Name der Veranstaltung: Software Engineering I

Teamname: Team 04

Namen: Teammitglied 1, Teammitglied 2, Teammitglied 3, Teammitglied 4

Inhaltsverzeichnis

Einleitung

Als semesterübergreifende Prüfungsleistung in SWE1 gibt es das Prüfungsformat des Entwurfes, dieser beinhaltet in Gruppenarbeit die Modellierung eines kleinen IT-Systems mit UML, sowie das Erstellen eines dazugehörigen User-Interface-Prototypen auf der Basis von HTML und Bash-Skripten.

An dieser Aufgabe haben ein semesterlang Teammitglied 4, Teammitglied 1, Teammitglied 3 und Teammitglied 2 arbeitsteilig gearbeitet. Nach einer kurzen Einigungsphase legte das Team einstimmig die Modellierung eines Ticket-Systems in Form eines Issues-Trackings-Systems als Semesterprojekt fest. Was genau damit gemeint ist und wie im Einzelnen unsere Ergebnisse aussehen, soll im Hauptteil dieses Textes beleuchtet werden. Hier soll erstmal ein Überblick zum groben Projektablauf gegeben werden.

Eingangs gab es einen recherchelastigen Teil der Modellierung. Hier wurde mittels Literatur analysiert, was professionelle Ticketsysteme leisten, woraus wir später für unseren Prototypen Anwendungsfälle ableiten konnten und entsprechende Aktivitätsdiagramme schrieben. Jene wurden erst auf Papier entwickelt und dann mit UMLet digitalisiert vorrangig für die UML-Diagramme verantwortlich war Teammitglied 2, während Teammitglied 3 parallel dazu die Wireframes für den html-Prototypen designte. Diese Punkte bildeten die Basis für die Prototypentwicklung, welche sich in zwei Phasen, nämlich Backend- und Frontendentwicklung unterteilt. Ein Team, bestehen aus Teammitglied 4, Teammitglied 1 und entwickelte maßgeblich an dem Prototypen und verschönerten die Anwendung abschließend durch passende Styles. Nach Backend-Schluss haben die anderen Teammitglieder, Teammitglied 3 und Teammitglied 2, die entwickelte Anwendung auf Funktion getestet. Kritik wurde wie bei den agilen Methoden sehr zeitnah eingearbeitet.

Aus der Retrosicht ähnelt unsere Arbeitsweise eher dem Wasserfallmodell, weil wir in den klar unterscheidbaren Arbeitsphasen: Planung, Modellierung, Implementierung und Tests vorgingen. Obwohl gegenwärtig statt jener statischen Arbeitsweise agile Vorgehen ein grundsätzlich besseres Softwareergebnis zu liefern versprechen, ist für die Größe unseres SWE1-Projekts das Wasserfallmodell völlig ausreichend. Ein Verfahren wie SCRUM würde unser Projekt künstlich aufblähen, als dass sich dadurch eine sinnvollere Struktur ergeben würde. [Teammitglied 2]

Modellierung

Anwendungsfalldiagramm

Die Frage, was ein Ticketsystem können muss, haben wir im Team viel diskutiert. Wesentlich dazu beigetragen haben neben dem Buch OTRS [1], diejenigen Teammitglieder, die durch ihre Berufserfahrung schon mit professionellen Ticketsystemen gearbeitet haben.

Daraus wurden von uns Anwendungsfälle entwickelt und in einem Use-Case-Diagramm dargestellt.

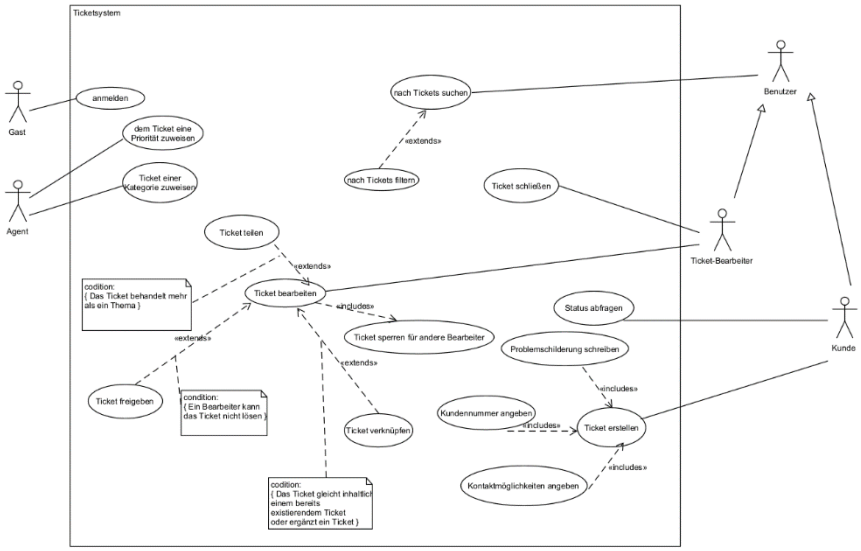


Abbildung 1 Anwendungsfalldiagramm

Dem Diagramm ist zu entnehmen, dass wir genau fünf Akteure identifiziert haben: der Gast, der Agent, der Benutzer, der Ticketbearbeiter und der Kunde, jene sollen unser System bedienen, steuern oder nutzen. Der Gast besitzt lediglich die Möglichkeit, sich im System anzumelden. Das hat den Hintergrund, dass eben nicht jeder Mensch die Tickets eines Unternehmens einsehen können sollte, sondern nur diejenigen, die mit dem Ticketsystem in jedweder Form arbeiten werden. Einmal angemeldet findet ein Rollentausch, in den Agenten, Ticketbesitzer oder den Kunden statt. Der Agent kann Tickets verwalten, indem er diesen Prioritäten und Kategorien zuweist. Jene Tickets landen dann in einer Liste, wo jeder Bearbeiter ein solches Ticket dynamisch bearbeiten kann.

