HTWK

Hochschule für Technik,
Wirtschaft und Kultur Leipzig

Fakultät Informatik und Medien

Bachelorarbeit

zur Erlangung des Grades

Bachelor of Science

im Studiengang Informatik

der Hochschule für Technik, Wirtschaft und Kultur Leipzig

Digitale Umsetzung eines Prozesses in der Hochschulverwaltung am Beispiel der Lehrplanung für ein Semester

Vorgelegt von

Fadi Mkhalale

Matrikelnummer: 79860

Erstprüfer: Prof. Dr. rer. nat. Thomas Riechert
Zweitprüfer: Tobias Höppner, M.Sc.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Digitalisierung ist für Hochschulen inzwischen ein zentrales Handlungsfeld. PwC beschreibt die Corona-Pandemie als Katalysator für den digitalen Wandel an Universitäten. die Studie spricht in diesem Zusammenhang von einem „zwangsweise neuen Schwung“. Zugleich fehlt in vielen Fällen ein einheitlicher Governance-Rahmen, der den digitalen Wandel steuert.[1]. Ein Bericht des HIS-Instituts für Hochschulentwicklung prognostiziert außerdem, dass nutzerorientierte digitale Angebote langfristig das entscheidende Differenzierungsmerkmal im Wettbewerb um Studierende und Forschende sein werden.[2]

Zur Fundierung stützt sich diese Arbeit auf Erkenntnisse der PwC-Studie Die Digitalisierung an den Universitäten steuern (PricewaterhouseCoopers, Januar 2021). Die Studie beleuchtet die Perspektive der Rektorate/Präsidien der 31 größten deutschen Universitäten (jeweils mehr als 25.000 Studierende). Sie wurde von Peter Detemple, Dr. Florian Kaufmann, Dr. Verena Holl, Dr. Christian Marettek und Jens Mattmüller verfasst, umfasst 40 Seiten und kombiniert eine systematische Auswertung der Organisationsstrukturen mit rund 20 explorativen Interviews, so liefert sie sowohl qualitative Einsichten als auch quantitative Status-quo-Angaben zur Umsetzung der Hochschuldigitalisierung.

Insbesondere große Universitäten sind organisatorisch komplex, viele Fachbereiche und unterschiedliche Verwaltungskulturen erschweren koordinierte Digitalisierungsmaßnahmen.[1] In der Praxis entstehen oft Bruchstellen: Formulare sind zwar elektronisch ausfüllbar, der weitere Ablauf erfolgt aber häufig über Ausdrucke und manuelle Schritte.[1] Teilweise digitalisierte Abläufe genügen nicht. Nur vollständig durchgängige End-to-End-Prozesse ermöglichen die gewünschten Effizienzsteigerungen. Die Studie liefert zahlen zur Einordnung: So geben rund 66 Prozent der Universitäten Prüfungs- und Notenbescheide elektronisch aus, Bewerbungen werden in etwa 55,8 Prozent der Fälle online bearbeitet. Vollständig digital abgewickelte Immatrikulationen finden sich bei rund 41,9 Prozent, Urlaubsanträge bei etwa 36,4 Prozent.[2]

Der Wissenschaftsrat betont ebenfalls den Handlungsbedarf: Zwar besteht Bewusstsein für die Notwendigkeit der Digitalisierung, die Umsetzung bleibt aber hinter den Erwartungen zurück.[2] Zugleich eröffnen voll digitalisierte Lehr- und Verwaltungsprozesse erhebliche Potenziale, etwa schlankere Abläufe und mehr Servicequalität durch automatisierte Verfahren oder personalisierte Lehrangebote. Vor diesem Hintergrund analysiert die vorliegende Arbeit die Digitalisierung von Lehrplanungsprozessen und Formularabläufen an Hochschulen.

1.2 Problemstellung und Forschungsfragen

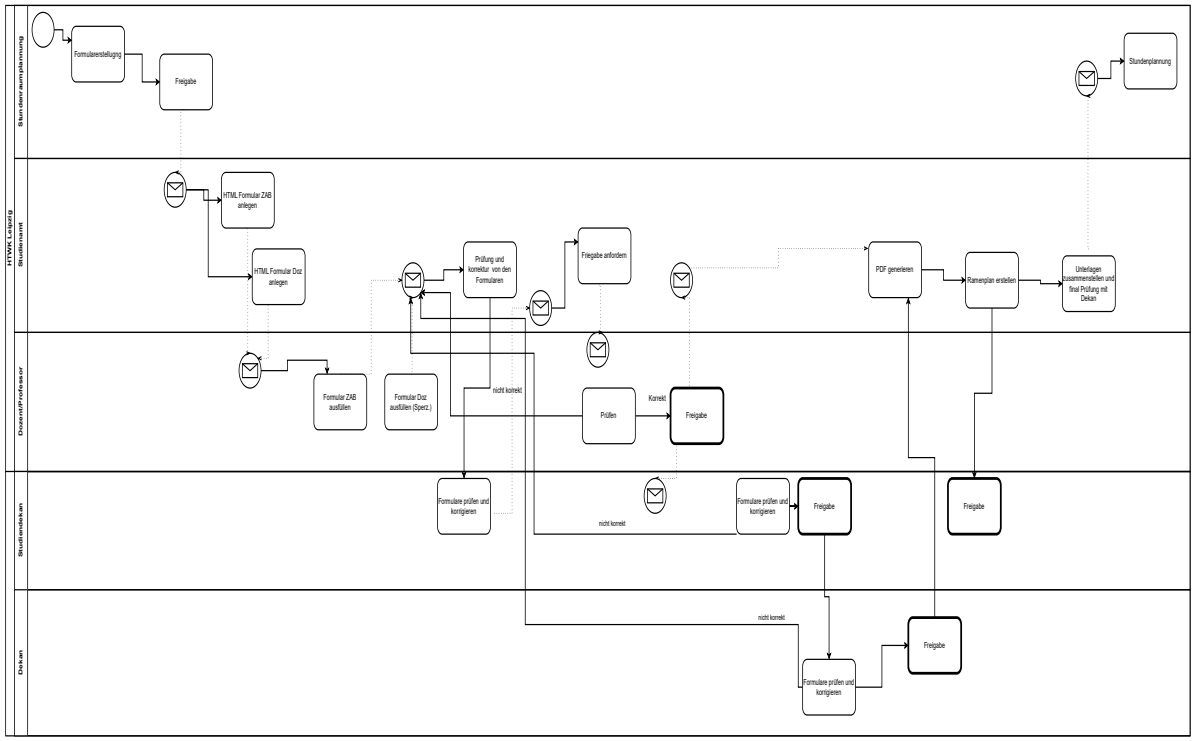

Abbildung 1.1: Ablauf des Prozesses vor dem Digitalisierung

In der Hochschulverwaltung erfolgt die Durchführung vieler Prozesse nach wie vor auf Papierbasis, einschließlich der Erfassung und Bearbeitung von Daten zu Lehrveranstaltungen. Für jeden Professor*in bzw. Dozent*in müssen Formulare ausgefüllt und an das Studienamt geschickt werden, um dort überprüft zu werden. Enthält ein Formular fehlerhafte oder unvollständige Angaben, wird es an die ursprüngliche einreichende Person mit einem Hinweis auf die entsprechenden Korrekturen zurückgesendet. In der Folge muss das gesamte Formular in der Regel neu erstellt und erneut eingereicht werden (Vgl. Abbildung 1.1). Dieser Ablauf erweist sich insbesondere bei einer größeren Anzahl an Formularen als ineffizient, fehleranfällig und zeitintensiv,sowohl für die einreichenden Personen als auch für die Mitarbeitenden des Studienamts.

Zu diesen Formularen zählen insbesondere das Zuarbeitsblatt sowie das Dozentenblatt, beides gehört zu den aufwendigsten Formularen an der Hochschule. Beide Formulare enthalten zahlreiche miteinander verknüpfte Angaben. Solche Abhängigkeiten erfordern umfassende Validierungs- und Plausibilitätsprüfungen, da unvollständige oder fehlerhafte Einträge regelmäßig Rückfragen nach sich ziehen. Im derzeitigen papierbasierten Ablauf führt dies dazu, dass beanstandete Formulare häufig vollständig neu ausgefüllt und erneut eingereicht werden müssen, dies erhöht sowohl den Zeitaufwand als auch die Fehleranfälligkeit. Vor diesem Hintergrund lassen sich Zuarbeitsblatt und Dozentenblatt als besonders zeitintensive und ressourcenbindende Verwaltungsinstrumente charakterisieren. Das **Zuarbeitsblatt** (das technische Detailblatt pro Lehrveranstaltung für das Studienamt) legt fest, wie eine einzelne Lehrveranstaltung im Stundenplan eingebettet ist (wann, wie oft, mit welchen Gruppen, wo). Das **Dozentenblatt** ist ein Formular, das von der Lehrkraft ausgefüllt wird, um ihre geplanten Lehrveranstaltungen, Zeiten und Präferenzen für das

Semester mitzuteilen. Es dient zugleich als persönliche Übersicht über das eigene Semesterpensum und als Planungsgrundlage für die Fakultät.

Aus dieser Problemstellung ergeben sich diese Forschungsfragen:

1. Ist es möglisch den Prozess den gesamten Prozess zu digitalisieren,von der Dateneingabe bis hin zur automatisierten Erstellung und Filterung von PDF-Dokumenten?

2. Inwiefern verändert die Digitalisierung die Prozessdauer, die Fehlerquote und die Nutzerzufriedenheit bei der Formularverarbeitung im Vergleich zum vorherigen Verfahren?

3. Welche wirtschaftlichen und ökologischen Effekte ergeben sich aus der Digitalisierung der Formularprozesse?

1.3 Aufbau der Arbeit

Nach der Darstellung der Problemstellung und der Formulierung der Forschungsfragen gibt dieses Kapitel einen Überblick über den Aufbau der Arbeit und zeigt, in welchen Kapiteln die zentralen Fragestellungen beantwortet werden. Kapitel 2 schildert die konzeptionellen Grundlagen und den Systemüberblick: es werden die Use Cases und Akteursrollen vorgestellt, die Systemarchitektur erläutert und die wesentlichen Systemabläufe inklusive Sequenzdiagrammen für Login, Datenprüfung, Freigabe und PDF-Generierung beschrieben zugleich stellt das Kapitel die konkrete Projektstruktur vor. Kapitel 3 begründet die Auswahl der eingesetzten Technologien, angefangen bei Webstandards über Node.js als Backend bis hin zur Entscheidung für JSON als zentrales Datenaustauschformat und den relevanten Bibliotheken, die in der Implementierung verwendet werden. Kapitel 4 setzt diese Grundlagen in den Stand der Technik: Verglichen werden bestehende Ansätze zur Formularverarbeitung und PDF-Erstellung, ihre Stärken und Schwächen sowie die Implikationen für Sicherheit und Datenschutz. Kapitel 5 dokumentiert die praktische Umsetzung der Anwendung, es beschreibt die Aufteilung der Aufgaben innerhalb des Projekts, die Server- und Frontend-Struktur, das Mapping der Eingabedaten auf die Vorlagen, die automatische Befüllung und die Verwaltungsoberfläche für generierte PDFs. Dabei wird transparent gemacht, welche Funktionalitäten für Login, Formular-Generator mit JSON sowie den PDF-Manager implementiert wurden. Kapitel 6 zeigt wesentliche Codeausschnitte und erklärt technische Entscheidungen an konkreten Beispielen, um Nachvollziehbarkeit und Reproduzierbarkeit zu gewährleisten. Kapitel 7 liefert die Evaluation der Lösung: Methodik, Messgrößen, Ergebnisse zur Effizienz und Fehlerreduktion sowie eine kurze Kosten- und Ökobilanz. Kapitel 8 diskutiert mögliche Erweiterungen wie PKI-basierte Signaturen, persistente Speicherung und KI-gestützte Validierung, und Kapitel 9 fasst die Arbeit zusammen, beantwortet die Forschungsfragen abschließend, benennt Limitationen der Umsetzung und gibt konkrete Empfehlungen für den produktiven Betrieb.

1.4 Anforderungen

Vor Beginn der Digitalisierungsarbeit wurden mehrere strukturierte Gespräche mit Mitarbeitenden des Studienamts geführt. Dabei wurde der aktuelle Ablauf Schritt für Schritt erläutert, von der Einreichung über Prüf- und Freigabeschritte bis hin zu typischen Rückfragen und Archivierungswegen. 1.1 Gleichzeitig wurden Erwartungen und konkrete Anforderungen aufgenommen, etwa funktionale Vorgaben, Qualitäts- und Datenschutzaspekte sowie Bedingungen für Signaturen und Exportfunktionen. Die aus diesen Gesprächen gewonnenen Erkenntnisse bildeten die Grundlage für die nachfolgende Priorisierung der Anforderungen (Muss, Soll, Kann) und die Formulierung der Akzeptanzkriterien.

Vollständige Digitalisierung des Prozesses (Muss) Beschreibung: Der gesamte Bearbeitungsablauf der Formulare erfolgt rein digital.
Akzeptanzkriterien: Alle Schritte sind über die Web-Anwendung verfügbar.

Ausfüllen beider Dokumente (Muss) Beschreibung: Sowohl das *Dozentenblatt* als auch das *Zuarbeitsblatt* können vollständig digital ausgefüllt werden.
Akzeptanzkriterien: Alle Felder des Original-PDFs sind im digitalen Formular abbildbar, generiertes PDF stimmt in Struktur und Pflichtfeldern überein.

Zugriffskontrolle (Muss) Beschreibung: Aus Datenschutzgründen dürfen nicht alle Personen Daten anderer einsehen
Akzeptanzkriterien: Nur berechtigte Rollen sehen bzw. bearbeiten jeweilige Daten.

Ausfülldaten sind sichtbar und auflistbar. (soll) Beschreibung: Die Daten der Professor*innen/Dozent*innen werden in einer Liste angezeigt, um sie zu überprüfen und damit Dokumente zu befüllen.
Akzeptanzkriterien: Die Daten sind für die entsprechenden Rollen sichtbar, nutzbar und leicht auffindbar.

Prüf- und Freigabeworkflow (Muss) Beschreibung: Berechtigte Personen können Dokumente prüfen, Richtigkeit bestätigen und digital unterschreiben/bestätigen.

Filter- und Suchfunktionen (kann) Beschreibung: Alle PDF Dokumente sind filterbar und suchbar.
Akzeptanzkriterien: Mindestens die häufigsten Filterkombinationen sind performant und liefern korrekte Ergebnisse.

PDF-Generierung (Muss) Beschreibung: Aus den eingegebenen Daten kann ein PDF erzeugt werden, das den Originalformularen in Layout und Pflichtfeldern entspricht.
Akzeptanzkriterien: Generiertes PDF enthält alle Pflichtangaben in der richtigen Position, Vergleichstest mit Muster-PDF bestanden.

Kapitel 2

Konzeption und Systemüberblick

2.1 Anwendungsfälle

Das folgende Use-Case-Diagramm zeigt den digitalisierten Ablauf der Formularverarbeitung und macht transparent, welche Akteure welche Schritte ausführen dürfen. Es visualisiert die Rollen, die Hauptfunktionen des Systems und die notwendigen Voraussetzungen (z. B. Link-Zugriff oder Anmeldung).

