

Siemens Schweiz AG

# Probe IPA

Individuelle Praktische Arbeit M223

Olivia Pawlowitz  
10.11.2017

## 1 Inhalt

2	Informieren.....	4
2.1	Vorwort .....	4
2.2	Risiken .....	4
2.3	Externe Einflüsse .....	4
2.4	Falsche Zeiteinschätzung.....	4
2.5	Datenverlust .....	4
2.6	Krankheit/Unfall .....	4
2.7	Projektorganisation .....	5
2.8	Lehrbetrieb und Durchführungsort .....	5
2.9	Kandidat .....	5
2.10	Fachvorgesetzter .....	5
2.11	Aufgabenstellung.....	6
2.12	Deklarationen .....	9
2.13	Vorkenntnisse.....	9
2.13.1	Fachliche Vorkenntnisse .....	9
2.14	Projektmethode IPERKA .....	10
2.15	Begründung der Wahl von IPERKA .....	10
2.16	Zeitmanagement .....	11
2.17	Struktur.....	11
2.18	Meilensteine.....	13
2.19	Arbeitsjournale.....	14
2.20	Tagesablauf Mittwoch, 01.11.2017 (Tag 1) .....	14
2.20.1	Journal .....	14
2.21	Tagesablauf Donnerstag, 02.11.2017 (Tag 2) .....	15
2.21.1	Journal .....	15
2.22	Tagesablauf Freitag, 03.11.2017 (Tag 3) .....	16
2.22.1	Journal .....	16
2.23	Tagesablauf Mittwoch, 08.11.2017 (Tag 4) .....	16
2.23.1	Journal .....	16
2.24	Tagesablauf Freitag, 10.11.2017 (Tag 5) .....	17
2.24.1	Journal .....	17
2.25	Rechner.....	18
2.26	Kurzfassung .....	19

2.26.1	Ausgangssituation .....	19
2.26.2	Umsetzung.....	19
2.26.3	Ergebnis .....	19
3	Planung.....	20
3.1	Anforderungen .....	20
3.1.1	Backend .....	20
3.1.2	Frontend .....	20
3.2	Umsetzung.....	20
3.2.1	Versionsverwaltung.....	20
3.2.2	Subversion (SVN) .....	20
3.3	Use Case .....	21
3.4	Mockups .....	27
3.5	Benutzeroberfläche .....	28
3.5.1	Backend .....	28
3.6	Datenbank .....	29
3.7	Tabellendefinitionen .....	30
3.7.1	users .....	30
3.7.2	tickets .....	31
4	Entscheidung .....	32
4.1.1	Versionsverwaltung.....	32
4.1.2	Datenbank .....	32
4.1.3	Programmiersprache.....	32
5	Realisierung .....	33
5.1	Umsetzung.....	33
	Funktionsweise.....	33
5.2	Klassen.....	34
5.2.1	Ticket->Data .....	34
5.2.2	Login->User.....	34
5.2.3	Login -> DB_con.....	34
5.2.4	Ticket-> Database .....	34
6	Kontrolle .....	35
6.1	Testphasen .....	35
6.2	Fehlerklassifizierungen.....	35
6.3	Testfälle .....	35

7	Auswertung .....	41
7.1	Möglichkeiten zur Weiterentwicklung .....	41
7.2	Wie ich die Arbeit empfunden habe .....	41
7.3	Was ich das nächste Mal verbessern könnte .....	42
7.4	Glossar .....	43
7.5	Quellenverzeichnis .....	43
7.6	Abbildungsverzeichnis .....	<b>Fehler! Textmarke nicht definiert.</b>
Es konnten keine Einträge für ein Abbildungsverzeichnis gefunden werden. .... <b>Fehler! Textmarke nicht definiert.</b>		
8	Anhang.....	44
8.1	Codeausschnitte .....	44
8.1.1	Ticketsystem : index.php .....	44
8.1.2	Ticketsystem : dbconfig.php.....	49
8.1.3	Ticketsystem : classAction.php.....	49
8.1.4	Login: login.php .....	52
8.1.5	Login: registration.php .....	55
8.1.6	Login: include/classUser.php.....	58
8.1.7	Login: include/dbconfig.php.....	60

## **2 Informieren**

### **2.1 Vorwort**

Anhand dieser Probe IPA-Dokumentation wird die gesamte, individuelle Praxisarbeit festgehalten und dokumentiert. Alle projektrelevanten Informationen sowie der eigentliche Projektverlauf sind somit klar ersichtlich. Diese Dokumentation wurde im November 2017 von Olivia Pawlowitz im Rahmen des Moduls 223 zur Vorbereitung auf den Lehrabschluss als Informatiker, Schwerpunkt Applikationsentwicklung, verfasst.

### **2.2 Risiken**

### **2.3 Externe Einflüsse**

Die Projektarbeit ist teilweise abhängig von externen Einflüssen, welche die planmässige Durchführung durcheinander bringen könnten. Zu den externen Einflüssen gehören beispielsweise Server, Netzwerk, Internet Zugang, usw.

### **2.4 Falsche Zeiteinschätzung**

Meistens ist es so, dass der Zeitplan nicht auf das Detail genau eingehalten werden kann. Damit das Projekt trotzdem planungsgemäss erledigt werden kann, wurden Zeitreserven in die Planung mit einbezogen.

### **2.5 Datenverlust**

Die Daten werden regelmässig gesichert, um den Verlust von relevanten Projektdaten vorzubeugen. Es besteht jedoch ein geringes Risiko, dass Daten während der Projektarbeit verloren gehen könnten.

### **2.6 Krankheit/Unfall**

Die Projektarbeit kann aus gesundheitlichen Gründen jederzeit unterbrochen werden. Der Endtermin des Projekts verschiebt sich entsprechend der verlorenen Zeit.

## 2.7 Projektorganisation

## 2.8 Lehrbetrieb und Durchführungsort

Firma Lehrbetriebsverbund Siemens Schweiz AG

Adresse Freilagerstrasse 40, 8047 Zürich

## 2.9 Kandidat

Name Frau Olivia Pawlowitz

Email olivia@pawlowitz.com

Tel. Mobil 079 551 69 50

## 2.10 Fachvorgesetzter

Name Angelo Aloise

Email angelo.aloise@siemens.com

## 2.11 Aufgabenstellung

### Projekt

Projekttitel: Erstellen einer Multi User Applikation

Ticketing System

Im Auftrag von : Remo Steinmann Modul 223

Auftragnehmer: Olivia Pawlowitz

Starttermin: 01. November 2017

Geplante Projektzeit: 5 Arbeitstage(40h) davon ca. 3 Tage Entwicklung(24h)

Erstellen einer Multi User Applikation, beinhaltend Frontend, Backend, Anbindung an relationaler Datenbank.

Wieso ein Ticket-System?

Es gibt verschiedene Varianten, Kundenanfragen in einem Unternehmen zu bearbeiten. Ticket-Systeme grenzen sich von anderen Methoden ab, sie speichern einzelne Anfragen und bieten dem User eine geniale Übersicht über alle Anfragen. Die Tickets können kategorisiert und unzählige Male abgerufen werden.

