



Bachelor's Thesis

Green VVZ

A web application to find and display modules related to sustainability at the University of Zurich.

Bodo Brägger

bodo.braegger@uzh.ch

16-731-671

Informatics and Sustainability Research

Department of Informatics

University of Zurich

Zurich, Switzerland

Advisor: Jan Bieser

Supervisor: Prof. Dr. Lorenz Hilty

Submission: 04.07.2019

Table of Contents

Abstract	4
Zusammenfassung	5
1 Introduction	6
1.1 Overview	6
1.2 Terminology	6
1.3 Problem Statement and Goals	7
2 Approach	9
2.1 Status Quo Analysis and Rebuild	9
2.2 Selection of Software Process Model	10
2.2.1 Waterfall Model	10
2.2.2 Spiral Model	11
2.2.3 Agile Manifesto and Models	12
2.2.4 Feature Driven Development	13
3 Requirements	15
3.1 Filter Modules by Semester	15
3.2 Filter Modules by Study Program	16
3.3 Improved Keyword Search	16
3.4 Matching the UZH Corporate Design	16
3.5 Ease of Setup	17
4 Implementation	18
4.1 Implementation Iterations	18
4.1.1 First Iteration: Revamping the Data Collection and Storage	18
<i>Requirements</i>	18
<i>Understanding the UZH Course Catalogue API</i>	19
<i>Data Integrity</i>	19
4.1.2 Second Iteration: Improving the Search, Decision on Fundamentals	19
<i>Requirements</i>	20

<i>Meeting with UZH Course Catalogue Expert</i>	20
<i>Speeding Up the Search: Parallelizing Web Requests</i>	21
<i>Front End Development and Feedback</i>	21
4.1.3 Third Iteration: Finalizing the Design	22
<i>User Feedback</i>	22
<i>Drop Down and Incremental Search</i>	23
<i>Overview and Traceability in the Module Tables</i>	23
<i>Client-Side Input Validation</i>	24
4.2 Technologies and Tools	24
4.2.1 Traceability: Git and GitHub	24
4.2.2 Back End: Flask, MariaDB, UZH OData API and Jinja2 Templating	25
4.2.3 Front End: JavaScript, jQuery and Ajax	26
4.2.4 Content Management System: Embedding into Magnolia	26
5 Documentation	28
5.1 User Guides	28
5.2 Code Documentation	28
6 Discussion	29
6.1 Reflection	29
6.2 Outlook	29
7 Conclusion	30
8 Bibliography	31
9 Tables and Figures	34
9.1 List of Tables	34
9.2 List of Figures	34
Appendix	35
10 Source Code	35
11 Documents	36
11.1 Guides	36
11.2 Code Documentation	57

Abstract

The result of this applied bachelor thesis is the analysis, extension and reworking of the web application built in the bachelor thesis *Nachhaltigkeits-Vorlesungsverzeichnis* by Lukas Grässle (Grässle, 2018).

The objective is to provide a dynamically generated list of courses related to sustainability topics based on the course catalogue of the University of Zurich (UZH). The web application is hosted on a server provided by the UZH Department of Informatics and a public view will be embedded into a webpage of the UZH Sustainability team. It also features a private view for administration of the modules and the keywords provided for the course catalogue search.

The purpose of this project is to rework and improve the previous tool and extend it by certain crucial features:

- Filtering modules according to study program
- Viewing courses of previous semesters
- Adding modules via automated recommendations based on the keyword search, for terms in titles, descriptions, or names of instructors, both for modules directly and for modules containing matching courses
- Comprehensive code documentation for reusability and extensibility

The importance of documentation is stressed, to make maintenance and further extension easier – as well as comprehensive user guides to make the tool accessible without additional training.

Zusammenfassung

Das Endprodukt dieser angewandten Bachelorarbeit ist die Analyse, Erweiterung und Überarbeitung der Webanwendung, die im Rahmen der Arbeit *Nachhaltigkeits-Vorlesungsverzeichnis* von Lukas Grässle (Grässle, 2018) entstanden ist.

Das Ziel der Webanwendung ist es, auf Basis des Vorlesungsverzeichnisses der Universität Zürich (UZH), eine dynamisch generierte Liste von Modulen zu Nachhaltigkeitsthemen bereitzustellen. Die Webanwendung wird auf einem Server des Instituts für Informatik UZH bereitgestellt und eine öffentliche Ansicht wird in eine Webseite des UZH-Nachhaltigkeitsteams eingebettet. Es enthält zudem eine Administratorenansicht zur Verwaltung der Module sowie der Schlüsselwörter für die Kurskatalogsuche.

Das vorherige Tool soll überarbeitet und verbessert sowie um die folgenden Funktionalitäten erweitert werden:

- Module nach Studienprogramm zu filtern
- Module früherer Semester einzusehen
- Das Speichern von Modulen aufgrund der automatisierten Stichwortsuche nach Lehrveranstaltungen
- Umfassende Code Dokumentation

Die Bedeutung einer umfassenden Dokumentation samt Bedienungsanleitung wird hervorgehoben, um die Benützung, Wartung und Erweiterung zu vereinfachen.

1 Introduction

In this chapter the initial problem statement and the project will be introduced. Furthermore, an overview over the body of this thesis will be presented.

1.1 Overview

The Green VVZ is a web application consisting of a front and back end, which interacts with the OData API of the official course catalogue of the University of Zurich (UZH) to store and serve modules relating to certain keywords.

The foundation is a keyword search, which finds modules associated with the search terms directly based on their title or description, as well as modules containing matching courses. These suggested modules are presented on an administrator view, where it is possible to curate the search terms, as well as choose which modules are to be saved in a whitelisted or blacklisted state. Furthermore, a public view containing only the whitelisted modules is made available, and both public and private administrator view offer filtering by semester and study program.

1.2 Terminology

This is a brief overview of the most important terms used throughout this thesis, and their meaning. The definition of the following terms comes from the Academic Program Development of UZH (University of Zurich, 2018):

Term	Meaning
Course	"A program of instruction, as in a college/university; A prescribed number of classes in a particular field of study (a course of study)." (University of Zurich, 2010a)
Module	"A module can consist of one or more courses and is, as such, the broadest term for course types at the University of Zurich." (University of Zurich, 2010b) This is what is most relevant to a student, as he is only able to book and receive credits for completed modules.
Study program	A part of a degree program, with a set number of ECTS points prescribed. It is common for a degree program (which is awarded with an actual degree diploma) to be made up of combination of a major and minor study program (University of Zurich, 2010c).

Table 1: Terms

For more in-depth information on UZH terminology, consult the Academic Program Development page directly (University of Zurich, 2018).

1.3 Problem Statement and Goals

The Informatics and Sustainability Research (ISR) group initially commissioned the *Nachhaltigkeits-Vorlesungsverzeichnis*, which was built as part of the bachelor's thesis of Lukas Grässle (Grässle, 2018), with the goal of displaying a list of modules relating to sustainability on the Sustainability website. Its functionality was more basic than the current state of the Green VVZ – with saved module data being limited to its title, and whether it is offered in the spring or fall semester, or both. No study program data was captured, and consequently, no filtering features were available. The written part of the thesis served as the documentation for it, but there was no working instance or other documentation to exemplify or explain the process of setting the tool up or maintaining and extending it. It was initially unclear whether the source code was available and complete, as the tool was never hosted on an UZH server before.

This, and especially the unavailability of the previous maintainer led to these initial questions:

- Is there a complete codebase available?
- Is the codebase up to date and working?

Because of the initial cost associated with simply setting up a new instance of the web application with minimal changes required to get it running, it was decided that the project should be hosted on a virtual private server hosted by the UZH. Furthermore, the importance of comprehensive documentation and guides on how to set the project up was realized, and guides should be made available to decrease the time and cost of setting up new instances of the application should the need arise. This led to the next question:

- Will the project integrate into a different serving environment seamlessly?

Answering these questions was the initial problem statement. The plan was to set up the project on an UZH server as it was prior to its support ceasing, and in a next step, define a new project along with requirements for extending it by crucial features and documentation.

On top of the initial problem statement and a working instance of the *Nachhaltigkeits-Vorlesungsverzeichnis* (Grässle, 2018), certain new key functionalities were required and their implementation establish the basis of this applied thesis. The following list contains the high-level goals for added functionality of the Green VVZ:

- The ability to filter modules according to semester in which they are offered.
- The ability to filter modules according to study program.

- The ability to view modules of previous semesters on the public front end as well as the private administrator view.
- The ability to add modules via automated recommendations based on the keyword search, for terms in titles and descriptions, both for modules directly and for modules containing matching courses.
- Matching the UZH corporate design for front-end components.
- The ability to easily set up a new instance and to get it operational with minimal training.

In a final step, comprehensive documentation of both back and front end was to be created where it was not already during development, including guides on how to set up an instance for both development and production.

The applied thesis was defined around the implementation of above functionality and the creation of the required documents. This work will serve to summarize the work done and elaborate on both the process and the end result. Furthermore, all end products can be found in the appendix.

2 Approach

In this section, the approach and the process of implementation will be explained. This chapter is organized both thematically and chronologically. For more technical details on the implementation, refer to chapter 4: *Implementation*.

2.1 Status Quo Analysis and Rebuild

As this project was not started from scratch, but as a continuation of the *Nachhaltigkeits-Vorlesungsverzeichnis* (Grässle, 2018), the first step was to analyse the status quo. As the maintainer was not available, and the current ISR team had no experience with the implementation and background of the *Nachhaltigkeits-Vorlesungsverzeichnis* (Grässle, 2018), it was not clear at first whether the codebase was readily available at all. Thankfully, a working link to the GitHub repository is available in the original thesis (Grässle, 2018, p.10), which hosts the complete code – with some configuration information stripped.

It was decided, in communication with Jan Bieser and Linde Warland, that the ISR Group wants to host the server on a machine of the UZH for the future to have better control over the project. Before further requirements were to be discussed and planned, it was a prerequisite to get the project up and running in its current state, in order to be able to test and see where functionality was missing and in what direction the project should go.

Since the application was to be deployed on an internal UZH server, it needed to conform to the hosting packages supplied by the Department of Informatics of UZH. Since this differed from the server setup used in the original project, certain changes had to be implemented and tested – most notably, the switch from Python 2.X to 3.X to avoid deprecation.

Once the project was up and running based on the initial code base, it was possible to formulate the high-level goals for this thesis, as defined in chapter 1.3: *Problem Statement and Goals*.

Since the added functionality required a considerably extended data basis in comparison to the original project, it required fundamental changes in the architecture and code. To keep the development manageable and comprehensible to possible team members or future maintainers, it is best to look into typical software process models and select a proper model after assessing the scale of the project.

Since this project was implemented by one person, with the possibility of future maintainers or teams working on it, a good balance between clarity of code, facility of development and minimal overhead through process activities were key.

2.2 Selection of Software Process Model

2.2.1 Waterfall Model

There are several major software process models: one of the oldest and most well-known is the waterfall model (Royce, 1987). In the waterfall model, development is segmented into clear phases, and development moves 'downstream', with moving on only when a step is verified and tested. Previous steps are revisited only if verification fails.

It is thus best used for large, well understood and complex systems. Since this comes with a lot of initial planning overhead, and the Green VVZ is built on top of an existing codebase where possible, the waterfall model was not deemed efficient.

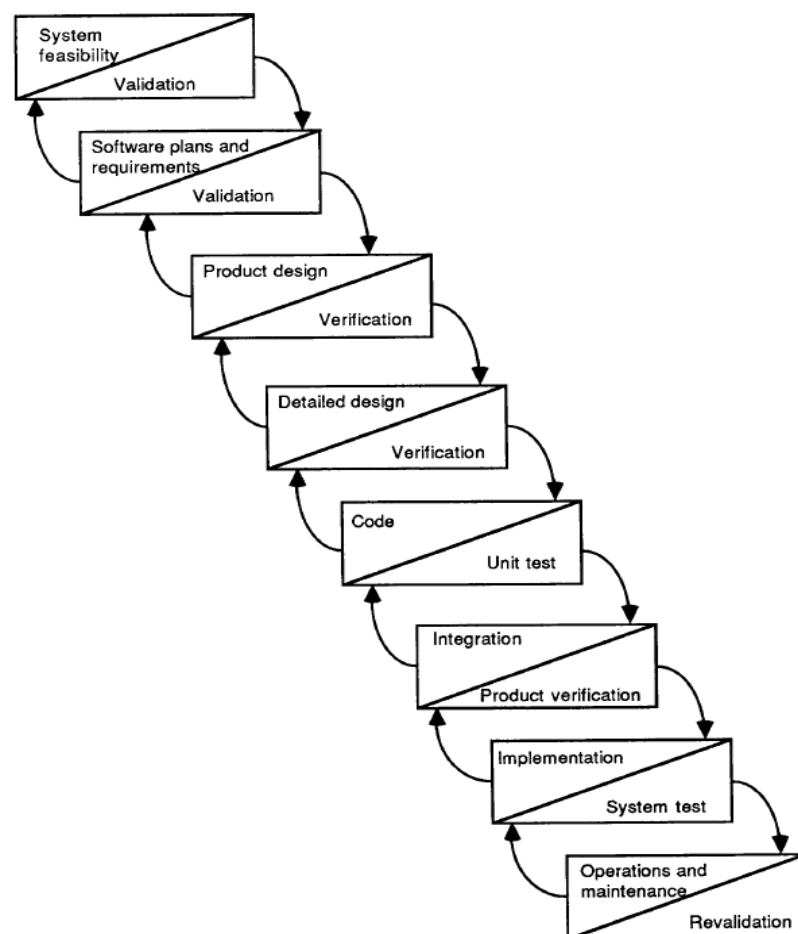


Figure 1: The waterfall model (B. W. Boehm, 1988, p. 62)

2.2.2 Spiral Model

Another important software process model is the spiral model (Boehm, 1988). It is cyclic in nature – meaning there are certain development cycles or evolutions with each their own planning, requirements and implementation phases. Boehm stresses that the spiral model is not just a cyclic application of the waterfall model – understanding it as such would risk failure – he goes as far as to call a cyclic waterfall model a “hazardous spiral look-alike” (B. W. Boehm & J Hansen, 2001, p. 5). The major factor that sets the spiral model apart is that it is risk based; in every iteration, the risk of the requirements to be implemented, and failure of implementation is assessed in terms of risk for the project.