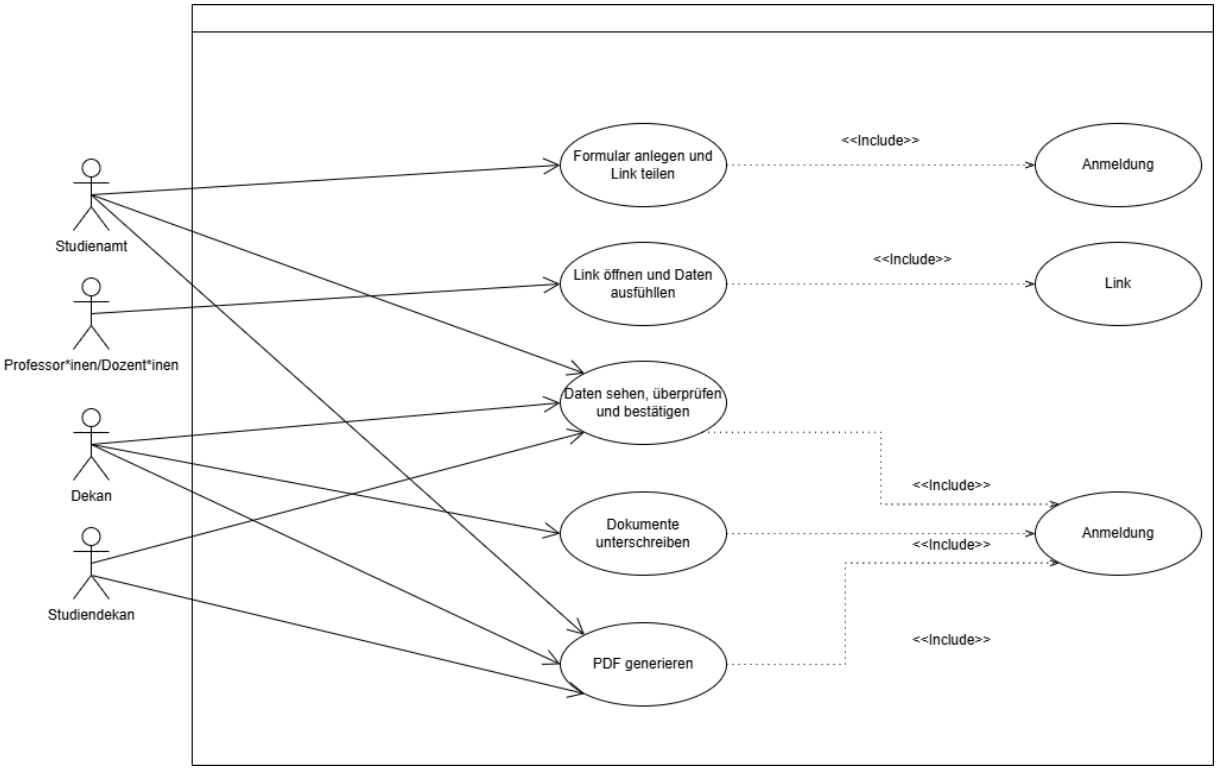

Abbildung 2.1: Use-Case-Diagramm - Akteure und zentrale Anwendungsfälle.

- Admins sind hier das Studienamt, der Dekan und der Studiendekan. Sie besitzen alle gemeinsame Rechte, denn alle können Daten sehen, überprüfen und bestätigen sowie PDFs aus den Daten für beide Dokumente generieren. Sie unterscheiden sich jedoch in folgenden Punkten: Nur das Studienamt hat Zugriff auf die andere Webanwendung und kann daher Formulare anlegen und die dazugehörigen Links an

die Professorinnen/Dozentinnen schicken. Der Dekan kann außerdem etwas, das die anderen nicht können: Nur er kann die Dokumente unterschreiben.

- Normale Anwender sind die Professorinnen/Dozentinnen. Sie haben nur Zugriff auf die Formulare, die für sie angelegt wurden, und können ausschließlich ihre eigenen Daten eingeben.

2.2 Systemarchitektur, Technischer Aufbau und Komponenten

Die Abbildung zeigt die strukturierte Aufteilung des Projekts in seine zentralen Komponenten und verdeutlicht, wie Dateien, Ordner und Funktionen innerhalb der Anwendung organisiert sind. Der Fokus liegt auf der logischen Gliederung des Systems. Die Darstellung macht nachvollziehbar, welche Module für welche Aufgaben zuständig sind und wie Frontend, Backend und Datenspeicher voneinander getrennt aufgebaut sind. Damit bildet das Diagramm eine Grundlage, um die Implementierung systematisch zu verstehen und die Funktionsbereiche klar voneinander abzugrenzen.

Ergänzend zur Abbildung umfasst die Projektstruktur folgende Elemente:

Docker Container-Umgebung

server.js (Backend) Zentrale Serverseite für Authentifizierung, Autorisierung, Datenoperationen und PDF-Verwaltung.

public (Frontend) Ordner mit allen clientseitigen Seiten und Skripten der Webanwendung. enthält:

- formulare.html, formulare.css, formulare.js zur Erstellung, Befüllung und Verarbeitung der Formulare, sowie die HTML-Grundstrukturen der beiden Formulartypen für die PDF-Erzeugung.

- login.html Oberfläche und Logik für die Anmeldung der Nutzerrollen.

- pdf-manage.html, pdf-manage.css, pdf-manage.js Seite zur Anzeige, Filterung und Verwaltung aller erzeugten PDF-Dokumente.

data Ordner zur Speicherung sämtlicher Formulardaten im JSON-Format. Enthält:

- dozenten ordner mit alle gespeicherten Daten der Professorinnen und Dozentinnen für das Dozentenblatt.

- zuarbeit ordner mit alle gespeicherten Daten der Zuarbeitsblätter.

config.json Konfigurationsdatei mit gehashten Zugangsdaten und systemweiten Einstellungen.

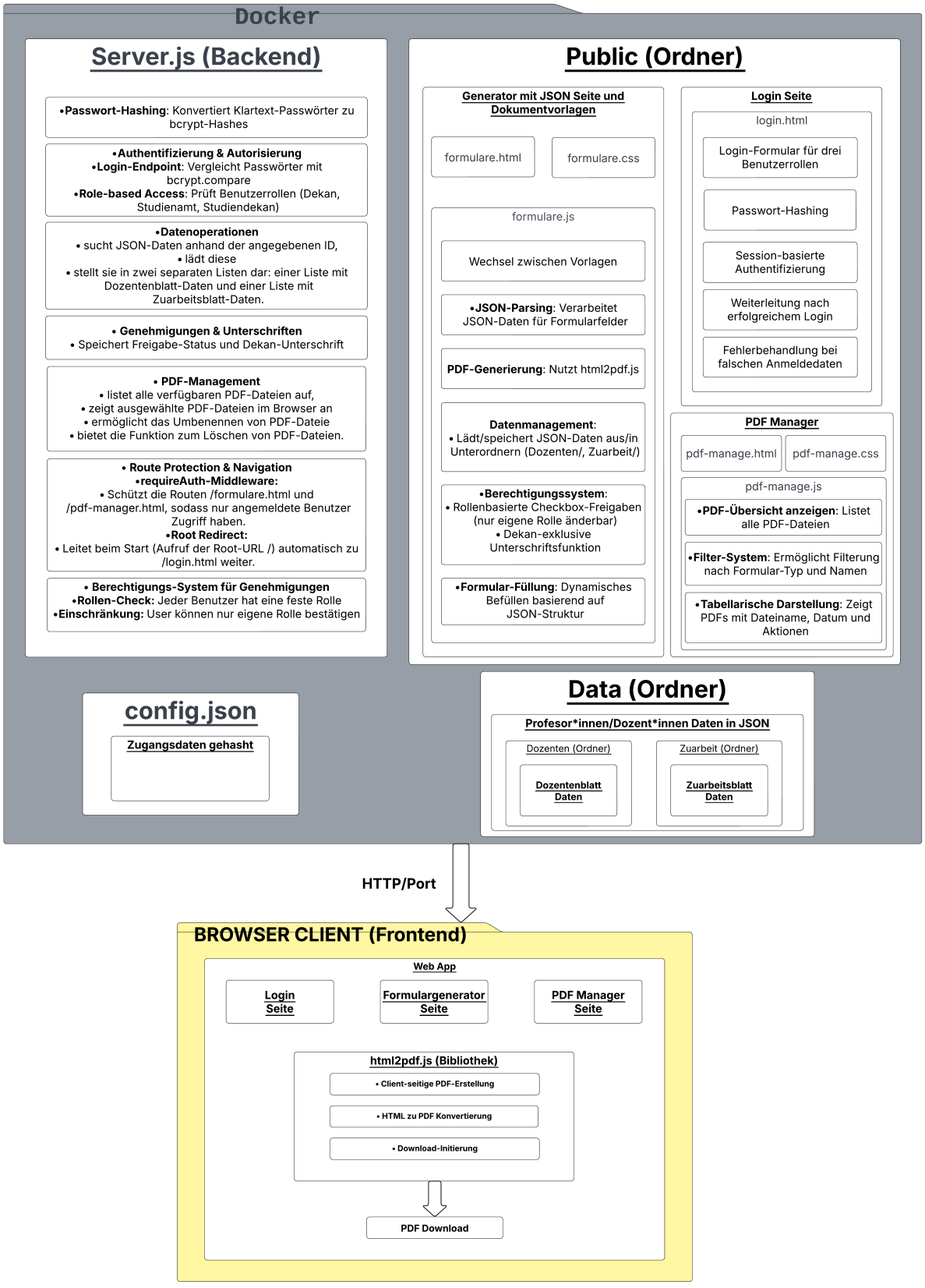

Abbildung 2.2: Systemarchitektur - Übersichtsdiagramm.

