

**Eignung eines Dateibasierten Speichersystems für eine Webapplikation für  
Dokumentationen**

Projektarbeit	I	II	III	IV
Nr.	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

vorgelegt am: 02.05.25

von: Musterman, Max

Matrikelnr: G000000PI

DHGE Campus: Gera

Studienbereich Technik

Studiengang: Komplexe Informatik

Kurs: KIA89

Ausbildungsstätte: Capgemini Deutschland GmbH

Betreuer: Musterman, Max

Musterman, Max

## **Sperrvermerk**

Die vorgelegte Projektarbeit mit dem Titel Eignung eines Dateibasierten Speichersystems für eine Webapplikation für Dokumentationen basiert auf internen, vertraulichen Daten und Informationen des Unternehmens BETRIEB.

Diese Projektarbeit darf nur vom Erst- und Zweitgutachter sowie berechtigten Mitgliedern des Prüfungsausschusses eingesehen werden. Eine Vervielfältigung und Veröffentlichung der Projektarbeit ist auch auszugsweise nicht erlaubt.

Die Vervielfältigung und Veröffentlichung der Projektarbeit sowie die Einsichtnahme durch Dritte bedarf der ausdrücklichen Zustimmung des Verfassers und des Unternehmens.

# Inhaltsverzeichnis

Abbildungsverzeichnis . . . . .	II
Tabellenverzeichnis . . . . .	III
Abkürzungsverzeichnis . . . . .	IV
1      Einleitung . . . . .	1
2      Markdown . . . . .	2
3      Django . . . . .	3
4      Umsetzung . . . . .	5
4.1    Organisation und Anzeigen der markdown Dateien . . . . .	5
4.2    Hochladen von markdown Dateien . . . . .	6
4.3    Bearbeiten von Texten . . . . .	7
5      Fazit . . . . .	9
5.1    Problemanalyse und mögliche Mängel der Applikation . . . . .	9
5.2    Beantwortung der wissenschaftlichen Frage . . . . .	9
5.3    Ausblick . . . . .	10
Literaturverzeichnis . . . . .	V
Anlagenverzeichnis . . . . .	VI
Ehrenwörtliche Erklärung	

## Abbildungsverzeichnis

Abbildung 1: Markdown Rohtext . . . . .	2
Abbildung 2: Markdown formatiert . . . . .	2
Abbildung 3: Eingebauter Django login . . . . .	4
Abbildung 4: Eingebautes Django Admin Fenster . . . . .	4
Abbildung 5: breite Leseansicht . . . . .	5
Abbildung 6: schmale Leseansicht . . . . .	6
Abbildung 7: Menü zur Kapitelauswahl . . . . .	6
Abbildung 8: Administrationsseite zur Bearbeitung eines Datenbankobjekts . . . . .	7
Abbildung 9: Textbearbeitungsseite . . . . .	8

## Tabellenverzeichnis

## Abkürzungsverzeichnis

<b>DHGE</b>	Duale Hochschule Gera Eisenach
<b>SQL</b>	Structured Query Language
<b>Bash</b>	Bourne-again shell
<b>JDK</b>	Java Development Kit
<b>VM</b>	Virtuelle Maschine

# 1 Einleitung

Für digitale Anwendungen ist es üblich eine Dokumentation bereitzustellen. Diese sind manchmal dem Code beigelegt, zumeist sind sie jedoch über eine Webseite abrufbar. Webseiten werden in **html!** (**html!**) geschrieben. Diese für jedes Programm von Grund auf neu zu erstellen ist sehr umständlich und setzt bei komplizierteren Strukturen eine gewisse Vertrautheit mit der Sprache voraus. Aus diesem Grund werden Dokumentationen häufig entweder mit bereits existierenden Programmen erstellt oder aus einer anderen Sprache generiert. Eine beliebte Sprache zum Schreiben von Dokumentationen ist Markdown.<sup>1</sup> Dokumentationen können auf mehrere Weisen erstellt werden, darunter automatisch aus Kommentaren die im Code stehen, oder manuell eingetippt. Die resultierenden Texte können auf verschiedene Weisen gespeichert werden. In dieser Arbeit wird eine dateibasierte Speicherlösung betrachtet, um die folgende Frage zu beantworten: Inwieweit ist eine dateibasierte Speicherlösung für eine Webapplikation zum Hochladen, Abspeichern und Darstellen von technischen Dokumentationen geeignet?

Das Ziel der Arbeit ist es, eine Applikation zu entwickeln, die eigenständig auf einem Server laufen kann. Diese soll sowohl Dateien, die mittels externer Programme, wie Dateimanagern, in den betreffenden Bereich geladen werden, als auch Dateien die über das Programm hochgeladen werden, erkennen und richtig gerendert darstellen können. Eine grundlegende Funktion zur Navigation zwischen den verschiedenen Dateien muss gegeben sein. Die Dateien sollen allesamt im Markdown Format abgespeichert sein. Die Praxisphase während welcher diese Arbeit entstanden ist, wurde in einem Team absolviert, welches Python als Hauptprogrammiersprache für sich festgelegt hat. Um von dem Wissen des Teams zu profitieren, wurde das Django-Framework für die Entwicklung der geforderten Applikation ausgewählt.