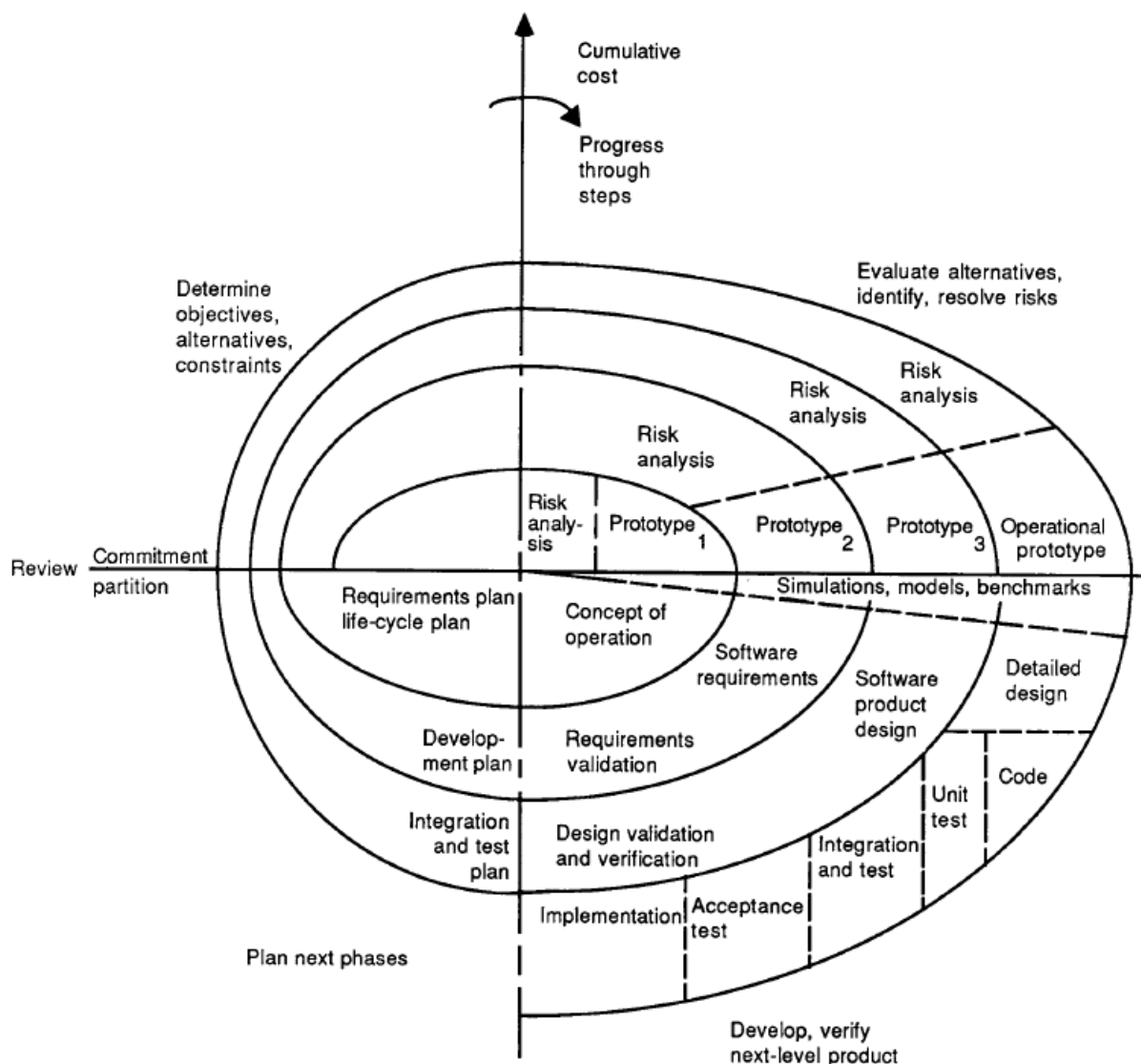


Figure 2: The spiral model (B. W. Boehm, 1988, p. 64)

Whilst this would be an overabundance for a one-man development team, the idea of iterations or evolutions in terms of continuous reassessment of the project, the requirements, and necessary changes is a good idea for teams of any size. Thus, this iteration focus was applied in the development of this project.

2.2.3 Agile Manifesto and Models

Even though there are several process models based on the agile principles, such as SCRUM (Schwaber, 1997), which is tried and true for small development teams (Rising & Janoff, 2000) or Extreme Programming (Beck, 1999), the process for a smaller project such as the Green VVZ was able to rely on the fundamental agile values and principles (Beck et al., 2001) without following any specific process model exactly. It is instead picking the relevant aspects of each process, such as the verification used in the waterfall model or the cyclic iterations introduced in the spiral model and used in several agile models, and adapting it to a project of scale.

The *Manifesto for Agile Software Development* (Beck et al., 2001) defines four major values each with their own counterpart:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan (p. 1)

Whilst these values are perfect for low risk and low criticality projects, which the Green VVZ can be classified as, there is one big deviation: comprehensive documentation became important due to the nature of often changing maintainers and team members, making direct collaboration and communication impossible long term. Thus, writing guides and documentation are an essential bridge in time between developers.

The more specific and explicitly defined agile principles (Beck et al., 2001) deemed most important for the successful development of this project were:

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

- Welcome changing requirements, even late in development. Agile processes harness change
- Deliver working software frequently
- [Owners / users] and developers must work together daily throughout the project.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Working software is the primary measure of progress. (pp. 2-3)

2.2.4 Feature Driven Development

With these values and principles in mind, there is a certain process model which fits the process for the Green VVZ closely. Coupled with the fact that there is a previously developed overall model, the *Nachhaltigkeits-Vorlesungsverzeichnis* (Grässle, 2018), and an evolving, but distinct feature list, make the newer and relatively simpler agile process of feature driven development (FDD) a good match for the Green VVZ as by FDD's description (Abrahamsson, Salo, Ronkainen, & Warsta, 2017; Palmer & Felsing, 2001). Development revolves around a set of features, grouped by functionality and commonalities in design and required changes. Thus, it is possible for the developer to group features according to ease of development and importance to the involved stakeholders, making it easy to keep an overview of the course of implementation for the developers and keeping it transparent towards the involved stakeholders. Users can be actively involved in the creation of the feature set, and also in the design phase. These were the key takeaways used for the development process of the Green VVZ.

Although the Green VVZ's development process does not match FDD exactly, as it is less formalized, and has less stringent iteration durations (with features being grouped into a single iteration of possible longer durations), it was a good basis for the structuration and assessment of the process.

Furthermore, FDD does not explicitly define the managing of requirements, which further matches the development of Green VVZ, with no formal requirements document nor user stories.

FDD features a 'Design by Feature' paradigm, where features are classed according to priority and dependencies, and then development progresses by planning by feature, then designing by feature, before finally implementing.

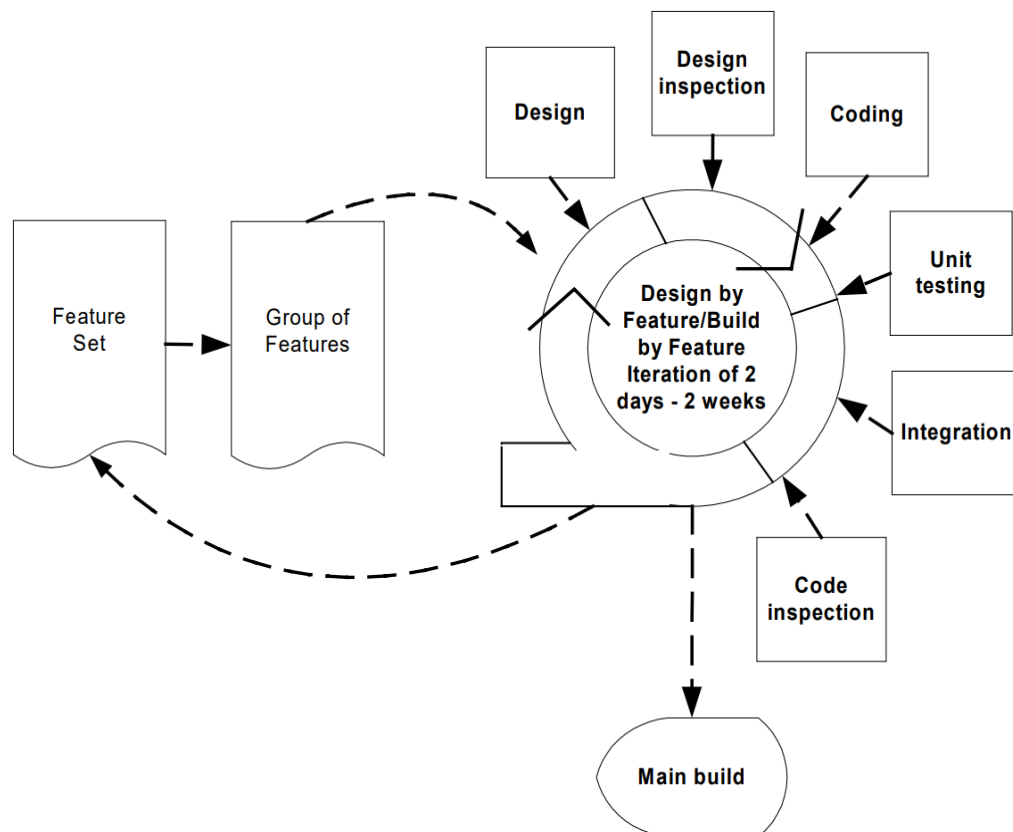


Figure 3: The design processes of FDD, (Abrahamsson et al., 2017, p. 50)

FDD's central 'Design by Feature' paradigm can be summarized as a small group of features being implemented in a time span of 2-14 days, and integrated into the product after their completion (Abrahamsson et al., 2017; Palmer & Felsing, 2001). This was perfect for the Green VVZ, with the initial feature list to work from, and the possibility of continuous feedback by user representatives (the ISR group). Unit testing, integration and code inspection were not applied in the Green VVZ, as they exceed the scope of the project.

In conclusion, the process to follow chosen for this project is best described as a small-team, iterative and highly agile oriented mode of development with an emphasis on responding to continuous feedback by the user and being ready for change. It is most closely matched by the FDD process (Abrahamsson et al., 2017; Palmer & Felsing, 2001), albeit even less formalized to make for less overhead. There were 3 major development cycles, marked by increased levels of communication with the ISR team in the beginning. The implementation is documented in chapter 4: *Implementation*.

3 Requirements

As general and atomic requirements were defined in the *Nachhaltigkeits-Vorlesungsverzeichnis* (Grässle, 2018), the creation of an explicit and formal requirements document was deemed unnecessary for a project of the scope of the Green VVZ. Instead, a set of goals (or features) was created in the beginning in consultation with user representatives of the ISR group and adapted throughout the course of development where necessary. Planning, implementation and feedback were based around these features, as detailed in chapter 4: *Implementation*.

The implementation requirements are listed in the following tables by feature, broken down by type of technology used for implementation. These implementation requirements written for each feature are manageable chunks which are implementable and verifiable independently.

3.1 Filter Modules by Semester

This feature also includes viewing modules of previous semesters.

