



Bachelorarbeit

**Entwicklung des Softwaretools
„Anforderungen und Testen“ für
das Anforderungs- und
Testmanagement im
Softwareentwicklungsumfeld**

Florian Brunotte

Abschlussarbeit

zur Erlangung des akademischen Grades

Bachelor of Science

Vorgelegt von	Florian Brunotte
am	01.02.2021
Referent	Prof. Dr. Mark Hastenteufel
Korreferent	Prof. Dr. Martin Damm

Schriftliche Versicherung laut Studien- und Prüfungsordnung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Mannheim, 01.02.2021

A handwritten signature in black ink, reading "Florian Brunotte". The signature is written in a cursive style with a horizontal line underneath it.

Florian Brunotte

Zusammenfassung

Das Ziel dieser Bachelorarbeit ist die Entwicklung eines leichtgewichtigen *Software-tools* für das *Anforderungs- und Testmanagement* im *Softwareentwicklungsumfeld*. Dafür gibt es zwar professionelle Tools, diese sind jedoch für Lehrzwecke zu komplex. Diese Lücke soll durch das Ergebnis dieser Bachelorarbeit gefüllt werden mit dem Tool „*Anforderungen und Testen*“. Eingesetzt werden kann diese Anwendung im Rahmen von kleinen Semesterprojekten, die dadurch unterstützt werden sollen, ohne vom Wesentlichen abzulenken.

Geschrieben wurde das Tool als *Webanwendung* in der Programmiersprache *Python* und mit dem *Webframework Django*. Ergänzend dazu wurde auch *HTML*, *CSS* und *JavaScript* verwendet. Durch einen *UI-Prototypen* konnte das generelle Aussehen der Anwendung getestet werden. Dadurch konnten bereits Rückmeldungen für die Implementierung erhalten werden.

Der Fokus liegt dabei auf der Erstellung von simplen und leicht verständlichen Benutzerschnittstellen, welche durch einen *Usabilitytest* auf ihre Benutzerfreundlichkeit überprüft werden. Daneben soll das *Softwaretool* auch über eine Datenbankanbindung verfügen, zum Beispiel an *PostgreSQL*. Aus diesem Grund spielt das Datenbankdesign ebenfalls eine wichtige Rolle in dieser Arbeit.

Inhaltsverzeichnis

1 Grundlagen	2
1.1 Anforderungs- und Testmanagement	2
1.2 Marktverfügbare Tools	3
1.3 Usability	9
2 Planung	11
2.1 Anforderungsanalyse	11
2.2 Entity-Relationship-Modell	17
2.3 User-Interface-Prototyp	19
3 Implementierung	23
3.1 Einrichtung	23
3.2 Admin-Seite	27
3.3 URLs	28
3.4 Modelle	28
3.5 Views	30
3.6 Templates und Filter	33
4 Features	35
5 Usability Test	38
6 Fazit	40

Kapitel 1

Grundlagen

Es soll ein leichtgewichtiges *Softwaretool* entwickelt werden, welches die Lehrveranstaltung unterstützt. Das *Tool* sollte sich auf die Kernfunktionalitäten beschränken, damit sich die Anwender auf das Wesentliche konzentrieren können. Andere *Softwaretools*, die auf dem Markt verfügbar sind, sind für den Einsatz in der Lehre nicht geeignet, da sie nicht nur zu komplex sind, sondern auch zu große Kosten mit sich bringen. Eine intuitive und simple Software, wie „*Anforderungen und Testen*“ soll hier Abhilfe schaffen.

Im Nachfolgenden wird das *Anforderungs- und Testmanagement* erklärt. Danach werden die professionellen Tools vorgestellt, an denen sich die Software „*Anforderungen und Testen*“ orientiert, bevor zuletzt das Thema *Usability* angesprochen wird.

1.1 Anforderungs- und Testmanagement

Im Rahmen des *Anforderungsmanagement* gibt es verschiedene Arten von Anforderungen, die bei der Entwicklung eines Systems erfasst werden sollten. Es gibt die *Produktbeziehungsweise Stakeholderanforderungen*, die aus der Kunden- oder Nutzersicht entstehen. Damit werden die zu erreichenden Ziele und der Zweck des Systems beschrieben, die die *Stakeholder* verfolgen [2].

Als *Stakeholder* werden Individuen, Gruppen oder Organisationen bezeichnet, die von einem Projekt beeinflusst werden, das Projekt beeinflussen oder durch das Ergebnis des Projekts beeinflusst werden [1]. Beispiele für *Stakeholder* sind die Kunden oder Geschäftspartner in einem Projekt. In diesem Projekt ist Herr Prof. Dr. Hastenteufel ein *Stakeholder*.

Dabei werden basierend auf den Anforderungen der *Stakeholder Software-Anforderungen* aufgestellt. Diese müssen von der Software erfüllt werden, damit der Kunde ein

System erhält, das seinen Vorstellungen entspricht [2]. Zur Überprüfung der Funktionalitäten der Software werden Tests geschrieben und durchgeführt. Die *Stakeholder* werden auf diesem Weg über die Qualität der Software informiert [3].

1.2 Marktverfügbare Tools

Es gibt verschiedene Tools mit diversen Gründen, die einen Ausschluss von der Benutzung in einer Lehrveranstaltung rechtfertigen. Sie können andere, eigene Begriffe verwenden, hohe Kosten nach sich ziehen oder zu komplex sein. Das Ziel der Lehrveranstaltung sollte das *Anforderungs- und Testmanagement* sein und nicht das Kennenlernen neuer Tools. Diese sollten die Lehrveranstaltung nur unterstützen und nicht dominieren. Das Tool sollte also möglichst selbsterklärend sein, sodass das Gelernte im Unterricht sofort angewendet werden kann. Zu den verschiedenen Tools, die es auf dem Markt gibt, werden an dieser Stelle die Aspekte präsentiert, die die Tools ungeeignet machen für den Einsatz in Semesterveranstaltungen. Daneben werden aber auch die Aspekte präsentiert, an denen sich die erstellte Software orientiert hat.

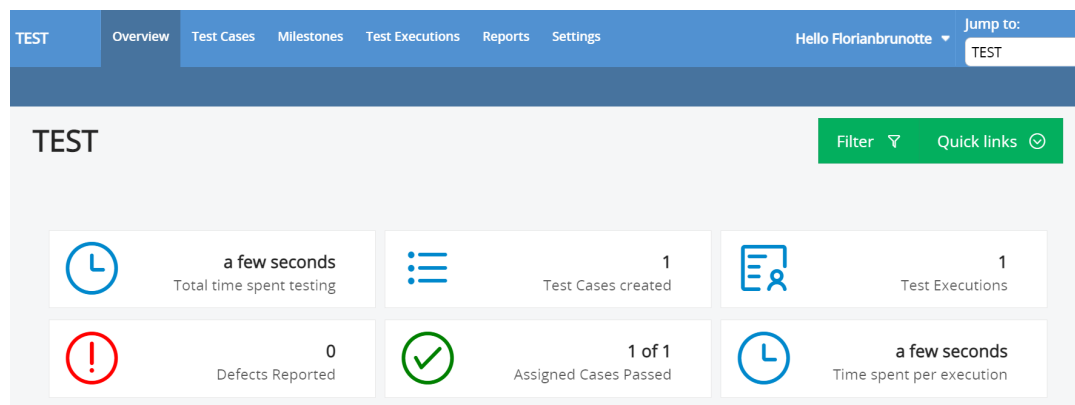


Abbildung 1.1: Das Dashboard in Testcollab, aus [14]

Das erste Tool, das an dieser Stelle vorgestellt werden soll, ist *Testcollab* [14]. Dieses Tool ist das einzige, das eine kostenlose Version anbietet. Bei der kostenlosen Variante können maximal drei Personen gleichzeitig an einem Projekt arbeiten, was im Semester eventuell zu einem Problem führen könnte, dass es somit zu viele Gruppen gibt [15]. Daneben gibt es noch die Limitierung, dass in einem Projekt nur maximal 200 *Testcases* und 300 *Testruns* erstellt werden können. Da es sich im Semester nur um kleine Projekte handeln wird, sollte diese Limitierung kein Problem darstellen.

Aus Sicht der Benutzerfreundlichkeit fällt bei der Eingabe der *Testschritte* auf, dass diese mit Leerzeilen dazwischen eingegeben werden sollen, wie in der Abbildung

Running Test Execution 24 January 2021

Progress:

0 %

00 : 01 : 16

Stop

testcase_test

Estimated time:

3 sec

ID:

1

Test Suite:

test_suit

Description:

N/A

Steps:

N/A

Expected Result:

N/A

Attachments:

N/A

Notes

Attachments

File

Datei auswählen

Keine ausgewählt

+ Add more attachments




Pass

Fail

Skip

Abbildung 1.2: Die Testrun Durchführung in Testcollab, aus [14]

Steps

H1 H2 H3 H4 H5 H6 | B |   

Enter steps like this: Here goes our first step. Press RETURN twice to leave a blank line between two consecutive steps.

Next Step goes here like this.

And finally our last step. Of course, you can add any numbers of steps.


 Use Wizard Icon in toolbar to insert reusable steps

Abbildung 1.3: Die Eingabe von Testschritten in Testcollab, aus [14]

1.3 präsentiert. Die Eingabe der Leerzeile könnte vergessen werden, dadurch wären statt vieler *Testschritte* nur ein großer *Testschritt* definiert. Sinnvoller wäre eine Implementierung, in der jeder *Testschritt* ein eigenes Feld hat, zum Beispiel bei der Implementierung als Tabelle. Das Design der *Testrunseite*, die in der Abbildung 1.2 zu sehen ist, könnte allerdings übernommen werden.

Ein Nachteil des Tools ist, dass es keine Aussage darüber macht bei welchem Schritt der *Testrun* durchgefallen ist. Es gibt nur eine Aussage darüber, ob die gesamte *Test Suite* erfolgreich war oder nicht.

Andererseits wurde aus dem *Dashboard*, das in der Abbildung 1.1 dargestellt ist, die Navigationsleiste am oberen Bildschirmrand übernommen, damit die Nutzer immer einen Überblick haben, an welcher Stelle sie sich gerade befinden. Gleichzeitig wurde die Begrüßung oben rechts auch übernommen, dadurch soll der erfolgreiche *Login* signalisiert werden. Ebenfalls gibt es in dem *Dashboard* ein Ausrufezeichen, wenn etwas fehlgeschlagen ist. Dadurch könnte die Aufmerksamkeit der Nutzer direkt auf die wichtigsten Probleme im Projekt gelenkt werden.

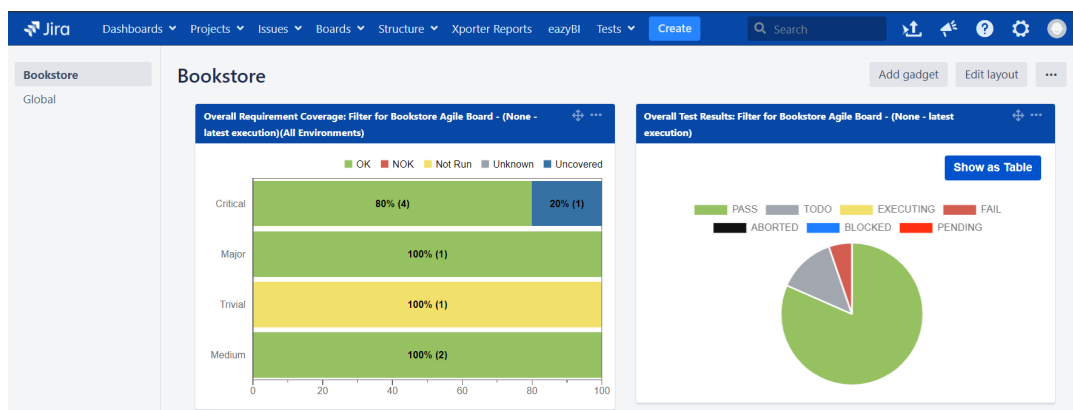


Abbildung 1.4: Das Dashboard in Jira, aus [34]

Das nächste Tool ist *Xray* [34]. Es ist eine Erweiterung des Tools *Jira*, welches für *agiles Projektmanagement* verwendet wird [24]. Auf den ersten Blick wirkt das Tool zu komplex, als das es eine Lehrveranstaltung unterstützend begleiten kann.

Die Visualisierung der Daten durch Kuchendiagramme auf dem *Dashbaord*, wie in der Abbildung 1.4 gesehen werden kann, könnte übernommen werden. Durch die Farben können die Nutzer sofort sehen, ob es ein Problem im Projekt gibt. Bei der Liste der *Requirements*, die in der Darstellung 1.5 abgebildet ist, kann der Status abgelesen werden. Auf diesem Weg können die Nutzer auch anhand der Farben erkennen, welche *Elemente* noch Aufmerksamkeit benötigen.

