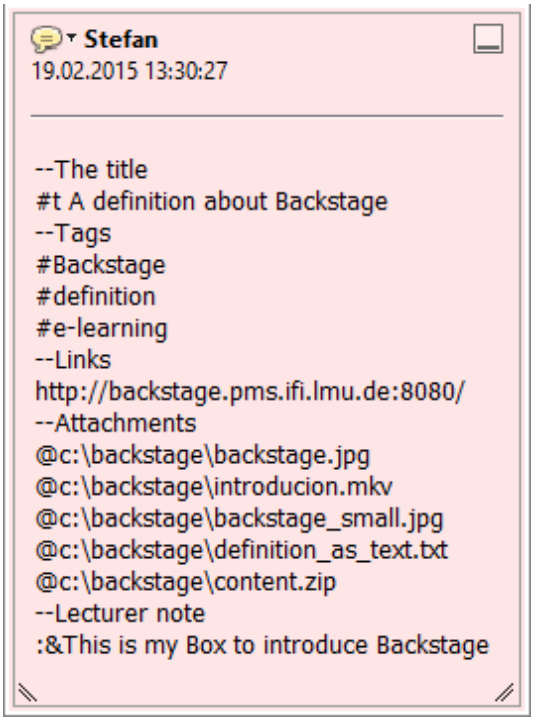


Fig. 3-f - Example of a description for a *filter box* in Adobe Reader XI

In this example, the title of the *filter box* is "A definition about *Backstage*". The *filter box* has several hashtags: "#Backstage", "#definition" and "#e-learning". There is a link that points to the *Backstage* homepage: "http://backstage.pms.ifi.lmu.de:8080". Moreover there are some files attached to the *filter box*: "c:\pictures\backstage.jpg", "c:\videos\introducion.mkv", "c:\pictures\backstage_small.jpg", "c:\texts\definition

_as_text.txt" and "c:\packed\content.zip". At the end there is a lecturer's note, where some notes about the *filter box* can be added, that are just visible for the lecturer.

3.2.2.3 Retrieve *filter boxes* from annotated PDF files

In the preparation of a lecture on *Backstage* additional extraction steps are necessary. Although this step could be integrated in Backstage we chose to implement the preparation process including the extraction of *filter boxes* in a separate application, which we refer to as *Box extractor*. The *Box extractor* has a counterpart on the server-side that receives all necessary files from the client part. This approach is in line with the current *Backstage* prototype that assumes that the preparations for a lecture are completed before a lecture is created on *Backstage*. This has the advantage that the *Box extractor* can be developed independently from Backstage.

The lecturer loads the PDF file into the *Box extractor*. The *Box extractor* provides the same structuring of courses into lectures as in *Backstage*, so that the content in the *Box extractor* can easily be synchronized with *Backstage*, so that all changes in *Backstage* are sent to the *Box extractor* and all changes in the *Box extractor* are sent to *Backstage*. A course on *Backstage* comprises lectures, and a lecture comprises a set of lecture slides. The *Box extractor* extracts the *filter boxes* and supplementary information and shows it to the lecturer providing the opportunity to check for mistakes. After that the lecturer can edit the existing *filter boxes*, but there is no possibility of creating new *filter boxes*.

The *box extractor* provides edit functionalities that allow modification of *filter boxes*. The edited slides can then be exported so as to ensure that the lecturer's PDF file remains consistent. The lecturer is also able to load a new version of the slides into the *Box extractor* and during this process she is able to merge the old and the new version, so that she does not have to annotate already former annotated slides again, but just the new ones.

After the imported slides are checked and approved, the *Box extractor* validates the existence of the attachments specified in the *filter box's* description and if the validation is successful it submits the slides and the attachments to *Backstage*. The server-side counterpart of the *Box extractor* receives the PDF file and the attachments. When the upload is finished the attachments are stored and the slides are converted to images and added to the lecturer's account. The additional support of attachments in *Backstage* tends *Backstage* to act as some kind of document management system that offers supporting materials for the students. The *Box extractor* is an external application on the *Backstage* server, so that the prototype implementation can be independent from the current *Backstage* system.

4 Conclusion

The focus of Chapter 3 is how pictures of slides can be divided into a composition of several sections. Therefore slides consist of several *filter boxes* that can carry some additional information and allow further possibilities how users can interact with the slides, which might be necessary to use *Backstage* with small screen devices. In this thesis four different approaches of creating *filter boxes* have been introduced and hereinafter the advantages and disadvantages of each approach will be discussed.

One advantage of representing the slides as HTML is, that HTML allows easy access and process structural elements within a document by standalone document parsers or within a browser using JavaScript[31]. The World Wide Web Consortium (W3C) has formulated a standardised API for accessing the HTML elements in JavaScript, the Document Object Model (DOM). The availability of good WYSIWYG editors allows for convenient authoring of slides. A *filter box* can be inserted in a HTML document as a *div* tag with a special class attribute and some extra attributes. To define a *filter b*ox on the slide in the WYSIWYG editor, on the one hand a button can be implemented. When the user clicks on the button she is able to mark a rectangular area. This area is now seen as a *filter box*. This *filter box* is inserted into the HTML document as mentioned above and the extra attributes can be inserted by selecting from predefined attributes. This makes sure that the *filter box* only contains necessary information, e.g. a name.

