ingenieur
wissenschaften
htw saar

Hochschule für
Technik und Wirtschaft
des Saarlandes
University of
Applied Sciences

Bachelor-Thesis

zur Erlangung des akademischen Grades
Bachelor of Science (B. Sc.)
an der Hochschule für Technik und Wirtschaft des Saarlandes
im Studiengang Kommunikationsinformatik
der Fakultät für Ingenieurwissenschaften

Enterprise-Resource-Planning-Software mit integrierter Dateiverwaltung und automatisierter Dokumentengenerierung: Konzeptionierung und Entwicklung auf Basis des Django Frameworks und LaTeX

vorgelegt von
Jan-Merlin Geuskens, 3580970
Laura-Ann Inge Schiestel, 3686779

betreut und begutachtet von
Prof. Dr. Helmut G. Folz

Saarbrücken, 30. September 2018

Selbständigkeitserklärung

Ich versichere, dass ich die vorliegende Arbeit (bei einer Gruppenarbeit: den entsprechend gekennzeichneten Anteil der Arbeit) selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.
Ich erkläre hiermit weiterhin, dass die vorgelegte Arbeit zuvor weder von mir noch von einer anderen Person an dieser oder einer anderen Hochschule eingereicht wurde.
Darüber hinaus ist mir bekannt, dass die Unrichtigkeit dieser Erklärung eine Benotung der Arbeit mit der Note „nicht ausreichend“ zur Folge hat und einen Ausschluss von der Erbringung weiterer Prüfungsleistungen zur Folge haben kann.

Saarbrücken, 30. September 2018

Jan-Merlin Geuskens,
3580970

Laura-Ann Inge Schiestel,
3686779

Zusammenfassung

Im Rahmen der Bachelor-Thesis wurde eine Webapplikation zur Verwaltung der Aufträge eines Unternehmens mit integrierter Dokumentengenerierung und Prozesssteuerung konzeptioniert und entwickelt.

Das Festhalten an Standardsoftware und eingefahrenen Prozessen führte zu Problemen bei der Entwicklung der Unternehmesberatung Profiteam. Die Unternehmensprozesse wurden immer komplexer und die aus der Not genutzte Softwarelösung immer unpassender für die Bearbeitung der Prozesse.

Die Lösung des Problems liegt in der Verwendung von Individualsoftware, die genau auf die Unternehmensprozesse zugeschnitten ist. Durch die Nutzung einer eigenen Softwarelösung und der damit verbundenen Flexibilität entsteht für das Unternehmen eine Situation, die viele positive Begleiterscheinungen mit sich bringt. Zum einen wird das Unternehmen unabhängiger von Anbietern proprietärer Software, zum anderen bilden sich Wettbewerbsvorteile, die dem Unternehmen die Möglichkeit geben zu wachsen. Sobald die Software erprobt ist, amortisiert sich die Entwicklung und es können zusätzliche Gewinne generiert und Geschäftsbeziehungen geknüpft werden, indem die Software an Mitbewerber verkauft bzw. lizenziert wird.

Zur Lösung der Probleme innerhalb der Prozessabläufe wird bei Profiteam eine Webapplikation auf Basis des auf Python basierten Django Frameworks entwickelt, welches sich durch hohe Sicherheit und gute Skalierbarkeit auszeichnet. Die im Zuge der Unternehmensprozesse zu erstellenden Dokumente werden mit Hilfe von LaTeX automatisch generiert, was den Arbeitsaufwand massiv reduziert. Gleichzeitig bietet die entwickelte Webapplikation den Zugriff auf das von ihr verwaltete Dateisystem, in welchem kundenspezifische Daten systematisch gespeichert werden, und verhindert somit Inkonsitenzen.

In dieser Arbeit wird die Vorgehensweise zur Entwicklung der Software, beginnend bei der Analyse der Probleme, beschrieben. Technologische Grundlagen sowie die der Software zu Grunde liegenden Konzepte werden erklärt.

Das positive Feedback durch die Stakeholder im Zuge der Evaluation zeigt, dass die im Rahmen dieser Arbeit entwickelten Konzepte und der vorgestellte Prototyp die prozesstechnischen Probleme des Unternehmens lösen können.

Die Arbeit wurde von zwei Personen erstellt, wobei Herr Jan-Merlin Geuskens die Hauptverantwortung für die Kapitel und Abschnitte über die Django-Applikation trägt (2.1, 3, 5.1, 5.2, 5.3, 6.1).

Frau Laura-Ann Inge Schiestel trägt die Hauptverantwortung für die Kapitel und Abschnitte zur Dokumentengenerierung (2.2, 4, 5.4, 6.2, 6.3).

Die nicht explizit aufgeführten Kapitel und Abschnitte unterliegen gemeinsamer Verantwortung.

„Die reinste Form des Wahnsinns ist es,
alles beim Alten zu lassen
und gleichzeitig zu hoffen,
dass sich etwas ändert.“

— Albert Einstein [21]

Danksagung

An dieser Stelle möchten wir uns bei unserem betreuenden Dozenten Prof. Dr. Helmut G. Folz bedanken, der für uns stets ein offenes Ohr hatte und uns beratend zur Seite stand. Desweiteren gilt unser Dank dem betreuenden Unternehmen, das es uns ermöglicht hat auf kreative Art und Weise die aufgetragene Problemstellung zu lösen und uns das Vertrauen entgegengebracht hat, in weiten Teilen eigenverantwortlich zu arbeiten.

Besonderer Dank gebührt unseren Familien und Freunden, die uns unermüdlich unterstützten, Tippfehler mit unglaublicher Präzision fanden und somit eine Arbeit dieser Qualität ermöglicht haben.

Inhaltsverzeichnis

1 Einleitung

Das Wachstum eines Kleinstunternehmes mit zwei Mitarbeitern zu einem mittleren Unternehmen mit über 100 externen Vertragspartnern kann, insbesondere im Hinblick auf die Unternehmensprozesse, eine Herausforderung darstellen.

Durch den frühzeitigen und korrekten Einsatz von Software kann dieser Vorgang unterstützt bzw. ermöglicht werden. Es ist allerdings auch möglich, dass ein Unternehmen zu lange an Standardsoftware festhält, welche nicht die Funktionalität und Flexibilität bieten kann, die benötigt wird. In zunehmendem Maße wird deshalb auf die Unternehmensprozesse zugeschnittene Individualsoftware eingesetzt. Diese bietet optimale Flexibilität und kann funktionell mit den Unternehmensprozessen wachsen. Da gerade bei kleinen Unternehmen oft das nötige Know-How für die Eigenentwicklung oder das nötige Kapital für die Entwicklung durch einen Dritten fehlen, wird häufig weiterhin auf die Verwendung günstiger Standardlösungen gesetzt. Dies wird dann problematisch, wenn die verwendete Software an ihre Leistungsgrenzen stößt und so schnell wie möglich ein Ersatz gefunden werden muss.

Da die Entwicklung von Individualsoftware bei nicht trivialen Unternehmensprozessen einige Zeit in Anspruch nimmt, wird das Wachstum des Unternehmens zunächst gedämpft. Allerdings wird die langfristige Wachstumsminderung durch Verwendung von nicht ausreichender Standardsoftware, im Vergleich zur Entwicklung von Individualsoftware, meist unterschätzt.

1.1 Motivation

Bei der Firma Profiteam Unternehmensberatung (Inh. Wolfram Sicks), im Folgenden „Profiteam“ genannt, wurde zu lange an bereits existierenden Lösungen der Firmen Microsoft und Google zur Bearbeitung der Geschäftsprozesse festgehalten. Durch immer komplexer werdende Geschäftsprozesse und damit einhergehende, wachsende Datenmengen, stoßen insbesondere die verwendete Tabellenkalkulation zur Kundenverwaltung sowie das verteilte Dateisystem Google-Drive an ihre Grenzen.

Das Unternehmen ist uns durch die, dieser Arbeit vorangegangen, Praxisphase bekannt, in der die Probleme bei der Auftragsverarbeitung bereits erkannt und der Geschäftsführung mitgeteilt wurden. Aufgrund dessen hat die Geschäftsführung des Unternehmens beschlossen mehr in die eigene IT-Infrastruktur zu investieren und uns angeboten, diese Abschlussarbeit in Kooperation mit dem Unternehmen zu absolvieren.

1.2 Aufgabenstellung und Zielsetzung

Im ersten Schritt soll nun eine Softwarelösung entwickelt werden, die die bestehende Kundenverwaltung, insbesondere im Hinblick auf Performance und Bedienbarkeit, verbessert. Im zweiten Schritt sollen alle im Kernprozess enthaltenen Rollen ihren je weiligen Teilprozess über die selbe Plattform bearbeiten können. Die vollständige Implementierung der Geschäftsprozesse in Software soll die Produktivität steigern und die Kommunikation vereinfachen. Im letzten Schritt soll die Erstellung von Energieberichten automatisiert werden, welche das zentrale Produkt des Unternehmens darstellen.

1.3 Aufbau der Arbeit

Diese Arbeit gliedert sich thematisch in zwei Teile. Im ersten Themenkomplex wird die Konzeptionierung und Implementierung einer Software zur Abbildung der Geschäftsprozesse betrachtet, im zweiten Themenbereich steht die Konzeptionierung und Implementierung eines Systems zur Erzeugung von BAFA-konformen Energieberichten, welches transparent in die zu entwickelnde Software inkludiert wird.

Zunächst werden die technologischen Grundlagen der Arbeit behandelt. Im Anschluss wird eine Analyse des aktuellen Systems durchgeführt und das Soll-Konzept definiert. Darauf aufbauend wird die Implementierung der Geschäftsprozesse innerhalb des Djangoprojekts erörtert. Abschließend wird das Ergebnis der Arbeit zusammengefasst und die zukünftige Entwicklung der Software betrachtet.

2 Grundlagen

Inhalt dieses Kapitels der Arbeit sind die technologischen Grundlagen, die für die Umsetzung benötigt werden.

2.1 Django Web Framework

In diesem Abschnitt werden die technologischen Grundlagen des Django Frameworks an Hand der frei zugänglichen Dokumentation sowie des offiziellen Django Tutorials erklärt.

2.1.1 Allgemeines

Django ist ein freies, quelloffenes, auf Python basiertes Web-Development Framework, das auf einem Model, Template, View (MTV)-System aufbaut, welches das geläufige Model-View-Presenter (MVP)-Entwurfsmuster implementiert[9]. Django ist 2005 unter einer 3-Klausel BSD-Lizenz veröffentlicht worden[10][11]. Das MTV-System teilt die Entwicklung einer Anwendung in die folgenden Bereiche auf:

- **Model:** Models in Django sind die Repräsentation der Daten im Programm. Hier werden auf einer abstrakten Ebene Attribute, Verhalten und Relationen zwischen Objekten festgelegt[19]. Über das Model eines Objektes kann, unabhängig von der tatsächlich verwendeten Datenbank, auf Daten zugegriffen werden.

- **Template:** Templates dienen bei Django der Präsentation der Daten, d.h. wie die Daten auf der Website angezeigt werden sollen[19]. Es handelt sich dabei um HTML-Dokumente, die mittels der Django Template Language variabel aufgebaut werden[8]. Es lässt sich erkennen, dass es sich um eine, funktional dem Präprozessor von C/C++ nicht unähnliche Sprache handelt, die verwendet wird, um den Aufbau eines HTML-Dokuments parametrierbar zu steuern.
  Darüber hinaus besteht die Möglichkeit Templates zu vererben, in anderen Templates zu inkludieren oder sie zur Laufzeit zu überschreiben[8]. Zur Laufzeit wird das Template auf Basis der vorhandenen Daten zu einem HTML-Dokument gerendert. Die gerenderten HTML-Dokumente werden im Anschluss von einem Webserver ausgeliefert.

- **View:** In den Views wird die Programmlogik implementiert. Eine View dient als Bindeglied zwischen Model und Template: Die View stellt die Daten der Models bereit, manipuliert sie und übergibt diese anschließend an das entsprechende Template[19].

2.1.2 Dateistruktur

Django gliedert die Entwicklung von Webanwendungen auch hinsichtlich der Dateistruktur in voneinander getrennte Bereiche. Im Folgenden ist der Aufbau der Dateistruktur eines kleinen Django Projekts zu sehen:

```
mysite/
    manage.py
    mysite/
        __init__.py
        settings.py
        urls.py
        wsgi.py
        polls/
                __init__.py
                admin.py
                apps.py
                migrations/
                        __init__.py
                models.py
                tests.py
                views.py
```

Listing 2.1: Dateistruktur eines Djangoprojekts [12]

Die in Listing 2.1 zu sehenden Dateien haben jeweils spezielle Bedeutungen für das Projekt. Im Folgenden werden zunächst die Dateien und Strukturen im übergeordneten Verzeichnis **mysite/** erklärt, im Anschluss erfolgt die Beschreibung für Dateien im Verzeichnis **polls/**. Die Beschreibung der Dateien und Ordner ist dem Django Tutorial entnommen[12]:

- Das *äußere* Verzeichnis **mysite/** dient lediglich als Container für das Django Projekt. Die Benamung spielt keine Rolle für die Anwendung.
- Das Skript **manage.py** ist ein Kommandozeilentool, das unterschiedliche Verwaltungsaufgaben für das Projekt erfüllt. So lässt sich beispielsweise ein integrierter Webserver auf der lokalen Maschine starten, der die Anwendung ausführt.
- Das *innere* Verzeichnis **mysite/** ist das tatsächliche Verzeichnis für das Python Paket des Projekts und wird verwendet, um Skripte die sich darin befinden an anderer Stelle zu inkludieren.
- Die Datei **mysite/__init__.py** ist eine leere Datei, die Python signalisiert, dass es sich bei dem Verzeichnis um ein Python-Paket handelt.
- In der Datei **mysite/settings.py** werden Konfigurationen für das Projekt vorgenommen. So werden hier beispielsweise Einstellungen zur verwendeten Datenbank, Apps etc. gesetzt.
- Die explizite Konfiguration der zu verwendenen URLs findet in der Datei **mysite/urls.py** mittels Regex statt.
- Die Datei **mysite/wsgi.py** dient als Einstiegspunkt für WSGI kompatible Webserver.

Die oben angegeben Dateien bilden das Grundgerüst eines Django Projekts, die Funktionalität ist in sog. Apps ausgelagert. Die Dateistruktur einer App wird im Folgenden anhand des in Listing 2.1 gezeigten Verzeichnisses **polls/** beschrieben:

- Auch hier findet sich die leere Datei **polls/__init__.py**, welche das Verzeichnis als Python-Paket markiert.
- In der Datei **polls/admin.py** werden Konfigurationen für die in Django integrierte Administrationsoberfläche definiert. So kann die Darstellung von zur App gehörenden Models auf der Administrations-Webseite spezifisch angepasst werden. Näheres ist dem Abschnitt Admin-Webseite zu entnehmen.
- Die Datei **polls/apps.py** dient der Deklarierung von Abhängigkeiten zu anderen Apps, die für die App „polls"gelten.
- Das Verzeichnis **polls/migrations/** nimmt eine besondere Rolle innerhalb der App ein, da der Inhalt i. d. R. vollständig automatisch generiert wird. Näheres zu Migrations findet sich im Abschnitt Datenbankmanagement und Migrations.
- In der Datei **polls/models.py** werden Definitionen für die appspezifischen Models gespeichert. Näheres zu Models findet sich im Abschnitt Models.
- In **polls/tests.py** werden Methoden definiert, um die eigene Anwendung auf die geplante Funktionsweise zu überprüfen.
- In der Datei **polls/views.py** sind die appspezifischen Views definiert.

Zusätzlich zu den genannten Dateien im Verzeichnis **polls/** können auch weitere, optionale Dateien erstellt werden. So ist es beispielsweise üblich zur App zugehörige Templates ebenfalls innerhalb des App-Verzeichnisses abzulegen und eine appspezifische **urls.py** zu schreiben, welche in der übergeordneten urls.py inkludiert wird.
Die Dateistruktur eines Djangoprojekts ist stark auf Modularisierung sowie *Plug and Play* von Apps ausgelegt.

2.1.3 Datenbankmanagement und Migrations

Django bietet eine Abstraktionsschicht zwischen Anwendungsdaten und Datenbankzugriffen. Durch die abstrakte Definition von Models (siehe auch Abschnitt 2.1.4) ist es möglich, die Konfiguration und Benutzung der Datenbank vollständig über Django zu steuern, ohne eine einzige manuelle SQL-Transaktion durchzuführen. Um dies zu ermöglichen, bedient sich Django dem Kommandozeilentool **manage.py**, welches unter anderem die Funktionalität der automatisierten Datenbankkonfiguration bietet. Die grundsätzliche Vorgehensweise gliedert sich hierbei in zwei Schritte, welche im Folgenden beschrieben werden:

2.1.3.1 Erzeugung von Migration-Files

Im appspezifischen Verzeichnis **migrations/** wird durch den Kommandozeilenbefehl *python manage.py makemigrations* automatisch eine Python Datei erzeugt, die der Umsetzung von Änderungen auf Datenbankebene dient, die in der Datei **models.py** gemacht wurden. Die erzeugte Datei baut dabei inkrementell auf vorherigen Migrations auf. Der in den Migration Files erzeugte Python Code ist gut lesbar und ermöglicht das Nachvollziehen von Änderungen an der Datenbankstruktur, was die Fehlersuche vereinfacht.
Bevor eine Änderung der Datenbank wirksam wird muss eine Migration ausgeführt werden.

2.1.3.2 Ausführen von Migrations

Durch den Kommandozeilenbefehl *python manage.py migrate* werden nun alle, noch nicht ausgeführten Migration-Files in der Reihnfolge ihrer Erstellung ausgeführt. Kommt es hierbei zu Problemen, bietet manage.py dem Entwickler diverse Interaktionsmöglichkeiten. So könnte beispielsweise bei der Migration eines neuen Feldes für ein bestehendes Model ein Fehler auftreten, wenn ein Feld das Attribut „blank=false“ besitzt und nicht angegeben wurde, wie mit bestehenden Datenbankeinträgen verfahren werden soll. Das Kommandozeilentool fängt diesen Fehler ab und bietet dem Entwickler die Möglichkeiten entweder einen einmaligen Default-Wert für exisitierende Models zu setzen oder aber den Migrationsvorgang abzubrechen und einen Default-Wert in **models.py** zu setzen. Django verhindert im Hinblick auf das oben angegebene Szenario also auch Fehler, die bei der Konfiguration der Datenbank entstehen können und zu Inkonsistenzen führen. Bevor eine Anwendung ordnungsgemäß funktionieren kann, müssen alle noch nicht angewandten Migration Files angewandt werden, damit im Sourcecode gemachte Änderungen auch für die Datenbank wirksam werden.

2.1.4 Models

Models werden in den appspezifischen **models.py** Dateien definiert und dienen als Abstraktionsebene zu den korrespondierenden Datenbanktabellen. Beispielhaft findet man unten stehend den Inhalt der **models.py** der App **Polls** aus dem Django Tutorial:

```
from django.db import models

class Question(models.Model):
    question_text = models.CharField(max_length=200)
    pub_date = models.DateTimeField('date published')


class Choice(models.Model):
    question = models.ForeignKey(Question, on_delete=models.CASCADE)
    choice_text = models.CharField(max_length=200)
    votes = models.IntegerField(default=0)
```

Listing 2.2: Inhalt der models.py der Polls App [13]

Jede in Listing 2.2 definierte Klasse bildet eine Subklasse von **django.db.models.Model**, jedes Feld wird durch die Instanz einer Feldklasse repräsentiert (z.B. Charfield und DateTimeField). Der angegebene Feldname wird von Django als Spaltenname für die Datenbank gesetzt. Der erste angegebene Parameter eines Feldes beschreibt den optionalen Namen des Feldes in einem für Menschen lesbaren Format („date published“ beim Feld pub_date). Manche Feldklassen benötigen einige, fest definierte Parameter. So ist es beispielsweise notwendig bei einem CharField die maximale Länge anzugebenen oder bei einem Fremdschlüssel (ForeignKey) die referenzierte Model-Klasse sowie das Verhalten beim Löschen der Referenz zu definieren (on_delete=models.CASCADE). Unter anderem lässt sich der Datei also entnehmen, dass eine **1:n** Relation zwischen Question und Choices besteht, eine Question maximal 200 Zeichen umfassen darf und beim Löschen der Question auch alle zugehörigen Choices gelöscht werden [13]. Die Trennung zwischen dem Verhalten von Objekten und der Anwendungslogik bietet insbesondere im Hinblick auf die Fehlervermeidung erhebliche Vorteile.

2.1.5 Views

Views dienen bei Django dem Bereitstellen von Webseiten, die einen speziellen Zweck erfüllen und ein spezielles Template benutzen. Jede View ist dabei eine Python-Funktion, die von Django ausgeführt wird, wenn die konfigurierte URL angefragt wird[14]. Die Definition einer View wird im Folgenden anhand der Index-Webseite der Polls App des Django Tutorials gezeigt:

```
from django.http import HttpResponse

from .models import Question


def index(request):
    latest_question_list = Question.objects.order_by('-pub_date')[:5]
    output = ', '.join([q.question_text for q in latest_question_list])
    return HttpResponse(output)
```

Listing 2.3: Auszug der views.py der Polls App [14]

In Listing 2.3 ist die Definition der View „index“ zu sehen. Der Funktion wird ein Objekt vom Typ request übergeben, welches hier nicht näher analysiert wird. Es ist allerdings möglich, diverse Metadaten aus dem request-Objekt zu extrahieren, und beispielsweise Entscheidungen auf Basis des anfragenden Browsers anders zu verarbeiten. Zunächst erfolgt die Erstellung einer Liste der neusten fünf Questions. Zu beachten ist, dass Django das darunter liegende, generierte SQL-Statement erst dann ausführt, wenn die Daten der Abfrage tatsächlich gebraucht werden. Dies ist sinnvoll, da beispielsweise ein Filtern der Questions nach einem bestimmten Kriterium in einem hyptothetischen nächsten Befehl unnötige Rechenlast auf dem Server erzeugen würde, da die Datenbankengine diese Aufgabe wesentlich effizienter erledigt als Python Code.
Im nächsten Schritt wird ein String generiert, der aus der Iteration über alle Questions im Ergebnis aus der vorherigen Zeile erzeugt wird. Erst zu diesem Zeitpunkt wird die generierte SQL-Abfrage ausgeführt. Abschließend wird der generierte String an den anfragenden Browser zurückgegeben. In der Praxis wird i.d.R. kein einfacher String zurückgegeben, sondern ein gerendertes Template, das die von der Datenbank abgefragten Daten ansprechend darstellt.

2.1.6 Django Template Language

Mit Hilfe der Django Template Language lassen sich HTML-Dokumente dynamisch generieren. Als Beispiel für ein solches Template dient die Index-Webseite der Polls App aus dem Django Tutorial:

```
{% if latest_question_list %}
    <ul>
    {% for question in latest_question_list %}
        <li><a href="/polls/{{ question.id }}/">{{ question.
            question_text }}</a></li>
    {% endfor %}
    </ul>
{% else %}
    <p>No polls are available.</p>
{% endif %}
```

Listing 2.4: index.html mit Django Template Language [14]

In Listing 2.4 ist ein HTML-Dokument zu sehen, in das Code der Django Template Language eingefügt wurde. Beim Rendern werden die Variablen über den Kontext von der View bereitgestellt und der Template Code ausgeführt. Der abgebildete Sourcecode erzeugt eine unnummerierte Liste mit den von der View übergebenen neuen Questions. Der Klick auf ein Listenelement führt dann zur Detailseite für die Question. Zu beachten ist hierbei, dass Template Language Befehle mit „{%“ beginnen und mit „%}“ enden. Stellen für einzusetzenden Text (oder auch HTML-Code) werden mit „{{“ begonnen und mit „}}“ beendet. Hier nicht dargestellt sind die Möglichkeiten, Methoden, Filter und anderen Code zur Renderzeit aus der Template Language heraus auszuführen.

2.1.7 Forms

Forms werden bei HTML dazu genutzt, Benutzereingaben zu strukturieren. Für HTML-Forms bestehen in Django Möglichkeiten die Generierung, Überprüfung und Verarbeitung von Forms in Templates über Python Code zu automatisieren. Django kann dabei nachfolgende Arbeitsschritte übernehmen:

- Verarbeitung und Restrukturierung von Daten, um sie für den Renderprozess vorzubereiten
- automatische Erstellung von HTML-Forms für die Daten
- Empfangen und Verarbeiten von eingereichten Forms und Daten

Um die genannten Schritte automatisch auszuführen, können u.a. sogenannte Form-Klassen erstellt werden[6]. Diese werden typischerweise in einer Datei namens **forms.py** definiert. Eine solche Datei könnte wie folgt aussehen:

```
from django import forms

class ContactForm(forms.Form):
    subject = forms.CharField(max_length=100)
    message = forms.CharField(widget=forms.Textarea)
    sender = forms.EmailField()
    cc_myself = forms.BooleanField(required=False)
```

Listing 2.5: forms.py: eine Form für ein Kontaktformular [6]

In Listing 2.5 ist die Definition einer HTML-Form für ein Kontaktformular aus Python Code zu sehen. Die Definition erfolgt ähnlich der eines Models, jedoch werden Felder von **django.forms** genutzt. Es existieren diverse, in Django integrierte Form-Felder, beispielsweise auch Date-Picker und Datei-Upload Felder. Des Weiteren ist es möglich, selbst neue Form-Felder zu definieren.

2.1.8 Nutzer und Gruppenverwaltung

Django bietet mit den Paketen **django.contrib.auth** und **django.contrib.contenttypes** integrierte Optionen zur Verwaltung von Benutzern, Gruppen und Zugriffsberechtigungen. Dabei werden Zugriffsberechtigungen je Model vergeben, was die Sicherheit der Anwendung zusätzlich erhöht. Die Verwaltung von Benutzern, Gruppen und Zugriffsberechtigungen erfolgt über die Administrationswebsite, welche im Abschnitt 2.1.9 erläutert wird. Die Vergabe von Rechten und Gruppen kann auch direkt in einer Python-Funktion erfolgen. Das Authentifizierungspaket ist individuell anpassbar und kann im Python-Code beispielsweise wie folgt verwendet werden:

```
from django.http import HttpRequest
from django.shortcuts import render
from django.core.exceptions import PermissionDenied

def addPolls(request):
    assert isinstance(request, HttpRequest)
    probe = request.user
    if probe.is_authenticated:
        if probe.groups.filter(name='Moderator').exists():
                if probe.has_perm('Polls.add_Question'):
                        return render(
                                request, 'add_polls.html',
                                {
                                        'creator' : probe.name,
                                }
                        )
    raise PermissionDenied()
```

Listing 2.6: Beispiel für die Verwendung des Authentifizierungspaketes [Eigene Darstellung]

In der in Listing 2.6 gezeigten View *addPolls* werden unterschiedliche Authentifizierungsabfragen ausgeführt. Zur besseren Übersichtlichkeit wurde der anfragende Nutzer (request.user) in der Variablen *probe* gespeichert und die IF-Abfragen, anstelle einer Verknüpfung der Bedingungen mit *and*, geschachtelt. Zunächst wird mittels **probe.is_authenticated** überprüft, ob der anfragende Benutzer angemeldet ist. Im Anschluss erfolgt die Überprüfung, ob der Benutzer der Gruppe *Moderator* angehört. Ist dem so, wird zusätzlich geprüft, ob die Berechtigung zum Erstellen von Questions vorhanden ist. Scheitert eine dieser Überprüfungen wird der Benutzer mit **raise permissionDenied()** auf eine HTTP-403 (Forbidden) Seite umgeleitet. Sind die Abfragen erfolgreich, wird das Template **add_polls.html** mit dem angegebenen Kontext gerendert.

2.1.9 Admin-Webseite

Django stellt eine integrierte Administrationswebseite zur Verfügung, die mit Minimalaufwand konfiguriert werden kann. Die Administrationswebseite ist standardmäßig unter der URL domainname.com/admin zu erreichen. Um die Administrationswebseite nutzen zu können, sind folgende Schritte notwendig:

1. Anlegen eines Administrator Accounts
2. Registrieren der Models, die auf der Admin Website bearbeitet werden sollen

Anlegen eines Administrator Accounts: Durch Eingabe des Kommandozeilenbefehls *python manage.py createsuperuser* wird ein Superuser angelegt. Nach erfolgreicher Eingabe von Benutzername, E-Mail-Adresse und Passwort, ist der Account einsatzbereit. Nach dem anschließenden, erfolgreichen Login ist initial folgende Ansicht zu sehen:

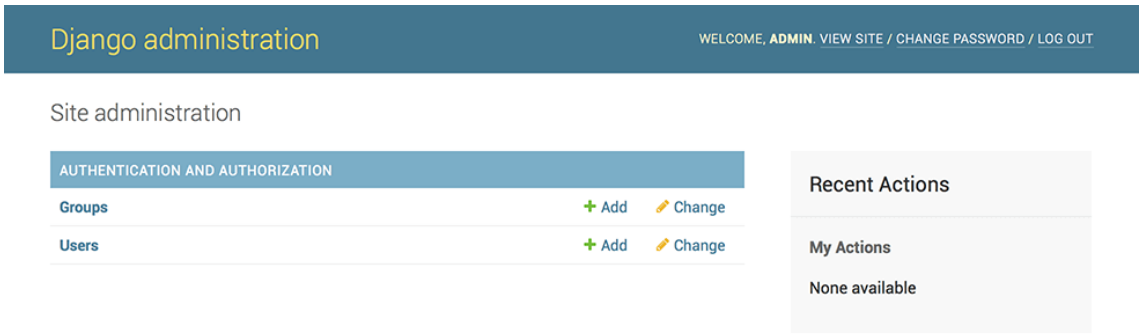

Abbildung 2.1: Initiale Ansicht der Admin Website nach erfolgreichem Login [13]

In Abbildung 2.1 ist die initiale Ansicht der Administrationswebsite zu sehen. Zunächst können nur Instanzen der Models *Group* und *User* hinzugefügt und geändert werden.