---

<sup>1</sup>Vgl. [Con20], S. 3

## 2 Markdown

Markdown ist eine sogenannte „markup language“ (Auszeichnungssprache) die 2004 eingeführt wurde.<sup>2</sup> Sie dient der plattformübergreifenden und softwareunabhängigen Formatierung von Texten. Markdown ist eine maschinenlesbare Sprache (Vgl. Abb. 1) zum Gestalten von Texten, welche es geeigneten Programmen erlaubt, die Texte richtig darzustellen (Vgl. Abb.2). Dies steht im Gegensatz zu **WYSIWYG!** (**WYSIWYG!**) Editoren, die ihre Texte in einem speziellen Dateiformat speichern, um die Formatierung zu erhalten. **WYSIWYG!** Editoren geben den Text immer so aus, wie er während des Bearbeitens angezeigt wird. Die Formatierung wird hier meist mit Schaltflächen eingestellt. Es ist aber auch möglich Markdown als Rohtext zu lesen, ohne dass es gerendert wurde. Dies ist einer der Vorteile von Markdown im Vergleich zu anderen Textformaten, wie zum Beispiel dem .docx Format von Microsoft Word.<sup>3</sup> Markdown ist der Standard für Texte auf mehreren großen Seiten im Internet, darunter GitHub und Reddit.<sup>4</sup> Auch innerhalb des Deutsche Bahn Konzerns ist Markdown verbreitet, da die meisten Projekte eine sogenannte „README.md“ Datei haben, die grundlegende Informationen über das Projekt enthält und in der Regel im Markdown Format gespeichert ist. Markdown kann mittels eines sogenannten „Parsers“ in **html!** umgewandelt werden.<sup>5</sup> Es kann überall als Beschreibungssprache eingesetzt werden. Zudem gibt es eine bereits vorgefertigte Lösungen zur Darstellung von Markdown. Aus diesen Gründen wurde Markdown als vorausgesetzte Formatierung der Texte gewählt.

```
1  Markdown is a light text markup format and a processor to convert that to HTML.
2  The originator describes it as follows:
3
4  > Markdown is a text-to-HTML conversion tool for web writers.
5  > Markdown allows you to write using an easy-to-read,
6  > easy-to-write plain text format, then convert it to
7  > structurally valid XHTML (or HTML).
8  >
9  > -- <http://daringfireball.net/projects/markdown/>
```

Abbildung 1: Markdown Rohtext

```
Markdown is a light text markup format and a processor to convert that to HTML. The originator describes it as follows:

Markdown is a text-to-HTML conversion tool for web writers. Markdown allows you to write using an easy-to-read, easy-to-write plain text
format, then convert it to structurally valid XHTML (or HTML).

-- http://daringfireball.net/projects/markdown/
```

Abbildung 2: Markdown formatiert

---

<sup>2</sup>Vgl. [Con20], S. 1

<sup>3</sup>Vgl. [Con20], S. 2f.

<sup>4</sup>Vgl. [Con20], S. 3

<sup>5</sup>Vgl. [Con20], S. 4f.



### 3 Django

Django ist ein Web-Framework in der Programmiersprache Python.<sup>6</sup> Es ist dafür gebaut worden möglichst schnell neue Webapplikationen zu entwickeln. Das Framework unterstützt Entwickler dabei, Redundanzen zu vermeiden, und die Wartung der Applikation zu reduzieren. Dank einem sehr geringen Einrichtungsaufwand und einer großen Menge an Dokumentationen und Hilfestellungen im Internet ist Django sehr einsteigerfreundlich.<sup>7</sup> Das Framework übernimmt unter anderem die Programmschnittstelle, Datenbanksteuerung und das **URL! (URL!) Management**.<sup>8</sup>

Django nutzt einen **ORM! (ORM!)** um die Datenbank darzustellen, was das Arbeiten mit ihr vereinfacht. Django übernimmt die Umwandlung der Befehle in Datenbankabfragen. Zudem hat es eine integrierte Nutzerauthentifizierung (Vgl. Abb. 3) und eine Administrationsseite (Vgl. Abb. 4), welches die direkte Bearbeitung der Daten innerhalb der Datenbank während des Betriebs ohne Schwierigkeiten zulässt.<sup>9</sup>

Um Objekte einer Klasse (Vgl. Abb. 4.1) auf der Administrationsseite zu bearbeiten (Vgl. Abb. 4.2), ist es nötig sie zu registrieren und die Strukturen vor dem Starten des Servers migrieren. Beim Registrieren der Modelle kann die eingebaut Admin-Klasse verändert werden (siehe Anhang 9).

Django nutzt für die Erstellung von Vorlagen, auch „Templates“ genannt, die „Django template language“, die es erlaubt mit einer einzigen **html!** Datei verschiedenen Inhalte zu präsentieren. Um die **html!**-Dateien den **URL!**s zuzuordnen werden sogenannte „Views“ genutzt, die die eingehende Anfrage bearbeiten und beliebigen Code ausführen können, bevor die **html!**-Vorlage aufgerufen wird. Sie werden auch genutzt, um der Vorlage den benötigten Kontext mitzugeben. Dieser besteht aus Variablen, welche dann mit der „Django template language“ angesprochen und benutzt werden können.

