TU WIEN

BACHELOR THESIS

# A new approach to present lecture slides online

*Author:*
Johannes DOSTAL

*Supervisor:*
Ao.Univ.Prof. Dipl.-Ing.
Dr.techn. Peter
PURGATHOFER

*A thesis submitted in fulfillment of the requirements for the degree of Bachelor of Science*

*in the*

Human Computer Interaction Group
Institute for Design & Assessment of Technology

March 7, 2017

# Declaration of Authorship

I, Johannes DOSTAL, declare that this thesis titled, "A new approach to present lecture slides online" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: 

Date: 7. March 2017

TU Wien

# *Abstract*

Faculty of Informatics
Institute for Design & Assessment of Technology

**A new approach to present lecture slides online**

by Johannes DOSTAL

This thesis introduces a new approach to present lecture slides online using a dynamic layout, additionally describing the way it has been conceived. In the next step, an overall picture of the implementation will be outlined and highly relevant parts will be explained in detail. A walkthrough will then show the resulting software from a front- and back-end perspective, including an explanation on how it will be operated. Finally, potential features and suggestions for future work will be mentioned.

# *Acknowledgements*

First of all, I would like to thank my supervisor Peter Purgathofer for the possibility to work on this exciting topic. I also want to thank him for the helpful conversations on the topic and even the cozy off topic chats, which have served to create a pleasant atmosphere to work in.
Thanks also to Rene Koller, who helped me with some troubleshooting in Aurora and Daniel Bauer, who was willing to proofread this thesis.
Furthermore, I want to thank my study colleagues for some input on my work and without whom I would have never been able to manage my studies up until now!
Finally, I want to acknowledge my family and friends, who supported me all the way through.

# Contents

# Chapter 1

# Introduction

## 1.1 State of the art

The lecture platform *Aurora* is used in the informatics lectures *Basics of Human Computer Interaction* (BHCI) and *Gesellschaftliche Spannungsfelder der Informatik* (GSI) at TU Wien. The code of the platform is open source and can be found on GitHub [1]. Aurora is based on the web framework Django [7], which is written in the programming language Python [12] and follows the model-view-template architectural pattern [11].
Aurora offers all information and functionality students need to successfully pass the courses. It shows students a summary of their progress in class and a newsfeed on the landing page. Furthermore, a challenge system is implemented, which serves as a sort of double blind peer review system, with which students review the submissions of others before the lecture team marks them.
Finally, all lecture slides are viewable and commendable in the system. This is the functionality to be improved with this work. Right now, students can choose a lecture date on the overview page 1.1, which displays all slides of the chosen date on one page, disregarding topic and affiliation 1.2. In order to display comments below each slide, a horizontal scroll layout was chosen. This confused a lot of students, therefore the note *hold down alt-key for mouse wheel horizontal scroll* had to be added to the page.

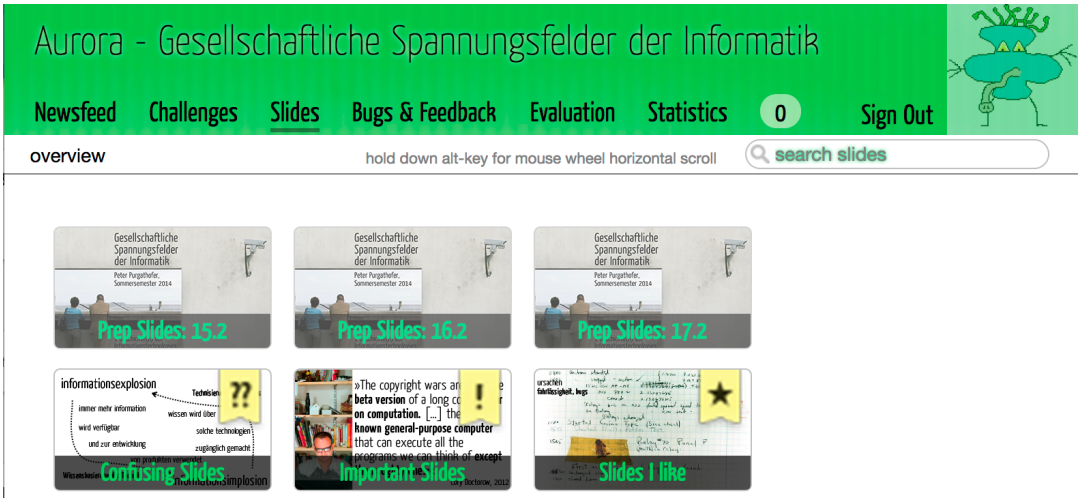



FIGURE 1.1: Overview of the lecture slides in the old version of Aurora.