Filtering by Semester [Back end]	
Type	Implementation Requirement
Database	Changing the database design to allow for saving of the same module with different semester / year data.
Flask	Processing and saving the semester data correctly from the UZH course catalogue, using the JSON interface.
Flask, Jinja2	Serving the stored module data in JSON format.
Flask, Jinja2	Serving the 8 relevant semesters based on the current date.
Filtering by Semester [Front end]	
Type	Implementation Requirement
HTML, Jinja2	Adapting the front-end templates to include the semester data placeholders in the table skeleton.
HTML, Jinja2, CSS	Implement a HTML selector for the relevant semesters.
JavaScript, jQuery	Writing custom tagging functions which filter modules according to semester data based on the selected semester. Give them an HTML class accordingly.
JavaScript, jQuery	Changing the table rendering functions to include the semester data for each rendered module / row.
CSS	Styling the tagged table rows to be hidden / fade in or out depending on their display status.

Table 2: Filtering by Semester Requirements

3.2 Filter Modules by Study Program

Filtering by Study Program [Back end]	
Type	Implementation Requirement
Database	Adding a new table to the database design to allow for saving of study program data.
Database	Adding a new table to the database to capture the many-to-many relationship between modules and study programs – and using foreign key constraints and cascade rules to ensure consistency.
Flask	Processing and then saving the study program data correctly from the UZH course catalogue, using the JSON interface.
Flask, Jinja2	Serving the stored study program data in JSON format, along with the module id data to match the many to many relationship based on study program selection.
Filtering by Study Program [Front end]	
Type	Implementation Requirement
JavaScript, jQuery	Writing custom tagging functions which filter modules according to study program to modules dictionary. Give them an according HTML class.
CSS	Styling the tagged table rows to be hidden / fade in or out depending on their display status.

Table 3: Filtering by Study Program Requirements

3.3 Improved Keyword Search

Improved Keyword Search [Back end]	
Type	Implementation Requirement
Flask	Rework UZH course catalogue API calls to use the JSON interface.
Flask	Perform the search, which is programmed for modules, also on courses. For each course, request the detail page and the modules which the course is part of. Retrieve those modules.
Flask	Find a way to reduce time complexity of the code – call the module-course search in parallel, delegate finding study programs to an own, designated function, only used when saving modules.
Improved Keyword Search [Front end]	
JavaScript, jQuery	Call the search function asynchronously on the front end using Ajax, to avoid long loading times.

Table 4: Improved Keyword Search Requirements

3.4 Matching the UZH Corporate Design

This feature is included across all front-end features and requirements.

Matching the UZH Corporate Design [Back end]	
Type	Implementation Requirement
CSS	Grab a copy of the minified UZH CSS rules, host on server to appropriate styling.
Matching the UZH Corporate Design [Front end]	
Type	Implementation Requirement
CSS	Use the UZH CSS classes in a nonintrusive way to keep the design consistent.

Table 5: Matching the UZH Corporate Design Requirements

3.5 Ease of Setup

To make sure the work done for this thesis is easily replicable and reusable, extensive documentation was written in the form of user guides and code comments. The documentation can be found in the appendix – including an export of the code comments.

Filtering by Study Program [Back end]	
Type	Implementation Requirement
Comments, Documentation (Python)	Make sure all necessary components, classes and functions are well documented, with type info where appropriate.
Database	Add an SQL file to the GitHub repository, which has the table creation statements inside.
Filtering by Study Program [Front end]	
Type	Implementation Requirement
Comments, Documentation (JavaScript)	Make sure all necessary components, classes and functions are well documented, with type info where appropriate.

Table 6: Ease of Setup Requirements

4 Implementation

This chapter will describe the concrete development process and give a technical overview of the implementation, detailing which tools were used in the making of the Green VVZ. For a concrete look at the end product, see the appendix – the user guides contain screenshots and explain the functionality. Alternatively, visit the UZH Sustainability page and try the live app.

4.1 Implementation Iterations

The background of the small-scaled FDD implementation process used in the Green VVZ's development is detailed in chapter 2.2: *Selection of Software Process Model*. It centred around three implementation cycles, each focusing on a specific set of features.

4.1.1 First Iteration: Revamping the Data Collection and Storage

In the first iteration, the basis for all changes was implemented – new database schemata and revamped API calls to the UZH course catalogue.

Requirements

With the high-level goals defined and the software process set, the next step was to break the goals down to initial requirements. The requirements as well as feedback and suggestions were compiled in close cooperation with the ISR team, functioning as the user role of the administrator.

The creation of a separate requirements document, explicitly defining all requirements to their full extent, was not deemed necessary, since there was an initial codebase with requirements to work for. Thus, only the requirements for the added functionality or necessary changes were documented. They were documented in informal fashion in chapter 3: *Requirements*. Requirements such as security, availability and the atomic requirements of button functionality were mostly maintained and are documented in the appendix of the *Nachhaltigkeits-Vorlesungsverzeichnis* thesis (Grässle, 2018).

All changes of the Green VVZ required more data than what the system was currently able to process; thus, learning how to get the new data and store it effectively was key before any other development could take place.

The initial requirements mainly dealt with back-end changes, and working out how to effectively gather, process, store and display information from the UZH course catalogue.

Understanding the UZH Course Catalogue API

Before implementing a new database design, it was important to understand how the data was supplied by UZH course catalogue OData API. In the *Nachhaltigkeits-Vorlesungsverzeichnis* initial form (Grässle, 2018), requests to the UZH back end were barely documented. Any information on what format the data was supplied in, and what end points the data was hosted at, had to be explored through trial and error since no documentation was available from the UZH course catalogue neither. Meetings with experts on the UZH course catalogue were not possible initially due to conflicting schedules. This took a while in the beginning, but once the basics were clear, development was able to advance more quickly.

Data Integrity

A big factor when working with data from an external API such as the UZH course catalogue is data integrity, especially when coupled with custom processing and display of the captured data. Since the database design of the initial code base was a lot simpler, with no restraints past the primary keys or relationships, it became a much bigger factor for this project. It was changed often during development with the gradual understanding of the UZH course catalogue API.

4.1.2 Second Iteration: Improving the Search, Decision on Fundamentals

In the second iteration, the keyword-based search was rewritten to not only look for matching modules in titles and descriptions directly, but instead also look for corresponding courses, and to retrieve all study programs featuring the matches.

Requirements

Now that the basis of the project was complete, the next set of requirements pertained around the automation aspect of the tool: the keyword search. Whilst it was previously constrained to finding matching modules, the results were essentially limited. There is no enforced convention to put the modules' topics strictly into its description or title, thus the keyword search would miss modules with information constrained to the contained courses. On top of that, it might feature modules with only a small matching course unit. The search was thus expanded to include courses as well.

It was debated whether to only show courses instead of modules, since that is essentially the units which students would actually partake in: the lectures, exercises, and so on. The project was briefly extended to accommodate for courses and study programs in the database, but it was not clear what unit would be best for the major stakeholder in this decision – the student.

Meeting with UZH Course Catalogue Expert

In order to decide on whether to present modules or courses in the public view, a meeting with Dr. Thomas Schwan, of the Business Applications of Zentrale Informatik UZH, was held. He is, at the time of writing, the supervisor of the UZH course catalogue and was able to give insight into the inner workings as well as upcoming changes of the UZH course catalogue.

He advised to keep modules as the main unit of display, reasoning that modules are what a student essentially gets credited for, and thus what is of most interest to a student. He further supplied a copy of the official documentation for the UZH course catalogue, which wasn't available beforehand. It is included in the code repository for future maintainers. All calls to the UZH course catalogue were reworked to use the JSON format, which is natively supported in the Flask library (Ronacher, 2019; Ronacher & Lord, 2010/2019) – the framework powering this project – and a lot more intuitive than the default XML documents. It was much less of an obstacle and the database design could be finalized: it was altered back containing modules only and study programs with foreign key dependencies to and from a 'module <-> study program' database table.

The search method still looked for courses, but only included the modules of which the courses are part of, as well as modules found directly. Furthermore, for every module, it retrieved the study programs that module is part of.

Speeding Up the Search: Parallelizing Web Requests

There was a problem with the keyword search now, because it performed very slow with an exponential runtime, or an approximate complexity class $T(n) = \text{EXPTIME}$, with $O(2^{n^k})$. To top that off, the algorithm included web requests to an external resource, the UZH course catalogue, in every iteration. Requests using keywords with a lot of results often took over 20 seconds to complete. This was due to limitations in the functions offered by the UZH course catalogue API: It is not possible to retrieve 'part of' information – e.g. what modules courses are part of, or what study programs modules are part of, for multiple entities at once. Thus, once a list of modules / courses was found, the detail page containing the 'part of' information had to be requested for each entity separately.

To fix this problem, multi-threading was used to parallelly call the 'part of' functions, allowing to launch all web requests simultaneously. True async code using asynchronous design patterns was not possible, as the basis of the project is the strictly synchronous framework Flask, and would have required a rework using an entirely different, asynchronous framework and a different server stack. Additionally, the added flexibility of asynchronous requests was not needed in the rest of the back-end functions.

To further reduce the time required, the module detail page was only requested when saving a module to the database, rather than when just displaying it as a search result. This reduced the code complexity by an exponential factor as well – distributing it to the actual saving action, which occurs a lot less frequently and not all at once.

Front End Development and Feedback

It was now possible to implement the front-end filtering functions for filtering by semester and study program, using the improved search and storage of the back end. The semester filter was implemented as a drop-down list and the study program filter using a search bar, which shows suggestions based on input. These elements are the same across the administrator and public view and were styled with the UZH corporate design in mind.

Since the amount of results increased in comparison to the previous search model, the user interface needed some remodelling to accommodate for the amount of information to be displayed. Whilst it would previously hide suggestions which are already white or blacklisted, it was decided in consultation with the advisor Jan Bieser, that this was not the clearest way

to deal with the already saved information from a user's perspective. Instead, the suggestions should always remain complete, with an indicator whether the module in question is already saved on the white or blacklist. This made it a lot easier to gauge how effective certain search terms are.

Furthermore, a problem of the initial *Nachhaltigkeits-Vorlesungsverzeichnis* (Grässle, 2018) was that it was unclear whether the app was loading data, or if it failed completely. All data for the page was loaded in one go, using synchronous jQuery (js.foundation, 2019a) and jQuery.Ajax() (js.foundation, 2019b) calls in the front end. This was inherently inefficient, as JavaScript and especially jQuery's Ajax – an abbreviation which stands for asynchronous JavaScript and XML – is built to make asynchronous web requests possible on the client side (Garrett, 2005). It thus made sense to display data on the front end as it becomes available, and showing loading indicators for parts which are still loading. This made the availability of the white and blacklist, as well as the search terms, practically instant, as these were hosted on a database on the same server. All data tables which can take longer due to being retrieved using HTTPS requests from external resources, feature a 'Loading...' placeholder if not yet available.

If the background updating fails, a corresponding message is also presented, to give proper feedback and disable silent background stalling.

4.1.3 Third Iteration: Finalizing the Design

In the span of the third and final iteration this applied bachelor's thesis, the focus lay on improving the looks and usability of the front end.

User Feedback

Before the final implementation cycle begun, feedback was gathered on the product as it was, in consultation with representatives for future users: the advisor, Jan Bieser, and the supervisor, Prof. Dr. Lorenz Hilty. Their feedback was valuable and pointed out several usability flaws. The first was the study program filter, which, to a student who is not versed in the tool, gave no indicator what kind of input is expected. Furthermore, a large amount of suggestions and saved modules can quickly become unwieldy to the administrators, and better tools to find modules and class them together were required. Finally, the expected input should not only be discernible, invalid input should be largely impossible, with immediate feedback to

the user on what is allowed and what is not. The implementations based on this feedback are detailed below.

Drop Down and Incremental Search

The study program filter worked on an incremental search only as it was first implemented. This was not very user friendly: No indicator to the correct format of the study program input was provided, and, if a specific study program did not exist, there was no information on that either.

The final study program filter shows a dropdown containing up to 15 results for the current semester: if there are more, only the first 10 are displayed, with an indicator to how many more there are. This is before any input. As incremental input is provided, the result list is narrowed down with feedback on the input, and which parts are matching. This way, the format is clear from the first 10 entries – and, whether or not study programs are found for a specific search input, feedback is given immediately.

Overview and Traceability in the Module Tables

To get a quick overview over each module table, a small badge with the current number of modules for the selected semester is shown next to each table header. The badge updates whenever the semester filter changes, or one of tables data is reloaded from the server – e.g. when a search is performed. This, in combination with each table being sortable on any column, makes the tables easy to navigate and modules easy to find and distinguish.

Furthermore, the way each stored module was found is saved: if it was manually added, it is marked as such. Otherwise, the search term with which it was found is saved with it. The search term is displayed for both suggestions and saved module in a designated column, and it is possible to sort on the search terms, to gauge their effectiveness, and how many of the found modules are then actually saved to the white or blacklist.

Finally, the suggestions table gets a new column: the save status. This column is used to identify and sort modules based on whether they are whitelisted, blacklisted, or not yet saved / new. This makes looking for new modules a breeze, despite the previously saved modules still appearing in the suggestions list.