Django hat Vorlagen für mehrere Anwendungen, darunter die Administrationsseite und Eingabefelder, vorinstalliert. Diese haben eine festgelegte Formatierung und Funktionsweise. Bei einigen Anwendungen kann es notwendig sein, diese vorgegebenen Vorlagen zu verändern. Standardmäßig werden die vorinstallierten Vorlagen vorrangig behandelt, doch das lässt sich ändern. Django hat verschiedene Renderer. Diese durchsuchen festgelegte Verzeichnisse nach Vorlagen. Die Reihenfolge in denen die Verzeichnisse durchsucht werden ist dabei vom Renderer Abhängig. Es ist außerdem möglich eigene Renderer zu erstellen, falls die Standardmäßigen Renderer nicht ausreichen.<sup>10</sup>

Django bietet mehrere Möglichkeiten zu Bereitstellung der Webseite, was die Skalierbarkeit

---

<sup>6</sup>Vgl. [Dja25]

<sup>7</sup>Vgl. [Vin22], S. 15

<sup>8</sup>Vgl. [CAD+20], S. 99–101

<sup>9</sup>Vgl. [CAD+20], S. 99–102

<sup>10</sup>Vgl. [Dja25]

Django administration

Username:

Password:

Log in

Abbildung 3: Eingebauter Django login

vereinfacht.<sup>11</sup> Im Rahmen dieser Projektarbeit wird Django in Docker bereitgestellt.

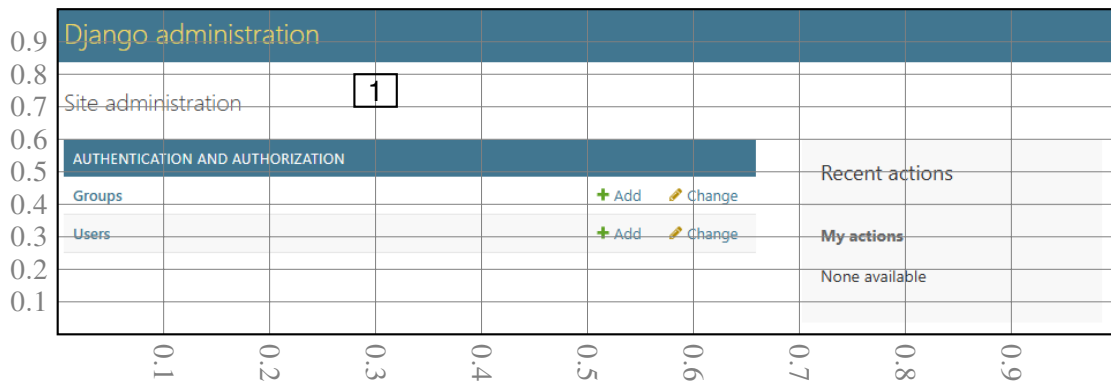


Abbildung 4: Eingebautes Django Admin Fenster

<sup>11</sup>Vgl. [CAD+20], S. 99–100

## 4 Umsetzung

### 4.1 Organisation und Anzeigen der markdown Dateien

Die Dateien sind alle im selben Verzeichnis gespeichert und folgen demselben Muster für ihre Dateinamen, in diesem Fall „Kapitel\_[NUM].md“. Dieses Muster wurde gewählt, da die vorliegenden Testdateien nach diesem Muster benannt sind. Alle Dateien die eine der beiden Bedingungen nicht erfüllen, werden ignoriert. Dies erlaubt einen einfachen Zugriff über den Dateipfad und erlaubt es die Dateien nicht in eine Datenbank schreiben zu müssen. Die Webseite bezieht den Text direkt aus den Dateien, indem sie sie öffnet, liest, umwandelt und dann den resultierenden Text an die Vorlage gibt. Welche Datei gelesen werden muss, wird in der **URL!** mitgegeben. Sie folgt dem Format „/r/[NUM]“. Django leitet diese Anfrage auf die entsprechende View weiter. Diese übergibt der Vorlage den höchsten Seitenindex, eine Liste aller Seiten und die aktuelle Seite, die in der **URL!** angegeben ist, als Kontext. Die Vorlage ruft einen sogenannten „Filter“ auf (siehe Anhang 8). Dieser ist eine Funktion in Python, welche die aktuelle Seitennummer als Eingabe nimmt, die entsprechende Datei öffnet und ausliest (siehe Anhang 1). Der Text der geöffneten Datei wird in der Funktion, mittels des „markdown2“ Moduls von Python vom Markdown Format in **html!** umgewandelt. Das Resultat wird an die Vorlage zurückgegeben und, mithilfe der benutzten „safe“ Option, als Teil des Dokuments dargestellt und richtig formatiert. Die Seite ist so formatiert, dass die Textbreite niemals über achthundert Pixel breit wird, um zu vermeiden, dass der Text unangenehm zu lesen wird (Vgl. Abb. 5). Ab einer Bildschirmbreite von weniger als eintausend Pixeln schaltet die Webseite in den Mobilien Modus um, der die Breite des Textes auf neunzig Prozent des Bildschirmes begrenzt und am oberen Rand Platz für die Steuerschaltflächen lässt.