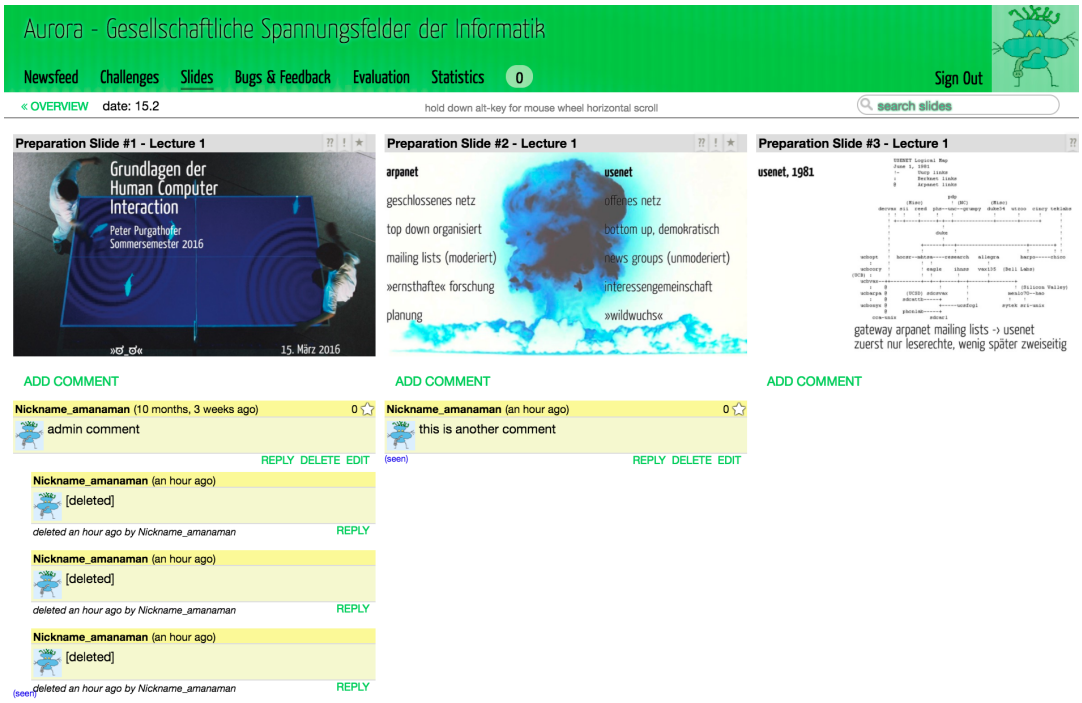



FIGURE 1.2: Slides view in the old implementation.

## 1.2 Conditions for new approach

Because of the static layout in which the slides are presented, it was not always easy for students to find the information they were looking for. Therefore, a new dynamic approach had to be developed. It should give lecturers the freedom to organize slides while creating them, without having to force the slides into a predetermined structure. The new layout should offer a way to structure slides by simply tagging them with a chapter and topic of choice. Furthermore, there should be a possibility to tag slides with simple keywords that fit the content of which the slides are comprised of. This should simplify finding the desired slides with the built-in search functionality.

# Chapter 2

# Concept for new approach

## 2.1 Finding the concept

Finding the optimal design for the new slides layout was a process that went through several iterations, involving the adjustment of a lot of small details. The whole process started with the desire to create a mobile app, capable of displaying slides on mobile devices. After drawing some sketches for the app, I took a first look at the source code of Aurora. Unfortunately, I discovered that the code of the slide implementation was generally messy and lacked proper documentation. These circumstances made it nearly impossible for me to implement an API for passing data to the mobile app. Therefore, my supervisor and I decided to create a new implementation for slides from scratch. The first approach we came up with was still static, but structured through topics instead of lecture dates.