2.3 Systemabläufe und Prozessfluss

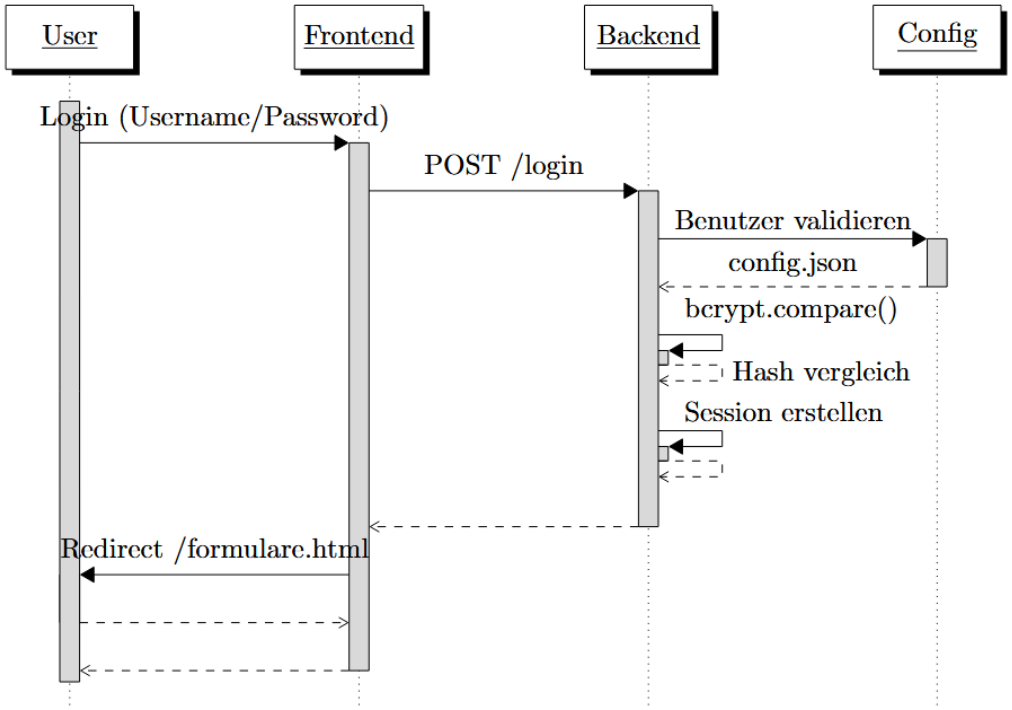

Abbildung 2.3: Sequenzdiagramm - Login mit bcrypt Hashing

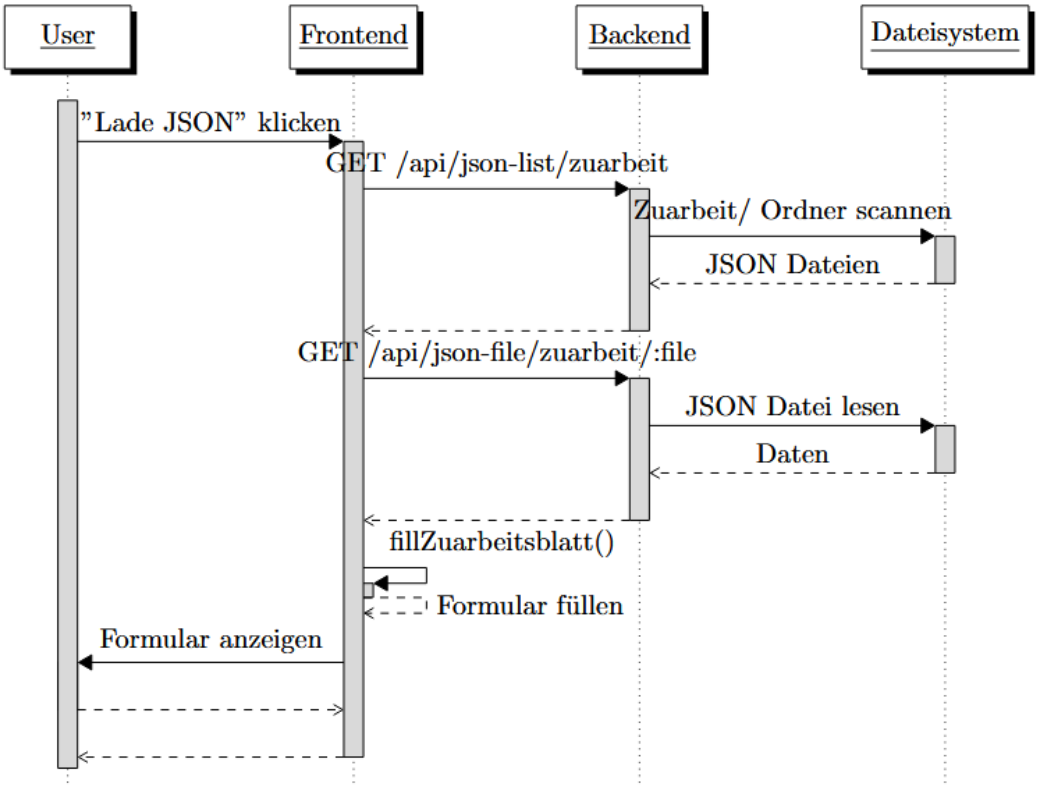

Abbildung 2.4: Sequenzdiagramm - JSON laden

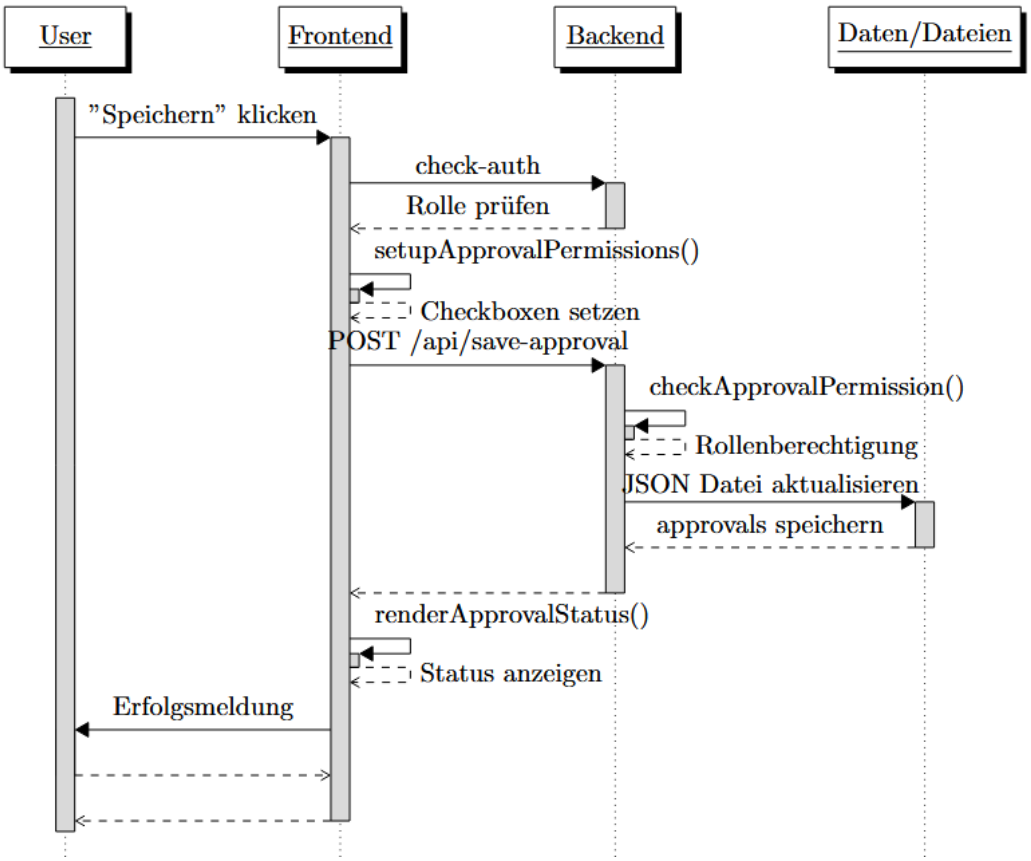

Abbildung 2.5: Sequenzdiagramm - Daten prüfen und freigeben

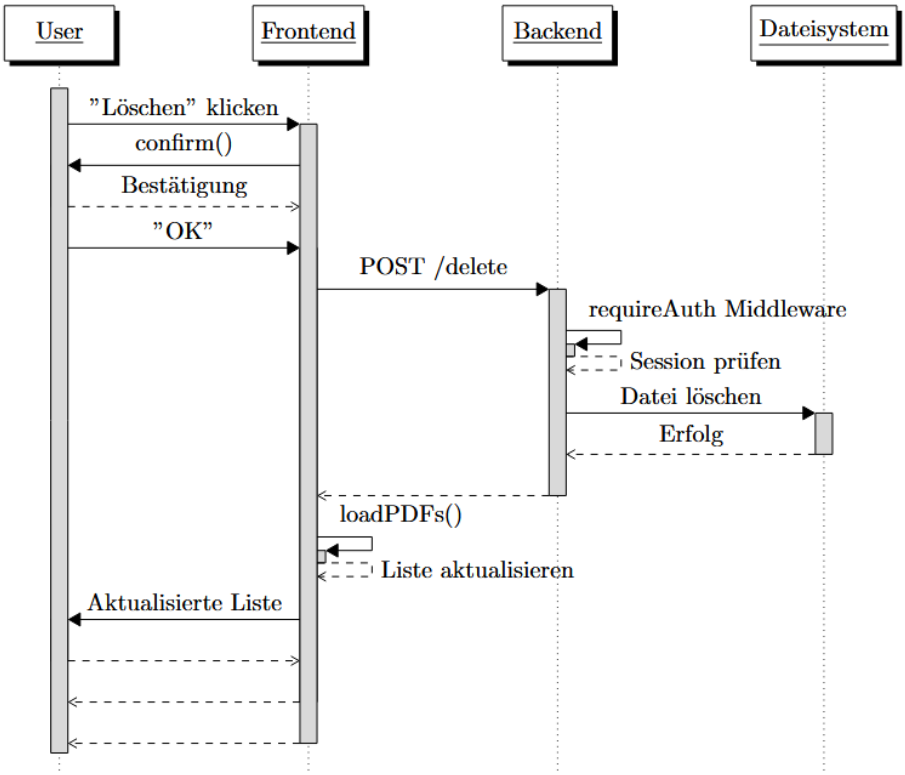

Abbildung 2.6: Sequenzdiagramm - PDF löschen

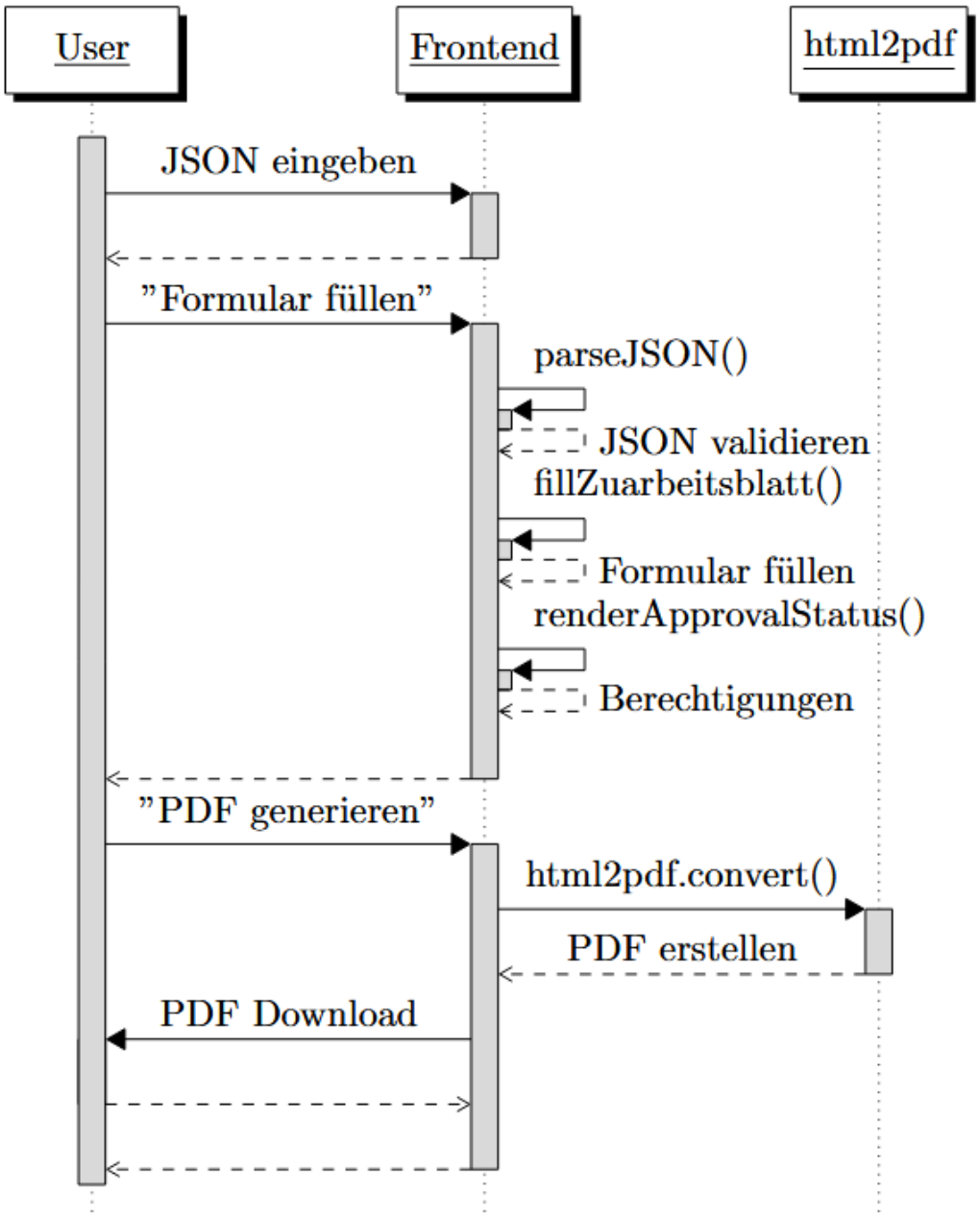

Abbildung 2.7: Sequenzdiagramm - PDF Generierung

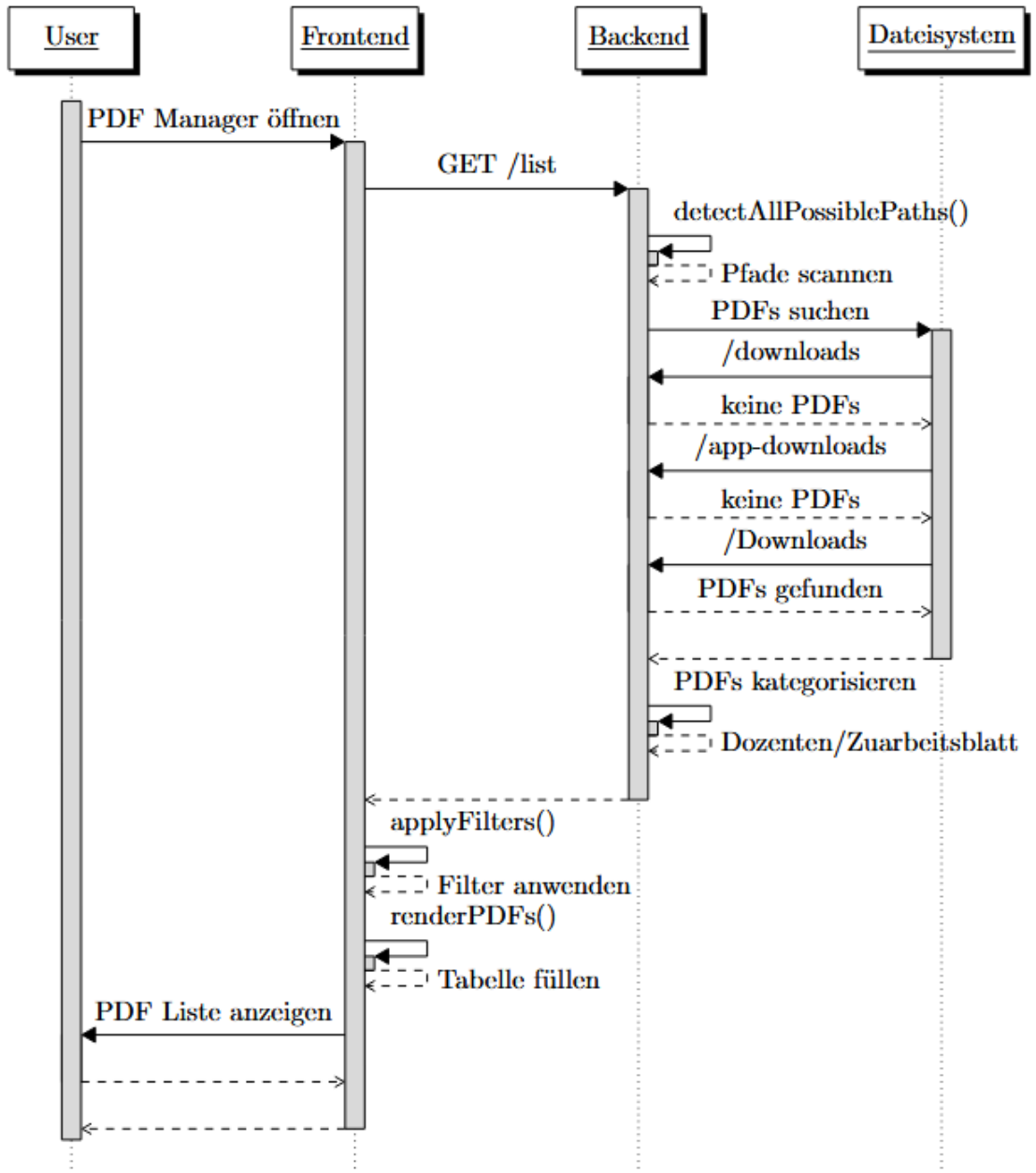

Abbildung 2.8: Sequenzdiagramm - PDF-Filter

Kapitel 3

Verwendete Technologien

3.1 Web Standarts

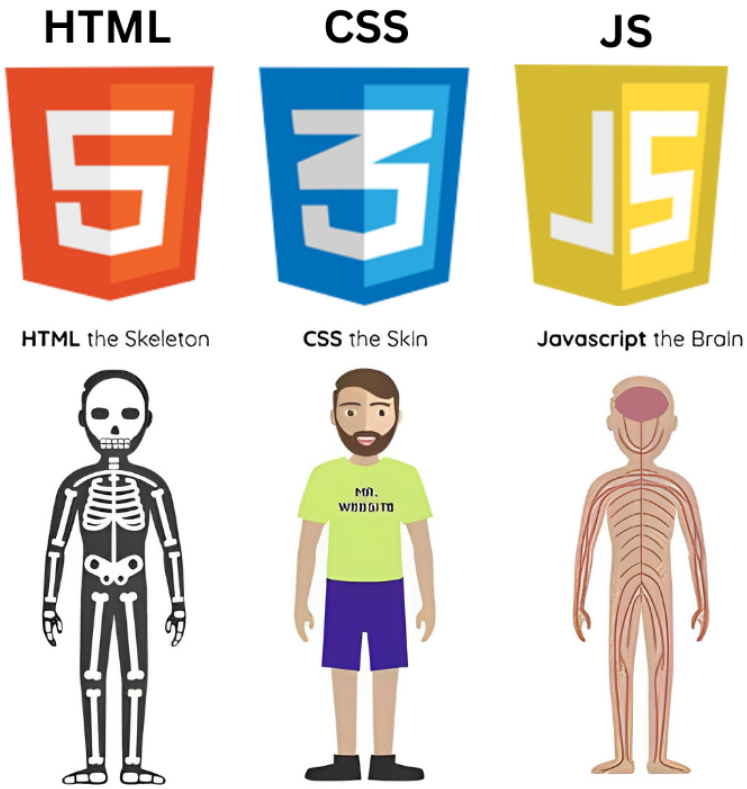

Abbildung 3.1: HTML als Skelett, CSS als Haut, JavaScript als Nervensystem

3.1.1 HTML

"Wurde verwendet, um die Grundstruktur der Webanwendung sowie die Formularvorlagen abzubilden, die als Basis für die spätere PDF-Erzeugung dienen."
HTML (HyperText Markup Language) ist die grundlegende Sprache, um Inhalte im Web zu strukturieren. Mit HTML werden Texte, Bilder, Links und andere Bestandteile einer Seite beschrieben, sodass Browser und Suchmaschinen die Anordnung und Bedeutung

erkennen können. Jedes HTML-Dokument beginnt mit `<!DOCTYPE html>` und hat als äußeres Element `<html>`. Darin stehen im `<head>` die Hintergrundinformationen und im `<body>` der sichtbare Teil der Seite.[3] HTML arbeitet mit sogenannten Tags und Elementen, die in einer Baumstruktur angeordnet sind. Zum Beispiel steht `<h1>` für eine Hauptüberschrift und `<p>` für einen Absatz. Mit HTML5 kamen zusätzliche Elemente wie `<header>`, `<nav>`, `<section>` oder `<footer>` dazu. Sie machen Seiten übersichtlicher und verbessern gleichzeitig die Zugänglichkeit sowie die Auffindbarkeit in Suchmaschinen.[4] Über Attribute wie `href`, `src` oder `alt` lassen sich Links, Bilder und Zusatzinfos genauer festlegen. HTML bildet die Grundlage für moderne Webseiten. Zusammen mit CSS und JavaScript entstehen daraus Anwendungen, die auch offline funktionieren können, Daten im Browser speichern oder Multimedia-Inhalte abspielen. Das Spektrum reicht von einfachen Startseiten bis hin zu komplexen Anwendungen, bei denen HTML die Basis und die Platzhalter für dynamische Inhalte liefert.[5]