Another advantage is that you do not have to convert the slides into a PDF Document. The rendering of a document is accomplished by the browser. Any further rendering process as the case for LaTeX is unnecessary. At the moment *Backstage* uses the PDF format, because it is independent from which kind of format the content is created. All common formats, like Open Office documents, Microsoft Office documents or documents created with Apple Software can be converted to PDF. A single page of the PDF document is shown to the students as an image.

As a disadvantage of PDF, PDF does not contain any logical information, so that all the slides have to be created with a web editor now, to get a HTML code in the end, which can be used to show the slides, without any conversion. The benefits are, e.g. that students are able to copy text from the slides or download the images on the slides. This is not possible, if the slide is shown as an image.

Moreover, the web editor does not allow the reuse of already created slides and Internet access is required while creating the slides. As a consequence is that the web editor is rather an option for creating new slides then creating existing slides in the web editor again. The software used to create slides provide a multitude of functionalities lecturers may find useful and miss when working with a web editor. It is quite difficult to provide an editor with similar functionalities so as to make the approach an actual alternative to office software, so there will also be some acceptance issues.

[31] http://de.selfhtml.org/javascript/

For a large part of working with *filter boxes* being connected to the Internet is not required. We thus realize a mode of operation in which most work can be done offline and an Internet connection is only required for submitting the *filter boxes* to a server. It is general practice to follow the approach described, see for example Dropbox[32].

LaTeX is a widespread option to create documents in the scientific environment, but even in this environment there are people that are using other applications like Apple Keynote or Microsoft PowerPoint. As one of the aims of *Backstage* is, that it is applicable for all kinds of lectures, it may be a problem that it is not possible to use existing Microsoft PowerPoint, PDF or other file formats, because they cannot be converted in a LaTeX document.

The advantage of the LaTeX approach is that standard LaTeX commands already allow parsing additional information about the content of the slide that might be useful for creating *filter boxes*, e.g. the *includegraphics* command specifies an image that can be placed on the slide by using *begin{figure}* … *end{figure}*. Moreover, it is possible to define an own command or environment, so that every part of the slides can be a *filter box* that can be parsed so that the position and the content of the *filter box* can be received.

Many lecturers use Microsoft PowerPoint to create their slides. This means a very low effort for the lecturers to create *filter boxes*, because they can use their already existing slides. The only thing that has to be done is to add the additional information of the *filter box* in the text field. The usage of text fields as *filter boxes* fits well human imagination of a *filter box* and the alternate text value offers a suitable option to add additional information for a *filter box*. After that, the current workflow in *Backstage* can be used. The only difference is that the lecturer uploads a Microsoft PowerPoint file instead of a PDF file. After that the Visual Basic script is used to parse the *filter boxes* and the file is transformed into a PDF file. From this step the workflow is the same as it is now.

There are LaTeX Plugins for PowerPoint, e.g. IguanaTex[33], but nevertheless the lecturer has to copy the LaTeX code manually. This means that lecturers that have existing slides in LaTeX format have to do additional work to convert their slides into Microsoft PowerPoint. Apache OpenOffice[34] and Apple Keynote[35] offer to export the slides as Microsoft PowerPoint file, but the lecturer has to check the slides after the export for errors in the transformation. Microsoft PowerPoint is not for free, so this means the lecturer has to spend money on the software even if she just uses it to create the *filter boxes* on the slides. This will probably decrease the acceptance of this approach for users that do not already own Microsoft PowerPoint.

Finally the approach of creating an own application has the advantage that it needs the least changes in the existing *Backstage*, so that a quick practical implementation can

[32] https://www.dropbox.com/
[33] http://www.jonathanleroux.org/software/iguanatex/
[34] http://www.openoffice.org/
[35] https://www.apple.com/de/mac/keynote/

be done. Besides the new application all the existing parts of *Backstage* can be used again in this approach. Another advantage is that the lecturer can continue using any application for creating the slides and all common PDF viewers to create the *filter boxes*. That also means that all the old slides can be used without any additional work and the lecturers are still able to work on their slides as long as they have their notebook or tablet regardless of having an internet connection or not. These two advantages lead to the conclusion that this approach fits well for an experimental practical implementation.

The best solution would be a hybrid environment. This means *Backstage* offers all the approaches mentioned above so that the lecturer is able to decide in which format she wants to create the slides. This brings along that the lecturer also is able to use her existing slides. The advantage for the *Backstage* system is that this approach will receive a high acceptability from the lecturers, because they are not forced to change anything and that there will be more and more slides in HTML format in the future.

But this approach is unsuitable for a prototype implementation, because many parts of *Backstage* have to be changed and new workflows have to be created. Besides, an integration of the *Box extractor* into *Backstage* increases *Backstage's* system complexity and thus likely makes maintenance of the software more difficult, but the main objective of this thesis is a proof of concept for the *filter box* approach.

As the *filter box* approach is an experimental approach this hybrid approach is too complex and too expensive to realize it. For testing the *filter box* approach it is better to use one of the approaches mentioned above and if the tests are successful, so that the *filter box* approach works as expected, then it makes sense to think about this hybrid approach again.

5 Towards an implementation

This chapter presents the concept for an implementation. In Section 5.1 the choice of an approach is made and Section 5.2 discusses the technical decisions for the implementation of the *Box extractor*.