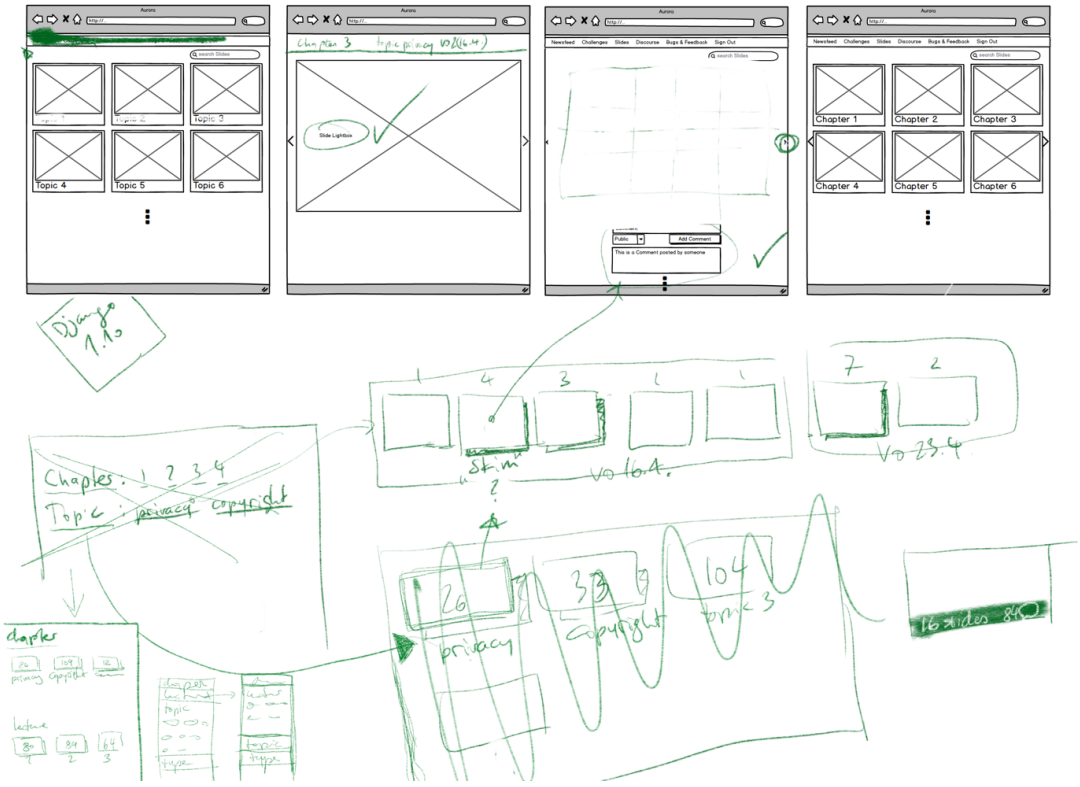



FIGURE 2.1: The sketches in the top are drawn with Balsamiq Mockups [4], and show the concept for the static layout. The green drawings are first ideas for the dynamic layout.

On the second meeting, we noticed that a static layout would force lecturers to fit their slides into predefined topics. It would also be complicated to add a new topic to the system every time one is needed and assign the slide to the right topic.
We started looking into a more flexible approach, that would not force lecturers into a given structure. We came up with the idea of simply tagging slides with the desired topic. We stuck to that concept and improved on it to order slides in a more structured manner.

## 2.2 Final concept

The final concept is shown in figure 2.2. It basically consists of three layers.
The bottom layer is an atomic container of slides, which belong together, inseparably. We call this slide container a *Slide Stack*. The bottom sketch in the image shows what a slide stack should look like. All assigned slides are displayed, along with a comment stream at the bottom. Instead of having a comment stream for every slide, one comment stream for a complete slide stack should help focus all comments belonging to a topic into one stream. This was a potential problem in the old system, since students did not always know where to add their comments, or they did not see that someone else already posted a similar comment below another slide.
The middle layer is *Topics*. Every slide stack is assigned to one or more topics. If a student views a topic page (*slide topics* in figure 2.2), all slide stacks assigned to this topic will be displayed.
The top layer of the structure is Categories. All topics are assigned to one category. There can be several topics with the same name, but each one can only be assigned to exactly one category. Categories are represented in the overview page (*slide contents* in figure 2.2). All allocated topics will be displayed for each category.