Beim Tool *Testlodge* muss vor dem Erstellen von *Requirements* ein *Requirement Document* erstellt werden und vor dem Erstellen von *Testcases* eine *Test Suite* [22]. Darin

Requirements List: Bookstore Requirements - (None - latest execution)(All Environments)			
Key	Summary	Status	
BOOK-40	Account Security	OK	
BOOK-33	[Demo] As a visitor, I can manage the bookstore Newsletter subscription	NOTRUN	
BOOK-13	As a visitor, I can navigate to the book details page	OK	
BOOK-12	As a visitor, I can navigate to the bookstore home page	UNCOVERED	
BOOK-8	As a visitor, I can manage my Shopping Basket	OK	
BOOK-7	As a visitor, I can search for books in the store	OK	
BOOK-6	As a visitor, I can checkout items in my basket	OK	
BOOK-5	As a visitor, I can change my locale	OK	
BOOK-1	As a visitor, I can manage my account	OK	
Showing 1 to 9 of 9 entries			First Previous 1 Next Last

Abbildung 1.5: Die Requirements Liste in Jira, aus [34]

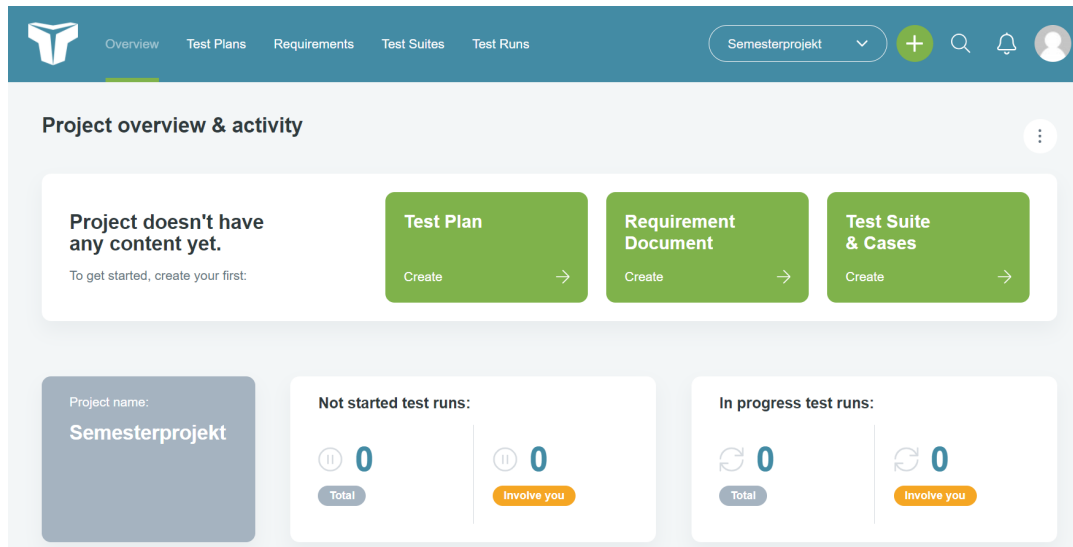


Abbildung 1.6: Das Dashboard von Testlodge, aus [22]

werden dann die jeweiligen Elemente gesammelt. Diese Option entfällt bei dem Tool „Anforderungen und Testen“. Es sollte sofort eine Möglichkeit geben neue *Elemente* zu erstellen, ohne über den Umweg einer weiteren Ebene gehen zu müssen.

Es sollte die Unterscheidung zwischen *Requirements*, *Testcases* und *Testruns* übernommen werden, damit die Nutzer besser unterscheiden können, was gerade bearbeitet wird. Diese Unterscheidung gab es nur in diesem Tool und kann in der Menüleiste in der Graphik 1.6 gesehen werden.

Testlodge bietet auch die Möglichkeiten an, *Testruns* an andere Leute im Projekt anzuweisen. Das ist ein weiteres Feature, das aufgegeben wird, um sich auf das Wesentliche zu begrenzen. Das Aussehen der Software „Anforderungen und Testen“ wird, aufgrund der Aufteilung der *Elemente*, am meisten von diesem Tool beeinflusst.

Ein weiteres Tool, welches für das *Anforderungs- und Testmanagement* entwickelt wurde, ist *Testcaselab* [16]. Inspirierend bei diesem Tool ist die Anzeige der *Elemente* als Liste auf der linken Seite mit der detaillierten Ansicht auf der rechten Seite. Diese Ansicht kann in der Abbildung 1.7 gesehen werden.

Das letzte Tool, das an dieser Stelle erwähnt werden soll, ist *QA Touch* [32]. Negativ auffallend bei diesem Tool ist die Navigation der Seite. Die Seite stellt Inhalte häufig mit Pop-up-Fenstern dar, wodurch es schwierig wird sich den jetzigen Standpunkt der Webseite klar zu machen, und das obwohl es eine Leiste gibt, die den Verlauf auf der Seite anzeigt. Diese Leiste kann in der oberen Bildhälfte der Abbildung 1.8 gesehen werden. Durch diese Unübersichtlichkeit ist dieses Tool für eine Benutzung in der Lehrveranstaltung eher ungeeignet.

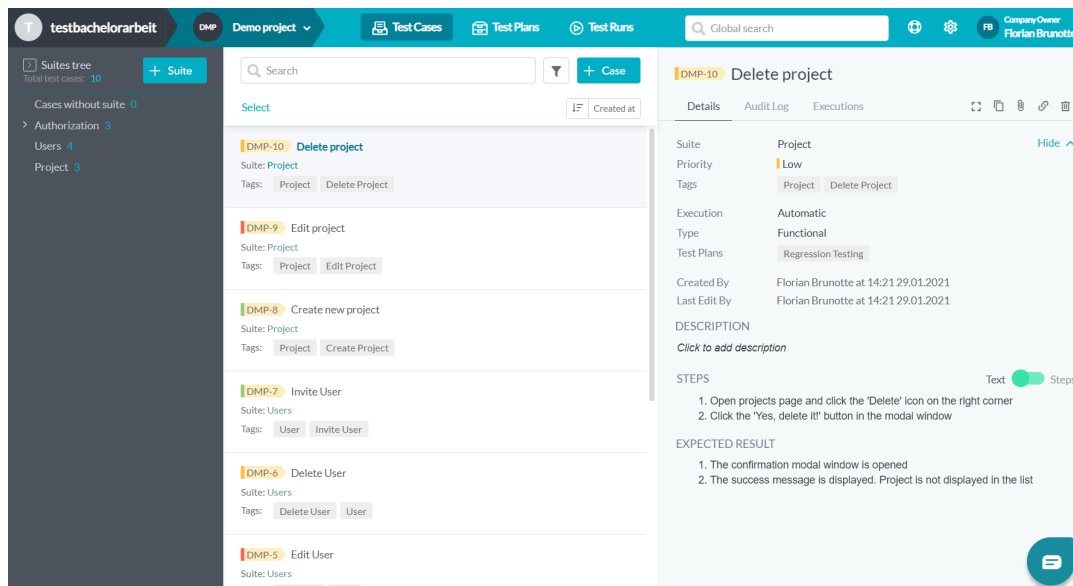


Abbildung 1.7: Das Dashboard von Testcaselab, aus [16]

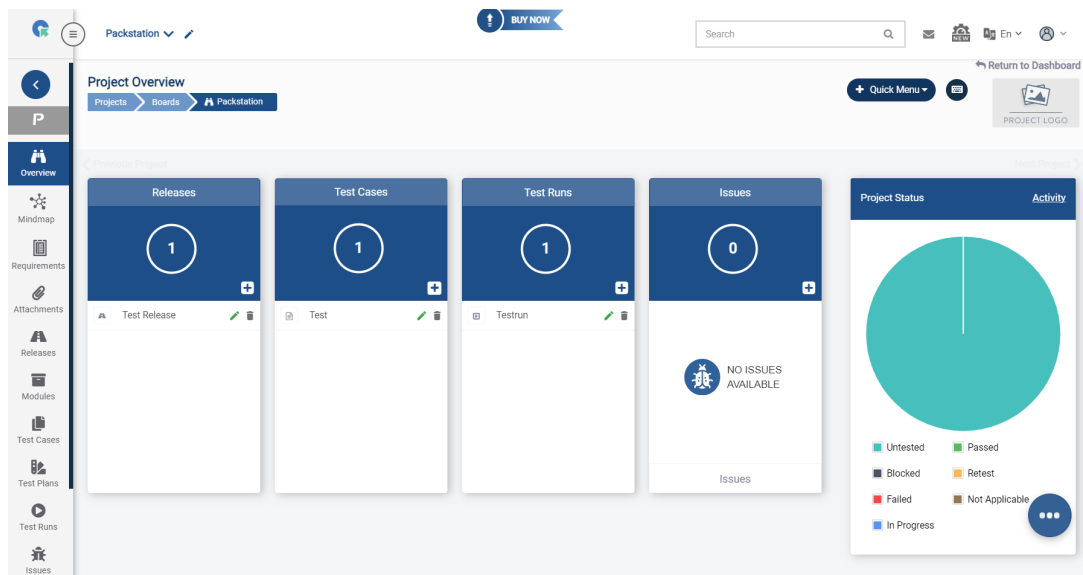


Abbildung 1.8: Das Dashboard von QA Touch, aus [32]

Tabelle 1.1: Zusammenfassung der Tools

Tool	Pro	Contra
Testcollab	kostenlose Variante wird angeboten	Eingabe der Testschritte nicht benutzerfreundlich und keine Aussage, welcher Testschritt fehlgeschlagen ist
Xray	Visualisierung macht Probleme sofort ersichtlich	hohe Komplexität und kostenlose Version ist nur als Free Trial verfügbar
Testlodge	Unterscheidet zwischen Requirements, Testcases und Testruns	kostenlose Version ist nur als Free Trial verfügbar
Testcaselab	Listenansicht der Elemente mit anschließender Detailansicht	kostenlose Version ist nur als Free Trial verfügbar
QA Touch	Verlauf der Seite wird angezeigt	ersichtliche Navigation der Seite wird durch Pop-up-Fenster eingeschränkt und kostenlose Version ist nur als Free Trial verfügbar

In der Tabelle 1.1 sind die vorgestellten Tools mit den Punkten dargestellt, die für und gegen sie sprechen. Ergänzend dazu soll an dieser Stelle noch gesagt werden, dass nur das Tool *Testcollab* eine kostenlose Variante anbietet, die auch zeitlich nicht begrenzt ist. Ein generelles Problem bei allen Tools ist die Tatsache, dass es bei keinem die Möglichkeit gab eine *Traceability-Matrix* zu erzeugen. Wie im späteren Kapitel 2.1 gezeigt, wurde so eine Matrix jedoch gefordert.

1.3 Usability

Die *Usability*, beziehungsweise die Gebrauchstauglichkeit, wird beschrieben als der Nutzungskontext, in dem ein Produkt durch einen Nutzer effektiv, effizient und zufriedenstellend genutzt werden kann. Der Nutzer möchte durch die Benutzung des Produkts ein Ziel erreichen [28]. Dahingegen wird ein *User Interface*, beziehungsweise eine Nutzungsschnittstelle, definiert als Bestandteile eines interaktiven Systems, die für den Nutzer nötig sind, um sein Ziel zu erreichen. Dazu stellt das *User Interface* Informationen und Steuerelemente zur Verfügung [27].

Eine Übersicht guter Praktiken beim Design von *User Interfaces* bieten dabei die *zehn Usability-Heuristiken nach Nielsen*. Diese sind in der Tabelle 1.2 zusammengetragen.

Tabelle 1.2: Die zehn Usability-Heuristiken nach Nielsen, aus [26] und [31]

Datei	Inhalt
1. Sichtbarkeit des Systemstatus	Die Nutzer sollten eine Rückmeldung bekommen zu ihren Interaktionen mit dem System. Darüber hinaus sollten sie auch wissen an welchem Punkt sie sich in einer Aktion befinden.
2. Übereinstimmung von System und realer Welt	Das System sollte Begriffe verwenden, die der Kunde aus der realen Welt erwarten würde.
3. Nutzerkontrolle und -freiheit	Falls Nutzer einen Fehler machen, sollten sie die Möglichkeit haben diesen Fehler zu korrigieren oder sogar den aktuellen Prozess abubrechen.
4. Konsistenz und Standards	Nutzer sollten nicht überlegen müssen, ob verschiedene in der Software verwendete Begriffe vielleicht doch das gleiche bedeuten. Stattdessen sollten die gleichen Begriffe für die gleichen Funktionen benutzt werden.
5. Fehlervermeidung	Es sollte vermieden werden, dass Nutzer fehlerhafte Eingaben tätigen können. Das System sollte generell Fehler vermeiden.
6. Erkennen anstatt Erinnern	Nutzer sollten die Software benutzen können ohne sich an Dialoge erinnern zu müssen.
7. Flexibilität und effiziente Nutzung	Abkürzungen sollten erfahrenen Nutzern eine effiziente Bedienung ermöglichen. Unerfahrene Nutzer sollten durch Anleitungen angelernt werden.
8. Ästhetisches und minimalistisches Design	Der Inhalt eines UI sollte auf das Wesentliche reduziert werden.
9. Hilfe bei Fehlern	Fehlermeldungen sollten das Problem konkret beschreiben und eine konstruktive Lösung vorschlagen. Außerdem sollten sie in leicht verständlicher Sprache geschrieben sein.
10. Hilfe und Dokumentation	Soweit möglich, sollten Nutzer ohne Hilfe die Software bedienen können. Falls nötig sollte sie aber bereitstehen.