Programmiersprache : PHP, JavaScript

IDE: Eclipse

DB: MySQL

Framework: Bootstrap

Library: JQuery

### Soll Zustand

Der User kann sich registrieren und sich mit den Anmeldedaten einloggen. Es wird ein neues Fenster geöffnet und der Benutzer sieht die Hauptseite. Dort besteht für ihn die Möglichkeit ein neues Ticket zu erstellen oder alle Tickets anzusehen. Der Benutzer kann ein Ticket ansehen und löschen.

Das Projekt wird mit PHP und JavaScript objektorientiert realisiert. Die Daten werden in einer MySQL Datenbank gespeichert. Es werden 5 Tabellen angelegt.

- Kategorie
- Abteilung
- Priorität
- Tickets
- User

Beim Login bzw. registrieren werde ich, zum übertragen der Daten, Ajax verwenden. Das Passwort wird jeweils verschlüsselt in die DB eingetragen.

Nachdem der User sich angemeldet bzw. registriert hat, wird dieser auf die Hauptseite weitergeleitet. Dort werden ihm zwei Möglichkeiten angeboten. Entweder er erstellt ein Ticket, er sieht sich alle Tickets an oder er loggt sich wieder aus.

### **Fallbeispiel 1: Ticket erstellen**

Der User möchte ein neues Ticket erstellen. Er hat nun die Möglichkeit sein Ticket zu erstellen und muss den jeweiligen Bereich für das Problem auswählen, die Wichtigkeit und die Abteilung in der er sich befindet. Der User gibt dem Ticket ein Titel und fügt ausserdem eine Beschreibung hinzu. Nun kann das Ticket veröffentlicht werden.

### **Spezifikationen**

Es dürfen eine maximale Anzahl Wörter in der Beschreibung enthalten sein. Diese Anzahl reduziert sich auf 50 Wörter. Ein Upload von Bildern sprich Screenshots ist möglich und auch erwünscht. Die Anzahl der Bilder, welche hochgeladen werden können ist unbegrenzt.

### **Fallbeispiel 2: Ticket ansehen**

Der Benutzer möchte nun alle Tickets ansehen. Es erscheint eine Tabelle mit allen erstellten Tickets. In dieser Tabelle sieht der User, wer Ticket xyz erstellt hat, welche Wichtigkeit es hat, zu welcher Abteilung es gehört und den Titel. Er kann die Tickets löschen und bearbeiten.

In der Ticketview ist es dem User möglich zu sehen, wann ein Ticket erstellt wurde: Datum & Uhrzeit.

### **Fallbeispiel 3: User abmelden**

Der User möchte sich abmelden und klickt auf den Button „Logout“.



## Anforderungen an die Applikation

-Folgende Anforderungen müssen erfüllt werden:

-Multi-User Applikation

-Relationale Datenbank

-Objektorientiert Programmieren

-Transaktionen

-Mehrere Clients müssen gleichzeitig auf den gleichen Datenbestand zugreifen

## Zusätzliche Kriterien

-194 Plausibilisierung der Benutzer-Eingaben

-166 Coding Style – lesbarer Code

-163 Design Doku

-123 Kommentar im Code

-164 Fehlerbehandlung

-130 Vollständiges ERM bzw. Datenmodell

-125 Gliederung des Programms

## Warum dieses Projekt zum Modul Multiuser-Applikationen passt:

Alle Anforderungen, welche Anfangs Modul gestellt wurden, können erfüllt werden.

## Herausforderung: PHP objektorientiert

Da ich bis zu diesem Zeitpunkt erst einmal PHP Objektorientiert programmiert habe, wird dies eine Hürde sein, in kürzester Zeit die Aufgabenstellung so genau wie möglich umzusetzen.

## 2.12 Deklarationen

## 2.13 Vorkenntnisse

In meiner bald 4-jährigen Lehre als Applikationsentwickler konnte ich mir die in diesem Abschnitt beschriebenen Vorkenntnisse erlernen, welche für das Projekt relevant sind.

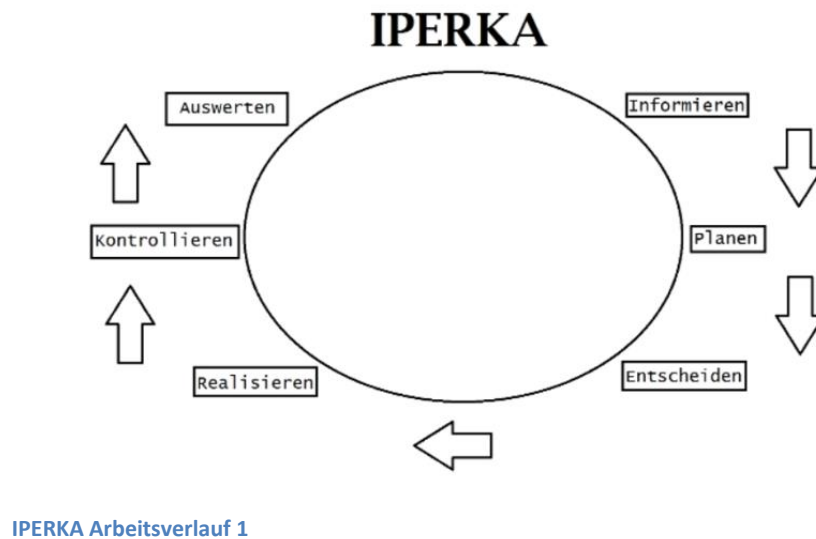
Projektrelevante Vorkenntnisse werden stichwortartig festgehalten:

### 2.13.1 Fachliche Vorkenntnisse

Thema	Kenntnisse	Bemerkungen
HTML	sehr gute Kenntnisse	Module absolviert und Selbststudium
CSS	sehr gute Kenntnisse	Selbststudium
JavaScript	gute Kenntnisse	Selbststudium
SQL	sehr gute Kenntnisse	Erfahrung aus Praxis, Modul
PHP	gute Kenntnisse	Selbststudium
Java	Gute Kenntnisse	1 Jahr Erfahrung in der IT-Abteilung

Fachliche Vorkenntnisse 1

## 2.14 Projektmethode IPERKA



In der „Abbildung 1: IPERKA Arbeitsablauf“ sind die einzelnen Projektphasen ersichtlich, welche mit IPERKA unterteilt werden. Das Ziel von IPERKA ist es, eine klare Struktur zu schaffen nach der Schrittweise gearbeitet wird. IPERKA enthält dabei alle relevanten Phasen eines Projekts.

## 2.15 Begründung der Wahl von IPERKA

Die Projektmanagement-Methode IPERKA haben wir schon am ersten Tag als Informatiker im Basislehrgang kennengelernt und stets bei jedem Projekt eingesetzt. Es bietet meiner Meinung nach einige Vorteile. Das Projekt lässt sich so in übersichtliche Teilschritte gliedern und es ist deshalb immer ersichtlich, in welchem Teilschritt man sich befindet.