5.1 Choice of an approach

As the main goal of this thesis is to receive *filter boxes* and give them to *Backstage* in an experimental environment, it is important to minimize the changes to *Backstage* and the current version of *Backstage* remains independent from the extension. The *Box extractor* approach is chosen so as to fulfil these requirements.

5.2 Technical decisions

In this section the technical decisions for the *Box extractor* will be discussed. At first a programming language in which the *Box extractor* should be written and a programming library that is able to read PDF documents with the annotations must be chosen.

Furthermore a protocol for the communication between the client and the server must be specified.

5.2.1 Programming language

Java[36] is chosen as programming language and JavaFX[37] is used for the visual presentation of the *Box extractor*, because it offers the implementation of a modern user interface for a Java application.

[36] https://www.java.com/
[37] http://www.oracle.com/technetwork/java/javase/overview/javafx-overview-2158620.html

5.2.2 PDF library

Apache PDFBox[38] is an external library for processing the PDF documents. There are also other libraries that might be usable[39] [40] [41] [42], but Apache PDFBox is simple to use, brings along a good documentation and Apache Software Foundation[43] is a well-known corporation with an active community of developers.

5.2.3 Client – server communication

The server side of the *Box extractor* is running a HTTP server waiting for HTTP requests from the client. The lecturer starts the upload of the slides in the client part of the *Box extractor*, so that the *Box extractor* sends the slides, the information about *filter boxes* and the attachments to the server. If the upload was successful the server stores the files on the file system and returns a confirmation to the client.

[38] https://pdfbox.apache.org/
[39] http://itextpdf.com/
[40] http://www.icesoft.org/java/projects/ICEpdf/overview.jsf
[41] http://pdfclown.org/
[42] http://pdfjet.com/os/edition.html
[43] http://en.wikipedia.org/w/index.php?title=Apache_Software_Foundation&oldid=648808355

5.2.4 Backstage – *Box extractor* communication

A first really trivial approach is that *Backstage* watches the directory, where the uploaded files are stored, for changes. When changes take place *Backstage* loads the new files and processes them internally (Fig. 5-a). There are three disadvantages. At first, if the checks on the file system is not based events, there will be many unnecessary directory checks, because no change took place since the last check. Furthermore if the interval between two checks is bigger to avoid the unnecessary checks, new uploaded files will not be processed in real time. At last, Backstage has to know how to process the stored files, e.g. how to retrieve the information about the *filter boxes* from the file.

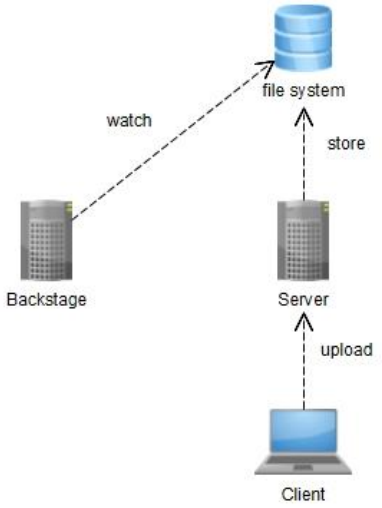


Fig. 5-a - Backstage as FileWatcher

The first two disadvantages mentioned above can be solved by using a trigger mechanism or checking for new files on file system events instead of a specific interval of time. This means the *Box extractor* or the operating system tells *Backstage* that there are new files, so that *Backstage* can start to process them (Fig. 5-b). But this does not solve the last disadvantage.

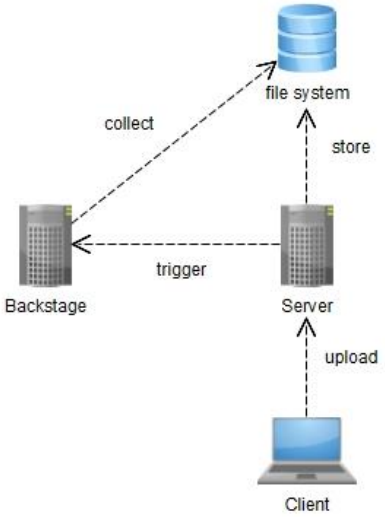


Fig. 5-b - Backstage informed by the *Box extractor*

To solve the last disadvantage a Representational State Transfer (REST) interface[44] is added to the *Box extractor* (Fig. 5-c). REST is based on the HTTP protocol. This REST interface provides *Backstage* the possibility to ask for information about the slides without having detailed information how and where the files and information are stored. For instance, a request can be: "Give me all *filter boxes* of slide 3" or "Give me the attachment with ID 14". The REST interface processes the request and provides the information as HTTP respond to *Backstage*.

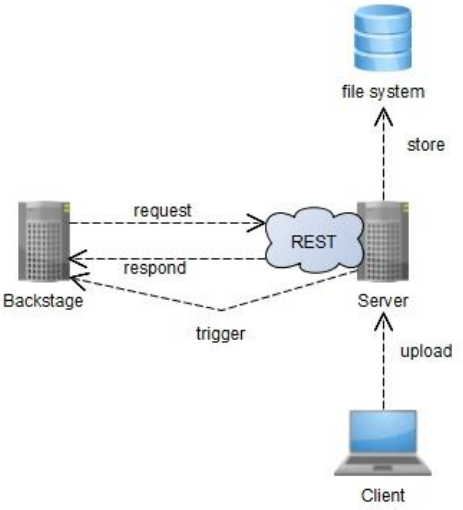


Fig. 5-c - Provide information using REST interface

[44] http://en.wikipedia.org/w/index.php?title=Representational_state_transfer&oldid=641781698

6 Implementation

The *Box extractor* consists of two components. One is the client part, the other one is the server part. To start the *Box extractor* as server, `--server` has to be added as command line parameter while starting the *Box extractor*. As the client has a graphical user interface the lecturer can interact with, the server is running without a graphical interface.