Kapitel 2

Planung

Vor der Implementierung der Anwendung sollte geklärt sein, welche Anforderungen, wie realisiert werden sollen. Basierend auf den vorgegebenen Anforderungen wurden *UML-Diagramme* erstellt. Darüber hinaus wurden sowohl ein *Entity-Relationship Diagramm* erstellt, mit dem das Datenbankdesign beschrieben wird, als auch ein *UI-Prototyp* vorgestellt, mit dem die Navigation und das Aussehen der Seite getestet werden kann.

2.1 Anforderungsanalyse

Die Anforderungen an die Software wurden von Herrn Prof. Dr. Hastenteufel in Form eines Anforderungsdokuments erhoben. Darin enthalten sind die folgenden Nutzerrollen:

- Admin
- Professor
- Student beziehungsweise Entwickler und/oder Tester
- Nutzer (umfasst alle Rollen)

Die Anforderungen an die Software sind in drei Funktionsblöcke aufgeteilt, welche in den Tabellen 2.1, 2.2 und 2.3 betrachtet werden können. Der Klärungsbedarf, der bei den Inhalten des *Dashboards* vorhanden war, konnte durch den *UI-Prototypen*, der in Kapitel 2.3 vorgestellt wird, gelöst werden. Zusätzlich wurde die Terminologie vereinbart, dazu wurden die englischen Begriffe ausgewählt. Die Anforderungen werden in der Software *Requirements*, die Testfälle *Testcases* und die Testfalldurchführungen *Testruns* genannt. Alle drei Begriffe werden unter der Bezeichnung *Element*

Tabelle 2.1: Funktionsblock Administration

Administration	
A10	Als Nutzer will ich mich einloggen können und einen persönlichen Startbildschirm mit meinen Projekten sehen (Bemerkung: Im optimalen Fall SSO mit SWT / HS Account)
A20	Als Admin will ich neue Projekte erstellen können
A30	Als Admin will ich Studenten zu Projekten zuordnen können

Tabelle 2.2: Funktionsblock Statistiken

Statistiken	
A40	<p>Als Professor will ich Statistiken zu den einzelnen Projekten sehen. (Bemerkung: Statistiken sind genauer zu definieren) Das können z.B. sein</p> <ul style="list-style-type: none"> • Anzahl der Anforderungen pro Projekt • Anzahl der Testfälle pro Projekt • Anzahl der durchgeführten Testruns pro Projekt • Für jede Anforderung die zugehörigen Testfälle (Testfallabdeckung) • Für jede Anforderung den Status der zugehörigen Testruns (Testabdeckung) • Für jeden Testfall die damit abgedeckten Anforderungen • Für jeden Testfall den Status der zugehörigen Testruns • Angaben über die mit der Testfalldurchführung verbrachte Zeit (minimal, maximal, Durchschnitt, Summe) • Ein übersichtliches Dashboard des Projektstatus (Klarungsbedarf) • Für jedes Teammitglied die Anzahl der Anforderung/Testfälle/Testruns
A50	Als Student will ich für mein aktuelles Projekt Statistiken sehen

Tabelle 2.3: Funktionsblock Anforderungs- und Testmanagement

Anforderungs- und Testmanagement	
A60	<p>Als Student will ich Anforderungen spezifizieren können. Bemerkung: Anforderungen sollten mindestens folgende Attribute enthalten:</p> <ul style="list-style-type: none"> • Eine Anforderungs-ID (automatisch generiert) • Einen Namen • Die eigentlichen Anforderungen • Kommentar • Bearbeiter (automatisch) • Datum der initialen Erstellung und letzte Änderung • Die damit verbundenen Testfälle • Kategorie (Klarungsbedarf)
A70	<p>Als Student will ich Testfälle spezifizieren können. Bemerkung: Testfälle enthalten mindestens folgende Attribute:</p> <ul style="list-style-type: none"> • Eine Testfall-ID (automatisch generiert) • Einen Namen • Vorbedingungen • Den eigentlichen Testfall aufgegliedert in: <ul style="list-style-type: none"> – Durchzuführende Schritte – Erwartetes Ergebnis – Tatsächliches Ergebnis (wird bei der Testfalldurchführung ausgefüllt) • Kommentar • Bearbeiter (automatisch) • Datum der initialen Erstellung und letzte Änderung • Die mit dem Testfall getesteten Anforderungen
A80	<p>Als Student will ich Testfälle durchführen können (Testfalldurchführung/Testruns). Bemerkung: Testfalldurchführungen enthalten mindestens folgende Attribute:</p> <ul style="list-style-type: none"> • ID des zugrundeliegenden Testfalls • Datum des Testruns • Kommentarfeld für z.B. Dokumentation der Testumgebung • Für jeden Schritt des Testfalls: pass oder fail • Für jeden Schritt des Testfalls: Kommentar • Gesamtstatus der Testdurchführung: berechnet aus den einzelnen Testschritten • Name des durchführenden Testers • Potentiell Anhänge für die Testevidenz (png, pdf, ...) • Dauer der Durchführung in min

zusammengefasst. Bei der Kategorie der *Requirements*, aus der Anforderung A60, wurde der Klärungsbedarf dadurch gelöst, dass die Kategorien erstmal nur eine Auswahl aus den Werten eins, zwei oder drei darstellen soll. Zuletzt hat es sich angeboten die beiden Nutzerrollen Admin und Professor in der Implementierung zusammen zu führen, da sie beide administrative Aufgabe erfüllen.

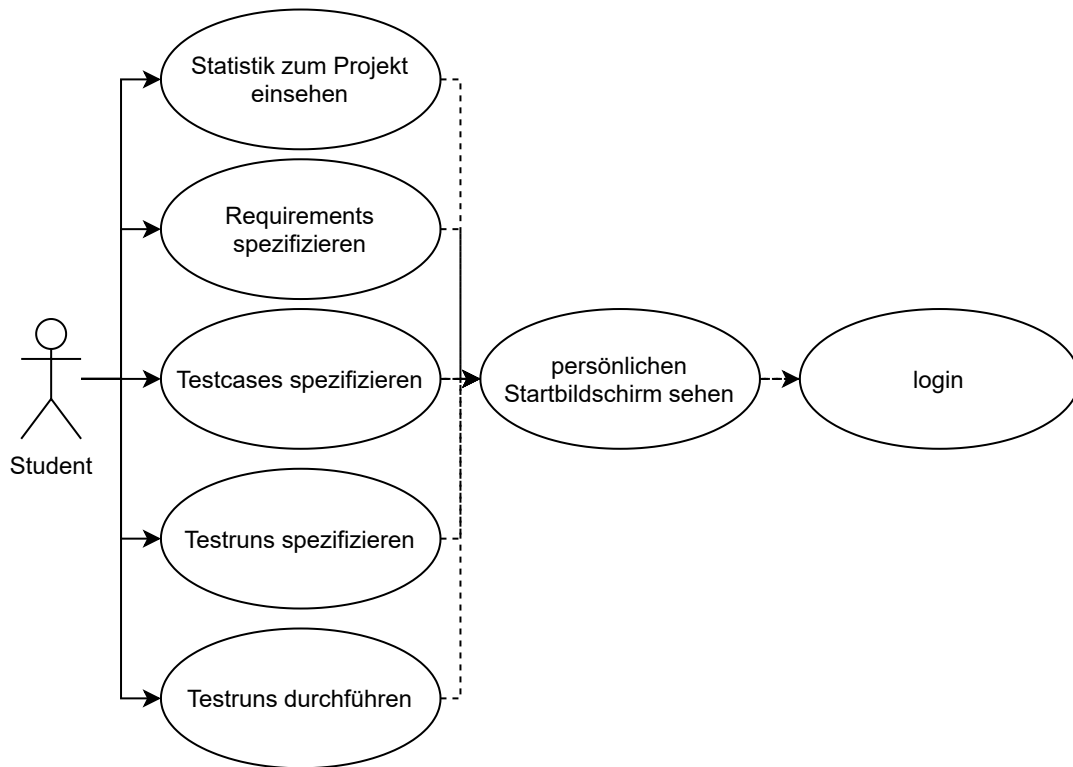


Abbildung 2.1: Die Use cases des Studenten, eigene Darstellung

Basierend auf den Anforderungen wurden sowohl *UML Use case Diagramme* als auch *UML Aktivitätsdiagramme* erstellt. Zuerst wurden die *Use cases* des Studenten in der Abbildung 2.1 gesammelt und die *Use cases* des Professors in Abbildung 2.2 zusammengetragen. Alle *Use cases* inkludieren dabei den *Use case* „persönlichen Startbildschirm einsehen“, welcher wiederum den *Use case* „login“ inkludiert.

Im nächsten Schritt wurden die *Aktivitätsdiagramme* erstellt, welche verdeutlichen sollen, wie sich die Software zu verhalten hat. Angefangen beim Login aus der Abbildung 2.3, soll dieser solange nach einem Benutzernamen und Passwort fragen, bis das Einloggen schließlich erfolgreich ist. Die Anforderung A30, in der nach der Möglichkeit neue Projekte hinzuzufügen gefordert wird, ist in der Abbildung 2.4 dargestellt. Hierbei kann ausgewählt werden, ob eine eigene, präferierte ID eingegeben werden soll. Die darauffolgende Anforderung A40, durch die es ermöglicht werden soll Studenten zu Projekten hinzuzufügen, ist in Abbildung 2.5 dargestellt. Es kann

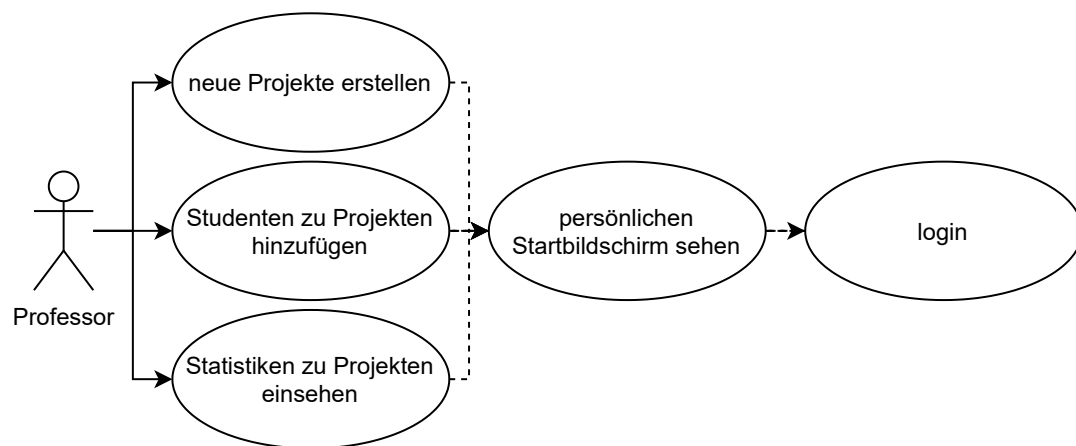


Abbildung 2.2: Die Use cases des Professors, eigene Darstellung

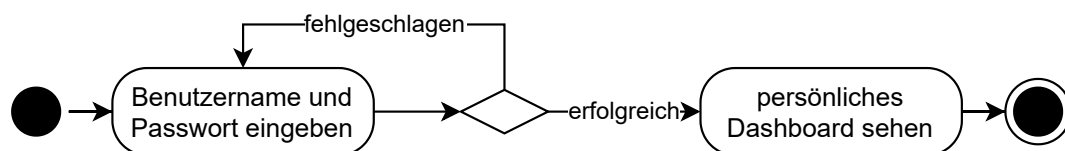


Abbildung 2.3: Login und persönlicher Startbildschirm, eigene Darstellung

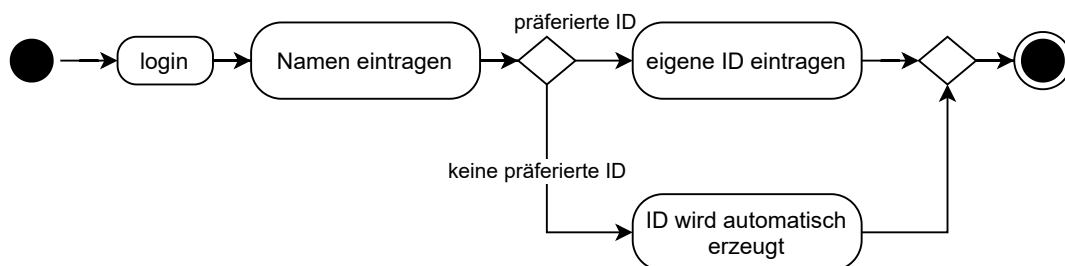


Abbildung 2.4: neue Projekte erstellen, eigene Darstellung

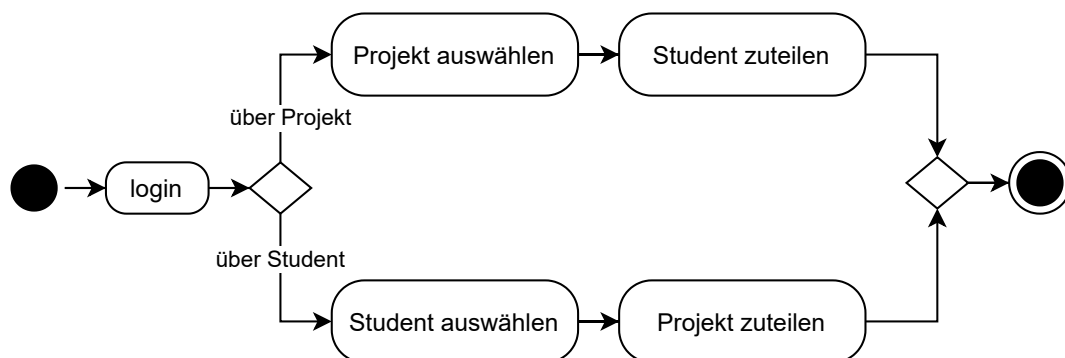


Abbildung 2.5: Studenten zu Projekten hinzufügen, eigene Darstellung



Abbildung 2.6: Statistiken zu Projekten einsehen als Professor, eigene Darstellung