Client-Side Input Validation

Since it was possible from the beginning to add modules manually, using their respective unique UZH course catalogue id, it was important to make that feature easy to use. The best way to add a module is to look up its id from its URL in the UZH course catalogue and copy paste it back. To avoid copy and pasting or typing invalid non-numerical ids, client-side input validation was implemented on top of the server-side validation taking place, to make it clear to the user what type of input is expected – non numerical input is impossible.

The same was implemented for text-based inputs, where leading and doubled whitespace gets trimmed down – and before input is submitted, trailing whitespace is removed as well. This is to ensure no empty, or single space search terms are saved for the keyword search, as it would result in a huge number of results, slowing the suggestions table loading of the program down significantly. The validation also helps for the study program filter, as there is no study program featuring multiple whitespace, or beginning with whitespace.

4.2 Technologies and Tools

In this subsection, the technologies used during development are introduced and their most important features in the Green VVZ explained. For a more atomic dissemination of some of the basic technologies used in the Green VVZ, consult the *Nachhaltigkeits-Vorlesungsverzeichnis* by Lukas Grässle (Grässle, 2018).

4.2.1 Traceability: Git and GitHub

To make it easier for people interested in using and developing the application, or for future maintainers to understand the code, not only documentation is important, but also traceability of the development. Git is an efficient, open-source version control system, based on decentralized source code repositories, created by Linus Torvalds, for the maintenance of the Linux kernel source code (Torvalds, 2005/2019). Every committed change is saved individually to the project history on a single character level for each changed file, with the functionality to view differences from before and after a change occurred. Each change is accompanied by a commit message, allowing the developer to explain what changes were implemented in

human readable form. This makes it possible to track the implementation of individual features from the very inception of a project, all the way to its release(s).

Furthermore, since changes are recorded precisely for each character in each line of each file, it allows for easy collaboration through its decentralized architecture: as long as changes do not concern the same line, they can be merged automatically by Git – and even if there are conflicting changes, the merging processes are well documented and explained in several guides, the most popular being the book *ProGit* (Chacon & Straub, 2013/2019).

An important tool to further add traceability is Git's branching: development can branch off to isolate working on certain changes and be merged back together once the changes are ready.

GitHub is a web service that serves primarily as a remote server for Git repositories. It further provides tools for organization of development, such as project development tools: issue lists, to-do lists, or visualizing the Git development branches, viewing the individual commits of a repository, and exploring the code at any given commit.

The Green VVZ's code is hosted on GitHub on a public repository at the time of submission.

4.2.2 Back End: Flask, MariaDB, UZH OData API and Jinja2 Templating

The Green VVZ follows a common web application architecture: there is a back end, containing the logic and storage capabilities – it is essentially a RESTful application interface, and a front end, which interfaces with the back end and is responsible for presentation, corresponding to the user interface.

The Green VVZ is written in Python, using the Flask microframework for web applications (Ronacher, 2019). It enables easy creation of HTML endpoints of the API, processing the requests and calling further Python functions to create HTML or JSON responses.

The Green VVZ's back end further uses the UZH course catalogue OData API, to request module data.

Finally, data input by the user's using the front end, or data from the UZH course catalogue needs to be stored. The best way to store data with the least amount of redundancy whilst maintaining consistency is a relational database management system (RDBMS). The Green VVZ uses the MariaDB fork of the MySQL RDBMS, as this is standard of the virtual private servers supplied by the UZH's Department of Informatics. The relational database allows

saving of relationships between objects and restricting the datatypes saved for each entity. This is especially useful when storing tightly linked entities such as modules and study programs – to make sure that the study program data is only saved if there is at least one module referencing it, which can be made automatic using foreign key restraints and cascade rules.

Finally, to optimize the front end, and to make it easy to call Python functions for the creation of views and templates, Flask comes with the Jinja2 templating language. Expressions in the HTML files can be replaced by return values of Python functions, which allows to introduce some necessary logic to the front end before serving

4.2.3 Front End: JavaScript, jQuery and Ajax

The front end is responsible for the layout and presentation of the data supplied by the back end and allows the user to interface with the server, to call functions and store data.

It is written in JavaScript with jQuery (js.foundation, 2019a) and makes heavy usage of jQuery.Ajax() (js.foundation, 2019b). This allows a gradual loading of the front-end page, important for the administrator view. Since the amount of data requested from the UZH course catalogue can be large, and the module-search's efficiency is limited due to technical limitations of the UZH course catalogue, the loading times for the suggestions can be a lot longer than any other data supplied by the Green VVZ's database directly. Consequently, in a strictly synchronous approach, the user would have to wait until all data is available until he is served a response. Using the asynchronous Ajax in-page web requests, a 'bare-bones' page template can be served, in which all data is requested and displayed asynchronously. This allows the administrators to view the page, the saved module data and the search terms, whilst the suggestions might still be loading the background for a few seconds. The time saved across each call adds to a lot smoother user experience and less frustration.

4.2.4 Content Management System: Embedding into Magnolia

The UZH web presence is managed using Magnolia 5, a popular content management system (CMS). It allows users without a technical background to quickly create and edit web content for the web presence of his enterprise, using a graphical user interface with predefined content options and styles.

The goal for the Green VVZ was to allow users to use the app from within the UZH Sustainability webpage. The most appropriate option for this would be to use an 'embedded page' in the Magnolia settings, or alternatively an iFrame object inside of a standard content page. Both options were not ideal for the Green VVZ, due to the fact that content of the front-end templates is loaded dynamically, and after the web view is generated, using Ajax calls. Additionally, the administrator view reloads the suggestions if a search term was added, further making the document embedded via an iFrame larger than its originally requested size.

A solution was required to enable an embedded iFrame to dynamically adapt its size depending on the embedded content. This is difficult, because the host webpage has no way to access the DOM or JavaScript of the embedded document if they are not from the same exact domain. The Green VVZ has its own domain name within the UZH domain architecture, which is not identical to the domain of the Sustainability pages. Thankfully, there exists a JavaScript library which enables automatic resizing of cross origin iFrames: iFrame Resizer (Bradshaw, 2013/2019). A script embedded on both the host and the embedded page allows for communication across them.

5 Documentation

The importance of documentation became evident in the beginning of this project and became even more apparent during the development. It was opted to create extensive inline code documentation, as well as user and developer guides to ensure smooth setup and use of the web application without additional training.

5.1 User Guides

User guides for both the public view and the administrator view were written in English and German. They explain in detail how to use the web application using the interface and functionality provided with screenshots and written explanations. They were written once the product was ready and correspond to the most up-to-date version of the Green VVZ at the time of submission.

Furthermore, a developer guide was written in English, to explain to future maintainers or developers how to set up an instance of the web application on a server, and how to set up a local development environment. It includes suggestions and pointers for making the server production ready. The developer guide is in the readme file of the GitHub Repository of the project (<https://github.com/bbodo/GreenVVZ/>), as well as the appendix of this thesis.

5.2 Code Documentation

To make the codebase readable and thus enable maintenance and extension of this project, both the back end and front end were documented and outfitted with type indicators for all functions. This also enables suggestions across most IDEs and advanced text editors, further making development easier.

The documentation was extracted and compiled to PDF using the documentation tool Docma, written by Onur Yildirim, for the JavaScript front end (Yildirim, 2019), and Python's Sphinx in the back end (Brandl, 2007).

6 Discussion

In the following sections the results will be discussed and a future outlook will be provided.

6.1 Reflection

The development of the Green VVZ was met with a few difficulties along the way, most notably the initial problem of setting the whole project up. It was difficult to grasp what specifics the server of the Green VVZ required in order to set up a testing environment, which functions the same way a productive environment would. Essentially, the whole project had to be deployed in full in order to be explored in the beginning, before a meaningful local development environment could be configured.

The second bigger issue was figuring out how to use the UZH OData API, as no documentation was available online and no meeting was possible initially to conflicting schedules. Trial and error allowed the project to progress until the meeting took place, which made further development run smooth.

Despite these two major issues, the implementation and writing phase was able to progress quickly and mostly trouble-free. I learnt a lot about web technologies, REST services and full stack development through the experience of designing and implementing a whole project.

6.2 Outlook

The Green VVZ is production ready and will be published as soon as possible. The adaptation and setup of new instances of the Green VVZ, for capturing modules based on different keywords is made easy using the provided documentation. This allows for the tool to be reused by all teams of UZH who could benefit from the automated search and storage options the Green VVZ provides.

A possible direction for maintenance and future extension would be to rewrite the JavaScript front end to rely on the new ECMAScript 6 functionality of modularization of JavaScript code. This would make it easier to isolate the code and get rid of the limited code duplication still present.

A different notion would be to rewrite the back end using a different web framework, or a different language altogether – with a focus on asynchronous code from the get-go – to speed up the key word search and improve the existing loading times.

7 Conclusion

This thesis details the process of implementation of the Green VVZ and explores research and theory on software projects. Rationale for the software development process is provided, in regard to the small size of the project and the development team – whilst keeping good practises and a future outlook in mind.

The requirements are provided to the extent relevant to a project of this scope, and the key highlights of the implementation are detailed through the three major development cycles. The technologies the Green VVZ relies on and was crafted with are introduced and explained.

The importance of good documentation became apparent during development, and additional extensive user guides are included to complement the inline code documentation. The code documentation was exported to human readable form, and all documentation efforts are placed in the appendix of this thesis: a total of 36 pages of documentation, including the user and developer guides.

Finally, a reflection on major issues during development and a future outlook is provided.

I want to thank the ISR group for affording me this opportunity and their assistance during the course of this thesis. I learned about a lot of new software engineering concepts, and cemented my skills in Python web development, becoming acquainted with the web framework Flask. I enjoyed it and I feel that the end product has value and can be put to use effectively, and hopefully it will allow students to choose courses relating to sustainability. This was made possible by the great assistance and motivation of the ISR group, helping me on every step along the way and allowing me to complete the thesis in due time. The open communication helped to overcome any hurdles along the way and the team was very accommodating to me.

8 Bibliography

- Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2017). Agile software development methods: Review and analysis. *ArXiv Preprint ArXiv:1709.08439*.
- Beck, K. (1999). Embracing change with extreme programming. *Computer*, (10), 70–77.
- Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., ... Jeffries, R. (2001). *Manifesto for agile software development*.
- Boehm, B. W. (1988). A spiral model of software development and enhancement. *Computer*, (5), 61–72.
- Boehm, B. W., & J Hansen, W. (2001). The Spiral Model as a Tool for Evolutionary Acquisition. *CrossTalk*, 14.
- Bradshaw, D. J. (2019). *Keep same and cross domain iFrames sized to their content with support for window/content resizing, in page links, nesting and multiple iFrames: Davidjbradshaw/iframe-resizer* [JavaScript]. Retrieved from <https://github.com/davidjbradshaw/iframe-resizer> (Original work published 2013)
- Brandl, G. (2007, 2009). Sphinx Documentation. Retrieved June 14, 2019, from <https://www.sphinx-doc.org/en/master/>
- Chacon, S., & Straub, B. (2019). *Pro Git 2nd Edition*. Retrieved from <https://github.com/progit/progit2> (Original work published 2013)
- Garrett, J. J. (2005). *Ajax: A new approach to web applications*.
- Grässle, L. (2018). *Nachhaltigkeits-Vorlesungsverzeichnis: Ein unterstützendes Tool zur Erkennung und Darstellung von Lehrveranstaltungen mit Nachhaltigkeitsbezug im Vorlesungsverzeichnis der Universität Zürich* (Bachelor's Thesis). University of Zurich, Zürich, Switzerland.
- js.foundation. (2019a, May 1). jQuery. Retrieved June 13, 2019, from <https://jquery.com/>
- js.foundation. (2019b, May 1). jQuery.ajax() | jQuery API Documentation. Retrieved June 13, 2019, from <https://api.jquery.com/jquery.ajax/>

- Palmer, S. R., & Felsing, M. (2001). *A practical guide to feature-driven development*. Pearson Education.
- Rising, L., & Janoff, N. S. (2000). The Scrum software development process for small teams. *IEEE Software*, 17(4), 26–32.
- Ronacher, A. (2019, May 2). Flask (A Python Microframework). Retrieved June 13, 2019, from <http://flask.pocoo.org/>
- Ronacher, A., & Lord, D. (2019). *The Python micro framework for building web applications.: Pallets/flask* [Python]. Retrieved from <https://github.com/pallets/flask> (Original work published 2010)
- Royce, W. W. (1987). Managing the development of large software systems: Concepts and techniques. *Proceedings of the 9th International Conference on Software Engineering*, 328–338. IEEE Computer Society Press.
- Schwaber, K. (1997). Scrum development process. In *Business object design and implementation* (pp. 117–134). Springer.
- Torvalds, L. (2019). *Git* [C]. Retrieved from <https://github.com/git/git> (Original work published 2005)
- University of Zurich. (2010a, May 31). Definition of the term “course.” Retrieved June 9, 2019, from <http://www.uniterm.uzh.ch/lists.php?termnr=1773>
- University of Zurich. (2010b, May 31). Definition of the term “module.” Retrieved June 9, 2019, from <http://www.uniterm.uzh.ch/lists.php?termnr=1523>
- University of Zurich. (2010c, May 31). Definition of the term “study program.” Retrieved June 9, 2019, from <http://www.uniterm.uzh.ch/lists.php?termnr=1548>
- University of Zurich. (2018, July 31). Grundlagen und Begriffe zur Studienstruktur. Retrieved June 8, 2019, from Studienangebotsentwicklung UZH website: <https://www.sae.uzh.ch/de/strukturen/grundlagen.html>

Yildirim, O. (2019). onury/docma: A powerful tool to easily generate beautiful HTML documentation from JavaScript (JSDoc), Markdown and HTML files. Retrieved June 14, 2019, from <https://github.com/onury/docma>

9 Tables and Figures

9.1 List of Tables

Table 1: Terms	6
Table 2: Filtering by Semester Requirements	15
Table 3: Filtering by Study Program Requirements	16
Table 4: Improved Keyword Search Requirements	16
Table 5: Matching the UZH Corporate Design Requirements	16
Table 6: Ease of Setup Requirements	17

9.2 List of Figures

Figure 1: The waterfall model (B. W. Boehm, 1988, p. 62)	10
Figure 2: The spiral model (B. W. Boehm, 1988, p. 64)	11
Figure 3: The design processes of FDD, (Abrahamsson et al., 2017, p. 50)	14

Appendix

10 Source Code

The source code is appended in electronical form and also available on:

<https://github.com/bbodo/Green VVZ.git>

11 Documents

11.1 Guides

The guides are appended on the following pages, in the following order:

1. Administrator view guide in English
2. Administrator view guide in German
3. Public view guide in English
4. Public view guide in German
5. PDF export of the developer guide in English.