6.1 Client

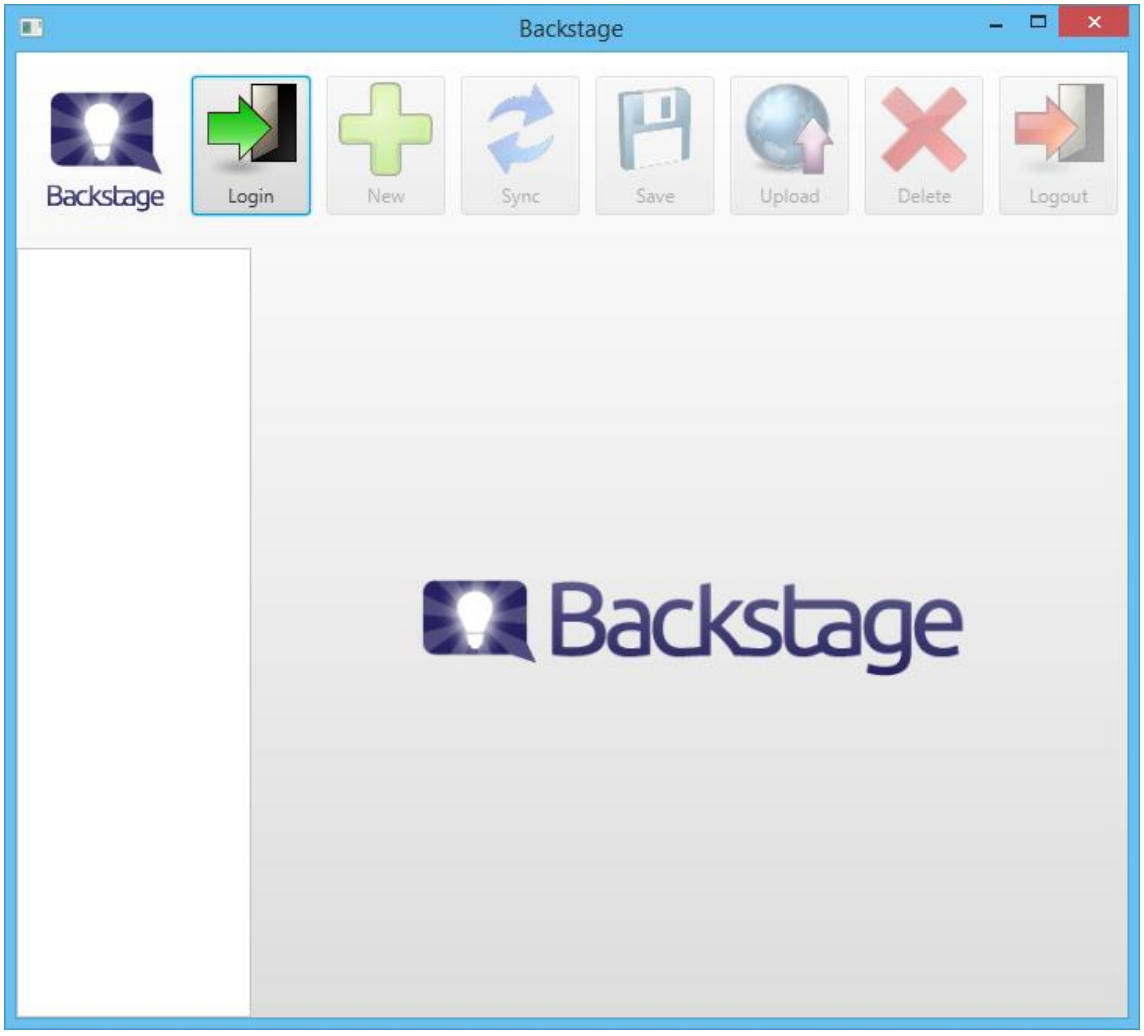


Fig. 6-a - The client part of the *Box extractor* after the start

As a first step after starting the *Box extractor* as client, the lecturer has to authenticate himself by entering her *Backstage* username and password. The login dialog (Fig. 6-b) can be opened by clicking the *Login* button.

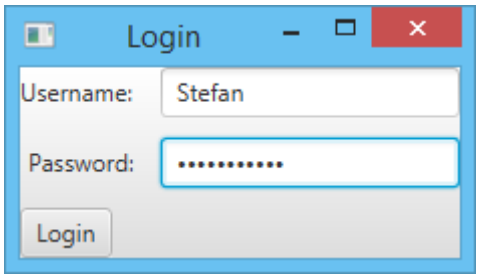


Fig. 6-b - Login dialog

In the *Box extractor* the lecturer has the opportunity to create new lectures and lessons by clicking the *New* button. After that she has to choose the PDF document the *filter boxes* have been added before and weather a new course with a new lecture should be created or a new lecture should be added to an existing course.

The *Box extractor* creates for every user a directory on the hard disk to store the pdf documents and a XML document containing the information of the *filter boxes*.