3.1.2 CSS

"Wurde eingesetzt, um das Layout und das visuelle Erscheinungsbild der Anwendung und der Vorlagen zu gestalten, sodass diese möglichst nah an den offiziellen PDF-Formularen orientiert sind."
CSS (Cascading Style Sheets) ist die Stylesheet-Sprache, mit der das Aussehen und Layout von HTML-Dokumenten definiert wird. CSS trennt das Design von der Inhaltsstruktur und ermöglicht so eine konsistente Gestaltung über mehrere Seiten hinweg. Stile werden in Regeln formuliert, die aus Selektoren und Deklarationen (Eigenschaft: Wert) bestehen. Externe Stylesheets, die in der Regel als .css-Dateien gespeichert werden, können in beliebig vielen HTML-Dokumenten eingebunden werden, wodurch Änderungen zentral vorgenommen werden können.[6]
Durch CSS lassen sich Schriftarten, Farben, Abstände, Ränder und Hintergrundgrafiken flexibel festlegen. Mit modernen Layout-Techniken wie Flexbox und Grid werden komplexe Raster- und Flex-Layouts realisierbar, ohne auf unsaubere Hacks zurückgreifen zu müssen. Zudem unterstützt CSS Media Queries, um responsive Designs für verschiedene Bildschirmgrößen zu erstellen, sodass Webseiten auf Smartphones, Tablets und Desktops optimal dargestellt werden.[7]
CSS optimiert die Zugänglichkeit, da visuelle Anpassungen möglich sind, ohne die semantische HTML-Struktur zu verändern. Animationen und Transitionen werden rein über CSS realisiert, um interaktive Effekte ohne JavaScript zu erzielen. Die Trennung von Struktur (HTML) und Stil (CSS) vereinfacht die Wartung und fördert wiederverwendbare Komponenten, wie man sie in CSS-Frameworks (z. B. Bootstrap) findet.[8]

3.1.3 JavaScript

"Wurde genutzt, um die Anwendungslogik im Browser umzusetzen, Formularinhalte zu verarbeiten, Eingaben zu validieren und die PDF-Erstellung zu steuern."
JavaScript ist eine interpretierte Skriptsprache, die ursprünglich zur Interaktivität in Webbrowsern entwickelt wurde. Heute ist JavaScript die universell unterstützte Programmiersprache im Browser und ermöglicht das dynamische Manipulieren des Document Object Model (DOM). Über Event-Handler (z. B. onclick, addEventListener) reagieren Skripte

auf Benutzerinteraktionen wie Klicks oder Tastatureingaben und verändern Inhalte in Echtzeit, ohne die Seite neu zu laden.[9]
Neben DOM-Manipulation bietet JavaScript moderne APIs für asynchrone Netzwerkaufrufe (Fetch, AJAX), lokale Datenspeicherung (LocalStorage, IndexedDB) und Multimedia (Canvas, Web Audio API). Mit Promises und async/await lassen sich komplexe Abläufe übersichtlich gestalten. JavaScript wird in Frontend-Frameworks wie React, Vue oder Angular eingesetzt, die komponentenbasierte Architekturen für umfangreiche Web-Apps bieten.[10]
JavaScript ist nicht auf den Browser beschränkt: Mit Node.js kann derselbe Code serverseitig ausgeführt werden, was Full-Stack-Entwicklung mit einer einzigen Sprache ermöglicht. Einsatzgebiete sind interaktive Benutzeroberflächen, Single-Page-Applications, Progressive Web Apps, Echtzeit-Kommunikation via WebSockets und serverseitige APIs. Auch in Bereichen wie Game-Development findet JavaScript Verwendung.[11]

3.2 Frameworks:Node.js

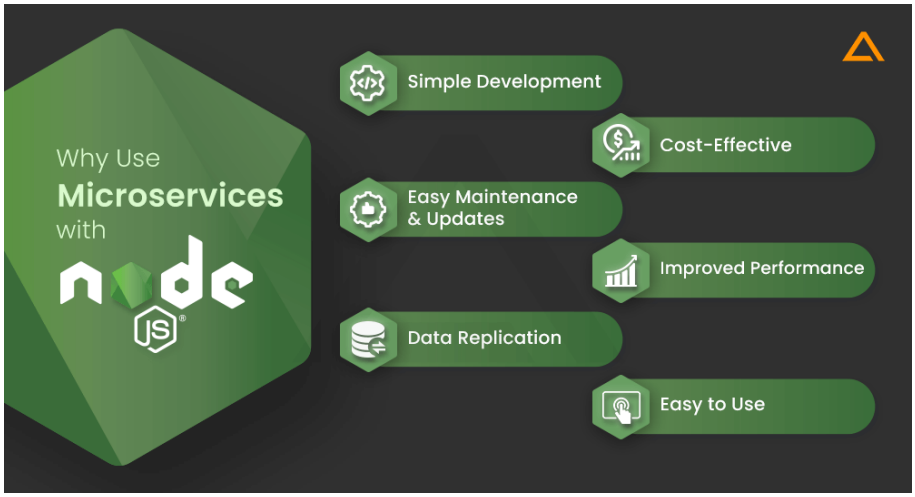

Abbildung 3.2: Vorteile Node.js

"Wurde genutzt, um die Anwendungslogik im Browser umzusetzen, Formularinhalte zu verarbeiten, Eingaben zu validieren und die PDF-Erstellung zu steuern."
Node.js ist eine offene, plattformübergreifende JavaScript-Laufzeitumgebung auf Basis der V8-Engine von Google Chrome. Anders als im Browser führt Node.js JavaScript auf dem Server aus und stellt Bibliotheken für Netzwerkoperationen und Dateisystemzugriff zur Verfügung. Kernmerkmal ist das Event-Loop-Modell: Eingehende I/O-Anfragen werden nicht blockierend behandelt, sondern asynchron verarbeitet, sodass mit einem einzigen Prozess tausende gleichzeitige Verbindungen möglich sind.[12]
Die Non-Blocking-Architektur von Node.js steigert die Effizienz bei I/O-intensiven Anwendungen wie Chats, Streaming und Echtzeit-Dashboards. Häufig kommen Frameworks wie Express zum Einsatz, um HTTP-Server schnell aufzusetzen und REST-APIs bereitzustellen. Der Node Package Manager (npm) bietet Zugriff auf über eine Million Pakete, die Funktionalitäten von Authentifizierung über Datenbankanbindung bis zu Logging abdecken.[13]

Node.js eignet sich gut für Microservices-Architekturen, da Anwendungen in unabhängige, modular entwickelte Dienste aufgeteilt werden können.[14] Zahlreiche Praxisbeispiele belegen die Leistungsfähigkeit von Node.js in produktiven Umgebungen: So berichtet PayPal von einer Reduktion der durchschnittlichen Antwortzeit um etwa 35[15]

3.3 Datenformat:JSON

Abbildung 3.3: JSON

"Wurde als Austausch- und Speicherformat für strukturierte Formulardaten eingesetzt."

Definition

JSON (JavaScript Object Notation) ist ein kompaktes, textbasiertes Datenformat zur Repräsentation strukturierter Informationen. Es wurde mit dem Ziel entworfen, eine sowohl für Menschen gut lesbare als auch für Maschinen effizient verarbeitbare Notation bereitzustellen. Aufgrund seiner geringen Syntaxkomplexität und der breiten Unterstützung in verschiedenen Programmiersprachen hat sich JSON als etabliertes Austauschformat für Web- und Mobilanwendungen durchgesetzt. [16]

JSON wird nicht nur zur Datenübertragung zwischen Client und Server verwendet, sondern dient häufig auch als leicht editierbares Konfigurations- oder Persistenzformat. Die einfache Textstruktur erleichtert das Debugging und die Wartung von Datenbeständen, weshalb JSON in vielen Entwicklungsumgebungen als Standardformat akzeptiert ist. [17, 16]

Syntax und Struktur

Ein gültiges JSON-Dokument besteht insbesondere aus zwei strukturellen Grundelementen: *Objekten* und *Arrays*. Objekte sind ungeordnete Sammlungen von Schlüssel-Wert-Paaren, die in geschweiften Klammern `{}` eingeschlossen werden. Arrays repräsentieren geordnete Listen von Werten und werden in eckige Klammern `[]` gesetzt. Schlüssel innerhalb eines Objekts sind Strings und werden durch Doppelpunkte vom zugehörigen Wert getrennt, mehrere Eigenschaften werden durch Kommata abgegrenzt. [17]

Datentypen

JSON unterstützt eine begrenzte Menge grundlegender Datentypen, typischerweise werden darin verwendet:

- Zeichenketten (Strings), in doppelten Anführungszeichen,
- Zahlen (Integer und Gleitkommazahlen),
- boolesche Werte `true` und `false`,
- das Literal `null`,
- verschachtelte Objekte,
- Arrays, die beliebige der genannten Typen enthalten können.

Der Standard erlaubt keine Kommentare innerhalb eines JSON-Dokuments, Verstöße gegen die Syntax führen in der Regel zur Ablehnung durch Parser. [17]

Anwendungsbereiche

JSON ist vor allem als Austauschformat zwischen Servern und Web- oder Mobilclients weit verbreitet, typische Web-APIs liefern Antworten im JSON-Format, die im Client mit Funktionen wie `fetch().then(res => res.json())` weiterverarbeitet werden. [16] Ferner wird JSON häufig als Konfigurationsformat verwendet, da einfache Textdateien eine unkomplizierte Änderung von Einstellungen ermöglichen, ohne den Anwendungscode anzupassen. [16]

3.4 Relevante Bibliotheken

3.4.1 html2pdf.js

"Wurde verwendet, um aus den vorbereiteten HTML-Vorlagen automatisiert PDF-Dokumente zu erzeugen."
html2pdf.js ist eine JavaScript-Bibliothek, die HTML-Inhalte vollständig im Browser in PDF-Dateien umwandelt.[18] Sie läuft komplett clientseitig ohne Server-Komponente und basiert auf html2canvas zur Rasterisierung von DOM-Elementen sowie jsPDF zur Erzeugung des PDFs.[19] Die Umwandlung erfolgt in mehreren Schritten: Zuerst wird die ausgewählte HTML-Quelle in ein Canvas-Element gerendert, anschließend als Bild exportiert und schließlich in ein PDF-Dokument eingebettet. Die Bibliothek nutzt ein Promise-basiertes API, das eine verkettete Konfiguration und individuelle Zwischenschritte erlaubt.[18] Zu den wichtigsten Merkmalen gehören konfigurierbare Seitenränder, Bildtyp und -qualität, Skalierung beim Canvas-Rendering sowie Ausgabeformat und -orientierung über jsPDF-Optionen. Zusätzliche Funktionen wie automatische Seitenumbrüche per CSS-Klasse runden das Feature-Set ab.[18]

Vorteile von html2pdf.js sind die einfache Integration per CDN (Content Delivery Network) oder npm und die Möglichkeit, das PDF direkt im Browser anzubieten.[19] Einschränkungen zeigen sich bei sehr komplexen Layouts, da der Inhalt als Bild gerendert wird, was zu großen Dateigrößen und fehlender Textselektierbarkeit führen kann.(eigene Analyse basierend auf Dokumentation).

3.4.2 pdf2htmlEX

"Wurde nicht verwendet. Geplant war, die vom Studienamt bereitgestellten Vorlagen direkt zu HTML zu konvertieren, um das originale Layout 1:1 zu übernehmen und Entwicklungsaufwand zu reduzieren."
pdf2htmlEX ist ein Kommandozeilen-Werkzeug, das PDF-Dokumente in semantisch reichhaltiges HTML überführt. Dabei bleiben Text, Schriftarten, Formatierungen und sogar mathematische Formeln weitestgehend erhalten. Die Ausgabe kann als einzelne HTML-Datei oder als dynamisch nachladende Seitenansicht erfolgen.[20]
Unter der Haube verwendet pdf2htmlEX moderne Web-Technologien wie HTML5, CSS3 und JavaScript, um PDF-Elemente pixelgenau zu platzieren. Intern greift es auf die Rendering-Engine von Poppler zurück und nutzt Fontforge, um eingebettete Fonts in Web-kompatible Formate zu konvertieren.[21]
Hervorzuheben ist die native Erkennbarkeit von Text, wodurch Suchfunktion und Textauswahl direkt im Browser möglich sind. Zusätzlich unterstützt das Tool Lesezeichen, Hyperlinks, SVG-Hintergründe und Typ-3-Schriften.[21] Die generierten Dateien sind oft sogar kleiner als das Original-PDF und eignen sich gut für die Web-Veröffentlichung.[20] Typische Vorteile liegen in der hohen Genauigkeit der Wiedergabe, der Textbarkeit und der responsiven Darstellung auf mobilen Geräten. Nachteilig sind der relativ aufwendige Build-Prozess, Abhängigkeiten zu externen Tools und eine in den letzten Jahren etwas unregelmäßige Wartung des Projekts.[21]

3.5 Containerverwaltung:Podman

Abbildung 3.4: Podman

"Wurde verwendet, um die Anwendung containerisiert und reproduzierbar auszuführen."
Podman ist ein quelloffenes Werkzeug zur Erzeugung, Ausführung und Verwaltung von Containern, Images, Volumes und Pods, es basiert auf der Bibliothek libpod und richtet sich an Entwickler und Betriebspersonal, die containerisierte Anwendungen handhaben wollen.[22] Im Gegensatz zu daemon-basierten Container-Engines läuft Podman daemonlos und ermöglicht den Betrieb von Containern im Benutzerkontext, was den Einsatz ohne Root-Rechte erleichtert und die Angriffsfläche verringert.[23] Podman bietet eine mit Docker vergleichbare Kommandozeile und integriert sich in ein Ökosystem für Image-Build und -Transfer, wodurch Migrationen von Docker-Workflows sowie die Erzeugung reproduzierbarer Images vereinfacht werden.[22]

3.6 Authentifizierungssystem

Ein Login-System ist die Komponente einer Webanwendung, die die Identifikation und Authentifizierung von Nutzerinnen und Nutzern übernimmt. Dabei gleichen Nutzerinnen und Nutzer ihre Anmeldedaten, in der Regel Benutzername und Passwort oder alternativ ein einmalig generierter Link ("Magic Link") oder biometrische Merkmale, mit den beim Server hinterlegten Datensätzen ab. Erst nach einer erfolgreichen Übereinstimmung erhält der Anwender Zugriff auf geschützte Bereiche und personalisierte Funktionen der Webseite. [24]
Ein Login-System bietet mehrere entscheidende Vorteile:

- **Sicherheitssteigerung:** Durch eindeutige Identitätsprüfung und geeignete Verschlüsselung der Zugangsdaten wird die Angriffsfläche für Phishing-, Social-Engineering- und Brute-Force-Angriffe reduziert. [25]
- **Erweiterbarkeit um Zwei-Faktor-Authentifizierung (2FA):** 2FA bildet eine zusätzliche Schutzschicht und bietet Schutz, selbst wenn Passwörter kompromittiert sind. [25]
- **Reduzierung der Angriffsfläche durch kennwortlose Verfahren:** Verfahren wie Magic Links, Hardware-Token oder biometrische Authentifizierung vermeiden die dauerhafte Passwortspeicherung und können so Risiken verringern. [24]
- **Protokollierung und Session-Management:** Login-Systeme erlauben die Protokollierung sicherheitsrelevanter Ereignisse (z. B. fehlgeschlagene Anmeldeversuche) und sollten Session-Timeouts bzw. automatische Sitzungsbeendigungen implementieren, um gestohlene Sitzungen zu begrenzen. [26]

Kapitel 4

Stand der Technik

4.1 Einordnung

Für die automatisierte Erzeugung und das automatisierte Befüllen von Formular-PDFs existieren verschiedene, technisch unterschiedliche Ansätze. Relevante Verfahren für dieses Projekt lassen sich in drei Klassen einteilen: (1) rein clientseitige Erzeugung von PDFs im Browser durch Rasterisierung von HTML [18], (2) das serverseitige Rendering mittels automatisierter Browser-Instanzen.[27] und (3) die direkte Manipulation des PDF-Formats und die Nutzung von Formulardatenstandards.[28][29] Die Auswahl einer geeigneten Strategie hängt maßgeblich von Anforderungen ab.