Abbildung 2.7: Statistiken zu Projekten einsehen als Student, eigene Darstellung

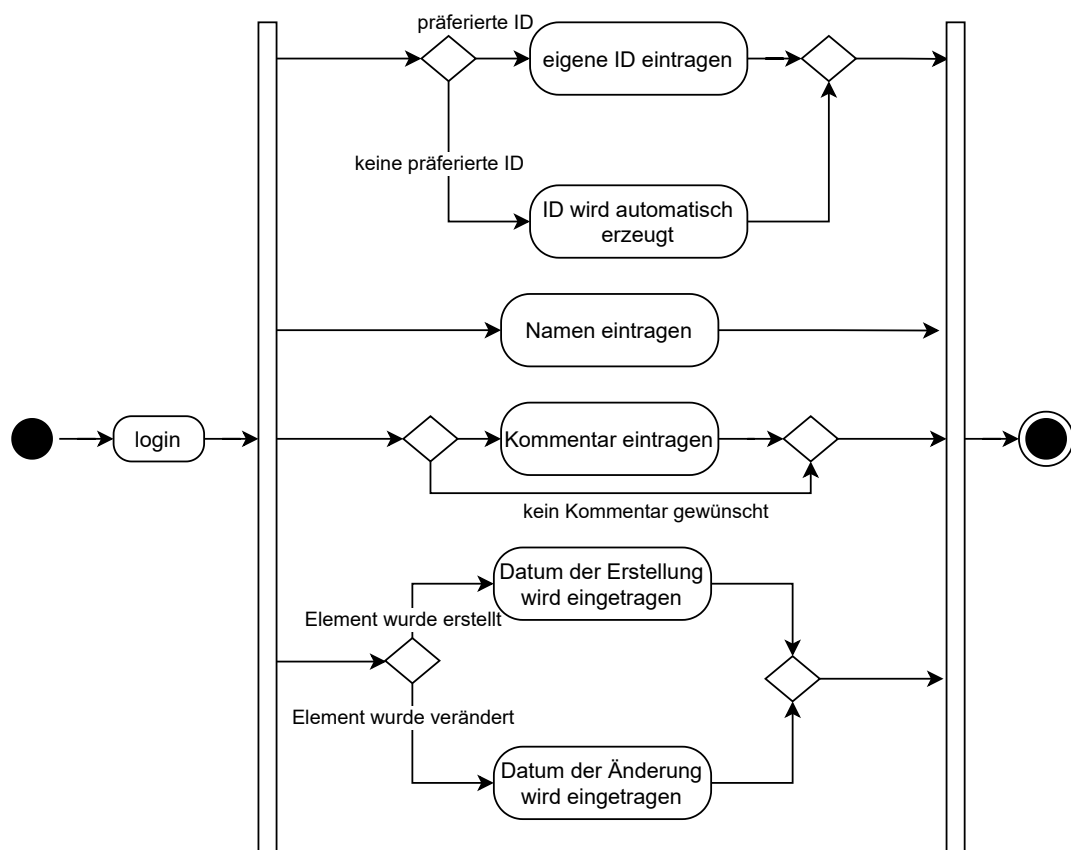


Abbildung 2.8: Ein Element erstellen oder ändern, eigene Darstellung

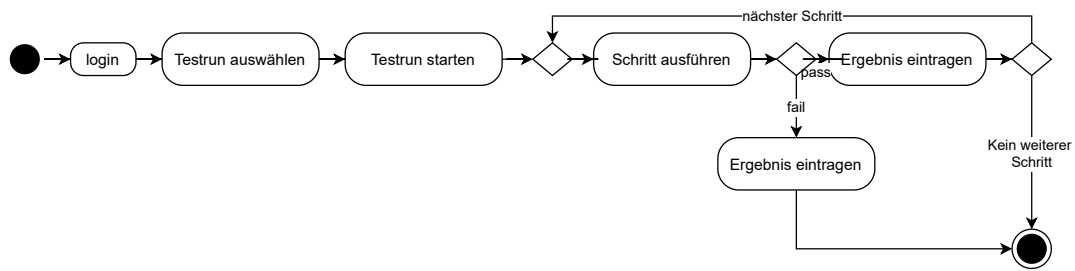


Abbildung 2.9: Einen Testrun durchführen, eigene Darstellung

entweder das Projekt ausgewählt werden, um einen Studenten hinzuzufügen oder es kann ein Student ausgewählt werden, um ihn einem Projekt hinzuzufügen.

Die Statistiken sollen für Studenten bereits nach dem Login sichtbar sein, wie in Abbildung 2.7 zu sehen ist. Anders sieht es da beim Professor aus, dieser muss vorher noch eine Gruppe auswählen, um dann die Statistik zu sehen. Dieser Vorgang ist in der Abbildung 2.6 dargestellt.

Die Anforderungen aus dem Funktionsblock Anforderungs- und Testmanagement wurden in der Abbildung 2.8 zusammengetragen. Hier soll gezeigt werden, dass die verschiedenen Attribute eines *Elements*, wie zum Beispiel der Name oder gegebenenfalls ein Kommentar eingetragen werden können. Hierbei sind nur die Attribute dargestellt, die jedes *Element* besitzt. Besondere Attribute, wie die Kategorie des *Requirement* wurden für diese Abbildung ausgelassen. Das letzte *Aktivitätsdiagramm*, die Abbildung 2.9, zeigt den Ablauf bei der Durchführung eines *Testruns*. Nach dem Login, dem Auswählen und Starten des *Testruns* werden solange *Testschritte* durchgeführt, bis es entweder keine *Schritte* mehr gibt oder bis ein *Schritt* nicht erfolgreich war. Wenn auch nur ein *Schritt* nicht erfolgreich ist, so gilt der ganze *Testrun* als durchgefallen.

2.2 Entity-Relationship-Modell

In der Graphik 2.10 kann das *Entity-Relationship-Modell* gesehen werden, das zur Grundlage des Datenbankdesigns dient. Im folgenden wird dieses Modell erklärt. Die *Primary-Keys* sind mit unterstrichenen Attributen dargestellt, während *Foreign-Keys* gestrichelte Unterstriche besitzen.

Damit sich ein Professor anmelden kann, braucht er eine Identifikation, wie eine Professorennummer, die ähnlich sein soll zu einer Matrikelnummer, sowie ein Passwort. Bei erfolgreicher Anmeldung wird sein Name angezeigt. Das Projekt wird von einem Professor erstellt, wobei ein Professor mehrere Projekte erstellen kann. Projekte haben eine ID und einen Namen, zum Beispiel „Packstation“.

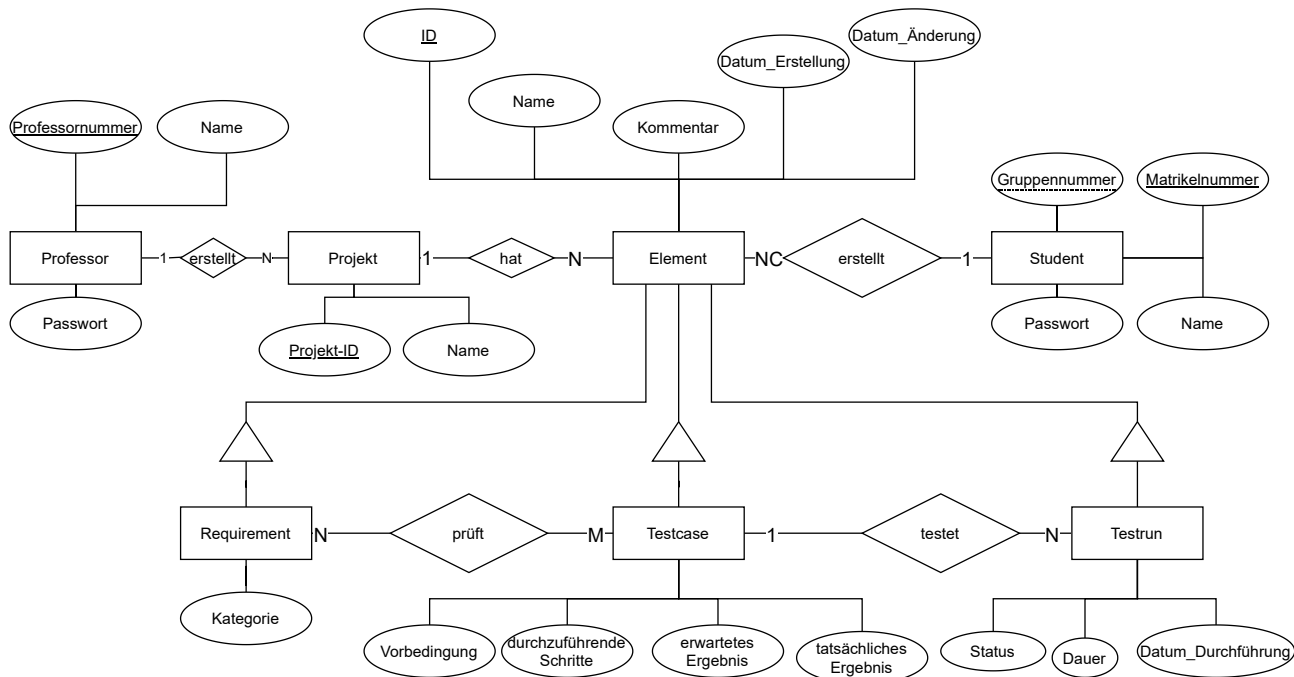


Abbildung 2.10: Das Entity-Relationship-Modell, eigene Darstellung

Ein Projekt kann mehrere *Elemente* haben, aber ein *Element* kann immer nur zu einem Projekt gehören. Die *Elemente* haben eine ID, einen Namen, einen Kommentar, ein Datum der Erstellung und ein Datum der letzten Änderung. Ein *Requirement* ist ein *Element* und kann einer Kategorie zugeordnet werden. Ein *Testcase* besitzt weitergehend die notwendigen Vorbedingungen für die durchzuführenden Schritte, das erwartete Ergebnis und das tatsächliche Ergebnis. Mehrere *Testcases* können mehrere *Requirements* abdecken. Ein *Testrun* hat einen Status, der aussagt, ob der *Testrun* erfolgreich ist oder nicht. Das ist realisierbar durch eine Auswahl zwischen den Möglichkeiten „pass“ und „fail“. Daneben gibt ein *Testrun* die Dauer an, die für ihn gebraucht wurde und das Datum an dem er durchgeführt wurde. Ein *Testrun* testet immer nur einen *Testcase*, aber ein *Testcase* kann von mehreren *Testruns* getestet werden. Ein Beispiel dazu ist, wenn der erste *Testrun* nicht erfolgreich war und ein zweiter Versuch nötig ist. Der *Testruns* kann nach der Durchführung nicht mehr geändert werden.

Ein Student braucht zur Anmeldung eine Matrikelnummer und ein Passwort, bei erfolgreicher Anmeldung wird sein Name angezeigt. Durch die Gruppennummer können Studenten in Gruppen auf die Projekte aufgeteilt werden. Die Gruppennummer soll dabei ein *Foreign-Key* auf die Projekt-ID sein. Ein Student kann kein, ein oder mehrere *Elemente* erstellen, aber ein *Element* wird immer nur einem Student zugeordnet.