Registrieren der Models: Wenn Instanzen von Models anderer Apps über die Administratorwebsite editierbar sein sollen, so muss das entsprechende Model registriert werden. Dies geschieht über einen Einzeiler in der zur App gehörenden *admin.py*.

```
from django.contrib import admin

from .models import Question

admin.site.register(Question)
```

Listing 2.7: admin.py: Registrierung des Models "Question"[13]

In Listing 2.7 ist der Inhalt der admin.py für das Beispielprojekt **Polls** aus dem Django Tutorial zu sehen. Die Registrierung des Models *Question* erfolgt über die Zeile *admin.site.register(Question)*. Zusätzlich zum einfachen Registrieren eines Models und der damit verbundenen automatischen Generierung einer zugehörigen Form, kann auch eine eigene Form in **forms.py** definiert und analog hierzu registriert werden. Nach erfolgter Registrierung des Models, ist dieses auch auf der Administrationswebseite verfügbar:

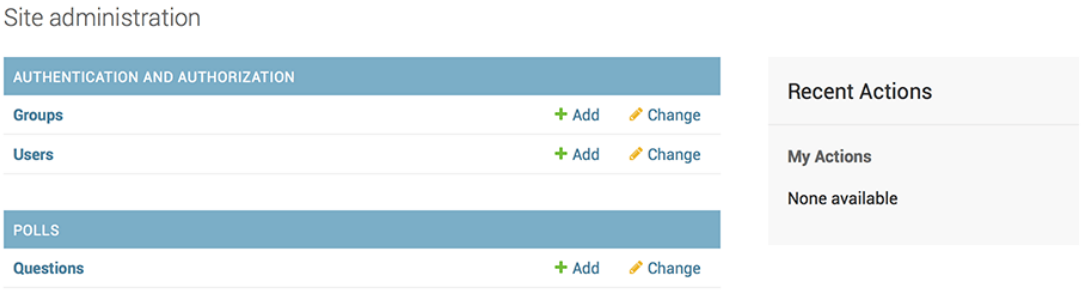

Abbildung 2.2: Ansicht der Administrationswebsite nach Registrierung der Question Models [13]

Über die Administrationswebseite können nun neue Questions erstellt, bearbeitet und gelöscht werden. Im Normalfall sind über die Administrationswebseite alle Felder eines Models editierbar, was allerdings auch vom Entwickler individuell angepasst werden kann.

2.1.10 Bereitstellung eines Projekts

Beim lokalen Testen des Projektes, kann der in Django integrierte Webserver mittels des Kommandozeilenbefehls *python manage.py runserver* gestartet werden. Das Projekt ist nun unter *http://127.0.0.1:8000* zu erreichen. Die Bereitstellung (Deploy) eines Django Projekts erfordert einen größeren Aufwand und ist für den durchschnittlichen Informatiker eine nicht triviale Aufgabe, da systemseitig bei falscher Konfiguration wenig bis keine Fehlermeldungen generiert werden. Der integrierte Webserver bietet weder die Sicherheitsaspekte noch die Geschwindigkeit eines dedizierten Webservers, weshalb ein Einsatz von Django in Produktionsumgebungen nur über einen separaten Webserver erfolgen sollte. Im Folgenden werden die Grundlagen der Bereitstellung unter Ubuntu 16.04 bei Verwendung des Apache2 Webservers kurz beschrieben. Die Installation des Apache2 Webservers ist nicht Teil der Beschreibung.

- Ausführen von *manage.py check –deploy* und beheben der ausgegebenen Warnings [15]
- Anlegen eines Virtual Enviroments (venv) im Projektordner mittels *virtualenv myProjectEnvName*
- Installation von mod_wsgi mittels *sudo apt-get install libapache2-mod-wsgi*
- Editieren der Apache-Konfiguration mit *sudo nano /etc/apache2/sites-available/000-default.conf* und Einfügen der aus Listing 2.8 entnehmbaren Zeilen in angepasster Form
- Anpassen der Zugriffsberechtigungen des Projektordners mit dem Kommando *chown*

Im folgenden Listing ist eine beispielhafte Konfigurationsdatei für den Apache Webserver gezeigt. Es ist darauf zu achten, dass die Pfade korrekt eingegeben werden und seitens des Betriebssystems Zugriffsberechtigungen für den Webserver auf die Verzeichnisse gesetzt sind, da keine verwendbaren Fehlermeldungen bei falscher Konfiguration erzeugt werden.

```
<VirtualHost *:80>
    [...]

    Alias /static /home/sammy/myproject/static
    <Directory /home/sammy/myproject/static>
        Require all granted
    </Directory>

    <Directory /home/sammy/myproject/myproject>
        <Files wsgi.py>
            Require all granted
        </Files>
    </Directory>

    WSGIDaemonProcess myproject python-home=/home/sammy/myproject/
        myprojectenv python-path=/home/sammy/myproject
    WSGIProcessGroup myproject
    WSGIScriptAlias / /home/sammy/myproject/myproject/wsgi.py

</VirtualHost>
```

Listing 2.8: Konfiguration des Apache Webservers für die Verwendung mit dem WSGI des Django Projekts [5]

In Listing 2.8 ist der Inhalt der Datei **/etc/apache2/sites-available/000-default.conf** zu sehen. Zunächst wird ein Alias für das /static Verzeichnis gesetzt, was dazu führt, dass der Webserver nun die angefragten statischen Resourcen in dem angegebenen Pfad sucht. Im Anschluss werden die Zugriffsrechte für den Projektordner und das Python WSGI gesetzt. Mit der *WSGIDaemonProcess* Direktive wird ein zusätzlicher Prozess im Daemon Mode definiert, der die Kommunikation zwischen Apache Webserver und der Python Anwendung über das WSGI übernimmt. Als Parameter werden hier der Name des Prozesses (myproject), das python home (in diesem Fall das venv des Projekts) und der Python-Path übergeben. Im Anschluss wird eine Prozessgruppe für den Daemon definiert, welche in diesem Beispiel aus Konsistenzgründen wie das Projekt heißt. Zuletzt wird für das WSGI die wsgi.py des Projekts mittels Alias als Root-Verzeichnis gesetzt. Dies führt dazu, dass Anfragen an die Root-Domain an wsgi.py weitegeleitet werden [5].

2.1.11 Funktionsweise

Die Bearbeitung eines HTTP-Request durch die Djangoanwendung gestaltet sich wie folgt: Der Webserver erhält vom Browser die URL der angefragten Seite. In der URL-Konfiguration von Django werden per Regular Expressions (Regex) URLs mit Views verknüpft. Dabei wird eine Liste von Verknüpfungen überprüft und bei der ersten Übereinstimmung die entsprechende View aufgerufen.
In der View kann nun das Objekt HTTP-Request genauer untersucht werden, also ob es sich z.B. um einen GET- oder POST-Typ handelt, welcher User die Anfrage gestellt hat, ebenso wie die IP-Adresse, Cookies und weitere Metadaten [16].

Im Anschluss werden die für das Rendern des zugehörigen Templates notwendigen Daten über die abstrakte Schnittstelle zur Datenbank erhoben. Hier wird auf explizite SQL-Abfragen verzichtet, da das Backend von Django automatisch über korrespondierende Python-Funktionen die SQL-Abfrage für die verwendete Datenbank erzeugt, ausführt und das Ergebnis als Liste von Objekten (Model Instanzen) zurückgibt. Diese Liste wird QuerySet genannt [7].

Zum Rendern wird von der View ein Context (Dictionary) an das Template übergeben. Im Context erfolgt die Zuweisung von in der View vorliegenden Daten zu den im Template benutzten Variablennamen.

Nach dem Rendern des Templates wird das erzeugte HTML-Dokument an die View zurückgegeben, welche dieses an den Webserver weitergibt.

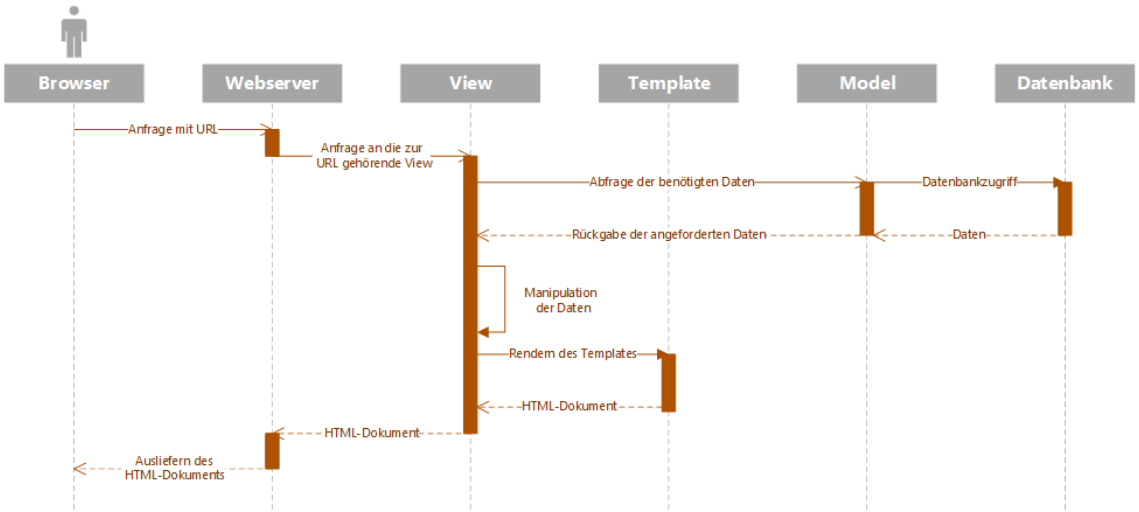

Abbildung 2.3: Zusammenarbeit der Komponenten bei der Anfrage einer Website durch den User [Eigene Darstellung]

Abbildung 2.3 zeigt die Kommunikation zwischen den Komponenten des Django Frameworks bei der Bearbeitung eines HTTP-Request. Zu beachten ist, dass nur Model, Template und View Bestandteile des Django-Frameworks sind. Die Objekte Browser, Webserver und Datenbank sind von Django unabhängig und austauschbar. Der URL-Dispatcher sowie die Funktion des WSGI wurden aus Gründen der Übersichtlichkeit nicht aufgenommen.

2.2 LaTeX

2.2.1 Allgemeines

LaTeX bezeichnet ein Textsatzsystem, welches das ursprüngliche TeX-System durch Makros vereinfacht. Es handelt sich nicht um „ordinäre Textverarbeitung“ [28]. Das Satzsystem Tex wurde von Donald Knuth von der Stanford University “zur Gestaltung schöner Bücher” [25, S. 1] entwickelt. Anfang der 80er Jahre begann Leslie Lamport LaTeX auf der Grundlage von Tex zu entwickeln [25, S. 2].
Layout und Inhalt werden voneinander abgegrenzt. Dies ermöglicht es dem Nutzer, sich auf den Inhalt und weniger auf das Format bzw. Design des Dokuments zu fokusieren. Die Einsatzmöglichkeiten von LaTex sind vielfältig: angefangen bei Artikeln, technischen und wissenschaftlichen Dokumentationen bis hin zu Büchern.

2.2.2 Funktionweise

Der Inhalt eines Dokumentes wird als Fließtext verfasst und um LaTeX-Befehle ergänzt. Durch Befehle können unter anderem Formatierungen vorgenommen sowie Graphiken, Tabellen und mathematische Formeln eingebunden werden.

2.2.2.1 Syntax

Prinzipiell haben die meisten LaTeX-**Befehle** die gleiche Form:

```
\befehlname[optionale Parameter]{verpflichtende Parameter}
```

Listing 2.9: Allgemeine Syntax eines LaTeX-Befehls [31]

Durch Befehle lassen sich z.B. Formatierungen (siehe Abschnitt 2.2.2.2) innerhalb eines Textes vornehmen. Durch die Befehle \begin{} und \end{} lassen sich **Umgebungen** erzeugen. Umgebungen definieren ein bestimmtes Verhalten innerhalb der Grenzen:

```
\begin{center}
Der Text innerhalb dieser definierten Umgebung wird zentriert
   dargestellt.
\end{center}
```

Listing 2.10: Beispiel für eine LaTeX-Umgebung [31]

Umgebungen werden beispielsweise für die Einbindung von Tabellen (siehe Abschnitt 2.2.6) und Graphiken (siehe Abschnitt 2.2.7) genutzt.

2.2.2.2 Formatierungsbefehle

Um Formatierungen wie in einem normalen Textverarbeitungsprogramm durchzuführen werden in LaTeX Befehle genutzt. Um einen punktuellen Fontwechsel während der Texteingabe festzulegen bedient man sich folgender Standardbefehle:

Tabelle 2.1: Standard-Fontwechselbefehle und Deklarationen [25, vgl. S. 355]

Befehl	**Aktion**
\textrm{...}	Text in Serifenschrift setzen
\textsf{...}	Text in serifenloser Schrift setzen
\texttt{...}	Test in `Schreibmaschinen-Schrift` setzen
\textmd{...}	Text in normaler Schriftstärke setzen
\textbf{...}	Text in **fett gedruckter** Schrift setzen
\textup{...}	Text in aufrechter Schrift setzen
\textit{...}	Text in *Kursivschrift* setzen
\textsl{...}	Text in *geneigter Schrift* setzen
\textsc{...}	Text in KAPITÄLCHEN setzen
\emph{...}	Text *hervorheben*
\textnormal{...}	Text in Grundschrift setzen

2.2.2.3 Makros

Durch das makrobasierte System von LaTeX ist es möglich Befehle und Umgebungen selbst zu definieren.
Werden bestimmte Befehlsfolgen häufig verwendet, ist es sinnvoll Makros bzw. Befehle für diese Fälle anzulegen. Dies geschieht über den \newcommand{}-Befehl:

```
\newcommand{befehl}[narg]{befehlsdefinition}
```

Listing 2.11: Definition eigener Befehle [25, S. 877]

Die Anzahl der Argumente ist durch $0 \leq narg \leq 9$ definiert.

Umgebungen sind äquivalent zu Befehlen zu defininieren:

```
\newenvironment{name}[narg]{begdef}{enddef}
```

Listing 2.12: Definition eigener Umgebungen [25, S. 880]

Befehle bzw. Umgebungen, die bereits existieren, können mit \renewcommand{} bzw. \renewenvironment{} neu definiert werden.

2.2.2.4 Dateitypen

LaTeX verwendet (schreibend und lesend) eine Vielzahl von Dateien beim Verarbeiten eines Dokumentes. Die wichtigsten Dateien bei einem LaTeX-Projekt sind die Quell-Dateien für die Eingabe (.tex-Dateien). I.d.R. existiert eine sog. Masterdatei, welche auf untergeordnete Dateien zugreift [25, S. 7].
Bei den .tex-Dateien handelt es sich um gewöhnliche "Textdateien", die mit einem beliebigen Texteditor bearbeitet werden können.
Tabelle 2.2 zeigt einen groben Überblick über die verwendeten Dateitypen und deren Funktion.

Tabelle 2.2: Übersicht über die von TEX und LATEX verwendeteten Dateitypen [25, S. 9]

	Dateityp	**Übliche Dateierweiterung(en)**
Dokumenteneingabe	Text	.tex .dtx .ltx
	Bibliographie	.bbl
	Index / Glossar	.ind / .gnd
Graphiken	interne	.tex
	externe	.ps .eps .tif .png .jpg .gif .pdf
Weitere Eingabedaten	Layout und Struktur	.clo .cls .sty
	Kodierungsdefinitionen	.def
	Sprachdefinitionen	.ldf
	Fontzugriffsdefinitionen	.fd
	Konfigurationsdaten	.cfg
Interne Kommunikation	Hilfsdaten	.aux
(Ein- und Ausgabe)	Inhaltsverzeichnis	.toc
	Abbildungs- / Tabellenverzeichnis	.lof / .lot
Low-Level TEX-Eingabe	Format	.fmt
	Fontmetriken	.tfm
Ausgabe	formatiertes Ergebnis	.dvi .pdf
	Protokoll	.log
Bibliographie BibTex)	Eingabe / Ausgabe	.aux / .bbl
	Datenbank / Stil / Protokoll	.bib / .bst / .blg
Index (MakeIndex)	Eingabe / Ausgabe	.idx / .ind
	Stil / Protokoll	.ist / .ilg

2.2.3 Aufbau eines Dokuments

Im Folgenden wird der grundlegende Aufbau eines LaTeX-Dokuments betrachtet.

2.2.3.1 Präambel

Jedes LaTeX-Dokument muss mit einer Dokumentenpräambel beginnen, welche Stilparameter definiert. Hierzu können Paket- oder Klassen-Dateien inkludiert werden. Es existieren fünf Standarddokumentenklassen bei LaTeX: article, report, book, slides und letter. Die Präambel muss sich über dem eigentlichen Inhalt des Dokuments befinden [25, S. 16].

2.2.3.2 Inhalt

Der Inhalt eines Dokuments befindet sich in folgender Umgebung unterhalb der Präambel:

```
\begin{document}
[...]
\end{document}
```

Listing 2.13: Umgebung des Inhalts eines LaTeX-Dokuments [Eigene Darstellung]

Innerhalb dieser Umgebung (gekennzeichnet mit [...]) können sowohl ein einfacher Textkörper stehen als auch .tex-Dateien inkludiert werden. Dadurch entsteht die Möglichkeit die Quelldateien aufzuteilen und einzelne Kapitel, Abschnitte oder sonstige Inhalte seperat auf verschiedene Dateien zu verteilen. Im Folgenden wird ein Beispiel für eine Aufteilung von LaTeX-Dateien aufgeführt:

```
\begin{document}
\include{Einleitung}
\include{Hauptteil}
\include{Schluss}
\end{document}
```

Listing 2.14: Aufteilung der .tex-Dateien [Eigene Darstellung]

Die Dateien “Einleitung.tex”, “Hauptteil.tex” und “Schluss.tex” werden durch den Befehl \include{} geladen und eingebunden. Die Präambel des Dokuments befindet sich lediglich in der “Masterdatei”.

2.2.3.3 Ausgabe

Durch Kompilieren der “Masterdatei” erhält man die formatierte Ausgabe. Dies geschieht üblicherweise durch das Kommando *pdflatex document*, wobei es sich bei *document* um den entsprechenden Dateinamen handelt.
Dies kann über die Konsole oder mittels einer GUI eines LaTeX-Editors geschehen. Statt pdflatex kann auch latex als Kommando verwendet werden. Beide Befehle nutzen das gleiche Programm: pdftex. *latex* erzeugt .dvi-Dateien, *pdflatex* erzeugt eine .pdf-Datei. Welches Kommando genutzt wird hängt von der Anwendung ab, wobei auch eine Kombination der Befehle möglich ist. Es ist zu beachten, welche Grafik-Dateitypen inkludiert werden sollen. Die gängigen Formate werden nur von pdflatex unterstützt [29]:

- latex: ps, eps, tiff
- pdflatex:jpeg, png, pdf

2.2.4 Gliederung und Dateistruktur

Ein LaTeX-Dokument kann wie folgt gegliedert werden:

```
\documentclass{scrreprt}

\begin{document}

\chapter{Kapitel 1}
Dies ist ein erstes Beispielkapitel.

\section{Abschnitt 1}
Durch die Section entsteht ein Abschnitt.

\subsection{Abschnitt 2}
Eine Subsection ist ein Abschnitt, der eine Ebene tiefer steht.

\end{document}
```

Listing 2.15: Beispielhafte Gliederung eines LaTeX-Dokuments [Eigene Darstellung]

Wird der Code aus Listing 2.15 mit pdflatex kompiliert, so wird folgende .pdf erzeugt:

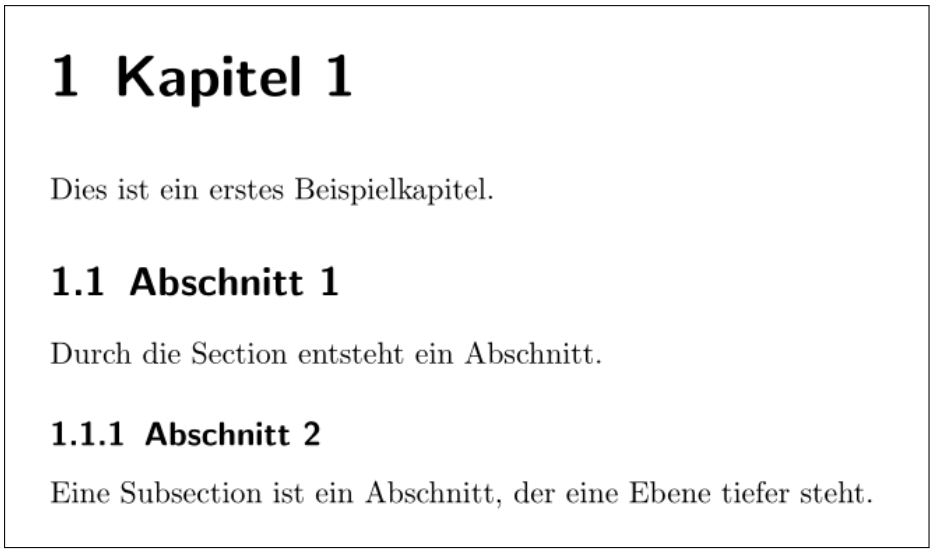

Abbildung 2.4: Beispiel einer erstellten PDF mit LaTeX [Eigene Darstellung]

Die wichtigsten Gliederungen sind:

- chapter
- section
- subsection
- subsubsection
- paragraph

Diese sind äquivalent zu Listing 2.9 anzugeben, z.B. *\subsubsection{Titel}*.
Die Dokumentenklasse "scrreprt" in der Dokumentenpräambel in Listing 2.15 ist keine Standarddokumentenklasse von LaTeX, sondern eine Klasse aus einem sog. KOMA-Script.

"KOMA-Script ist eine Sammlung von Klassen und Paketen für LaTeX [...] und existiert seit 1994. [...] Ziel des Projekts war eine neue Dokumentation zu erstellen und zu pflegen." [22]
Darüber hinaus existieren noch weitere KOMA-Scripte, die die ursprünglichen Klassen ergänzen [25, S. 244]:

- scrartcl
- scrbook
- scrlttr2

2.2.4.1 Verzeichnisse

Die wichtigsten Verzeichnisse sind Inhalts-, Abbildungs-, und Tabellen-Verzeichnis. Diese Verzeichnisse können von LaTeX automatisch erzeugt werden. Hervorzuheben ist das Inhaltsverzeichnis, die *Table Of Contents (TOC)*. Darin werden die Titel aller Abschnitte mit entsprechender Seitennummer hinterlegt [25, S. 48]. Durch Konfiguration kann die Formatierung angepasst werden, beispielsweise bis zu welcher Ebene die Kapitel bzw. Abschnitte im Verzeichnis enthalten sein sollen.
Die beiden anderen Verzeichnisse verhalten sich ähnlich. Die Informationen zu den *Lists Of Figures (LOF)* und *Lists of Tables (LOT)* werden in den entsprechenden .lof- und .lot-Dateien angelegt. Dazu gehört die jeweilige Beschreibung der Abbildung bzw. der Tabelle sowie die Angabe von Kapitelnummer und Seitenzahl.
Beschreibungen werden mit dem Befehl *\caption{Beschreibung der Abbildung bzw. der Tabelle}* eingefügt.
Um die Verzeichnisse in das Dokument einzubinden werden folgende Befehle verwendet:

- \tableofcontents
- \listoffigures
- \listoftables

Diese sind entweder am Anfang der Datei oder in einer seperaten Konfigurationsdatei zu nutzen.

2.2.5 Referenzen und Sprungmarken

Um Querverweise innerhalb eines Dokuments zu nutzen, stehen folgende Standard-Befehle zur Verfügung:[25, S. 70]

- \label{schlüssel}
- \ref{schlüssel}
- \pageref{schlüssel}

Ein Label mit selbst gewähltem Schlüssel (Zeichenkette) wird an die Stelle gesetzt, auf die man verweisen möchte, z.B. auf einen Abschnitt.
Mit \ref{} innerhalb eines Textes kann dann die entsprechende Stelle referenziert werden, \pageref{} referenziert die entprechende Seite:

```
\section{Beispielabschnitt} \label{abschnitt1}
Innerhalb des Dokumentes kann nun beispielsweise auf den Abschnitt
\ref{sec:abschnitt1} auf Seite \pageref{sec:abschitt1} verwiesen
    werden.
```

Listing 2.16: Beispiel für eine Referenzierung [Eigene Darstellung]

Der Code führt zu folgender Ausgabe:

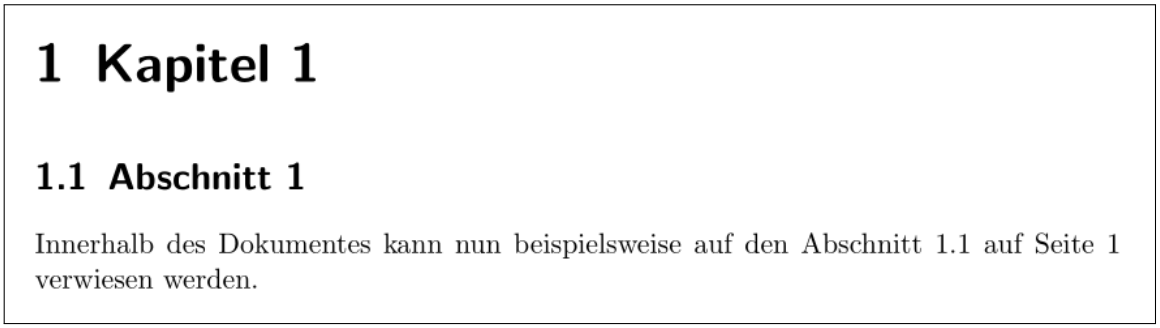

Abbildung 2.5: Referenzierungen mit LaTeX [Eigene Darstellung]

Nach dem gleichen Prinzip können auch Bilder, Tabellen oder sonstige Inhalte des Dokumentes mit einem Label versehen werden.

2.2.6 Tabellen

Für die Erstellung von Tabellen gibt es diverse Pakete, die verschiedene Umgebungen importieren. LaTeX verfügt über zwei Standard-Umgebungen für Tabellen: die häufig genutzte tabular-Umgebung und die tabbing-Umgebung [25, S. 247].
Die Syntax für die tabular-Umgebung lässt sich wie folgt darstellen:

```
\begin{tabular}[pos]{spalten-dekl}
        [...]
\end{tabular}
```

Listing 2.17: Syntax für eine tabular-Umgebung [25, vgl. S. 247]

Die Spalten-Deklaration bestimmt die Spaltenanzahl und Ausrichtung:[25, vgl. S. 251]

- c: Zentrierte Spalte
- l: Linksbündige Spalte
- r: Rechtsbündige Spalte
- |: Vertikale Trennlinie der Spalten

Das folgende Beispiel veranschaulicht die Syntax:

```
\begin{tabular}{|c|c|}
        \hline
        Zeile 1.1 & Zeile 1.2 \\
        \hline
        Zeile 2.1 & Zeile 2.2 \\
        \hline
\end{tabular}
```

Listing 2.18: Beispiel für eine Tabelle mit tabular [Eigene Darstellung]

Zeile 1.1	Zeile 1.2
Zeile 2.1	Zeile 2.2

Abbildung 2.6: Erzeugung einer einfachen Tabelle mit tabular [Eigene Darstellung]

Wird das Beispiel in Listing 2.18 kompiliert, wird die Ausgabe 2.6 erzeugt. Die horizontalen Trennlinien werden durch \hline zwischen den Zeilen eingefügt. Durch den &-Operator werden die einzufügenden Zellenihnalte voneinader abgegrenzt.

2.2.7 Grafiken

Zu inkludierende Abbildungen bzw. Grafiken sollten sich in einer figure-Umgebung befinden. Dadurch ist es möglich, eine Abbildungsbeschreibung hinzuzufügen und diese im Abbildungsverzeichnis einzufügen (siehe Abschnitt 2.2.4.1).

```
\begin{figure}
 \includegraphics[pos]{path}
 \caption{Beschreibung der Abbildung}
\end{figure}
```

Listing 2.19: Abbildungen mit includegraphics inkludieren [Eigene Darstellung]

\includegraphics kann sowohl mit dem *graphic* als auch mit dem *graphicx* Paket genutzt werden, allerdings mit unterschiedlichen Paramteren [4, S. 9].
Graphicx stellt eine Weiterntwicklung zum ursprünglichen Standard-Paket *graphics* dar [4, S. 6].

2.2.8 Bedingte Ausführung

Wie aus tatsächlichen Programmiersprachen bekannt, gibt es auch in TeX die Möglichkeit if-Abfragen zu nutzen. Zwar existieren auch Pakete für LaTex, welche die Nutzung von if-Abfragen und Schleifen vereinfachen sollen, allerdings reichen die ursprünglichen TeX-Befehle für die Anwendungen dieser Arbeit vollkommen aus.

```
\if
[...]
\else
[...]
\fi
```

Listing 2.20: Allgmeine Syntax für If-Abfragen in TeX [Eigene Darstellung]

Das Listing 2.20 zeigt die Struktur mit der if-Abfragen in TeX genutzt werden können. Die gängigsten vordefinierten if-Abfragen in TeX sind:[18, vgl. S. 15f][27]

- \ifnum
  - prüfe auf einen Integer-Wert oder Vergleich zweier Integer
- \ifdim
  - Dimensions- bzw. Längenwerte können verglichen werden
- \ifx
  - Vergleich zweier Makros möglich
- \iftrue
  - Prüfung, ob ein Makro existiert
- \iffalse
  - Prüfung, ob ein Makro nicht existiert

2.2.9 Pakete

In diesem Abschnitt werden die Pakete aufgeführt, die für den praktischen Teil der Dokumentenautomatisierung mit LaTeX wichtig sind.