They are also attached in electronical form, in this document and as separate documents. Furthermore, they can also be found in the Git repository (in the 'docs/guides' folder). It is available on:

<https://github.com/bbodo/GreenVVZ/tree/master/docs/guides>

The developer guide is also embedded into the Git readme.md file, available on:

<https://github.com/bbodo/GreenVVZ>

or

<https://github.com/bbodo/GreenVVZ/blob/master/README.md>



Green VVZ

Admin Interface – User Guide

Bodo Brägger

This is the admin interface user guide document in English. This documentation is also available in German. A separate document for the public interface will be available in both English and German as well.

This document will detail the major page elements and their usage – although it is encouraged to explore the application and learn by trial and error, consulting this document where functionality seems unintuitive.

Page Elements

1 Overview

Green VVZ Admin

Studienprogramm

Filter löschen

Frühlingssemester 2019 ▼

▼ Angezeigte Elemente (Whitelist) 2			
Name des Moduls	Suchbegriff	Semster	Verbergen
Modulnummer (8-Stellige Zahl in der URL zum Modul)		Modul hinzufügen	
→ UWW 174 Nachhaltigkeit und Gesellschaft	Nachhaltigkeit	FS 19	Verbergen
→ Vertiefungsmodul: Weltgesellschaft, Globalisierung	Nachhaltigkeit	FS 19	Verbergen

► Suchbegriffe

► Vorschläge basierend auf Suchbegriffen 10

► Verborgene Elemente (Blacklist) 7

The page consists of two main areas: The filter bar at the top, and the four tables containing the data. Only one table is selected at a time – folded out and highlighted in blue. A table is viewable by clicking the respective header, expanding its contents. Clicking the header again will collapse the table's contents back down.

Each major element is explained in the following sections.

2 Filter Bar

2.1 Semester Filter

Green VVZ Admin

Frühlingssemester 2019 ▼

Alle Semester
 Herbstsemester 2019
Frühlingssemester 2019
 Herbstsemester 2018
 Frühlingssemester 2018
 Herbstsemester 2017
 Frühlingssemester 2017

Name des Moduls	Suchbegriff	Semster	
Modulnummer (8-Stellige Zahl in der URL zum Modul)			
→ UWW 174 Nachhaltigkeit und Gesellschaft	Nachhaltigkeit	FS 19	Verbergen
→ Vertiefungsmodul: Weltgesellschaft, Globalisierung	Nachhaltigkeit	FS 19	Verbergen

Suchbegriffe

Vorschläge basierend auf Suchbegriffen 10

Verborgene Elemente (Blacklist) 7

The semester filter on the top right hand of the page allows selection of a specific semester, or all saved semesters. This affects all three module tables: The Whitelist, Suggestions, and Blacklist.

2.2 Study Program Filter

Green VVZ Admin

Herbstsemester 2018 ▼

	Suchbegriff	Semster	Verbergen
Modul hinzufügen			
Accounting and Finance: Nebenfach 30			
Allgemeine Ausbildung (1UF): Unterrichtsfach			
Allgemeine Ausbildung (2 UF): 1. Unterrichtsfach			
Allgemeine Ausbildung ZusUF: zus. Unterrichtsfach			
Allgemeine Wirtschaftswissenschaften: Nebenfach 30			
Banking: Nebenfach 30			
Banking and Finance: Hauptfach 150			
Banking and Finance: Hauptfach 90			
Banking and Finance: Nebenfach 30			
Banking and Finance: Nebenfach 60			
... 15 weitere Studienprogramme für Module im gewählten Semester mit Filter: B			
→ Seminar on Responsible Leadership (S)	Sustainability	HS 18	Verbergen
→ Sustainability and the Finance Sector (S)	Sustainability	HS 18	Verbergen
→ Umwelt und Nachhaltigkeit im naturwissenschaftlichen Unterricht	Nachhaltigkeit	HS 18	Verbergen
→ UWW 152 Ecohealth	Sustainability	HS 18	Verbergen

Suchbegriffe

Vorschläge basierend auf Suchbegriffen 14

Verborgene Elemente (Blacklist) 5

The study program filter is a dropdown complemented by an incremental search. This filter works **on top of the semester filter**, and **only affects the whitelist**. Clicking into the input field shows the first 10-15 study programs for modules in the current semester, and how many more there are. Typing will then filter according to the input – again limiting the total results. Using the cursor and click or arrow keys to select one of the study programs will show all modules of the whitelist in the specified semester and study program.

3 Data Tables

The module tables Whitelist, Suggestions and Blacklist each have the same basic set of information:

- The name of the module, which is also a link to the entry in the course catalogue.
- The search term with which the module was found.
- The semester in which the specific module is offered.
- A column containing buttons to save the module to the white- or blacklist.

The data is loaded asynchronously onto the page – whatever is available first will be displayed first. Tables which are currently loading feature a grey overlay, which disappears when loading is complete.

▼ Vorschläge basierend auf Suchbegriffen 10				
Module data is loading...	Suchbegriff	Semester	Anzeigen/Verbergen	Status

For each user visiting the admin interface, the system updates the saved modules once per day. This is signified using a blue header at the top of the tables. Updating can be forced on the same browser by accessing the page using a private window or deleting cookies.

Each table is sortable on each header featuring a ▼ or ▲ symbol.

3.1 Whitelist

Green VVZ Admin

Studienprogramm

Filter löschen

Herbstsemester 2019

Angezeigte Elemente (Whitelist) 10

Name des Moduls	Suchbegriff	Semster	Verbergen
Modulnummer (8-Stellige Zahl in der URL zum Modul)		Modul hinzufügen	
→ CHE 328 Green Chemistry	Sustainability	HS 19	Verbergen
→ Digitalization and Sustainable Development (V)	Sustainability	HS 19	Verbergen
→ Entwicklung & Nachhaltigkeit (Seminar)	Nachhaltigkeit	HS 19	Verbergen
→ Environmental and financial sustainability (L)	Sustainability	HS 19	Verbergen
→ Nachhaltigkeitsberichterstattung (S) (Corporate sustainability disclosure)	Sustainability	HS 19	Verbergen
→ Nachhaltigkeitskommunikation und Konsumentenverhalten	Nachhaltigkeit	HS 19	Verbergen
→ Seminar on Responsible Leadership (S)	Sustainability	HS 19	Verbergen
→ Sustainability and the Finance Sector (S)	Sustainability	HS 19	Verbergen
→ Topics in Open Economy Macroeconomics (L)	Sustainability	HS 19	Verbergen
→ UWW 152 Ecohealth	Sustainability	HS 19	Verbergen

Suchbegriffe

Vorschläge basierend auf Suchbegriffen 14

Verborgene Elemente (Blacklist) 4

The whitelist contains saved modules which **will be displayed to the students in the public interface**. The first row allows for manual saving of a module to a specific semester, using the unique **module id**, found at the **end** of its URL in the course catalogue – and having the corresponding semester selected in the filter: studentservices.uzh.ch/uzh/anonym/vvz/index.html#/details/2018/4/SM/50728226

Modules in the whitelist can be moved to the blacklist using the ‘Verbergen’ button. The whitelist is the only table to which the study program filter applies.

3.2 Search Terms

Green VVZ Admin

Studienprogramm

Filter löschen

Frühlingssemester 2019 ▼

▶ Angezeigte Elemente (Whitelist) 2

▼ Suchbegriffe

Begriff	Entfernen
Suchbegriff für Titel, Beschrieb oder Kürzel	Suchbegriff speichern
Nachhaltigkeit	Entfernen
Sustainability	Entfernen

▶ Vorschläge basierend auf Suchbegriffen 10

▶ Verborgene Elemente (Blacklist) 7

The search terms are the basis for the suggested modules shown in the next table. You can add and delete them using the respective buttons. After any change, the suggestions are reloaded.

3.3 Suggestions

Green VVZ Admin

Studienprogramm

Filter löschen

Frühlingssemester 2019 ▼

▶ Angezeigte Elemente (Whitelist) 2					
▶ Suchbegriffe					
▼ Vorschläge basierend auf Suchbegriffen 10					
Name des Moduls ▲	Suchbegriff ◆	Semester ◆	Anzeigen/Verbergen	Status ◆	
→ Asset Pricing (V+Ü)	Nachhaltigkeit	FS 19	Anzeigen Verbergen	Verborgen	
→ Mediendidaktik für den Fremdsprachenunterricht	Nachhaltigkeit	FS 19	Anzeigen Verbergen	Neu	
→ Naturwissenschaften und Nachhaltigkeit vermitteln (Teaching Science and Sustainability); Naturwissenschaftsdidaktische Grundlagen 1	Sustainability	FS 19	Anzeigen Verbergen	Verborgen	
→ Schwerpunkt 3 Forschungsseminar: Nutzung, Rezeption, Wirkung (zweisemestrig)	Nachhaltigkeit	FS 19	Anzeigen Verbergen	Verborgen	
→ ST 6E: Anwendung von Wissen	Nachhaltigkeit	FS 19	Anzeigen Verbergen	Verborgen	
→ Sustainable Investing (L)	Sustainability	FS 19	Anzeigen Verbergen	Verborgen	
→ UWW 174 Nachhaltigkeit und Gesellschaft	Nachhaltigkeit	FS 19	Anzeigen Verbergen	Angezeigt	
→ Vertiefungsmodul: Weltgesellschaft, Globalisierung	Nachhaltigkeit	FS 19	Anzeigen Verbergen	Angezeigt	
→ Voice training and presentation skills in the sciences and medicine	Sustainability	FS 19	Anzeigen Verbergen	Verborgen	
→ Wahlmodul: Mensch - Tier - Umwelt	Nachhaltigkeit	FS 19	Anzeigen Verbergen	Verborgen	
▶ Verborgene Elemente (Blacklist) 7					

Modules in the suggestions table can be saved to white- or blacklist using the respective button. The buttons are greyed out if the module is already saved in one of the lists. This is also reflected in the column 'Status', with which the table is also sortable. This gives an overview of elements already saved in either list, or new suggestions which are not yet processed.

3.4 Blacklist

Green VVZ Admin

Studienprogramm

Filter löschen

Frühlingssemester 2019 ▼

▶ Angezeigte Elemente (Whitelist) 2

▶ Suchbegriffe

▶ Vorschläge basierend auf Suchbegriffen 10

▼ Verborgene Elemente (Blacklist) 7

Name des Moduls	Suchbegriff	Semster	Anzeigen
→ Asset Pricing (V+Ü)	Nachhaltigkeit	FS 19	Anzeigen Löschen
→ Naturwissenschaften und Nachhaltigkeit vermitteln (Teaching Science and Sustainability); Naturwissenschaftsdidaktische Grundlagen 1	Sustainability	FS 19	Anzeigen Löschen
→ Schwerpunkt 3 Forschungsseminar: Nutzung, Rezeption, Wirkung (zweisemestrig)	Nachhaltigkeit	FS 19	Anzeigen Löschen
→ ST 6E: Anwendung von Wissen	Nachhaltigkeit	FS 19	Anzeigen Löschen
→ Sustainable Investing (L)	Sustainability	FS 19	Anzeigen Löschen
→ Voice training and presentation skills in the sciences and medicine	Sustainability	FS 19	Anzeigen Löschen
→ Wahlmodul: Mensch - Tier - Umwelt	Nachhaltigkeit	FS 19	Anzeigen Löschen

The blacklist features a button ‘Anzeigen’ per module, which moves the module to the whitelist, and a button ‘Löschen’, which deletes the module from the database altogether – it can still appear in the suggestions after deletion.