2.3 User-Interface-Prototyp

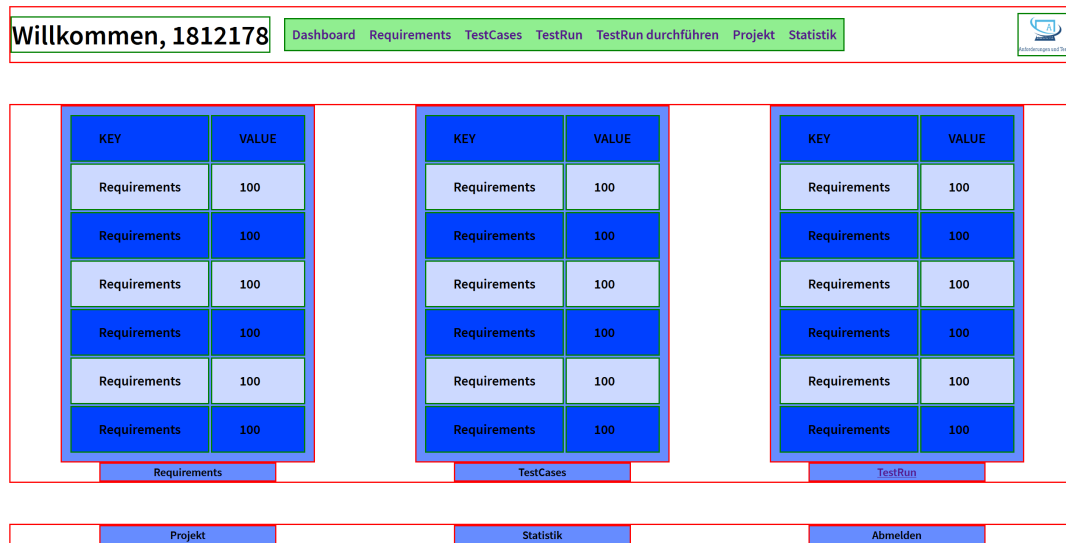


Abbildung 2.11: Das Dashboard, eigene Darstellung

Statistiken					
Dashboard	Requirements	TestCases	TestRuns	TestRun durchführen	Projekt
Statistiken					
<input type="radio"/> TestCase <input checked="" type="radio"/> TestRun <input type="radio"/> Team					TestCase Coverage: Req mit TestCase / Req
X	TestCase 1	TestCase 2	TestCase 3	TestCase 4	
Req 1	X				
Req 2		X			
Req 3					
Req 4		X			

Abbildung 2.12: Die Statistik Seite, eigene Darstellung

Der Prototyp, um das *User Interface* zu testen, wurde mit der kostenlosen Testversion der Software *Axure RP 9* erstellt [25]. Mit dem entwickelten Prototypen konnte durch eine Demo ein erster Test zur *Usability* gemacht werden. Der Prototyp basiert auf UI Skizzen, die auf der einen Seite selbst erstellt wurden und auf den UI Skizzen, die Herr Prof. Dr. Hastenteufel für das Projekt erstellt hat.

Auf den Abbildungen des Prototyps können bereits die grundlegenden Ideen zum Aufbau der Seite gesehen werden. Es soll eine Menüleiste am oberen Bildschirmrand


TestCase			Dashboard	Requirements	TestCases	TestRuns	TestRun durchführen	Projekt	Statistiken	
Neuer TestCase	Suche...	Auswählen	Speichern und TestRun erstellen		Speichern		Löschen			
TestCase 1 Name ID Kategorie TestCase 2 Name ID Kategorie TestCase 3 Name ID Kategorie			Name							
			ID							
			Vorbedingung							
			durchzuführende Schritte							
			erwartetes Ergebnis							
			tatsächliches Ergebnis							
			Ersteller							
			Datum_Erstellung							
			Datum_Änderung							
			Requirements		Requirement 1 Name ID					
TestRuns		TestRun 1 Name ID								

Abbildung 2.13: Die Erstellung eines Testcases, eigene Darstellung


TestRun durchführen		Dashboard	Requirements	TestCases	TestRuns	TestRun durchführen	Projekt	Statistiken	
TestRun 1 Name ID Kategorie TestRun 2 Name ID Kategorie TestRun 3 Name ID Kategorie		Überarbeiten							
		Starten							
		Vorbedingung: Die Packstation muss eingeschaltet sein							
		Schritt-nummer	Beschreibung	Erwartetes Ergebnis	Tatsächliches Ergebnis	Pass	Fail		
		1	Der Nutzer scannt seine Kundenkarte			<input type="radio"/>	<input type="radio"/>		
		2							
3									

Abbildung 2.14: Die Durchführung eines Testruns, eigene Darstellung

geben, dabei soll die aktuelle Seite markiert sein. Für die *Requirements*, *Testcases* und die *Testruns* soll es auf dem *Dashboard* je eine Tabelle geben, wie in Abbildung 2.11 gezeigt. Die angezeigten Daten sollen der Anforderung A40 entsprechen, beispielsweise wird die Anzahl der Anforderung pro Projekt visualisiert.

In der Anforderung A40 wurde auch eine Darstellung der verknüpften *Elemente* gefordert, also die *Testfallabdeckung* und die *Testabdeckung* der *Requirements*. Nach den UI-Skizzen, die von Herrn Prof. Dr. Hastenteufel angefertigt wurden, sollte diese Anforderung durch eine *Traceability-Matrix* realisiert werden.

Eine *Traceability-Matrix* bietet die Möglichkeit die Verknüpfung zwischen den *Requirements* und den *Testcases* zu verfolgen. Sie wird darüber hinaus noch ergänzt durch die Information, welchen Status der *Testrun* aufweist [17].

In der Abbildung 2.12 kann die *Traceability-Matrix* gesehen werden. Sie ist aufgeteilt in zwei Ansichten. Die erste Ansicht zeigt nur die *Requirements* und die *Testcases*. Falls diese zusammengehören, gibt es in dem entsprechenden Feld ein X als Markierung dafür. In der zweiten Ansicht wird das X, das die Verknüpfung anzeigt mit den entsprechenden Farben markiert. Rot markierte Verknüpfungen zeigen an, dass der letzte *Testrun* nicht erfolgreich war. Eine grüne Markierung signalisiert, dass der letzte *Testrun* erfolgreich war. Daneben gibt es noch eine blaue Markierung, falls zu dem *Testcase* noch keine *Testruns* durchgeführt wurden.

In der nächsten Graphik 2.13 kann die Bearbeitung eines *Elements* gesehen werden, hier wird beispielsweise die Bearbeitung eines *Testcases* gezeigt. Die Bearbeitungsseite für die *Requirements*, *Testcases* und *Testruns* sollte für alle *Elemente* gleich aussehen. Es gibt aber pro Seite andere Attribute zum Eintragen, zum Beispiel kann beim *Testcase* eine Vorbedingung eingetragen werden.

Zuletzt sei hier noch die Durchführung eines *Testruns* in der Abbildung 2.14 gezeigt. Nachdem ein *Testrun* aus der Liste an der linken Seite ausgewählt wurde, kann dieser ausgeführt werden. Dabei sind die Schritte in einer Tabelle angeordnet und zu jedem Schritt kann das tatsächliche Ergebnis und der Status eingetragen werden. Nur wenn alle Schritte erfolgreich durchgeführt werden konnten, wird der *Testrun* insgesamt als erfolgreich angesehen. Wenn auch nur ein Schritt durchgefallen ist, dann ist auch der komplette *Testrun* durchgefallen.

Es gab bei der Software *Axure RP 9* die Option aus dem erstellten Prototypen die entsprechenden *HTML-Seiten* zu generieren. Die auf diesem Weg generierten Seiten bestanden aber zu großen Teilen aus durchnummerierten *div-Elementen*. Um eigene Anpassungen, beispielsweise durch *CSS* zu machen, waren diese Seiten leider nicht brauchbar. Ein Beispiel dazu ist in dem Listing 2.1 dargestellt. In Zeile 26 kann das Wort „Statistiken“ gesehen werden, das in der Statistik Seite oben links neben der Menüleiste angezeigt wird.


```
1  <body>
2    <div id="base" class="">
3
4      <!-- Unnamed (Rectangle) -->
5      <div id="u256" class="ax_default box_1">
6        <div id="u256_div" class=""></div>
7        <div id="u256_text" class="text "
8          style="display:none; visibility: hidden">
9          <p></p>
10        </div>
11      </div>
12
13      <!-- Unnamed (Rectangle) -->
14      <div id="u257" class="ax_default box_1">
15        <div id="u257_div" class=""></div>
16        <div id="u257_text" class="text "
17          style="display:none; visibility: hidden">
18          <p></p>
19        </div>
20      </div>
21
22      <!-- Unnamed (Rectangle) -->
23      <div id="u258" class="ax_default heading_1">
24        <div id="u258_div" class=""></div>
25        <div id="u258_text" class="text ">
26          <p><span>Statistiken</span></p>
27        </div>
28      </div>
```

Listing 2.1: generierter HTML-Code der Statistik-Seite

Kapitel 3

Implementierung

Django wurde von einem Team, dass sich mit der Verwaltung von Zeitungsartikeln auf einer Webseite beschäftigt, dazu entwickelt, häufig auftretende Webentwicklungsaufgaben schnell und einfach realisieren zu können. Auf die Idee kam das Team, als sich bei einer Vielzahl an Webseiten ein gleiches Design mit gleich aufgebaute Code sichtbar machte [4]. Als *Web-Framework* unterstützt *Django* die Entwicklung von Webseiten, indem Dienste, wie die Datenbankbindung, zur Verfügung gestellt werden [6]. Ausgewählte Aspekte der Softwareentwicklung mit *Django* sowie Ausschnitte des Codes sind an dieser Stelle erklärt.

3.1 Einrichtung

In der Tabelle 3.1 sind die Materialien aufgelistet, die zur Erstellung des Tools verwendet wurden. Da *Django* in *Python* geschrieben ist, ist diese Programmiersprache die Grundvoraussetzung zum Arbeiten mit dem *Web-Framework* [23].

Zuerst muss in *PyCharm* ein neues Projekt angelegt werden. Der Vorteil bei der Nutzung von *PyCharm* als Entwicklungsumgebung ist, dass dabei auch direkt ein neues *virtual environment* erstellt wird. Dadurch werden die Erweiterungen, die mit

Element	Version
integrierte Entwicklungsumgebung/IDE	PyCharm Community 2020.2
Datenbank	PostgreSQL mit pgadmin 4.20
Programmiersprache	Python 3.7
Web-Framework	Django 3.1.2

Tabelle 3.1: Die benutzten Materialien für die Entwicklung der Software

Datei/Ordner	Inhalt
/anforderungen_und_testen	Der äußere Ordner ist ein Container für das Projekt, der Name ist nicht wichtig und kann beliebig umbenannt werden
manage.py	In dieser Datei sind die Kommandozeilenbefehle enthalten, wie zum Beispiel <code>runserver</code>
anforderungenundtesten/	der innere Ordner ist das eigentliche Python Package
settings.py	Einstellungen die für das komplette Django Projekt gelten sind hier enthalten
urls.py	Die URLs für das gesamte Projekt sind in dieser Datei enthalten
asgi.py	Für ASGI kompatible Webserver
wsgi.py	Für WSGI kompatible Webserver

Tabelle 3.2: Die erstellten Inhalte nachdem ein neues Projekt erstellt wurde

`pip` installiert werden, nur für das aktuelle *virtual environment* installiert und nicht global [11]. Mit dem Befehl `pip install django` kann *Django* installiert werden [12].

Der Befehl `django-admin startproject anforderungen_und_testen` erstellt ein neues *Django-Projekt* und der dazu notwendige Code wird generiert. Dieser kann in der Tabelle 3.2 gesehen werden [12]. Ergänzend dazu ist in der Abbildung 3.1 die komplette Projektstruktur abgebildet.

Als nächstes wird der Befehl `cd ./anforderungenundtesten` benutzt, um in den richtigen Ordner zu wechseln, welcher die `manage.py` Datei enthält. Über den Befehl `python manage.py runserver` kann der integrierte *Development Server* von *Django* gestartet werden. Falls die Abbildung 3.2 zu sehen ist, ist die Installation erfolgreich [12].

Über den Befehl `python manage.py startapp aut` wird eine neue *App* angelegt wodurch weitere Dateien generiert werden. Diese sind in der Tabelle 3.3 dargestellt und erklärt [12]. Es gibt einen Unterschied zwischen einem Projekt und einer *App* in *Django*. Eine *App* ist eine *Webanwendung*, die etwas machen kann. Während ein Projekt eine Ansammlung von Einstellungen und *Apps* ist. Ein Projekt kann mehrere *Apps* enthalten und eine *App* kann in verschiedenen Projekten enthalten sein [12].