2.2.9.1 fancyhdr

Das Paket *fancyhdr* dient der einfachen Gestaltung von Kopf- und Fußzeilen.
Nach der Dokumentenpräambel werden folgende Befehle benötigt:

```
\usepackage{fancyhdr}
\pagestyle{fancy}
```

Listing 2.21: Befehle für das Paket fancyhdr [26, S. 5]

Das folgende Layout kann mit fancyhdr erzeugt und geändert werden:

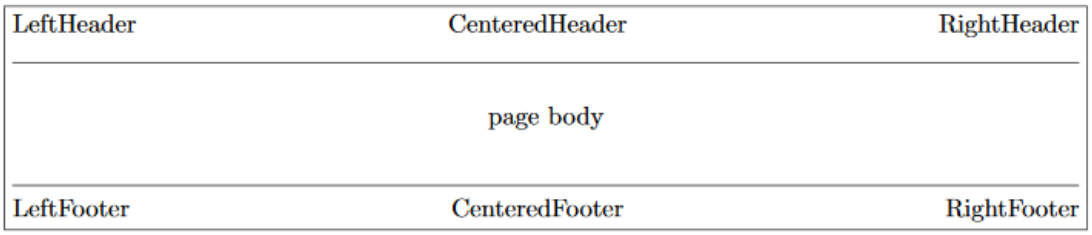

Abbildung 2.7: Seitenlayout, welches mit fancyhdr und geändert werden kann [26, S. 5]

2.2.9.2 titlesec

titlesec nutzt die vorhandenen Makros und überschreibt diese teilweise oder komplett. Mit Hilfe dieses Pakets kann beispielweise das Format einer Überschrift eines Abschnitts in wenigen Schritten bzw. mit wenig Aufwand angepasst werden.
Mit dem Befehl *titleformat* kann das Layout einer Überschrift wie folgt geändert werden:

```
\titleformat{<command>}[<shape>]{<format>}{<label>}{<sep>}{<before-
    code>}[<after-code>]
```

Listing 2.22: Syntax des Befehls titleformat [3, S. 4]

Das Attribut *command* bezeichnet die Art der Überschrift (Chapter, Section usw.). Die restlichen Optionen dienen dem Layout.
Nähere Informationen zu Optionen und den verschiedenen Funktionen von *titlesec* ist der Dokumentation des Pakets zu entnehmen.[1]

[1] http://ctan.math.utah.edu/ctan/tex-archive/macros/latex/contrib/titlesec/titlesec.pdf

2.2.9.3 TIkZ

Um ein Diagramm in LaTeX mit dem Paket TIkZ zu erzeugen wird eine Umgebung benötigt:[30, vgl. S. 31]

```
\usepackage{tikz}
\begin{tikzpicture}
\draw(-1.5,0)--(1.5,0);
\draw(0,-1.5)--(0,1.5);
\end{tikzpicture}
```

Listing 2.23: Umgebung des Pakets TIkZ [30, S. 31]

Das Beispiel in Listing 2.23 erzeugt ein einfaches Diagramm, welches aus einem unbeschrifteten Koordinatensystem besteht.
Mit Hilfe dieser Umgebung können diverse Diagramme erstellt werden, sowohl einfache Funktionen als auch animierte Objekte [30].

2.2.9.4 Weitere Pakete

In diesem Abschnitt werden weitere Pakete aufgelistet; eine nähere Beschreibung entfällt aufgrund der Übersichtlichkeit und der Menge an genutzten Paketen.

- lscape
  - stellt die Umgebung *landscape* bereit, um Teile des Dokuments quer darzustellen
- setspace
  - dient der Änderung des Zeilenabstands, sowohl mit Hilfe von Umgebungen als auch einfachen Makros
- lastpage
  - bietet Funktionen, um die gesamte Seitenanzahl eines Dokuments zu extrahieren
- geometry
  - erlaubt das Seitenlayout zu ändern, z.B. Seitenränder, Abstände etc.
- xcolor
  - ermöglicht einfache Farbänderungen, z.B. Schriftfarbe, Hintergrund etc.
- wrapfig
  - erlaubt es, einzufügende Bilder textumflossen darzustellen

3 Analyse der Auftragsverarbeitung

In diesem Kapitel wird der IST-Zustand der Auftragsverarbeitung in der Firma Profiteam erläutert und ein SOLL-Konzept zur Optimierung erarbeitet.

3.1 Das Unternehmen

Die Firma Profiteam Unternehmensberatung (Inhaber: Wolfram Sicks) ist ein mittelständisches Unternehmen mit neun Mitarbeitern, das sich als Kernkompetenz mit dem Prozessmanagement bei der Durchführung von Energieberatungen profiliert. Profiteam selbst ist Kooperationspartner und Subunternehmer der SGS-TüV Saar GmbH, welche den Themenbereich Energieberatung an die Firma Profiteam ausgelagert hat. Die von Profiteam angebotene Dienstleistung „Energieberatung Mittelstand“ ist aktuell Gegenstand einer Förderung durch das BAFA.[1] Im Zuge der Neuentwicklung eines softwarebasierten Systems für Profiteam, das auf die Dienstleistung „Energieberatung Mittelstand“ zugeschnitten ist, werden im Abschnitt IST-Zustand die Rollen und Aufgaben von Mitarbeitern und Subunternehmern des Unternehmens analysiert. Auf Basis dieser Analyse werden im darauffolgenden Abschnitt SOLL-Konzept Muss- und Kann-Kriterien für die zu entwickelnde Software definiert.

3.2 IST-Zustand

3.2.1 Außendienst

Die Firma Profiteam bedient sich zahlreicher externer Vertriebsagenturen, die unabhängig auf Honorarbasis tätig werden.

3.2.1.1 Aufgaben

Die externen Mitarbeiter im Außendienst haben folgende Aufgaben:

- Akquise von Neukunden
- Übergabe von unterschriebenem Vertrag und Vollmacht an Profiteam
- Übergabe des fertigen Energieberichts an den Kunden durch die authorisierte Agentur des Außendienstmitarbeiters
- Rechnungsstellung an Profiteam

3.2.1.1.1 Akquise

Die externen Mitarbeiter im Außendienst nutzen jeweils eigene, nicht standardisierte Systeme um potenzielle und tatsächlich akquirierte Kunden zu verwalten.

3.2.1.1.2 Übergabe von Dokumenten

Die Übergabe der Unterlagen eines akquirierten Kunden erfolgt nicht standardisiert via E-Mail, Fax oder durch ein anderes Medium an das Backoffice-Team. Es exisitiert keine automatische Überprüfung, ob die übergebenen Unterlagen angekommen und vollständig sind.

3.2.1.1.3 Übergabe des Energieberichts

Nach erfolgter Beratung wird dem Kunden der Energiebericht durch die authorisierte Agentur des Außendienstmitarbeiters überreicht. Die Übergabe des Berichts an den Außendienstmitarbeiter erfolgt standardisiert vom Backoffice-Team per E-Mail. Anschließend erfolgt die Rückmeldung durch den Außendienstmitarbeiter an das Backoffice-Team über das standardisierte Formular „Übergabebestätigung“, da das Datum der Berichtsübergabe festgehalten werden muss.

3.2.1.1.4 Rechnungsstellung

Je nach Leistung der Agentur[1] werden durch Profiteam unterschiedliche Vergütungssätze gezahlt. Die Außendienstmitarbeiter stellen nach erfolgter Energieberatung eine Rechnung in entsprechender Höhe an Profiteam. Die Übergabe der Rechnung an Profiteam erfolgt i.d.R. via E-Mail.

3.2.1.2 Systematische Probleme

Zusammenfassend sind nachfolgende Probleme zu erkennen oder wurden durch die Außendienstmitarbeiter oder das Backoffice-Team angemerkt:

- Die Außendienstmitarbeiter haben lediglich beschränkten Zugriff auf das Dateisystem[2] und keinen Zugriff auf die Kundenverwaltungstabelle von Profiteam, weshalb es häufig zu telefonischen Rückfragen kommt.
- Die Verwaltung potenzieller und tatsächlich akquirierter Kunden gestaltet sich als schwierig, da die Agenturen i.d.R. auch für andere Unternehmen tätig sind.
- Die Übergabe von Dokumenten erfolgt nicht automatisiert und standardisiert, was im Backoffice zu erhöhtem Verwaltungsaufwand führt.
- Da zwischen Kundenakquise und Übergabe des Energieberichts mehrere Monate liegen können, fehlt den Außendienstmitarbeitern die Rückmeldung über den Zustand des Auftrags.
- Für die Rechnungsstellung verwaltet Profiteam für jede Agentur eine Provisionsliste, auf der aufgelistet ist, über welchen Betrag der Außendienstmitarbeiter die Rechnung für den zurückliegenden Monat stellen kann. Diese kann der Außendienstmitarbeiter einsehen und die Rechnung auf Basis dieser Tabelle erstellen. Die Rechnungsstellung ist nicht automatisiert.

[1] Anzahl akquirierter Neukunden je Monat
[2] Nur die Ordner der von ihnen akquirierten Kunden sind ihnen zugänglich

3.2.2 Backoffice

Die Verwaltung der Aufträge wird von einem kleinen Backoffice-Team durchgeführt, das auch produktive Tätigkeiten wahrnimmt.

3.2.2.1 Aufgaben

Die Mitarbeiter im Backoffice haben folgende Aufgaben:

- Verwaltung der Tabellenkalkulation zur Kundenverwaltung
- Anlegen von Neukunden und Stellen von Förderanträgen
- Verwaltung des Dateisystems
- Überwachung der Tabellenkalkulation
- Disposition der Sachverständigen
- Ansprechpartner bei Fragen zu Aufträgen
- Überwachung des Prozessfortschritts bei Aufträgen
- Überwachung der durch das BAFA gesetzten Fristen für Zuwendungen
- Korrekturlesen von Energieberichten

3.2.2.1.1 Auftragsverwaltungssystem

Zur Verwaltung der Aufträge wird ein Google Spreadsheet[3] verwendet. In diesem Spreadsheet existiert für jeden Kunden eine Zeile, in der mehr als 70 Datenfelder vorhanden sind. Die Tabelle umfasst zum Zeitpunkt der Erstellung dieser Arbeit knapp 800 Kundeneinträge. Da die Tabelle vollständig von Hand geführt wird, existiert kein Fehlerabfang gegen falsche oder unplausible Eingaben. Zum 01.06.2018 wurde der Prozess umstrukturiert, was zu einem neuen Registerblatt in der Kundenverwaltungstabelle führte. Neukunden werden nun in diesem Registerblatt angelegt. Das alte, ursprüngliche Registerblatt wird parallel weitergeführt, bis die alten Aufträge fertiggestellt sind.

3.2.2.1.2 Anlegen von Neukunden

Nach Übergabe der notwendigen Dokumente durch den Außendienst wird für den Kunden eine neue Zeile in der Kundenverwaltungstabelle angelegt und die Daten aus den erhaltenen Dokumenten werden in die Tabelle übertragen. Zusätzlich wird für den Neukunden ein Ordner im verteilten Dateisystem Google Drive angelegt, in welchem sämtliche Dokumente zum Auftrag gespeichert werden. Im Anschluss wird mit der Vollmacht des Kunden ein Förderantrag beim BAFA für eine „Energieberatung Mittelstand"gestellt. Die Tabellenkalkulation ist nur den angestellten Mitarbeitern zugänglich. Innerhalb der Ordnerstruktur existieren Zugriffsberechtigungen auf die Kundenordner für die beteiligten Akteure.

[3] Eine Tabellenkalkulation, die von mehreren Usern gleichzeitig über einen Browser benutzt werden kann

3.2.2.1.3 Verwaltung des Dateisystems

Das bereits erwähnte verteilte Dateisystem Google Drive wird von den Mitarbeitern im Backoffice gepflegt. Für jeden Kunden werden sämtliche E-Mail- Korrespondenz sowie alle zum Auftrag gehörenden Dokumente manuell in der Ordnerstruktur abgelegt.

3.2.2.1.4 Überwachung der Tabellenkalkulation

Da die Tabelle zur Auftragsverwaltung von Hand geführt wird, ist es notwendig, zyklisch manuell auf Fehler zu prüfen. Dies geschieht über die Filter-Funktionalität der Tabellenkalkulation. Es werden zyklisch[4] Filterkombinationen angewandt, die nicht plausible Zeilen ausgeben. Welche Filterkombinationen angewandt werden, ist nicht standardisiert.

3.2.2.1.5 Verteilung der Aufträge an Ingenieure

Nach erfolgreichem Anlegen eines Neukunden wird ein Ingenieur bzw. Ingenieurbüro beauftragt, die Energieberatung durchzuführen. Dies geschieht durch das Verschieben des Kundenordners in den persönlichen Ordner des Ingenieurs in Google Drive und dem Versand einer E-Mail an den Ingenieur. Kriterien für die Zuweisung eines Auftrags sind u.a. die Distanz zum Kunden und die Auslastung des Ingenieurs.

3.2.2.1.6 Ansprechpartner bei Fragen zu Aufträgen

Aufgrund des beschränkten Zugriffs auf Dateisystem und Kundenverwaltung durch Außendienstmitarbeiter und Ingenieure kommt es häufig zu Rückfragen, die vom Backoffice i.d.R. telefonisch beantwortet werden müssen.

3.2.2.1.7 Überwachung des Prozessfortschritts bei Aufträgen

In der Kundenverwaltungstabelle wird der aktuelle Prozessstatus durch farbliches Markieren des Kundennamens signalisiert. Diese Einfärbung erfolgt manuell. Da für die Auftragsdurchführung sowohl interne als auch extern gesetzte Fristen gelten, ist es notwendig, zyklisch auf künftige Fristabläufe zu prüfen. Dies geschieht unregelmäßig und manuell über die Filterfunktion der Tabellenkalkulation. Bei Unstimmigkeiten bzw. demnächst ablaufenden Fristen informiert das Backoffice-Team den entsprechenden Teilnehmer und klärt die Ursache für das Problem.

3.2.2.1.8 Korrekturlesen von Energieberichten

Um einen einheitlichen Qualitätsstandard beim Kernprodukt Energiebericht zu gewährleisten, werden alle von den Sachverständigen gefertigten Energieberichte vor Weitergabe an die Außendienstmitarbeiter Korrektur gelesen. Tipp- und Syntaxfehler werden manuell korrigiert. Eine Überprüfung der Korrektheit von Berechnungen findet nicht statt.

[4] i. d. R. einmal pro Woche

3.2.2.2 Systematische Probleme

Zusammenfassend sind nachfolgende Probleme zu erkennen oder wurden durch das Backoffice-Team angemerkt:

- Die Performance der Filterfunktion innerhalb der Tabellenkalkulation ist ungenügend.
- Daten, die an anderer Stelle vorhanden sind und nicht als Redundanz benötigt werden, müssen im Zuge der Auftragsbearbeitung kopiert werden.
- Die manuelle Verwaltung des Dateisystems ist nicht praktikabel.
- Das Dateisystem umfasst mittlerweile über 60 GB, die auf den Rechnern aller berechtigten Personen redundant vorhanden sind.
- Die Beantwortung von Anfragen zum Auftragsstatus kostet zusätzliche Zeit. Die notwendigen Zugriffsberechtigungen machen die Vorgehensweise beim aktuellen System allerdings alternativlos.
- Das Controlling der Kundenverwaltungstabelle ist nicht automatisiert und unzuverlässig.

3.2.3 Sachverständige

Profiteam beauftragt zahlreiche angestellte Sachverständige sowie externe Ingenieure und Ingenieurbüros mit der Durchführung der „Energieberatung Mittelstand“.

3.2.3.1 Aufgaben

Die Sachverständigen haben nachfolgende Aufgaben:

- Rückmeldung über die Annahme eines zugewiesenen Auftrags
- Terminvereinbarung zur Ortsbegehung
- Ortsbegehung
- Anfertigen des Energieberichts
- Übergabe des Energieberichts an das Backoffice
- Rechnungsstellung

3.2.3.1.1 Rückmeldung über die Annahme eines zugewiesenen Auftrages

Der Sachverständige quittiert die Zuweisung eines Auftrages entweder durch E-Mail an das Backoffice-Team, via Telefon oder bei angestellten Sachverständigen auch durch persönliche Absprache. Das Datum der Auftragsannahme wird in der Auftragsverwaltungstabelle gespeichert.

3.2.3.1.2 Terminvereinbarung zur Ortsbegehung

Der Sachverständige vereinbart mit dem Kunden einen Termin zur Ortsbegehung und Datenaufnahme. Der Termin soll dem Backoffice mitgeteilt werden und in der Auftragsverwaltungstabelle gespeichert werden. In der Praxis geschieht dies allerdings nicht.

3.2.3.1.3 Ortsbegehung

Im Zuge der Ortsbegehung werden vom Sachverständigen die technischen Daten von Maschinen und Anlagen, der Gesamtenergieverbrauch, die Energiekosten sowie weitere Rahmenbedingungen ermittelt und nicht standardisiert notiert. Es werden Fotos von Räumen und Typenschildern gefertigt.

3.2.3.1.4 Anfertigen des Energieberichts

Im Anschluss an die Ortsbegehung fertigt der Sachverständige auf Grundlage seiner Notizen und Fotografien den Energiebericht an. Zur Erstellung bedient sich der Ingenieur einer Sammlung von in Excel erstellten Tools, die der Erstellung von Tabellen und Diagrammen dienen. Aus den jeweiligen Tools werden die Diagramme und Tabellen extrahiert und in ein Word-Dokument eingefügt. Der Sachverständige vervollständigt den Energiebericht mit persönlichen Empfehlungen. Die Erstellung des Energieberichts ist nicht standardisiert.

3.2.3.1.5 Übergabe des Energieberichts an das Backoffice

Nach Abschluss der Erstellung des Energieberichts legt der Sachverständige den Bericht im Kundenordner ab und benachrichtigt das Backoffice via E-Mail.

3.2.3.1.6 Rechnungsstellung

Die externen Sachverständigen stellen nach erfolgter Berichtfertigstellung eine Rechnung an Profiteam. Die Höhe der Rechnung ist abhängig von der Anzahl der aufgewendeten Arbeitstage und dem Honorarsatz des Ingenieures bzw. des Ingenieurbüros.

3.2.3.2 Systematische Probleme

Zusammenfassend sind folgende Probleme zu erkennen oder wurden von den Sachverständigen selbst angemerkt:

- Die Zuweisung von Aufträgen erfolgt teilweise unkoordiniert.
- Die Erstellung des Energieberichts erfordert ein hohes Maß an Kopiervorgängen zwischen unterschiedlichen Dateien.
- Vom Sachverständigen aufgenommene Daten sind nicht wieder auffindbar[5].
- Die Rechnungsstellung ist nicht automatisiert.

3.2.4 Geschäftsführung

Die Geschäftsführung von Profiteam ist nicht direkt in die Auftragsverarbeitung eingebunden und übernimmt Managementaufgaben sowie die Rechnungsverwaltung. Durch die Verwendung des Spread-Sheets in der Auftragsverwaltung ist es für das Management schwierig strategische Entscheidungen auf Basis der vorhandenen Daten zu treffen. D.h. die benötigten Daten sind vorhanden, aber nicht effektiv nutzbar. Es gibt wenige bis keine aussagekräftige, statistische Auswertungen.

3.2.5 Zusammenfassung

Die bisher genutzten Softwarelösungen und Prozessabläufe bieten nicht mehr die nötige Effizienz, um ein weiteres Wachstum des Unternehmens zu ermöglichen. Durch die Verwendung von Google Drive, der parallelen Nutzung von unterschiedlichen Versionen der gleichen Tools sowie der suboptimalen Zugriffsrechte auf Dateien und Ordnerstrukturen, entstehen vermeidbare Probleme. Die Kommunikation zwischen den Prozessbeteiligten verläuft nicht standardisiert und führt zu Ablauffehlern.
Insgesamt existiert im Unternehmen ein großes Potenzial für die Automatisierung von Teilprozessen durch Software.

[5] D.h. ein Zugriff auf die erhobenen Daten kann nur manuell durch Lesen des Energieberichts erfolgen.

3.3 SOLL-Konzept

Im folgenden werden die Muss- sowie die Kann-Kriterien für eine Softwarelösung erörtert, die zusammen mit der Geschäftsführung festgelegt wurden. Im Anschluss wird auf Basis der Kriterien eine Auswahl für die zu verwendenden Frameworks getroffen.

3.3.1 Muss-Kriterien

Das neue, softwarebasierte System muss nachfolgende Kriterien erfüllen, welche in funktionale und nicht funktionale Anforderungen unterteilt wurden.

3.3.1.1 Funktionale Anforderungen

Folgende funktionale Anforderungen wurden durch die Analyse erarbeitet oder durch die Geschäftsführung festgelegt:

- Die vom Sachverständigen erhobenen Daten sollen für statistische Auswertungen verfügbar sein.
- Die Ablösung des verteilten Dateisystems Google Drive durch eine zentrale Speicherung der Dateien ohne redunante Kopien auf den Rechnern der Beteiligten ist verbindlich.
- Das System soll gleichzeitig als Kommunikationsplattform für die beteiligten Akteure dienen.
- Die Tools zur Erstellung eines Energieberichts sollen neu entwickelt und in das zukünftige System eingebunden werden, sodass eine automatische Generierung der Energieberichte ermöglicht wird.
- Die durch die Analyse des IST-Zustands ausgemachten Probleme müssen (abgesehen von der Rechnungsstellung) durch die Softwarelösung behoben werden.

3.3.1.2 Nichtfunktionale Anforderungen

Die folgenden nichtfunktionalen Anforderungen wurden durch die Analyse erarbeitet oder durch die Geschäftsführung festgelegt:

- Eine einfache, intuitive Bedienbarkeit und geringe Einarbeitungszeit müssen gewährleistet sein.
- Es muss eine vielseitige Lösung sein, die möglichst alle beteiligten Akteure einbezieht.
- Erweiterbarkeit und Wartung sollen einfach sein.
- Eine spätere Lizenzierbarkeit/Verkaufsoption der Softwarelösung muss möglich sein.

3.3.2 Kann-Kriterien

In diesem Abschnitt werden die erarbeiteten Kann-Kriterien nach funktionalen und nichtfunktionalen Gesichtspunkten aufgeführt.

3.3.2.1 Funktionale Anforderungen

Folgende funktionale Anforderungen wurden als Kann-Kriterien für die Software festgelegt:

- Die Rechnungsstellung für Außendienstmitarbeiter und Sachverständige soll inkludiert sein.
- Informationen über den Status des Auftrags sollen per E-Mail an den Kunden versandt werden.
- Die automatisierte Generierung von Statistiken für das strategische Management.

3.3.2.2 Nichtfunktionale Anforderungen

Es wurde nur eine nichtfunktionale Anforderung ausgemacht, die als Kann-Kriterium für die Software festgelegt werden konnte:

- Die Bedienung soll wahlweise auch über eine App auf dem Smartphone bzw. Tablet möglich sein.

3.3.3 Schlussfolgerungen

Aus den angegebenen Muss-Kriterien lassen sich einige Schlussfolgerungen ziehen, auf die im Folgenden näher eingegangen wird:

- Das GUI muss die Zustimmung der beteiligten Akteure erhalten, um eine generelle Ablehnung des neues Systems durch die Benutzer zu verhindern.
- Um die Verwendung des Systems für den Anwender zu vereinfachen, sollte jeder Anwender nur den Teil des Systems nutzen können, den er für die Bearbeitung seiner Aufgaben braucht.
- Die einfache Wiederverwendung der vom Sachverständigen erhobenen Daten erfordert eine datenbankgestützte Anwendung.
- Die zentrale Speicherung der im Prozessverlauf entstehenden Dokumente erfordert mindestens einen Server, der über das Internet zu erreichen ist, da Anwender von verschiedenen Orten auf die Daten zugreifen.
- Um die einfache Erweiterbarkeit und Wartung zu gewährleisten, ist es notwendig, Änderungen an der Software bei allen Anwendern gleichzeitig durchführen zu können.
- Um das System zu einem späteren Zeitpunkt an Dritte lizenzieren oder verkaufen zu können, ist es notwendig, bei der Entwicklung überwiegend freie Software und Pakete zu verwenden.
- Das einzubindende Kommunikationssystem für die beteiligten Akteure sollte einen kundenspezifischen Anteil besitzen, d.h. Nachrichten, die im Kontext der Energieberatung eines Kunden stehen, sollten nur für die beteiligten Akteure sichtbar sein, um die Organisation zu vereinfachen.

- Es muss eine Rechte- und Rollenverwaltung implementiert werden, um Zugriffsberechtigungen auf Daten und Teilprozesse zu realisieren.

Aufgrund der oben angegebenen Schlussfolgerungen können nun die Rahmenbedingungen der Entwicklung festgelegt werden:

Die Möglichkeit zum gleichzeitigen Update der Anwendersoftware erfordert entweder eine, in den Anwendungsclient integrierte Anfrage an einen über das Internet erreichbaren Versionsverwaltungsservice, oder die Verwendung einer serverlastigen Webanwendung. In beiden Fällen ist jedoch ein Server zur Verwaltung einer zentralen Datenbank notwendig. Im Folgenden wird eine Gegenüberstellung von Webanwendung und Desktopapplikation vorgenommen:

Tabelle 3.1: Gegenüberstellung von Webanwendung und Desktopapplikation [Eigene Darstellung]

Attribut	**Desktopapplikation**	**Webanwendung**
Zugriff	lokale Installation erforderlich	Browserzugriff über das Internet
Updates	Versionsinkompatibilität möglich	zentrale Versionssteuerung
Erweiterbarkeit	Update der lokalen Installation erforderlich	automatische Erweiterung durch Update des Servers
Wartung	schwierigere Wartung	leichtere Wartung
Lastverteilung	clientlastig	serverlastig
Skalierbarkeit	gut skalierbar, abhängig von Clientressourcen	gut skalierbar, unabhängig von Clientressourcen
Entwicklungsaufwand	relativ hoch durch Abhängigkeit vom Betriebssystem	relativ gering durch Unabhängigkeit vom Betriebssystem

3.3.4 Zusammenfassung

Wie dem Abschnitt Schlussfolgerungen und insbesondere der Tabelle 3.1 zu entnehmen ist, bietet sich die Entwicklung einer Webanwendung für die Lösung der Problemstellung bei Profiteam an.

4 Analyse des Energieberichts

Die Erstellung des Energieberichts fällt in den Aufgabenbereich des Sachverständigen und wurde bereits in Abschnitt 3.2.3.1.4 erwähnt. Gegenstand dieses Kapitels ist der IST-Zustand, also die Inhalte eines Energieberichts sowie die aktuelle Vorgehensweise bei der Energieberichtserstellung innerhalb des Unternehmens. Darauf aufbauend wird ein SOLL-Konzept erarbeitet, welches die Optimierungen und Wünsche des Stakeholders beinhaltet.

4.1 IST-Zustand

Ein Energiebericht, der von einem Sachverständigen verfasst wird, ist generell nicht standardisiert. Allerdings existieren Vorgaben (durch das BAFA) bezüglich der verbindlichen Inhalte. Design und Text des Dokuments sind größtenteils dem Sachverständigen überlassen. Demnach ist das Format bzw. das Design eines erstellten Energieberichts für ein Unternehmen je nach ausführendem Sachverständigen unterschiedlich.

4.1.1 Inhalte des Energieberichts

Grundsätzlich hat der zu erstellende Energiebericht folgenden Aufbau:[23]

1. Zusammenfassende Darstellung (Ergebnisse des Energieberichts)
2. Informationen zum Hintergrund des betrachteten Unternehmens
3. Darstellung des IST–Zustands
4. Möglichkeiten zur Verbesserung der Energieeffizienz
5. Anhang

4.1.1.1 Zusammenfassende Darstellung

Der Energiebericht beginnt gemäß der BAFA-Richtlinien mit einer „Zusammenfassenden Darstellung“. Darin sind die Ergebnisse des Energieaudits im untersuchten Unternehmen aufgeführt:

- Textliche Zusammenfassung
- Ergebnisübersicht
- Zusammenfassung der Einsparpotenziale

Bei der Ergebnisübersicht handelt es sich um eine tabellarische Zusammenfassung der vorgeschlagenenen Maßnahmen. Die Zusammenfassung der Einsparpotenziale gliedert die möglichen Einsparungen tabellarisch und grafisch nach der Menge der potenziell eingesparten Energie.

4.1.1.2 Informationen zum Hintergrund des Unternehmens

Das darauf folgende Kapitel des Energieberichts stellt das zu untersuchende Unternehmen, den Sachverständigen und den Ablauf des Audits kurz vor:

- Allgemeine Informationen über das Unternehmen
- Informationen über den Energieberater und die methodische Vorgehensweise
- Kontext der Energieberatung
- Beschreibung des auditierten Objektes
- Relevante Normen und Vorschriften

Die „Allgemeinen Informationen über das Unternehmen“ sind i.d.R. der Website des Kunden entnommen. In diesem Kapitel sind die Standorte und Schwerpunkte der Untersuchung aufgeführt.
Die „Relevanten Normen und Vorschriften“ sind abhängig von dem Unternehmen.

4.1.1.3 Darstellung des IST-Zustands

Das Kapitel konzentriert sich auf die Aufnahme des Zustands zum Zeitpunkt des Audits. Der IST-Zustand wird mithilfe von diversen Tabellen, Diagrammen und vor Ort aufgenommenen Bildern dargestellt:

- Überblick des Energiebezugs
- Energieverbraucheranalyse
- Energiebilanz
- Lastganganalyse
- Energieleistungskennzahlen – EnPI
- Detailanalyse einzelner Energieverbraucher

4.1.1.4 Möglichkeiten zur Verbesserung der Energieeffizienz

Auf Grundlage des IST-Zustands werden in diesem Kapitel des Energieberichts Maßnahmen vorgeschlagen, dafür anwendbare Zuschüsse aufgelistet und deren Wirtschaftlichkeit belegt:

- Vorgeschlagene Maßnahmen
- Informationen über anwendbare Zuschüsse
- Wirtschaftlichkeitsanalyse
- Kriterien für die Rangfolge von Maßnahmen

Die Maßnahmen werden mit einer Vielzahl an Tabellen und Diagrammen veranschaulicht. Je nach vorgeschlagener Maßnahme sind auch Satellitenbilder vorhanden, beispielsweise um den Standort für eine mögliche Photovoltaik-Anlage zu bestimmen.