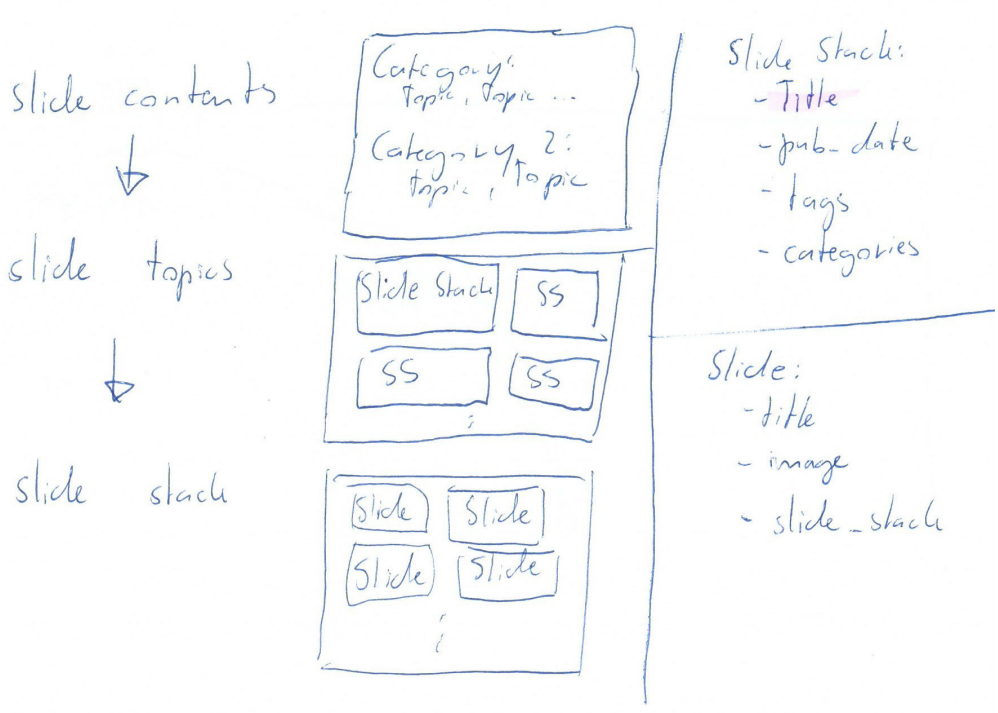



FIGURE 2.2: Sketches of the page hierarchy.

# Chapter 3

# Implementation

Since there was a possibility of using the slides functionality as a standalone application, it was implemented in a separate project. This project can be found on Bitbucket [3]. Later on it was merged into Aurora, after the decision to use it within the rest of the functionality was made. The code written in this work can be found in the *New Slides* branch in GitHub [2]. Using the Django framework [7] as base for Aurora, the implementation follows the model-view-template architectural pattern [11]. In this section, the most important functionality of the model and view layer will be explained, while the template layer is a dynamically generated HTML & CSS view, which will be shown in section 4.1.

## 3.1 Model layer

The model layer persists only of two models; a *Slide* model and a *Slide Stack* model. Because categories and topics are generated dynamically, they are not persisted as models.

### 3.1.1 Slide Model

The slide model represents a single slide entity and is always assigned to one slide stack. It consists of the following variables:

- **Title**: Displayed title.
- **Image**: Here, the image of the slide is saved.
- **Text content**: Plain text from the stored image.
- **Tags**: Keywords for search optimization.
- **Slide stack**: The slide stack to which the slide is assigned to.

### 3.1.2 Slide Stack Model

The slide stack model represents the slide stack as described in chapter 2.2. It is used for most operations within the package and stores the following variables:

- **Title**: Displayed title.
- **Slug**: Automatically generated slug for URL addressing, based on title.
- **Publish date**: Defines when the slide stack becomes available to students.
- **Tags**: Keywords for search optimization.

- **Categories**: Stores all categories separated by commas. One category looks like follows: *Category_Topic*.
- **Course**: Defines which course the slide stack belongs to (either GSI or BHCI).