Neben den automatisch erstellen Dateien und Ordnern mussten noch weitere Elemente erstellt werden, zum Beispiel die `urls.py` Datei in der *App*. Dadurch kann das Projekt auf die *URLs* der *App* zugreifen. Im Ordner `static` befinden sich die auf der Webseite verwendeten Bilder, während sich die *HTML-Dateien* in dem Ordner `templates` befinden. Die Datei `choices.py` beinhaltet die für die in `forms.py`

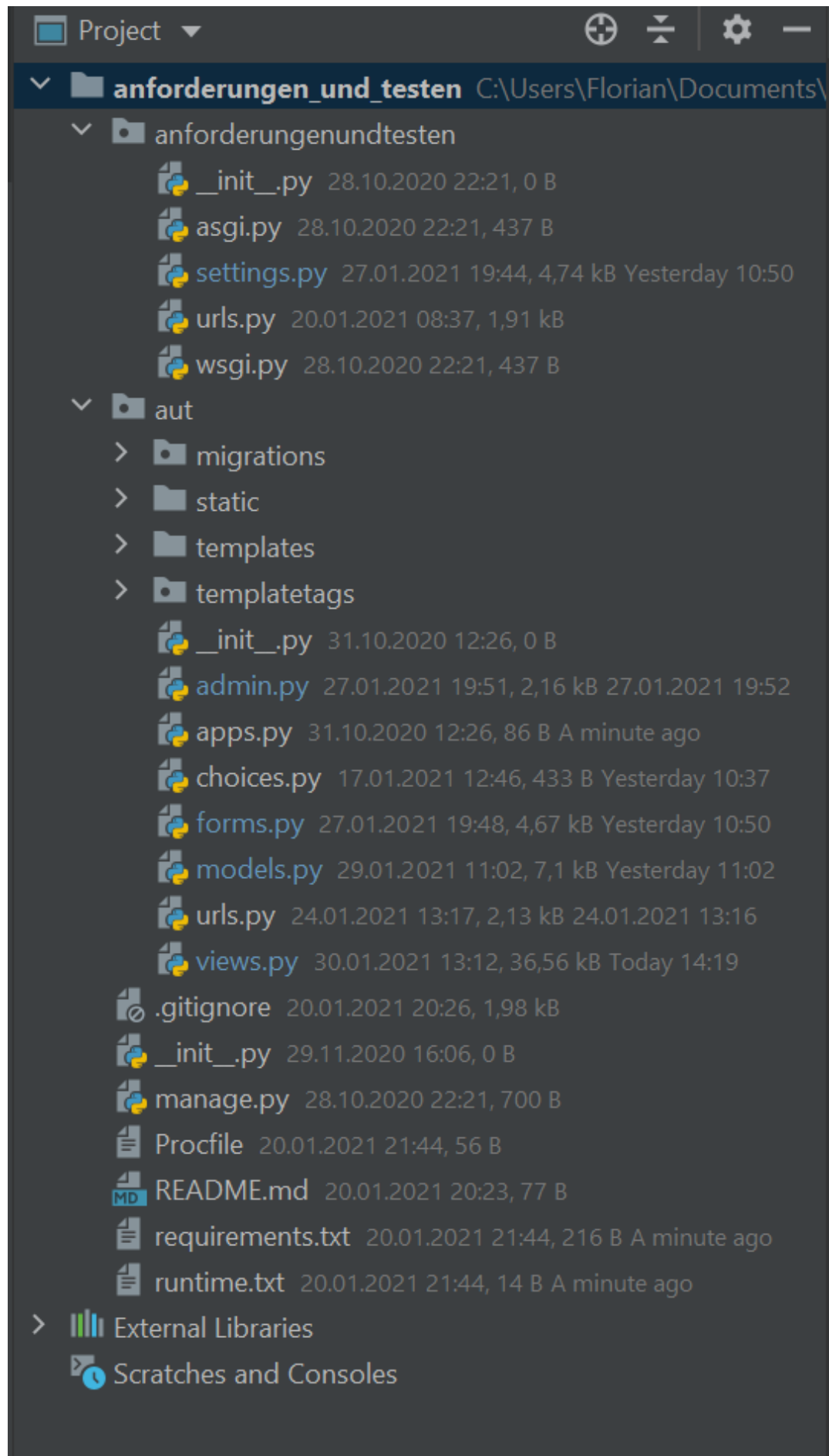


Abbildung 3.1: Die komplette Struktur der Anwendung, eigene Darstellung

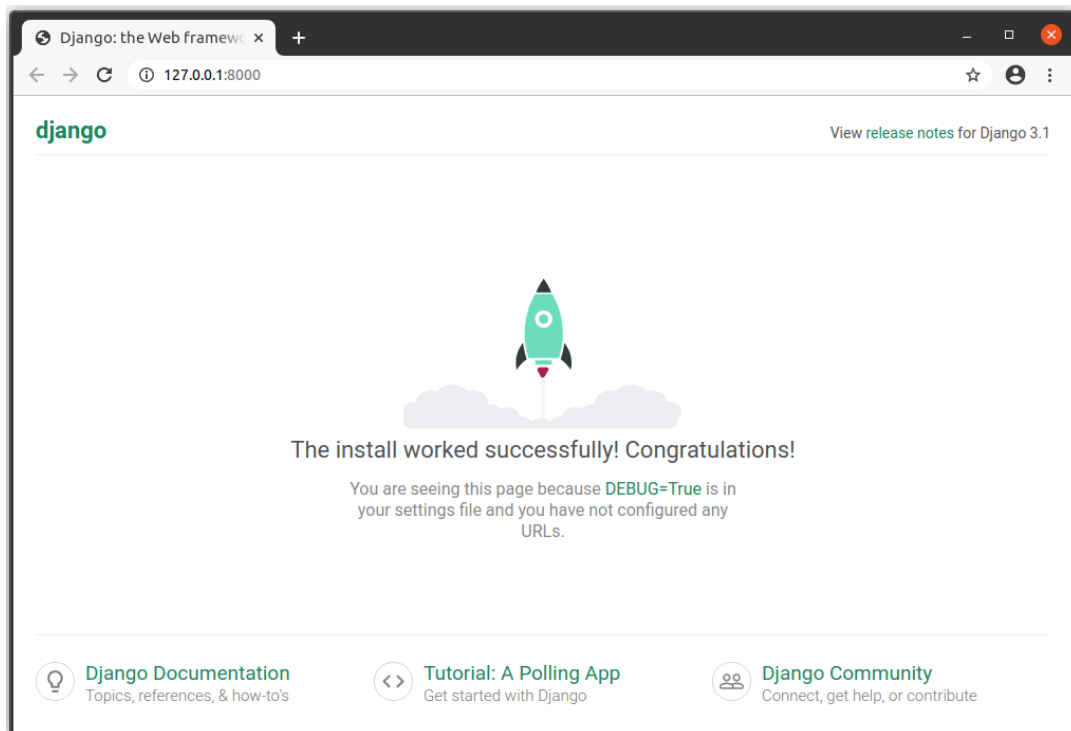


Abbildung 3.2: Die Installation war erfolgreich, Abbildung aus [11]

Datei/Ordner	Inhalt
/aut	Der Ordner beinhaltet die App
__init__.py	Durch diese leere Datei wird der Ordner als Python Package registriert
admin.py	In dieser Datei können die erstellten Modelle für den Admin registriert werden zum Bearbeiten
apps.py	Die Datei enthält den Namen der App
migrations/	Der Ordner beinhaltet die Datenbankmigrationen
models.py	Hier werden die Modelle definiert, die die Daten der Anwendung realisieren sollen
views.py	die hier erstellten Funktionen werden über die URLs aufgerufen und befüllen die Templates mit Daten

Tabelle 3.3: Die erstellten Dateien und Ordner nachdem eine App erstellt wurde

definierten *Forms* benötigten Auswahlentscheidungen für die Nutzer. Zuletzt ist in dem Ordner `templatetags` eine Datei enthalten in der sich die selbst geschriebenen *Filter* befinden, welche in Kapitel 3.6 erklärt sind. Wichtige Einstellungen in der `settings.py` Datei beinhalten das Registrieren der *App*, die Auswahl der Datenbank und das Einstellen der Sprache sowie der Zeitzone [7].

3.2 Admin-Seite

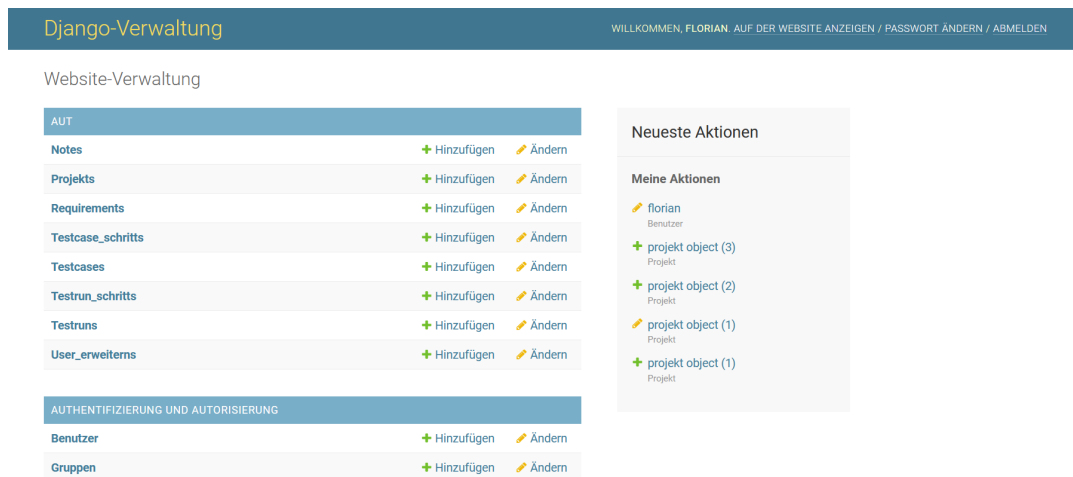


Abbildung 3.3: Die Admin-Seite, eigene Darstellung

Die Administrationsseite kann bei Bedarf automatisch von *Django* erstellt werden. Mit dieser Seite können für die registrierten Modelle aus der `admin.py` Datei Elemente hinzugefügt, gelöscht und geändert werden. Eine andere wichtige Funktion ist die Verwaltung der User der Webseite. Die Admin-Seite ist in der Abbildung 3.3 zu sehen. Für die Modelle können Berechtigungen hinzugefügt oder entfernt werden. Standardmäßig können alle angelegten Nutzer alle Modelle bearbeiten. Die Seite erfüllt den Zweck, dass kein eigenes *Backend-Interface* erstellt werden muss, um Daten und Inhalte verwalten zu können. Außerdem ist es gedacht, diese Seite auch vom Kunden oder den Mitarbeitern im realen Umfeld benutzen zu lassen [13]. Der Befehl `createsuperuser` erstellt einen Admin. Dazu muss noch ein Namen und ein Passwort eingegeben werden. Eine E-Mail ist optional einzugeben [9]. Diese Seite wird auch in der Anwendung „Anforderungen und Testen“ verwendet, da so die Nutzerverwaltung bereits realisiert werden konnte. Die Anwendungsfälle für die Admin-Seite sind:

- Ein Nutzer hat sein Passwort vergessen und er braucht ein Neues.

- Der Admin möchte eine neue Gruppe hinzufügen, wie in der Anforderung A20 gefordert.
- Ein Nutzer möchte in eine andere Gruppe eingeteilt werden. Dieser Anwendungsfall basiert auf der Anforderung A30.

3.3 URLs

```
1 urlpatterns = [  
2     path('aut/', include('aut.urls')),  
3     path('', RedirectView.as_view(url='/aut/')),  
4 ]
```

Listing 3.1: URLs des Projekts

Es gibt eine `urls.py` Datei für das Projekt und eine pro *App*. Von dem Projekt kann die Grundseite auf die Seite der *App* umgeleitet werden. Im Listing 3.1 wird zum einen die `urls.py` Datei der *App* inkludiert und zum Anderen der Hauptpfad des Projekts auf die *App* umgelenkt. In dieser Arbeit ist der Hauptpfad „aut“ [8].

```
1 urlpatterns = [  
2     #Basis Views  
3     path('', views.view_dashboard, name='view_dashboard'),  
4     path('requirement', views.view_requirement, name='view_requirement'),  
5     path('testcase', views.view_testcase, name='view_testcase'),  
6     path('testrun', views.view_testrun, name='view_testrun'),  
7     path('statistik', views.view_statistik, name='view_statistik'),  
8  
9     path('requirement/<int:pk>', views.edit_requirement, name='requirement_change'),  
10    path('requirement/create/', views.edit_requirement, name='requirement_create'),  
11    path('requirement/create/<int:pk>', views.edit_requirement, name='requirement_create'),  
12    ...
```

Listing 3.2: URLs der App

In der `urls.py` Datei der *App*, von der ein Ausschnitt im Listing 3.2 zu sehen ist, wird jeweils eine *URL* oder ein *URL-Schema* mit einer Funktion aus der `views.py` Datei verknüpft. Hierbei können noch Parameter angegeben werden, die an die *View-Funktionen* übergeben werden sollen. Sobald eine *URL* übereinstimmt, wird die damit verknüpfte *View* aufgerufen [20].

3.4 Modelle

Die Modelle der *App* werden basierend auf dem zuvor definierten *ER-Modell* aus Kapitel 2.2 erstellt mit leichten Abwandlungen. Im Listing 3.3 ist das Modell der

```

1 class requirement(models.Model):
2     #Private Keys, Foreign Keys and andere Beziehungen:
3     req_pk_requirementid = models.AutoField(primary_key=True, null=False, unique=True)
4     req_fk_ersteller = models.ForeignKey(User, on_delete=models.SET_NULL, null=True, blank=True)
5
6     #Attribute:
7     req_name = models.CharField(max_length=100, null=True, blank=True)
8     req_kommentar = models.CharField(max_length=300, null=True, blank=True)
9     req_datum_erstellung = models.DateTimeField(auto_now_add=True, null=True, blank=True)
10    req_datum_aenderung = models.DateTimeField(auto_now=True, null=True, blank=True)
11    req_beschreibung = models.CharField(max_length=300, null=True, blank=True)
12
13    #Funktionen:
14    def __str__(self):
15        return "r_" + str(self.req_pk_requirementid) + ": " + str(self.req_name)
16
17    def get_id(self):
18        return "r_" + str(self.req_pk_requirementid)
19
20    def get_absolute_url(self):
21        return reverse('aut:requirement_change', args=[str(self.req_pk_requirementid)])
22
23    #Metaoptionen
24    class Meta:
25        ordering = ["req_pk_requirementid"]

```

Listing 3.3: Das Modell der Requirements

Requirements angegeben. An diesem Beispiel soll das Beschreiben der Modelle durch *Django* erklärt werden. Der *Primary-Key* wird normalerweise automatisch generiert, aber damit ein verständlicherer Name benutzt werden kann, wurde dieser überschrieben. Die *Foreign-Keys* benötigen als besonderes Merkmal nur den Namen des anderen Modells das sie referenzieren. Die weiteren Attribute wurden mit den Standard Feldern, wie zum Beispiel *CharField*, definiert. Jedem Feld können außerdem Attribute zugewiesen werden. Die wichtigsten Attribute sind `blank=True` und `null=True`. Durch diese beiden Attribute dürfen die Felder sowohl leer sein, als auch in den *Forms* leer angezeigt werden [8]. Es kann gesehen werden, dass die „is-a“-Beziehung dadurch realisiert wurde, dass die *Requirements*, *Testcases* und *Testruns* alle die allgemeinen Attribute der Relation *Element* aus dem *ER-Diagramm* erhalten haben. Diese Realisierungsart war einfacher, da so keine *JOIN-Operationen* ausgeführt werden müssen, wenn beispielsweise die allgemeinen Attribute wie der Name eines *Testruns* gebraucht werden.