Es gibt zahlreiche Projektmanagement-Methoden, die vielleicht geeigneter für die Entwicklung einer Applikation wären, doch für so ein kleines Projekt, wie die Probe IPA, ist IPERKA die einfachste und übersichtlichste Methode. Zudem haben andere Methoden den Nachteil, dass sie zu viel Zeit für die Planung beanspruchen, was für die Probe IPA nicht sehr passend wäre.

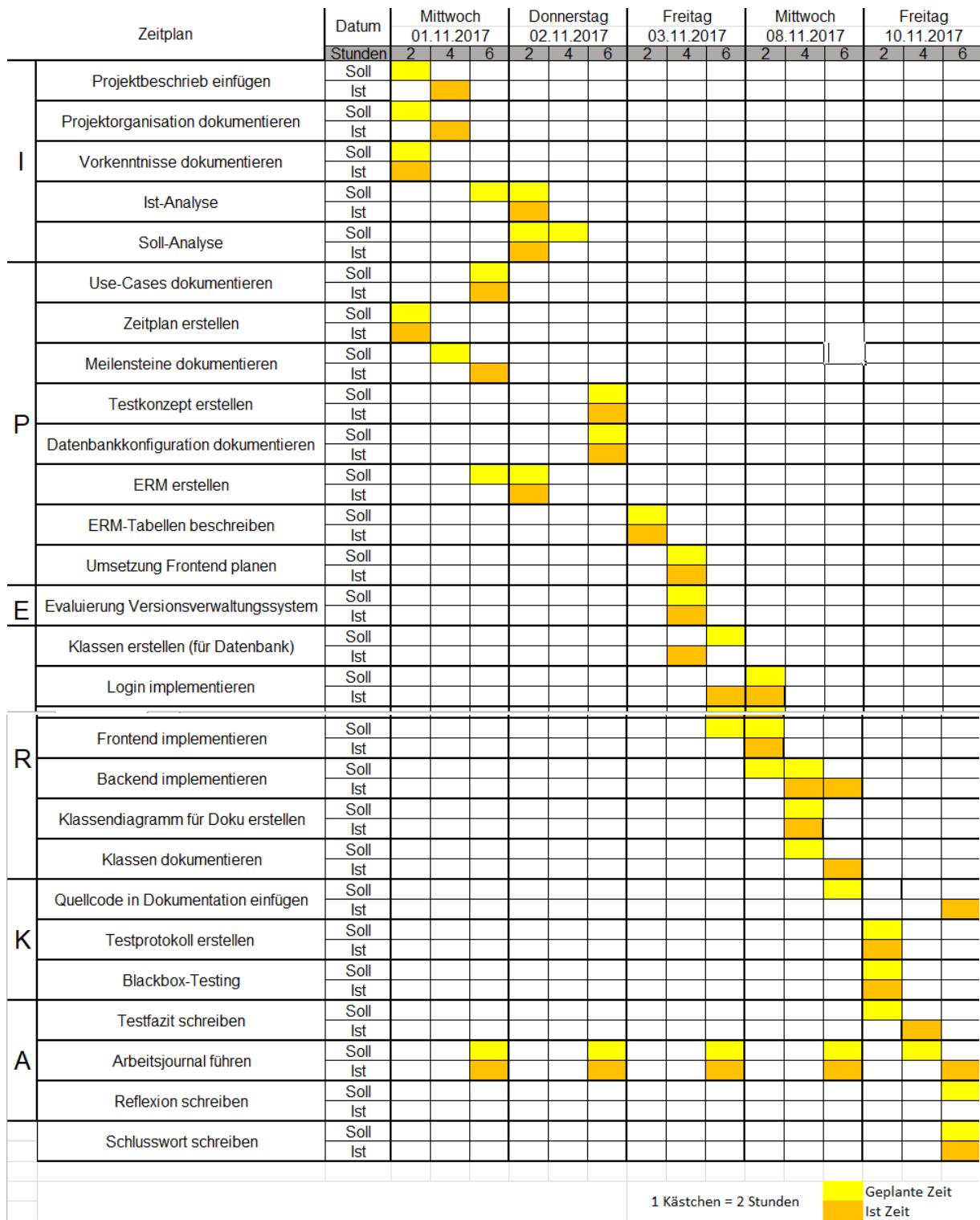
## 2.16 Zeitmanagement

### 2.17 Struktur

Es macht Sinn, den Zeitplan nach der IPERKA Methode zu strukturieren, da ich diese Projektmanagement-Methode gewählt habe. Der Zeitplan ist dabei in zwei verschiedene Bereiche gegliedert, nämlich in Hauptarbeitsschritte und wiederkehrende Arbeitsschritte. Es wird in Form eines GANTT-Diagramms dargestellt.

#### **Begründung Wahl 2-Stunden-Blöcke**

Ich habe mich für 2-Stunden-Blöcke entschieden, da die meisten Aufgaben so viel Zeit in Anspruch nehmen. 1-Stunden-Blöcke wären für ich viel zu unübersichtlich und ich würde meine SOLL Zeit ziemlich unterschätzen, sodass der IST Plan zu sehr abweichen würde.



Zeitplan 1

## 2.18 Meilensteine

Meilenstein	Beschreibung	Datum
<b>Informieren</b> SOLL Zeitplan	Der Zeitplan für die SOLL Stunden ist fertig definiert.	01.11.2017
<b>Informieren</b> Use Cases	Use Cases sind geschrieben und in die Dokumentation implementiert	01.11.2017
<b>Planen</b> ERM	Klassendiagramm erstellt	02.11.2017
<b>Entscheiden</b> Klassen erstellt für DB Login implementiert Frontend implementiert	Entscheidungen werden gefällt bezüglich Versionierungsverwaltung, Umsetzung	02.11.2017
<b>Realisieren</b> Klassen dokumentieren	Frontend und Backend	03.11.2017
<b>Kontrollieren</b> Testfälle, Testphasen	Applikation wird getestet und Testfälle sowie Testphasen werden dokumentiert	08.11.2017
<b>Auswerten</b> Weiterentwicklung, Verbesserungsvorschläge	Was ist gut gelaufen, was ist schlecht gelaufen, was kann verbessert werden?	10.11.2017

### Meilensteine 1

## 2.19 Arbeitsjournale

### 2.20 Tagesablauf Mittwoch, 01.11.2017 (Tag 1)

Zeit	Erledigte Arbeiten	Erfüllt
8.15 – 10.00	Zeitplan	Ja
12.30 – 13.00	Grundgerüst Doku	Ja
13.00 – 14.00	Use Case dokumentieren	Ja
14.00 – 14.30	Meilensteine	Ja
14.30 – 16.30	ERM erstellen (anfangen)	Nein

#### 2.20.1 Journal

Ich bin pünktlich um 8.15 in der Berufsbildung eingetroffen, leider konnte ich keinen geeigneten Laptop mitnehmen. Freundlicherweise habe ich einen von der Berufsbildung erhalten.

Als erstes habe ich begonnen den Zeitplan zu erstellen, die erste Version musste um 10.00 Uhr stehen. Da ich noch kein Office auf dem Laptop hatte, habe ich den Zeitplan mithilfe von Excel Online erstellt. Um ca. 9:00 Uhr bekam ich von Remo ein Office Paket, somit konnte ich meinen Zeitplan in Excel übernehmen.