On the left side of the graphical user interface the lecturer can see her existing courses and lectures in a tree structure (Fig. 6-c).

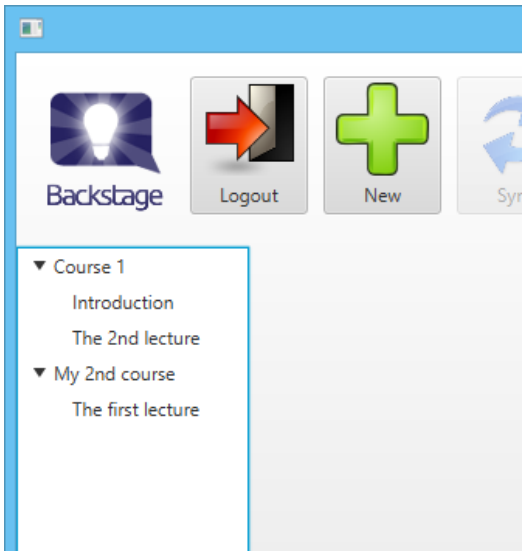


Fig. 6-c - The user's courses and lectures

A lecture can be opened by clicking on the lecture's title. In this view the lecturer can navigate through the slides (Fig. 6-d mark 1) and choose a single slide by clicking on the small picture of a slide. After that some details of the slide are shown (Fig. 6-d mark 2). By default the details of the first slide are shown. The *filter boxes* of the slide are shown (Fig. 6-d mark 3) so that the lecturer has the opportunity to validate and edit the content of each *filter box*.

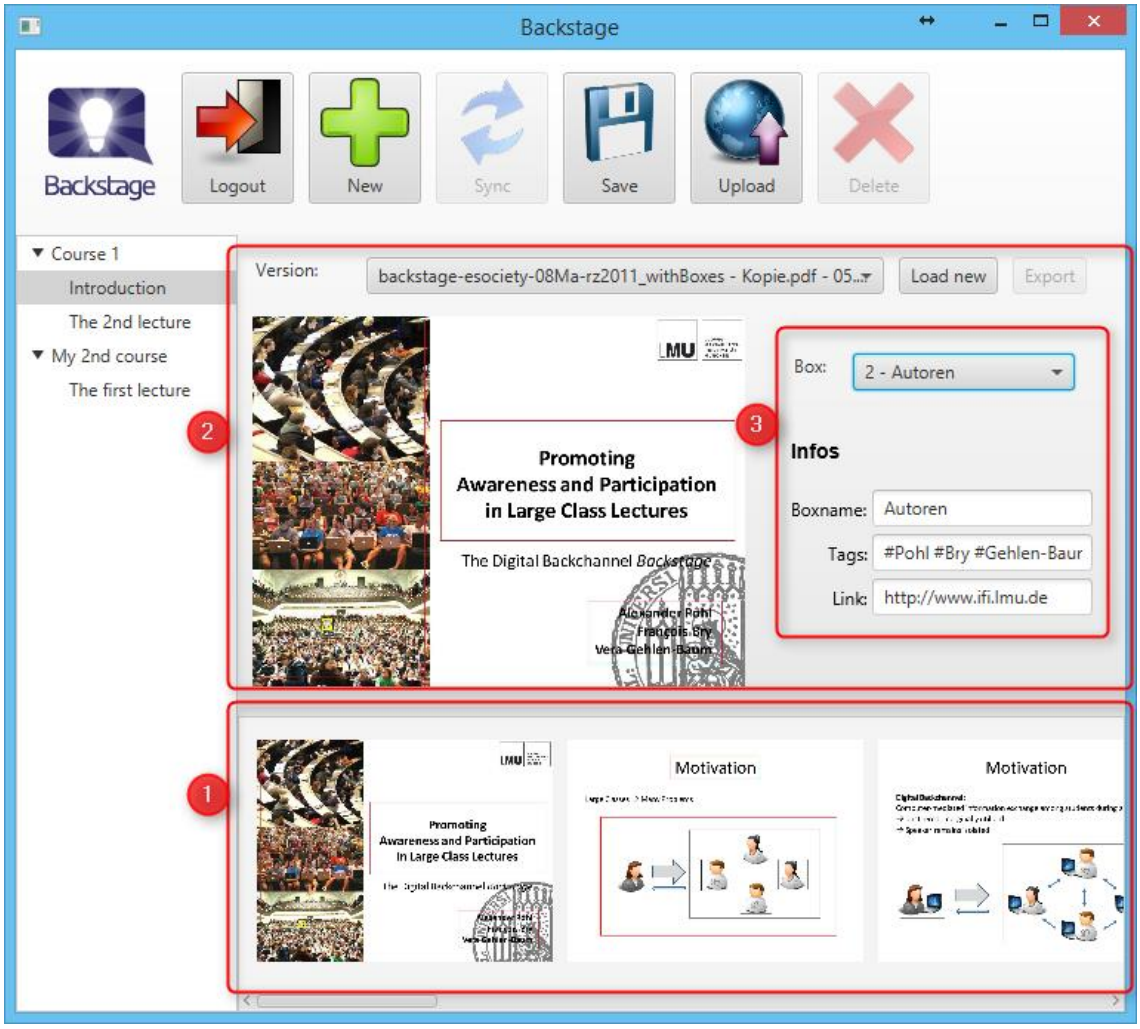


Fig. 6-d - Detail view of a lecture

Furthermore a new version of a lecture can be added by clicking the *Load new* button. After choosing the new PDF document, a view (Fig. 6-e) offers the possibility to merge the old and the new version. Therefore small pictures of the slides of the old and the new version are shown. Slides of these two versions can be copied into a new version by marking the desired slides and copying them into the third part of the view by using drag and drop. By clicking on the *Save* button, a new pdf document containing the chosen slides and *filter boxes* is created and added as new version to the detail view of the lecture.