Der Ticketbearbeiter ist die Schlüsselfigur, die Tickets abschließen, bearbeiten und freigeben kann. Darüber hinaus kann er Tickets teilen oder verknüpfen, um effektiv an der Lösung von Kundenanliegen zu arbeiten. Der Prozess “ein Ticket sperren” geschieht dabei immer, wenn ein Ticket bearbeitet wird, denn das Ticket wird hier nur für andere Bearbeiter gesperrt, damit nicht mehrere an einem Ticket arbeiten können.

An dieser Stelle sei gesagt, dass wir, um die Komplexität des Prototyps einzuschränken, nicht jeden Anwendungsfall umgesetzt haben. So kann unser Prototyp beispielsweise keine Tickets zusammenführen. [Teammitglied 1]

Klassendiagramm

Um im Folgeschritt die einzelnen Akteure und ihr Zusammenspiel mit den Tickets zu verdeutlichen, nutzen wir ein Klassendiagramm:

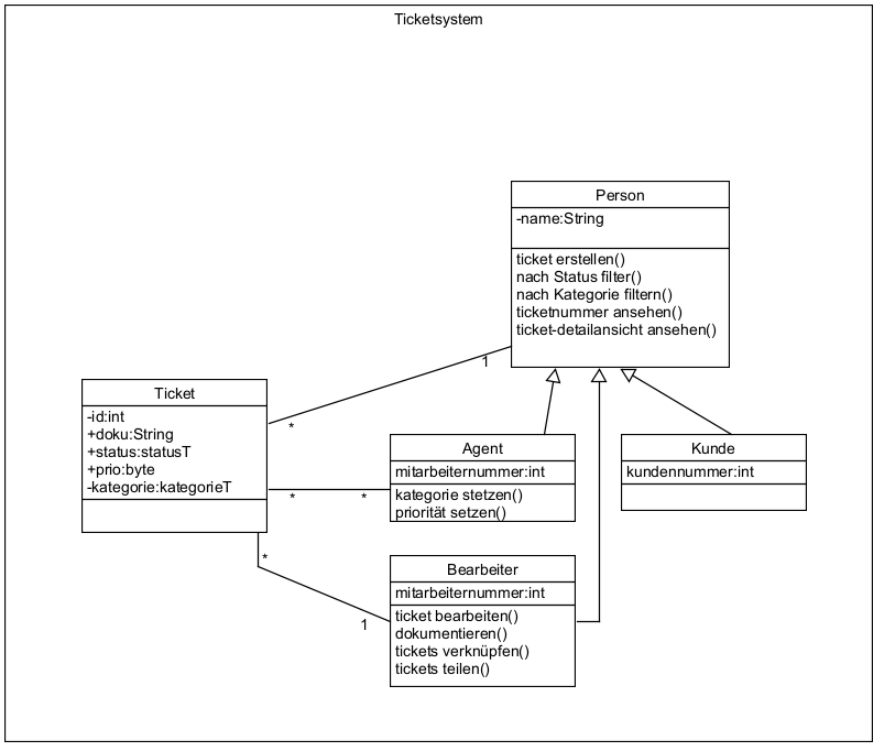


Abbildung 2 Klassendiagramm

Dem Diagramm ist zu entnehmen, dass es die Personen: Agent, Bearbeiter und Kunde gibt, die über die Methoden der Person auch noch eigene Methoden kennzeichnen. So kann der Agent nicht nur Kategorien setzen, sondern eben auch nach Tickets filtern (->Vererbung).

Außerdem ist aus den Assoziationen zu erkennen, dass ein Agent mehrere Tickets ihre Werte zuweist und entsprechend mehrere Tickets von einem Agenten zugewiesen worden sind. Oder auch, dass ein Ticketbearbeiter grundsätzlich mehrere Tickets bearbeiten kann, aber jedes Ticket nur von genau einem Bearbeiter zur gleichen Zeit angetastet wird.

Außerdem können wir an dem Diagramm die Attribute für zum Beispiel die Tickets ablesen, die uns vorgeben, welche Filterfunktionen im nächsten Schritt zu implementieren sind. Genau das ist ein Punkt der Aufgabenvielfalt von Klassen, neben der Typisierung von Objekten und der damit einhergehenden Erleichterung bei objektorientierter Programmierung. [2] [Teammitglied 2]

Aktivitätsdiagramme

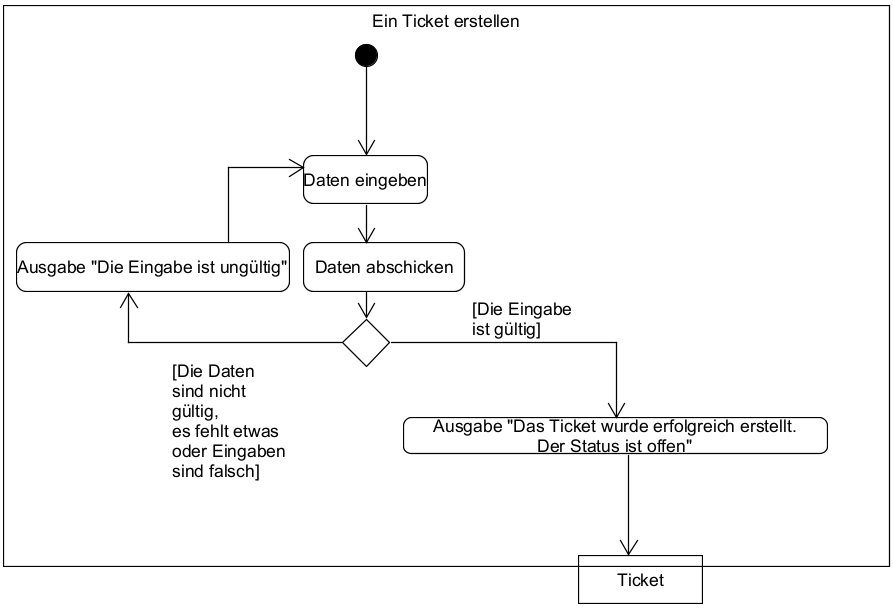


Abbildung 3 Aktivitätsdiagramm "Ticket erstellen"