Green VVZ

Administratoren Ansicht – Benutzeranleitung

Bodo Brägger

Dies ist die deutsche Benutzeranleitung für die Administratoren Ansicht. Diese Benutzeranleitung ist auch verfügbar auf Englisch. Eine Benutzeranleitung für die öffentliche Ansicht wird ebenfalls auf Englisch und Deutsch zur Verfügung gestellt.

Dieses Dokument wird die wichtigsten Elemente der Administratoren Ansicht erklären – es wird aber empfohlen, einfach auszuprobieren und diese Anleitung zu Rate zu ziehen, wenn etwas unklar ist.

Interaktive Elemente

1 Übersicht

Green VVZ Admin

Studienprogramm

Filter löschen

Frühlingssemester 2019 ▼

▼ Angezeigte Elemente (Whitelist) 2

Name des Moduls	Suchbegriff	Semster	Verbergen
Modulnummer (8-Stellige Zahl in der URL zum Modul)			
Modul hinzufügen			
→ UWW 174 Nachhaltigkeit und Gesellschaft	Nachhaltigkeit	FS 19	Verbergen
→ Vertiefungsmodul: Weltgesellschaft, Globalisierung	Nachhaltigkeit	FS 19	Verbergen

► Suchbegriffe

► Vorschläge basierend auf Suchbegriffen 10

► Verborgene Elemente (Blacklist) 7

Die Seite besteht aus zwei Hauptbereichen: Die Filterleiste am oberen Rand und die vier Tabellen mit den Daten.

Nur eine der Tabellen kann ausgewählt werden, was durch den blauen Hintergrund und die ausgeklappten Daten visualisiert wird. Durch das Klicken auf einen der Titel wird die jeweilige Tabelle aufgeklappt, ein erneuter Klick schliesst die Tabelle wieder.

Die beiden Bereiche werden in den folgenden Kapiteln erklärt.

2 Filterleiste

2.1 Semesterfilter

Green VVZ Admin

Frühlingssemester 2019 ▾
Alle Semester
Herbstsemester 2019
Frühlingssemester 2019
Herbstsemester 2018
Frühlingssemester 2018
Herbstsemester 2017
Frühlingssemester 2017

Name des Moduls	Suchbegriff	Semster	
Modulnummer (8-Stellige Zahl in der URL zum Modul)			
→ UWW 174 Nachhaltigkeit und Gesellschaft	Nachhaltigkeit	FS 19	<input type="button" value="Verbergen"/>
→ Vertiefungsmodul: Weltgesellschaft, Globalisierung	Nachhaltigkeit	FS 19	<input type="button" value="Verbergen"/>

Suchbegriffe

Vorschläge basierend auf Suchbegriffen 10

Verborgene Elemente (Blacklist) 7

Der Semesterfilter ist eine Dropdownliste am oberen rechten Rand, die das Filtern nach einem der Semester oder das Anzeigen aller Semester ermöglicht. Dieser Filter betrifft alle drei Modultabellen: Die Whitelist, Vorschläge und die Blacklist.

2.2 Studienprogrammfilter

Green VVZ Admin

Herbstsemester 2018 ▾

	Suchbegriff	Semster	Verbergen
Accounting and Finance: Nebenfach 30			
Allgemeine Ausbildung (1UF): Unterrichtsfach			
Allgemeine Ausbildung (2 UF): 1. Unterrichtsfach			
Allgemeine Ausbildung ZusUF: zus. Unterrichtsfach			
Allgemeine Wirtschaftswissenschaften: Nebenfach 30			
Banking: Nebenfach 30	uren der Nachhaltigkeit (Beginn FS	Nachhaltigkeit HS 18	<input type="button" value="Verbergen"/>
Banking and Finance: Hauptfach 150	Sustainability	HS 18	<input type="button" value="Verbergen"/>
Banking and Finance: Hauptfach 90	Sustainability	HS 18	<input type="button" value="Verbergen"/>
Banking and Finance: Nebenfach 30	Sustainability	HS 18	<input type="button" value="Verbergen"/>
Banking and Finance: Nebenfach 60	Sustainability	HS 18	<input type="button" value="Verbergen"/>
... 15 weitere Studienprogramme für Module im gewählten Semester mit Filter: B	Sustainability	HS 18	<input type="button" value="Verbergen"/>
→ Seminar on Responsible Leadership (S)	Sustainability	HS 18	<input type="button" value="Verbergen"/>
→ Sustainability and the Finance Sector (S)	Sustainability	HS 18	<input type="button" value="Verbergen"/>

Der Studienprogrammfilter ist ein Dropdown in Kombination mit inkrementeller Suche am oberen linken Rand. Es wird eine **engere Auswahl aufgrund der Module im gewählten Semester** gefiltert, wobei dies **nur Module in der Whitelist betrifft**.

Wenn in das Eingabefeld geklickt wird, werden die ersten 10-15 Studienprogramme für Module im gewählten Semester gezeigt, mit einem Indikator für die weitere Anzahl der Studienprogramme die den Modulen des gewählten Semesters entsprechen. Bei Eingabe werden Studienprogramme gezeigt, die der Eingabe entsprechen. Per Mausklick oder den Pfeiltasten können Studienprogramme ausgewählt werden. Es werden nur Module in der Whitelist gezeigt, die **Teil des gewählten Studienprogramms und Semesters** sind.

3 Datentabellen

Die Modultabellen Whitelist, Vorschläge und Blacklist haben die gleiche Grundstruktur. Sie enthalten:

- Die Namen der Module, die gleich noch einen Link ins Vorlesungsverzeichnis der UZH darstellen.
- Der Suchbegriff, womit das Modul gefunden wurde.
- Das Semester des Moduls.
- Buttons, um das Modul in der White- oder Blacklist zu speichern oder zu löschen.

Die Daten werden asynchron (gleichzeitig) in die Seite geladen – was zuerst geladen wird, wird zuerst angezeigt. Was noch am Laden ist hat ein graues Transparent darüber, welches bei Verfügbarkeit der Daten verschwindet.

▼ Vorschläge basierend auf Suchbegriffen (0)				
Module data is loading...	Suchbegriff	Semester	Anzeigen/Verbergen	Status

Für jeden Besucher werden die Module einmal pro Tag mit dem Vorlesungsverzeichnis synchronisiert, was durch eine blaue Hinweisbox gezeigt wird. Diese Synchronisation kann erzwungen werden, indem man die Seite in einem privaten Fenster öffnet oder die Cookies löscht.

Die Tabellen sind sortierbar nach den Kopfzeilen Einträgen, die ein ▼ oder ein ▲ Symbol enthalten.

3.1 Whitelist

Green VVZ Admin

Studienprogramm

Filter löschen

Herbstsemester 2019 ▼

▼ Angezeigte Elemente (Whitelist) 10

Name des Moduls	Suchbegriff	Semster	Verbergen
Modulnummer (8-Stellige Zahl in der URL zum Modul)		Modul hinzufügen	
→ CHE 328 Green Chemistry	Sustainability	HS 19	Verbergen
→ Digitalization and Sustainable Development (V)	Sustainability	HS 19	Verbergen
→ Entwicklung & Nachhaltigkeit (Seminar)	Nachhaltigkeit	HS 19	Verbergen
→ Environmental and financial sustainability (L)	Sustainability	HS 19	Verbergen
→ Nachhaltigkeitsberichterstattung (S) (Corporate sustainability disclosure)	Sustainability	HS 19	Verbergen
→ Nachhaltigkeitskommunikation und Konsumentenverhalten	Nachhaltigkeit	HS 19	Verbergen
→ Seminar on Responsible Leadership (S)	Sustainability	HS 19	Verbergen
→ Sustainability and the Finance Sector (S)	Sustainability	HS 19	Verbergen
→ Topics in Open Economy Macroeconomics (L)	Sustainability	HS 19	Verbergen
→ UWW 152 Ecohealth	Sustainability	HS 19	Verbergen

► Suchbegriffe

► Vorschläge basierend auf Suchbegriffen 14

► Verborgene Elemente (Blacklist) 4

Die Whitelist beinhaltet die Module, die **den Studenten in der öffentlichen Ansicht angezeigt** werden. Die erste Reihe ist ein Eingabefeld, in welchem Module aus dem Vorlesungsverzeichnis manuell hinzugefügt werden können. Es wird die Modul-ID und das entsprechend ausgewählte Semester benötigt. Die **Modul-ID** findet sich bei URLs im Vorlesungsverzeichnis **ganz am Ende**.

studentservices.uzh.ch/uzh/anonym/vvz/index.html#/details/2018/4/SM/50728226

Module der Whitelist können mittels des 'Verbergen' Buttons in die Blacklist verschoben werden.

3.2 Suchbegriffe

Green VVZ Admin

Studienprogramm

Filter löschen

Frühlingssemester 2019 ▼

Angezeigte Elemente (Whitelist) 2

Suchbegriffe

Begriff	Entfernen
Suchbegriff für Titel, Beschrieb oder Kürzel	Suchbegriff speichern
Nachhaltigkeit	Entfernen
Sustainability	Entfernen

Vorschläge basierend auf Suchbegriffen 10

Die Suchbegriffe sind die Grundlage der Suche, welche die Vorschläge generiert. Sie können mit den entsprechenden Buttons hinzugefügt oder entfernt werden.

3.3 Vorschläge

Green VVZ Admin

Studienprogramm

Filter löschen

Frühlingssemester 2019 ▼

Angezeigte Elemente (Whitelist) 2

Suchbegriffe

Vorschläge basierend auf Suchbegriffen 10

Name des Moduls ▲	Suchbegriff ◆	Semester ◆	Anzeigen/Verbergen	Status ◆
→ Asset Pricing (V+Ü)	Nachhaltigkeit	FS 19	Anzeigen Verbergen	Verborgen
→ Mediendidaktik für den Fremdsprachenunterricht	Nachhaltigkeit	FS 19	Anzeigen Verbergen	Neu
→ Naturwissenschaften und Nachhaltigkeit vermitteln (Teaching Science and Sustainability); Naturwissenschaftsdidaktische Grundlagen 1	Sustainability	FS 19	Anzeigen Verbergen	Verborgen
→ Schwerpunkt 3 Forschungsseminar: Nutzung, Rezeption, Wirkung (zweisemestrig)	Nachhaltigkeit	FS 19	Anzeigen Verbergen	Verborgen
→ ST 6E: Anwendung von Wissen	Nachhaltigkeit	FS 19	Anzeigen Verbergen	Verborgen
→ Sustainable Investing (L)	Sustainability	FS 19	Anzeigen Verbergen	Verborgen
→ UWW 174 Nachhaltigkeit und Gesellschaft	Nachhaltigkeit	FS 19	Anzeigen Verbergen	Angezeigt
→ Vertiefungsmodul: Weltgesellschaft, Globalisierung	Nachhaltigkeit	FS 19	Anzeigen Verbergen	Angezeigt
→ Voice training and presentation skills in the sciences and medicine	Sustainability	FS 19	Anzeigen Verbergen	Verborgen
→ Wahlmodul: Mensch - Tier - Umwelt	Nachhaltigkeit	FS 19	Anzeigen Verbergen	Verborgen

Verborgene Elemente (Blacklist) 7

Die vorgeschlagenen Module können in die White- oder Blacklist gespeichert werden mittels der entsprechenden Buttons. Falls ein Modul schon gespeichert ist, ist der entsprechende Button deaktiviert. Dies ist auch ersichtlich im 'Status', wobei ein Modul als 'Neu' angezeigt wird, wenn es in keiner der beiden Listen gespeichert ist.

3.4 Blacklist

Green VVZ Admin

Studienprogramm

Filter löschen

Frühlingssemester 2019 ▼

▶ Angezeigte Elemente (Whitelist) 2

▶ Suchbegriffe

▶ Vorschläge basierend auf Suchbegriffen 10

▼ Verborgene Elemente (Blacklist) 7

Name des Moduls	Suchbegriff	Semster	Anzeigen
→ Asset Pricing (V+Ü)	Nachhaltigkeit	FS 19	Anzeigen Löschen
→ Naturwissenschaften und Nachhaltigkeit vermitteln (Teaching Science and Sustainability); Naturwissenschaftsdidaktische Grundlagen 1	Sustainability	FS 19	Anzeigen Löschen
→ Schwerpunkt 3 Forschungsseminar: Nutzung, Rezeption, Wirkung (zweisemestrig)	Nachhaltigkeit	FS 19	Anzeigen Löschen
→ ST 6E: Anwendung von Wissen	Nachhaltigkeit	FS 19	Anzeigen Löschen
→ Sustainable Investing (L)	Sustainability	FS 19	Anzeigen Löschen
→ Voice training and presentation skills in the sciences and medicine	Sustainability	FS 19	Anzeigen Löschen
→ Wahlmodul: Mensch - Tier - Umwelt	Nachhaltigkeit	FS 19	Anzeigen Löschen

Die Blacklist hat einen 'Anzeigen' Button, womit ein Modul in die Whitelist verschoben wird, und einen 'Löschen' Button, womit ein Modul ganz aus der Datenbank gelöscht wird – es kann aber noch immer als Vorschlag mit Status 'Neu' erscheinen.