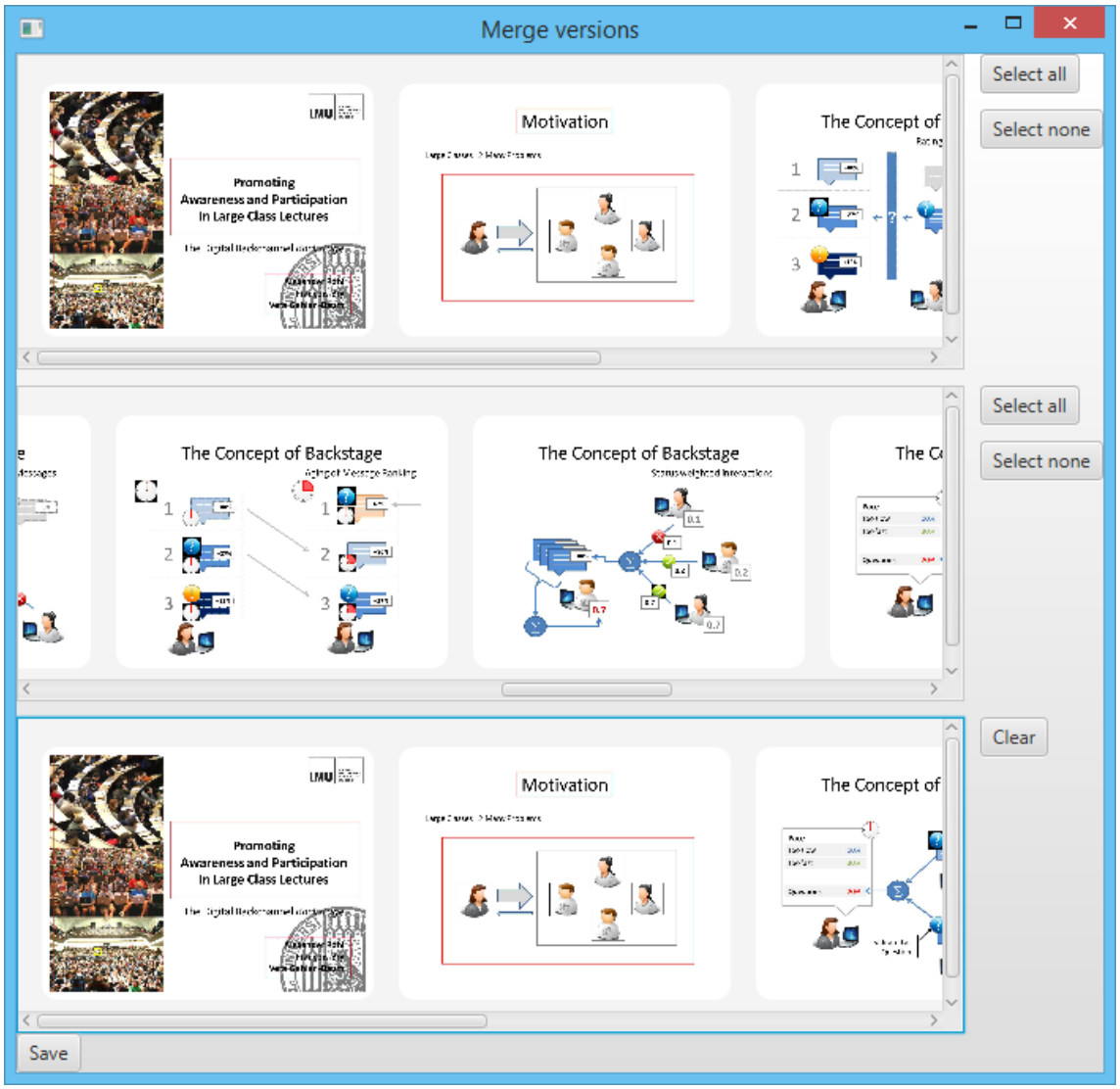


Fig. 6-e - Merging an old and a new version of slides

The lecturer can upload the slides to the server part of the *Box extractor* by clicking the *Upload* button. The *Box extractor* checks if all the attachments mentioned in the *filter boxes* exist and after that the pdf document, the information about the *filter boxes* and the attachments are uploaded using HTTP. The attachments are Base64[45] encoded to avoid problems concerning character encoding.

[45] http://en.wikipedia.org/w/index.php?title=Base64&oldid=642379746

6.2 The Server

The server part of the *Box extractor* is structured as shown in Figure 6-c. This means the server is waiting for a HTTP request from one of the clients. As soon as a request arrives the *Box extractor* creates a directory for the client's user on the hard disk and stores the PDF document, the attachments and a XML document containing all the information about the *filter boxes*. When the upload is complete a HTTP link, specified in the configuration of the server, is called to tell *Backstage* that there is a new course or lecture.