Im Folgenden Teil unserer Zusammenarbeit haben wir für ausgewählte Aktionen Diagramme ausgearbeitet, die eben jede beschreiben. Ein sehr einfaches Diagramm, das die Grundlage für unser Ticketsystem liefert, ist das Diagramm “Ticket erstellen”. Diese Aktion betrifft den Kunden, es wird eine Dateneingabe überprüft und das Ticket anschließend als Objekt zurückgegeben. Diese Übergabe ersetzt den klassischen Endknoten. [Teammitglied 2]

Tickets bearbeiten

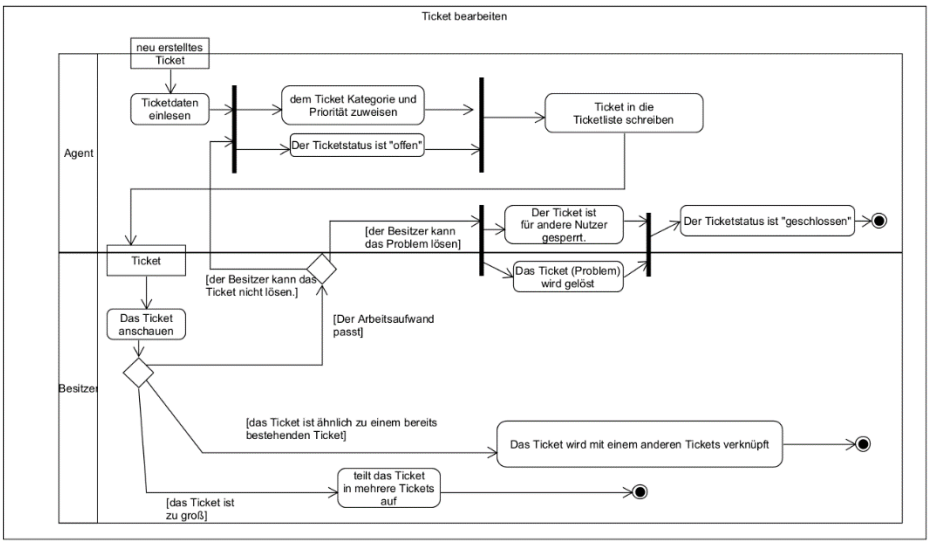


Abbildung 4 Aktivitätsdiagramm “Ticket bearbeiten”

Natürlich haben wir die wohl wichtigste Aktion in unserem Projekt modelliert, das Ticketbearbeiten. Schließlich stellt genau das den Zweck eines Ticketsystems dar. Diese Aktion besitzt im Wesentlichen zwei Akteure (Agent und Besitzer), die jeweils

zwei Aktivitätsbereiche (Schwimmbahnen) haben. Beginnend mit dem Objekt “neu erstelltes Ticket”, werden zunächst die Ticketdaten abgerufen (Ticketdaten einlesen). Nun müssen zwei Aktionen erfolgen, bevor das Ticket an einen Ticketbearbeiter weitergegeben werden kann. Der Agent muss die Kategorie und die Priorität festlegen und der Ticketstatus muss auf offen gesetzt werden, damit alle möglichen Ticketbearbeiter wissen, dass dieses Ticket noch unbearbeitet ist. Dieser Prozess wird durch eine Parallelisierung im Aktivitätsdiagramm dargestellt. Erst danach erscheint das Ticket in der Liste aller Ticketbesitzer. Im Szenario der Ticketbearbeitung können für den Ticketbesitzer/-bearbeiter jetzt folgenden Fälle auftreten:

Im ersten Fall, wenn das Ticket zu umfangreich ist, wird es in mehrere Tickets aufgeteilt. An dieser Stelle endet unsere Modellierung, was nach dem Modellverständnis von Stachowiak dem verkürzenden und pragmatischen Merkmal entspricht. Verkürzend deswegen, weil wir diesen Fall nicht für die entstehenden Teiltickets weiter modellieren, die den vorangestellten Bearbeitungsprozess ebenso durchlaufen würden. Das pragmatische Merkmal bezieht sich auf die Zweckgebundenheit. Das Aktivitätsdiagramm “Ticket bearbeiten” bildet von der Realität nur eben den Abschnitt aus, indem tatsächlich ein Ticket bearbeitet wird. Alle anderen Fälle werden in diesem Diagramm nicht näher beleuchtet, weshalb auch im zweiten Fall, wenn ein ähnliches Ticket bereits vorhanden ist, nicht weiter modelliert wird.

Nur im dritten Fall, wenn der Arbeitsaufwand angemessen ist, geht die Modellierung weiter. Es folgt die nächste Fallunterscheidung: “ist das Ticket für den Bearbeiter lösbar”. Wenn dem so, werden die Aktionen "Problem wird gelöst" und "Das Ticket für andere Nutzer wird gesperrt" gleichzeitig (Parallelisierungsknoten) durchgeführt. Anschließend erfolgt die Synchronisation, und mit Erfolg wird der Ticketstatus auf "geschlossen" gesetzt.

Damit ist die Bearbeitung des Tickets beendet. Dieser Bearbeitungsprozess ermöglicht eine organisierte Arbeitsweise, indem Besitzer und Agenten die Möglichkeit haben, sich durch Telefon und die Kontaktdaten auszutauschen. Wir hätten in unserem Prototyp auch einbauen können, dass der Besitzer und der Agent bei Rückfragen miteinander kommunizieren zu können, wie in vielen anderen Ticketsysteme. Aber damit es zu einer erfolgreichen Interaktion mit zwischen den beiden Akteuren kommen kann, wie in Artikel [3, S.137] beschrieben, ist eine professionelle, moderne und geplante Kommunikation entscheidend. Die Nutzer müssen diesem Ansatz gerecht werden, um den digitalen Wandel in IT-Organisationen voranzutreiben." [Teammitglied 1]