Neben den normalen Feldern, können in den Modellen auch noch Methoden und Metadaten definiert, beziehungsweise angegeben, werden. Eine generelle wichtige Funktion, die implementiert werden sollte, ist das Überschreiben der `__str__` Funktion, da sie angibt, wodurch die *Elemente* repräsentiert werden sollen [8]. Die andere Funktion `get_absolute_url` kann dazu genutzt werden zu jedem *Requirement* die entsprechende *URL* anzugeben, die zu der Bearbeitungsseite führt und den *Primary-Key* als Parameter übergibt. Durch diese Beschreibung der Modelle kann *Django* die entsprechenden Tabellen erstellen und übernimmt die Kommunikation mit der Datenbank [13].


```
1 class user_erweitern(models.Model):  
2     user = models.OneToOneField(User, on_delete=models.CASCADE, null=True, blank=True)  
3     gruppennummer = models.ForeignKey('projekt', on_delete=models.SET_NULL, null=True, blank=True)  
4     rolle = models.CharField(max_length=1, choices=ROLLEN, null=True, blank=True)
```

Listing 3.4: Die Erweiterungen des Users

Um den Nutzer zu implementieren wurde das vorgegebene *User-Modell* von *Django* erweitert. Die Erweiterungen sind in dem Listing 3.4 zu sehen. Um die *Elemente* den einzelnen Gruppen zuzuordnen, brauchte der Nutzer das Attribut `gruppennummer`. Das Attribut ist dabei ein *Foreign-Key* auf die erstellten Projekte. Eine Abweichung vom *ER-Modell* stellt die Realisierung der Studenten und des Professors dar. Da beide Nutzergruppen im *ER-Modell* als eigene Relationen definiert sind, sollten sie in der Implementierung eigene Modelle erhalten. Stattdessen hat es sich angeboten das *User-Modell* um das Attribut `rolle` zu erweitern.

Nachdem die Modelle erstellt wurden, müssen die Befehle `makemigrations` und `migrate` angewandt werden, da so sämtliche Änderungen an den Modellen ausgeführt werden. Mit `makemigrations` werden die `migration`-Dateien erstellt, um gegebenenfalls die Befehle zu kontrollieren, die von *Django* erstellt wurden. Das Ausführen der Migration geschieht durch den Befehl `migrate`. Damit werden alle notwendigen Tabellen erstellt und Änderungen durchgeführt [13].

3.5 Views

Der generelle Ablauf bei einem *HTTP Request* ist in der Darstellung 3.4 abgebildet. In *Django Applikationen* werden die verschiedenen Aufgaben auf verschiedene Dateien aufgeteilt. Die *URLs* werden mit den entsprechenden *Views* verlinkt. Darüber hinaus können bestimmte Parameter aus der *URL* an die *Views* zum weiteren Verarbeiten übergeben werden. In den *Views* können die benötigten Daten, die als Modelle in der `models.py` Datei definiert sind, angegeben werden. Die *HTML-Templates* werden dann im nächsten Schritt mit diesen Daten basierend auf der Logik in den *Views* befüllt, dafür haben sie an den entsprechenden Stellen befüllbare Platzhalter. Abschließend generiert die *View* die *HTTP Response*. Diese Aufteilung wird „Model View Template“ Architektur genannt [4] und kann mit der „Model View Controller“ Architektur verglichen werden [5].

Beispielsweise ist in Listing 3.5 die *View* dargestellt, um alle *Requirements* einer Usergruppe herauszufiltern. Die einzelnen *Views* sind als Funktionen in *Python* definiert [4]. Zuerst werden in Zeile zwei alle Nutzer mit der gleichen Gruppennummer herausgefiltert, bevor die *Requirements* aller Nutzer aus der Datenbank geholt werden. Die *Requirements* werden dann noch in Zeile drei nach absteigender Folge der IDs

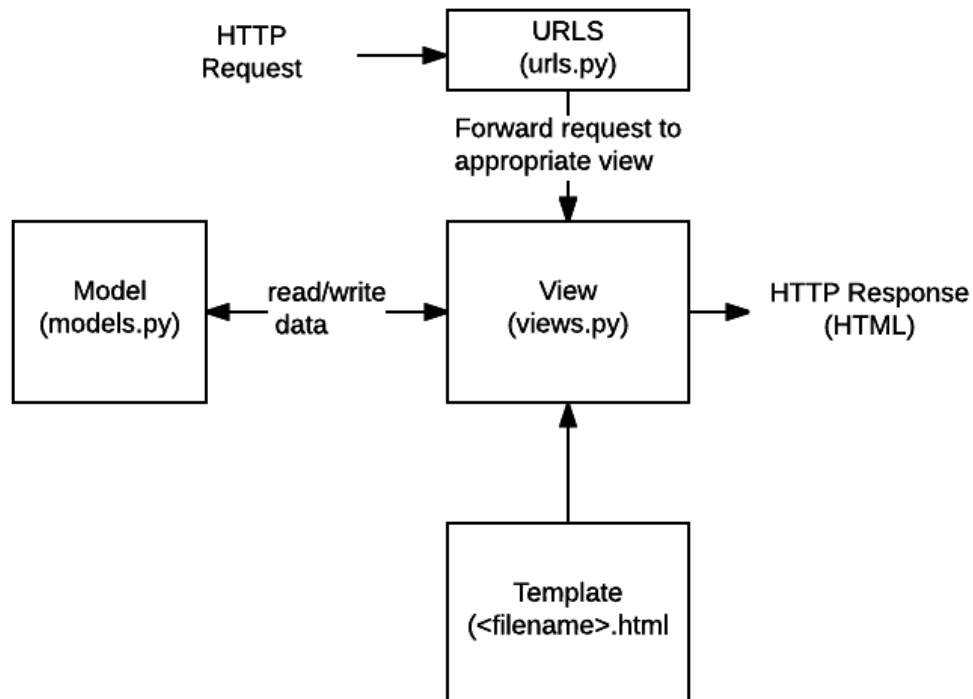


Abbildung 3.4: Der generelle Ablauf eines HTTP Requests, aus [4]

```
1 def view_requirement(request):
2     users = User.objects.filter(user_erweitern__gruppennummer=request.user.user_erweitern.gruppennummer)
3     req_for_usergroup = requirement.objects.filter(req_fk_ersteller__in=users).order_by('-req_pk_requirementid')
4
5     context = {
6         'requirements': req_for_usergroup,
7     }
8     return render(request, 'aut/010_requirement.html', context=context)
```

Listing 3.5: View-Funktion der Requirements Seite

Willkommen, florian

Dashboard Statistik Requirements TestCases TestRuns Logout

Neues Requirement

r_9: Test 4

r_8: Neues Requirement

r_7: Requirement

r_6: Test 3

r_3: Test 2

r_2: Test 1

r_9: Test 4

Speichern Löschen

Erstellung: 30. Januar 2021 13:28 Änderung: 30. Januar 2021 14:49 Ersteller: florian

Name	Test 4
TestCases	<input checked="" type="checkbox"/> tc_2: Test Testcase
Beschreibung	
Kommentar	

Notizen

Abbildung 3.5: Requirements-Liste auf der Requirements Seite, eigene Darstellung

sortiert, damit die neuesten *Requirements* in der Liste oben stehen. Wenn diese Variable an das *Template* übergeben werden soll, dann muss sie im *Context* angegeben werden. Es wird ein Name in Anführungsstrichen angegeben, der dann im *Template* benutzt werden kann. Der Einfachheit halber wurde immer derselbe Name, wie der, der Variablen verwendet. Zusammen mit dem *Template* muss dieser *Context* dann zum rendern in der Zeile acht aufgerufen werden. In der Abbildung 3.5 kann die auf diesem Weg erstellte Liste am linken Bildschirmrand gesehen werden. Der nähere Aufbau der *Templates* ist im Kapitel 3.6 beschrieben.

```

1 def edit_requirement(request, pk=None):
2     requ_instance, created = requirement.objects.get_or_create(req_pk_requirementid=pk)
3     if created == True:
4         requ_instance.req_fk_ersteller = request.user
5         requ_instance.save()
6     ...

```

Listing 3.6: View-Funktion der Requirements Seite

Eine weitere *View*, die eine andere Möglichkeit von *Django* präsentiert, ist in dem Listing 3.6 dargestellt. Hierbei wurde aus der *URL* noch ein Parameter übergeben und kann in der Funktion als Variable benutzt werden. Gebraucht wurde diese Funktionalität, damit sowohl neu erstellte *Elemente*, die zum Beispiel durch den Button „Neues Requirement“ aus der Abbildung 3.5 erstellt werden, als auch bereits vorhandene *Elemente*, bearbeitet werden können. Neu erstellte *Elemente* besitzen noch keinen *Primary-Key*, da sie noch nicht abgespeichert wurden. Dafür bekommen sie den default Wert *None* aus Zeile eins als *Primary-Key*. Ein bereits vorhandenes *Element* würde an dieser Stelle seinen *Primary-Key* angeben, zum Beispiel kann das

Element „r_2: Test 1“ über seinen *Primary-Key* zwei erhalten werden. Die Funktion `get_or_create` aus der Zeile zwei prüft, ob es zu dem angegebenen *Primary-Key* ein *Element* gibt, ansonsten erstellt es ein neues *Element*, zum Beispiel, wenn der *Primary-Key* *None* als Wert hat. Daneben gibt die Funktion auch noch Auskunft darüber, ob das *Element* erstellt wurde oder nicht. Diese Information wird als *Boolean* an die Variable `created` übergeben. Diese kann dann abgefragt werden und falls sie *True* ist, wird der Variable der aktuell eingeloggte Nutzer als Ersteller übergeben. Danach wird das *Element* abgespeichert, damit es auch einen *Primary-Key* erhält.

```
1 SELECT * FROM Requirements WHERE req_pk_requirementid = pk;
```

Listing 3.7: Die entsprechenden SQL-Anweisungen zu Listing 3.6

Django übernimmt die Kommunikation mit der Datenbank. Darum müssen keine *SQL-Anweisungen* per Hand geschrieben werden. Beispielsweise ist im Listing 3.7 die *SQL-Anweisung* angegeben, die aus dem Listing 3.6 generiert wird.

3.6 Templates und Filter

```
1 {% extends "aut/000_base_bar.html" %}
2
3 {% block titel %}
4 <title>Dashboard</title>
5 {% endblock %}
6
7 {% load aut_extras %}
8 ...
9 <tr>
10     <td>Requirements ohne TestCase</td>
11     <td {{ num_no_testcase|rote_markierung }}>{{num_no_testcase}}</td>
12 </tr>
```

Listing 3.8: Das Template des Dashboards

Innerhalb der *Tempaltes* gibt es ein paar Besonderheiten, die im Listing 3.8 gezeigt werden sollen. Beispielsweise unterstützen die *Templates* Vererbung. So kann ein *Template*, wie in dieser Arbeit, die Menüleiste realisieren und an die anderen *Templates* vererben. Das Schlüsselwort `extend` ist dazu in Zeile eins angegeben. Auf diesem Weg konnte verhindert werden, den gleichen Code an verschiedenen Stellen zu wiederholen. Ein weiterer Vorteil dadurch ist, dass Änderungen an nur einer Stelle zentral durchgeführt werden müssen. Funktionen werden in geschweiften Klammern mit Prozentzeichen geschrieben. Daneben können Blöcke definiert werden, die dann von den erbenden *Templates* ausgefüllt werden können. In Zeile drei wird beispielsweise der Block „titel“ vom erbenden *Template* überschrieben. Auf diese Weise kann

eine einheitliche Struktur erzeugt werden [20].

```
1 {% for user in users %}
2     <tr>
3         <th> {{ user.username }} </th>
4         <td> {{ user.requirement_set.count }}</td>
5         <td> {{ user.testcase_set.count }}</td>
6         <td> {{ user.testrun_set.count }}</td>
7     </tr>
8 {% endfor %}
```

Listing 3.9: Das Template des Dashboards

In *Templates* können auch Flusskontrollstrukturen, wie zum Beispiel eine `For` Schleife, verwendet werden. Das ist im Listing 3.9 dargestellt. Auf Variablen wird mit doppelt geschweiften Klammern zugegriffen. Es können aber auch auf die Attribute der Variablen zugegriffen werden, wie in der Zeile drei verdeutlicht werden soll. Eine weitere besondere Syntax sind die *Filter*, die in *Templates* benutzt werden können. Diese *Filter* nehmen den Wert links von der *Pipe* und wenden darauf definierte Funktionen an, beispielsweise die `count` Funktion in der Zeile vier. Dabei können auch eigene Funktionen verwendet werden. Dazu muss die Datei „`aut_extra`“ mit diesen Funktionen erst über den Befehl `load aut_extra` geladen werden, wie in der Zeile sieben vom Listing 3.8 zu sehen ist [20].