## 3.2 View Layer

In order to show the dynamic structure of categories and topics correctly, helper methods and variables are used. They are found in the file *structures.py*. Right now, there are two of the same class, once for BHCI and once for GSI. Once for BHCI and once for GSI. Since these lectures will be merged and revised next year, the duplicated classes can be deleted and adapted for only one course easily.

### 3.2.1 Structure

The class *Structure* 3.1 computes a variable named *structure* (line 26) with the following layout: **[[Category, Topic, Topic, Topic,...]]**. So it is a list containing lists. Each of the inner lists contains at least two strings. The first one represents the name of the category, the second and onward are all topics assigned to this category.

```
1  class Structure:
2
3      @staticmethod
4      def create_structure():
5
6          structure = []
7          for slide_stack in SlideStack.objects.all():
8              for tup in slide_stack.list_category_tuples:
9                  try:
10                     inner_list = next(i for i in structure if
11                         i[0].lower() == tup[0].lower())
12
13                     category = inner_list.pop(0)
14                     try:
15                         next(it for it in inner_list if
16                             it.lower() == tup[1].lower())
17                     except StopIteration:
18                         inner_list.append(tup[1])
19                     inner_list.insert(0, category)
20
21                 except StopIteration:
22                     structure.append([tup[0], tup[1]])
23
24         return structure
25
26     structure = create_structure.__func__()
```

FIGURE 3.1: Code to generate the structure to be displayed.

To create this structure the function loops through all slide stack objects and checks if the structure list contains the Category_Topic tuples. If not, it will be appended to to the list. If the topic does not exist but the category does, the topic will be added to the list of the existing category.

### 3.2.2 Data Structure

Because querying all slide stack objects each time the page is loaded would lead to an immense amount of queries and corresponding loading times, this is done once when the server is turned on and stored in the class *DataStructure* 3.2. The data structure has the given format: **(String:category, [(String:topic title, [SlideStack])])**. In order to compute it, the method iterates through the structure created by the class Structure. For every Category_Topic tuple it searches all slide stack objects, to check if it contains the same tuple. If it does, it will be appended to the structure at the correct position.

```
class DataStructure:

    @staticmethod
    def create_data_structure():

        data_structure = []

        for lst in Structure.structure:
            category = lst.pop(0)

            topics = []
            for topic in lst:
                category_topic = category + '_' + topic
                slide_stacks = []
                for ss in SlideStack.objects.all():
                    if category_topic.lower() in (x.lower()
                        for x in ss.list_categories):
                        slide_stacks.append(ss)

                topics.append((topic, slide_stacks))

            data_structure.append((category, topics))

        Structure.redefine_structure()
        return data_structure

    data_structure = create_data_structure.__func__()
```

FIGURE 3.2: This code fills the structure computed with the appropriate data.

### 3.2.3 Views

The view methods are responsible to collect all necessary data and pass them to the templates, where they are rendered. The three main views are already shown in figure 2.2. There are two additional views, which will be described in the listing below.

- **Slides**: This method creates the overview page, showing all categories and topics. 4.1a
- **Slide Topics**: Here, a view of all slide stacks assigned to the given topic is created.
- **Slide Stack**: Shows all slides allocated to the chosen slide stack.
- **Search**: Searches for all slide stacks fitting the given search criteria. The view generated looks equally to *Slide Topics*.
- **Refresh Structure**: This is an internal functionality, only available for system administrators. When called, it refreshes the structures stored in Structure and DataStructure. Since these are created when the server starts, they have to be refreshed manually after adding new slides to the system.

## 3.3 Third party libraries and code snippets

A lot of functionality is owed to third party libraries and code snippets. They will be introduced and their usage briefly explained.

- **Unique Slugify**: A unique slug is needed to identify slide stacks vie URL correctly. Since Django provides only a slugify method, that is not capable of making them unique, this code snippet [13] is used instead.
- **Easy Thumbnails**: Easy thumbnails [8] is a python package, capable of generating thumbnails of images in the desired size.
- **FrameScrub**: FrameScrub [10] is a JavaScript and CSS plug-in, that mimics the iPhoto album mouseover event for websites.
- **Bootstrap**: Bootstrap is a front-end CSS framework. It is primary used for its responsive design. It allows to view slides on either small or large displays. The used version is Bootstrap 4 alpha [5].
- **Ekko Lightbox**: Ekko lightbox [9] is a lightbox specially created for the use in combination with Bootstrap. Since most other lightboxes have code conflicts with Bootstrap, it is not advisable to use one not coordinated with bootstrap.