Der Prototyp

Weiter möchten wir genau auf den Prototypen eingehen und exemplarische Interaktionen anhand von Screenshots mit Bezug zu unseren entsprechenden UML-Diagrammen erläutern. Beim Aufrufen unserer Website des Prototyps wird zunächst ein Anmeldefenster angezeigt, das eine Authentifizierung erfordert, um auf das Ticketsystem zugreifen zu können. Die Authentifizierung erfolgt im Code mittels einer If-Bedingung, welche auf die genaue Schreibweise der Eingaben für den Benutzernamen und für das dazugehörige Passwort prüft (case-sensitive).

```
#!/usr/bin/env bash

echo "Content-Type: text/html"
echo

username=$(echo $QUERY_STRING | grep -o 'username=[^&]*' | sed 's/^username=//g')
password=$(echo $QUERY_STRING | grep -o 'password=[^&]*' | sed 's/^password=//g')


if test "$username" = "theo" && test "$password" = "[redacted]"; then
  ./ts_kundedashboard.sh
elif test "$username" = "bruno" && test "$password" = "[redacted]"; then
  ./ts_admindashboard.sh
else
echo "<html><head><title>Login fehlgeschlagen</title></head><body><h1>Login fehlgeschlagen</h1><p>Anmeldedaten ungültig.</p></body></html>"
fi

```

Abbildung 5 ts_loginskript.sh

Die Anmeldefunktionalität ermöglicht uns, die spezifischen Funktionen und Zugriffsrechte möglichst realitätsnah zu demonstrieren. Während der Anwender Tickets lediglich erstellen und sie danach nur lesen kann, ermöglicht die Bearbeiterrolle zusätzlich das Ändern von Ticketeigenschaften. In unserem Ticketsystem ist dies wie folgt zu erkennen:

Ticket aus Sicht des Anwenders, nachdem es erstellt wurde:

Ticket 6

Kd.Name: Theo

Kategorie: IT-Support

Bearbeiter: Agent

Status: Neu

Telefonnummer: 123456789

Priorität: 3

Datum	Dokumentation
20.02.2024 14:19	**Hallo, mein Rechner startet nicht. Hilfe!!**

Zum Dashboard

Abbildung 6 https://informatik.hs-bremerhaven.de/docker-step2023-team-04-web/cgi-bin/ts_kundeticketdetail.sh?ticketid=6

Ticket aus Sicht des Bearbeiters, nachdem das Ticket gesperrt wurde und bereits eine Dokumentation hinterlegt ist:

Ticket 6

Kd.Name: Theo

Kategorie: IT-Support

Bearbeiter: Bruno

Status: Bearbeitung

Telefonnummer: 123456789

Priorität: 3

Datum	Dokumentation
20.02.2024 14:19	Hallo, mein Rechner startet nicht. Hilfe!!
20.02.2024 14:20	Hallo, ich habe das Ticket gesperrt und werde nun mit der Problemlösung fortfahren.

Dokumentation:

Ticketstatus ändern
Kategorie setzen Prio
Speichern

Abbildung 7 https://informatik.hs-bremerhaven.de/docker-step2023-team-04-web/cgi-bin/ts_kundeticketdetail.sh?ticketid=6

Insbesondere möchten wir den Aspekt der Statusänderung hervorheben, wozu wir ein passendes Zustandsdiagramm erstellt haben.

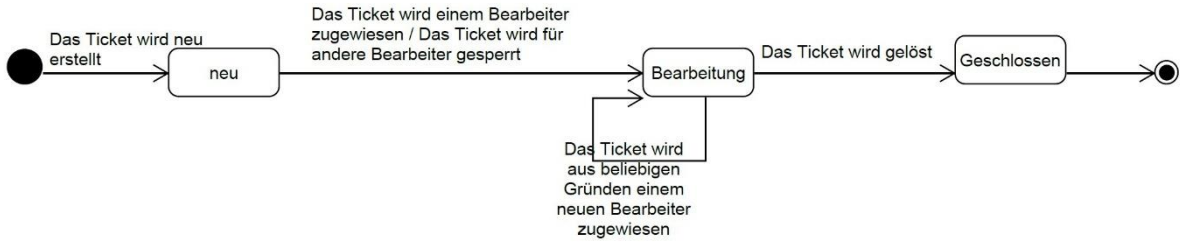


Abbildung 8 Zustandsdiagramm "Ticket"

In diesem wird der Lebenszyklus eines Tickets dargestellt. Wenn ein Nutzer ein neues Ticket erstellt, wird es automatisch mit dem Ticketstatus „neu" versehen.