4.1.1.5 Anhang

Der Anhang baut sich individuell je nach Unternehmen auf. Inhalte sind beispielsweise zusätzliche Berechnungen und Tabellen.

4.1.2 Erstellung des Energieberichts

Der Energiebericht wird größtenteils manuell vom bearbeitenden Sachverständigen erstellt.
Nach der Annahme eines von Profiteam zugewiesenen Auftrags vereinbart der Sachverständige einen Termin für eine Ortsbegeheung im zu untersuchenden Unternehmen (siehe Abschnitt 3.2.3).
Nachdem die technischen Daten von Maschinen und Anlagen vor Ort mittes Notizen festgehalten wurden, beginnt die Erstellung des Energieberichts. Dazu werden die aufgenommenen Daten in die verschiedenen Excel-Tabellen eingetragen und die benötigten Diagramme erzeugt. Anhand der Vorgaben wird der Bericht erstellt und die erstellten Tabellen, Diagramme und aufgenommenen Fotos in ein Word-Dokument eingefügt. Der Sachverständige ergänzt das Dokument um Empfehlungen und weitere Informationen.
Im Anschluss wird das Dokument zu einer .pdf-Datei konvertiert und übergeben. Die Backoffice-Mitarbeiter kontrollieren und korrigieren ggf. den Bericht.

4.1.2.1 Excel-Tools

Es existieren diverse Vorlagen und Versionen der Excel-Tools, die zur Erstellung des Energieberichts durch die Sachverständigen verwendet werden. Es wurde sich darauf geeinigt eine bestimmte Version zu untersuchen, die zum aktuellen Zeitpunkt genutzt wird. Diese wird im Folgenden näher betrachtet.
Es handelt sich um eine Sammlung von verschienden Tools, genauer mehreren Excel Sheets, die aufeinander aufbauen.

4.1.2.1.1 Jahresverbräuche

Das erste Sheet „Dateneingabe“ beinhaltet eine Tabelle, in die die vorhandenen Jahresverbräuche und Kosten für die genutzten Energieträger aufgenommen werden. Diese werden in die Kategorien Strom, Heizenergie und Fuhrpark unterteilt:

- Strom
  - gesamt
  - Strom Netz
  - Strom BHKW
  - Strom PV
- Heizenergie
  - Gas
  - Öl
  - Fernwärme
  - Pellets
  - Sonstiges

- Fuhrpark
  - Treibstoff
  - Flüssiggas
  - Fernwärme

Diese Daten werden für die Jahre aufgenommen, für die Informationen vorliegen. Je nach Unternehmen unterscheidet sich die Anzahl der Datensätze.
Da die Daten nicht der gleichen Einheit entsprechen werden diese umgerechnet und zwischengespeichert.

4.1.2.1.2 Verbraucher

Nach Eingabe der bekannten Jahresverbräuche werden die Verbraucher im Unternehmen tabellarisch aufgelistet und sind nach folgenden Kriterien gegliedert:

- Heizung / Warmwasser
- Büro / Verwaltung
- Produktion
- Lager
- Umwandlungsanlagen
- Sonstiges
- Fuhrpark

Jede dieser Kategorien wird weiter unterteilt nach Art des Verbrauchers, beispielsweise „Beleuchtung“ in der Kategorie „Büro/Verwaltung“. Die maximal aufzunehmenden Daten sind durch die vorhandene Anzahl der Zeilen im entsprechenden Excel-Sheet beschränkt, da die hinterlegten Formeln bei einfachem Hinzufügen einer Zeile nicht funktionieren.
Aus dieser Tabelle wird im darauffolgenden Sheet eine weitere Tabelle mit den Verbraucher-Daten, die erhoben wurden, gefüllt. Der Sachverständige kopiert diese Tabelle in den Energiebericht. Ebenso wird mit den Jahresverbräuchen verfahren. Es existiert ein Sheet, das eine formatierte Tabelle pro Jahr, für welches Daten bekannt sind, erzeugt. Gleichzeitig sind auf diesen Sheets unsortiert Einträge zu finden, die Berechnungen enthalten und in weiteren Sheets verwendet werden.

4.1.2.1.3 Energieeffizienzmaßnahmen

Aufgrund der aufgenommenen Daten empfiehlt der Sachverständige Energieeffizienzmaßnahmen für das untersuchte Unternehmen. Diese werden manuell in eine Tabelle eingetragen, welche dann die jeweiligen Amortisationszeiten je Maßnahme berechnet.

4.1.2.1.4 Tabellen und Diagramme

Der Energiebericht enthält eine Vielzahl an Tabellen, die teilweise aus den Excel-Tools erzeugt und kopiert werden und teilweise manuell erstellt werden. Je nach Unternehmen enthält ein Bericht ca. 15 - 20 Tabellen unterschiedlicher Größe. Weiterhin enthalten die Excel-Sheets diverse statische Tabellen, in denen beispielsweise Umrechungsfaktoren und

aktuelle Energiepreise enthalten sind.
Die benötigten Diagramme werden aus den bereits ausgefüllten Excel-Tabellen generiert. Je nach Unternehmen werden bis zu 13 Diagramme in einen Energiebericht aufgenommen.

4.2 SOLL-Konzept

4.2.1 Muss-Kriterien

Nach Gesprächen mit den Stakeholdern[1] ergibt sich der Wunsch nach einer automatisierten Energieberichtserstellung mit definierten Anforderungen, welche in den nachfolgenden Abschnitten beschrieben werden. Alle Anforderungen sind Muss-Kriterien.

4.2.1.1 Funktionale Anforderungen

- Standardisierung des Inhalts der Energieberichte unter Berücksichtigung der BAFA-Kriterien[2] für Beratungsberichte.
- Automatisierte Erstellung des Berichts in PDF-Format auf Grundlage der erhobenen Daten des Sachverständigen:
  - Automatisierte Erstellung der Tabellen
  - Automatisierte Erstellung der Diagramme
  - Möglichkeit die aufgenommenen Bilder einzubinden
  - Möglichkeit eigene Texte einzubinden
- Einbindung der Generierung in die Django-Applikation
- Die Datenaufnahme für den Energiebericht soll innerhalb der Django-Applikation stattfinden

4.2.1.2 Nichtfunktionale Anforderungen

Folgende nichtfunktionale Anforderungen wurden festgelegt:

- Standardisierung des Designs
- Das Generierungssystem soll transparent eingebunden werden, d.h. es soll für den Benutzer unsichtbar sein.

4.2.2 Schlussfolgerungen

In diesem Abschnitt werden Schlussfolgerungen aus der Analyse gezogen, welche als Basis für die Konzeptionierung eines Systems zur automatischen Generierung von Energieberichten dienen soll.

[1] hier: Geschäftsführung von Profiteam sowie die Sachverständigen

4.2.2.1 Datenaufnahme

Die Verwendung der Excel-Tools muss im Zuge der Berichtsautomatisierung entfallen. Stattdessen soll es entsprechende Eingabemasken auf der Website geben, die die Sachverständigen durch den Prozess leiten. Die Reihenfolge der Eingabe muss im Grundsatz bestehen bleiben, um eine geringe Einarbeitungszeit zu ermöglichen. Redundante Informationen der Tools müssen entfernt und konstante Kennzahlen wartbar gespeichert werden.

4.2.2.2 Design

In Abstimmung mit der Geschäftsführung wird aus vorhandenen Energieberichten eine Vorlage gewählt, die das grundsätzliche Design festlegt und in Zukunft für alle Berichte gelten soll. Es ist die Zustimmung des Autors des Berichts einzuholen, da insbesondere eine kommerzielle Nutzung vorgesehen ist.

5 Konzeptionierung

In diesem Kapitel werden die Rahmenbedingungen für die zu entwickelnde Software erörtert und die Konzeptionierung für beide Teile dieser Arbeit, die Django-Applikation und die Berichtsautomatisierung, besprochen.

5.1 Rahmenbedingungen des Projekts

Im Rahmen einer in der Vergangenheit für das Fach Internettechnologien durchgeführten Projektarbeit, die in Kooperation mit der Firma Profiteam stattfand, wurde die Website des Unternehmens mit Hilfe des Django Web Frameworks neu erstellt. Das durch die Projektarbeit entstandene Django-Projekt soll nun um eine weitere Applikation erweitert werden, die die geforderte Funktionalität implementiert. Zur besseren Aufteilung der Arbeitsbereiche und zur Übersichtlichkeit werden die Themen Berichtsautomatisierung und die Abbildung der Geschäftsprozesse in getrennten Django Apps erarbeitet und auch in dieser Arbeit getrennt voneinander betrachtet.

5.1.1 Infrastruktur

Für die Produktionsumgebung des Projekts stehen folgende Ressourcen zur Verfügung, die bei der Planung des Projekts berücksichtigt werden müssen.

Tabelle 5.1: Spezifikation des V-Servers [Eigene Darstellung]

Prozessor	6 virtuelle Kerne
Arbeitsspeicher	12 GB
Massenspeicher	600 GB kombiniert aus SSD und HDD
Betriebssystem	Ubuntu 16.04 LTS
Netzwerkanbindung	jeweils 500Mbit/s für Up- und Downlink
Berechtigung	Root-Zugriff

Auf dem genutzen V-Server werden mehrere Git-Repositories und Datenbanken für die Software gehosted. In das Deploy-Repository werden manuell ausgewählte Entwicklungsstände des Development-Repositories gepushed. Das Deploy-Repository verfügt über sog. Git Hooks, also Skripte die gestartet werden, wenn eine bestimmte Git-Operation erfolgt. Erfolgt ein Update (Push) in das Deploy-Repository, so wird automatisch der Code und das Datenbankschema in der Produktionsumgebung auf den Stand des Deploy-Repositorys angepasst. Vor jedem Deploy einer neuen Version wird ein zusätzliches Datenbank-Backup erstellt, da durch die neue Version i.d.R. Änderungen an der Datenstruktur stattfinden werden. Zur Entwicklung wird eine separate Testdatenbank genutzt, um unerwünschte Wechselwirkungen mit der Produktionsumgebung auszuschließen.

Weiterhin werden Git-Repositories für jedes der LaTeX-basierten Dokumente gehosted. So kann die Entwicklung der Dokumentengenerierung zu einem Großteil von der Entwicklung der Webapplikation losgelöst erfolgen. Ist die Entwicklung einer LaTeX-Vorlage ab-

geschlossen, so wird die Dateistruktur im Development-Repository des Django-Projekts eingebunden und anschließend die Funktionalität, also die automatische Generierung des Dokumentes, über Django realisiert.

5.2 Vorgehensweise und Entwicklungsziele

Es wird zunächst eine Alpha-Version der Software entwickelt, die die Funktionalitäten für die Rollen Außendienstmitarbeiter, Sachverständiger und Backoffice abbildet sowie die Dateiverwaltung implementiert. Anschließend erfolgt eine Testphase mit einer geschlossenen Personengruppe, in der Fehler behoben werden und Feedback von den betroffenen Akteuren eingeholt wird. Nach erfolgreichem Abschluss der Testphase werden sukzessive Module für die Rollen Kunde, Rechnungsbearbeitung und Geschäftsführung eingeführt, wobei jeder Implementierung eine erneute Testphase mit den Beteiligten folgt, bevor ein Release im Hauptprogramm stattfindet. Durch diese Vorgehensweise soll sichergestellt werden, dass die Software die Erwartungen der Akteure erfüllt und stets ein funktionierendes System im produktiven Einsatz ist.
Bei Abschluss dieser Arbeit soll das Entwicklungsziel „closed Alpha“ erreicht sein.

5.3 Django-Applikation

In diesem Abschnitt werden die Konzeptionierung der Django-Applikation beschrieben sowie grundsätzliche Entscheidungen für die Umsetzung des Projekts getroffen und erörtert. Basis für die Konzeptionierung ist der Abschnitt „SOLL-Konzept“ in Kapitel 3.

5.3.1 Benutzeroberfläche

Für die Erzeugung der Benutzeroberfläche wird, aus Gründen der späteren Lizenzierbarkeit, ein freies CSS-Theme verwendet. Dies spart Entwicklungszeit, da nur Anpassungen in der Anordnung der GUI-Elemente vorgenommen werden müssen und grundlegende Designarbeiten auf ein Minimum reduziert werden. Die Benutzeroberfläche soll als Abwandlung eines Dashboards implementiert werden, um Übersichtlichkeit und intuitive Bedienbarkeit zu fördern. Um die Wartbarkeit der Software zu verbessern, soll das Rendern der aufgerufenen Webseite über mehrere Templates realisiert werden, die kontextabhängig inkludiert werden. Es ist weiterhin geplant, dass die Benutzeroberfläche aus einem statischen, gruppen- und benutzerübergreifenden Teil besteht sowie aus einem spezifischen, individuell auf die Rolle des Benutzers zugeschnittenen Teil. Der Zugriff auf das Dateisystem soll ohne zusätzliche externe Werkzeuge über die Benutzeroberfläche möglich sein, was auch den Upload von Dateien beinhaltet.

5.3.2 Rollen

Nutzer der Software müssen einen registrierten Account besitzen, wobei die Registrierung vorerst nur über den Administrator des Systems erfolgen kann. Im Zuge der Registrierung wird dem Nutzer eine Rolle, respektive Gruppe, im Authentifizierungssystem von Django zugewiesen, die mit seiner realen Rolle im Prozess korrespondiert. Der Benutzer kann nur die Funktionen des Systems nutzen, die für seine Gruppe freigeschaltet sind. Spezifische Sonderrechte für einzelne Benutzer einer Gruppe sind nicht vorgesehen, aber technisch möglich. Der Prozessfortschritt eines Auftrags soll allen am Auftrag beteiligten Benutzern angezeigt werden, um unnötige Nachfragen im Backoffice zu vermeiden.

5.3.3 Teilprozesse

Teilprozesse bzw. die Aufgaben der Akteure sollen standardisiert als Model in Django abgebildet werden. Es ist vorgesehen, dass zu jedem Kunden für jeden Teilprozess eine entsprechende Model-Instanz existiert, in der die für die Aufgabe relevanten Daten gespeichert werden. Gleichzeitig wird der Gesamtfortschritt des Prozesses über die Abarbeitung der Teilprozesse verfolgt und gesteuert. Die Bearbeitung der Teilprozesse soll über Formulare erfolgen, was die Modularisierung vereinfacht und genutzt werden kann, um den Benutzer aktiv durch den Prozess zu führen. Alle Benutzer sollen über den aktuellen Status aller Teilprozesse über das GUI informiert werden, jedoch nur den Teilprozess bearbeiten können, der ihnen zugewiesen und aktuell zu bearbeiten ist. Weiterhin bietet sich durch die Unterteilung in einzelne Aufgaben die Möglichkeit, automatisch auf Fristabläufe oder anstehende Termine zu prüfen. Näheres dazu ist dem Abschnitt 5.3.6 zu entnehmen.

5.3.4 Nachrichtenaustausch

Der Nachrichtenaustausch soll kundenspezifisch implementiert werden, d.h. das Nachrichtenmodel soll eine Referenz zur jeweiligen Kundeninstanz besitzen. Am kundenspezifischen Nachrichrichtenaustausch sollen nur die beteiligten Akteure mitwirken können. Dies soll über Referenzen zu den entsprechenden User-Objekten realisiert werden. Wird ein Beteiligter aus dem System ausgeschlossen, so sollen die von ihm erstellten Nachrichten nicht gelöscht werden. Nachrichten sollen zeitlich, personell und kundenspezifsch zugeordnet, sowie als „neu“ bzw „gelesen“ nachvollziehbar kategorisiert werden können. Nachrichten sollen nur vom jeweiligen Autor sowie dem Administrator gelöscht werden können. Das Nachrichtensystem wird einen Teil umfassen, der allgemeine, für alle Nutzer gültige, Informationen beinhaltet. Diese werden optisch von den kundenspezifischen Nachrichten abgegrenzt. So können alle Nutzer über geplante Updates und Wartungszeiten informiert werden. Nutzer werden informiert, wenn im Nachrichtenbereich eines Kunden, für den der Nutzer eine aktive Rolle im Prozess wahrnimmt, neue Nachrichten veröffentlicht werden.

5.3.5 Dateiverwaltung

Die Dateiverwaltung soll weiterhin kundenspezifisch realisiert werden, wobei die Dateien aus Performancegründen nicht in der Datenbank, sondern in einer Ordnerstruktur gespeichert werden, was auch die spätere Migration von alten Datenbeständen erleichtert. Dennoch muss eine referenzbasierte Einbindung in die Datenbank erfolgen, die primär auf dem Dateipfad basieren wird. Das Herunterladen von Dateien ist nur für beteiligte Akteure über das GUI vorgesehen, ein direkter Download, beispielsweise über das FTP, darf nicht erfolgen. Die kundenspezifischen Dateien werden auf der Benutzeroberfläche angezeigt und weitere Dateien können über das GUI hinzugefügt werden. Dateien von abgeschlossenen Aufträgen dürfen nicht mehr manipuliert werden.

5.3.6 Automatisiertes Controlling

Durch die Modularisierung des Gesamtprozesses in Teilprozesse wird das Controlling vereinfacht, da stets ersichtlich ist, welcher Teilprozess gerade bearbeitet wird. Um automatisch nahende Fristen zu erfassen werden Alarme konzipiert welche sowohl eine Referenz zum Kunden, als auch zum betroffenen Akteur besitzen. Ein Alarm soll aktiv sein und angezeigt werden, bis er als erledigt makiert wurde, wobei die Erledigung durch Nutzeraktionen oder automatisiert erfolgen kann. Bei jedem Login eines Benutzers werden alle Alarme seiner Kunden überprüft und nicht erledigte Alarme dem Benutzer angezeigt. Ein Alarm kann nur durch die Behebung der Ursache erledigt werden. Ursächlich ist i. d. R. ein Fristablauf oder eine notwendige Handlung durch den Benutzer. Weiterhin sollen Alarme auch verwendet werden, wenn ein neuer Teilprozess zur Bearbeitung freigegeben wurde. In diesem Fall wird der Alarm mit dem gültigen Abschließen des entsprechenden Formulars als erledigt markiert.

5.3.7 Administration

Die Administration der Software erfolgt zunächst über den kombinierten Einsatz der von Django bereitgestellten Admin-Webseite für die Webanwendung selbst sowie dem Einsatz von *ssh* und den damit ermöglichten Optionen zur Verwaltung und Sicherung des Dateisystems sowie der Datenbank. Im Zuge der weiteren Entwicklung können bestimmte administrative Tätigkeiten auf die Benutzer umgelagert werden, beispielsweise die Änderung von Kontaktdaten und Passwörtern. Funktionale Updates können für die innerbetriebliche Anwendung über das „Deploy“-Repository ausgerollt werden.

5.4 Berichtsautomatisierung

Der folgende Abschnitt beschäftigt sich mit der Konzeptionierung der Berichtsautomatisierung für Energieberichte auf Grundlage des SOLL-Konzepts aus Kapitel 4.2.

5.4.1 Allgemeines

Für die Erzeugung des Energieberichts wird LaTeX genutzt. Im Gegensatz zu dem Textverarbeitungsprogramm Word von Microsoft bietet LateX eine plattformunabhängige Möglichkeit für die Generierung der gewünschten PDF. Bei anderen Programmen ist die Wahrscheinlichkeit höher, dass die Generierung im Zuge eines Updates nicht mehr einwandfrei funktionieren würde.

5.4.2 Vorgehensweise

Die Generierung des Berichts wird über die Website ausgelöst. Durch die Betätigung eines Buttons wird vom Webserver eine notwendige Subroutine gestartet, welche eine Konfigurationsdatei für LaTeX mit entsprechenden Daten aus der Datenbank füllt.
Anschließend wird das LaTeX-Dokument vom Server kompiliert, wozu pdflatex ausgeführt wird. Die erzeugte PDF wird in einen vorgegebenen Ordner des Webservers verschoben.
Im Anschluss wird die erzeugte Datei mit Hilfe eines Models für Django verpackt. Näheres zum verwendeten Model ist dem Abschnitt „Dateiverwaltung“ im Kapitel 6 zu entnehmen.
Bei erfolgreicher Generierung erhält der Benutzer über die Webseite die Rückmeldung, dass der Energiebericht erstellt wurde.
Das Sequenzdiagramm in Abbildung 5.1 veranschaulicht die geplante Vorgehensweise:

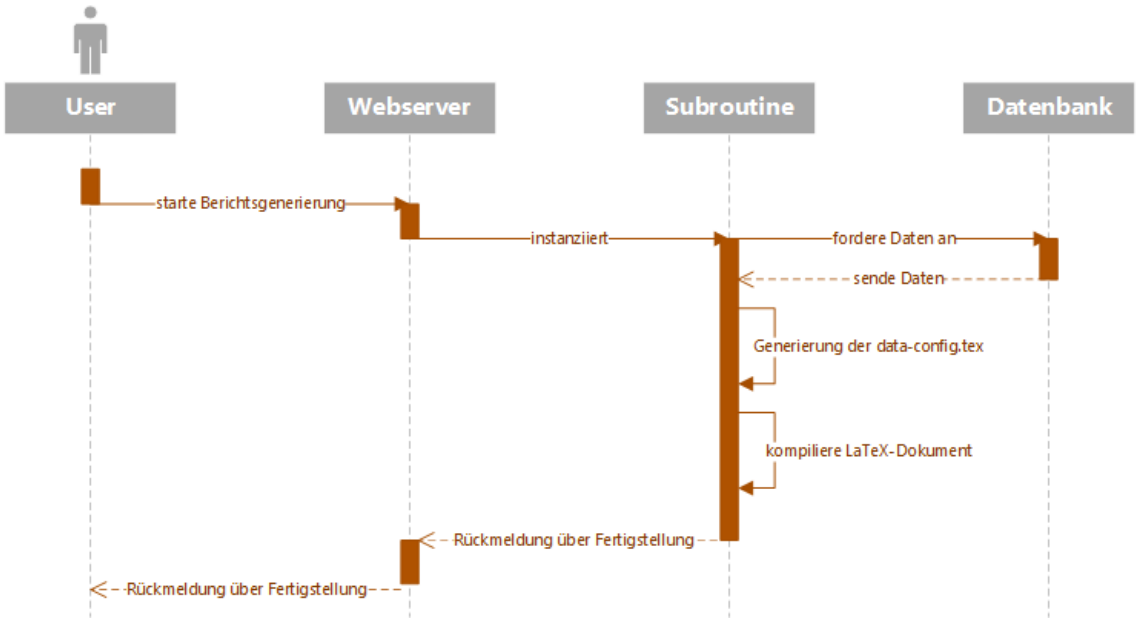

Abbildung 5.1: Geplante Vorgehensweise der Berichtsgenerierung [Eigene Darstellung]

5.4.3 LaTeX-Dateistruktur

Für die Berichtsgenerierung ist folgende Dateistruktur vorgesehen:

- energiebericht.tex
- general-config.tex
- data-config.tex
- mindestens eine .tex-Datei pro Kapitel

energiebericht.tex bezeichnet die Haupt-Datei des Energieberichts. Innerhalb dieser Datei werden die einzelnen Kapitel und die Konfigurationsdateien importiert. Die Datei **general-config.tex** soll die erforderlichen Pakete importieren und grundsätzliche Layout-Anpassungen beinhalten.
Die vom untersuchten Unternehmen abhängigen Daten werden in der Datei **data-config.tex** als Latex-Makros gespeichert und in den entsprechenden Kapiteln verwendet.

5.4.3.1 data-config.tex

Die Datei data-config.tex wird mit Makros gefüllt, die die Daten aus der Datenbank beinhalten. Hierzu wird der Befehl *\newcommand* verwendet. Für jeden kundenspezifischen Wert oder Textbaustein wird ein solches Makro benötigt. Die Werte, die mittels Eingabemaske in die Datenbank eingefügt wurden, werden dem korrespondierenden Latex-Makro zugeordnet. Die erstellten Makros werden in den verschiedenen Kapiteln des Energieberichts verwendet.

5.4.3.2 general-config.tex

Die Datei general-config.tex legt u.a. die Formatierung für den Energiebericht fest. Im Folgenden werden die Formatierungswünsche des Stakeholders kurz zusammengefasst.

5.4.3.2.1 Kopf- und Fußzeile

Die Kopfzeile des Energieberichts soll lediglich das SGS-TÜV-Saar-Logo beinhalten. Die Fußzeile soll mit einem grauen Balken vom Inhalt des Berichts abgetrennt werden. Linksbündig soll „Energieaudit:„ und der jeweilige Kunde bzw. der Firmenname stehen. Rechtsbündig soll sich die aktuelle Seitennummer sowie die Gesamtseitenanzahl befinden.

5.4.3.2.2 Farbschema und Schriftart

Kapitel- und Abschnittsüberschriften sollen sich formal und farblich unterscheiden. Bei der Kapitelüberschrift soll mit einem Grauton gearbeitet werden. Bei den Abschnittsüberschriften werden, je nach Ebene, unterschiedliche Blautöne verwendet. Hierfür werden verschiedene Umgebungen definiert.
Bei der Schriftart gibt es keine expliziten Vorgaben. Man hat sich auf eine serifenlose Schriftart geeinigt, die Arial ähneln soll.

5.4.3.2.3 Tabellen und Diagramme

Der Energiebericht beinhaltet eine Vielzahl an Tabellen und Diagrammen, die kundenspezifische Daten enthalten.
Für die Generierung der Tabellen sind Django-Models vorgesehen, die die Daten für den Energiebericht kapseln. Das Layout der Tabellen wird in LaTeX implementiert.
Die Daten der Tabellen werden häufig in Diagrammen (Pie-Charts, Balken-Diagramme) zusammenfassend dargestellt. Für diesen Zweck sollen entsprechende Umgebungen definiert werden.

5.4.3.3 Sonstige Dateien

Für jedes Kapitel des Energieberichts (siehe 4.1) existiert mindestens eine zugehörige .tex-Datei. Darin werden verwendetete Textbausteine eingefügt und um die Variablen aus der data-config.tex ergänzt. Nicht in jedem Energiebericht sind alle Kapitel enthalten, da nur die Kapitel, die den spezifischen Kunden betreffen, inkludiert werden sollen. Beispielsweise soll es kein Kapitel über Druckluft geben, wenn bei dem Kunden keine Druckluftinstallation vorhanden ist.

5.4.4 Datenerfassung und -verarbeitung

Die in 4.1.2.1 beschriebenen Excel-Tools sollen in der Django-Applikation abgebildet werden, da die so erhobenen Daten die Grundlage des zu erstellenden Energieberichts darstellen.
Die Sachverständigen sollen die Möglichkeit haben, die aufgenommenen Daten mittels Eingabemasken direkt über die Webseite zu bearbeiten.
Folgende Formulare sind hierzu mindestens notwendig:

- Jahresverbräuche
- Verbraucherübersicht
- Vorgeschlagene Energieeffizienzmaßnahmen

Für die Aufnahme der elektrischen Verbraucher sind mehrere Formulare geplant. Für jede Kategorie (siehe 4.1.2.1.2) muss die Möglichkeit bestehen, beliebig viele Verbraucher aufzunehmen. Weitere Daten werden in anderen, vorangehenden Teilprozessen innerhalb der Django-Applikation erhoben und im Energiebericht verarbeitet.

6 Implementierung

In diesem Kapitel werden Auszüge der Implementierung der Projektarbeit für die beiden Teile, die Django-Applikation und automatisierte Dokumentengenerierung, besprochen.

6.1 Django-Applikation

Aus der Entwicklung der Django-Applikation werden in diesem Abschnitt ausgewählte Teilaspekte aufgezeigt und beschrieben.

6.1.1 Benutzeroberfläche

Für die Implementierung der Benutzeroberfläche wurde das freie Bootstrap 4 Theme „SB Admin“ verwendet, welches unter Berücksichtigung der Anforderungen entsprechend angepasst wurde[24]. Aus Gründen der Übersichtlichkeit dieser Arbeit werden nicht alle Views mit Screenshots aufgeführt. Als Beispiel für das Basislayout dient die Kundenübersicht für einen neu angelegten Testkunden.

In Abbildung A.1 im Anhang wird die kundenspezifische Übersicht gezeigt, nachdem der Neukunde vom Außendienst übergeben wurde und die Stammdaten durch das Backoffice ergänzt wurden. Diese Ansicht steht allen beteiligten Akteuren zur Verfügung und erlaubt die schnelle Überprüfung des Prozessstatus.

6.1.1.1 Struktur des Layouts

Die Ansicht, die in Abbildung A.1 gezeigt wird, wird aus der Kombination zweier Templates gerendert. Basis aller Views der App ist eine Layout-HTML-Datei, die für das Rendern von Header, Navbar und Footer verantwortlich ist. Dieses Template besitzt einen Abschnitt, in dem der spezifische Inhalt einer View eingefügt wird. Optisch ist dieser Bereich durch einen weißen Hintergrund zu erkennen.