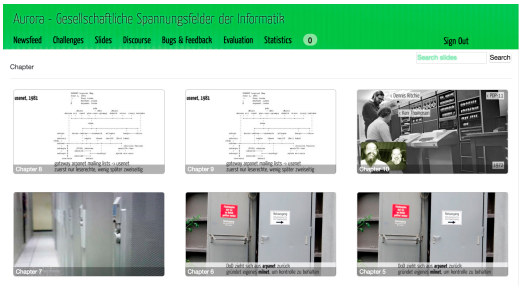
# Chapter 4

# Software walkthrough

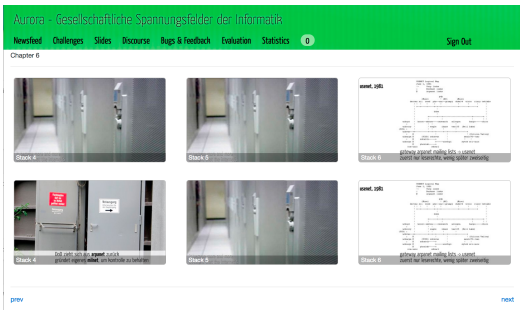
In this chapter the new slides front- and back-end are explained. While the front-end is what students are going to see, in the back-end lecturers can upload and organize their slides.

## 4.1 Front-end

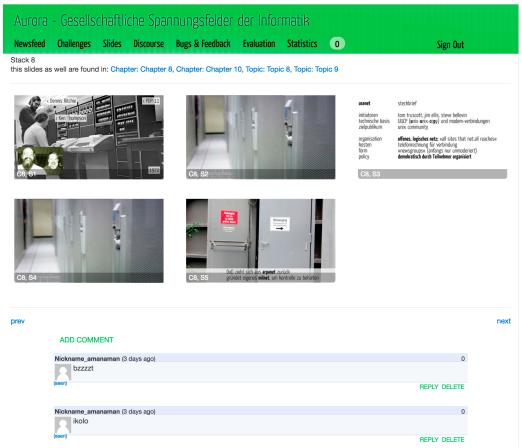
Here students will find all lecture slides uploaded to the new, dynamic system. The structure is as described in chapter 2.2. Figure 4.1a shows the overview page, where all categories and the allocated topics are displayed. When choosing one of the topics, students will see the slide topic page as shown in figure 4.1b.



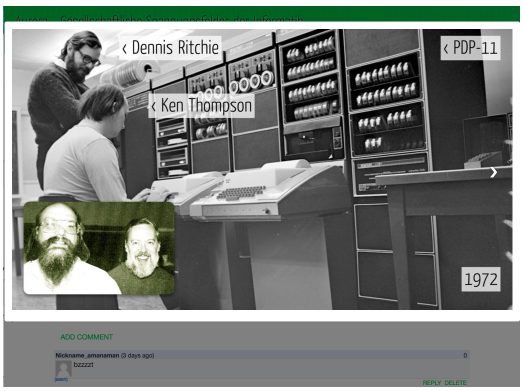

(A) Overview page, shows the categories and all assigned topics.




(B) Content of a topic: All allocated slide stacks are displayed.




(C) View of an slide stack. The assigned slides as well as a comment stream are shown.




(D) After a click on a slide in slide stack, a lightbox is opened.