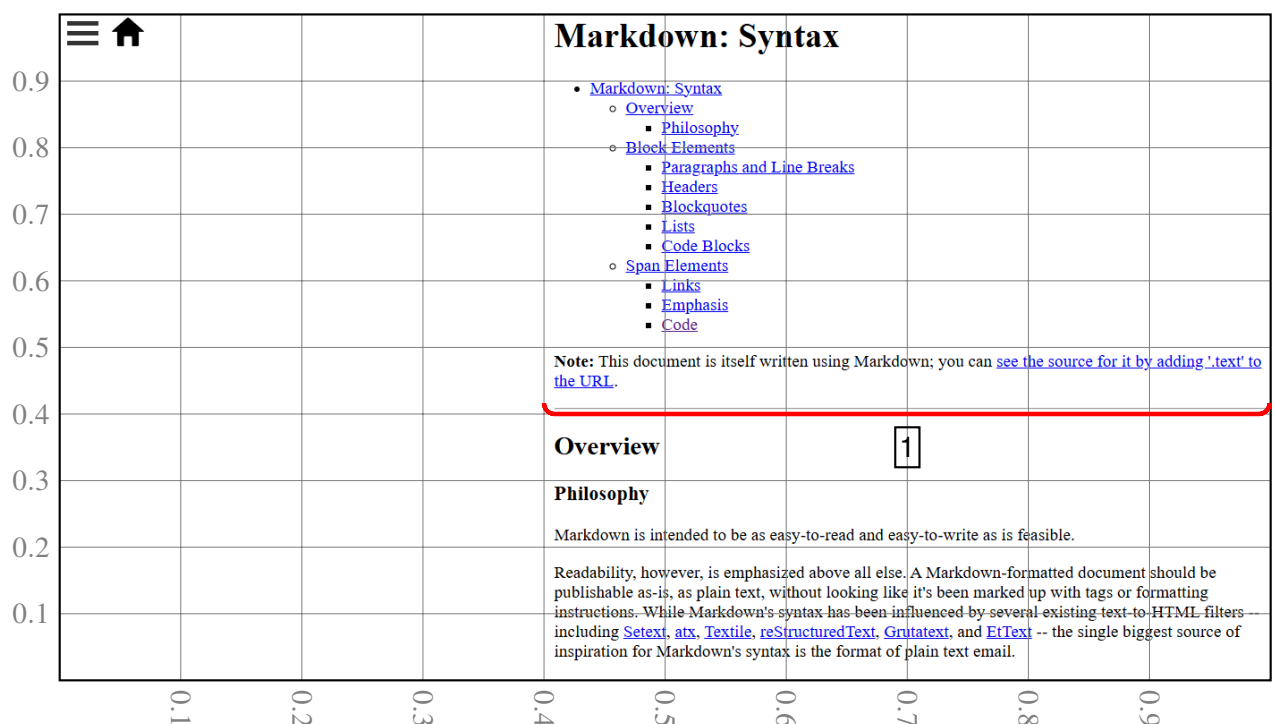


Abbildung 5: breite Leseansicht

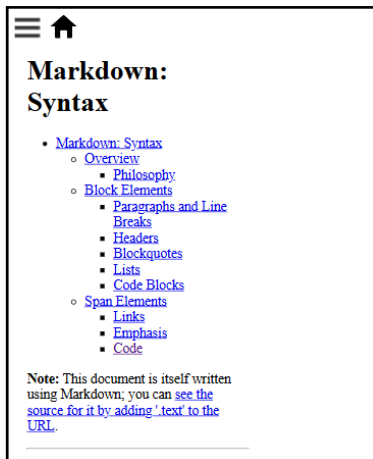


Abbildung 6: schmale Leseansicht

Für die Navigation werden die Kapitelnummern genommen, die im Link enthalten sind und an die Vorlage weitergegeben. Das Wechseln der Seiten ist entweder durch die manuelle Eingabe des Links möglich, durch ein Menü (Vgl. Abb. 7), indem vor oder zurückgeblättert wird, oder durch die Startseite, die eine Liste aller Kapitel beinhaltet. Um die Liste aller Kapitel zu erhalten, wird im View vom ersten Kapitel hochgezählt, bis keine passende Datei mehr gefunden wird und die Liste als Kontext an die Vorlage weitergegeben. Diese Liste wird genutzt um die Navigationsleiste zu erstellen. Um die Titel der Seiten anzuzeigen, wird die Liste Objekt für Objekt durchgegangen und jede Nummer an einen Filter weitergegeben. Dieser Filter gibt die erste beschriebene Zeile aus, wobei alle vorangehenden Leer- und Steuerzeichen ignoriert werden. (siehe Anhang 2). Diese Navigation funktioniert, indem der Benutzer auf die Seite mit der entsprechenden Zahl weitergeleitet wird.

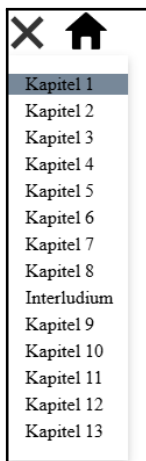


Abbildung 7: Menü zur Kapitelauswahl

## 4.2 Hochladen von markdown Dateien

Für das Hochladen von Dateien sollte hier die Administrationsseite benutzt werden, um die eingebaute Nutzer Authentifizierung zu benutzen. Es gibt zwei Möglichkeiten das Ziel zu erreichen.

Die erste Möglichkeit ist eine Erstellung einer eigenen Vorlage, das von der Administrationsseite zugänglich ist. Diese Möglichkeit erfordert eine Abänderung der Vorlagen für die Administrationsseite. Diese wurde erst am Ende der Arbeit entdeckt, was die Umsetzung innerhalb des Zeitramens nicht mehr zuließ. Bevor diese Möglichkeit gefunden wurde, wurde davon ausgegangen, dass eine komplett eigene Administrationsseite hätte entwickelt werden müssen. Dies wäre ein zu großer Aufwand gewesen um ihn zu rechtfertigen.

Die zweite Möglichkeit besteht darin, die Datenbank als Stellvertreter für den Ordner zu nutzen. Django bietet die Möglichkeit eine Dateireferenz in einem Datenbankobjekt zu speichern. Dies hat allerdings einige Nachteile. Unter anderem werden standardmäßig die Dateien entweder überschrieben oder die Dateinamen werden durch Zufallszahlen abgeändert, um das Überschreiben von Dateien zu vermeiden. Außerdem werden bereits vorhandene Dateien nicht in die Datenbank übertragen.