6.1.1.1.1 Header

Im Header wird stets angezeigt, welchen Gruppen der angemeldete Anwender zugehörig ist. So ist der Abbildung A.1 zu entnehmen, dass der angemeldete Benutzer alle drei vorläufig vorgesehenen Gruppen inne hat und somit über volle Berechtigungen verfügt. Weiterhin werden im Header symbolisch die Anzahl der ungelesenen Nachrichten sowie die Anzahl der offenen Alarme angezeigt. Ein Klick auf das entsprechende Symbol öffnet ein Drop-Down-Menü, in dem eine Vorschau der jeweils neusten drei Nachrichten bzw. Alarme angezeigt wird. Ein Klick auf eine Nachricht bzw. einen Alarm innerhalb des Drop-Down-Menüs führt zur korrespondierenden Detailansicht. Die Suchfunktion wird innerhalb der Pre-Alpha-Version der Software noch nicht verwendet, ist aber für eine spätere Version vorgesehen.

6.1.1.1.2 Navigationsbar

Im linken Bereich befindet sich die Navigationsbar der Anwendung. Über das Element „Kunden“ erhält der anfragende Nutzer eine Liste aller Kunden, an denen er ein beteiligter Akteur ist. Diese Liste wird sortiert nach Erstelldatum der Kunden und nach Prozessstatus angezeigt. Die Elemente „Statistiken“ und „Tabellen“ sind für eine spätere Version reserviert, um rollenspezifisch Daten zu visualisieren. Über das Element „Einstellungen“ können Benutzer ihr Passwort ändern, Kontaktdaten aktualisieren und andere, generelle Einstellungen vornehmen. Das Element „Feedback“ wird bis zur Veröffentlichung der Vollversion, möglicherweise auch darüber hinaus, genutzt. Hier findet sich zum einen die Möglichkeit für die Anwender, Bugreports oder Wunschfeatures per Freitext zu speichern. Zum anderen können alle Nutzer alle eingereichten Bugreports bzw. Wunschfeatures sehen und deren Bearbeitungsstatus verfolgen.

6.1.1.2 Struktur des eingefügten Inhalts

In Abbildung A.1 ist im Bereich mit weißem Hintergrund der durch die spezifische View eingefügte Inhalt zu sehen. In der oberen, grau gefärbten Box wird immer der Titel der aktuellen Ansicht eingefügt. Im darunter liegenden Bereich sind in diesem Beispiel vier Teilbereiche zu erkennen, die optisch voneinander abgegrenzt sind.

6.1.1.2.1 Stammdaten

Die Stammdaten des Kunden werden im oberen Bereich tabellarisch angegeben. Auf der linken Seite finden sich die Kontaktdaten zum Verantwortlichen im Kundenunternehmen. Ein Klick auf die angegebene E-Mail-Adresse öffnet den Standard-E-Mail-Client des Anwenders mit der Vorauswahl einer neuen E-Mail an die angegebene Adresse. Auf der rechten Seite ist die Adresse des Kunden zu sehen sowie der aktuelle Status des Prozesses. Ein Klick auf die Adresse öffnet maps.google.com mit Markierung des entsprechenden Standortes in einem neuen Tab.

6.1.1.2.2 Jobs

In diesem Bereich werden die Teilprozesse visualisiert. Abgeschlossene Teilprozesse werden mit einem Häkchen versehen. Teilprozesse die zu bearbeiten sind, werden durch einen gelben, rotierenden Pfeil gekennzeichnet. Gesperrte Teilprozesse werden durch ein Stopp-Symbol visualisiert. Ein Teilprozess kann nur von Mitgliedern der Gruppe bearbeitet werden, für die er zur Bearbeitung vorgesehen ist. Mitglieder anderer Gruppen haben keine Möglichkeit die entsprechende Formulare aufzurufen. Ein Teilprozess der mit dem Stopp-Symbol markiert ist, kann nicht ausgeführt werden bis alle vorherigen Teilprozesse abgeschlossen sind. Hinter jedem Teilprozess, der in der Kundenübersicht dargestellt wird, steht mindestens ein weiterer Teilprozess, der mit einem oder mehreren Formularen verknüpft ist. Diese Designentscheidung wurde getroffen, um die Übersichtlichkeit der Anwendung zu verbessern.

6.1.1.2.3 Dateien

In Bereich „Dateien“ werden von Benutzern hochgeladene sowie vom System generierte Dateien angezeigt. Ein Klick auf einen entsprechenden Eintrag führt zum Öffnen der Datei in einem neuen Tab (bei Vorliegen im PDF-Format) oder zum Download der Datei (bei einem Format, das nicht zur Anzeige im Browser geeignet ist). Werden Dateien im Zuge des Gesamtprozesses überschrieben, so finden sich die alten Dateiversionen grau

hinterlegt am unteren Ende der Liste. Auch andere Umstände, wie z.B. ein Fristablauf, können dazu führen, dass eine Datei als „deprecated“, also veraltet, markiert wird.

6.1.1.2.4 Nachrichten

Im Bereich Nachrichten befindet sich eine Eingabeform, die es dem Anwender ermöglicht eine neue Nachricht zu senden. Diese so veröffentlichte Nachricht wird allen, an diesem Auftrag beteiligten Akteuren, angezeigt. Es wird differenziert, welche Akteure eine Nachricht gelesen haben. Nach einem Klick auf die Detailansicht der Nachricht über das Dropdown-Menü im Layout oder dem Aufruf der in Abbildung A.1 gezeigten Seite, wird die Nachricht als gelesen markiert. Nachrichten können nur vom Autor oder dem Administrator gelöscht werden.

6.1.1.3 Templatetags und Context Processors

In diesem Beispiele für die Verwendung von Templatetags und Context Processors innerhalb der Anwendung gezeigt und erklärt.

6.1.1.3.1 Templatetags

Um innerhalb der Django Template Language zusätzliche, selbst definierte Funktionen nutzen zu können, ist die Verwendung von sogenannten Templatetags notwendig. Als Beispiel dient hier die Konvertierung der Kundenadresse in einen Querystring für Google Maps. Zunächst wird die Definition des Templatetags gezeigt, anschließend wird die Verwendung des Template Tags beschrieben.

```
from django import template

register = template.Library()

# used for converting address into maps.google.com compatible Format
@register.filter
def to_google_format(value):
    google_value = value.replace(",", "%2C")
    google_value = google_value.replace(" ", "+")
    google_value = google_value.replace("|", "%7C")

    return google_value
```

Listing 6.1: Inhalt der Datei to_google_format.py [Eigene Darstellung]

In Listing 6.1 ist die Definition des notwendigen Templatetags angegeben. Über den Decorator *@register.filter* wird die Methode der Libraryinstanz für Template Tags hinzugefügt[17]. Die notwendigen Zeichenersetzungen sind in der Dokumentation von Google Maps zu finden[20]. Nicht implementierte Zeichenersetzungen sind bereits durch die Eingabeforms für Adressen ausgeschlossen worden.

```
[...]
{% load to_google_format %}
[...]
 <a data-toggle="tooltip" data-placement="top" title="Adresse in
    Google Maps öffnen"
                href="https://maps.google.com/?q={{adresse.ort|
                    to_google_format}}%2C{{adresse.strasse|
                    to_google_format}}+{{adresse.hausnummer}}"
                target="_blank">
                [...]
 </a>
```

Listing 6.2: Verwendung des Template Tags in der Kundenübersicht (Ausschnitt) [Eigene Darstellung]

In Listing 6.2 wird die Verwendung des Template Tags in HTML gezeigt. Zunächst muss das Template Tag explizit für dieses Template geladen werden. Dies geschieht mit der Zeile **{% load to_google_format % }**. Durch die Verwendung des Template Tags werden zum Renderzeitpunkt die Teilstrings der Adresse in ein zu Google Maps kompatibles Format konvertiert.

6.1.1.3.2 Context Processors

Wenn Daten in mehreren Views gebraucht werden ist die Verwendung eines Context Processors sinnvoll. Ein definierter Context Processor wird beim Rendern jeder View im Hintergrund aufgerufen und ausgeführt. Somit können wiederkehrende Datenmuster einmalig definiert werden und stehen anschließend in allen Views zur Verfügung. Im Folgenden wird die Definition eines Context Processors gezeigt:

```
from django.http import HttpRequest

from webapp.forms import BugReportForm
from webapp.common_functions import *

[...]

def always(request):
    assert isinstance(request, HttpRequest)
    if request.user.is_authenticated:
        meinekunden = getKunden(request)
        meineNachrichten = getNachrichten(request, meinekunden)
        meineAlarme = getAlarme(request, meinekunden)
        alarmCount = meineAlarme.count()
        newsCount = meineNachrichten.count()
        bugreportform = BugReportForm()
        return {
            'groups': request.user.groups.all(),
            'year': datetime.now().year,
            'news': meineNachrichten[:3],
            'newsCount': newsCount,
            'alarme': meineAlarme[:3],
            'alarmCount': alarmCount,
            'meineKunden': meinekunden,
            'bugreportform': bugreportform,
    }
    return {}
```

Listing 6.3: Ausschnitt aus context_processors.py [Eigene Darstellung]

In Listing 6.3 ist die Definition des Context Processors „Always“ zu sehen. Bei jedem Rendervorgang wird der entsprechende Code ausgeführt und implizit dem „aktiv“ übergebenen Context hinzugefügt. Neben allgemeinen Daten, die innerhalb des Layouts verarbeitet werden (siehe auch Abschnitt 6.1.1.1) können im Context Processor auch Forms übergeben werden. Zu beachten ist, dass ein Context Processor immer einen Context zurückgeben muss, auch wenn dieser leer ist. Damit der Context Processor aktiv wird, ist es notwendig ihn in der Datei **settings.py** der Liste aktiver Context Processors hinzuzufügen:

```
[...]
TEMPLATES = [
        {
                'BACKEND': 'django.template.backends.django.DjangoTemplates',
                'DIRS': [],
                'APP_DIRS': True,
                'OPTIONS': {
                        'context_processors': [
                                'webapp.context_processors.always',
                                'django.template.context_processors.debug',
                                'django.template.context_processors.request',
                                'django.contrib.auth.context_processors.auth',
                                'django.contrib.messages.context_processors.messages',
                        ],
                },
        },
]
[...]
```

Listing 6.4: Ausschnitt aus der Datei settings.py [Eigene Darstellung]

In Listing 6.4 sind die im Projekt verwendeten Context Processors deklariert. Zu beachten ist, dass jeder Context Processor einzeln aktiviert werden muss. Die voreingestellten Einträge dienen unter anderem der Authentifizierung (sodass Gruppen und Berechtigungen in Templates verwendet werden können) sowie dem integrierten Message System von Django, welches es ermöglicht sogenannte „flash messages“ an einzelne Browser auszuliefern und eine Middleware für dieses System bereitstellt.

6.1.2 Implementierung der Prozessabfolge

Die Implementierung der Abfolge der Teilprozesse erfolgt auf Basis des zusätzlichen Feldes „Status“ für das Kunden-Model. Der Status des Auftrags wird als natürliche Zahl im Model gespeichert, was es insbesondere während der Entwicklung vereinfacht, einzelne Teilprozesse durch direkte Datenbankmanipulation zu testen. Der Wert des Statusfeldes entspricht dem Schlüssel einer Hashmap, deren Wert dem korrespondierenden Status im Klartext entspricht. Der Gesamtprozess schreitet voran, indem am Ende eines Teilprozesses der Status des Auftrags auf den Wert des nächsten Teilprozesses gesetzt wird.
Auch innerhalb der Benutzeroberfläche findet das Statusattribut Anwendung bei der Anzeige der Teilprozesse, sowie bei der Visualisierung innerhalb der Kundenübersicht. Die Businesslogik der Teilprozesse wurde, wie in der Konzeptionierungsphase geplant, in separaten Views implementiert, um die Modularität sowie die Wartbarkeit zu vereinfachen. Im Folgenden werden Beispiele für die Verwendung des Statusattributs gezeigt:

```
class Kunde(models.Model):
[...]
    status = models.PositiveSmallIntegerField(choices=AUFTRAGSSTATUS)
    created_at = models.DateTimeField(auto_now_add=True)
    last_edit_at = models.DateTimeField(auto_now=True)
[...]
```

Listing 6.5: Ausschnitt des Kunden-Models aus models.py [Eigene Darstellung]

In Listing 6.5 ist unter anderem die Definition des Statusfeldes zu sehen. Gültige Werte für das Feld sind über die Tupel-Liste AUFTRAGSSTATUS definiert, welche hier aus Gründen der Übersichtlichkeit nicht gezeigt wird. Die Felder created_at und last_edit_at werden automatisch beim erstmaligen Erstellen (created_at) bzw. bei jedem Speichern des Models (last_edit_at) beschrieben. Die Anzeige des Auftragsstatus innerhalb der Benutzeroberfläche erfolgt über die automatisch von Django generierte Funktion {{kunde.get_status_display}}, die den Status im Klartext innerhalb eines Templates zurückgibt.

Für die Anzeige der Teilprozesse wurde ein zusätzliches Template entwickelt, das eine parametrierbare Darstellung eines Teilprozesses erlaubt und die Lesbarkeit des Source-Codes verbessert.

```
{% load staticfiles %}
<a class="list-group-item list-group-item-action" {% if kunde.status >= jobstatus %}
    href="{{linkto}}"{% endif %}>
            <div class="media">
                <div class="job_status">
                    {% if kunde.status < jobstatus %}
                    <img src="{% static 'admin/stop.png'%}">
                    {% endif %}
                    {% if kunde.status == jobstatus %}
                    <img src="{% static 'admin/arrows.png'%}">
                    {% endif %}
                    {% if kunde.status > jobstatus %}
                    <img src="{% static 'admin/okay.png'%}">
                    {% endif %}
                </div>
                <div class="media-body jobstate_bo" >
                    <strong>{{job}}</strong>
                </div>
            </div>
</a>
```

Listing 6.6: Die Datei BO-job-display.html [Eigene Darstellung]

In Listing 6.6 ist der Quelltext des für die Darstellung von Teilprozessen für Backoffice-tätigkeiten verwendeten Templates zu sehen. Die URL zum entsprechenden Formular ({{linkto}}) wird nur in den auszuliefernden HTML-Code geschrieben, wenn der Auftragsstatus größer oder gleich dem als Parameter übergebenen **jobstatus** ist. Je nach Status erscheint ein anderes Symbol neben dem Auftragstext ({{job}}). Beispielhaft wird nun die Verwendung in der Auftragsübersicht für die Vorbereitung durch Backoffice-Mitarbeiter gezeigt.

```
{% extends "layout2.html" %}
{% block content %}
{% load staticfiles %}
 <div class="card-header">
        <i class="fa"></i> Aufgaben
    </div>
    <div class="list-group list-group-flush small">

        {% url 'webapp:stammdaten' dokunde=kunde.firmenname as stammdaten %}
        {% include "jobs/BO-job-display.html" with jobstatus=1 job='Stammdaten editieren
            ' linkto=stammdaten %}

        {% url 'webapp:datenblatt' dokunde=kunde.firmenname as datenblatt %}
        {% include "jobs/BO-job-display.html" with jobstatus=2 job='Datenblatt
            einpflegen und Klasse festlegen' linkto=datenblatt %}

        {% url 'webapp:svdispo' dokunde=kunde.firmenname as svdispo %}
        {% include "jobs/BO-job-display.html" with jobstatus=3 job='Sachverständigen
            disponieren' linkto=svdispo %}

        {% url 'webapp:sverklaerung' dokunde=kunde.firmenname as sverklaerung %}
        {% include "jobs/BO-job-display.html" with jobstatus=4 job='Vollmacht und
            Erklärung mit disponiertem SV hochladen' linkto=sverklaerung %}

        {% url 'webapp:foerderantrag' dokunde=kunde.firmenname as foerderantrag %}
        {% include "jobs/BO-job-display.html" with jobstatus=5 job='Förderantrag stellen
            und einpflegen' linkto=foerderantrag %}

    </div>
{% endblock %}
```

Listing 6.7: Die Datei vorbereitungBO.html [Eigene Darstellung]

In Listing 6.7 ist die Verwendung des in Listing 6.6 gezeigten Templates zu sehen. Die jeweilige URL wird vor jeder Inklusion des parametrierten Templates in einer lokalen Variablen gespeichert, um die Übergabe an das inkludierte Template zu ermöglichen. Dies ist notwendig, da die Parameter eines Templates nur durch Leerzeichen getrennt übergeben werden, die URL selbst jedoch über einen eigenen Parameter verfügt, welcher bei direkter Übergabe als zusätzlicher Parameter des Templates interpretiert werden würde. Das Ergebnis der Kombination der in Listing 6.6 und Listing 6.7 gezeigten Templates für einen Kunden mit Status „4 – Vorbereitung durch Backoffice“ ist in Abbildung 6.1 zu sehen.

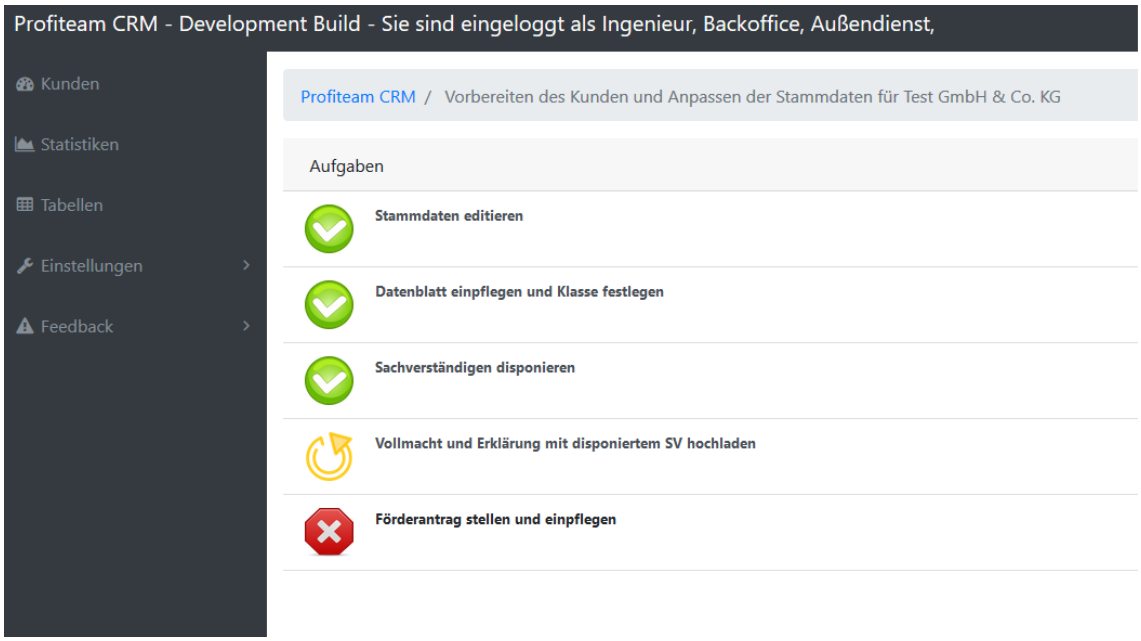

Abbildung 6.1: Übersicht der Teilprozesse für Backoffice-Mitarbeiter im Zuge der Auftragsvorbereitung (Ausschnitt) [Eigene Darstellung]

6.1.3 Dateiverwaltung

Eine besondere Rolle für das Unternehmen nimmt die Dateiverwaltung ein, da das bisherige System, basierend auf Google Drive, mit zunehmender Anzahl an Clients und Daten inperformanter wurde. Die Implementierung der vom Stakeholder gewünschten Funktionalität wird in den folgenden Abschnitten erläutert.

6.1.3.1 Dateistruktur

Die bisherige Dateistruktur sah je Kunden einen Ordner vor, indem alle Dokumente zu diesem Kunden abgelegt wurden. Dazu gehörte auch E-Mail-Korrespondenz unter den beteiligten Akteuren. Dieses System wurde, die Ordnerstruktur betreffend, beibehalten, jedoch wird die E-Mail-Korrespondenz durch das implementierte Nachrichtensystem nahezu vollständig abgelöst. Resultierend daraus müssen in Zukunft lediglich .pdf, .doc bzw .docx sowie .jpg Dateien auf dem Server gespeichert werden.

6.1.3.1.1 Generierung der internen Ordnerstruktur

Der Upload von Dateien geschieht immer über ein spezifisches Formular, welches an einen Teilprozess gekoppelt ist, bei dem im Zuge der Bearbeitung Daten anfallen, die nicht direkt in der Datenbank gespeichert werden sollen. Um die Integration innerhalb der Anwendung zu ermöglichen, wurde ein „Wrapper"-Model entwickelt, welches Metadaten zur tatsächlich vorhandenen Datei speichert und Django zur Verfügung stellt.

```
[...]
def get_upload_path(instance, filename):

    ext = filename.split('.')[-1]
    if ext in valid_extensions:
        return "Kunden/{name}/{dateiname}.{extension}"
        .format(
        name=str(instance.kunde.pk) + '_' + instance.kunde.firmenname,
        dateiname=str(instance.kunde.pk) + '_' + instance.filename,
        extension=ext
        )
    else:
        raise ValidationError(u'Ungültiges Dateiformat!')

[...]
class FileModel(models.Model):
    kunde = models.ForeignKey(Kunde, on_delete=models.CASCADE, null=False)
    filename = models.CharField(max_length=100, blank=False, null=False)
    file = models.FileField(upload_to=get_upload_path)
    status = models.PositiveSmallIntegerField(default=1)
    type = models.CharField(max_length=100, blank=False, null=False)

    def set_type(self):
        ext = self.file.name.split('.')[-1]
        if ext in valid_extensions:
            if ext == 'pdf':
                self.type = 'application/pdf'
            elif ext == 'docx' or ext == 'doc':
                self.type = 'application/msword'
            elif ext == 'jpg':
                self.type = 'image'
            else:
                self.type = 'UNDEFINED'
        self.save()
[...]
```

Listing 6.8: Dateiverarbeitung in Django – Auszug aus models.py [Eigene Darstellung]

In Listing 6.8 sind die relevanten Ausschnitte aus der Datei models.py für die implementierte Dateiverarbeitung gezeigt. Zunächst wurde das FileModel erstellt, welches der Verwaltung der Dateien dient. Für jede, über ein Uploadformular hinzugefügte Datei, wird nach erfolgreichem Upload eine FileModel-Instanz erzeugt. Im FileModel werden der Fremdschlüssel zum zugehörigen Kunden-Model, der Dateiname innerhalb des Systems, der relative Pfad zur Datei sowie der Status der Datei gespeichert. Außerdem wird der MIME-Type für die Bereitstellung eines späteren Downloads gespeichert. Durch die Verwendung der Funktion **get_upload_path** werden automatisch der interne Dateiname sowie der Speicherpfad gesetzt. Um systemweit eindeutige Pfade zu erhalten, wird sowohl beim Namen des Kundenordners als auch beim Dateinamen selbst der Primärschlüssel des Kunden eingefügt.

Somit ist ausgeschlossen, dass eine Datei im Ordner eines anderen Kunden gespeichert wird, der zufällig den gleichen Namen besitzt. Die hochgeladenen Dateien werden vor dem Speichern gemäß dem Attribut „filename“ umbenannt. Der Status der Datei gibt an, ob sie aktuell oder veraltet ist und wird automatisch vom System gesetzt, wenn im Zuge des Prozesses eine neuere Version eingefügt wird. Veraltete Dateien sind weiterhin über die Benutzeroberfläche zugänglich, werden allerdings optisch von aktuellen Dateien abgegrenzt.

6.1.3.2 Upload

Der Upload von Dateien ist nur im Zuge der Ausführung von Teilprozessen möglich, wobei der Anwender in jedem der entsprechenden Teilprozesse auf die hinzuzufügenden Dateien hingewiesen wird.

6.1.3.2.1 Dateiupload-Form

Der Upload von Dateien, die mit einem Teilprozess anfallen, geschieht immer innerhalb einer einzigen, für diesen Teilprozess erstellten Form.

```
[...]
class VertragZWBUploadForm(forms.Form):
    vertrag = forms.FileField(required=True, label='Unterschriebener Vertrag',
        allow_empty_file=False, max_length=100, widget=forms.FileInput(attrs={'accept':'
        application/pdf'}), validators=[checkFileSize])
    zwb = forms.FileField(required=True, label='Zuwendungsbescheid', allow_empty_file=
        False, max_length=100, widget=forms.FileInput(attrs={'accept':'application/pdf'
        }), validators=[checkFileSize])
    class Meta:
        fields = "__all__"
[...]
```

Listing 6.9: Eine Form für den Dateiupload – Auszug aus forms.py [Eigene Darstellung]

In Listing 6.9 ist die Definition der Datei-Uploadform für den Teilprozess „Übergabe von unterschriebenem Vertrag und Zuwendungsbescheid durch den Außendienstmitarbeiter“ zu sehen. Es werden lediglich pdf-Dateien bis zu einer maximalen Länge von 100 Zeichen für den Dateinamen akzeptiert. Über den Validator „checkFileSize“ ist sichergestellt, dass nur Dateien bis zu 10 MiB Dateigröße vom Server akzeptiert werden. Leere Dateien werden ebenfalls abgelehnt.

6.1.3.2.2 Dateiupload-Template

Die Benutzeroberfläche für diesen Dateiupload wird aus dem Template in Listing 6.10 generiert.

```
{% extends "layout2.html" %}
{% block content %}
{% load staticfiles %}
<form enctype="multipart/form-data" id="form" action="" method="post">
    {% csrf_token %}
    <table class="table table-border">
    {% for field in form %}
        <tr>
        {{ field.errors }}
            <td>{{ field.label_tag }}
            {% if field.field.required %}<span class="required">*</span>{% endif %}
            </td><td> {{ field }} </td>
        </tr>
    {% endfor %}
    </table>
    <div style="text-align: center">
    <a class="tooltips" data-toggle="tooltip" data-placement="top" title="Absenden">
        <button type="submit" name="send" onclick="return confirm('Sind Sie sicher?');">
         Absenden
        </button>
    </a>
    </div>
</form>
{% endblock %}
```

Listing 6.10: Das Template zum betrachteten Teilprozess [Eigene Darstellung]

Dem Listing 6.10 ist zu entnehmen, dass die Eingabefelder für den Dateiupload dynamisch instanziiert werden und unterschieden wird, ob es sich beim betrachteten Feld der Form um ein erforderliches Feld (required=True) oder ein optionales Feld (required=False) handelt[1]. Im ersten Fall wird dem Namen des Feldes (label_tag) ein * angestellt, in letzterem nicht. Zu beachten ist, dass das Attribut *enctype* der Form den Wert „multipart/form-data“ enthält, da sonst die Kodierung der Daten beim Upload nicht korrekt erfolgt. Die Einreichung der Daten geschieht über das Klicken eines Buttons auf der Benutzeroberfläche, was eine zusätzliche Sicherheitsabfrage auslöst, die dem versehentlichen Versand der Form-Daten vorbeugen soll.

[1] Siehe Listing 6.9

6.1.3.2.3 Dateiupload-View

Im Folgenden wird ein Ausschnitt aus der View gezeigt, die mit der Form und dem Template aus den vorherigen Abschnitten korrespondiert.

```
def vertragupload(request, dokunde):
    [...]
        if request.method == 'POST':
            vzwbform = VertragZWBUploadForm(request.POST, request.FILES)
            if vzwbform.is_valid():
                vertrag = FileModel()
                vertrag.kunde = workonkunde
                vertrag.filename = 'Vertrag unterschrieben'
                vertrag.file = request.FILES['vertrag']
                vertrag.set_type()
                vertrag.save()

                zwb = FileModel()
                zwb.kunde = workonkunde
                zwb.filename = "Zuwendungsbescheid"
                zwb.file = request.FILES['zwb']
                zwb.set_type()
                zwb.save()
                workonkunde.status = 7
                workonkunde.save()
                messages.success(request, 'Vertrag und Zuwendungsbescheid wurden
                    erfolgreich gespeichert')
                createAlarm(workonkunde, Group.objects.all().get(name='Backoffice'), 7,
                    'Eingang Vertrag und Zuwendungsbescheid', 2)
                setAlarmDone(workonkunde, 'Vertragsangebot wurde erstellt')
            else:
                messages.error(request, 'Die Dateien müssen als .pdf vorliegen!')
     [...]
```

Listing 6.11: View für das Handling eines Dateiuploads – Auszug aus views.py [Eigene Darstellung]

In Listing 6.11 ist der relevante Ausschnitt aus der View „vertragupload“ zu sehen. Die Überprüfung von Nutzerrechten, die Initialisierung von lokalen Variablen sowie der abschließende Code zum Rendern des Templates wurden der Übersichtlichkeit halber entfernt.
Zunächst werden die Daten aus der eingereichten Form auf Gültigkeit überprüft. Dies wird mit der Methode is_valid(), die von Django automatisch generiert wird, überprüft. Im Fall der Gültigkeit der Daten werden für die beiden, in der Form übertragenen, Dateien FileModels instanziiert und die entsprechenden Daten werden in die Instanz übertragen. Nach Speichern der Instanzen wird der Anwender über das Messages-System von Django via „flash message“ darüber informiert, ob der Dateiupload erfolgreich war. Im Anschluss wird ein Alarm für den zuständigen Mitarbeiter im Backoffice generiert, der ein Ablaufdatum von 7 Tagen hat und ihn zum Erledigen des nächsten Teilprozesses auffordert. Letztlich wird noch der Alarm, der den Außendienstmitarbeiter aufgefordert hat die Dateien hochzuladen, deaktiviert. Nähere Informationen zu Alarmen sind Abschnitt 6.1.5 zu entnehmen.