Green VVZ

Public Interface – User Guide

Bodo Brägger

This is the public interface user guide document in English. This documentation is also available in German. A separate document for the admin interface will be available in both English and German as well.

This document will detail the major page elements and their usage – although it is encouraged to explore the application and learn by trial and error, consulting this document where functionality seems unintuitive.

Page Elements

1 Overview

Green VVZ Public

Studienprogramm	Filter löschen	Herbstsemester 2019 ▼
Module der UZH mit Nachhaltigkeitsbezug		
Name des Moduls	Semester (FS = Frühjahressemester, HS = Herbstsemester)	
→ CHE 328 Green Chemistry	HS	19
→ Digitalization and Sustainable Development (V)	HS	19
→ Entwicklung & Nachhaltigkeit (Seminar)	HS	19
→ Environmental and financial sustainability (L)	HS	19
→ Nachhaltigkeitsberichterstattung (S) (Corporate sustainability disclosure)	HS	19
→ Nachhaltigkeitskommunikation und Konsumentenverhalten	HS	19
→ Seminar on Responsible Leadership (S)	HS	19
→ Sustainability and the Finance Sector (S)	HS	19
→ Topics in Open Economy Macroeconomics (L)	HS	19
→ UWW 152 Ecohealth	HS	19

The page consists of two main areas: the filter bar at the top and the table containing the module data.

Each major element is explained in the following sections.

2 Filter Bar

2.1 Semester Filter

Green VVZ Public

Studienprogramm

Filter löschen

Frühlingssemester 2019 ▼

Module der UZH mit Nachhaltigkeitsbezug	
Name des Moduls	Semester (FS = Frühjahressemester, HS = Herbstsemester)
→ UWW 174 Nachhaltigkeit und Gesellschaft	FS 19
→ Vertiefungsmodul: Weltgesellschaft, Globalisierung	FS 19

The semester filter on the top right hand of the page allows the selection of a specific semester or all saved semesters.

2.2 Study Program Filter

Green VVZ Public

b

Filter löschen

Herbstsemester 2019 ▼

Accounting and Finance: Nebenfach 30	
Allgemeine Wirtschaftswissenschaften: Nebenfach 30	
Banking: Nebenfach 30	
Banking and Finance: Hauptfach 150	
Banking and Finance: Hauptfach 90	
Banking and Finance: Nebenfach 30	
Banking and Finance: Nebenfach 60	
Banking and Finance (Fast Track): Hauptfach	
Betriebswirtschaftslehre: Hauptfach 150	
Betriebswirtschaftslehre: Hauptfach 90	
... 16 weitere Studienprogramme für Module im gewählten Semester mit Filter: b	
→ Nachhaltigkeitskommunikation und Nachhaltigkeitsmanagement	
→ Seminar on Responsible Leadership (S)	HS 19
→ Sustainability and the Finance Sector (S)	HS 19
→ Topics in Open Economy Macroeconomics (L)	HS 19
→ UWW 152 Ecohealth	HS 19

The study program filter is a dropdown complemented by an incremental search. This filter works **on top of the semester filter**. Clicking into the input field shows the first 10-15 study programs for modules in the current semester, and how many more there are. Typing will then filter according to the input – again limiting the total results. Using the cursor and click or arrow keys to select one of the study programs will show all modules in the specified semester and study program.

3 Modules

The table containing the module data shows the name of each module, which also functions as a link to the course catalogue of UZH, as well as the semester in which a module is offered.



Green VVZ

Öffentliche Ansicht – Benutzeranleitung

Bodo Brägger

Dies ist die deutsche Benutzeranleitung für die öffentliche Ansicht. Diese Benutzeranleitung ist auch verfügbar auf Englisch. Eine Benutzeranleitung für die Administratorenansicht wird ebenfalls auf Englisch und Deutsch zur Verfügung gestellt.

Dieses Dokument wird die wichtigsten Elemente der öffentlichen Green VVZ Ansicht erklären – es wird aber empfohlen einfach auszuprobieren, und diese Anleitung zu Rate zu ziehen, wenn etwas unklar ist.

Interaktive Elemente

1 Übersicht

Green VVZ Public

Studienprogramm

Filter löschen

Herbstsemester 2019 ▼

Module der UZH mit Nachhaltigkeitsbezug	
Name des Moduls	Semester (FS = Frühjahressemester, HS = Herbstsemester)
→ CHE 328 Green Chemistry	HS 19
→ Digitalization and Sustainable Development (V)	HS 19
→ Entwicklung & Nachhaltigkeit (Seminar)	HS 19
→ Environmental and financial sustainability (L)	HS 19
→ Nachhaltigkeitsberichterstattung (S) (Corporate sustainability disclosure)	HS 19
→ Nachhaltigkeitskommunikation und Konsumentenverhalten	HS 19
→ Seminar on Responsible Leadership (S)	HS 19
→ Sustainability and the Finance Sector (S)	HS 19
→ Topics in Open Economy Macroeconomics (L)	HS 19
→ UWW 152 Ecohealth	HS 19

Die Seite besteht aus zwei Hauptbereichen: Der Filterleiste am oberen Rand, und der Tabelle mit den Moduldaten.

Die beiden Bereiche werden in den folgenden Kapiteln erklärt.

2 Filterleiste

2.1 Semesterfilter

Green VVZ Public

Studienprogramm

Filter löschen

Frühlingssemester 2019 ▼

Module der UZH mit Nachhaltigkeitsbezug	
Name des Moduls	Semester (FS = Frühjahressemester, HS = Herbstsemester)
→ UWW 174 Nachhaltigkeit und Gesellschaft	FS 19
→ Vertiefungsmodul: Weltgesellschaft, Globalisierung	FS 19

Der Semesterfilter ist eine Dropdownliste am oberen rechten Rand, die das Filtern nach einem oder das Anzeigen aller Semester ermöglicht.

2.2 Studienprogrammfilter

Green VVZ Public

b

Filter löschen

Herbstsemester 2019 ▼

Accounting and Finance: Nebenfach 30	
Allgemeine Wirtschaftswissenschaften: Nebenfach 30	
Banking: Nebenfach 30	
Banking and Finance: Hauptfach 150	
Banking and Finance: Hauptfach 90	
Banking and Finance: Nebenfach 30	
Banking and Finance: Nebenfach 60	
Banking and Finance (Fast Track): Hauptfach	
Betriebswirtschaftslehre: Hauptfach 150	
Betriebswirtschaftslehre: Hauptfach 90	
... 16 weitere Studienprogramme für Module im gewählten Semester mit Filter: b	
→ Seminar on Responsible Leadership (S)	
→ Sustainability and the Finance Sector (S)	HS 19
→ Topics in Open Economy Macroeconomics (L)	HS 19
→ UWW 152 Ecohealth	HS 19

Der Studienprogrammfilter ist ein Dropdown in Kombination mit inkrementeller Suche am oberen linken Rand. Es wird eine **engere Auswahl aufgrund der Module im gewählten Semester** gefiltert. Wenn in das Eingabefeld geklickt wird, werden die ersten 10-15 Studienprogramme für Module im gewählten Semester gezeigt, mit einem Indikator für die weitere Anzahl der Studienprogramme die den Modulen des gewählten Semesters entsprechen. Bei Eingabe werden Studienprogramme gezeigt, die der Eingabe entsprechen. Per Mausklick oder den Pfeiltasten können Studienprogramme ausgewählt werden.

3 Modultabelle


Die Modultabelle enthält die Namen der Module und das Semester in welchem sie angeboten werden. Die Namen sind auch Links ins Vorlesungsverzeichnis der UZH.



Developer's Guide

Branch: master ▾

Find file Copy path



GreenVVZ / README.md

 **bbodo** Update README.md
02ce985 just now

2 contributors  

Raw Blame History

110 lines (92 sloc) | 7.42 KB

Green VVZ

Overview

GreenVVZ: A web application to facilitate storage and display of modules related to sustainability topics, based on the course catalogue of the University of Zurich.

Built for the Informatics and Sustainability Research group at the Department of Informatics of the University of Zurich, under supervision of Prof. Dr. Lorenz Hilty.

This project consists of two main parts: a flask back end, interfacing with the UZH course catalogue, as well as a flask / Jinja2 / jQuery front end, interfacing with the flask back end.

A foreword for the following guides: Trial and error, as long as you are not running a productive server, are a great tool to learn how software works. This guide might not be complete, and the technology you run it on might not be the exact same, but it should help you figure out the necessary steps to get it running. **Should anything need changes, please create a pull request!**

Developer's Guide for hosting the project

(This is needed to test any database functionality. Best bring this guide to your UZH technician, he will be able to guide you where this guide might fail.)

To set up a server hosting this tool, which will be required to implement and test significant changes, follow these steps:

1. Request a Virtual Private Server (VPS) at the Department of Informatics (or set up your own). Root privileges are not required, with exception of privileges to (re)start certain services. More to that later. Make sure the VPS comes with the following software installed:
 - A Linux distribution. *UZH uses Debian.*
 - A user / production folder with read/write privileges.
 - read / write privileges to the config folders of systemd/systemctl.
 - open ports, at least port 80, 8080, 443 and 8443.
 - Git.
 - A MySQL variant. *UZH uses MariaDB.* Also, a MariaDB account which can create a database, tables, and read/write to it. Standard procedure is one DB user / application. *UZH supplied the accounts for me.*
 - Python ≥ 3.5 .
 - pip, venv.
 - An HTTP server. *UZH uses Apache.* Make sure to request privileges to run `sudo systemctl restart apache2`.
 - A working SSL certificate.
2. In your user directory, create a virtualenv (*UZH supplied one for me*):
 - To create a local environment to host the needed python libraries, if necessary and applicable, either `python3 -m pip install virtualenv`, Or `apt-get install python3-venv`.
 - Create a virtual python environment on your VPS with either `python3 -m virtualenv venv_[choose a name]` Or `python3 -m venv venv_[choose a name]` in your user directory. For this example, let's say you chose `greenvvz` as the name.
 - This will create a folder `venv_greenvvz`
 - use `source venv_greenvvz/bin/activate` to activate the virtual env.
3. Set up this project to be served:
 - First of all, get a copy of this project on the server: `git clone https://github.com/bbodo/greenvvz.git`, to clone this repository into the folder `greenvvz`. Then `cd greenvvz` to navigate to the project directory.
 - With the virtualenv activated, use `pip install -r requirements.txt` to install all necessary packages to host the server.
 - Once you have have a database user example: `dbuser` with password `dbuserpassword`, set up a database named `dbname`, and decided on a unique secret key `yoursecretkey`, only known to you, and not saved in the repository, proceed to the next step.
 - add the following lines to the `venv_greenvvz/bin/activate_this.py` file, to make sure the server has the necessary information:

```
os.environ["FLASK_ENV"]="development" # Comment line out for production!
os.environ["DB_USER"]="dbuser"
```

```
os.environ["DB_PASSWORD"]="dbuserpassword"
os.environ["DB_NAME"]="dbname"
os.environ["SECRET_KEY"]="yoursecretkey"
```

4. Set up the HTTP server to serve files using the Web Server Gateway Interface (WSGI) in your user/www directory. For this example, let's say your username is `USER`.

- Put the application.wsgi file there, with the following contents(*the helpful techs at UZH did this for me*):

```
import sys

activate_this = '/home/USER/venv_greenvvz/bin/activate_this.py'
with open(activate_this) as file_:
    exec(file_.read(), dict(__file__=activate_this))

sys.path.insert(0, '/home/USER/greenvvz')
from main import app as application
```

5. After this, you should be good to go! Check out if your server is running at <https://yourserver.ifi.uzh.ch>! If you run into errors, make sure to check out the logs you set up in step 4, or UZH kindly set up for you.