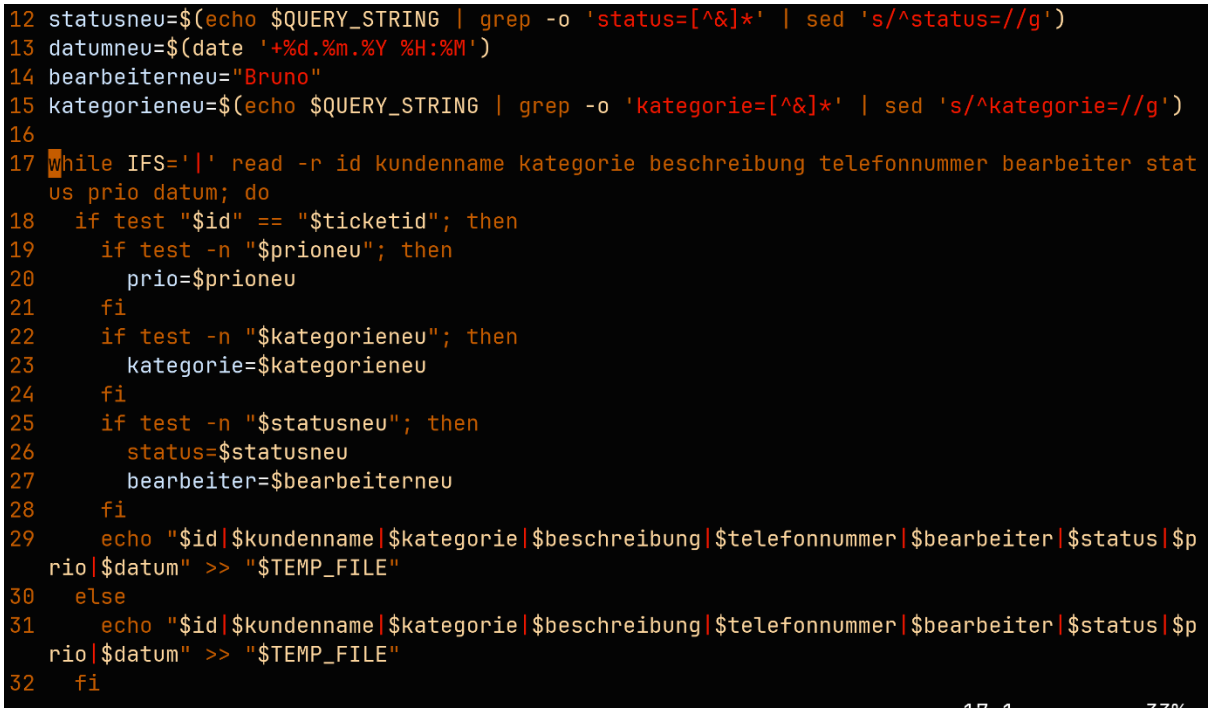
Im unteren Bild zu erkennen in Zeile 10.

```
1  #!/usr/bin/env bash
2
3  TICKET_FILE="ts_tickets.csv"
4
5  kundenname=$(echo $QUERY_STRING | grep -o 'kundenname=[^&]*' | sed 's/^kundenname=//g ;
   s/+/ /g ; s/%C3%A4/ä/g ; s/%C3%B6/ö/g ; s/%C3%BC/ü/g ; s/%C3%84/Ä/g ; s/%C3%96/Ö/g ; s/%
   C3%9C/Ü/g ; s/%2C/,/g ; s/%3F/?/g ; s/%21/!/g')
6  kategorie=$(echo $QUERY_STRING | grep -o 'kategorie=[^&]*' | sed 's/^kategorie=//g')
7  beschreibung=$(echo $QUERY_STRING | grep -o 'beschreibung=[^&]*' | sed 's/^beschreibung=
   //g; s/+/ /g ; s/%C3%A4/ä/g ; s/%C3%B6/ö/g ; s/%C3%BC/ü/g ; s/%C3%84/Ä/g ; s/%C3%96/Ö/g
   ; s/%C3%9C/Ü/g ; s/%2C/,/g ; s/%3F/?/g ; s/%21/!/g')
8  telefonnummer=$(echo $QUERY_STRING | grep -o 'telefonnummer=[^&]*' | sed 's/^telefonnumm
   er=//g')
9  bearbeiter="Agent"
10 status="Neu"
11 prio=3
12 datum=$(date '+%d.%m.%Y %H:%M')
13
14 zeile=$(wc -l < "$TICKET_FILE")
15 neueid=$((zeile + 1))
16
17 echo "$neueid|$kundenname|$kategorie|$beschreibung|$telefonnummer|$bearbeiter|$status|$p
   rio|$datum" >> "$TICKET_FILE"
```

Abbildung 9 ts_kundeticketspeichern.sh

Sobald das Ticket einem Bearbeiter zugewiesen oder vom Bearbeiter selbst gesperrt wurde, ändert sich der Status in „Bearbeitung".

Im unteren Bild zu erkennen in Zeile 14 und Zeilen 25-27.



```
12 statusneu=$(echo $QUERY_STRING | grep -o 'status=[^&]*' | sed 's/^status=//g')
13 datumneu=$(date '+%d.%m.%Y %H:%M')
14 bearbeiterneu="Bruno"
15 kategorieneu=$(echo $QUERY_STRING | grep -o 'kategorie=[^&]*' | sed 's/^kategorie=//g')
16
17 while IFS='|' read -r id kundenname kategorie beschreibung telefonnummer bearbeiter stat
   us prio datum; do
18   if test "$id" == "$ticketid"; then
19     if test -n "$prioneu"; then
20       prio=$prioneu
21     fi
22     if test -n "$kategorieneu"; then
23       kategorie=$kategorieneu
24     fi
25     if test -n "$statusneu"; then
26       status=$statusneu
27       bearbeiter=$bearbeiterneu
28     fi
29     echo "$id|$kundenname|$kategorie|$beschreibung|$telefonnummer|$bearbeiter|$status|$p
   rio|$datum" >> "$TEMP_FILE"
30   else
31     echo "$id|$kundenname|$kategorie|$beschreibung|$telefonnummer|$bearbeiter|$status|$p
   rio|$datum" >> "$TEMP_FILE"
32   fi
```

Abbildung 10 ts_ticketupdate.sh

Von hier aus könnte aus beliebigen Gründen das Ticket einem neuen Bearbeiter (sofern im System gepflegt) zugewiesen werden, wodurch der Status unverändert bleibt. Unser Prototyp stellt derzeit nur einen Bearbeiter zur Verfügung. Nachdem das Problem bzw. Ticket gelöst wurde, wird der Ticketstatus durch den Bearbeiter auf „geschlossen“ geändert.

Jede Statusänderung spiegelt einen entscheidenden Abschnitt in der Bearbeitung und Ticketlösung wider und zeigt auf, dass bei Auftreten neuer Probleme der Prozess neu beginnt. Dieser Ablauf kann beliebig oft erfolgen und unterstützt vor allem die Arbeit im Projektmanagement. Wie in einem anderen Artikel [4, S.170] zu lesen ist, helfen die automatisierten Prozesse in einem Ticketsystem einen Überblick über Arbeitspakete von Projekten inklusive ihrer Deadlines, Ressourcen, Zugehörigkeiten usw. zu erhalten. Dies trägt vor allem zur Strukturierung bei und erleichtert auch die Erstellung einer digitalisierten Dokumentation. Infolgedessen wird eine hohe Effizienz in den Arbeitsabläufen erzielt, indem transparent und zeitsparend gearbeitet wird. [Teammitglied 4]

WireFrames

WireFrames werden im Modellierungsprozess genutzt, um erste Grafische Benutzer Oberflächen zu modellieren und visuell darzustellen[5, S.30]. Nachdem wir unsere Funktionen festgelegt haben, haben wir angefangen unsere Seiten zu gestalten. Dafür haben wir das Tool Balsamiq verwendet welches gewählt wurde, weil es bereits Erfahrungen damit in der Gruppe gab. Für unser Projekt haben wir 4 Seiten erstellt und die Funktionen verteilt.