Ich war schon ziemlich zufrieden mit meiner ersten Version des Zeitplans, jedoch sind mir noch ziemlich viele Sachen eingefallen. Noch vor dem Mittagessen habe ich mich an das Grundgerüst der Dokumentation gesetzt. Ich habe begonnen die verschiedenen Projektphasen von IPERAK zu implementieren.

Um Use Case zu erstellen habe ich Umler verwendet. Das hat ganz gut geklappt jedoch müssen diese noch weiter ausgebaut werden in der Beschreibung. Dies werde ich morgen abschliessen. Ausserdem konnte ich noch Meilensteine definieren. Diese habe ich aus meinem Zeitplan entnommen. Leider hat es heute nicht mehr gereicht, um das ERM fertig zu stellen.

Ich hoffe ich kann dies morgen nachholen, ohne das mein Zeitplan sich verzögert.

## 2.21 Tagesablauf Donnerstag, 02.11.2017 (Tag 2)

Zeit	Erledigte Arbeiten	Erfüllt
8.15 – 10.00	Ist-Soll Analyse	Ja
10.00 – 11.30	ERM	Ja
12.30 – 13.30	Testkonzept	Ja
14.00 – 16.30	Datenbankkonfiguration dokumentieren	Ja

### 2.21.1 Journal

Heute konnte ich pünktlich um 8.15 mit meiner Arbeit beginnen. Als erstes habe ich meinen Zeitplan angeschaut, um nachzusehen, welche Aufgaben ich für den heutigen Tag definiert habe.

Von 8.15-10.00 Uhr habe ich mich an die Ist-Soll Analyse gesetzt. Ich bin gut vorangekommen so dass ich später dann mit dem ERM fortfahren konnte. Bis zur Mittagspause war mein Aufgabenbereich also gut gedeckt. Ich bin schneller vorangekommen als gedacht und konnte die fehlende Zeit von gestern wieder aufholen. Den Nachmittag habe ich damit verbracht mein Testkonzept zu schreiben, ich habe mir alle Testphasen notiert und überlegt, was alles zu testen ist. Mein Testkonzept ist nun vorbereitet und ich kann am Schluss nur noch meine Testdaten einfügen.

Um 13 Uhr hatte ich mein erstes Expertengespräch und ich habe meinem Experten Remo den Zeitplan und mein Arbeitsjournal von gestern gezeigt. Ausserdem habe ich meinen aktuellen Standpunkt erläutert.

Nachdem alle mit dem Expertengespräch durch waren, haben wir einen Input erhalten, wie wir das erste Expertengespräch verbessern könnten. Das fand ich sehr gut und hilfreich. Es muss auf sehr viele kleine Details geachtet werden, damit das Gespräch erfolgreich verläuft.

Morgen werde ich mit der Realisierung beginnen.



## 2.22 Tagesablauf Freitag, 03.11.2017 (Tag 3)

Zeit	Erledigte Arbeiten	Erfüllt
8.15 – 9.00	Frontend planen	Ja
10.00 – 11.30	ERM Tabellen beschreiben	Ja
12.30 – 13.30	Klassen erstellen für DB	Ja
14.00 – 16.30	Frontend implementieren	Ja

### 2.22.1 Journal

Heute habe ich pünktlich um 8.15 mit meiner Arbeit begonnen. Die ersten Stunden habe ich damit verbracht, die Umsetzung des Frontend zu planen und meine Tabellen zu beschreiben, allerdings habe ich diese noch nicht so ausführlich beschrieben wie ich es eigentlich wollte, das werde ich gegen Schluss noch nachholen. Ich habe simple Tabellen erstellen, anhand von diesen Tabellen konnte ich nochmals die Datentypen und die kurze Beschreibung aufzeigen. Nach Beschreiben der Tabellen habe ich sie auch direkt erstellt.

Ich erstellte das File dbconfig.php und konfigurierte die Datenbankverbindung.

Gegen Nachmittag habe ich die Datenbankverbindung hergestellt und angefangen das Frontend zu realisieren. Mit der Implementation bin ich heute schon ziemlich weit gekommen, ich habe mir mein Layout vorgezeichnet, so wie ich es haben möchte und versucht nach zu bauen. Zuerst hatte ich Mühe, weil ich ein paar Flüchtigkeitsfehler gemacht habe und etwas unkonzentriert war. Jedoch habe ich nach der Pause wieder Fuss gefasst und kam gut voran. Ich bin schon sehr zufrieden mit meinem Frontend, jedoch muss ich nächsten Mittwoch zügig arbeiten, denn sonst wird es knapp.

## 2.23 Tagesablauf Mittwoch, 08.11.2017 (Tag 4)

Zeit	Erledigte Arbeiten	Erfüllt
8.15 – 9.00	Frontend implementieren	Ja
09.00 – 11.30	Login implementieren	Ja
12.30 – 14.30	Backend implementieren	Ja
15.40 – 16.15	Klassen dokumentieren	Ja

### 2.23.1 Journal

Heute konnte ich pünktlich um 8.15 Uhr starten. Ich habe am Freitag angefangen das Frontend zu implementieren. Um ca. 9.00 Uhr hatte ich das Frontend soweit fertig. Später dann habe ich begonnen das Login zu implementieren, anfangs hatte ich ein paar Schwierigkeiten mit der Datenbank Verbindung aber schlussendlich hat sich herausgestellt, das waren nur Flüchtigkeitsfehler.

Gegen Mittag hatte ich mein zweites Expertengespräch, ich habe ihm meinen aktuellen Stand vermittelt und ihm Ausschnitte meines Codes gezeigt. Ich habe wichtige Rückmeldungen erhalten, die mir so gar nie aufgefallen wären. Des Weiteren hatte ich die Möglichkeit Fragen zu stellen. Es war hilfreich zu hören, was ich noch besser machen kann.

Das Wichtige war heute, dass ich mit dem Backend soweit abschliessen konnte, denn ich habe für den Freitag keine Zeit mehr um weiter zu implementieren. Ich bin sehr weit gekommen, jedoch habe ich ein Problem und zwar kann ich nur immer ein Ticket erfassen. Ich habe im Internet nach Lösungen gesucht, leider bin ich zu keinem plausiblen Ergebnis gekommen. Ich hoffe, dass ich am

Freitag eine Lösung finde, obwohl ich dies eigentlich nicht zeitlich geplant habe. Trotzdem ist es mir wichtig, dass mein Projekt funktioniert.