Kapitel 4

Features

tc_4: Testcase			
Erstellung: 31. Januar 2021 11:06 Änderung: 31. Januar 2021 11:07 Ersteller: florian			
Name	Testcase		
Requirement	<input checked="" type="checkbox"/> r_11: Ein Requirement		
TestSchritte	Schritt	erwartetes Ergebnis	Löschen?
	Nach diesem Schritt		<input type="checkbox"/>
	Folgt der nächste		<input type="checkbox"/>
Neue Zeile			
Beschreibung			

Abbildung 4.1: Das Bearbeiten eines Testcase, eigene Darstellung

Die Software „Anforderungen und Testen“ bietet verschiedene Features an, um das Anforderungs- und Testmanagement zu unterstützen. Es können Requirements, Testcases und Testruns erstellt werden. Dazu können die gewünschten Felder aus den Anforderungen A60, A70 und A80 mit Werten befüllt werden. In der Abbildung 4.1 ist beispielsweise die Bearbeitungsseite der Testcases dargestellt.

Daneben bietet die Software auch die wichtige Funktion, Testruns nicht nur zu definieren, sondern diese auch durchzuführen. Die Durchführung ist in der Abbildung 4.2 zu sehen. Dabei wird die Zeit, die gebraucht wird um ihn durchzuführen, gestoppt [30]. Erst wenn bei allen Testschritten der Status „pass“ ausgewählt wurde, gilt dieser Testrun als erfolgreich durchgeführt. Die Logik, die das prüft, ist

tr_33: Testrun

00:00:34

Schritt	erwartetes Ergebnis	tatsächliches Ergebnis	Status
Nach diesem Schritt	es funktioniert	es funktioniert wirklich	<input checked="" type="radio"/> pass <input type="radio"/> fail
Folgt der nächste Schritt	es funktioniert ebenfalls		<input checked="" type="radio"/> pass <input type="radio"/> fail
TestRun Abschießen			

Abbildung 4.2: Die Durchführung eines Testruns, eigene Darstellung


```

1 if all(item == list_mit_Ergebnissen[0] == 'p' for item in list_mit_Ergebnissen):
2     testr_instance.testr_status = 'p'
3 else:
4     testr_instance.testr_status = 'f'

```

Listing 4.1: Der Generator Ausdruck

im Listing 4.1 dargestellt und wird als *Generator-Ausdruck* realisiert. Die Variable `list_mit_Ergebnissen` enthält dabei den Status aller *Testschritte* als Liste [33]. Bevor der Nutzer die Seite verlässt, zum Beispiel um den *Testrun* abzuschließen, wird er um eine Bestätigung der Aktion gebeten. Dadurch soll verhindert werden, dass der Nutzer aus Versehen den *Testrun* frühzeitig beendet [29].

Willkommen, florian
Dashboard Statistik Requirements TestCases **TestRuns** Logout


Neuer Testrun
tr_39: Testrun

tr_39: Testrun
Erstellung: 31. Januar 2021 20:21 Änderung: 31. Januar 2021 20:22 Ersteller: florian

Name	Testrun			
Beschreibung	Das hier ist ein Testrun			
Kommentar				
Status	p			
Dauer	0:00:15			
TestCase	tc_4: Testcase			
Schritte	Schritte	erwartetes Ergebnis	tatsächliches Ergebnis	Status
	Nach diesem Schritt	es funktioniert	es funktioniert wirklich	p
	Folgt der nächste Schritt	es funktioniert ebenfalls	dieser Schritt auch	p

Abbildung 4.3: Die Anzeige bei einem abgeschlossenen Testrun, eigene Darstellung

Nachdem ein *Testrun* durchgeführt wurde, ist er nicht mehr editierbar. Stattdessen zeigt er, wie in Abbildung 4.3 gezeigt, den getesteten *Testcase* mit seinen Schritten und Ergebnissen an. Eine farbige Markierung an der linken Seite in der Listenübersicht zeigt ebenfalls an, ob der *Testrun* erfolgreich war oder nicht.

Während der *Login* und *Logout* auf der von *Django* vorgegebenen Authentifizierung basiert [10], musste der *Signup* Prozess extra eingebracht werden [19]. Der Nutzen davon soll sein, dass sich die Studenten eigene Accounts anlegen können mit einem Benutzernamen, Passwort und einer Gruppennummer. Dadurch muss der Admin,

beziehungsweise der Professor, nicht für jeden Student einen eigenen Account anlegen.

Kapitel 5

Usability Test

Damit ein *Usabilitytest* durchgeführt werden konnte, wurde das Tool „Anforderungen und Testen“ auf der Webseite *Heroku* hochgeladen und steht unter dem Link <https://anforderungen-und-testen.herokuapp.com/aut/> zum Testen bereit [18].

Der Inhalt des *Usabilitytests* bestand darin, jeweils ein *Requirement*, *Testcase* und *Testrun* zu erstellen. Dazu wurden Beispieldaten in einer Anleitung angegeben. Darüber hinaus sollte ein *Testrun* durchgeführt werden, bevor zuletzt die Statistik Seite überprüft werden sollte.

Daraus folgten verschiedene Rückmeldungen, die zu einer Verbesserung der Software führen sollten. Der Notizen-Button sollte größer gemacht werden, damit er auffälliger wird. Die Seite auf die der *Testrun* durchgeführt wird, sollte auch überarbeitet werden und die einzutragenden Felder ersichtlicher machen. Eine Tabelle sollte hier verwendet werden, um das zu erreichen.

Nachdem ein *Element* abgespeichert wurde, war nur sofort ersichtlich, dass etwas gespeichert wurde, wenn sich der Name verändert hat. Basierend auf diesem Feedback wurde in *HTML* eine *Snackbar* beziehungsweise ein *Toast* realisiert. Dadurch bekommen die Nutzer ein Feedback, wenn der „Speichern“-Button gedrückt wird [21].

Die Begrenzung der erlaubten Zeichen in den Eingabefeldern wurde ebenfalls überdacht, da die Nutzer nicht genug Platz hatten, um die Texte einzugeben. Die Länge der Felder wurde somit vergrößert.

Auf dem *Dashboard* sollte es eine bessere Unterscheidung geben zwischen anklickbaren Buttons und der Tabelle. Durch ein neues Design der Buttons stechen diese nun besser hervor.

Eine weitere Rückmeldung war, dass die Gebrauchsanleitung in irgendeiner Form der Software beiliegen sollte, um vor allem die Farben zu erklären. Dazu wurden

auf den Seiten der *Requirements*, *Testcases* und *Testruns* Erklärungen geschrieben, die angezeigt werden, falls kein *Element* ausgewählt ist.

Kapitel 6

Fazit

Die Kategorie, die als Attribut bei den *Requirements* gefordert wurde, wurde in dieser Version nicht implementiert. In einer zweiten Version könnten die Kategorien ausgebaut werden, um so die verschiedenen Arten von Anforderungen, wie in Kapitel 1.1 aufgelistet, zu unterscheiden. Der Nutzer könnte ebenfalls in der Lage sein, eigene Kategorien zu definieren. Dann würde sich anbieten eine Suche oder einen Filter zu implementieren. Da diese Features eher ablenken würden, war es vorzuziehen, sich diese für eine zweite Version zu merken.

Eine Überarbeitung des *Dashboards* in Form einer Verschönerung könnte ebenfalls sinnvoll sein, um die Informationen schneller vermitteln zu können. Es könnten Diagramme und andere Darstellungsformen, wie die vorgestellten Tools in Kapitel 1.2 sie benutzen, verwendet werden.

Abschließend kann zusammengefasst werden, dass die gestellten Anforderungen durch die Software „*Anforderungen und Testen*“ erfüllt werden und die Erstellung eines leichtgewichtigen *Softwaretools* für das *Anforderungs- und Testmanagement* im *Softwareentwicklungsumfeld* somit erfolgreich abschlossen werden konnte.

Literaturverzeichnis

- [1] *A guide to the project management body of knowledge (PMBOK guide)*. Project Management Institute, Newtown Square, Pennsylvania, fifth edition edition, 2013.
- [2] Mark Hastenteufel and Sina Renaud. *Software als Medizinprodukt: Entwicklung und Zulassung von Software in der Medizintechnik*. Springer Lehrbuch. 2019.
- [3] Paul C. Jorgensen. Exploratory Testing. In Paul Jorgensen, editor, *Software testing*, pages 369–374. Auerbach, Boca Raton, 2018.

Online-Quellen

- [4] Django introduction - Learn web development | MDN. <https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction>, 22.01.2021.
- [5] MVC (model view controller) :: Modell-Ansicht-Steuerung :: ITWissen.info. <https://www.itwissen.info/MVC-model-view-controller-Modell-Ansicht-Steuerung.html>, 25.01.2021.
- [6] Web-Framework :: web framework :: ITWissen.info. <https://www.itwissen.info/Web-Framework-web-framework.html>, 25.01.2021.
- [7] Django Tutorial Part 2: Creating a skeleton website - Learn web development | MDN. https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/skeleton_website, 27.01.2021.
- [8] Django Tutorial Part 3: Using models - Learn web development | MDN. <https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Models>, 27.01.2021.
- [9] Django Tutorial Part 4: Django admin site - Learn web development | MDN. https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Admin_site, 27.01.2021.
- [10] Django Tutorial Part 8: User authentication and permissions - Learn web development | MDN. <https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Authentication>, 27.01.2021.
- [11] Setting up a Django development environment - Learn web development | MDN. https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/development_environment, 27.01.2021.

- [12] Writing your first Django app, part 1 | Django documentation | Django. <https://docs.djangoproject.com/en/3.1/intro/tutorial01/>, 28.01.2021.
- [13] Writing your first Django app, part 2 | Django documentation | Django. <https://docs.djangoproject.com/en/3.1/intro/tutorial02/>, 28.01.2021.
- [14] Best Test Management Software & Test Case Management Tool. <https://testcollab.com/>, 29.01.2021.
- [15] Pricing | Test Collab. <https://testcollab.com/pricing/>, 29.01.2021.
- [16] TestCaseLab - Powerful tool for QA engineers. <https://testcaselab.com/>, 29.01.2021.
- [17] What is Requirements Traceability Matrix (RTM)? Example Template. <https://www.guru99.com/traceability-matrix.html>, 29.01.2021.
- [18] Cloud Application Platform | Heroku. <https://www.heroku.com/>, 31.01.2021.
- [19] Create Advanced User Sign Up View in Django | Step-by-Step - DEV Community. <https://dev.to/coderasha/create-advanced-user-sign-up-view-in-django-step-by-step-k9m>, 31.01.2021.
- [20] Django at a glance | Django documentation | Django. <https://docs.djangoproject.com/en/3.1/intro/overview/>, 31.01.2021.
- [21] How To Create a Snackbar / Toast. https://www.w3schools.com/howto/howto_js_snackbar.asp, 31.01.2021.
- [22] Online Test Case Management Tool - TestLodge. <https://www.testlodge.com/>, 31.01.2021.
- [23] Quick install guide | Django documentation | Django. <https://docs.djangoproject.com/en/3.1/intro/install/>, 31.01.2021.
- [24] Atlassian. Jira | Issue & Project Tracking Software | Atlassian. <https://www.atlassian.com/software/jira>, 29.01.2021.
- [25] Axure. Axure RP 9 - Prototypes, Specifications, and Diagrams in One Tool. <https://www.axure.com/>, 28.01.2021.
- [26] Johannes Borchard. 10 Usability Heuristiken nach Nielsen - Systeme bewerten ohne zu fluchen. <https://www.usabilityreport.de/usability-heuristiken-nielsen>, 25.02.2020.

- [27] ISO. ISO 9241-110:2020. <https://www.iso.org/standard/75258.html>, 29.01.2021.
- [28] ISO. ISO 9241-11:2018. <https://www.iso.org/standard/63500.html>, 29.01.2021.
- [29] Khan. Warn user before leaving web page with unsaved changes. <https://stackoverflow.com/questions/7317273/warn-user-before-leaving-web-page-with-unsaved-changes>, 2011.
- [30] Mark. plain count up timer in javascript. <https://stackoverflow.com/questions/5517597/plain-count-up-timer-in-javascript>, 2011.
- [31] Nielsen Norman Group. 10 Usability Heuristics for User Interface Design. <https://www.nngroup.com/articles/ten-usability-heuristics/>, 29.01.2021.
- [32] QA Touch. Test Case Management Tool | Software Test Management tool | QA Touch. <https://www.qatouch.com/>, 19.01.2021.
- [33] Stack Overflow. Python: determine if all items of a list are the same item - Stack Overflow. <https://stackoverflow.com/questions/3787908/python-determine-if-all-items-of-a-list-are-the-same-item>, 31.01.2021.
- [34] Xray. Xray - Cutting Edge Test Management for Jira. <https://www.getxray.app/>, 15.01.2021.