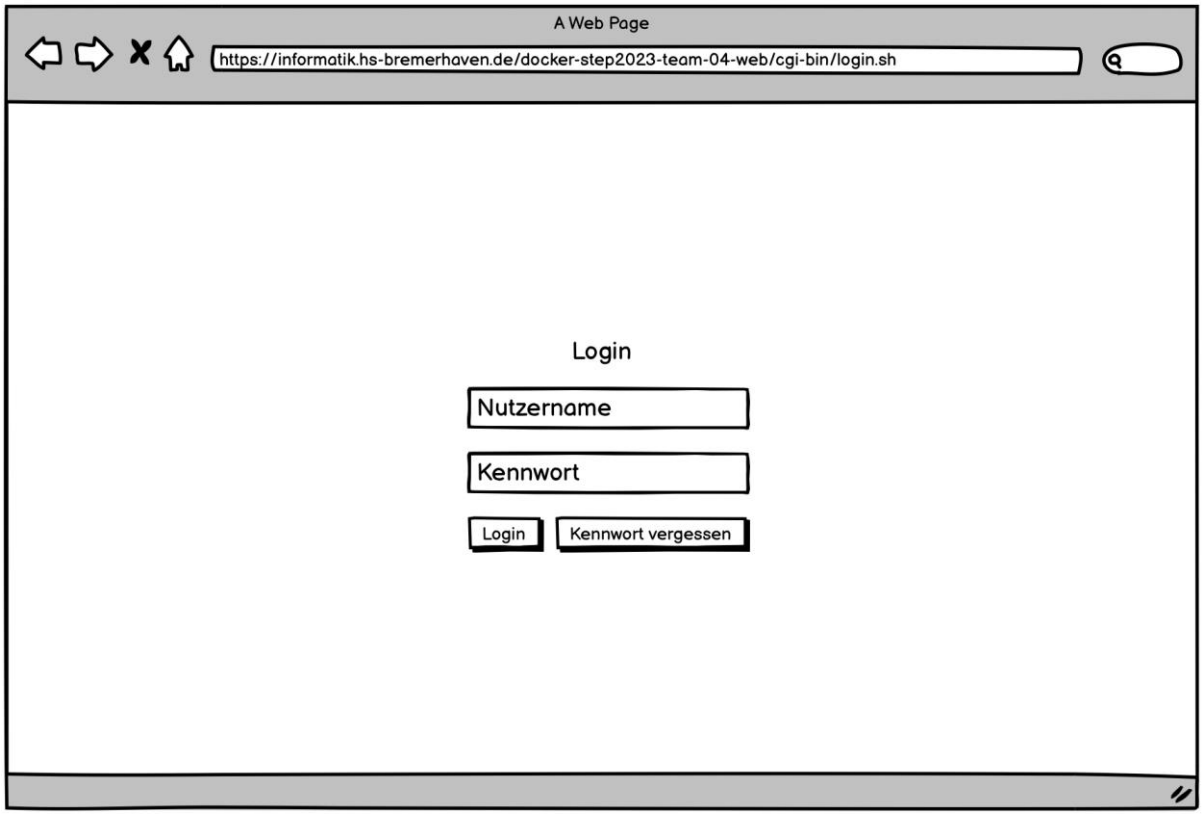


Abbildung 11 Wireframe "Login"

Für den Login haben wir einen sehr simplen Aufbau gewählt. Neben den Textfeldern für die Zugangsdaten und dem "Login" Button hatten wir ein "Kennwort vergessen" Button eingebaut. Dieser ist jedoch im Laufe des Prozesses entfernt wurden, weil die Funktionalität dahinter für den Prototypen nicht relevant war und wir uns auf die Kernfunktionen konzentrieren wollten.