6.1.3.3 Download

Der Download von Dateien kann nicht direkt erfolgen, sondern wird über eine spezielle View bereitgestellt, welche im Folgenden analysiert wird.

```
[...]
def download(request, dokunde, file):
    assert isinstance(request, HttpRequest)
    fileObject = FileModel.objects.all().filter(kunde__firmenname__exact=dokunde)
    fileObject = fileObject.get(filename__exact=file)
    if fileObject and fileObject.file:
        if isMember(request) and isWorker(request, Kunde.objects.get(firmenname=dokunde)
            ):
            if os.path.exists(os.path.join(settings.MEDIA_ROOT, fileObject.file.path)):
                path = os.path.join(settings.MEDIA_ROOT, fileObject.file.path)
                with open(path, 'rb') as f:
                    response = HttpResponse(f.read(),
                        content_type=fileObject.type)
                    response['Content-Disposition'] = 'inline; filename=' +os.path.
                        basename(path) return response

            raise Http404
        raise PermissionDenied()
    raise Http404
[...]
```

Listing 6.12: View für das Handling von Downloads – Auszug aus views.py [Eigene Darstellung]

In Listing 6.12 ist die Definition der View zum Herunterladen von Dateien zu sehen. Bevor die Datei zum Download angeboten wird, wird zunächst überprüft, ob die Dateireferenz in der Datenbank existiert. Wenn dem so ist, wird anschließend geprüft, ob der Benutzer angemeldet und Mitarbeiter ist. Danach folgt die Prüfung, ob der Mitarbeiter an der Auftragsverarbeitung für diesen Kunden beteiligt ist. Erst dann wird versucht die tatsächliche Datei bereitzustellen. Da der MIME-Type der Datei in der FileModel-Instanz gespeichert wurde, kann er nun für die Antwort vom Server wiederverwendet werden. Ein direkter Download der Dateien durch unbefugte Personen ist durch die verwendete Vorgehensweise ausgeschlossen.

6.1.4 Nachrichtensystem

In diesem Abschnitt wird die Implementierung des Nachrichtensystems beschrieben, welches genutzt wird, um die kundenspezifische Kommunikation der Akteure zu bündeln und zu dokumentieren. Zunächst wird die Definition des Nachrichtenmodels beschrieben. Im Anschluss erfolgt die Beschreibung der Nutzung innerhalb der Software.

6.1.4.1 Definition einer Nachricht

Der Nachrichtenaustausch erfolgt, wie in der Konzeptionierungsphase geplant, kundenbezogen. Im nachfolgenden Listing ist der Quelltext des Nachrichtenmodels zu sehen:

```
class Nachricht(models.Model):
    author = models.ForeignKey(User, on_delete=models.PROTECT)
    kunde = models.ForeignKey(Kunde, on_delete=models.CASCADE)
    message = models.CharField(max_length=500, blank=False)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
    is_read = models.BooleanField(default=False)
    ing_has_read = models.BooleanField(default=False)
    vertrieb_has_read = models.BooleanField(default=False)
    backoffice_has_read = models.BooleanField(default=False)

        def save(self, *args, **kwargs):
        if not self.is_read and self.ing_has_read and self.vertrieb_has_read and \
        self.backoffice_has_read:
            self.is_read = True
        super().save(*args, **kwargs)


    @python_2_unicode_compatible
    def __str__(self):
        formatedDate = self.created_at.strftime("%d.%m.%Y -- %H:%M")

        return self.author.username + ", " + formatedDate + ":  " + self.message
```

Listing 6.13: Model zur Realisierung von Nachrichten – Auszug aus models.py [Eigene Darstellung]

In Listing 6.13 ist die Definition des Nachrichtenmodels zu sehen. Eine Nachricht wird über zwei Fremdschlüssel immer einem Autor sowie einem Kunden zugeordnet. Das Löschen des User-Objekts des Autors wird unterbunden, wenn noch Nachrichten von diesem Benutzer existieren[2]. Beim Löschen des mit der Nachricht korrespondierenden Kunden-Objekts wird auch die Nachricht gelöscht. Für die Nachricht selbst, die ein Nutzer speichert, stehen 500 Zeichen zur Verfügung, wobei leere Nachrichten nicht erlaubt sind. Für jede Nachricht werden automatisch das Erstelldatum sowie das Datum der letzten Änderung gespeichert. Weiterhin sind drei Boolean-Felder im Nachrichtenmodel enthalten, die es ermöglichen für jede Nachricht zu überprüfen, welcher Akteur sie bereits gelesen hat. Das Feld is_read wird automatisch beim Speichern des Models gesetzt, wenn alle gruppenspezifischen has_read Felder wahr sind. Die Methode __str__() mit dem Decorator *@python_2_unicode_compatible*[3] definiert den Rückgabewert, wenn das Objekt zum Datentyp String konvertiert wird.

[2] Das Löschen von Benutzern ist aufgrund der von Django implementierten Benutzerverwaltung nicht notwendig, da Benutzerkonten auf den Status „inaktiv“ gesetzt werden können, was jegliche Aktionen durch dieses Konto unterbindet.

[3] Der Decorator dient der Abwärtskompatibilität mit Python2. Unter Benutzung von Python3 hat er keine Funktion.

6.1.4.2 Verwendung des Models

Der Datenbankzugriff über das Nachrichtenmodel ist sowohl für die Anzeige von existierenden Nachrichten als auch für das Hinzufügen neuer Nachrichten erforderlich. Im Folgenden wird sowohl gezeigt, wie den Benutzer betreffende Nachrichten gefiltert werden als auch wie neue Nachrichten durch Benutzer erstellt werden.

6.1.4.2.1 Bestimmung neuer Nachrichten

Neue Nachrichten werden für einen Benutzer als solche gekennzeichnet, wenn das Booleanfeld, das mit der Gruppe des Benutzers korrespondiert, auf false gesetzt ist. Die entsprechend notwendigen Funktionen werden im unten stehenden Listing 6.14 gezeigt und im Anschluss erklärt.

```
def getKunden(request):
    kding = Kunde.objects.none()
    kdad = Kunde.objects.none()
    kdbo = Kunde.objects.none()
    meinekunden = Kunde.objects.none()
    if isIng(request.user):
        kding = Kunde.objects.filter(ingenieur__username__exact=request.user.username)
    if isAD(request.user):
        kdad = Kunde.objects.filter(kundeVertrieb__username__exact=request.user.username
            )
    if isBO(request.user):
        kdbo = Kunde.objects.filter(kundeBackoffice__username__exact=request.user.
            username)
        neuekunden = Kunde.objects.filter(kundeBackoffice__username__exact=None)
        kdbo = (kdbo | neuekunden)

    meinekunden = (kding | kdad | kdbo).distinct()
    return meinekunden

def getNachrichten(request, meinekunden):
    msging = Nachricht.objects.none()
    msgad = Nachricht.objects.none()
    msgbo = Nachricht.objects.none()
    meineNachrichten = None
    if isIng(request.user):
        msging= Nachricht.objects.all().filter(kunde__in=meinekunden).filter(
            ing_has_read=False)
    if isAD(request.user):
        msgad = Nachricht.objects.all().filter(kunde__in=meinekunden).filter(
            vertrieb_has_read=False)
    if isBO(request.user):
        msgbo = Nachricht.objects.all().filter(kunde__in=meinekunden).filter(
            backoffice_has_read=False)

    meineNachrichten = (msging | msgad | msgbo).distinct().order_by('-created_at')
    return meineNachrichten
```

Listing 6.14: Bestimmung neuer Nachrichten - Auszug aus common_functions.py [Eigene Darstellung]

In Listing 6.14 sind die Definitionen der Funktionen *getKunden()* sowie *getNachrichten()* zu sehen. Die Funktion getKunden() extrahiert aus allen Kunden genau jene, bei denen der Benutzer ein Akteur im Prozess ist. Eine Ausnahme bilden hierbei neue Kunden, bei denen noch kein Mitarbeiter des Backoffice-Teams disponiert wurde. Diese Kunden werden allen Mitarbeitern des Backoffices angezeigt. Die Verwendung des Operators „|“ mit den gruppenspezifischen Querysets bewirkt eine UNION-Verknüpfung der Abfragen, welche für Nutzer mit mehreren Rollen erforderlich ist[4]. Zu beachten ist auch hier,

[4]Damit ist insbesondere der Administrator gemeint.

dass innerhalb der Funktion keine Datenbankabfrage stattfindet, sondern lediglich ein resultierendes Queryset gebildet wird.

In der Funktion *getNachrichten()* werden, analog zur Bestimmung der Kunden die ein Nutzer betreut, die ungelesenen Nachrichten extrahiert, die für den Nutzer bestimmt sind. Diese werden nach Erstelldatum absteigend sortiert zurückgegeben[5].

6.1.4.2.2 Übergabe an Templates

Der Aufruf der in Listing 6.14 gezeigten Funktionen erfolgt innerhalb des Contextprocessors „Always“, welcher im Abschnitt 6.1.1.3.2 gezeigt wurde. Die Übergabe an alle Templates ist notwendig, da die Verwendung der Daten innerhalb des Layout-Templates stattfindet und diese somit in jeder View benötigt werden.

6.1.4.2.3 Benutzung innerhalb der Templates

Nachrichten werden in der App grundsätzlich an zwei unterschiedlichen Stellen angezeigt. Zum einen werden die drei neusten Nachrichten über das Drop-Down-Menü im Layout-Template angezeigt, zum anderen werden auf der Übersichtsseite des Kunden alle kundenspezifischen Nachrichten visualisiert. Im Folgenden wird zuerst die Verwendung im Layout-Template gezeigt und im Anschluss der Nachrichtenbereich der Kundenübersicht analysiert.

```
<div class="dropdown-menu" aria-labelledby="messagesDropdown">
        <h6 class="dropdown-header">{{newsCount}} neue Nachrichten:</h6>
        {% for nachricht in news %}
        <div class="dropdown-divider"></div>
                <a class="dropdown-item" href="{% url 'webapp:bearbeitekunde'
                        % dokunde=nachricht.kunde.firmenname %}">
                <strong>{{nachricht.author}} : {{nachricht.kunde.firmenname}}</strong>
                <span class="small float-right text-muted">{{nachricht.created_at |
                        date:"d.m.Y H:i"}}</span> <div class="dropdown-message
                                small">{{nachricht.message}}</div> </a>
        {% endfor %}
        <div class="dropdown-divider"></div>
        <a class="dropdown-item small" href="{% url 'webapp:nachrichten' %}">Alle
        % Nachrichten ansehen</a>
</div>
```

Listing 6.15: Anzeige der Nachrichten im Layout (Ausschnitt) [Eigene Darstellung]

In Listing 6.15 ist der Ausschnitt des Templates zum Rendern des Layouts zu sehen, der für das Nachrichten-Drop-Down Menü zuständig ist. Für jede in „news“ übergebene Nachricht wird ein neues Element im Menü angelegt. Beim Klick auf eine Nachricht wird der Benutzer zu der Übersichtsseite des Kunden weitergeleitet, zu dem die Nachricht gehört.

[5] D. h. die neuste Nachricht ist das erste Element des Resultats.

```
[...]
{% for nachricht in kundennachrichten %}
        <div class="list-group-item list-group-item-action">
                <div class="media">
                        <div class="media-body wordwrap">
                                <strong>{{ nachricht.author }}: {{ nachricht.message }}
                                    </strong>
                                <div class="text-muted smaller">Vom {{ nachricht.
                                    created_at}}</div>
                        </div>
                        {% if request.user == nachricht.author %}
                        <div id="trash">
                                <form action="" method="post">
                                        {% csrf_token %}
                                        {{ deletemessage }}
                                        <input type="hidden" name="item_id" value="{{
                                            nachricht.id}}"/>
                                        <a class="tooltips" data-toggle="tooltip" data-
                                            placement="top"
                                           title="Nachricht löschen">
                                                <button type="submit" name="delete"
                                                    onclick="return confirm('Nachricht
                                                    löschen? ');"
                                                            style="border: 0;
                                                                background: none;">
                                                    <i class="fa fa-trash"></i>
                                                </button>
                                        </a>
                                </form>
                        </div>
                        {% endif %}
                </div>
        </div>
{% endfor %}
[...]
```

Listing 6.16: Anzeige der Nachrichten in der Kundenübersicht (Ausschnitt) [Eigene Darstellung]

In Listing 6.16 ist der Ausschnitt des Templates zum Rendern der Kundenübersicht zu sehen, der für die Generierung der Anzeige von kundenspezifischen Nachrichten verantwortlich ist. Die Renderengine iteriert über alle Elemente, die im Context als „kundennachrichten“ übergeben wurden. Wenn der Autor einer Nachricht der anfragende Benutzer ist, wird zusätzlicher Code generiert, der ein Papierkorb-Symbol neben der Nachricht anzeigt. Ein Klick auf dieses Symbol führt zum Löschen der entsprechenden Nachricht[6]. Zur Spezifizierung der Nachricht wird der Primärschlüssel der Nachricht als “hidden input“ beim Rendern des Templates der Form hinzugefügt.

6.1.4.3 Erstellen von Nachrichten

In diesem Abschnitt werden die Erstellung von Nachrichten beschrieben und die relevanten Ausschnitte aus dem Quellcode erklärt. Im Folgenden wird zunächst die Definition der Form zum Erfassen von Nachrichten gezeigt. Im Anschluss wird der Ausschnitt aus dem Template gezeigt, in dem die Form genutzt wird. Danach werden die relevanten Ausschnitte der zugehörigen View gezeigt und das Verhalten beschrieben.

[6] Die entsprechende Überprüfung, ob der anfragende Benutzer tatsächlich der Autor ist, erfolgt aus Sicherheitsgründen erneut auf Seiten des Servers.

6.1.4.3.1 Form

Im Folgenden wird die Form gezeigt und beschrieben, die für das Erfassen neuer Nachrichten genutzt wird:

```
class NachrichtKundeForm(ModelForm):
    class Meta:
        model = Nachricht
        fields = ['message', ]
        labels = {
            'message': 'Neue Nachricht',
        }
```

Listing 6.17: Form zum Verarbeiten von Nachrichten – Auszug aus forms.py [Eigene Darstellung]

In Listing 6.17 ist die Definition der implementierten Form zu sehen. Es handelt sich um eine Form, die von ModelForm erbt und somit als verbindliches Attribut in der Subklasse *Meta* das zugehörige Model benötigt. Ohne Spezifizierung, welche Felder und Labels des Models genutzt werden sollen, wären alle Felder durch den Benutzer editierbar. Durch die Spezifizierung auf das Feld „message“ ist es dem Benutzer jedoch nur möglich die tatsächliche Nachricht zu übergeben. Über den Kontext „labels“ werden den spezifizierten Feldern explizit für eine Form Namen[7] gegeben, die dem Eingabefeld voranstehen.

6.1.4.3.2 Template

Im Folgenden wird der Ausschnitt aus dem Template zur Kundenübersicht gezeigt, der für das Erstellen neuer Nachrichten verantwortlich ist.

```
[...]
<form action="" method="post" id="send_msg">
        {% csrf_token %}
        {{ form }}
        <a class="tooltips" data-toggle="tooltip" data-placement="top" title="Nachricht
                senden">
                <button type="submit" name="send" onclick="return confirm('Nachricht
                        senden? ');" style="border: 0; background: none;">
                <i class="fa fa-plus"></i>
                </button>
        </a>
</form>
[...]
```

Listing 6.18: Erstellen von Nachrichten in der Kundenübersicht (Ausschnitt) [Eigene Darstellung]

In Listing 6.18 ist der Ausschnitt aus dem Template der Kundenübersicht zu sehen, in dem die Form zur Eingabe neuer Nachrichten enthalten ist. Im Gegensatz zu vorher gezeigten Einbindungen von Django Forms wird an dieser Stelle auf ein Zerlegen der Form in ihre Bestandteile verzichtet. Die Angabe von {{ form }} erzeugt für jedes Feld einer Form eine Tabellenzeile mit den Spalten Feldtitel und Eingabefeld. In diesem Fall rendert {{ form }} zum in Listing 6.19 angegebenen Quelltext.

```
<tr>
        <th><label for="id_message">Neue Nachricht:</label></th>
        <td>
                <input type="text" name="message" maxlength="500"
                required id="id_message"/>
        </td>
</tr>
```

Listing 6.19: Expandierter Templatecode (Ausschnitt) [Eigene Darstellung]

[7] sog. label tags

6.1.4.3.3 View

Im Folgenden wird der Ausschnitt aus der View zur Kundenübersicht gezeigt, der für die Verarbeitung von neuen Nachrichten zuständig ist.

```
[...]
if request.method == 'POST':
        if 'send' in request.POST:
                form = NachrichtKundeForm(request.POST)
                if form.is_valid():
                        nachricht = form.save(commit=False)
                        nachricht.author = request.user
                        nachricht.kunde = workonkunde
                        if isAD(request.user):
                                nachricht.vertrieb_has_read = True
                        if isIng(request.user):
                                nachricht.ing_has_read = True
                        if isBO(request.user):
                                nachricht.backoffice_has_read = True
                        nachricht.save()
                        return HttpResponseRedirect(request.path_info)
                else:
                        form = NachrichtKundeForm
                        deletemessage = deletemessage
[...]
```

Listing 6.20: Ausschnitt aus der View zur Kundenübersicht – Verarbeiten von Nachrichten [Eigene Darstellung]

In Listing 6.20 ist der Quelltext zu sehen, der bei einer POST-Anfrage der Kundenübersicht ausgeführt wird. Hervorzuheben ist hier die Zeile *nachricht = form.save(commit=False)* mit der zwar die Daten der Form in der Variablen „nachricht“ gespeichert werden, der Datenbankzugriff aber verhindert wird. Dadurch ergibt sich im Anschluss die Möglichkeit, Attribute der noch unvollständigen Model-Instanz zu setzen. Erst nachdem die Daten in der View vervollständigt wurden, wird die Nachricht in der Datenbank gespeichert.

6.1.5 Alarmsystem

In diesem Abschnitt wird die Implementierung des Alarmsystems beschrieben, welches genutzt wird, um Nutzer über von ihnen auszuführende Aktionen zu informieren. Alarme steuern den Verlauf des gesamten Prozesses. Weiterhin dienen Alarme dem automatisieren Controlling, inbesondere im Hinblick auf zukünftig verstreichende Fristen. Im Folgenden wird zunächst die Definition des Alarmmodels gezeigt. Im Anschluss wird die Verwendung innerhalb der Software an einem Beispiel erläutert.

6.1.5.1 Definition eines Alarms

In unten stehendem Listing 6.21 ist die Definition des Models zu sehen, das dem konzeptionierten Alarm entspricht.

```
[...]
ALARMTYPE = (

    (1, 'Erinnerung'),
    (2, 'Arbeitsauftrag'),
    (3, 'Sonstiges'),
)
[...]
class Alarm(models.Model):
    kunde = models.ForeignKey(Kunde, on_delete=models.CASCADE, default=None)
    type = models.PositiveSmallIntegerField(choices=ALARMTYPE, default=3)
    topic = models.CharField(max_length=50)
    created_at = models.DateTimeField(auto_now_add=True)
    alarmtime = models.DateTimeField()
    isDone = models.BooleanField(default=False)
    targetGroup = models.ForeignKey(Group, on_delete=models.CASCADE, default=None)

    @python_2_unicode_compatible
    def __str__(self):
        formatedDate = self.alarmtime.strftime("%d.%m.%Y -- %H:%M")
        createDate = self.created_at.strftime("%d.%m.%Y -- %H:%M")

        return self.Kunde + ": " + self.topic + "fällig am " + formatedDate + ",
            erstellt am " + createDate
[...]
```

Listing 6.21: Model zur Realisierung von Alarmen – Auszug aus models.py [Eigene Darstellung]

In Listing 6.21 ist zunächst die Definition der Alarmtypen zu sehen. Diese Unterscheidung ist notwendig, da das Alarmsystem mehrere, voneinander unabhängige Funktionen erfüllt. Alarme werden stets über einen Fremdschlüssel mit einem Kunden verknüpft und beinhalten in 50 Zeichen eine kurze Beschreibung des Alarms. Weiterhin werden Erstelldatum und die Alarmzeit, also die Zeit bis der Alarm erledigt sein muss, gespeichert. Alarme werden grundsätzlich nicht gelöscht, sondern nur als erledigt markiert. Ziel eines Alarms ist immer ein einzelner Benutzer, der indirekt über den Kunden und die Gruppenzugehörigkeit bestimmt wird[8]. Auch hier wurde eine __str__() Methode definiert, die den Alarm in einem gut lesbaren Format zurückgibt.

6.1.5.2 Verwendung des Models

In diesem Abschnitt wird die Verwendung von Alarmen beschrieben. Zunächst werden zwei Hilfsfunktionen gezeigt, die benötigt werden um den Quelltext übersichtlich zu halten.

[8] Dieses Vorgehen dient der Laufzeitverbesserung, da bei einem Query der Kunden die anhängenden User-Objekte der Akteure nicht mitgeladen werden müssen.

```
[...]
def createAlarm(kunde, targetgroup, frist, topic, type):
    alarm = Alarm()
    alarm.kunde = kunde
    alarm.targetGroup = targetgroup
    alarm.alarmtime = date.today() + timedelta(days=frist)
    alarm.topic = topic
    alarm.type = type
    alarm.save()

def setAlarmDone(kunde, topic):
    alarm = Alarm.objects.all().get(kunde=kunde, topic=topic)
    alarm.isDone = True
    alarm.save()

[...]
```

Listing 6.22: Hilfsfunktionen zur Benutzung von Alarmen – Auszug aus common_functions.py [Eigene Darstellung]

In Listing 6.22 sind zwei Hilfsfunktionen zu sehen, die genutzt werden um Alarme zu erstellen bzw. als erledigt zu markieren. Zu beachten ist, dass erst die Kombination aus Kunde und Topic ein Alarmobjekt eindeutig identifiziert. Die Benutzung der Funktionen im Kontext des Abschlusses eines Teilprozesses wurde bereits in Listing 6.11 im Abschnitt 6.1.3.2.3 gezeigt und erklärt.

6.1.5.3 Anzeige von Benutzernamen

Bei Django werden bei der Ausgabe von User-Objekten grundsätzlich die Benutzernamen angezeigt. Dieses Verhalten ist für die entwickelte Anwendung unerwünscht. Anstelle der manuellen Ersetzung des Benutzernamens durch Vor- und Nachnamen an allen notwendigen Stellen musste eine Lösung gefunden werden, die den Aufwand minimal hält und die Wartbarkeit gewährleistet. Django bietet hier die Möglichkeit über die Methode **add_to_class** nachträglich Attribute und Methoden in Klassen von Dritten zu installieren oder zu überschreiben. Die Anwendung zur Lösung dieses Problems ist in Listing 6.23 zu sehen.

```
[...]
@python_2_unicode_compatible
def get_name(self):
    if self.first_name or self.last_name:
        return self.first_name + ' ' + self.last_name
    else:
        return self.username

User.add_to_class("__str__", get_name)
[...]
```

Listing 6.23: Nachträgliches Überschreiben der __str__ Methode des Usermodels – Auszug aus models.py [Eigene Darstellung]

Alternativ zum gezeigten Vorgehen wäre es auch möglich gewesen, ein neues Usermodel zu erstellen, das von *django.contrib.auth.models.User* erbt und die zusätzliche Methode implementiert. Dies hätte allerdings den Nachteil, dass das neue Model inkompatibel zum Authentifizierungssystem von Django wäre, was eine Überarbeitung des gesamten Quelltextes erfordern würde. Durch Verwendung der gezeigten Lösung ist auch eine Kompatibilität beim Aktualisieren von Django selbst gewährleistet.

6.2 Automatisierung der Vertragsgenerierung

Nach der Konzeptionierung der automatisierten Energieberichtserstellung sollte die Implementierung erfolgen. Durch eine kurzfristige Änderung des Zeitplans durch den Stakeholder wurde die Implementierung einer automatisierten Vertragserstellung außerplanmäßig priorisiert.
Dementsprechend wurde zunächst die Analyse und Konzeptionierung zur Vertragsgenerierung vorangestellt. Die Vorgehensweise gestaltete sich äquivalent zur Planung und Konzeptionierung der automatisierten Energieberichtserstellung.

Der zu automatisierende Vertrag umfasst ein zweiseitiges Dokument, welches kundenspezifische Daten sowie Daten, die abhängig vom bearbeitenden Sachverständigen sind, enthält.
Die Dateistruktur des Projekts gestaltet sich analog zu der Konzeptionierung beim Energiebericht (5.4.3). Aufgrund des geringeren Umfangs des zu erzeugenden Dokuments werden, abgesehen von der Haupt-Datei [9], keine weiteren .tex-Dateien benötigt.

6.2.1 Aufbau des Vetrags

Die variablen Teile des Vertrags beschränken sich auf folgenden Punkte:

- Die Anschrift des Kunden
- Die Energieverbrauchsklasse (angegeben in kWh/Jahr)
- Die Kontaktdaten und BAFA-Nummer des Sachverständigen
- Die Unterschriftsfelder
- Das aktuelle Datum

Die restlichen Inhalte sind statisch und können in der Datei Vertragsangebot.tex direkt eingefügt werden.

[9] hier: Vertragsangebot.tex

6.2.2 Vorgehensweise

Zunächst wurde das Vorgehen mit einer statisch angelegten und mit Beispieldaten gefüllten data-config.tex getestet:

```
[...]
% Daten zur Energieverbrauchsklasse
\newcommand{\Energieverbrauchsklasse}{C}
\newcommand{\maxWertKWHA}{150.000}
\newcommand{\minWertKWHB}{150.001}
\newcommand{\maxWertKWHB}{500.000}

% Anschrift des Kundens
\newcommand{\Firma}{Testfirma GmbH}
\newcommand{\Str}{Teststraße}
\newcommand{\Hnr}{1}
\newcommand{\zusatz}{a}
\newcommand{\Plz}{12345}
\newcommand{\Ort}{Musterstadt}
[...]
```

Listing 6.24: Auszug der lokal angelegten data-config.tex des Vertrags [Eigene Darstellung]

Es werden eigene Makros (siehe 2.2.2.3) verwendet, die in der Vetragsangebot.tex aufgerufen werden.
Die Werte *maxWertKWHA*, *minWertKWHB* und *maxWertKWHB* dienen einer If-Abfrage im LaTeX-Dokument zur Anpassung der Überschrift. Je nach vorliegender Energieverbrauchsklasse werden die dazugehörigen Bedingungen (kWh/Jahr) angepasst.

```
\begin{center}
        \large\textbf{Auftrag zur Durchführung einer
        Energieberatung}\\(Energieverbrauchsklasse \
            Energieverbrauchsklasse\ -
                \ifx\Energieverbrauchsklasse\EBMA
                        bis zu \maxWertKWHA\ kWh/Jahr)
                \fi
                \ifx\Energieverbrauchsklasse\EBMB
                        \minWertKWHB\ bis \maxWertKWHB\ kWh/Jahr)
                \fi
                \ifx\Energieverbrauchsklasse\EBMC
                        über \maxWertKWHB\ kWh/Jahr)
                \fi
\end{center}
```

Listing 6.25: If-Abfrage bzgl. der Energieverbrauchsklasse [Eigene Darstellung]

Die if-Abfrage, die im Listing 6.25 verwendet wird (\ifx) ist ein vordefinierter TeX-Befehl, siehe Abschnitt 2.2.8 Bedingte Ausführung.

6.2.3 Layout

Das Layout des Vertrags war vorgegeben und wurde in LaTeX adaptiert. Die Anpassungen wurden in der general-config.tex vorgenommen und beziehen sich auf Schriftart, Kopfzeile, Fußzeile und Seitenformat. Es wurden unter anderem folgende Pakete verwendet:

- Dokumentenklasse:
  - scrartcl
- Schrift:
  - inputenc
  - helvet
  - setspace
- Kopf- und Fußzeile:
  - fancyhdr
  - graphicx
  - lastpage
- Format:
  - geometry
  - xcolor

```
\fancyhead[L]{\includegraphics[height=1.6cm]{Graphics/sgs.png}\
    hfill\includegraphics[height=1.6cm]{Graphics/tuev.png}\hspace
    {-1.6cm}}

\fancyfoot[L]{\color{lightGrey}{Seite \thepage\ von \pageref{
    LastPage}}}
```

Listing 6.26: Kopf- und Fußzeile mit fancyhdr gestaltet [Eigene Darstellung]

Listing 6.26 zeigt einen Ausschnitt aus der general-config.tex. Hier werden Kopf- und Fußzeile des Dokuments definiert. Die Kopfzeile enthält linksbündig das SGS-Logo, rechtsbündig das Tüv-Saar-Logo. In der Fußzeile wird rechtsbündig die aktuelle und die Gesamtseitenzahl angezeigt.

```
\usepackage{geometry}[includehead, includefoot,headheight=5080pt]
 \geometry{a4paper,left=25mm,right=25mm,top=35mm,bottom=25mm}
```

Listing 6.27: Nutzung des Pakets geometry [Eigene Darstellung]

Mit dem Paket *geometry* aus Listing 6.27 werden die Seitenverhältnisse angepasst (Seitenabstände, Größe von Fuß- und Kopfzeile).

```
\usepackage{xcolor}
\definecolor{lightGrey}{rgb}{0.5, 0.5, 0.5}
\definecolor{lighterGrey}{rgb}{0.8,0.8,0.8}
```

Listing 6.28: Definition eigener Farben mit dem Paket colorx [Eigene Darstellung]

Die definierten Farben aus Listing 6.28 werden u.a. für die Schriftfarbe der Seitenangabe der Fußzeile genutzt (siehe 6.26).

6.2.4 Datenerfassung

Für den Energiebericht ist eine explizite Dateneingabe vorgesehen. Bei der automatisierten Vertragsgenerierung werden die Daten implizit erhoben, d.h. nicht explizit im Zuge eines Teilprozesses erhoben, sondern aus Daten mehrerer Teilprozesse zusammengetragen.

Die Daten werden aus folgenden Funktionen der views.py extrahiert:

- datenblatt
- stammdaten

Die nachfolgende Tabelle B.1 vermittelt einen Überblick über die genutzten Makros in LaTeX (data-config.tex), die korrespondierenden Daten in Django und die Herkunft der Daten.

6.2.5 Vorbereitung der individuellen Generierung

Für den individuellen, auf den jeweiligen Kunden angepassten Vertrag, wird die data-config.tex aus Python-Code erzeugt. Die erzeugte Konfigurationsdatei wird analog zum lokalen Test in das LaTeX-Projekt eingebunden.