Um einige dieser Nachteile auszugleichen wurden eigene Speicher- und Löschfunktion entwickelt. Die Speicherfunktion prüft zuerst ob es bereits eine Datei mit dieser Nummer gibt und, falls sie Unterschiede aufweist, verschiebt sie diese in einen anderen Ordner, um zu vermeiden, dass Daten verloren gehen. Dasselbe geschieht beim Löschen, wobei die Datei wird in einen anderen Ordner verschoben und der Dateiname wird mit einem Zeitstempel versehen wird. Zudem wurde der Aufruf der Administrationsseite so abgeändert, dass über jede Datei im Ordner iteriert und für sie ein Objekt erstellt wird (Vgl. Abb. 8). Da die Darstellung der Dateien, und somit auch das Abrufen der Daten, unabhängig von der Datenbank geschieht, muss dieser Schritt nicht beim Start der Webseite ausgeführt werden.

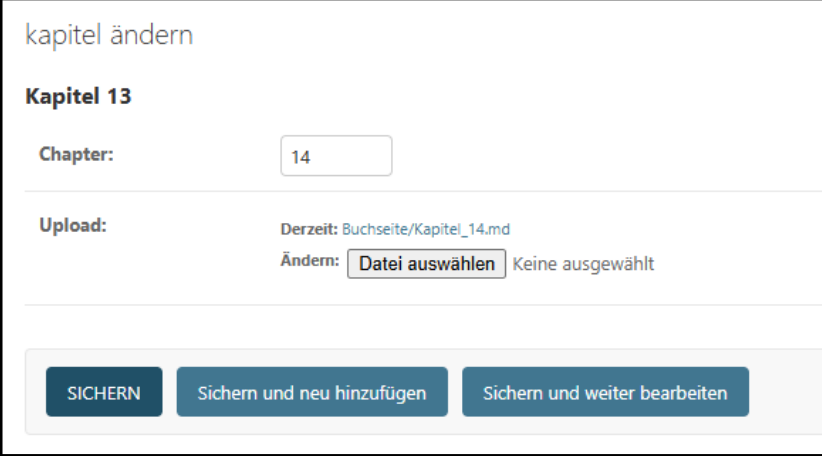
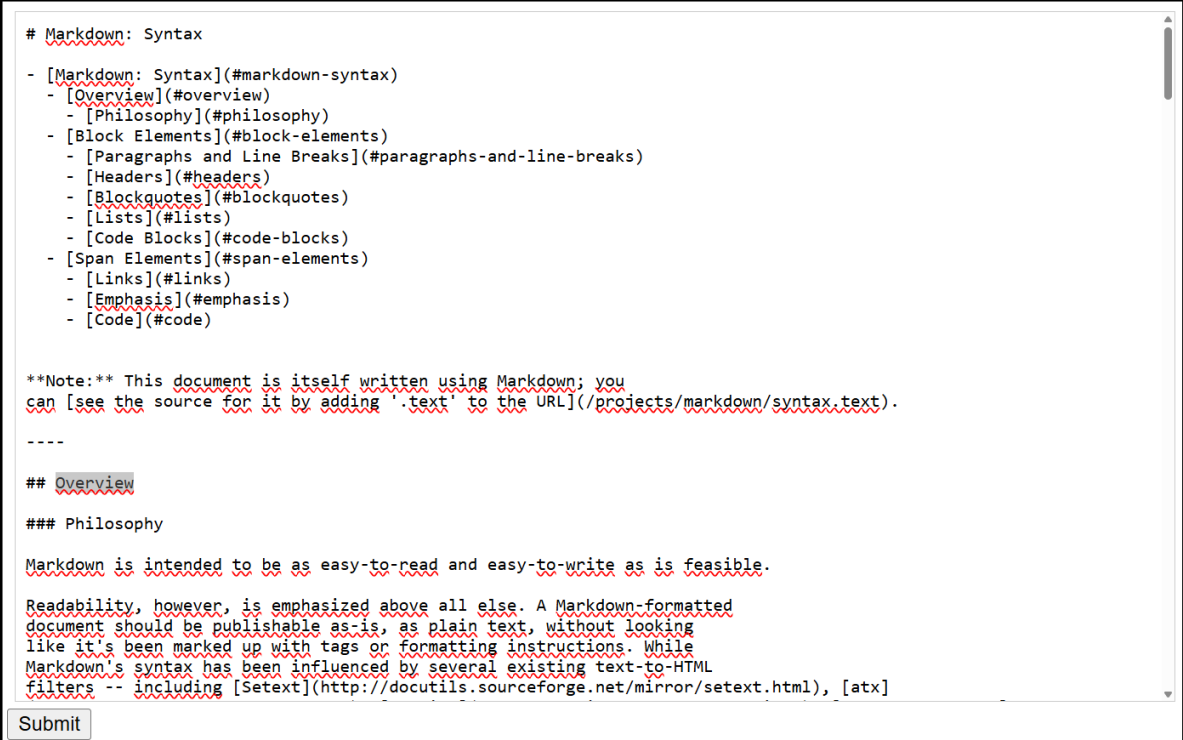