## 2.24 Tagesablauf Freitag, 10.11.2017 (Tag 5)

Zeit	Erledigte Arbeiten	Erfüllt
8.15 – 10.00	Testprotokoll ausfüllen	Ja
10.00 – 11.15	Fehlerbehebung DB	Ja
12.20 – 13.00	Fehlerbehebung DB	Ja
13.00 – 13.30	Kurzfassung schreiben	Ja
13.30 – 15.00	Code in Doku implementieren	Ja
15.00 -16.30	Quellverzeichnis & Arbeitsjournal	Ja

### 2.24.1 Journal

Um 8.15 Uhr habe ich begonnen mit dem Ausfüllen des Testprotokolls. Ich habe meine Applikation mittels Blackbox Testing getestet. Hier ist mir als allererstes aufgefallen, dass ich am Mittwoch dieses Problem hatte, dass ich immer nur ein Ticket erfassen kann. Ich habe das Testing abgeschlossen und mich an das Problem begeben. Nach unzähligen Versuchen, konnte ich noch immer keine Lösung für mein Problem finden. Ich habe meinen Kollegen Toshiki um Rat gefragt und er meinte, dies könnte im Zusammenhang mit der Datenbank liegen. Also habe ich meine Tabellen überprüft und gesehen, dass ich meine id nicht als Auto\_Increment angekreuzt hatte und es funktionierte! Endlich! Ich weiss nicht, wieso ich nicht eher darauf gekommen bin. Jedenfalls bin ich nun froh, dass es geklappt hat. Es wäre sehr blöd gewesen, wenn ich nur ein Ticket erfassen könnte.

Die Kurzfassung und das Implementieren des Codes habe ich mir für das Ende aufgehoben. Schritt für Schritt habe ich den Code in meine Dokumentation kopiert.

Der Tag verlief ein bisschen stressig, nächstes Mal werde ich auf jeden Fall noch einen Puffer einbauen, falls etwas nicht ganz nach Plan läuft oder ich wie dieses Mal ein Problem beheben muss.

## 2.25 Rechner

Die Web-Applikation wird an einem Standard LocalClient der Siemens Schweiz AG entwickelt.

Technische Daten

<b>Hersteller</b>	<b>Lenovo</b>
<b>Modell</b>	L570
<b>CPU</b>	Intel Core i5-7300U
<b>Arbeitsspeicher</b>	8.00 GB
<b>Betriebssystem</b>	Windows 10 Pro

Rechner 1

## Lokaler Entwicklungsserver

Als lokalen Entwicklungsserver verwende ich XAMPP.

<b>XAMPP 3.2.2</b>
<b>Apache</b>
<b>PHP</b>
<b>MySQL</b>

Entwicklungsserver 1

## Entwicklungswerkzeug

Als Entwicklungswerkzeug wird die integrierte Entwicklungsumgebung (IDE) Eclipse eingesetzt und bietet zahlreiche Funktionen für die Entwicklung von PHP-Applikationen.

## 2.26 Kurzfassung

### 2.26.1 Ausgangssituation

Ich habe im Rahmen der Probe IPA ein Ticket System erstellt. Mit einem Ticket System werden Kundenanfragen zuverlässig bearbeitet. Jede Anfrage wird einer spezifischen Nummer zugeordnet und so abgearbeitet. Meistens werden Ticket Systeme verwendet, um Bug Reports zu beheben. Der Tester erstellt ein Ticket und meldet den Fehler, der Administrator bearbeitet das Ticket und setzt dieses dann auf „done“.

### 2.26.2 Umsetzung

Ich habe das Projekt mittels der Projektmanagement-methode IPERKA umgesetzt. Anhand der UseCases wird auf die einzelnen Funktionen konkret eingegangen. Das Ticket System beinhaltet ein Login und ein Registrierungsformular. Diese sind zur Benutzer Authentifizierung gedacht. Das Ticket System beinhaltet ein Formular und eine Tabelle. Im Formular gibt der User seine Ticket Details ein, wie Titel, Abteilung, Wichtigkeit, Kategorie und Beschreibung. Dem User steht die Möglichkeit zu Verfügung die Tickets nach Belieben zu bearbeiten und zu löschen.

### 2.26.3 Ergebnis

Meine Webapplikation welche als mein Projekt fungiert, ist einsetzbar und funktioniert.

## 3 Planung

### 3.1 Anforderungen

Die Anforderungen an die Applikation werden anhand von Kriterien definiert. Die Musskriterien müssen dabei zwingend erfüllt werden.

#### 3.1.1 Backend

Kriterium	MUSS	KANN
Login	x	
Logout		x
Registrieren	x	
Ticket erstellen	x	
Ticket bearbeiten	x	
Ticket löschen	x	
Bilder uploaden		x

#### Backend 1

#### 3.1.2 Frontend

Für das Backend wurden folgende Kriterien definiert:

Kriterium	MUSS	KANN
Login Fenster wird dargestellt	x	
Ticket Tabelle wird abgebildet	x	
Delete Funktion	x	
Update Funktion	x	

#### Frontend 1

## 3.2 Umsetzung

### 3.2.1 Versionsverwaltung

Die Wahl des Sicherungssystems in der Siemens Schweiz AG ist generell dem Entwickler überlassen, da es keine Richtlinien dafür gibt. In gewissen Abteilungen haben sich aber gewisse Standards etabliert, bei den einen ist es SVN und bei den anderen ist es Git. In diesem Abschnitt werden die Funktionalität und der Unterschied zwischen diesen beiden Varianten erläutert.

#### 3.2.2 Subversion (SVN)

Subversion arbeitet mit einem sogenannten Repository, in welchem die Änderungen gespeichert werden und einer „Working Copy“, die den aktuellen Stand enthält, in welcher der Entwickler seine Änderungen durchführt. Es werden dabei, aber nur die Unterschiede zu bereits vorhandenen Ständen übertragen. Dabei ist die Verbindung zum Repository erforderlich um Aktionen durchzuführen. Die Versionierung erfolgt dabei in Form einer Revisionszählung. Bei jeder Änderung wird die Revision um eins hochgezählt.

## Git

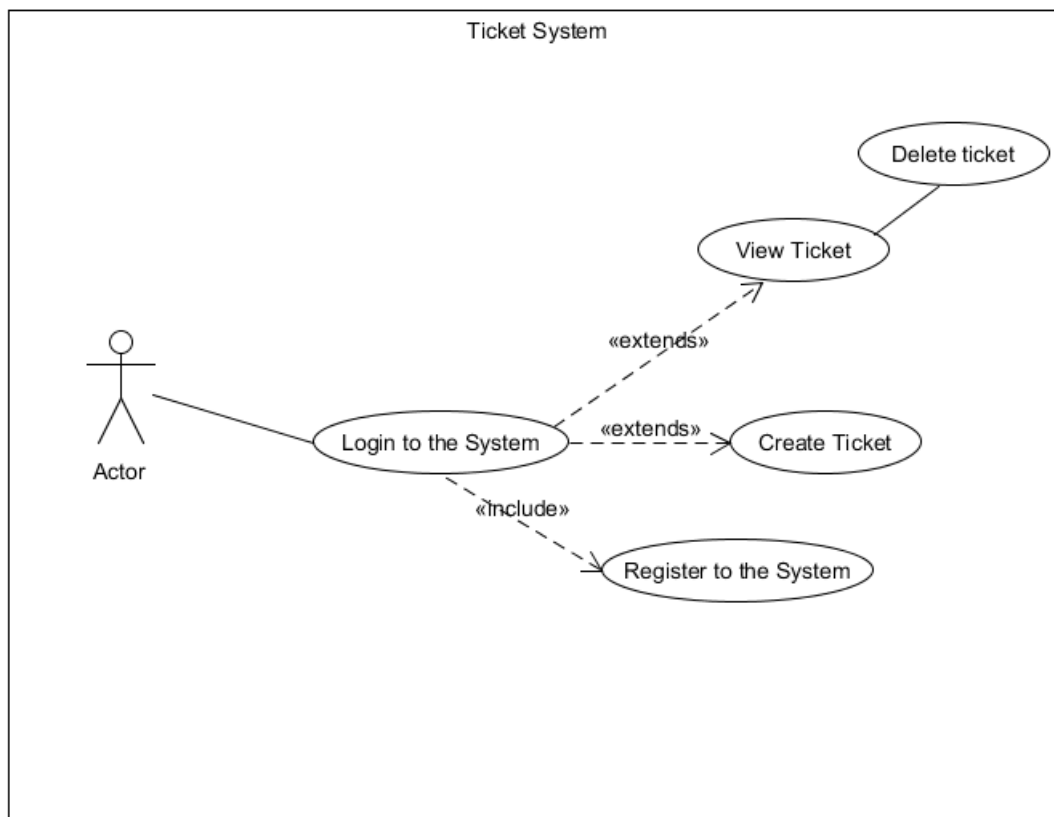
Git hat viele Ähnlichkeiten zu Subversion, doch es unterscheidet sich gänzlich bei der Funktionsweise. Wie auch unter Subversion arbeitet man hier mit einem sogenannten Repository. Es gibt jedoch keine zentrale Stelle, in welcher alle Änderungen gespeichert werden. Jeder Benutzer besitzt eine lokale Kopie des gesamten Repositories. Die meisten Aktionen können so lokal und ohne Netzwerkzugriff ausgeführt werden. Bei Git ist das Repository gleichzeitig auch die Working Copy.