Now *Backstage* has the possibility to receive the PDF document, the attachments and the *filter box* information by using the REST interface of the *Box extractor*.

7 Conclusion and Future Work

Backstage has been introduced to offer a backchannel for a better communication during large-class lectures. As *Backstage* has been designed to be accessed from laptops and desktop computers, there are several problems to use *Backstage* with small screen devices as smartphones. The goal of this thesis has been to develop a concept for reconstructing relevant parts of a slide. Therefore, in Chapter 2, *filter boxes* have been introduced to mark relevant parts on a slide with rectangles. Moreover, in the description of these *filter boxes*, it is allowed to add supplementary information and material. In Chapter 3 different alternatives for the representation of slides in *Backstage* have been discussed. Whereas Section 3.1 has presented a new approach to create slides in HTML by using a WYSIWYG editor or converting a LaTeX document, Section 3.2 has carried on the current workflow of *Backstage* where PDF slides are transformed into raster graphics to show them on the website. For both approaches, it is possible to create *filter boxes* that mark relevant parts on the slides. The advantages and disadvantages of each approach have been discussed in Chapter 4 and the conclusion has been, that the best solution would be a hybrid environment that offers all approaches and the lecturer is able to decide which one she wants to use. But in the current version of *Backstage*, slides are integrated by rasterizing PDF slides. This, this thesis focuses on the creation of *filter boxes* on the basis of PDF slides, although alternative approaches may as well be worth considering. Finally the decision has been to implement a prototypic application independently from *Backstage* that extracts the *filter boxes* from PDF files. Chapter 5 has described the concept of the implementation and Chapter 6 has presented the design and usage of the implementation.

The *filter box* approach offers many new possibilities for *Backstage*, but also difficulties that have to be solved. *Filter boxes* need not only be relevant parts on slides marked by rectangles, also the annotations posted by the students and lecturers are able to be seen as *filter boxes*, as well as one or several slides can be a *filter box*. Furthermore with a suitable addressing scheme, *filter boxes* are able to be called by links, so that *filter boxes* can be referred and new ways of navigating through *filter boxes* can be implemented. The *filter box* approach is also not limited to smartphones, even the other users can benefit from the supplementary information and material the approach provides.

However, the (semi-)automatic posting of annotations mentioned in Section 2.4 and the different approaches in Chapter 3 for creating slides and *filter boxes* have to be analyzed. Therefore a user studies will help to find out which of the introduced approaches is acceptable for the lecturers and what kind of (semi-)automatic posting is best to determine a suitable location for an annotation posted by accessing *Backstage* with a smartphone. Moreover, the performance of processing a PDF file in the implemented application should be improved to ensure a better usability and an alternative Java library to replace Apache PDFBox used in the application should be found, because PDFBox is not able to process all kind of fonts and graphical elements of a PDF file. Finally, in the server part of the application further request for the REST interface can be useful to simplify the interaction between *Backstage* and the application.

Bibliography

[1] J. Cuseo, “The empirical case against large class size: Adverse effects on the teaching, learning, and retention of first-year students” in *The Journal of Faculty Development*, 21(1), pp. 1-22, 2007.

[2] W. J. McKeachie, “Research on College Teaching: The Historical Background” in *Journal of Educational Psychology*, 82(2), pp. 189-200, 1990.

[3] R. Andersson and T. Roxa, “Encouraging Students in Large Classes” in *SIGCSE '00 Proceedings of the thirty-first SIGCSE technical symposium on Computer science education*, pp. 176-179, 2000.

[4] K. A. Rocca, “Student Participation in the College Classroom: An Extended Multidisciplinary Literature Review” in *Communication Education*, 59(2), pp. 185-213, 2010.

[5] F. Bry, V. Gehlen-Baum, and A. Pohl, “Promoting Awareness and Participation in Large Class Lectures: The Digital Backchannel Backstage,” in *Proceedings of the IADIS Int. Conf. e-society*, Spain, Avila, pp. 27–34, 2011.

[6] F. Bry and A. Pohl, “Backstage: A Social Medium for Large Classes,” in *Campus Transformation – Education, Qualification and Digitalization*, F. Keuper and H. Arnold, Eds. Berlin: Logos Verlag, pp. 255–280, 2014.

[7] A. Pohl, F. Bry, J. Schwarz, and M. Gottstein, “Sensing the Classroom: Improving Awareness and Self-Awareness of Students with Backstage,” in *Proceedings of the International Conference on Interactive and Collaborative Learning*, 2012.

[8] V. Gehlen-Baum, A. Pohl, A. Weinberger, and F. Bry, “Backstage – Designing a Backchannel for Large Lectures,” in *Proceedings of the European Conference on Technology Enhanced Learning, Saarbrücken, Germany (18-21 September 2012)*, 2012.

[9] A. Pohl, V. Gehlen-Baum, and F. Bry, “Enhancing the Digital Backchannel Backstage on the Basis of a Formative User Study,” in *International Journal of Emerging Technologies in Learning (iJet)*, vol. 7, pp. 33–41, 2012.

[10] D. Baumgart, A. Pohl, V. Gehlen-Baum, and F. Bry, “Providing Guidance on Backstage, a Novel Digital Backchannel for Large Class Teaching,” in *Education in a Technological World: Communicating Current and Emerging Research and Technological Efforts*, A. Mendez-Vilas, Ed. Badajoz, Spain: Formatex Research Center, pp. 364–371, 2012.