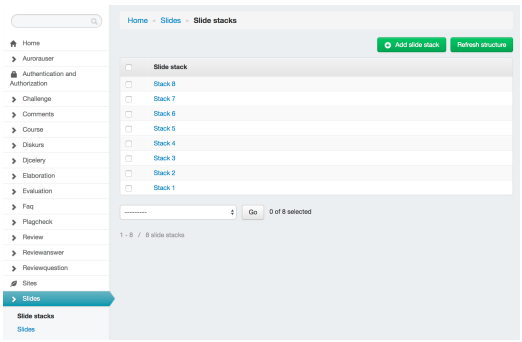
This page lists all slide stacks, assigned to the chosen topic. Going further and choosing one of the slide stacks leads to the view of figure 4.1c. Here students can view single slides and enlarge them with a lightbox (figure 4.1d).
Beside the front-end navigation, the implemented responsive design is worth mentioning. To achieve this, the CSS & JavaScrip front-end framework Bootstrap is used in version 4 alpha build 5 [5]. To change the number of slides in a row, the grid-system of Bootstrap is applied [6]. Depending on the size of the display, a fitting number of slides for one row is chosen. Therefore, five display widths are predefined by Bootstrap: **Extra small** <576px, **Small** >=576px (not used here), **Medium** >=768px, **Large** >=992px and **Extra large** >=1200px. Whenever the browser window size changes, the order of slides will also be redefined dynamically.

## 4.2 Back-end

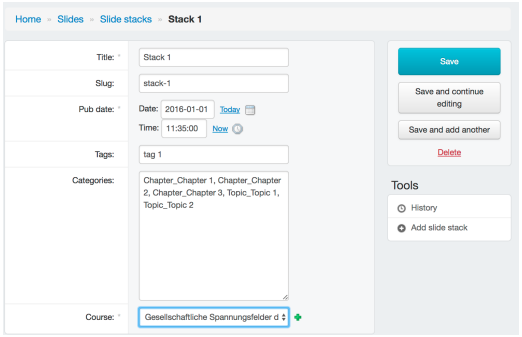
The back-end is for lecturers and authorized staff only. Here, they can upload and (re-)organize their slides. There are two approaches to do so. On the one hand, there is a GUI for easy creating and editing single slides and slide stacks. On the other hand, a shell is offered for easy uploading of several slides at the same time, but it is not suitable for editing variables of slides.

### 4.2.1 GUI

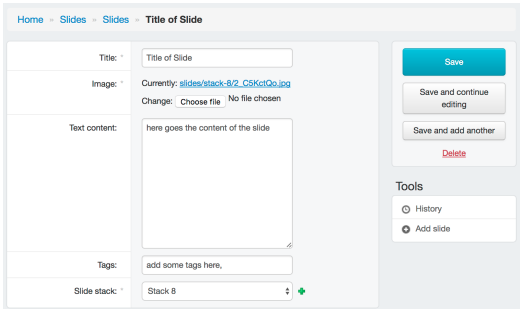
The admin site of the system offers a GUI where slides 4.2c and slides stacks 4.2b can be created and managed separately. Here, all variables can be viewed and edited as desired. Only slugs for slide stacks can not be changed manually (and are not needed to be so).




(A) Admin panel for slides.




(B) Editing dialogue for a slide stack.




(C) Editing dialogue for a slide.

After adding slides, it is important to refresh the data structure (described in chapter 3.2.2) stored by the server. This structure is automatically generated after the server is started. When new slides are uploaded, they are not automatically considered by the data structure, unless the server is restarted. Therefore, the data structure has to be refreshed with the button in the upper right corner of figure 4.2a. Depending on the amount of slides stored on the server, refreshing the data structure can take some minutes, since all slides have to be called several times in different queries.