## Unterschiede

Die wichtigsten Unterschiede im Überblick:

- Git ist schneller als Subversion
- Subversion lässt sich einfacher bedienen
- Unter Subversion ist die Verbindung zum Repository zwingend notwendig
- Git speichert bei einer Änderung das ganze Dateisystem ab, Subversion hingegen nur die Änderungen

## 3.3 Use Case



Use Case 1

Use Case	
<b>Use Case</b>	User anmelden
<b>Beschreibung</b>	Der User möchte sich einloggen
<b>Ziel / Ergebnis</b>	Am Ende sollte der User die Möglichkeit haben, Zugriff auf das Ticket System zu haben.
<b>Kategorie</b>	primär (notwendig u. Häufig verwendet)
<b>Vorbedingungen</b>	Die einzige Vorbedingung wäre, dass der User überhaupt die Möglichkeit hat sich zu registrieren
<b>Nachbedingungen</b>	<p>Erfolg: Wenn der User erfolgreich seinen Benutzernamen und sein Passwort eingegeben hat.</p> <p>Fehl Schlag: Der Benutzer kann sich nicht einloggen. (Benutzername oder Passwort falsch)</p>
<b>Invarianten</b>	Die Funktion kann für alle definierten Personen erstellt werden.
<b>Akteure</b>	Der User, der sich einloggen möchte ist der Akteur
<b>Auslösendes Ereignis</b>	UseCase wird initiiert, wenn das Programm gestartet wird und der User sich erfolgreich eingeloggt hat.
<b>Ablaufbeschreibung</b>	<p><b>Beschreibung des Standardfalls</b></p> <ol style="list-style-type: none"> <li>1. User startet das Programm</li> <li>2. User klickt in das Textfeld "Username"</li> <li>3. User klickt in das Textfeld „Passwort“</li> <li>4. User gibt Userame und Passwort ein</li> <li>5. User klickt auf „login“</li> <li>6. User wird weitergeleitet nach erfolgreichem Login</li> </ol> <p><b>Alternativen des Standardfalls</b></p> <p>6.1 Anstelle von „Login“ klickt der User auf „Registrieren“ falls dieser sich noch nicht registriert hat.</p>

---

Use Case	
Use Case	User registrieren
Beschreibung	Der User möchte sich registrieren
Ziel / Ergebnis	Am Ende sollte der User die Möglichkeit haben, sich in das System einzuloggen
Kategorie	primär (notwendig u. Häufig verwendet)
Vorbedingungen	Die einzige Vorbedingung wäre, dass der User überhaupt die Möglichkeit hat das Fenster „Registrieren“ zu öffnen
Nachbedingungen	<p>Erfolg: Wenn der User erfolgreich seinen Benutzernamen, sein Passwort, seinen Namen, und seine Email Adresse eingegeben hat.</p> <p>Fehl Schlag: Der Benutzer kann sich nicht einloggen, da seine Daten nicht stimmen. Er hat zum Beispiel eine falsche Emailadresse eingegeben oder der Benutzer ist schon vorhanden.</p>
Invarianten	Die Funktion kann für alle definierten Personen erstellt werden.
Akteure	Der User, der sich registrieren möchte ist der Akteur
Auslösendes Ereignis	UseCase wird initiiert, wenn das Programm gestartet wird und der User sich erfolgreich registriert hat.
Ablaufbeschreibung	<p><b>Beschreibung des Standardfalls</b></p> <ol style="list-style-type: none"><li>1. User startet das Programm</li><li>2. User klickt in das Textfeld „Name“</li><li>3. User klickt in das Textfeld „Username“</li><li>User klickt in das Textfeld „Email“</li><li>User klickt in das Textfeld „Passwort“</li><li>4. User gibt seine Daten ein</li><li>5. User klickt auf „register“</li></ol>