4.2 Clientseitige HTML→PDF-Rasterisierung

Eine verbreitete, einfach zu integrierende Methode ist die reine Client-seitige Erzeugung von PDFs im Browser, beispielsweise mittels `html2pdf.js`, das intern auf `html2canvas` und `jsPDF` zurückgreift. [18].

die HTML-Darstellung wird im Browser in ein Bild überführt und dieses Bild in ein PDF eingebettet. Typische Implementierungen kombinieren Canvas-Rendering mit einer PDF-Bibliothek und laufen vollständig im Browser. Das erlaubt die direkte Erzeugung eines PDFs auf dem Endgerät ohne Server. Der wesentliche Nachteil dieses Ansatzes liegt in der häufig rasterbasierten Ausgabe: Inhalte werden als Bitmap gerendert, wodurch Text nicht mehr als semantisch auswählbarer Text vorliegt. [18]

4.3 Serverseitiges Rendering mit Headless-Browsern

Headless-Browser wie Puppeteer/Headless-Chrome steuern eine echte Browser-Engine auf einem Server und erzeugen daraus PDFs, dazu wird eine automatisierte Browserinstanz gestartet, die Seiten wie in einem normalen Browser rendert und als PDF exportiert. Diese Methode nutzt die volle Rendering-Fähigkeit moderner Browser, sodass Text- und Vektorinformationen erhalten bleiben und die resultierenden PDFs durchsuchbar und druckfähig sind. Der Betrieb von Browser-Binaries erzeugt jedoch zusätzlichen Verwaltungsaufwand und Ressourcenbedarf auf Server-Seite, Skalierung und sichere Isolation sind zu bedenken. [27]

4.4 Direkte PDF-Manipulation und Formularstandards

Für das automatische Befüllen bestehender PDF-Formulare ist die direkte Manipulation des PDF-Formats oft die robusteste Vorgehensweise. JavaScript-Bibliotheken wie `pdf-lib` bieten APIs zum Auslesen und Befüllen von AcroForm-Feldern sowie zum Einbetten von Schriftarten und Finalisieren von Formularen. [29]

Als Datenaustauschformate für Formulardaten haben sich FDF und XFDF etabliert, Adobe dokumentiert Support und typische Anwendungsfälle solcher Formdata-Formate für AcroForms. [28]

4.5 Vergleich der Ansätze

- **Clientseitige Rasterisierung (z. B. html2pdf.js):** Einfache Integration und kein Serverbetrieb notwendig, aber oft Rasterausgabe und fehlende Textsemantik. [18]
- **Headless-Browser (Puppeteer):** Liefert typischerweise qualitativ hochwertige, vektorbasierte PDFs mit erhaltenem Text, erfordert jedoch serverseitige Infrastruktur. [27]
- **PDF-Manipulation / AcroForms (pdf-lib, FDF/XFDF):** Besonders robust für Formular-Workflows, vorteilhaft wenn AcroForm-Felder vorhanden sind oder erzeugt werden können. [29]

4.6 Relevanz für dieses Projekt

Im Rahmen dieses Projekts wurde experimentell die Kombination pdf2htmlEX → interaktive HTML-Bearbeitung → html2pdf.js untersucht. Während die mit `pdf2htmlEX` erzeugten HTML-Dateien optisch nahe am Original lagen, erwiesen sie sich aufgrund ihrer positionsbasierten Struktur und fehlenden semantischen Gruppierung als wenig zuverlässig für eine direkte, strukturierte JSON-Befüllung. Aus diesem Grund wurde ein rekonstruktiver Ansatz verfolgt: Die Vorlagen wurden manuell in semantisches HTML/CSS überführt, um stabile Feldzuordnungen und eine verlässliche Datenbindung sicherzustellen (vgl. Implementierungskapitel, eigene Arbeit).

Kapitel 5

Implementierung der Anwendung

Die Arbeit besteht aus zwei Projekten, die ich gemeinsam mit meinem Kommilitonen Nouralrahman Hussain umgesetzt habe. Er ist für die Erstellung der Formulare und den Dateneingang zuständig, ich bin für die Ausgabe verantwortlich. Meine Aufgaben umfassen das Einlesen und Filtern der Eingabedaten, die Auswahl der passenden Vorlage, das automatische Ausfüllen, die Freigabe durch die zuständigen Personen und die Generierung des finalen PDF-Dokuments.
Die Anwendung besteht aus mehreren Webseiten und einem Node.js-Server, der die Seiten verbindet, den Datenaustausch steuert und verschiedene Funktionalitäten bereitstellt. Wenn Professorinnen oder Dozenten ihre Angaben in das JSON-Formular eingeben, werden die ausgefüllten Daten als JSON auf dem Server gespeichert.

5.0.1 Login

Zu Beginn steht eine Login-Seite. Nutzer müssen sich anmelden, damit die Daten vertraulich bleiben und nur berechtigte Personen Zugriff auf die Formularinhalte haben. Nach erfolgreichem Login gelangt der Nutzer zur Hauptseite, dem Generator, auf dem die zentralen Funktionen zusammenlaufen. 5.2).

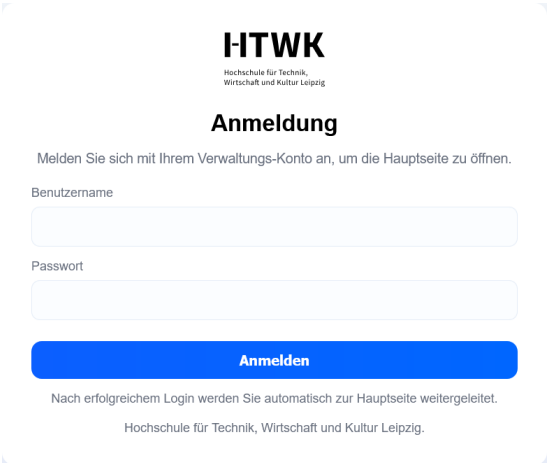

Abbildung 5.1: Login-Seite

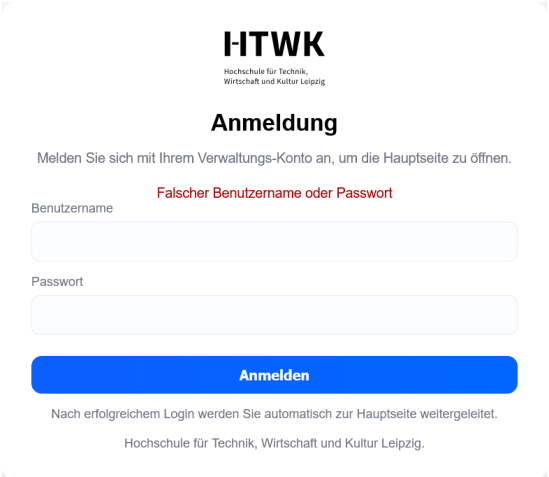

Abbildung 5.2: Login-Seite mit Fehlermeldung bei ungültigen Anmeldedaten

5.0.2 Generator mit JASON (Hauptseite)

Auf der Hauptseite wählt der Nutzer Semester, Jahr und die Planungswochen, diese Angaben werden unmittelbar in den Formularen berücksichtigt. Anschließend wird per Schaltfläche die gewünschte Formularvorlage ausgewählt, die in einer Vorschau dargestellt wird. Ein Eingabefeld erlaubt das Schreiben oder Bearbeiten von JSON-Daten. Beispiel-JSONs können mit einem Klick in das Eingabefeld geladen werden, um die Bedienung zu erleichtern. 5.3).

Abbildung 5.3: JSON Eingabe

Unter „Zuarbeitsdaten bzw. Dozentendaten“ lassen sich zuvor eingereichte Datensätze abrufen. Dies ist entweder über eine Liste aller gespeicherten Datensätze möglich oder durch direkte Eingabe einer Formular-ID. Mit der Schaltfläche „Auswahl in Eingabe übernehmen“ werden die ausgewählten Daten in das Eingabefeld übernommen. Über die Aktion „Formular aus JSON ausfüllen“ wird das Formular automatisch befüllt, die Vorschau zeigt anschließend das spätere Dokument exakt an. Für Dozentenblatt und Zuarbeitsblatt existieren separate Listen und Eingabefelder, damit die Daten sauber getrennt und leicht überschaubar bleiben. 5.4).

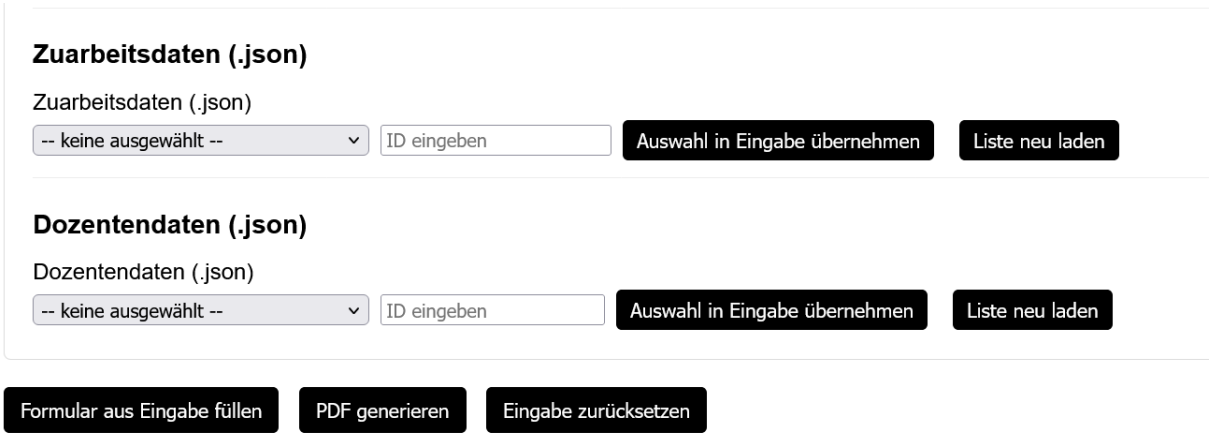

Abbildung 5.4: Gespeicherte Daten

Nach der Dateneingabe überprüft das Prüfungsamt die Angaben. Mit einem Häkchen im Feld „Studienamt“ bestätigt das Prüfungsamt die Richtigkeit und speichert die Änderung. Der Dekan erhält anschließend die Formular-ID, prüft die Daten ebenfalls und setzt bei Zustimmung ein weiteres Häkchen. Danach trägt er seine Unterschrift in das Eingabefeld ein, das nur für ihn sichtbar ist, und klickt auf „Unterschreiben“. Anschließend wird das Formular von ihm digital unterschrieben. In der Übersicht ist dokumentiert, wer die Daten bereits bestätigt hat. Wenn alle erforderlichen Stellen ihre Bestätigung gegeben haben, kann das Studienamt die PDF-Generierung auslösen und das fertige PDF herunterladen. 5.5).
Die Vorschau-Dokumente sind als HTML-Seiten umgesetzt, erstellt mit HTML, CSS und JavaScript. Die Umwandlung von HTML in PDF erfolgt mit einer Bibliothek, die die Seiten unter definierten Parametern in druckbare PDF-Dokumente konvertiert.

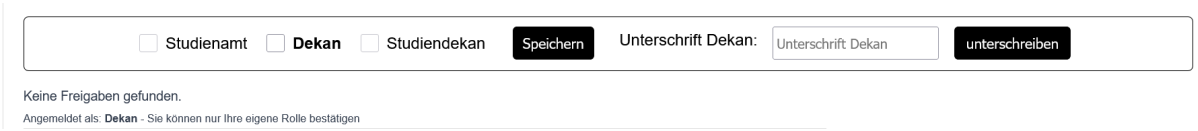

Abbildung 5.5: Freigabe und Unterschrieft

Die Vorlagen

Ich habe versucht, die für den Prozess erforderlichen Vorlagen-PDFs (Dozentenblatt und Zuarbeitsblatt) mithilfe der Bibliothek pdf2htmlEX direkt in HTML zu konvertieren, um diese anschließend interaktiv ausfüllbar zu machen und anschließend mit html2pdf.js wieder zu PDF zu rendern. Dieses Vorgehen hätte erhebliches Zeitersparnis versprochen und gleichzeitig das ursprüngliche Erscheinungsbild der Vorlagen bewahrt. Die resultierenden HTML-Dateien entsprachen zwar visuell weitgehend der Ausgangs-PDF, jedoch war der zugrunde liegende Quellcode derart unstrukturiert, dass eine anschließende programmatische Befüllung mit JSON-Daten nicht möglich war. Insbesondere ergab sich eine fehlende semantische Struktur und eine starke Abhängigkeit von positionsbasiertem Layout, wodurch die Zuordnung von Formularfeldern zu Datenpunkten nicht stabil implementierbar war. Aus diesem Grund entschied ich mich, die beiden Vorlagen manuell in HTML und CSS nachzubauen. Dieser rekonstruktive Ansatz ermöglichte die gewünschte Ausfüllbarkeit und eine zuverlässige Datenbindung, sodass das Ziel, die Vorlagen automatisiert mit JSON-Daten zu befüllen, letztlich erreicht werden konnte. 5.6). 5.7).

HTWK Leipzig
DS
ID:

Dozentenblatt

Wintersemester 2025/26
Präsenzplanung
(ggf. mit digitalen Anteilen)

Jeder Lehrende füllt bitte nur **ein** Dozentenblatt aus!

Titel _____ **Vorname** _____ **Name** _____

Fakultät/Bereich: ☐ FAS ☐ FB ☐ FDIT ☐ FIM ☐ FING ☐ FWW ☐ HSZ
☐ MNZ ☐ Gast ☐ HonProf

Arbeitszeit (Pflichtangabe, außer Gast) ◯ Vollzeit _____ ◯ Teilzeit _____

E-Mailadresse (Pflichtangabe) _____

Telefonisch erreichbar unter (optional) _____

Bitte geben Sie in der Tabelle Ihre Einsätze (auch Bedienfunktionen) an!

Durchzuführende Lehrveranstaltungen

Rows will be added dynamically

lfd. Nr.	Fak	Stg.	FS	Gruppen	Modulnr./ LE	Modulname (Kürzel)	Reale SWS (Präsenz)			Digital	Bemerkung
							V	S	P		

Abbildung 5.6: Vorlage - Dozentenblatt

HTWK Leipzig DS ID:	**Zuarbeit** **Stunden – Raumplanung**	Wintersemester 2025/26 Präsenzplanung (ggf. mit digitalen Anteilen)

Wintersemester 2025/26: Planungswochen 42–58

Fakultät **Studiengang** _____ **Fachsemester** ____ **Gruppen** _____

Nummer und Bezeichnung des Moduls

Nummer und Bezeichnung der Lehrveranstaltung/ des Teilmoduls

Gesamt SWS (bezogen auf einen Beispiel - Studenten):			
	Vorlesung	**Seminar**	**Praktikum**
davon SWS			
Raumanforderung			
Technikanforderung (Campus - Bereich)			

Hinweis: 1 SWS = 1 WS (Wochenstunde: 45 min) in jeder Woche des Semesters; geplant werden nur Lehreinheiten: 1 LE=2 WS=90 min

Sind keine weiteren Erläuterungen vorhanden, wird für den Charakter der LV Präsenz angenommen.