### 4.2.2 Python shell

Beside the GUI, slides can be added to the system via a python shell. This allows uploading multiple slides faster than adding them separately to the system via the GUI. In order to do so, the script shown in figure 4.3 has to be filled and executed. The necessary variables are commented in the script and described in the chapters 3.1.2 and 3.1.1.
After the script is executed, the data structure has to be refreshed via the GUI. The data structure does not persist in the database. Since the upload-shell does not share variables with the server-shell, this has to be done via the GUI, which has access to the server-shell, and therefore can change the data structure.

```
from django.core.management.base import BaseCommand
from django.core.files import File
from django.utils import timezone

import random
import os

from Slides.models import SlideStack, Slide
from Course.models import Course
from AuroraProject.settings import STATIC_ROOT


# Fill in data for SlideStack here:
slide_stack_title = ''        # String
slide_stack_publish_date = timezone.now()
# YYYY-MM-DD HH:MM[:ss[.uuuuuu]]
# selecting now: timezone.now()
slide_stack_tags = ''
# Strings seperated with comma # optional variable
slide_stack_categories = '' # String with given layout:
# Chapter_Topic. Multiple categories can be seperated by ','
# optional variable
slide_stack_course = Course.objects.get(short_title='gsi')
# replace gsi with hci if needed
# Fill in end

SlideStack.objects.create(
        title=slide_stack_title,
        tags=slide_stack_tags,
        categories=slide_stack_categories,
        pub_date=slide_stack_publish_date,
        course=slide_stack_course)

# Fill in data for Slide here:
slide_title = ''        # String
slide_image_path = ''        # right now the folder/file has
# to be in 'img' folder on server
slide_text_content = ''        # String # optional variable
slide_tags = ''        # Strings, seperated with commas
# optional variable
slide_slide_stack = SlideStack.objects.last()
# the last created slide stack will be used,
# for other allocation use the GUI
# Fill in end

path = os.path.join(STATIC_ROOT, 'img/%s' % slide_image_path)
Slide.objects.create(title=slide_title,
        text_content=slide_text_content, tags=slide_tags,
        slide_stack=slide_slide_stack)
Slide.objects.last().image.save(im, File(open(path, 'rb')))
```

FIGURE 4.3: With this code, slides can be created on the server.

# Chapter 5

# Summary and future work

## 5.1 Technical conclusion and future implementations

In this work, a new approach for presenting lecture slides online has been designed and realized for the lectures BHCI and GSI. The implementation itself works fine and some bugs found by my supervisor have already been fixed, but there is still room for improvements. The setup process of the software could be streamlined, since some code has to be commented out and in during the process. Furthermore for test environments populating demo data has to be reworked. Right now, there are problems with some data from the old script, that seems to be incompatible with the used database.
A feature not implemented yet, requested by my supervisor, was to display a comment counter for each slide stack in slide topics 4.1b. Since I did not implement the comment functionality and no such function exists, it has to be implemented before it can be added to the slide stacks. Another feature a colleague of mine mentioned, was the possibility to offer downloads of whole slide stacks. Right now, the only way to do so is to manually download the pictures of each slide separately.
Although the chosen implementation has a responsive design, it stands to discussion to implement a native smart-phone app. This would allow design choices optimized for mobile only and remove mobile unoptimized functionality such as the challenge system, which would lead students to the assumption they can solve their tasks on the go and impair the quality of submissions.

## 5.2 Evaluation and improvement of approach

Pending minor technical improvements aside, an evaluation with students is more relevant for further improvements. This will be done during my employment as tutor in the upcoming semester. As such, some students, who know the old approach already will be invited for an interview. Furthermore, all students could be asked about their opinion and how they would improve the functionality in a survey.
Based on the feedback gained, further improvements could be made to adapt the approach better suit the students needs.

# Bibliography


[1] *Aurora*. https://github.com/martflu/aurora. Accessed: 2017-02-05.

[2] *Aurora - New Slides*. https://github.com/martflu/aurora/tree/new_slides. Accessed: 2017-02-06.

[3] *Aurora slides standalone application*. https://bitbucket.org/e1226714/aurora-lite/src. Accessed: 2017-02-06.

[4] *Balsamiq Mockups*. https://balsamiq.com/. Accessed: 2017-02-06.

[5] *Bootstrap 4*. https://v4-alpha.getbootstrap.com/. Accessed: 2017-02-07.

[6] *Bootstrap 4, Grid System*. https://v4-alpha.getbootstrap.com/layout/grid/. Accessed: 2017-02-10.

[7] *Django*. https://www.djangoproject.com/. Accessed: 2017-02-06.

[8] *Easy Thumbnails*. https://github.com/SmileyChris/easy-thumbnails. Accessed: 2017-02-07.

[9] *Ekko Lightbox*. http://ashleydw.github.io/lightbox/. Accessed: 2017-02-07.

[10] *FrameScrub*. http://www.mikcro.com/frameScrub/. Accessed: 2017-02-07.

[11] Julia Plekhanova. "Evaluating web development frameworks: Django, Ruby on Rails and CakePHP". In: *Institute for Business and Information Technology* (2009).

[12] *Python*. https://www.python.org/. Accessed: 2017-02-06.

[13] *Unique Slugify*. https://github.com/defcube/django-unique-slugify. Accessed: 2017-02-07.