---



---

**Use Case**

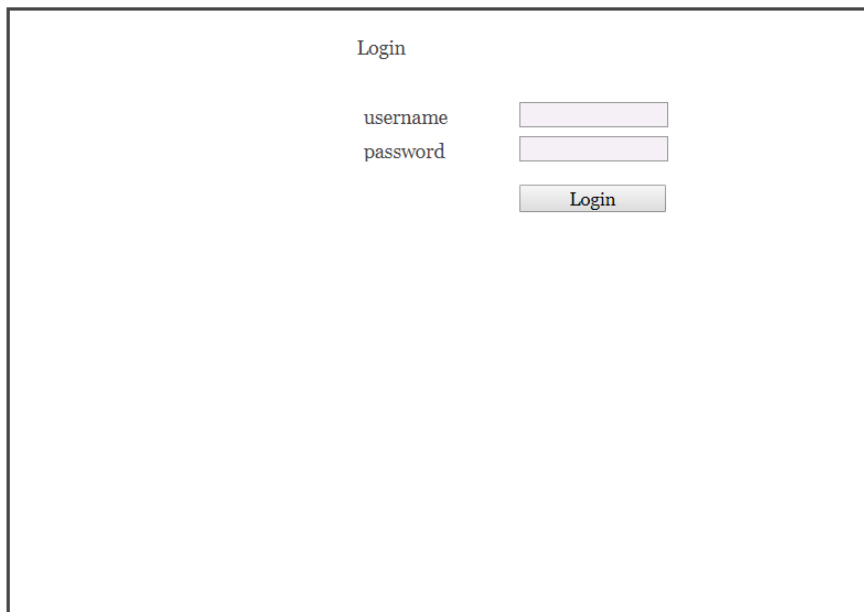
<b>Use Case</b>	Create Ticket
<b>Beschreibung</b>	Der User möchte ein neues Ticket erstellen
<b>Ziel / Ergebnis</b>	Am Ende sollte der User die Möglichkeit haben, ein neues Ticket zu erstellen
<b>Kategorie</b>	primär (notwendig u. Häufig verwendet)
<b>Vorbedingungen</b>	Die einzige Vorbedingung wäre, dass der User überhaupt die Möglichkeit hat sich einzuloggen um auf das Ticket System zugreifen zu können
<b>Nachbedingungen</b>	<p>Erfolg: Der User kann ein neues Ticket erstellen, indem er das Formular ausfüllt und auf speichern klickt.</p> <p>Fehlschlag: Der Benutzer kann das Ticket nicht speichern.</p>
<b>Invarianten</b>	Die Funktion kann für alle definierten Personen erstellt werden.
<b>Akteure</b>	Der User, der ein neues Ticket erstellen möchte ist der Akteur
<b>Auslösendes Ereignis</b>	UseCase wird initiiert, wenn der User erfolgreich ein neues Ticket erstellt hat.
<b>Ablaufbeschreibung</b>	<b>Beschreibung des Standardfalls</b> <ol style="list-style-type: none"> <li>1. User startet das Programm</li> <li>2. User klickt in das Textfeld "Ticket Name"</li> <li>3. User klickt in das Textfeld „Category“</li> <li>User klickt in das Textfeld „Departement“</li> <li>User klickt in das Textfeld „Priority“</li> <li>User klickt in das Textfeld „Description“</li> <li>4. User gibt seine Daten ein</li> <li>5. User klickt auf „speichern“</li> <li>6. Erstelltes Ticket wird angezeigt</li> </ol>

---

Use Case	
Use Case	View Ticket
Beschreibung	Der User möchte alle Tickets ansehen
Ziel / Ergebnis	Am Ende sollte der User die Möglichkeit haben alle Tickets anzusehen
Kategorie	primär (notwendig u. Häufig verwendet)
Vorbedingungen	Die einzige Vorbedingung wäre, dass der User überhaupt die Möglichkeit hat sich einzuloggen um auf das Ticket System zugreifen zu können
Nachbedingungen	Erfolg: Der User kann auf das System zugreifen und sieht die Tabelle mit allen erstellten Ticktes.  Fehlschlag: Der Benutzer kann nicht auf das System zugreifen und sieht die Tabelle nicht.
Invarianten	Die Funktion kann für alle definierten Personen erstellt werden.
Akteure	Der User, der die Tickets ansehen möchte ist der Akteur
Auslösendes Ereignis	UseCase wird initiiert, wenn der User erfolgreich ein neues ein Ticket aufrufen kann.
Ablaufbeschreibung	<b>Beschreibung des Standardfalls</b> 1. User startet das Programm 2. User loggt sich ein 3. User kann Tabelle mit Tickets sehen 4. Erstelltes Ticket wird angezeigt

<b>Use Case</b>	
<b>Use Case</b>	Delete Ticket
<b>Beschreibung</b>	Der User möchte ein Ticket löschen
<b>Ziel / Ergebnis</b>	Am Ende sollte der User die Möglichkeit haben ein Ticket zu löschen
<b>Kategorie</b>	primär (notwendig u. Häufig verwendet)
<b>Vorbedingungen</b>	Die einzige Vorbedingung wäre, dass der User überhaupt die Möglichkeit hat Tickets zu erstellen um diese später auch wieder zu löschen.
<b>Nachbedingungen</b>	<p>Erfolg: Der User kann mit dem Button „löschen“ ein Ticket löschen.</p> <p>Fehlschlag: Der Benutzer kann kein Ticket erstellen. Somit kann er auch keine Tickets löschen.</p>
<b>Invarianten</b>	Die Funktion kann für alle definierten Personen erstellt werden.
<b>Akteure</b>	Der User, der das Ticket löschen möchte ist der Akteur
<b>Auslösendes Ereignis</b>	UseCase wird initiiert, wenn der User erfolgreich ein Ticket löschen kann
<b>Ablaufbeschreibung</b>	<p><b>Beschreibung des Standardfalls</b></p> <ol style="list-style-type: none"> <li>1. User startet das Programm</li> <li>2. User loggt sich ein</li> <li>3. User kann Tabelle mit Tickets sehen</li> <li>4. User klickt auf „löschen“</li> <li>5. Ticket wird entfernt</li> </ol>

### 3.4 Mockups



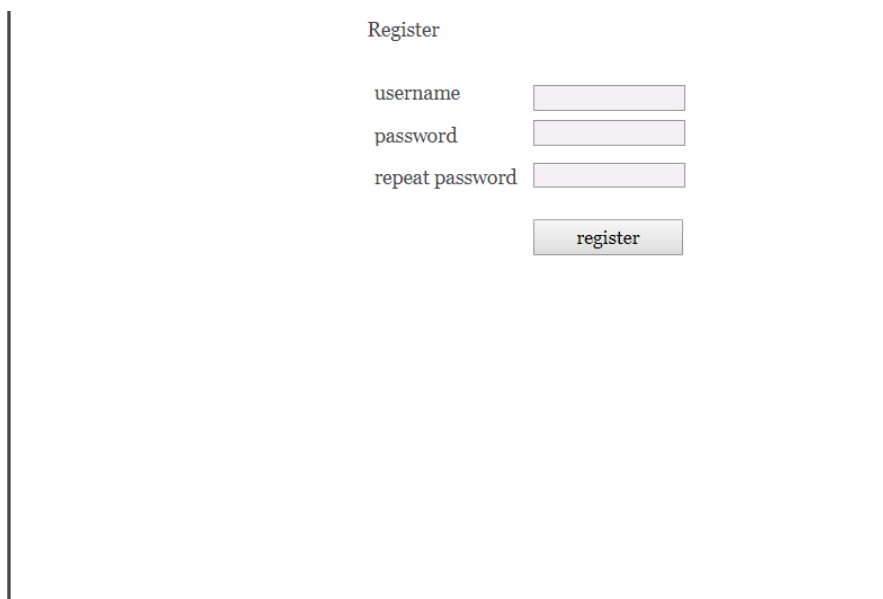
Mockup of a Login form. The form is titled "Login" and contains two input fields: "username" and "password". Below the input fields is a "Login" button.

Login

username

password

#### Mockups 1



Mockup of a Register form. The form is titled "Register" and contains three input fields: "username", "password", and "repeat password". Below the input fields is a "register" button.

Register

username

password

repeat password

Ticket System

Ticket Titel

Category

Departement

Priority

ID	Category	Departement	Priority	

### 3.5 Benutzeroberfläche

Die Applikation ist in zwei Bereiche aufgeteilt, das Frontend und das Backend. Für beides existieren keine Richtlinien und ich kann frei wählen.