Developer's Guide for a local setup

1. Make sure you have git and python ≥ 3.5 installed.
2. Set up a local dev environment in a folder of your liking: `python3 -m virtualenv venv_[choose a name]` Or `python3 -m venv venv_[choose a name]`. For this example, let's say you chose `greenvvz` as the name.
 - This will create a folder `venv_greenvvz`.
 - use `venv_greenvvz/bin/activate` to activate the virtual env.
3. Git clone this project to a folder of your liking: `git clone https://github.com/bbodo/greenvvz.git`.
4. Activate the virtualenv, and navigate to the project folder. `python 3 -m pip install -r requirements.txt` to get all necessary packages.
5. Add the following information to your environment, either using the virtualenv file or manually (this depends on your system, google how to add environmental variables):

```
os.environ["FLASK_ENV"]="development"
os.environ["DB_USER"]="dbuser"
os.environ["DB_PASSWORD"]="dbuserpassword"
os.environ["DB_NAME"]="dbname"
os.environ["SECRET_KEY"]="yoursecretkey"
```

5. Once this is done, you are good to go! Use `flask run` to use your local server on localhost:5000!

Guide to set up a working instance on Magnolia:

6. Now, to host your server on an UZH page in Magnolia, do the following:
- Create a page `greenvvz-admin` for the administrator view. I recommend Inhaltsseite 1-Spaltig.
 - In the Content subsection, add an HTML element with the following contents:

```
<iframe id="greenvvz-admin-iframe" class="mod mod-iframe" src="https://yourserver.ifi.uzh.ch/st

<script type="text/javascript" src="https://yourserver.ifi.uzh.ch/st

<script>
  iFrameResize({
    log: false,
    // heightCalculationMethod: 'max',
  },
  '#greenvvz-admin-iframe')
</script>
```

- Create a page `greenvvz-public` for the public view
 - In the Content subsection, add an HTML element with the following contents:

```
<iframe id="greenvvz-public-iframe" class="mod mod-iframe" src="http

<script type="text/javascript" src="https://yourserver.ifi.uzh.ch/st

<script>
  iFrameResize({
    log: false,
    // heightCalculationMethod: 'max',
  },
  '#greenvvz-public-iframe')
</script>
```


11.2 Code Documentation

The code documentation is appended on the following pages, organized by file. It starts with the back-end Python documentation:

1. main.py
2. models.py
3. helpers.py
4. updateModules.py

Then, the front-end JavaScript documentation follows:

5. admin.js
6. public.js
7. filter.js

It is also attached in electronical form, in this document and as separate documents. Furthermore, it can also be found in the Git repository in the 'docs' folder. It is available on:

<https://github.com/bbodo/GreenVVZ/tree/master/docs>

Back end: main.py Documentation

main.add_module()

Add module to database. required in POST request body: SmObjId, PiqYear, PiqSession, whitelisted, searchterm

main.add_searchterm()

Add searchterm to DB, term is supplied in form data

main.admin()

Administrator front end view

main.app = <Flask 'main'>

main.check_which_saved(modules: list)

Check which modules are saved, and mark them as either white- or blacklisted accordingly

main.cors = <flask_cors.extension.CORS object>

main.db_config = {'database': 'greenvvzdb', 'host': '127.0.0.1', 'password': 'greenvvzpw', 'user': 'greenvvz'}

main.find_modules_for_course(course: dict)

Request detail page for course object, add Module subobjects(dict) as list to given course object

main.find_studyprograms_for_module(SmObjId: int, PiqYear: int, PiqSession: int) → list

Request detail page for module object, add Studyprogram subobjects(dict) as list to given module obj

main.flag_module(module_id: int)

Flag saved module as whitelisted or blacklisted, depending on request.args.get('whitelisted')

main.get_blacklist()

main.get_modules(whitelisted: bool)

Get modules saved in the database, either blacklisted or whitelisted, as JSON response

main.get_searchterms()

get all search terms from DB

main.get_studyprograms()

Get distinct studyprograms associated with modules in the whitelist

main.get_studyprograms_modules()

Get Module-Studyprogramids associations as a dictionary

main.get_whitelist()

main.hello_world()

Hello World test view

main.info()

Information about the API

main.public()

Public front end view

main.remove_module(module_id: int)

remove module from database by id

main.remove_searchterm(*searchterm_id: int*)

remove searchterm from DB via id

main.require_appkey(*view_function*)

decorator for checking the api-key, making unauthorized requests impossible

main.save_studyprograms_for_module(*module_id: int, studyprograms: list*)

Save studyprogams for module in database, establish relationship

main.search()

get modules based on search terms, marking those already on white- and blacklist

main.search_upwards()

Find course matches, then find containing modules, # and containing study programs

main.update()

Update saved modules to match their course catalogue counterparts, be there any changes

main.wrap_execute_for_modules_in_course(*course*)

Wrapper function to be able to parallelize finding studyprograms for modules

Back end: models.py Documentation

`class models.Globals`

Bases: **object**

`URI_prefix = 'https://studentservices.uzh.ch/sap/opu/odata/uzh/vvz_data_srv/'`

`class models.Module(SmObjId: int, PiqYear: int, PiqSession: int)` ¶

Bases: **object**

Class to hold Module logic and data. Hardly used to full potential, could rework to use more of this.

find_module_values() → dict

Check if module with given SmObjId and session data exists in course catalogue, get values if it does

get_module() → dict

Get this module's variables if module exists, else None

set_module(values: dict)

sets module to provided values

Back end: helpers.py Documentation

`helpers.current_year()` → int
returns current year as int

`helpers.get_current_sessions(num_prev_semesters: int = 4)` → list
Get next, current, and last num_prev_semesters sessions

`helpers.get_session(ref_date=datetime.date(2019, 6, 2), target_date=None)` → dict
returns a dictionary containing the values 'year' and 'session' as of ref_date (default is today) ref_date to compare to the target_date, which is feb of the ref year by default

SPRING: PREV YEAR 004 FALL: SAME YEAR 003


Back end: updateModules.py Documentation

`updateModules.update_modules()` → bool

For each saved module, check if it still exists and/or changed, match in DB

Front End: Admin.js

Documentation

 `add_to_blacklist(module_id=undefined, SmObjId, PiqYear, PiqSession, title, searchterm) ⇒ void` global

Write table row for module in blacklist, append to blacklist table

Parameters

- `module_id` : `Number` Default: `undefined`

numerical part of the module CSS selector id, matches DB id

- `SmObjId` : `Number` Required

course catalogue id

- `PiqYear` : `Number` Required

course catalogue year

- `PiqSession` : `Number` Required


course catalogue year

- `title` : `String` Required

module title

- `searchterm` : `String` Required

searchterm which found the module in the course catalogue

 `add_to_searchterms(id, term) ⇒ void` global

Write table row for searchterm and add to DOM.

Parameters


- `id` : `Number` Required

Documentation

- **term** : `String`

Required

searchterm value

 `add_to_suggestions(module_id=undefined, SmObjId, PiqYear, PiqSession, title, whitelisted, searchterm) ⇒ void` `global`

Write table row for module in suggestions, append to suggestions table

Parameters

- **module_id** : `Number`

Default: `undefined`

numerical part of the module CSS selector id, matches DB id

- **SmObjId** : `Number`

Required

course catalogue id

- **PiqYear** : `Number`

Required

course catalogue year

- **PiqSession** : `Number`

Required

course catalogue year

- **title** : `String`

Required

module title

- **whitelisted** : `Boolean`


Required

whitelisted the whitelist status

- **searchterm** : `String`

Required

Documentation

 `add_to_whitelist(module_id=undefined, SmObjId, PiqYear, PiqSession, title, searchterm) ⇒ void` global

Write table row for module in whitelist, append to whitelist table

Parameters

- `module_id` : Number Default: undefined

numerical part of the module CSS selector id, matches DB id

- `SmObjId` : Number Required

course catalogue id

- `PiqYear` : Number Required

course catalogue year

- `PiqSession` : Number Required

course catalogue year

- `title` : String Required

module title

- `searchterm` : String Required

searchterm which found the module in the course catalogue

 `checkUpdatedCookie()` ⇒ void global

Check if the "updated_recently" cookie exists - if it doesn't, updateModules()

Documentation

 `convert_session_to_string(session, year)` \Rightarrow `String` global

Convert session to human readable span element as string.

Parameters


- `session` : `Number` Required

session code, either 3, 4, 003, 004.

- `year` : `Number` Required

module year data.

Returns: `String` — A span containing the humanreadable semester and year.


 `delete_blacklisted_module(module_id)` \Rightarrow `void` global

Delete module from the DB.

Parameters

- `module_id` : `Number` Required

DB id for module to delete

 `delete_searchterm(id)` \Rightarrow `void` global


Delete searchterm from the DB.

Parameters

- `id` : `Number` Required

DB id for searchterm to delete

Documentation

 `flag_in_suggestions(module_id, whitelisted)` \Rightarrow void

global

Flag module as whitelisted or blacklisted in the suggestions, changing Status and (de)activates corresponding buttons.


Parameters

- `module_id` : `Number` Required

the numerical part of the CSS selector

- `whitelisted` : `Boolean` Required

the whitelist status

 `getCookie(cname)` \Rightarrow `String` global


Get cookie by name

Parameters


- `cname` : `String` Required

name of the cookie

Returns: `String` — the value of the cookie

 `populate_blacklist()` \Rightarrow void global

Request blacklisted modules from server, add them to DOM.

 `populate_searchterms()` \Rightarrow void global


Request searchterms from server, add them to DOM.

Documentation


Request studyprograms for selected semester from server, as well as studyprogamid_moduleids list, add them to global JS scope.

 `populate_suggestions()` \Rightarrow void global

Request found modules from server, add them to DOM.

 `populate_whitelist()` \Rightarrow void global

Request whitelisted modules from server, add them to DOM.

 `post_module_to_db(module_id=undefined, SmObjId, PiqYear, PiqSession, whitelisted, searchterm)` \Rightarrow void global

Flag tablerows for modules OF THE WHITELIST ONLY contained in the studyprogram input via #studyprogram_input.

Parameters

- `module_id` : Number Default: undefined

numerical part of the module CSS selector id.

- `SmObjId` : Number Required

course catalogue id.

- `PiqYear` : Number Required

course catalogue year.

- `PiqSession` : Number Required

course catalogue year.


Documentation

whitelist status of the module to save.

- **searchterm** : `String`

Required

searchterm which found the module in the course catalogue

 **remove_from_searchterms(id)** \Rightarrow void `global`


Remove searchterm from the DOM.

Parameters

- **id** : `Number`

Required

numerical part of CSS selector id for searchterm to delete, matches DB id.

 **remove_module(module_id)** \Rightarrow void `global`


Remove module from the DOM.

Parameters


- **module_id** : `Number`

Required

the numerical part of the CSS selector, matches DB id.


 **save_module()** \Rightarrow void `global`

Save module from SmObjId input into the DB, using the selected semester or current if all selected.

 **save_searchterm()** \Rightarrow void `global`

Save searchterm from input into the DB.

Documentation

 `setCookie(cname, cvalue, exdays)` \Rightarrow void global

Save a cookie which expires after exdays days.

Parameters

- `cname` : `String` Required


name of the cookie

- `cvalue` : Required

value of the cookie

- `exdays` : `Number` Required

days until cookie expires

 `update_whitelist_status(module_id, whitelisted)` \Rightarrow void global

Flag tablerows for modules OF THE WHITELIST ONLY contained in the studyprogram input via #studyprogram_input.

Parameters

- `module_id` : `Number` Required


numerical part of the module CSS selector id, matches the DB id for saved modules.

- `whitelisted` : `Boolean` Required

whitelist status of the module to save.

 `updateModules()` \Rightarrow void global

Documentation

 `write_tr_prefix_for_list(module_id=undefined, SmObjId, PiqYear, PiqSession, title, searchterm) ⇒ String` global

Generate generalizable part of table row for a given module - what differs are buttons and suffixes.

Parameters

- `module_id` : `Number` Default: undefined

numerical part of the module CSS selector id, mismatches DB id if in suggestions

- `SmObjId` : `Number` Required

course catalogue id

- `PiqYear` : `Number` Required

course catalogue year

- `PiqSession` : `Number` Required

course catalogue year

- `title` : `String` Required

module title

- `searchterm` : `String` Required

searchterm which found the module in the course catalogue

Returns: `String` — String matching an opened tr DOM element, with td elements inside

Front End: Public.js

Documentation



`convert_session_to_string(session, year)` ⇒ `String` global

Convert session to human readable span element as string. DIFFERENT FROM admin.js

Parameters

- `session` : `Number` Required

The session code, either 3, 4, 003, 004.

- `year` : `Number` Required

the module year data.

Returns: `String` — A span containing the humanreadable semester and year.

Documentation built with [Docma](#).

Front End: Filter.js

Documentation


 **ClearStudyProgramFilter()** \Rightarrow void global

Clear the studyprogram input

 **FlagSelectedSemester()** \Rightarrow jQuery() global

Flag tablerows for modules in the semester chosen by the dropdown selector "semester".

Returns: jQuery() — The tablerows of modules matching the currently selected semester.


 **FlagSelectedStudyprogram()** \Rightarrow jQuery() global

Flag tablerows for modules OF THE WHITELIST ONLY contained in the studyprogram input via #studyprogram_input.

Returns: jQuery() — The tablerows of modules in the whitelist matching the currently selected studyprogram.

 **monkeyPatchAutocomplete()** \Rightarrow void global

monkey patch into jquery autocomplete, for custom rendering and highlighting found elements, and if necessary, searching only from beginning of result (commented out)

 **ShowSelectedModules()** \Rightarrow jQuery() global

Render items using CSS classes according to their selection status

Returns: jQuery() — The tablerows of modules matching the current filters.