Abbildung 8: Administrationsseite zur Bearbeitung eines Datenbankobjekts

#### 4.3 Bearbeiten von Texten

Das Bearbeiten von Texten erfordert ein Eingabefeld, dessen Inhalt extrahiert werden kann und das den existierenden Inhalt der zu bearbeitenden Datei bereits beinhaltet. Es gibt bei Django vordefinierte Funktionen und Vorlagen, die das eingeben und abspeichern von Daten erlauben. Diese sogenannten „forms“ basieren auf der **html!** Funktion mit demselben Namen. Sie erlauben eingegebene Daten als „Post“ oder „Get“ Anfragen an eine definierte html-Adresse

weiterzugeben. Bei der Post Methode verpackt der Browser die Daten, schickt sie an den Server und erwartet eine Antwort. Bei der Get Methode werden die eingegebenen Daten in einen String verpackt und direkt als URL verwendet. Die Get Methode sollte genutzt werden, wenn die Eingabe den Status der Daten auf dem Server nicht ändert, in diesem Fall, nur wenn die Texte nicht geändert werden. Die Post Methode, die hier auch verwendet wird, sollte genutzt werden, wenn Daten auf dem Server geändert werden sollen.<sup>12</sup>. Die Daten können im View der Webseite extrahiert werden können, wenn das **html!** Formular richtig konfiguriert ist (siehe Anhang 4). Hier werden sie direkt im View in die entsprechenden Dateien gespeichert und der Benutzer auf die Leseseite des bearbeiteten Textes weitergelitet. Django hat vordefinierte Vorlagen für Formularfelder, die „Widgets“ genannt werden.. Um eine nutzbare Formatierung des Eingabefeldes zu erreichen (Vgl. Abb. 9), mussten diese hier überschrieben werden. Django stellt dafür einen Renderer ein, der es erlaubt Vorlagen zu überschreiben, wenn sie im selben Unterordner des Template Ordners liegen und den selben Namen haben (siehe Anhang 5, 6 und 7).



```
# Markdown: Syntax

- [Markdown: Syntax](#markdown-syntax)
- [Overview](#overview)
  - [Philosophy](#philosophy)
- [Block Elements](#block-elements)
  - [Paragraphs and Line Breaks](#paragraphs-and-line-breaks)
  - [Headers](#headers)
  - [Blockquotes](#blockquotes)
  - [Lists](#lists)
  - [Code Blocks](#code-blocks)
- [Span Elements](#span-elements)
  - [Links](#links)
  - [Emphasis](#emphasis)
  - [Code](#code)

**Note:** This document is itself written using Markdown; you
can [see the source for it by adding '.text' to the URL](/projects/markdown/syntax.text).

----

## Overview

### Philosophy

Markdown is intended to be as easy-to-read and easy-to-write as is feasible.

Readability, however, is emphasized above all else. A Markdown-formatted
document should be publishable as-is, as plain text, without looking
like it's been marked up with tags or formatting instructions. While
Markdown's syntax has been influenced by several existing text-to-HTML
filters -- including [Setext](http://docutils.sourceforge.net/mirror/setext.html), [atx]
```

Abbildung 9: Textbearbeitungsseite

<sup>12</sup>Vgl. [Dja25], Working with forms

## 5 Fazit

### 5.1 Problemanalyse und mögliche Mängel der Applikation

Da diese Applikation den Inhalt der Seiten über das Dateisystem des unterliegenden Betriebssystems bezieht, ist es durch dieses in der Geschwindigkeit und Stabilität begrenzt. Dateisysteme sind in der Regel weniger effizient und können so weniger Last aushalten als Datenbanken.<sup>13</sup> Zudem erlaubt der aktuelle Aufbau nur eine sehr simple, sequenzielle Organisation. Eine weitgehende Umstrukturierung wäre nötig um eine komplexere Organisation zu ermöglichen. Zudem werden in der aktuellen Version Lücken im sequenziellen Aufbau als Ende der Auflistung angesehen, was das Löschen von Dateien die nicht die letzte Datei in der Auflistung sind, gefährlich macht. Dieses Problem ist lösbar, wurde allerdings nicht rechtzeitig genug entdeckt um eine Lösung rechtzeitig zum Ende der Arbeit zu implementieren. Neben diesen Aspekten führt das Löschen oder Bearbeiten von Dateien aktuell zu einer vollständigen Kopie der alten Datei. Bei den geringen Größen von Markdown Dateien und der aktuellen Menge von diesen Dateien ist dies kein großes Problem. Wenn aber viele Änderungen vorgenommen wurden, wird die Menge an Dateien die gespeichert sind, den Überblick über Änderungen, und somit das rückgängig machen eben dieser Änderungen, erschweren, wodurch regelmäßige Wartung vermutlich erforderlich wird, um dem entgegenzuwirken. Zudem setzt sich bei mehreren gleichzeitigen Änderungen an einer Datei, diejenige durch, die als letztes bestätigt wird. Es werden keine Änderungen verworfen. Die überschriebenen Änderungen werden zwischengespeichert. Überschriebene und gelöschte Dateien werden in dem selben Ordner abgelegt. Eine Implementierung von Git würde dieses Problem lösen. Datenbanken hingegen haben von Haus aus eine Funktion um Änderungen rückgängig zu machen, was die Implementierung von externen Systemen überflüssig macht. Der einzige Nachteil von Datenbanken ist die Schwierigkeit die Texte aus der Datenbank zu extrahieren, ohne dass der Server aktiv ist.<sup>14</sup> Diese Einschränkung müsste entweder so akzeptiert werden, oder ein eigenes Programm geschrieben werden, dass die Texte aus der Datenbank extrahieren kann.