Weitere Möglichkeiten sind (siehe Hinweise auf Dozentenblatt):

digital asynchron (wird nur einmal im Semester im Plan in der Zeit von 07:00-07:30 Uhr eingesetzt),

digital asynchron mit zeitlicher Begrenzung (kann bei Bedarf auch mehrmals eingesetzt werden, wenn z.B. die KW definiert ist),

digital synchron.

Abbildung 5.7: Vorlage - Zuarbeitsblatt

5.0.3 PDF manager

Über einen separaten PDF-Manager sind alle generierten PDFs einsehbar, dort lassen sich Einträge filtern, löschen, anzeigen oder umbenennen.
Die Umsetzung erfolgt vollständig über das lokale Dateisystem des Rechners (z.B. Windows oder macOS), wobei das Backend mit Node.js folgende API-Endpunkte bereitstellt:

- GET /list: listet alle PDF-Dateien im Downloads-Ordner auf
- GET /view: zeigt eine ausgewählte PDF-Datei im Browser an
- POST /rename: benennt eine PDF-Datei um
- POST /delete: löscht eine PDF-Datei

5.8).

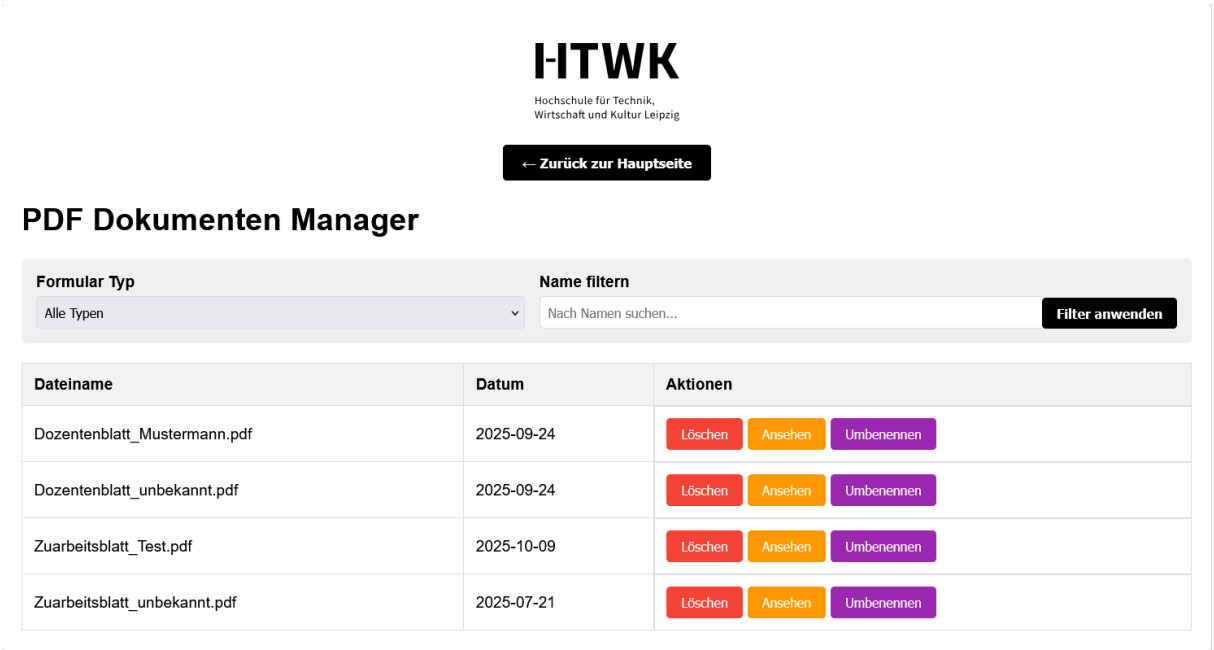

Abbildung 5.8: PDF Dokumenten Manager

5.1 Warum JSON? Vergleich von mit alternativen Datenformaten

Vorteile von JSON gegenüber den anderen Formaten

JSON zeichnet sich in formularbasierten Web-Workflows besonders durch die folgenden Aspekte aus:

- **Native Integration in Web-Technologie.** JSON ist Bestandteil des Web-Stacks, Browser und Node.js stellen native Funktionen zum Parsen und Serialisieren bereit. Dadurch reduziert sich Boilerplate beim Einlesen, Validieren und Übertragen von Formularwerten. [30]
- **Natürliche Abbildung verschachtelter Strukturen.** Formularstrukturen mit Gruppen, Listen und verschachtelten Feldern lassen sich direkt als Objekte und Arrays abbilden, ohne zusätzliche Modelltransformationen. [30]
- **Gutes Tooling und Validierungsoptionen.** Durch JSON Schema existiert ein standardisierter Weg, Struktur und Constraints formal zu beschreiben, was Validierung, Dokumentation und automatische Form-Generierung ermöglicht. [31]
- **Lesbarkeit und Debuggability.** JSON bleibt menschenlesbar, was Debugging und Review vereinfacht, binäre Formate sind dagegen schwerer direkt zu untersuchen. [30][39]
- **Upgrade-Pfad zu Semantik.** Falls später Linked Data nötig wird, erlaubt JSON-LD eine direkte Erweiterung von JSON, sodass bestehende JSON-Pipelines weiterverwendbar bleiben. [32]

Format	Kurze Definition	Hauptvorteile
JSON	Textbasiertes, hierarchisches Datenaustauschformat, an JavaScript-Objektnotation angelehnt. [30]	Nativ in Browsern, einfaches Parsen/-Serialisieren, natürliche Abbildung verschachtelter Objekte, großes Ökosystem (Validatoren, Bibliotheken), menschenlesbar. [30][31]
JSON-LD	JSON-Erweiterung zur Einbettung von Linked-Data-Informationen mittels `@context` und URIs, bleibt JSON-kompatibel. [32]	Ermöglicht Semantik und Interoperabilität ohne Syntaxwechsel zu RDF, einfach in bestehende JSON-Pipelines integrierbar. [32]
RDF / Turtle	RDF modelliert Aussagen als Tripel (Subjekt, Prädikat, Objekt), Turtle ist eine kompakte, lesbare Textsyntax dafür. [33][34]	Geeignet für Ontologien und vernetzte Daten, eindeutige Identifikation über URIs, mächtige Abfragen via SPARQL, hohe Interoperabilität im Semantik-Ökosystem. [33][34]
XML (inkl. XSD / XSLT)	Generische Markup-Sprache für strukturierte Dokumente, XSD für Schema-Validierung, XSLT für Transformationen. [35][36]	Ausgereifte Standards für Validierung (XSD) und Transformation (XSLT), geeignet für dokumentennahe Workflows und komplexe Layout-Transformationen. [35][36]
CSV	Zeilenbasiertes, kommasepariertes Format für tabellarische, flache Daten (RFC 4180). [37]	Extrem einfach und weit verbreitet, kompatibel mit Tabellenkalkulationen und vielen ETL-Werkzeugen,geringes Overhead für flache Tabellen. [37]
YAML	Menschenlesbare Auszeichnungssprache für strukturierte Daten, unterstützt Kommentare und Referenzen, ist ein Superset von JSON (YAML 1.2). [38]	Sehr lesbar für Konfigurationen, erlaubt Kommentare und Anker/Referenzen, oft genutzt für Konfigurationsdateien. [38]
JSON Schema	Vokabular zur formalen Beschreibung und Validierung von JSON-Dokumenten (Schemas, Constraints). [31]	Ermöglicht strikte Validierung, automatische Dokumentation und Form-Generierung, erhöht Datenqualität und Reproduzierbarkeit. [31]
Protocol Buffers	Binäres, schema-basiertes Serialisierungsformat von Google, erzeugt Typartefakte für viele Sprachen. [39]	Sehr kompakt und performan, strikte Typisierung über Schema, geeignet für latenz- und bandbreitenkritische Anwendungen. [39]

Tabelle 5.1: Kurzdefinitionen und Hauptvorteile gängiger Datenformate (quellenbezogen).

5.2 Zeitlicher Ablauf und Qualitätsvergleich der Formularprozesse vor und nach der Digitalisierung

Die angegebenen Zeiten wurden im Rahmen des Projekts selbst erhoben und dienen als Orientierung. Sie können je nach Bearbeiter stark schwanken, etwa durch unterschiedliche Erfahrung mit dem bisherigen Ablauf oder dem Umgang mit der Webanwendung.

Kriterium	**Vor der Digitalisierung**	**Nach der Digitalisierung**
Bearbeitungszeit pro Formular	ca. 30 Minuten	ca. 5 Minuten
Fehlerrate bei Eingaben	Hoch (z.B. fehlerhafte oder unvollständige Angaben)	Niedrig (Pflichtfelder, validierbar)
Übertragung	Manuell durch Abtippen	Automatisch durch JSON-Verarbeitung
Dateiformat	Papierdokument, eingescanntes PDF oder aus Word exportiert	Generiertes, strukturiertes PDF
Flexibilität bei Änderungen	Gering, neue Version muss manuell eingereicht werden	Hoch, Änderungen jederzeit im Webformular möglich
Rückfragen durch Verwaltung	Häufig wegen Unklarheiten	Selten durch den Zugang und Freigabe
Wiederauffindbarkeit	Papierarchiv oder manuelle Ablage	Automatische Benennung + Filter
PDF-Erzeugung	Manuell eingescannt oder per Word generiert	Direkt per Knopfdruck
Unterschriften	Händisch auf Papier nach dem Ausdrucken	Digital
Aufwand und Kosten	deutlich höher	deutlich geringer 7.1)

Tabelle 5.2: Vergleich der Prozesse vor und nach der Digitalisierung der Hochschulformulare

Kapitel 6

Wesentliche Codeausschnitte und Erläuterungen

6.1 server.js (Backend)

Initiales Hashing (startup)

```
async function ensureHashesInConfig(configPath) {
  if (!fs.existsSync(configPath)) throw new Error(`config.json nicht gefunden`);
  const cfg = await readJson(configPath);
  if (!Array.isArray(cfg.users)) cfg.users = [];

  const usersWithPlain = cfg.users.filter(u => u.password && !u.passwordHash);
  if (usersWithPlain.length === 0) return cfg;

  await backupConfigIfExists(configPath);
  for (const u of usersWithPlain) {
    try {
      const plain = String(u.password);
      const hash = await bcrypt.hash(plain, BCRYPT_COST);
      u.passwordHash = hash;
      delete u.password;
      console.log(`Gehashed: ${u.username}`);
    } catch (err) {
      console.error('Fehler beim Hashen von', u.username, err);
    }
  }
  await writeJsonSecure(configPath, cfg); // Mode 0o600
  return cfg;
}
```

Listing 6.1: ensureHashesInConfig: Klartext-Passwörter erkennen, hashen, Backup erstellen

Erklärung. Beim Start wird `config.json` auf Klartext-Passwörter geprüft. Falls vorhanden: Backup anlegen, Passwörter mit bcrypt (Cost = BCRYPT$_{C}OST)hashen$, $Klartext - Feldlschen, Dateimitrestriktiven Rechtenzurckschreiben$.

Session-Setup

```
let config = require(CONFIG_PATH);
app.use(session({
  secret: config.sessionSecret || 'geheimes_session_secret_ndern_in_produktion',
  resave: false,
  saveUninitialized: false,
  cookie: { secure: false, maxAge: 24 * 60 * 60 * 1000 } // secure:true in
    ↪Produktion
}));
```

Listing 6.2: Session-Middleware: secret aus config nutzen

Session-Secret kommt aus `config.json`. In Entwicklung `cookie.secure=false`; in Produktion `secure:true` zusammen mit HTTPS.

Login-Route (bcrypt-basiert)

```
app.post('/login', async (req, res) => {
  const { username, password } = req.body;
  if (!username || !password) return res.redirect('/login.html?error=1');

  const user = users.find(u => u.username === username);
  if (!user) { await new Promise(r => setTimeout(r,200)); return res.redirect('/
    ↪login.html?error=1'); }

  // Rckfall, falls config noch Klartext enthlt (nur temporr)
  if (user.password && !user.passwordHash) {
    if (user.password === password) { req.session.user = user; return res.
    ↪redirect('/formulare.html'); }
    else return res.redirect('/login.html?error=1');
  }

  try {
    const ok = await bcrypt.compare(String(password), String(user.passwordHash))
    ↪;
    if (ok) { req.session.user = { username: user.username, role: user.role ||
    ↪null }; return res.redirect('/formulare.html'); }
    else return res.redirect('/login.html?error=1');
  } catch (err) {
    console.error('Login-Fehler beim Vergleichen der Hashes:', err);
    return res.redirect('/login.html?error=1');
  }
});
```

Listing 6.3: POST /login: Vergleicht Passwort gegen `passwordHash`

Erklärung. Authentifizierung per `bcrypt.compare`. Ein temporärer Fallback für Klartext-Passwörter bleibt, bis alle Konfigurationen gehasht sind. Serverseitig sollten zusätzlich Lockout und Rate-Limit laufen.

Speichern von Genehmigungen und Signaturen

```
app.post('/api/save-approval', requireAuth, checkApprovalPermission, async (req,
    ↪res) => {
  // payload: { filename?, id?, approvals: {Dekan:'ja'|'nein', ...} }
  // Bestimme target (Zuarbeit|Dozenten), backup der Datei, schreibe gefilterte
    ↪approvals zurck.
});

app.post('/api/save-signature', requireAuth, async (req,res) => {
  // Nur Dekan darf speichern; payload enthlt filename|id|signature
  // Bestimme target unter Zuarbeit/Dozenten, backup, schreibe signature ins
    ↪modul.dkanUnterschrift oder dozent.dekanUnterschrift
});
```

Listing 6.4: POST /api/save-approval und POST /api/save-signature (Kurzfassung)

Erklärung. Endpoints prüfen Rolle, ermitteln die Zieldatei via `filename` oder `id`, legen Backups an und schreiben nur erlaubte Felder zurück.

6.2 formulare.js (Frontend-Logik)

PDF-Erzeugung (html2pdf)

```
function generatePDF() {
  const element = document.getElementById("form-container");
  const opt = {
    margin: 0,
    filename: (() => {
      const isZuarbeit = window.currentTemplate === "zuarbeit";
      const name = (document.getElementById(isZuarbeit ? "kw-name" : "nachname")
    ↪?.textContent || "unbekannt").trim();
      const prefix = isZuarbeit ? "Zuarbeitsblatt_" : "Dozentenblatt_";
      return `${prefix}${name || "unbekannt"}.pdf`;
    })(),
    image: { type: "jpeg", quality: 0.98 },
    html2canvas: {
      scale: 2,
      scrollY: 0,
      onclone: function (clonedDoc) {
        const elementsToProtect = [".section","table","tr","td","th",".notes",".
    ↪signature-area",".zuarbeit-header-box",".planungswochen-display"];
        elementsToProtect.forEach(selector => clonedDoc.querySelectorAll(
    ↪selector).forEach(el => {
          el.style.pageBreakInside = "avoid"; el.style.breakInside = "avoid";
          if (selector === "table") { el.style.display = "table"; el.style.width
    ↪ = "100%"; }
        }));
        clonedDoc.querySelectorAll("table").forEach(t => t.style.maxWidth = "190
    ↪mm");
```

```
22          clonedDoc.querySelectorAll("p, li, .text-block").forEach(el => { el.
      ↪style.widows = "3"; el.style.orphans = "3"; });
23        }
24      },
25      jsPDF: { unit: "mm", format: "a4", orientation: "portrait" }
26    };
27    html2pdf().set(opt).from(element).save();
28  }
```

Listing 6.5: Konfiguration von html2pdf und Aufruf

Erklärung. Optimiertes `html2pdf` Setup: höhere Auflösung, page-break-Avoid-Styling per `onclone` und dynamischer Dateiname je nach Template.