Um die Datei data-config.tex zu füllen, wird zunächst die Funktion addcommand definiert:

```
def addcommand(self, latexcommand, val, file):
        file.write("\\newcommand{\\" + latexcommand + "}{" +
        self.escape_latexcommand(str(val)).replace('None', '') + "
            }\n")
```

Listing 6.29: Auszug aus der Klasse zum Füllen der config-data.tex [Eigene Darstellung]

Die Funktion wird genutzt, um die benötigten Variablen per *\newcommand* in einer Datei anzulegen.
Mit Ausführen der Funktion *addcommand* wird der definierten Datei (file) eine Zeile hinzugefügt. *latexcommand* steht hier für den Namen des Makros. Der zugewiesene Wert *val* wird aus der Datenbank extrahiert. Im Anschluss wird dieser zu einem String konvertiert und LaTeX-spezifische Sonderzeichen (z.B. &, % etc.) werden mit der Funktion *escape_latexcommand* ersetzt. Anschließend werden Einträge mit „None“, d.h. ohne hinterlegten Wert, durch einen leeren String ersetzt.
Der Einsatz der Funktion *addcommand* gestaltet sich wie folgt:

```
        try:
            self.addcommand("Energieverbrauchsklasse", ebmklasse.
                type, file)
            self.addcommand("Pauschalkosten", ebmklasse.
                pauschalkosten,file)
            self.addcommand("Firma", kunde.get_firmenname(), file)
[...]
```

Listing 6.30: Benutzung der Methode addcommand [Eigene Darstellung]

6.2.6 Generierung und Einfügen in Django

Um pdflatex auf dem Server auszuführen, ohne die Django Applikation durch eine synchrone Anfrange unnötig zu belasten, wird ein Subprozess aus der Django-Funktion gestartet:

```
subprocess.Popen( (
                                    'pdflatex' +
                                    outputname +
                                    pdfParam +
                                    osseparator +

                                    'pdflatex' +
                                    outputname +
                                    pdfParam +
                                    osseparator +

                                    cleanerprocess
                                    ), shell=True)
```

Listing 6.31: Starten der Subroutine zum Erstellen der .pdf Datei [Eigene Darstellung]

In Listing 6.31 ist der Aufruf der Subroutine zu sehen, die für die Ausführung von pdflatex sowie das anschließende Verschieben der erzeugten Datei verantwortlich ist.

6.2.6.1 Zusammensetzung der Subroutine

Als Argument für *subprocess.Popen()* dient ein String, der auch analog in der Konsole eingegeben werden könnte. Dieser String setzt sich aus mehreren Komponenten zusammen, die im aktuellen Abschnitt beschrieben werden.

6.2.6.1.1 outputname

outputname ist der ausgefüllte parameter *–jobname* von pdflatex. Näheres hierzu ist der man-page von pdflatex[10] zu entnehmen. Gleichzeitig dient der Parameter der Definition des Dateinamens, da es vorkommen könnte, dass gleichzeitig mehrere Dateien kompiliert werden.

6.2.6.1.2 pdfparam

Hier wird zur Kompilierzeit angegeben, aus welcher Datei die erstellten Makros zu entnehmen sind.

```
pdfparam = " \def\dataconfig{" + configPath.replace('&', '^&') +"}\
    input{vertragEBM.tex}"
```

Listing 6.32: Übergabe der data-config.tex zur Kompilierzeit als Latexmakro [Eigene Darstellung]

In Listing 6.32 ist die Definition des Übergabeparameters zu sehen. Der „Umweg“ über ein Makro ist notwendig, da es nicht möglich ist, die zu kompilierenden Dateien auf direktem Weg an pdflatex zu übergeben. Der Pfad der data-config.tex wird via Makro übergeben, welches nach der Ausführung durch pdflatex die Datei data-config.tex nachlädt. Die Ersetzung von &-Zeichen im Pfad zur data-config.tex ist notwendig, da anonsten das Zeichen vom Kommandozeileninterpreter verarbeitet wird.

[10] https://linux.die.net/man/1/pdflatex

6.2.6.1.3 osseperator

osseperator bezeichnet eine vom Betriebssystem abhängige Zeichenfolge, die i.d.R. von der entsprechenden Konsole zur Trennung von Kommandos benutzt wird.
Unter Windows ist dies durch die Folge ' && ' gegeben. Unter Linux wird ' ; ' genutzt.

6.2.6.1.4 cleanerprocess

Der String *cleanerprocess* beschreibt eine Folge von (betriebssystemabhängigen) Kommandozeilenbefehlen, die die Ausgabedatei in den korrekten Ordner verschiebt und die erzeugten LaTeX-Hilfsdateien löscht.

6.2.6.1.5 doppeltes Kompilieren

Für dieses Dokument muss *pdflatex* zweimal ausgeführt werden, um sicherzustellen, dass der Verweis auf \pageref{lastPage} funktioniert:

```
\fancyfoot[L]{\color{lightGrey}{Seite \thepage\ von \pageref{
    LastPage}}}
```

Listing 6.33: Abhängigkeit von LastPage in der Definition des Footers [Eigene Darstellung]

Bei einmaligem Kompilieren steht in der Fußzeile beispielsweise „Seite 1 von ?“.
Erst beim zweiten Durchlauf wird die Gesamtanzahl der Seiten korrekt dargestellt, da pdflatex zur Kompilierzeit ohne Hilfsdateien keine Informationen zur Größe des Dokuments zur Verfügung stehen. Dies gilt auch für Verzeichnisse und Querverweise, welche im Energiebericht vorhanden sind.
Das Argument *shell=true* ist notwendig, um eine Verkettung von Kommandos und damit auch die korrekte Abfolge der Befehle zu gewährleisten. Würde für jedes Kommando ein neuer Prozess gestartet werden, könnte die zeitliche Reihenfolge der Fertigstellung nicht gewährleistet werden.

6.2.6.2 Einfügen in Django

Nachdem pdflatex ausgeführt wurde, wird der Vertrag den Dateien des Kunden über ein FileModel hinzugefügt und steht somit anschließend zum Download bereit:

```
[...]
vertrag = FileModel()
vertrag.kunde = kunde
vertrag.filename = "Vertragsangebot"
vertrag.file.name = "Kunden/{name}/{filename}".format(name=dokunde,
    filename=filename)
vertrag.set_type()
vertrag.save()
[...]
```

Listing 6.34: Packen der erstellten Datei in einem FileModel [Eigene Darstellung]

In Listing 6.34 ist die Einbindung der bereits existierenden Datei in das in Django implementierte Dateisystem zu sehen. Da in der Datenbank lediglich der Pfad zur Datei gespeichert wird, ist eine Einbindung extern erstellter Dateien ohne Probleme möglich. Die erstellte PDF kann nun über die Kundenübersicht von den berechtigten Benutzern heruntergeladen werden.

6.3 Automatisierung des Energieberichts

Nach der Vertragsautomatisierung, welche als Testlauf für das erarbeitete Konzept diente, beginnt die Automatisierung des Energieberichts. Die grundsätzliche Vorgehensweise gestaltet sich äquivalent zur Vertragsgenerierung, wobei die Datenerhebung selbst einen großen Umfang des Arbeitsaufwandes einnimmt. In diesem Abschnitt werden die implementierten Lösungen gezeigt und beschrieben.

6.3.1 Layout

Als Vorlage für das optische Layout des Energieberichts wurde, in Abstimmung mit der Geschäftsführung, ein abgeschlossener Energiebericht von Dipl.-Ing. (Univ.) Olaf Singendonk verwendet, der sich mit der Verwendung einverstanden erklärt hat. In diesem Abschnitt wird die Adaption des vorgegebenen Layouts in LaTeX beschrieben.

6.3.1.1 Format

Die Dokumentenklasse des Energieberichts ist *scrreprt* des KOMA-Scripts mit den gängigen Parametern „oneside“, „paper=a4“ und der Schriftgröße „fontsize=12pt“.
Außerdem verfügt der zu generierende Energiebericht, abweichend von der genutzten Vorlage, über Inhalts-, Abbildungs- und Tabellenverzeichnis. Die Titelseite des Energieberichts wird auch mit kundenspezifischen Daten gefüllt.

6.3.1.2 Titelseite

Das Layout der Titelseite ist gesondert zur Gestaltung des restlichen Dokuments zu betrachten.

```
\begin{titlepage}\linespread{1.5}\selectfont
\begin{minipage}[t]{0.5\textwidth}
\begin{flushleft}
\vspace{0pt}
\hspace*{-10pt}
\textcolor{lightGrey}{SGS-TÜV Saar GmbH}
\newline
\hspace*{-10pt}
\textcolor{lightGrey}{Am TÜV 1}
\newline
\hspace*{-10pt}
\textcolor{lightGrey}{66280 Sulzbach}
\end{flushleft}
\end{minipage}
\begin{minipage}[t]{0.5\textwidth}
\begin{flushright}
\vspace{0pt}
\includegraphics[width=0.35\textwidth]{Graphics/tuev.png}
\hspace*{10pt}
\end{flushright}
\end{minipage}
```

Listing 6.35: Gestaltung der Kopfzeile der Titelseite des Energieberichts [Eigene Darstellung]

In Listing 6.35 wird die Kopfzeile der Titelseite gestaltet. Linksbündig ist die Adresse der Firma SGS-TÜV-Saar GmbH eingebunden, rechtsbündig das TÜV-Logo. Die benötigten Abstände wurden mit Hilfe der Makros \vspace und \hspace realisiert.

```
\begin{center}
       \Huge\colorbox{lightGrey}{\makebox(460,
       30)[c]{\vspace{-5pt}\color{white} Energieberatung
          Mittelstand}}
                \newline
                \newline
                \linespread{1.0}
                \large\textcolor{lightGrey}{im Rahmen des
                   Förderprogramms des BAFA nach}
                \newline
                \hspace*{50pt}\large\textcolor{lightGrey}{\DinNorm}
                \newline
                \newline
                \hspace*{40pt}\large\textcolor{lightGrey}{
                   ABSCHLUSSBERICHT}
                \newline
                \hspace*{-50pt}\large\textcolor{lightGrey}{
                   Energieberatung Mittelstand: EBM
                \EBMNr}\\
                \begin{figure}[htpb!]
                \center
                        \includegraphics[width=0.95\textwidth]{\
                           pathImageStandort}
                \end{figure}
                \linespread{1.5}
                \normalsize\textcolor{lightGrey}{\Firma}\\
                \normalsize\textcolor{lightGrey}{\Str\ \Hnr}\\
                \normalsize\textcolor{lightGrey}{\Plz\ \Ort}\\
                \vspace*{40pt}
                \small\textcolor{lightGrey}{Erstellt am: \today}
    \end{center}
```

Listing 6.36: Verwendung des Pakets fancyhdr im Energiebericht [Eigene Darstellung]

Der in Listing 6.36 zu sehende Quelltext beschreibt das Layout des Bodys der Titelseite des Energieberichts. Dieser besteht aus einer Überschrift innerhalb einer Colorbox, einem Bild des untersuchten Standortes und der Adresse der Firma.

6.3.1.3 Kopf- und Fußzeilen

Um die Kopf- und Fußzeile der Vorlage entsprechend zu gestalten, wurde, wie auch bei der Vertragsgenerierung, das Paket *fancyhdr* verwendet.

```
\usepackage{fancyhdr}
\pagestyle{fancyplain}

\rhead{\includegraphics[height=1.6cm]{Graphics/tuev.png}}

\renewcommand{\headrulewidth}{0pt}
\renewcommand{\footrulewidth}{5pt}
\renewcommand{\footrule}{\hbox to\headwidth{\color{lighterGrey}\
   leaders\hrule
height \footrulewidth\hfill}}

\lfoot{\color{lightGrey}{Audit: \Firma}}
\rfoot{\color{lightGrey}{Seite \thepage\ von \pageref{LastPage}}}
```

Listing 6.37: Verwendung des Pakets fancyhdr im Energiebericht [Eigene Darstellung]

6.3.1.4 Überschriften

```
\newcommand{\colorchapter}[1]{%
  \colorbox{lightGrey}{\parbox{\dimexpr\textwidth-2\fboxsep}{\
     textcolor{white}\thechapter\
  \textcolor{white}{#1}}}}
```

Listing 6.38: Definition der Kapitelüberschriften [Eigene Darstellung]

In Listing 6.38 wird ein neues Makro angelegt, das genutzt wird, um den Kapitelüberschriften den in der Vorlage vorhandenen, eingefärbten grauen Rahmen hinzuzufügen. Unter Verwendung des Pakets *colorx* wird eine „colorbox“ mit hellgrauer Farbe angelegt. Darin wird eine „parbox“ definiert (Box mit Blocksatz) und die Schriftfarbe wird für diesen Befehl zu weiß geändert. Die angegebenen Parameter der parbox dienen der Größenanpassung (hier nahezu Textbreite).
Analog zum Vorgehen in Listing 6.38 werden eingefärbte Rahmen auch bei Sections und Subsections hinzugefügt. Hierzu werden *colorsection* und *colorsubsection* definiert. Die Farbgebung wurde an zentraler Stelle in der Datei general-config.tex implementiert, um einfache Änderbarkeit zu gewährleisten.

```
\titleformat{name=\chapter}[block]
  {\sffamily\Large}
  {}
  {0pt}
  {\colorchapter}
\titlespacing*{\chapter}{0pt}{\baselineskip}{\baselineskip}
```

Listing 6.39: Einbinden des neuen Makros über das Paket Titlesec [Eigene Darstellung]

In Listing 6.39 wird das neu erstellte Makro \colorchapter mit Hilfe des Paketes *titlesec* auf alle Kapitelüberschriften angewandt.

Colorbox wurde auf ähnliche Weise für viele Layout-Anpassungen genutzt, u.a. für Anmerkungen, die immer das gleiche Format haben.

6.3.2 Datenverarbeitung

In diesem Abschnitt werden die Datenaufnahme und -verarbeitung betrachtet, die einen großen Teil der Implementierung für die Automatisierung des Energieberichts eingenommen haben. Die Erhebung der Daten erfolgt auf Basis der bereits in Abschnitt 4.1.2.1 erwähnten Excel-Tools, welche für die Django-Applikation in einzelne Formulare und Eingabemasken aufgetrennt wurden. Beispielhaft wird hier die Implementierung einiger Formulare gezeigt, die direkt zur Erstellung des Energieberichts beitragen.

6.3.2.1 Energetische Gesamtsituation

Teil des Energieberichts ist die Darstellung der energetischen Gesamtsituation des betrachteten Unternehmens innerhalb der letzten Jahre. Zu diesem Zweck existiert im zu Grunde liegenden Excel-Tool eine Tabelle, in der entsprechende Verbrauchskennzahlen eingetragen wurden.

6.3.2.1.1 Model

Für die Aufnahme der Daten in der Django-Applikation wurde ein Model definiert, das einen solchen Datensatz aufnehmen kann.

```
class Jahresverbrauch(models.Model):
    kunde = models.ForeignKey(Kunde, on_delete=models.CASCADE, default=None)
    jahr = models.PositiveSmallIntegerField(validators=[MaxValueValidator(2200),
        MinValueValidator(1800)], blank=False)
    strom = models.FloatField(validators=[MinValueValidator(0.0)], blank=True)
    stromkosten = models.FloatField(validators=[MinValueValidator(0.0)], blank=True)
    gas = models.FloatField(validators=[MinValueValidator(0.0)], blank=True, null=True,
        default=0.0)
    gaskosten = models.FloatField(validators=[MinValueValidator(0.0)], blank=True, null=
        True, default=0.0)
    oel = models.FloatField(validators=[MinValueValidator(0.0)], blank=True, null=True,
        default=0.0)
    oelkosten = models.FloatField(validators=[MinValueValidator(0.0)], blank=True, null=
        True, default=0.0)
    fernwaerme = models.FloatField(validators=[MinValueValidator(0.0)], blank=True, null
        =True, default=0.0)
    fernwaermekosten = models.FloatField(validators=[MinValueValidator(0.0)], blank=True
        , null=True, default=0.0)
    fluessiggas = models.FloatField(validators=[MinValueValidator(0.0)], blank=True,
        null=True, default=0.0)
    fluessiggaskosten = models.FloatField(validators=[MinValueValidator(0.0)], blank=
        True, null=True, default=0.0)
    pellets = models.FloatField(validators=[MinValueValidator(0.0)], blank=True, null=
        True, default=0.0)
    pelletskosten = models.FloatField(validators=[MinValueValidator(0.0)], blank=True,
        null=True, default=0.0)
    sonstiges = models.FloatField(validators=[MinValueValidator(0.0)], blank=True, null=
        True, default=0.0)
    sonstigeskosten = models.FloatField(validators=[MinValueValidator(0.0)], blank=True,
         null=True, default=0.0)

    treibstoff = models.FloatField(validators=[MinValueValidator(0.0)], blank=True, null
        =True, default=0.0)
    treibstoffkosten = models.FloatField(validators=[MinValueValidator(0.0)], blank=True
        , null=True, default=0.0)
    trfluessiggas = models.FloatField(validators=[MinValueValidator(0.0)], blank=True,
        null=True, default=0.0)
    trfluessiggaskosten = models.FloatField(validators=[MinValueValidator(0.0)], blank=
        True, null=True, default=0.0)
[...]
```

Listing 6.40: Attribute des Models Jahresverbrauch [Eigene Darstellung]

In Listing 6.40 ist der erste Teil der Definition des Models zur Aufnahme eines Datensatzes für einen Jahresverbrauch zu sehen. Zur besseren Übersichtlichkeit wurden Attribute und Methoden in getrennten Listings dargestellt.

```
[...]
    def OelToKWH(self):
        if not self.oel:
            return 0.0
        else:
            return self.oel * TREIBSTOFF_TO_KWH

    def FlGasHeizToKWH(self):
        if not self.fluessiggas:
            return 0.0
        else:
            return self.fluessiggas * FLGAS_TO_KWH

    def PelletsToKWH(self):
        if not self.pellets:
            return 0.0
        else:
            return self.pellets * PELLETS_TO_KWH

    def TreibstoffToKWH(self):
        if not self.treibstoff:
            return 0.0
        else:
            return self.treibstoff * TREIBSTOFF_TO_KWH

    def TrFlGasToKWH(self):
        if not self.trfluessiggas:
            return 0.0
        else:
            return self.trfluessiggas * TREIBSTOFF_TO_KWH

    # ohne Fuhrpark
    def getGesamtKWH(self):
        return self.OelToKWH() + self.FlGasHeizToKWH() + self.PelletsToKWH() + self.
            strom + self.sonstiges + self.gas

    def getGesamtKWHFurhpark(self):
        return self.TreibstoffToKWH() + self.TrFlGasToKWH()

    # ohne Fuhrpark
    def getGesamtKosten(self):
        return self.stromkosten + self.gaskosten + self.oelkosten
        + self.fernwaermekosten + self.fluessiggaskosten
        + self.pelletskosten + self.sonstigeskosten

    def getGesamtKostenFuhrpark(self):
        return self.treibstoffkosten + self.trfluessiggaskosten
```

Listing 6.41: Methoden des Models Jahresverbrauch [Eigene Darstellung]

Um die ursprüngliche Tabelle korrekt abzubilden, wurden die in Listing 6.41 abgebildeten Methoden implementiert, die für den Energiebericht nötige Berechnungen durchführen. Die zu sehenden Umrechnungsfaktoren wurden aus dem Excel-Tool übernommen, ebenso wie die verwendeten Formeln.

6.3.2.1.2 Form

Die Definition der Form für die Eingabe von Jahresverbräuchen ist in unten stehendem Listing 6.42 zu sehen.

```
class JahresVerbrauchForm(ModelForm):
    class Meta:
        model = Jahresverbrauch
        fields = "__all__"
        exclude = ['kunde', 'gesamtkwh', 'gesamtkwhfuhrpark',]
        labels = {
            'strom': 'Strom [kWh}',
            'gas': 'Gas [kWh]',
            'trfluessiggas': 'Treibstoff Flüssiggas [Liter]',
            'fluessiggas': 'Flüssiggas [Liter]',
            'pellets': 'Pellets [kg]',
            'sonstiges': 'Sonstiges [kWh]',
            'treibstoff': 'Treibstoff [Liter]',
            'fernwaerme': 'Fernwärme [kWh]',
            'oel': 'Öl [Liter]',
        }
```

Listing 6.42: Form zur Aufnahme von Jahresverbräuchen [Eigene Darstellung]

In Listing 6.42 ist die Definition der Form zu sehen. Grundsätzlich werden alle Felder des Models Jahresverbrauch für die Form freigegeben. Durch Angabe des Attributs *exclude* in der Subklasse Meta, werden die definierten Felder aus der Form entfernt. Im Attribut *labels* werden die Beschreibungen inklusive Einheiten für die Eingabefelder hinterlegt.

6.3.2.1.3 Template

Bei der Benutzung der Form innerhalb eines Templates wurde darauf geachtet, dass die Darstellung der Form visuell dem zu Grunde liegenden Excel-Tool nahe kommt. Im folgenden Listing wird das entsprechende Template gezeigt.

```
<form id="form" action="" method="post">
    {% csrf_token %}
    <div>
        <strong>Jahr: </strong> {{form.jahr}}
    <a class="tooltips" data-toggle="tooltip" data-placement="top" style="margin-left:
        40px" title="Absenden">
       <button type="submit" name="send" onclick="return confirm('Sind Sie sicher?');">
           Absenden
       </button>
    </a>
    </div>
     <table class="table table-bordered" id="jahresverbrauchTable"  cellspacing="0"
         style="margin-top: 20px; float: left;">
          <thead>
                <tr>
                    <th>Energieträger</th>
                    <th>Verbrauch</th>
                    <th>Kosten [€]</th>
                </tr>
          </thead>
                {% for field in form %}
                        {% if field != form.jahr %}
                                {% if forloop.counter|divisibleby:2 %}
                                <tr>
                                        <td>{{field.label_tag}}</td>
                                {% endif %}
                                        <td>{{field}}</td>
                                {% if forloop.counter0|divisibleby:2 %}
                                        </tr>
                                {% endif %}
                        {% endif %}
                {% endfor %}
          <tr> <td>Gesamt [kWh]</td><td><strong>wird berechnet</strong></td><td><strong>
              wird berechnet</strong></td></tr>
          <tr> <td>Fuhrpark Gesamt [kWh]</td><td><strong>wird berechnet</strong></td><td>
              <strong>wird berechnet</strong></td></tr>
     </table>
</form>
 {% for entry in jvdatasets %}
 <table class="table table-bordered jahresverbrauchreview"  cellspacing="0" style="
     margin-top: 20px; float: left;
 {% if forloop.counter0|divisibleby:2 %}background-color: #f2f2f2; {% endif %}    ">
          <thead>
                <tr>
                    <th>Verbrauch in {{entry.jahr}}</th>
                    <th>Kosten [€] in {{entry.jahr}}</th>
                </tr>
          </thead>
     <tr><td>{{entry.strom}}</td><td>{{entry.stromkosten}}</td></tr>
     <tr><td>{{entry.gas}}</td><td>{{entry.gaskosten}}</td></tr>
     <tr><td>{{entry.oel}}</td><td>{{entry.oelkosten}}</td></tr>
     <tr><td>{{entry.fernwaerme}}</td><td>{{entry.fernwaermekosten}}</td></tr>
     <tr><td>{{entry.fluessiggas}}</td><td>{{entry.fluessiggaskosten}}</td></tr>
     <tr><td>{{entry.pellets}}</td><td>{{entry.pelletskosten}}</td></tr>
     <tr><td>{{entry.sonstiges}}</td><td>{{entry.sonstigeskosten}}</td></tr>
     <tr><td>{{entry.treibstoff}}</td><td>{{entry.treibstoffkosten}}</td></tr>
     <tr><td>{{entry.trfluessiggas}}</td><td>{{entry.trfluessiggaskosten}}</td></tr>
     <tr><td>{{entry.getGesamtKWH}}</td><td>{{entry.getGesamtKosten}}</td></tr>
     <tr><td>{{entry.getGesamtKWHFurhpark}}</td><td>{{entry.getGesamtKostenFuhrpark}}</
         td></tr>
 </table>
 {% endfor %}
```

Listing 6.43: Template zur Aufnahme von Jahresverbräuchen [Eigene Darstellung]

Das in Listing 6.43 dargestellte Template besteht aus zwei Teilen. Zum einen ist im oberen Bereich die Implementierung der Form zu sehen, zum anderen wird im unteren Bereich für jeden bereits bestehenden Eintrag eine Tabelle gerendert. Für die Darstellung der Form wurde das sich wiederholende Muster „Energieträger, Energiekosten“ innerhalb der Anordnung der Attribute des Models ausgenutzt. Aufgrund diese Umstandes werden unter Benutzung des Zählers der For-Schleife (forloop.counter, beginnend bei 1 sowie forloop.counter0, beginnend bei 0), die über die Felder iteriert und jeweils zwei Eingabefelder in einer Zeile gerendert, bevor eine neue Zeile begonnen wird. Das Ergebnis nach Rendern ist in Abbildung B.1 zu sehen.
Bereits eingegebene Datensätze werden als eigenständige Tabellen neben der Form angezeigt, wobei jeder zweite Datensatz zur Verbesserung des Kontrasts grau hinterlegt ist.

6.3.2.2 Erfassung der Verbraucher

Ein weiterer zentraler Punkt bei der Datenerhebung im Zuge des Energieberichts ist die Aufführung aller Verbraucher, die sich am Standort des Kunden befinden. Erst wenn die Summe der berechneten Energieverbräuche aller aufgenommenen Verbraucher mindestens 95 % des Energiebedarfs des letzten Jahres erreicht, wird die Datenaufnahme und die damit verbundene Berechnung als gültig und aussagekräftig betrachtet. In diesem Abschnitt wird die Datenerhebung der Verbraucher für den Energiebericht beschrieben.

6.3.2.2.1 Model

Zunächst wurde ein Model für die Aufnahme der Daten eines Verbrauchers erstellt.

```
class DataSet(models.Model):
    kunde = models.ForeignKey(Kunde, on_delete=models.CASCADE, default=None)
    kategorie = models.PositiveSmallIntegerField(choices=VERBRAUCHERKATEGORIE)
    subkategorie = models.PositiveSmallIntegerField(choices=SUBKATEGORIE)
    anzahl = models.PositiveSmallIntegerField()
    beschreibung = models.CharField(max_length=200)
    baujahr = models.PositiveSmallIntegerField(blank=True, null=True, validators=[
        MaxValueValidator(2200),
        MinValueValidator(1800)
    ])
    leistung = models.FloatField(blank=False, null=False)
    energietraeger = models.PositiveSmallIntegerField(choices=ENERGIETRAEGER)
    laufzeit = models.PositiveSmallIntegerField(blank=False, null=False)
    nutzungsfaktor = models.FloatField(blank=False, null=False, validators=[
        MinValueValidator(0.0)
    ])

    def verbrauch(self):
        if not self.anzahl or not self.leistung or not self.laufzeit or not self.
            nutzungsfaktor:
            return 0.0
        else:
            return self.anzahl * self.leistung * self.laufzeit * self.nutzungsfaktor

    @python_2_unicode_compatible
    def __str__(self):
        return self.kunde.get_firmenname() + ": " + str(self.anzahl) + " " + self.
            beschreibung + " " + str(
            self.verbrauch()) + " kWh"
```

Listing 6.44: Model zur Aufnahme von Verbraucherdatensätzen [Eigene Darstellung]

In Listing 6.44 ist die Definition des Models für einen Verbraucher, gemäß dem zu Grunde liegenden Excel-Tool, zu sehen. Der Energieverbrauch wird aus den Attributen des Models in der Methode *verbrauch* berechnet.

6.3.2.2.2 Form

Die zum in Listing 6.44 dargestellten Model zugehörige Form ist in Listing 6.45 zu sehen.

```
class DataSetForm(ModelForm):
    class Meta:
        model = DataSet
        fields = "__all__"
        exclude = ['kunde',]
```

Listing 6.45: Form zur Aufnahme von Datensätzen [Eigene Darstellung]

6.3.2.2.3 Template

Um die Dateneingabe so nah wie möglich am, den Benutzern bereits bekannten, Excel-Tool zu halten, wurde eine tabellarische Darstellung der Form sowie der bereits eingegebenen Datensätze implementiert. Die Abbildung des Templates an dieser Stelle würde den Rahmen sprengen und ist deshalb unter B.1 im Anhang zu finden.

7 Evaluation

In diesem Kapitel werden die Ergebnisse der Arbeit im Hinblick auf die Erfüllung der funktionalen und nichtfunktionalen Anforderungen evaluiert. Die Django-Applikation sowie das Modul zur automatischen Dokumentengenerierung werden gemeinsam betrachtet, da letzteres als transparente Komponente in die Applikation inkludiert wurde. Zunächst werden die funktionalen und nichtfunktionalen Anforderungen auf Erfüllung überprüft.