[11] A. Pohl, V. Gehlen-Baum, and F. Bry, “Introducing Backstage – A Digital Backchannel for Large Class Lectures,” *Interact. Technol. Smart Educ.*, vol. 8, no. 3, pp. 186–200, 2011.

[12] M. Gottstein, "Increasing Social Awareness of the Lecturer on the Backchannel Backstage", Bachelorarbeit/bachelor thesis, Institute of Computer Science, LMU, Munich, 2012.

[13] J. Hadersberger, “Backstage Mobile – Merging Usability Guidelines with Educational Requirements", Diplomarbeit/diploma thesis, Institute of Computer Science, LMU, Munich, 2012.

[14] H. Oberdiek, “The zref package”, available at ftp://distrib-coffee.ipsl.jussieu.fr/pub/mirrors/ctan/macros/latex/contrib/oberdiek/zref.pdf, 2012.

Appendix

Directory structure of the *Box extractor*:

```
--Box_extractor.jar
--startClient.bat
--startServer.bat
--startClient.sh
--startServer.sh

--conf
   |
   |--clientSettings.ini
   |--serverSettings.ini

--src
   |
   |--client
   |      |
   |      |--common
   |      |      |
   |      |      |--ClientSettings.java
   |      |
   |      |--controller
   |      |      |
   |      |      |--LoadPDFStatusController.java
   |      |      |--LoginController.java
   |      |      |--MainController.java
   |      |      |--PDFMergeController.java
   |      |
   |      |--dialogfx
   |      |      |
   |      |      |--DialogFX.java
   |      |      |--DialogFXBuilder.java
   |      |      |--Dialog-accept.jpg
   |      |      |--Dialog-error.jpg
   |      |      |--Dialog-info.jpg
   |      |      |--Dialog-question.jpg
   |      |
   |      |--exceptions
   |      |      |
   |      |      |--CannotConnectToBackstage
   |      |                              UrlException.java
   |      |
   |      |--model
   |      |      |
   |      |      |--content
   |      |      |      |
   |      |      |      |--AnnotationInfo.java
   |      |      |      |--Attachment.java
   |      |      |      |--AttachmentType.java
   |      |      |      |--Course.java
   |      |      |      |--CourseManager.java
   |      |      |      |--Lecture.java
   |      |      |      |--Slide.java
   |      |      |      |--SlideList.java
   |      |      |
```

```
|      |      |--pdf
|      |      |      |
|      |      |      |--PDFManager.java
|      |      |      |--PDFMergeModel.java
|      |      |
|      |      |--upload
|      |      |      |
|      |      |      |--Connection.java
|      |      |      |--HTTPConnection.java
|      |      |      |--MultipartUtility.java
|      |      |
|      |      |--LoginModel.java
|      |      |--MainModel.java
|      |      |--PopupDialog.java
|      |      |--XMLManager.java
|      |
|      |--view
|      |      |
|      |      |--DropDownDialog.java
|      |      |--InputDialog.java
|      |      |--LoadPDFStatusView.java
|      |      |--LoginView.java
|      |      |--MainView.java
|      |      |--PDFMergeView.java
|      |      |--styleInputDialog.css
|      |      |--styleLoadPDFStatusView.css
|      |      |--styleLoginView.css
|      |      |--styleMainView.css
|      |      |--stylePDFMergeView.css
|      |      |--loading.gif
|      |      |--backstage_logo.png
|      |      |--backstage_logo_klein.png
|      |      |--delete.png
|      |      |--login.png
|      |      |--logout.png
|      |      |--new.png
|      |      |--save.png
|      |      |--sync.png
|      |      |--upload.png
|
|--main
|      |
|      |--Launcher.java
|
|--server
|      |
|      |--common
|      |      |
|      |      |--ServerSettings.java
|      |
|      |--model
|      |      |
|      |      |--content
|      |      |      |
|      |      |      |--AnnotationInfo.java
|      |      |      |--Attachment.java
|      |      |      |--AttachmentType.java
|      |      |      |--Course.java
|      |      |      |--Lecture.java
```

```
|      |      |      |--Slide.java
|      |      |      |--SlideList.java
|      |      |
|      |      |--upload
|      |      |      |
|      |      |      |--MyHttpServer.java
|      |      |
|      |      |--XMLManager.java
```

Starting the *Box extractor*:

- Edit *clientSettings.ini* and *serverSettings.ini* in the *conf* directory if necessary
- Start the server and the client component by running *startClient.(sh|bat)* and *startServer.(sh|bat)*
  The client component is started by running the *Box_extractor.jar*, the server component is started by running the *Box_extractor.jar* with *--server* as parameter

Using the REST interface:

`/course/`

➔ returns all courses

`/course/<name of course>/lecture/`

➔ returns all lectures for a course

`/course/<name of course>/lecture/<name of lecture>/`

➔ returns the name of all annotations of a lecture

`/course/<name of course>/lecture/<name of lecture>/slide/[0-9]+/`

➔ returns the name of all annotations of the slide with the chosen number

```
/course/<name of course>/lecture/<name of
lecture>/annotation/<name of annotation>/
```

➔ returns the content of the chosen annotation

Further interfaces can be defined in *src/server/model/upload/MyHttpServer.java*