### 5.2 Beantwortung der wissenschaftlichen Frage

Die dateibasierte Speicherlösung die hier umgesetzt wurde, erfüllt alle gestellten Anforderungen. Sie hat allerdings einige Nachteile, die besonders bei größeren Projekten offensichtlich werden. Größere Projekte, wie das Django-Framework, haben Dokumentationen mit über einhundert Seiten. Bei einem solchen Volumen an Daten verlieren die Vorteile dieses Ansatzes, hauptsächlich die Möglichkeit die Daten direkt aus über das Dateisystem zu extrahieren und zu modifizieren, an Wichtigkeit. Mit so vielen Seiten die Dateistruktur schnell unübersichtlich wird. Im Gegenzug gewinnen die Vorteile von Datenbankbasierten Ansätzen an Wichtigkeit.

---

<sup>13</sup>Vgl. [Gar08]

<sup>14</sup>Vgl. [Dja25]

Große Projekte haben häufig viele Beitragende, was die Wahrscheinlichkeit erhöht, dass mehrere Personen die selbe Seite zur selben Zeit bearbeiten wollen. Dies wird von der Datenbank nativ unterstützt, während hier ein Modul, zum Beispiel git, dafür eingebaut werden muss. Für kleinere Projekte die von wenigen oder nur einer einzigen Person betrieben werden, ist der Dateibasierte Ansatz eine solide und nützliche Variante.

### 5.3 Ausblick

Diese Applikation hat in ihrer aktuellen Form noch mehrere Verbesserungsmöglichkeiten, aber es gibt auch andere Ansätze, um dieselbe Funktionsweise zu erhalten, die diese Applikation hat. Erstens ist es möglich die Datenbank, aus dem Prozess des Uploads und Löschsens von Dateien mittels der Administrationsseite, zu entfernen. Dies ist leider zu spät im Projekt erkannt worden um es noch innerhalb der vorgegebenen Zeit umsetzen zu können. In der Theorie ist das Erreichen des Ziels möglich, indem die Vorlage der Administrationsseite überschrieben wird, um eine Schaltfläche für die Weiterleitung auf eine Upload Seite hinzuzufügen. Diese Upload Seite würde die Basis Vorlage für alle Administrationsseiten erweitern und somit den Stil der Administrationsseite nicht brechen. Wie exakt diese Seite funktionieren würde ist noch nicht ausgearbeitet, aber mithilfe der „Forms“ von Django ist dies auf jeden Fall möglich, da die Administrationsseite dies ermöglicht, wenn ein Datenbankobjekt ein entsprechendes Feld hat. Zweitens ist eine Erweiterung der Ordnungsstruktur umsetzbar, die es ermöglichen würde komplexere Zusammenhänge zwischen den einzelnen Dateien darzustellen. Dafür ist eine Auflösung der festgelegten Pfadstruktur notwendig. Diese könnte durch eine Datei mit verschiedenen möglichen Strukturen ersetzt werden, oder an einer Festen Bezeichnungsstruktur für Dateien Festhalten und stattdessen Unterordner nutzen um die Dateien besser zu strukturieren. Drittens wäre ein Hybridapplikation, die sowohl eine Datenbank, als auch das Dateisystem, nutzt, möglich. Die Datenbank würde die Standorte und genauen Pfade der Dateien enthalten, während die Texte selbst in den Dateien gespeichert werden. Dafür hat Django ein eingebautes Datenfeld, dass nach diesem Prinzip fungiert und dafür genutzt werden könnte. Diese Variante erlaubt es weiterhin die Daten zu extrahieren, falls der Server zusammenbricht und nur noch über das Dateisystem auf die Daten zugegriffen werden kann. Es erlaubt nativ aber nicht die automatische Aufnahme von Dateien in die Datenbank. Dafür müsste eine extra Funktion eingebaut werden, dass die Dateien scannt, und unbekannte Dateien in die Datenbank einfügt. Diese müssten dann von einem Administrator händisch eingeordnet werden oder mit einer Struktur im Dateinamen automatisch zugeordnet werden. Viertens ist es eine Option den gesamten Text direkt in der Datenbank zu speichern. Diese Option erlaubt es nicht mehr die Texte zu extrahieren, wenn die Seite nicht aktiv ist und das Hinzufügen von Texten, indem eine Datei in den entsprechenden Ordner gelegt wird, trifft auf dieselben Schwierigkeiten wie bei der dritten Option. Hinzu kommt, dass diese Option auch auf der Administrationsseite keine Dateien mehr akzeptieren würde, wie die dritte Option es könnte, wenn dafür keine gesonderte Funktion gebaut wird. Diese Funktion müsste die hochgeladenen Dateien öffnen, den Text extrahieren und in das entsprechende Feld eintragen. Dies ist technisch möglich, allerdings ist wieder eine Änderung der Vorlage für die Administrationsseite notwendig.



Django ist am ehesten auf die Optionen drei und vier ausgelegt, auch wenn die anderen Optionen nicht unmöglich sind. Abschließend wäre es durchaus sinnvoll die erste, dritte und vierte Option eine Stresstest zu unterziehen um einen aussagekräftigen Vergleich der verschiedenen Methoden zu haben.