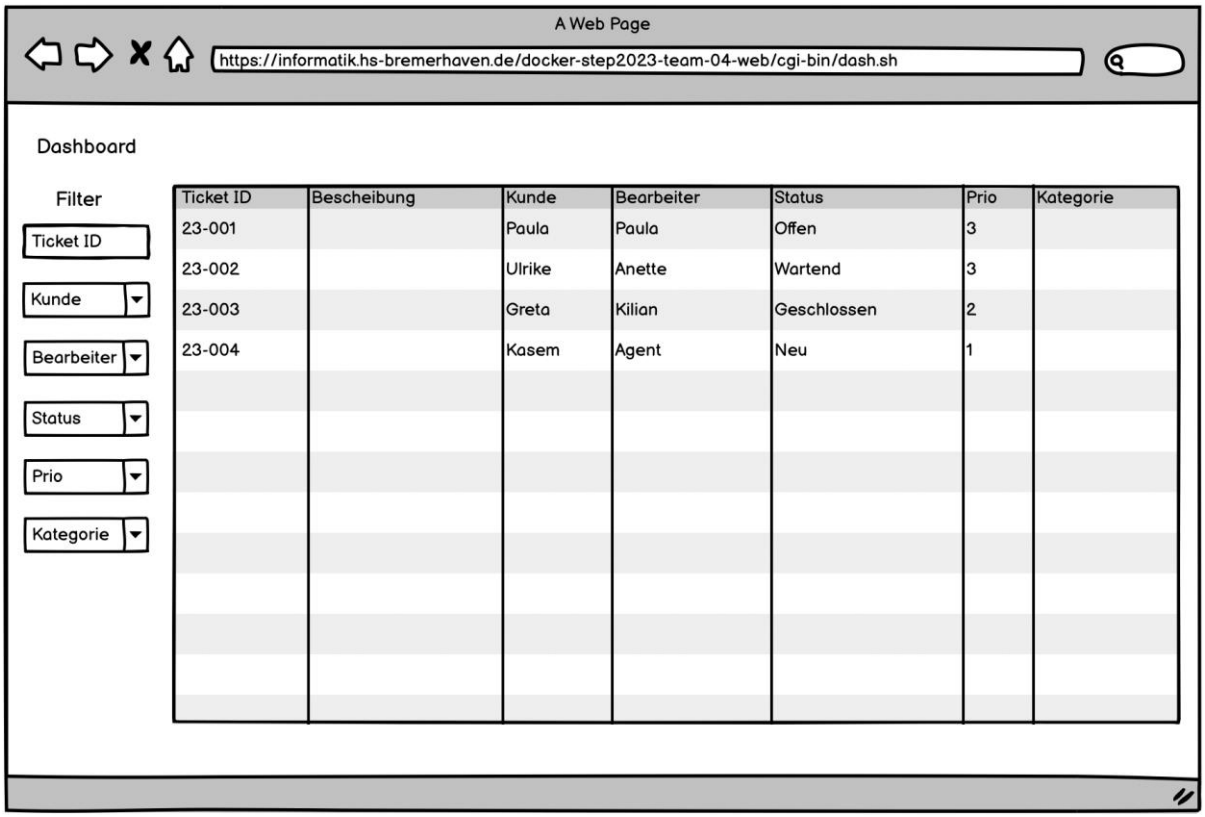


Abbildung 12 Wireframe "Übersichtsseite"

Das Dashboard stellt das Zentrum unseres Tickets Systems da. Es zeigt auf einem Blick welche Tickets da sind, welchen Status sie besitzen, wem Sie gehören und wer Sie bearbeitet. Zudem kommen noch Informationen wie eine kurze Beschreibung, die Kategorie und die Priorität. Von hier aus kann ein Bearbeiter auf alle Tickets zugreifen und ein Kunde neue Tickets erstellen oder seine vorhandenen einsehen. Beim Design haben Teammitglied 4 und ich, unsere Erfahrungen mit Ticketsystemen genutzt, um eine möglichst einfache und Intuitive Oberfläche zu gestalten.

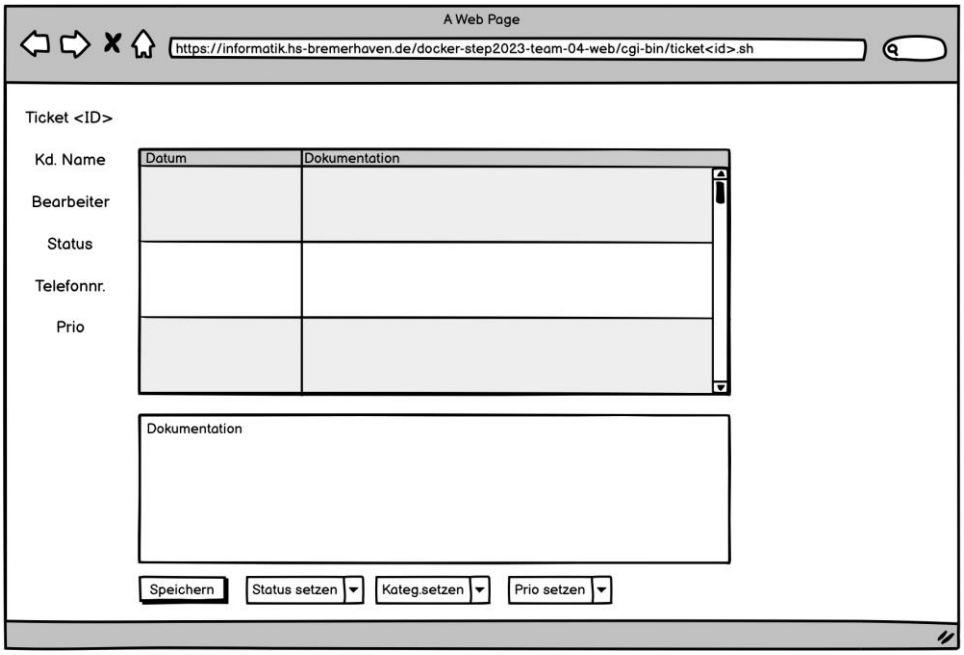


Abbildung 13 Wireframe "Ticketübersicht - Bearbeiter"

Auf der Bearbeitungsseite sind nochmal alle Informationen aufgeführt. Dazu kommen noch Kontaktinformationen in Form einer Telefonnummer und eine Bearbeitungshistorie, in der alle Handlungen zur Bearbeitung des Tickets dokumentiert werden können.