Client: Dekan-Unterschrift speichern (kurz)

```
1  async function saveDekanSignature() {
2    const userRole = await getUserRole();
3    if (userRole !== 'Dekan') { alert('Nur der Dekan kann unterschreiben.');
       ↪return; }
4    const signature = document.getElementById('unterschrift-dekan').value.trim();
5    if (!signature) { alert('Bitte Unterschrift eingeben.'); return; }
6    const target = await determineTargetFromUIOrTextarea();
7    const payload = { filename: target.filename, id: target.id, type: target.type,
       ↪ signature };
8    const r = await fetch('/api/save-signature', { method:'POST', headers:{'
       ↪Content-Type':'application/json'}, body: JSON.stringify(payload) });
9    if (!r.ok) throw new Error('Speichern fehlgeschlagen');
10   alert('Unterschrift gespeichert.');
11 }
```

Listing 6.6: saveDekanSignature: Payload bauen und POST an /api/save-signature

Erklärung. Client baut Payload mit `filename` oder `id` plus `signature` und POSTet an den Endpoint. Frontend prüft Rolle vor dem Absenden, Autorisierung muss serverseitig stattfinden.

6.3 Formulare.html (UI)

Kontrollen: Checkboxen, Unterschrift-Input, Buttons

```
1  <!-- Checkboxen fr Genehmigungen -->
2  <label><input type="checkbox" id="checkbox-studienamt"> Studienamt</label>
3  <label><input type="checkbox" id="checkbox-dekan"> Dekan</label>
4  <label><input type="checkbox" id="checkbox-studiendekan"> Studiendekan</label>
5  <button id="save-approval-btn" type="button" class="btn">Speichern</button>
6
7  <!-- Dekan-Unterschrift -->
8  <label for="unterschrift-dekan">Unterschrift Dekan:</label>
9  <input id="unterschrift-dekan" type="text" />
```

```
10 <button id="save-signature-btn" type="button" class="btn">unterschreiben</
     ↪button>
```

Listing 6.7: Kontrollen: Checkboxen, Unterschrift-Input, Buttons

Erklärung. UI-Elemente sind vorhanden, das JS kontrolliert Sichtbarkeit und Enablement je nach Rolle und führt die POSTs an die Endpoints aus.

6.4 login.html

```
1  <main class="login-root" aria-live="polite">
2    <div class="login-card" role="main">
3      <img src="https://upload.wikimedia.org/wikipedia/de/1/16/HTWK-Logo.svg" alt=
       ↪"HTWK Logo" class="logo" />
4      <h1>Anmeldung</h1>
5      <p class="subtitle">Melden Sie sich mit Ihrem Verwaltungs-Konto an, um die
       ↪Hauptseite zu ffnen.</p>
6
7      <div class="error" id="errorMsg" role="alert" style="display: none;"></div>
8
9      <form id="login-form" method="post" action="/login" autocomplete="off"
       ↪novalidate>
10       <label class="field"><span>Benutzername</span>
11         <input id="username" name="username" required aria-label="Benutzername"
       ↪/>
12       </label>
13       <label class="field"><span>Passwort</span>
14         <input id="password" name="password" type="password" required
       ↪aria-label="Passwort" />
15       </label>
16       <button class="btn" type="submit" id="submitBtn">Anmelden</button>
17     </form>
18
19     <div class="note">Nach erfolgreichem Login werden Sie automatisch zur
       ↪Hauptseite weitergeleitet.</div>
20     <div class="footer">Hochschule fr Technik, Wirtschaft und Kultur Leipzig.</
       ↪div>
21   </div>
22 </main>
```

Listing 6.8: Formular: Struktur, Felder und ARIA-Hinweise

Erklärung. Das Formular schickt sensible Daten per POST an `/login`. `autocomplete=öff"` und `novalidate` geben dem Script die Kontrolle über das Verhalten im Browser, das heißt aber nicht, dass Validierung entfallen darf. `role=älert"` markiert den Container für serverseitige Fehlermeldungen, damit Screenreader die Nachricht ankündigen. Security-Reminder: Passwörter nur über HTTPS übertragen, serverseitig CSRF-Schutz, Rate limiting und Logging-Regeln (Passwörter niemals loggen).

```
1 (function(){
2   const form = document.getElementById('login-form');
3   const err = document.getElementById('errorMsg');
```

```
  const submitBtn = document.getElementById('submitBtn');

  function showError(msg){
    err.textContent = msg;
    err.style.display = 'block';
  }
  function clearError(){
    err.textContent = '';
    err.style.display = 'none';
  }

  const params = new URLSearchParams(location.search);
  if (params.get('error')) showError('Falscher Benutzername oder Passwort');
  if (params.get('loggedout')) {
    showError('Erfolgreich abgemeldet');
    setTimeout(clearError, 3500);
  }

  form.addEventListener('submit', function(ev){
    clearError();
    const u = (form.username.value || '').trim();
    const p = (form.password.value || '').trim();
    if (!u || !p){
      ev.preventDefault();
      showError('Bitte Benutzername und Passwort ausfllen.');
      return;
    }
    submitBtn.disabled = true;
    submitBtn.textContent = 'Anmelden...';
    setTimeout(()=>{
      submitBtn.disabled = false;
      submitBtn.textContent = 'Anmelden';
    }, 10000);
  });
})();
```

Listing 6.9: Clientseitiges Verhalten: Validierung, Fehleranzeige, Button-State

Erklärung. Das Script verhindert leere Submits, zeigt Statusmeldungen aus der URL und vermindert Doppelabschickungen durch temporäres Deaktivieren des Buttons. Die Re-Enable-Timeout ist ein UX-Fallback, echte Schutzmaßnahmen gegen Brute-Force gehören auf den Server (Rate limiting, Account lockout, IP-Blocklists). Ein asynchroner Login-Flow mit klaren HTTP-Statuscodes wäre robuster und barrierefreundlicher.

Kapitel 7

Auswertung

7.1 Detaillierte Kosteneinsparungen

Kostenfaktor	Papierprozess	Digitaler Prozess	Einsparung (€/Formular)
Druckkosten	0,12 € [40]	0 €	0,12 €
Patronenkosten	0,05 €[41]	0 €	0,05 €
Wartung und Betrieb Drucker 50 €/Jahr, wenn 10 000 Seiten/Jahr	0,005 €[42]	0 €	0,005 €
Personalaufwand 3 520,10 €im Monat [43][44]	bei 30min 12,18 €	bei 5min 2,03 €	10,15 €
Gesamtkosten pro Formular	**12,355 €**	**2,03 €**	**10,325 €**

Monatsgehalt brutto:

$$E9a, Stufe1 = 3520.10$$

Monatliche Arbeitszeit:

$$40\,Stunden/Woche \Rightarrow 173{,}33\,Stunden/Monat$$

Bruttostundenlohn:

$$\frac{3520.10}{173.33\,h} \approx 20.30/h$$

Arbeitgeber-Nebenkosten (ca. 20%):

$$20.30/h \times 1.20 = 24.36/h$$

Bearbeitungszeit pro Formular:

$$bei\ 2.5\,Minuten = \frac{2.5}{60} = 0.0417\,h$$

$$bei\ 5\,Minuten = \frac{5}{60} = 0.0833\,h$$

$$bei\ 20\,Minuten = \frac{20}{60} = 0.333\,h$$

$$bei\ 30\,Minuten = \frac{30}{60} = 0.5\,h$$

Kosten pro Formular (digital):

$$bei\ 2.5\,Minuten\ 24.36/h \times 0.0833\,h \approx 1.02$$

$$bei\ 5\,Minuten\ 24.36/h \times 0.0417\,h \approx 2.03$$

Kosten pro Formular

$$bei\ 20\,Minuten\ 24.36/h \times 0.3333\,h \approx 8.12$$

$$bei\ 30\,Minuten\ 24.36/h \times 0.5\,h \approx 12.18$$

7.1.1 Skalierungseffekte und Jahresszenario

Menge Formulare p.a.	Einsparung je Formular (10,32 €)	Gesamteinsparung p.a.
100	10,32 €	1 032 €
1 000	10,32 €	10 320 €
5 000	10,32 €	51 600 €

Schon bei moderater Nutzung von 1 000 Formularen jährlich amortisieren sich Investitionen in die Webanwendung deutlich mit einer Einsparung von fast 7 280 €.

7.2 Vertiefte Ökobilanz

Umweltindikator	Papier (10 Blatt)	Digital	Reduktion
CO_2-Äquivalent (kg)	0,05[45]	0,00648	87,0 %
Energieverbrauch (kWh)	0,65[45]	0,0202	96,9 %
Wasserverbrauch (l)	2,51[45]	0,19	92,4 %
Holz (kg)	0,11[45]	0	100 %

Erläuterungen zur digitalen Ökobilanz

CO_2-Äquivalent

- Leseenergie Laptop: 23,4 g CO_2e pro Stunde Lesedauer [46]

- Lesedauer für 10 Seiten 12,5 min → 4,88 g CO
- Datentransfer 1 GB 5 kWh also für 1 MB PDF: 0,005 kWh 0,005 kWh × 321 g CO_2e/kWh = 1,61 g CO_2e [46][47]
- **Summe:** 6,48 g CO_2e

Energieverbrauch

- Lesedauer:4,88 g / 321 g/kWh = 0,0152 kWh
- Datentransfer: 0,005 kWh
- **Summe:** 0,0202 kWh

Wasserverbrauch

- Indirekt (Strombereitstellung): 2 gal/kWh × 3,785 L/gal = 7,57 L/kWh → 0,153 l
- Direkt (Data-Center-Kühlung): 0,48 gal/kWh × 3,785 L/gal = 1,82 L/kWh → 0,037 l [48]
- **Summe:** 0,19 l

Holz: 0 kg

7.3 Langfristige Umweltwirkungen

- **CO_2-Bilanz über 5 Jahre** (1 000 Formulare je 1 Blatt im Jahr):

$$1\,000 \times (0{,}005kg - 0{,}000648kg) \times 5 = 21{,}76kgCO$$

  $_2$-Äquivalent gespart

- **Wasserersparnis über 5 Jahre** ((1 000 Formulare je 1 Blatt im Jahr):

$$1\,000 \times (0{,}251l - 0{,}019l) \times 5 = 1\,160lWassergespart$$

Langfristig leistet die Digitalisierung einen signifikanten Beitrag zum Ressourcenschutz und zur Reduktion klimaschädlicher Emissionen.

Kapitel 8

Ausblick und zukünftige Erweiterungen

8.1 Digitale Signaturen (PKI-basiert)

Um das Canvas-Element kryptographisch abzusichern, kann das Programm in der Zukunft auf eine X.509-basierte Public Key Infrastructure zurückgreifen, die auf Zertifikate, Certificate Revocation Lists und eine Certification Authority basiert [49]. Die Implementierung erfolgt über die Web Crypto API im Browser, die das Signieren von Blob-Daten (z. B. Hash des generierten PDFs oder der Canvas-Bilder) mittels RSA oder ECDSA direkt unterstützt [50].
Um Signaturen ins PDF einzubetten, eignet sich zum Beispiel die Bibliothek PDF-LIB. Sie kann PAdES-kompatible Signaturobjekte erzeugen und direkt im Dokument platzieren [51].

8.2 Einbindung einer SQLite-Datenbank

Um die Anwendung weiterzuentwickeln, ist es sinnvoll, eine einfache Datenbank wie SQLite zu nutzen. Dadurch können erzeugte PDFs dauerhaft gespeichert werden. Das Studienamt hat damit bei der PDF-Filterung direkten Zugriff auf alle PDFs, ohne die im Fazit erwähnten zusätzlichen Schritte. Das sorgt für einen reibungslosen Ablauf.

8.2.1 Technische Umsetzung

1. **Wahl der Datenbank: SQLite** SQLite erfordert keinen gesonderten Server und speichert alle Informationen in einer Datei, was den Verwaltungsaufwand reduziert [52].

2. **Server-Stack** Express wird als Web-Framework für Node.js eingesetzt [53]. Das `sqlite3`-Modul dient zur Verbindung mit der SQLite-Datenbank [54]. Multer wird verwendet, um `multipart/form-data`-Uploads zu empfangen und zu verarbeiten [55].

3. **Upload-Prozess** Der Browser erstellt das PDF und sendet es über die Fetch-API als `multipart/form-data` an `POST /upload`. Multer nimmt den Upload entgegen, legt die Datei im festgelegten Verzeichnis ab und liefert den Dateipfad zurück [55]. Mit dem `sqlite3`-Modul fügt Express einen Eintrag in der Tabelle `pdf_documents` hinzu, dieser enthält den Dateipfad sowie Metadaten wie Dozent, Semester und Zeitstempel [53, 54].

8.3 KI-gestützte Validierung (zur Erkennung von Tippfehlern)

Zur Verbesserung der Eingabequalität kann ein NLP-Modul eingebunden werden, das Textarea-Inhalte tokenisiert, Part-of-Speech-Tagging durchführt und Rechtschreib- sowie Grammatikfehler markiert [56].

spaCy ist eine kostenlose Open-Source-Bibliothek für Natural Language Processing (NLP) in Python, die speziell für den produktiven Einsatz entwickelt wurde. Sie ermöglicht es, aus großen Textmengen verlässliche Einblicke zu gewinnen und robuste NLP-Pipelines zu erstellen. [57] Ein Framework wie spaCy mit einem deutschen Modell erlaubt mittels REST-API oder direkt im Browser (z. B. nlp.js) die Integration von Rechtschreibprüfung, Syntaxanalyse und Vorschlägen zur Satzkorrektur.

Kapitel 9

Fazit und Evaluation

9.1 Fazit

Tabelle 9.1: Erfüllungsgrad der Anforderungen

Anforderungen	Erfüllt	Teilweise erfüllt	Nicht erfüllt
1. Vollständige Digitalisierung des Prozesses	✓		
2. Ausfüllen beider Dokumente	✓		
3. Zugriffskontrolle	✓		
4. Ausfülldaten sind sichtbar und auflistbar	✓		
5. Prüf- und Freigabeworkflow	✓		
6. Filter- und Suchfunktionen		✓	
7. PDF-Generierung	✓		

Alle in Abschnitt „Anforderungen“ genannten Punkte wurden umgesetzt und funktionieren wie erwartet. 9.1).komplexe Daten werden automatisiert in die offiziellen HTWK-Formulare überführt und als druckfertige PDFs ausgegeben. Bei Punkt 6 ist die Funktionalität grundsätzlich gegeben und erfüllt ihren Zweck, weist jedoch Einschränkungen auf. Die erzeugten PDF-Dateien werden vom Browser standardmäßig im lokalen Downloads-Ordner gespeichert. Damit der Filter vorhandene PDFs zuverlässig erkennt und einliest, müsste die Webanwendung auf diesen Ordner zugreifen. In der Praxis sind zusätzliche Schritte erforderlich. Da diese Anforderung aber in der Spezifikation als optionale Funktion („kann“) eingeordnet war, handelt es sich um eine wünschenswerte Zusatzfunktion. Technisch liegt eine schlichte, aber robuste Architektur vor: ein Node.js-Backend, HTML-Templates für beide Blatttypen, automatisierte Befüllung und ein klarer Export-Workflow mit definierten Dateinamenskonventionen und Signaturfeldern für Verantwortliche und Dekanat. Die Integration von Rollen (Studienamt, Dekanat, Admin) und die Möglichkeit, PDFs nur für Berechtigte einzusehen, schafft die notwendige Trennung zwischen Erfassung und Verwaltung. Kurz: das System leistet, was es versprechen soll, und reduziert manuellen Aufwand spürbar.