# Literaturverzeichnis

## Literatur

- [CAD+20] Chen, Songtao, Ahmmed, Shahed, Deming, Chunhua u. a.: „Django Web Development Framework: Powering the Modern Web“, American Journal of Trade and Policy Vol. 7 Issue 3/2020 ISSN 2313-4755 o.O., o.V., 2020
- [Con20] Matt Cone (Hrsg.): „Markdown guide“, o.O., Matt Cone, 2020
- [Dja25] Django Software Foundation: „Django Website“, 2025, Abgerufen am 10.3.2025 von: URL: <https://docs.djangoproject.com/en/5.2/>
- [Gar08] Hector Garcia-Molina: „Database systems: the complete book“, o.O., Pearson Education India, 2008
- [Vin22] William S. Vincent: „Django for Professionals“, o.O., Still River Press, 2022

## Anlagenverzeichnis

Anhang 1: Filter für Überschrift .....	VII
Anhang 2: Filter zur Umwandlung des Textes .....	VII

## Anhang 1: Filter für Überschrift

```
1
2 from django import template
3 import markdown2
4 import os
5
6 register = template.Library()
7
8 @register.filter
9 def markdown_to_html(number):
10     file_path = os.getcwd() + f"/data/Buchseite/Kapitel_{number}.md"
11     with open(file_path, 'r') as file:
12         text = file.read()
13         file.close()
14     return markdown2.markdown(text, extras=['tag-friendly'])
15
```

## Anhang 2: Filter zur Umwandlung des Textes

```
1 from django import template
2 import os
3 register = template.Library()
4
5 @register.filter
6 def getfirstline(Kapitel):
7     line = " "
8     with open(os.getcwd()+f"/data/Buchseite/Kapitel_{Kapitel}.md") as file:
9         if (os.stat(os.getcwd()+
10 f"/data/Buchseite/Kapitel_{Kapitel}.md").st_size == 0):
11             return 'empty file'
12         while not(line.upper().isupper()):
13             line = file.readline().strip('\n')
14             line = line[2:]
15     return line
16
```

## Anhang 3: Django View

```
1 @login_required(login_url="/admin/login/")
2 def edit(request, chapter_id=1):
3     template = "reading/edit.html"
4     rep_cour = os.getcwd()
5     file_path = rep_cour + f"/data/Buchseite/Kapitel_{chapter_id}.md"
6     if request.method == "POST":
7         # create a form instance and populate it with data from the request:
8         form = EditForm(request.POST)
9         # check whether it's valid:
10        if form.is_valid():
11            # process the data in form.cleaned_data as required
12            text = form.cleaned_data["text"]
13
14            if os.path.isfile(file_path):
15                a = time.time()
16                shutil.move(file_path, f"data/deleted/Kapitel_" + str(chapter_id) +
17                            "__" + str(a) + ".md")
18            with open(file_path, "w") as file:
19                file.write(text)
20            # redirect to a new URL:
21            return HttpResponseRedirect("/r/" + str(chapter_id))
22        else:
23            return HttpResponse("error")
24
25    # if a GET (or any other method) we'll create a blank form
26    else:
27        inhalt = ""
28        if os.path.isfile(file_path):
29            with open(file_path, "r") as f:
30                inhalt = f.read()
31        form = EditForm(initial={'text': inhalt})
32        a = 1
33        b = []
34        chapter_count = rep_cour + f"/data/Buchseite/Kapitel_{a}.md"
35        while os.path.isfile(chapter_count):
36            b += [a]
37            a += 1
38            chapter_count = rep_cour + f"/data/Buchseite/Kapitel_{a}.md"
39        context = {
40            "chapter": chapter_id,
41            "chapterlist": b,
42            "form": form
43        }
44
45    return render(request, template, context)
46
```

## Anhang 4: HTML Darstellung des „forms“

```
1 <div class="container">
2     <form action="/edit/{{chapter}}" class = "form" method="post">
3         {% csrf_token %}
4         {{ form }}
5         <input type="submit" value="Submit">
6     </form>
7 </div>
8
```

## Anhang 5: Vorgegebenes Textarea Widget

```
1 <textarea name="{ widget.name }" {% include "django/forms/widgets/attrs.html" %}>
2 {% if widget.value %}{ widget.value %}{% endif %}</textarea>
3
```

## Anhang 6: Modifiziertes Textarea Widget

```
1 <textarea name="{ widget.name }"
2 {% include "django/forms/widgets/attrs.html" %} class="inputPane">
3 {% if widget.value %}{ widget.value %}{% endif %}</textarea>
4
```

## Anhang 7: Django Template Renderer

```
1 FORM_RENDERER = 'django.forms.renderers.TemplatesSetting'
2
```

## Anhang 8: Filter Aufruf in HTML Vorlage

```
1 <div>
2     <div class ="Text">
3         {{ page|markdown_to_html|safe }}
4     </div>
5 </div>
6
```

## Anhang 9: Registrierung Klasse für Adminisrationsseite

```
1 from django.contrib import admin
2 from reading import file_migration
3 # Register your models here.
4
5 from reading.models import *
6
7
8 @admin.register(Kapitel)
9 class KapitelAdmin(admin.ModelAdmin):
10     def get_changelist(self, request, **kwargs):
11         x = 1
12         while file_migration.create_database_entry(x):
13             x += 1
14         everything = Kapitel.objects.all()
15         for one in everything:
16             if not os.path.isfile(one.file_path):
17                 one.delete()
18                 print('delete ' + one.name)
19         return super().get_changelist(request)
20
```

## Ehrenwörtliche Erklärung

Ich erkläre hiermit ehrenwörtlich,

1. dass ich meine Projektarbeit mit dem Thema:

### **Eignung eines Dateibasierten Speichersystems für eine Webapplikation für Dokumentationen**

ohne fremde Hilfe angefertigt habe,

2. dass ich die Übernahme wörtlicher Zitate aus der Literatur sowie die Verwendung der Gedanken anderer Autoren an den entsprechenden Stellen innerhalb der Arbeit gekennzeichnet habe und
3. dass ich meine Projektarbeit bei keiner anderen Prüfung vorgelegt habe.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

---

Ort, Datum

---

Unterschrift