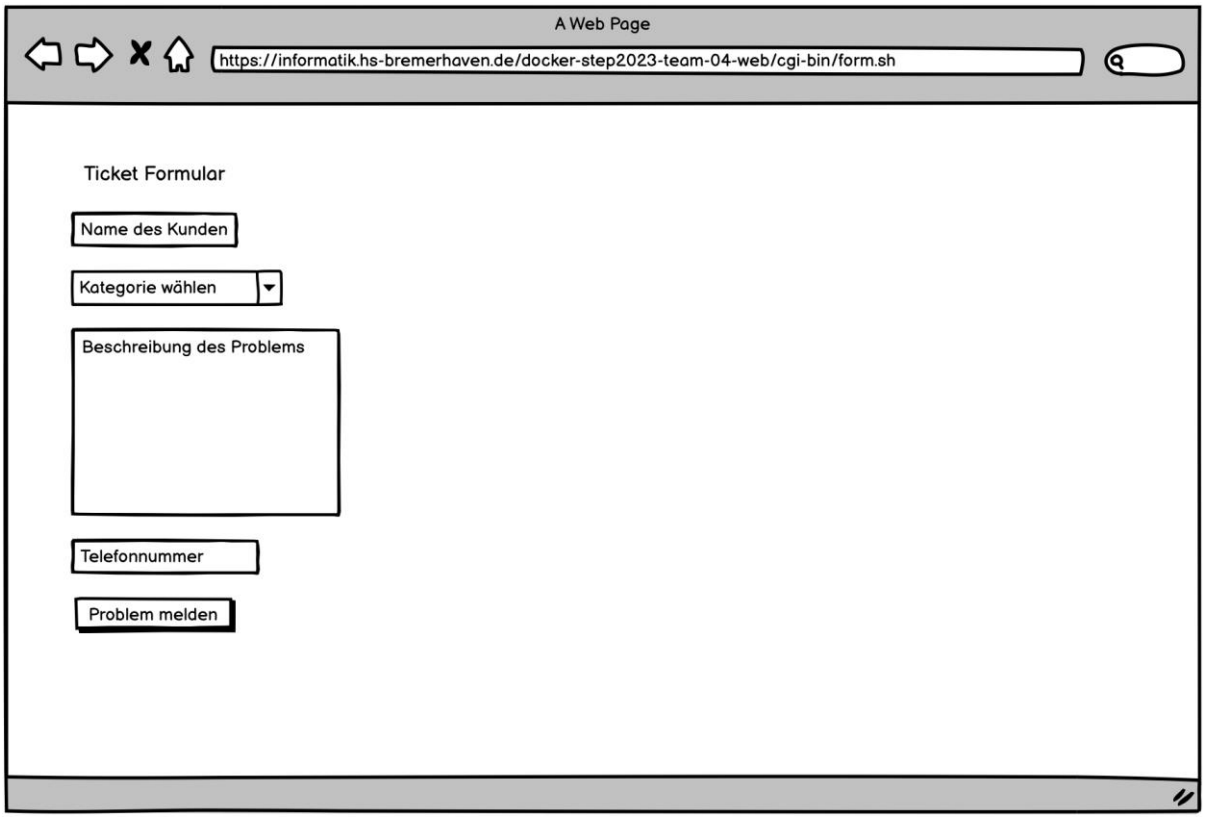


Abbildung 14 Wireframe "Ticket erstellen - Kunde"

Das Formular zur Erstellung von neuen Tickets beschränkt sich auf die wichtigsten Informationen. Es muss der Name des Kunden, die Kategorie des Auftrags, eine kurze Beschreibung und die Telefonnummer angegeben werden, bevor das Problem mit der Schaltfläche abgeschickt werden kann. [Teammitglied 3]

Fazit

Die Zusammenarbeit im SWE1-Projekt hat uns auf unterschiedlich Art und Weise bereichert.

Auf technischer Ebene haben uns Modelle eine Herangehensweise an größere Projekte gezeigt, die eben nicht mit einem oder zwei Skripten zu lösen sind. Wir konnten für unsere Projekt so ein konzeptionelles Gerüst bauen, das durch die unterschiedlichen Anwendungsfälle in der Implementierung sehr gut modularisiert werden konnte. Wir konnten uns durch die Modellerstellung auf ein Verständnis von Ticketsystemen einigen, damit jedes Teammitglied eine ähnliche Vorstellung von Issues-Tracking-Systemen hat. Die Bedeutung gleicher Vorstellungen sollte dabei nicht unterschätzt werden, denn viele Feinheiten und Abstimmungen in der Implementierung erledigen sich dadurch fast von selbst.

Auf menschlicher Ebene ist eine Erkenntnis, dass Formulierungen in Modellen, von denen man selbst dachte, sie seien maximal eindeutig, in der Regel doch nicht eindeutig waren, und man gut darüber diskutieren kann, um zu einer für uns klar verständlichen Formulierung zu kommen. So haben wir viele Möglichkeiten, Ticketsysteme zu modellieren betrachtet. Eine Frage, die dabei sehr lange diskutiert wurde, ist inwiefern unser Agent ein Mensch oder eine Maschine ist. Um uns die Implementierung einer zufälligen Priorisierung zu ersparen, haben wir uns dann für einen menschlichen Agenten entschieden. Mitunter auch, weil das einer wirklichen Priorisierung durch Projektmanagern deutlich näherkommt.

Außerdem ist eine Erkenntnis, dass wir in unserem Team unterschiedliche Stärken haben, die wir einzusetzen lernen mussten. So sind einige Teammitglieder eher theoriegetrieben, wären andere sich besonders gerne praktisch programmieren. Nicht zuletzt soll eine Erkenntnis sein, wie unendlich wichtig gemeinsame Teammeetings sind und dass einen Termin zu finden auch immer wieder Zeit in Anspruch nimmt, ebenso wie die zeitliche Dynamik in Projekten, dass es eben Phasen gibt, in denen Probleme auftauchen und mehr gearbeitet werden muss, ebenso wie Phasen, in denen es etwas stiller ist.[Teammitglied 2]

Abbildungsverzeichnis

Literaturverzeichnis

[1] Schürmann, Tim, Praxishandbuch OTRS, 2018, O'Reilly

[2] Rumpe, Bernhard, Modellierung mit UML, in Xpert.press, S.18, 2011. http://dx.doi.org/10.1007/978-3-642-22413-3

[3] Dörr, K.M. (2019), "Tue Gutes und rede richtig darüber – Eine Anleitung zur professionellen End-User-Kommunikation für IT-Organisationen" In: Aengenheyster, S., Dörr, K. (eds) Praxishandbuch IT-Kommunikation. Springer Gabler, Berlin, Heidelberg.

Doi: https://doi.org/10.1007/978-3-662-57965-7_8

[4] J.Berger, „Digital und kollaborativ: Arbeitsaufträge via Tickets in der internen Projektkommunikation", *Information – Wissenschaft & Praxis,* Bd. 73, Nr. 4, S. 167-171, 2022, doi: 10.1515/iwp-2022-2222.

[5] Berenbrink, V., Purucker, J. & Bahlinger, T. Die Bedeutung von Wireframes in der agilen Softwareentwicklung. HMD 50, 27–34 (2013). https://doi.org/10.1007/BF03340793