Abschließend: Die Arbeit liefert eine praxisnahe, umsetzbare Lösung für ein konkretes Problem an der Hochschule. Sie zeigt zugleich, wo Grenzen liegen und welche Maßnahmen nötig sind, um die Anwendung produktiv und datenschutzkonform zu betreiben. Für mich

persönlich war das Projekt eine wichtige Übung in Systemdenken: Anforderungen, Datenqualität, Sicherheit und Usability müssen Hand in Hand entwickelt werden. Wenn diese Bausteine weitergedacht werden, ist die Anwendung reif für den Einsatz - und nützlich für alle, die Formulare zuverlässig und effizient verarbeiten wollen.

9.2 Einordnung der Ergebnisse im Hinblick auf die Forschungsfragen

Die Arbeit belegt, dass eine durchgängige Digitalisierung des Prozesses von der Datenerfassung bis zur PDF-Erstellung technisch realisierbar ist. Gegenüber dem Papierprozess verkürzt sich die Bearbeitungszeit deutlich und durch Pflichtfelder plus Validierung treten weniger Fehler auf, was die Bedienung für die Anwender spürbar erleichtert. Rechnet man Druck- und Verwaltungsaufwand hoch, ergeben sich nennenswerte Kosteneinsparungen; zugleich sinkt der Papierverbrauch und damit der ökologische Fußabdruck. Für den produktiven Betrieb sind noch Ergänzungen nötig, etwa dauerhafte Ablage, robuste Authentifizierung und Signaturen. Details und Zahlen stehen in Kapitel 5 und 7.

Eidesstattliche Versicherung

Hiermit versichere ich, Fadi Mkhalale, dass die Bachelorarbeit mit dem Titel "Digitale Umsetzung eines Prozesses in der Hochschulverwaltung am Beispiel der Lehrplanung für ein Semester" nicht anderweitig als Prüfungsleistung verwendet wurde und diese Bachelorarbeit noch nicht veröffentlicht worden ist. Die hier vorgelegte Bachelorarbeit habe ich selbstständig und ohne fremde Hilfe abgefasst. Ich habe keine anderen Quellen und Hilfsmittel als die angegebenen benutzt. Diesen Werken wörtlich oder sinngemäß entnommene Stellen habe ich als solche gekennzeichnet.

Leipzig, 14. Dezember 2025 Unterschrift

Literaturverzeichnis

[1] Peter Detemple, Dr. Florian Kaufmann, Dr. Verena Holl, Dr. Christian Marettek und Jens Mattmüller , "Digitalisierung an universitäten pwc-studie 2021," 2021. [Online]. Available: https://www.pwc.de/de/branchen-und-markte/oeffentlicher-sektor/digitalisierung-an-universitaeten.html

[2] Peter Detemple, Dr. Florian Kaufmann, Dr. Verena Holl, Dr. Christian Marettek und Jens Mattmüller, "die digitalisierung an den universitäten steuern," 2021. [Online]. Available: https://www.pwc.de/de/branchen-und-markte/oeffentlicher-sektor/pwc-die-digitalisierung-an-den-universitaeten-steuern.pdf

[3] A. Corbo and B. Whitfield, "What is html?" Online, n.d., seiten-Angaben: zuletzt aktualisiert, Zugriff am 2025-12-09. [Online]. Available: https://builtin.com/software-engineering-perspectives/html

[4] K. Chris, "What is html - definition and meaning of hypertext markup language," 24-08-2021. [Online]. Available: https://www.freecodecamp.org/news/what-is-html-definition-and-meaning

[5] GeeksforGeeks, "Top 10 uses of html in the real world," Online tutorial, 05 Aug, 2025. [Online]. Available: https://www.geeksforgeeks.org/html/uses-of-html

[6] W3Schools, "Css introduction," Online tutorial, n.d., zugriff am 2025-09-26. [Online]. Available: https://www.w3schools.com/Css/css_intro.asp

[7] GeeksforGeeks, "Uses of css," Online tutorial, 23-07-2025, zugriff am 2025-09-26. [Online]. Available: https://www.geeksforgeeks.org/css/uses-of-css

[8] MDN Web Docs, "What is css?" Online documentation, 13-11-2025, zugriff am 2025-09-26. [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Styling_basics/What_is_CSS

[9] ——, "What is javascript?" Online documentation, 30-11-2025, zugriff am 2025-09-26. [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Scripting/What_is_JavaScript

[10] I. Kantor, "An introduction to javascript," Online tutorial, 08-08-2020, zugriff am 2025-09-29. [Online]. Available: https://javascript.info/intro

[11] GeeksforGeeks, "What is javascript used for?" Online tutorial, 12-05-2025, zugriff am 2025-09-29. [Online]. Available: https://www.geeksforgeeks.org/javascript/uses-of-javascript

[12] Node.js Project / OpenJS Foundation, “Introduction to node.js,” Official documentation, n.d., offizielle Node.js-Dokumentation Zugriff am 2025-10-01. [Online]. Available: https://nodejs.org/en/learn/getting-started/introduction-to-nodejs

[13] GeeksforGeeks, “Node.js tutorial - geeksforgeeks,” Online tutorial, 23-09-2025, zugriff am 2025-09-31. [Online]. Available: https://www.geeksforgeeks.org/node-js/nodejs

[14] freeCodeCamp.org, “What exactly is node.js and why should you use it?” Online article, 02.06.2017, zugriff am 2025-10-01. [Online]. Available: https://www.freecodecamp.org/news/what-exactly-is-node-js-and-why-should-you-use-it-8043a3624e3c

[15] A. Avram, “Paypal switches from java to javascript,” 29-11-2013. [Online]. Available: https://www.infoq.com/news/2013/11/paypal-java-javascript

[16] Jeffrey Erickson, “What is json?” Online documentation, 04-04-2024, oracle Content / Dokumentation, Zugriff am 2025-10-02. [Online]. Available: https://www.oracle.com/de/database/what-is-json

[17] D. Kreiter, “Json verstehen: Grundlagen und anwendungsbereiche,” Blog / Tutorial, 2024, publikationsdatum auf der Seite. [Online]. Available: https://www.derinformatikstudent.de/json

[18] Erik Koopmans,Wilco Breedt and Chris Hansohn (GitHub), “html2pdf.js,” GitHub repository, n.d., repository / maintainer: eKoopmans, Zugriff am 2025-10-08. [Online]. Available: https://github.com/eKoopmans/html2pdf.js

[19] eKoopmans, “html2pdf.js (npm package),” npm package page, n.d., zugriff am 2025-10-08. [Online]. Available: https://www.npmjs.com/package/html2pdf.js

[20] pdf2htmlEX project, “pdf2htmlex,” Project website, n.d., zugriff am 2025-10-06. [Online]. Available: https://pdf2htmlEX.github.io/pdf2htmlEX/

[21] pdf2htmlEX (GitHub), “pdf2htmlex - github repository,” GitHub repository, n.d., zugriff am 2025-10-06. [Online]. Available: https://github.com/pdf2htmlEX/pdf2htmlEX

[22] Podman Project, “What is podman?” Project documentation, n.d., zugriff am 2025-11-06. [Online]. Available: https://docs.podman.io/en/latest/

[23] Red Hat, “What is podman? - red hat,” Company documentation, 20-06-2024, red Hat Webseite, Zugriff am 2025-11-06. [Online]. Available: https://www.redhat.com/en/topics/containers/what-is-podman

[24] des Bundeskanzleramtes Österreich, “Authentifizierung ohne passwort,” Online portal, 01-06-2022, portal, Zugriff am 2025-10-09. [Online]. Available: https://www.onlinesicherheit.gv.at/Services/News/Authentifizierung-ohne-Passwort.html

[25] Bundesamt für Sicherheit in der Informationstechnik, “Technische betrachtung: Sicherheit bei 2fa-verfahren,” BSI Webseitendokumentation, n.d., zugriff am 2025-10-09. [Online]. Available: https://www.bsi.bund.de/DE/Themen/Verbraucherinnen-und-Verbraucher/Informationen-und-Empfehlungen/Cyber-Sicherheitsempfehlungen/Accountschutz/Zwei-Faktor-Authentisierung/Bewertung-2FA-Verfahren/bewertung-2fa-verfahren_node.html

[26] OWASP, “Session management cheat sheet,” Online cheat sheet, n.d., zugriff am 2025-10-12. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html

[27] Puppeteer Project, “Puppeteer - pdf generation (official guide),” Project documentation, n.d., zugriff am 2025-11-11. [Online]. Available: https://pptr.dev/guides/what-is-puppeteer

[28] Adobe Inc., “Fdf format support for acroforms / acrobat help,” n.d., adobe Support, Zugriff am 2025-11-19. [Online]. Available: https://helpx.adobe.com/acrobat/kb/acrobat-forms-form-data-web.html

[29] Hopding, “pdf-lib (documentation),” n.d., project docs, Zugriff am 2025-11-11. [Online]. Available: https://github.com/Hopding/pdf-lib#features

[30] MDN Web Docs, “Json - javascript,” Online documentation, n.d., mDN-Seite, Zugriff am 2025-10-15. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON

[31] JSON Schema Working Group, “what is json schema - official website,” Specification website, n.d., zugriff am 2025-10-16. [Online]. Available: https://json-schema.org/overview/what-is-jsonschema

[32] Manu Sporny, Dave Longley,Gregg Kellogg, Markus Lanthaler, Pierre-Antoine Champin and Niklas Lindström, “Json-ld 1.1 - a json-based serialization for linked data,” 16-07-2020. [Online]. Available: https://www.w3.org/TR/json-ld11/

[33] G. Schreiber, Y. Raimond, and W3C, “Rdf 1.1 primer,” 24-06-2014, w3C Working Group Note. [Online]. Available: https://www.w3.org/TR/rdf11-primer/

[34] E. P. David Beckett, Tim Berners-Lee and G. Carothers, “Turtle - rdf 1.1,” 2014, w3C Recommendation. [Online]. Available: https://www.w3.org/TR/turtle/

[35] Tim Bray, Jean Paoli, Sperberg-McQueen, Eve Maler, and François Yergeau, “Extensible markup language (xml) 1.0,” 26-11-2008, w3C Recommendation. [Online]. Available: https://www.w3.org/TR/xml/

[36] W3C, “Xslt 3.0,” W3C specification, 08.06.2017, zugriff am 2025-10-15. [Online]. Available: https://www.w3.org/TR/xslt20/

[37] Y. Shafranovich, “Rfc 4180: Common format and mime type for csv files,” 10-2005. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc4180

[38] I. d. N. Oren Ben-Kiki, Clark Evans, “Yaml ain’t markup language,” 01-010-2021, spec revision. [Online]. Available: https://yaml.org/spec/1.2.2/

[39] Google / Protocol Buffers Team, “Protocol buffers - official documentation,” n.d., offizielle Docs, Zugriff am 2025-10-16. [Online]. Available: https://protobuf.dev/

[40] Superpatronen.de, “Wie setzen sich druckkosten zusammen?” Ratgeber-Artikel, n.d., zugriff am 2025-10-23. [Online]. Available: https://www.superpatronen.de/ratgeber/druckkosten-zusammensetzung-berechnung

[41] Sascha Kolditz, “Wie viel kostet es, eine seite zu drucken?” Blogpost, 07-08-2022, zugriff am 2025-10-23. [Online]. Available: https://www.fairtoner.de/blog/wie-viel-kostet-es-eine-seite-zu-drucken/

[42] tectonika GmbH, “Druckerwartung - kosten, ablauf und warum sie wichtig ist,” Ratgeber-Artikel, n.d., zugriff am 2025-10-23. [Online]. Available: https://tectonika.de/wartung-fuer-drucker/

[43] LSF Sachsen, “Entgelttabelle für beschäftigte in den entgeltgruppen 1 bis 15,” 2025, pDF. [Online]. Available: https://www.lsf.sachsen.de/download/Tarif/Entgelttabelle_TV-L_ab_01.02.2025.pdf

[44] kommunalforum.de, “Beispiele für stellen in entgeltgruppe e 9a tv-l,” Website article, n.d., zugriff am 2025-10-26. [Online]. Available: https://www.kommunalforum.de/e_9_a_tv_l.php

[45] Papiernetz / Initiative Pro Recyclingpapier, “nachhaltigkeitsrechner,” Website tool, n.d., zugriff am 2025-10-26. [Online]. Available: https://www.papiernetz.de/informationen/nachhaltigkeitsrechner/

[46] Zuza Nazaruk, “Is digital more environmentally friendly than paper?” 18-10-2020, artikel. [Online]. Available: https://except.eco/knowledge/is-digital-more-environmentally-friendly-than-paper/

[47] Nowtricity, “Current emissions in germany,” Online dashboard, n.d., live data, Zugriff am 2025-10-26. [Online]. Available: https://www.nowtricity.com/country/germany/

[48] WaterCalculator.org, “Data centers, digital lifestyles and water use,” 09-11-2018, artikel. [Online]. Available: https://watercalculator.org/footprint/data-centers-water-use/

[49] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk, “Internet x.509 public key infrastructure certificate and certificate revocation list (crl) profile (rfc 5280),” 05-2008. [Online]. Available: https://tools.ietf.org/html/rfc5280

[50] W3C, “Web cryptography,” W3C specification, 22-04-2025, zugriff am 2025-10-31. [Online]. Available: https://www.w3.org/TR/WebCryptoAPI/

[51] PDF-LIB, “Pdf-lib - create and modify pdf documents in any javascript environment,” Project / documentation, n.d., zugriff am 2025-11-02. [Online]. Available: https://pdf-lib.js.org/

[52] S. D. Team, “Sqlite is serverless,” 16.04.2025, sQLite official docs, Zugriff am 2025-11-07. [Online]. Available: https://www.sqlite.org/serverless.html

[53] O. J. Foundation, “Express.js - fast, unopinionated, minimalist web framework for node.js,” Project documentation, n.d., zugriff am 2025-11-08. [Online]. Available: https://expressjs.com/

[54] npm package: sqlite3, “node-sqlite3,” n.d., maintainers Daniel Lockyer,Konstantin Käfer, Dane Springmeyer, Will White... Zugriff am 2025-11-08. [Online]. Available: https://www.npmjs.com/package/sqlite3

[55] Wojciech Maj, "Multer - node.js middleware for handling multipart/form-data," GitHub repository, n.d., zugriff am 2025-11-11. [Online]. Available: https://github.com/expressjs/multer

[56] Marvin Erdner, "Languagetool http api," 16-06-2025, zugriff am 2025-10-29. [Online]. Available: https://languagetool.org/insights/de/beitrag/umformulieren/

[57] Explosion, "spacy 101: Everything you need to know," Project documentation, n.d., zugriff am 2025-10-29. [Online]. Available: https://spacy.io/usage/spacy-101