7.1 Funktionale Anforderungen

Zur besseren Übersicht wird die Evaluation in tabellarischer Form dargestellt:

Tabelle 7.1: Evaluation der funktionalen Anforderungen [Eigene Darstellung]

Anforderung	**Typ**	**Status**
Verfügbarkeit erhobener Daten für statistische Auswertungen	Muss-Kriterium	erfüllt
Ablösung des verteilten Dateisystems Google Drive	Muss-Kriterium	erfüllt
Kommunikationsplattform für die Akteure eines Auftrags implementiert	Muss-Kriterium	erfüllt
Neuentwicklung der Tools zur Erstellung der Energieberichte	Muss-Kriterium	teilweise erfüllt
Lösung der prozesstechnischen Probleme	Muss-Kriterium	teilweise erfüllt
Standardisierung des Inhalts von Energieberichten	Muss-Kriterium	teilweise erfüllt
Automatische Generierung des Energieberichts	Muss-Kriterium	teilweise erfüllt
Einbindung der Generierung in die Django-Applikation	Muss-Kriterium	erfüllt
Datenaufnahme für den Energiebericht innerhalb der Django-Applikation	Muss-Kriterium	teilweise erfüllt
Inklusion der automatischen Rechnungsstellung	Kann-Kriterium	nicht erfüllt
Statusinformationen per E-Mail an Kunden	Kann-Kriterium	nicht erfüllt
Automatische Generierung von Statistiken	Kann-Kriterium	nicht erfüllt

7.2 Nichtfunktionale Anforderungen

Auch die Evaluation der nichtfunktionalen Anforderungen wird aus Gründen der Übersichtlichkeit in tabellerischer Form dargestellt:

Tabelle 7.2: Evaluation der nichtfunktionalen Anforderungen [Eigene Darstellung]

Anforderung	**Typ**	**Status**
Einfache Bedienbarkeit	Muss-Kriterium	erfüllt
Einbeziehung aller beteiligten Akteure	Muss-Kriterium	erfüllt
Erweiterbarkeit und Wartung	Muss-Kriterium	erfüllt
Lizenzierbarkeit bzw. Verkaufsoption	Muss-Kriterium	erfüllt
Standardisierung des Designs der Energieberichte	Muss-Kriterium	erfüllt
Transparente Einbindung des Generierungssystems	Muss-Kriterium	erfüllt
Bedienbarkeit auf Smartphone bzw. Tablet	Kann-Kriterium	teilweise erfüllt

7.3 Feedback durch die Stakeholder

Am 10.09.2018 wurde ein Prototyp der Software im Rahmen einer Mitarbeiterversammlung vorgestellt. Bei dieser Präsentation waren Personen aus allen Anspruchsgruppen vertreten. Die Reaktionen auf die Vorführung der Software waren ausnahmslos positiv. Insbesondere die automatische Vertragsgenerierung sowie das integrierte Nachrichtensystem fanden großen Zuspruch. Die (ohne ausdrücklichen Wunsch der Stakeholder) implementierte Feedback-Funktionalität wurde ebenfalls mit Begeisterung aufgenommen und lässt auf gute Annahme der neuen Software schließen.

7.4 Schlussfolgerungen und Erklärung der Ergebnisse

Die Evaluation der in den Kapiteln „Analyse der Auftragsverarbeitung“ und „Analyse des Energieberichts“ festgelegten Anforderungen zeigt, dass die Ziele der Arbeit größtenteils erfüllt wurden. Die Gründe für die lediglich teilweise Erfüllung einiger Anforderungen liegen in der Priorisierung anderer Teilaspekte durch den Stakeholder sowie in den ausführlichen und notwendigen Analyse - und Konzeptionierungsphasen, welche die verbleibende Entwicklungszeit reduzierten. So konnte zwar die Generierung der Vertragsangebote als Machbarkeitsstudie für die Generierung der Energieberichte dienen, führte jedoch andererseits zur nicht vollständigen Fertigstellung der automatisieren Berichtsgenerierung.

8 Zusammenfassung

In diesem Kapitel werden die Ergebnisse der Arbeit zusammengefasst und die mögliche zukünftige Entwicklung des Projekts beschrieben.

8.1 Stand der Entwicklung zum Abschluss dieser Arbeit

Das im Abschnitt 5.2 gesetzte Entwicklungsziel einer geschlossenen Alpha-Version konnte nicht erreicht werden. Dennoch wurde die Basis einer umfassenden Software für das Unternehmen Profiteam geschaffen, in der bereits viele, grundsätzliche Rahmenbedingungen erarbeitert und implementiert wurden. Es wurde mit der Generierung der Vertragsangebote die Durchführbarkeit der konzeptionierten Vorgehensweise für die Dokumentengenerierung bewiesen.

8.2 Zukünftige Entwicklung und Ausblick

Die in dieser Arbeit entwickelte Software wird zukünftig eine zentrale Rolle innerhalb des Unternehmens einnehmen. Ebenso wird sie weiterentwickelt und verbessert werden, um mit den Unternehmensprozessen zu wachsen. Gespräche zur Lizenzierung der Software durch das Unternehmen finden bereits statt. Bis zur Fertigstellung einer Vollversion werden – unserer persönlichen Einschätzung nach – noch mindestens 6 bis 12 Monate Entwicklungszeit benötigt. Durch das geplante, inkrementell iterative Entwicklungsmodell kann die Software allerdings bereits vor Ablauf dieser Zeitspanne verwendet werden. Durch geänderte Rahmenbedingungen, die aktuell einen massiven Anstieg der Auftragslage zur Folge haben, wird die Verwendung einer spezialisierten Software unabdingbar, da die Menge an Aufträgen auf Basis der alten Softwarelösungen und Prozessen nicht mehr bearbeitet werden kann.

8.3 Schlusswort

Zusammenfassend lässt sich feststellen, dass die frühzeitige Entwicklung von Individualsoftware positiven Einfluss auf die Prozesse des Unternehmens hat. So konnten im Rahmen dieser Arbeit Ablauffehler behoben werden und effizientere Prozesse implementiert werden. Durch die Verwendung von Technologien, die im Rahmen des Studiums nicht behandelten wurden, konnten wir uns persönlich weiterentwickeln und mit Unterstützung des Unternehmens und unseres betreuenden Dozenten relativ frei die Umsetzung des Projekts gestalten.

Wir freuen uns auf die weitere Zusammenarbeit mit dem Unternehmen und sehen der zukünftigen Entwicklung positiv entgegen.

Literatur

[1] Bundesamt für Wirtschaft und Ausfuhrkontrolle. *Energieberatung im Mittelstand*. abgerufen am 06.08.2018. 2018. URL: http://www.bafa.de/DE/Energie/Energieberatung/Energieberatung_Mittelstand/energieberatung_mittelstand_node.html.

[2] Bundesamt für Wirtschaft und Ausfuhrkontrolle. *Hinweise zur Erstellung eines Beratungsberichts*. abgerufen am 06.08.2018. 2018. URL: www.bafa.de/SharedDocs/Downloads/DE/Energie/ebm_hinweise_beratungsbericht.pdf?__blob=publicationFile&v=5.

[3] Javier Bezos. *The titlesec, titleps and titletoc Packages*. abgerufen am 27.07.2018. 2016. URL: http://tug.ctan.org/tex-archive/macros/latex/contrib/titlesec/titlesec.pdf.

[4] D. P. Carlisle. *Packages in the 'graphics' bundle*. abgerufen am 13.07.2018. 2017. URL: http://ctan.math.washington.edu/tex-archive/macros/latex/required/graphics/grfguide.pdf.

[5] DigitalOcean. *How To Serve Django Applications with Apache and mod_wsgi on Ubuntu 16.04 | DigitalOcean*. abgerufen am 18.07.2018. 2017. URL: https://www.digitalocean.com/community/tutorials/how-to-serve-django-applications-with-apache-and-mod_wsgi-on-ubuntu-16-04.

[6] Django. *Django Dokumentation - Forms*. abgerufen am 22.07.2018. 2018. URL: https://docs.djangoproject.com/en/2.0/topics/forms/.

[7] Django. *Django Dokumentation - QuerySet*. abgerufen am 23.07.2018. 2018. URL: https://docs.djangoproject.com/en/2.1/ref/models/querysets/.

[8] Django. *Django Dokumentation - Templates*. abgerufen am 22.07.2018. 2018. URL: https://docs.djangoproject.com/en/2.0/topics/templates/.

[9] Django. *Django FAQ*. abgerufen am 20.07.2018. 2018. URL: https://docs.djangoproject.com/en/dev/faq/general/.

[10] Django. *Django Hauptseite*. abgerufen am 18.07.2018. 2018. URL: https://docs.djangoproject.com/.

[11] Django. *Django Lizenz*. abgerufen am 18.07.2018. 2013. URL: https://github.com/django/django/blob/master/LICENSE.

[12] Django. *Django Tutorial - Teil 1*. abgerufen am 18.07.2018. 2018. URL: https://docs.djangoproject.com/en/2.0/intro/tutorial01/.

[13] Django. *Django Tutorial - Teil 2*. abgerufen am 22.07.2018. 2018. URL: https://docs.djangoproject.com/en/2.0/intro/tutorial02/.

[14] Django. *Django Tutorial - Teil 3*. abgerufen am 22.07.2018. 2018. URL: https://docs.djangoproject.com/en/2.0/intro/tutorial03/.

[15] Django. *Django documentation*. 2018. URL: https://docs.djangoproject.com/en/2.1/howto/deployment/checklist/.

[16] Django. *Django documentation*. abgerufen am 22.07.2018. 2018. URL: https://docs.djangoproject.com/en/2.0/ref/request-response/.

[17] Django. *Django documentation*. abgerufen am 18.07.2018. 2018. URL: https://docs.djangoproject.com/en/2.0/.

[18] Dr. Christian Feuersänger. *Notes On Programming in TEX*. abgerufen am 30.07.2018. 2018. URL: http://pgfplots.sourceforge.net/TeX-programming-notes.pdf.

[19] Nigel George. *The Model-View-Controller Design Pattern - Python Django Tutorials*. abgerufen am 25.07.2018. 2017. URL: https://djangobook.com/model-view-controller-design-pattern/.

[20] Google. *Google Maps documentation*. abgerufen am 20.07.2018. 2018. URL: https://developers.google.com/maps/documentation/urls/url-encoding.

[21] Burkhard Heidenberger. *Zitat Albert Einstein*. abgerufen am 25.09.2018. 2018. URL: https://www.zeitblueten.com/news/zitate-albert-einstein/.

[22] Markus Kohm. *KOMA-Script Documentation Project*. abgerufen am 20.07.2018. URL: https://komascript.de/.

[23] *Merkblatt für die Erstellung eines Beratungsberichts*. abgerufen am 06.08.2018. 2017. URL: http://www.bafa.de/SharedDocs/Downloads/DE/Energie/ebm_hinweise_beratungsbericht.pdf?__blob=publicationFile&v=5.

[24] David Miller. *SB-Admin Bootstrap Theme*. abgerufen am 18.07.2018. 2018. URL: https://startbootstrap.com/template-overviews/sb-admin/.

[25] Frank Mittelbach. *Der Latex-Begleiter*. München: Pearson Studium, 2005. ISBN: 978-3-86326-733-9.

[26] Piet van Oostrum. *Page layout in LaTeX*. abgerufen am 27.07.2018. 2017. URL: http://mirrors.ibiblio.org/CTAN/macros/latex/contrib/fancyhdr/fancyhdr.pdf.

[27] Matthias Pospiech. *LaTeX Variablen, If Abfragen und Schleifen*. abgerufen am 13.07.2018. 2018. URL: http://www.matthiaspospiech.de/blog/2008/04/13/latex-variablen-if-abfragen-und-schleifen/.

[28] Christine Römer. *Erste Schritte mit TeX, LaTeX und Co*. abgerufen am 13.07.2018. URL: http://www.dante.de/tex/TeXAnfaenger.html.

[29] Thomas Schlager. *LaTeX@ TU Graz - Hintergrundwissen*. abgerufen am 20.07.2018. 2013. URL: https://latex.tugraz.at/latex/hintergrund.

[30] Till Tantau. *The TikZ and PGF Packages Manual for version 3.0.1*. abgerufen am 27.07.2018. 2017. URL: http://pgf.sourceforge.net/pgf_CVS.pdf.

[31] *Tutorial für Einsteiger [LaTeX@TU Graz]*. abgerufen am 13.07.2018. URL: https://latex.tugraz.at/latex/tutorial.

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Abkürzungsverzeichnis

BAFA Bundesamt für Wirtschaft und Ausfuhrkontrolle

BHKW Blockheizkraftwerk

BSD Berkeley Software Distribution

CSS Cascading Style Sheets

EnPI Energy Performance Indikator

FTP File Transfer Protocol

GUI Graphical User Interface

HDD Hard Disk Drive

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

LaTeX Lamport TeX

LOF List Of Figures

LOT List Of Tables

MiB Mebibyte

MIME-Type Internet Media Type

MVP Model-View-Presenter

MTV Model, Template, View

PDF Portable Document Format

PV Photovoltaik

Regex Regular Expression

SQL Structured Query Language

SSD Solid State Drive

ssh Secure Shell

TOC Table Of Contents

URL Uniform Resource Locator

venv virtual environment

WSGI Web Server Gateway Interface

Anhang

A Web-Applikation

A.1 Screenshots

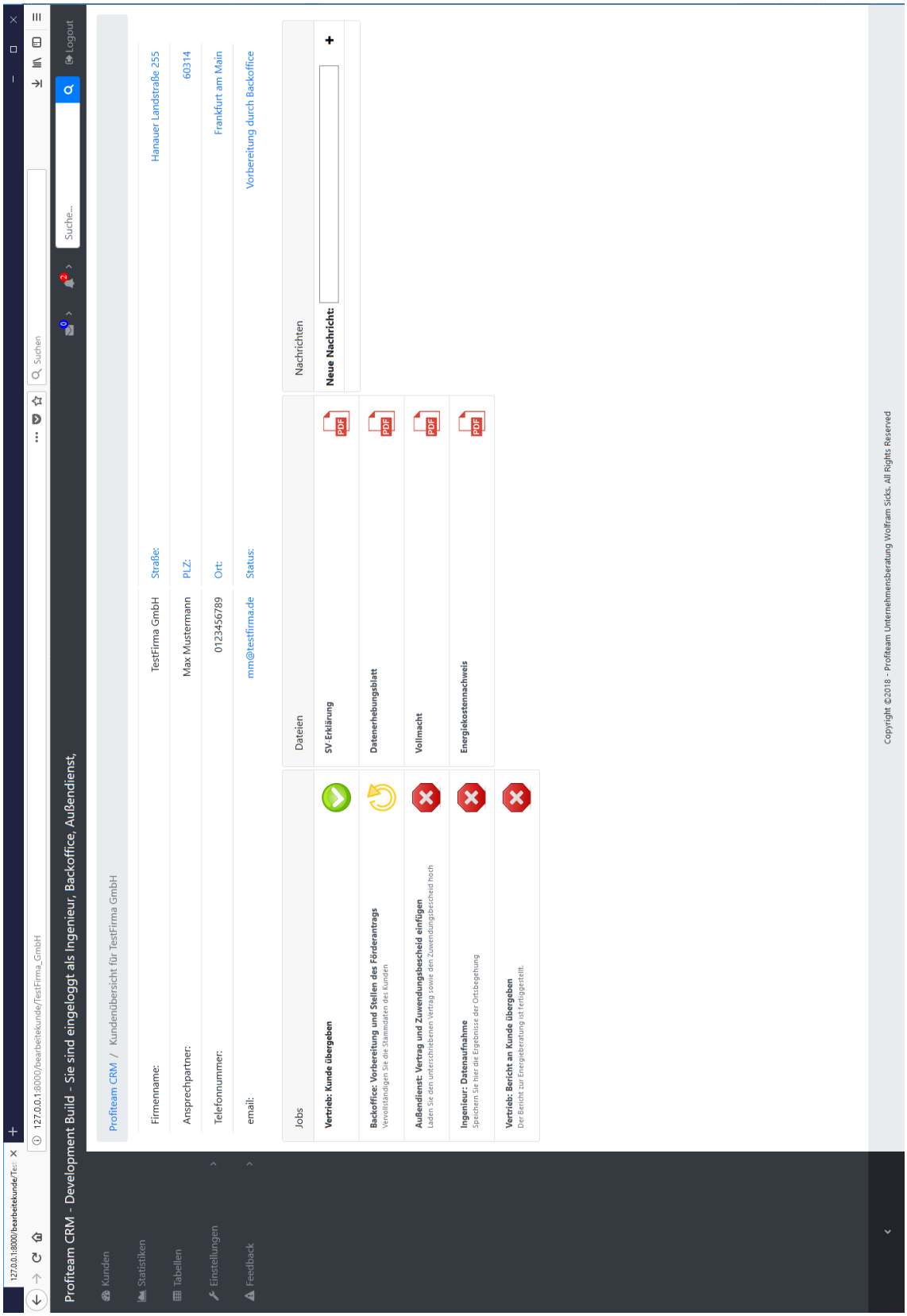

Abbildung A.1: Kundenspezifische Primäransicht innerhalb der Applikation nach Anlegen [Eigene Darstellung]

B Dokumentenautomatisierung

B.1 Datenaufnahme

B.1.1 Screenshots

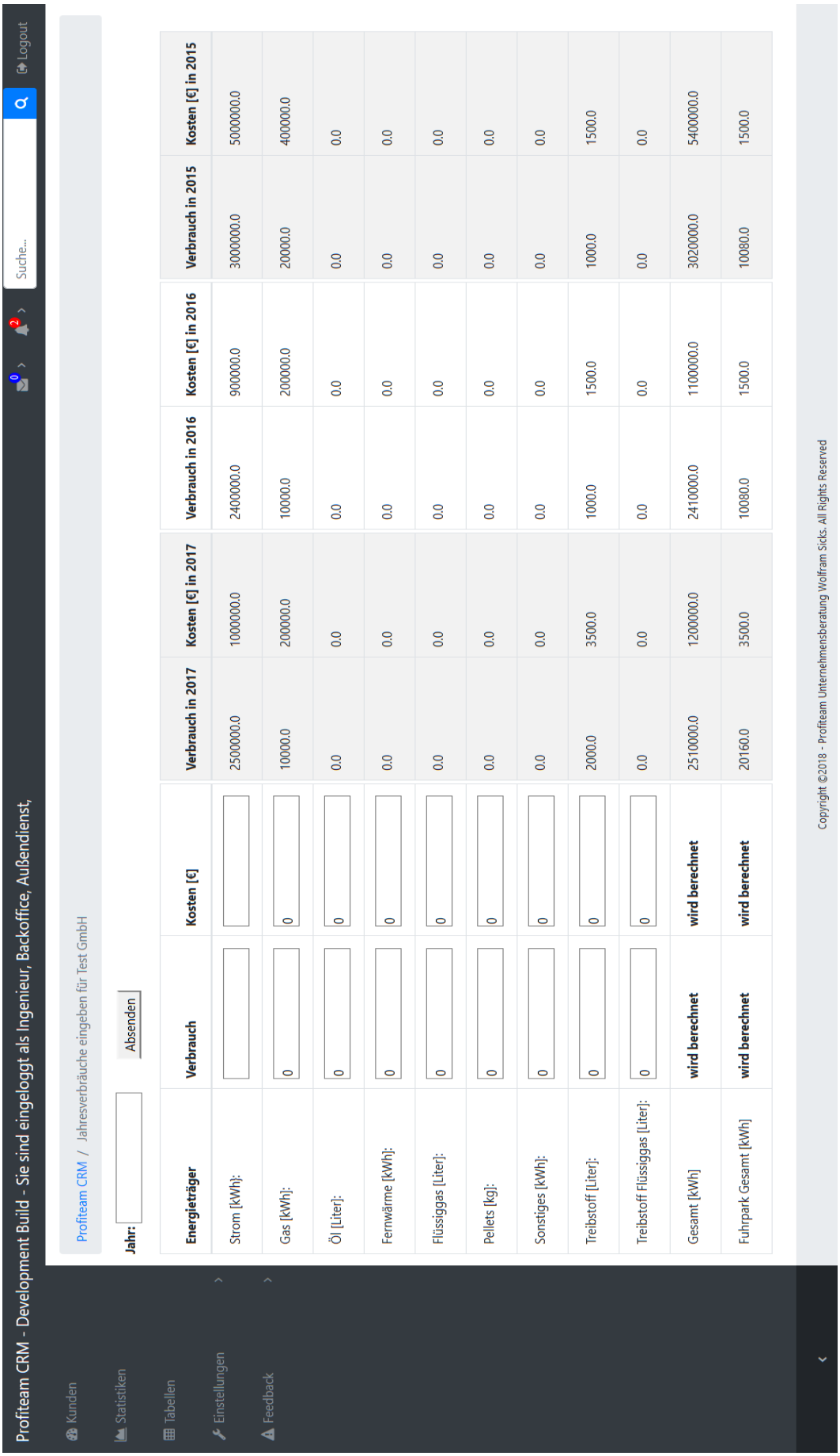

Abbildung B.1: gerendertes Template für die Eingabe der Jahresverbräuche (Beispieldaten) [Eigene Darstellung]

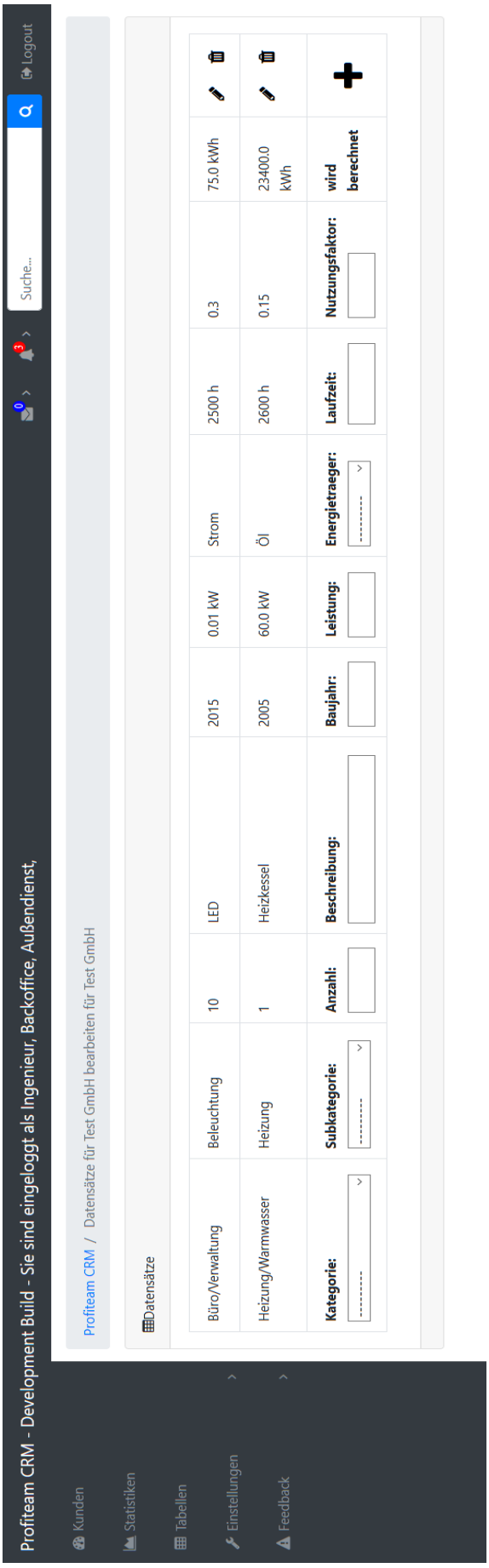

Abbildung B.2: gerendertes Template für die Eingabe der Verbraucher (Beispieldaten) [Eigene Darstellung]

B.1.2 Templates

```
{% extends "layout2.html" %}
{% block content %}
{% load staticfiles %}
<div class="card mb-3">
    <div class="card-header">
        <i class="fa fa-table"></i>Datensätze
    </div>
    <div class="card-body">
        <div class="table-responsive">
            <div id="EditDSModal" class="modal fade" tabindex="-1" role="dialog" aria-
                labelledby="EditDSModalLabel" aria-hidden="true">
                <div class="modal-dialog">
                    <div class="modal-content">
                        <div class="modal-header">
                            <h4 class="modal-title">Datensatz editieren</h4>
                            <button type="button" class="close" data-dismiss="modal"
                                aria-hidden="true">&times;  </button>
                        </div>
                        <div class="modal-body">
                            <p>placeholder</p>
                        </div>
                        <div class="modal-footer">
                            <button type="button" class="btn btn-default" data-dismiss="
                                modal">Abbrechen</button>
                            <button type="button" class="btn btn-primary">Änderungen
                                speichern</button>
                        </div>
                    </div>
                </div>
            </div>
            <table class="table table-bordered" id="dataTable" width="100%" cellspacing=
                "0">

                {% if kd_datasets|length >= 3 %}
                <thead>
                <tr>
                    <th>Kategorie</th>
                    <th>Unterkategorie</th>
                    <th>Anzahl</th>
                    <th>Beschreibung</th>
                    <th>Baujahr</th>
                    <th>Leistung</th>
                    <th>Energieträger</th>
                    <th>Laufzeit</th>
                    <th>Nutzungsfaktor</th>
                    <th>Verbrauch</th>
                    <th>Aktion</th>
                </tr>
                </thead>
                {% endif %}
                <tfoot>
                <tr>
                    <form id="form" action="" method="post">
                        {% csrf_token %}
                        {% for field in form %}
                        <th> {{ field.errors }}
                            {{ field.label_tag }} {{ field }}
                        </th>
                        {% endfor %}
                        <th>wird berechnet</th>
                        <th id="save_ds">
                            <div class="col-md-4"></div>
                            <div class="col-md-4">
                            <a class="tooltips" data-toggle="tooltip" data-placement="
                                top" title="Zeile speichern">
                                <button type="submit" name="send"
                                        onclick=""
                                        style="border: 0; background: none;">
                                    <i class="fa fa-plus"></i>
```

```
                                </button>
                            </a>
                            </div>
                            <div class="col-md-4"></div>
                        </th>
                    </form>
                </tr>
                </tfoot>
                <tbody>
                {% for dataset in kd_datasets %}
                <tr>
                    <td>{{ dataset.get_kategorie_display }}</td>
                    <td>{{ dataset.get_subkategorie_display }}</td>
                    <td>{{ dataset.anzahl }}</td>
                    <td>{{ dataset.beschreibung }}</td>
                    <td>{{ dataset.baujahr }}</td>
                    <td>{{ dataset.leistung }} kW</td>
                    <td>{{ dataset.get_energietraeger_display }}</td>
                    <td>{{ dataset.laufzeit }} h</td>
                    <td>{{ dataset.nutzungsfaktor }}</td>
                    <td>{{ dataset.verbrauch }} kWh</td>
                    <td>
                        <div id="action_container">
                        <div id="edit_ds">
                            <form action="" method="post">
                                {% csrf_token %}
                                {{ deletemessage }}
                                <input type="hidden" name="item_id" value="{{dataset.id
                                    }}"/>
                                <a class="tooltips" data-toggle="modal" data-target="#
                                    EditDSModal" data-placement="top" >
                                    <button type="submit" name="edit"
                                            onclick=""
                                            style="border: 0; background: none;">
                                        <i class="fa fa-pencil"></i>
                                    </button>
                                </a>
                            </form>
                        </div>
                        <div id="delete_ds">
                            <form action="" method="post">
                                {% csrf_token %}
                                {{ deletemessage }}
                                <input type="hidden" name="item_id" value="{{dataset.id
                                    }}"/>
                                <a class="tooltips" data-toggle="tooltip" data-placement
                                    ="top" title="Zeile löschen">
                                    <button type="submit" name="delete"
                                            onclick="return confirm('Sind sie sicher,
                                                dass Sie diesen Eintrag löschen möchten
                                                ?');"
                                            style="border: 0; background: none;">
                                        <i class="fa fa-trash"></i>
                                    </button>
                                </a>
                            </form>
                        </div>
                        </div>
                    </td>
                </tr>
                {% empty %}
                <p>noch keine Datensätze vorhanden vorhanden</p>
                {% endfor %}
                </tbody>
            </table>
        </div>
    </div>
    <div class="card-footer small text-muted"></div>
</div>
{% endblock %}
```

Listing B.1: Template zur Aufnahme der Verbraucher [Eigene Darstellung]

B.2 Datenverarbeitung

Tabelle B.1: Herkunft und Beschreibung der erhobenen Daten [Eigene Darstellung]

data-config.tex	**Django**	**Beschreibung**	**Erhebung**
\Energieverbrauchsklasse	ebmklasse.type	Verbrauchsklasse A, B oder C	Datenblatt-View
\maxWertKWHA	KlasseABorder - 1	entspricht 150000 kWh	Konstante
\minWertKWHB	KlasseABorder	entspricht 150001 kWh	Konstante
\maxWertKWHB	KlasseBBorder	entspricht 500.000 kWh	Konstante
\Pauschalkosten	ebmklasse.pauschalkosten		abhängig von ebmklasse
\Firma	kunde.get_firmenname()	Firmenname des Kunden	Stammdaten-View
\Str	adresse.strasse	Straße der Firma	Stammdaten-View
\Hnr	adresse.hausnummer	Hausnummer der Firma	Stammdaten-View
\zusatz	adresse.zusatz	optionaler Adresszusatz der Firma	Stammdaten-View
\Plz	adresse.plz	Postleitzahl der Firma	Stammdaten-View
\Ort	adresse.ort	Ort der Firma	Stammdaten-Model
\V	ing.first_name	Vorname des Sachverständigen	Userobjekt
\N	ing.last_name	Nachname des Sachverständigen	Userobjekt
\BeraterID	Bafanr.objects.all().get(user__exact=ing).bafanr	Bafanummer des Sachverständigen	Bafanr-Model
\BeraterStr	ingkontakt.strhnr	Straße und Hausnummer des Sachverständigen	Kontaktdaten-Model
\BeraterPlz	ingkontakt.plz	Postleitzahl des Sachverständigen	Kontaktdaten-Model
\BeraterOrt	ingkontakt.ort	Wohnort des Sachverständigen	Kontaktdaten-Model
\Nr	ingkontakt.telefon	Telefonnummer des Sachverständigen	Kontaktdaten-Model
\Mail	ingkontakt.email	Mail-Adresse des Sachverständigen	Kontaktdaten-Model

Kolophon

Dieses Dokument wurde mit der LaTeX-Vorlage für Abschlussarbeiten an der htw saar im Bereich Informatik/Mechatronik-Sensortechnik erstellt (Version 2.1). Die Vorlage wurde von Yves Hary und André Miede entwickelt (mit freundlicher Unterstützung von Thomas Kretschmer, Helmut G. Folz und Martina Lehser).
Daten: (F)10.95 – (B)426.79135pt – (H)688.5567pt