#### 3.5.1 Backend

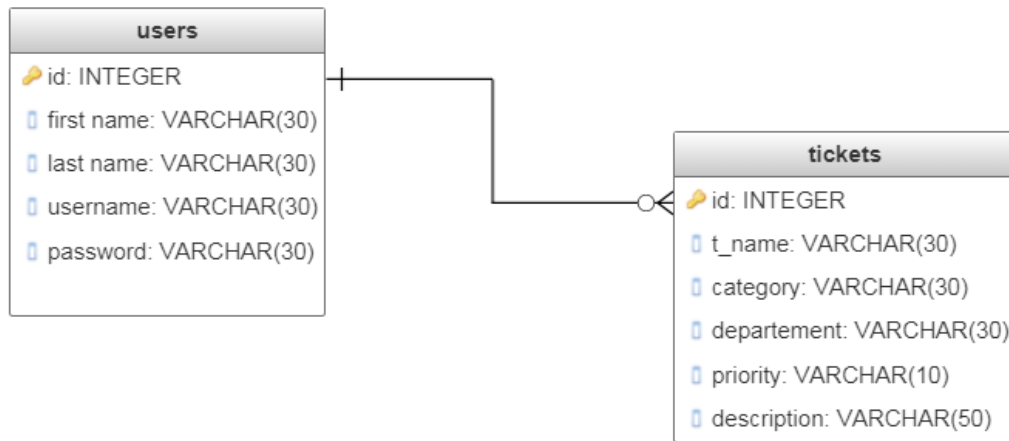
Da ich bis jetzt die meisten Backend mit Bootstrap realisiert habe und ich dies vom Design her sehr ansprechend finde, verwende ich auch bei diesem Projekt das CSS Framework.

Bootstrap Version: v3.3.7

<http://getbootstrap.com/>

### 3.6 Datenbank

Das ERM dient dazu, Daten zu modellieren ohne technische Aspekte zu beachten. Das ERM stellt hier eine Tabellenstruktur dar, die für das Projekt in eine eigene Datenbank hinzugefügt wird.



ERM 1

## 3.7 Tabellendefinitionen

### 3.7.1 users

Spaltenname	Typ	PK	NN	Beschreibung
id	INT(10)	X	X	Primärschlüssel der Tabelle
First name	Varchar(30)		X	Vorname des Benutzers
Last name	Varchar(30)		x	Nachname des Benutzers
Username	Varchar(30)		X	Benutzername
Password	Varchar(30)		X	Passwort des Benutzers

**tbl users 1**

### 3.7.2 tickets

Spaltenname	Typ	PK	NN	Beschreibung
id	INT(10)	X	X	Primärschlüssel der Tabelle
t_name	Varchar(30)		X	Name des Tickets
category	Varchar(30)		X	In welche Kategorie: Bug Report etc.
departement	Varchar(30)		X	Zu welcher Abteilung der Bug Report gehört
priority	Varchar(10)			Wie hoch die Wichtigkeit des Tickets ist
description	Varchar(50)		X	Beschreibung des Tickets

tbl tickets 1



## 4 Entscheidung

### 4.1.1 Versionsverwaltung

Für die Versionsverwaltung ist Git die Beste Wahl, doch da dies ein relativ kleines Projekt ist, macht es keinen grossen Unterschied, ob man Subversion oder Git einsetzt. Da ich mich nicht so gut mit Subversion auskenne habe ich mich dazu entschieden, weiterhin Git für die Versionsverwaltung zu verwenden.

### 4.1.2 Datenbank

Das Design der Datenbank und die Tabellendefinitionen wurden bereits in der Planung erstellt. Ich werde eine MySQL Datenbank verwenden.

### 4.1.3 Programmiersprache

Ich verwende die Programmiersprache PHP. Ich habe mich für diese Sprache entschieden, da ich schon vertiefte Grundkenntnisse erlangen konnte und täglich auch damit zu tun habe in meiner Abteilung. Ausserdem kann ich PHP Objekt orientiert programmieren, welches ein MUSS Kriterium ist für dieses Projekt.

## 5 Realisierung

### 5.1 Umsetzung

Für die Entwicklung der Login -Applikation wird gemäss der Entscheidung das CSS-Framework Bootstrap verwendet.

#### Funktionsweise

Der User kann sich anhand des Formulars registrieren. Das File habe ich registration.php genannt. Es besteht hier für den User die Möglichkeit sich zu registrieren um sich später anzumelden. Die Daten werden in der DB gespeichert.

### Registration

Name:	<input type="text"/>
Username:	<input type="text"/>
Email:	<input type="text"/>
Password:	<input type="password"/>
<input type="button" value="Register"/>	
<input type="button" value="Click to login"/>	

#### Registration 1

Nach dem der User sich erfolgreich registriert hat, kann dieser sich anmelden. Dazu gibt er seine Anmeldedaten in das Login Formular ein, das File habe ich login.php genannt. Die Daten werden anhand des classUser.php files in der DB überprüft.

### Login

Username or Email:	<input type="text"/>
Password:	<input type="password"/>
<input type="button" value="Login"/>	
<input type="button" value="New User"/>	

#### Login 1

Nach dem der User sich erfolgreich angemeldet hat, hat dieser Zugriff auf das Ticket System. Hier kann dieser ein neues Ticket erfassen. Dieses wird dann in der Tabelle unten angezeigt. Der User kann das Ticket nach Belieben löschen oder bearbeiten.

# Ticket System

Logout

Ticket Details

Ticket Name

Ticket name

Category

Category

Departement

Departement

Priority

Priority

Description

Description

Store

Number	Ticket Name	Category	Departement	Priority	Description		
--------	-------------	----------	-------------	----------	-------------	--	--

Ticket System 1

Links oben kann der User den Logout Button betätigen, um sich auszuloggen.

## 5.2 Klassen

### 5.2.1 Ticket->Data

Diese Klasse beschreibt die Funktionen des Ticket Formulars sowie der Tabelle.

### 5.2.2 Login->User

In dieser Klasse werden die Funktionen zum authentisieren der Benutzer erstellt.

### 5.2.3 Login -> DB\_con

Es wird eine Datenbankverbindung hergestellt.

### 5.2.4 Ticket-> Database

Es wird eine Datenbankverbindung hergestellt.

## 6 Kontrolle

### 6.1 Testphasen

Im folgenden Abschnitt werden die Testphasen und Methoden aufgezeigt, nach denen die Applikation getestet wird.

Was wird getestet?

Die normalen Anwendungen der Applikation werden in diesen Testphasen getestet. Die Testfälle richten sich nach den definierten Funktionen. Der Schwerpunkt wird dabei auf die Funktionalität der Muss Kriterien gelegt.

### 6.2 Fehlerklassifizierungen

Fehlerklasse	Beschreibung
<b>OK</b>	keine Abweichungen eingetreten
<b>Kleine Abweichung</b>	Die Funktionalität ist zwar gegeben, aber nicht vollständig
<b>Mittlere Abweichung</b>	Die Funktionalität kann nur eingeschränkt gebraucht werden
<b>Grobe Abweichung</b>	Die Funktionalität ist nicht gegeben oder Durchführung führt zu einem Absturz

#### Fehlerklassifizierung 1

### 6.3 Testfälle

Testfall 01	<b>Einloggen</b>
<b>Ausgangszustand</b>	Login Fenster wird angezeigt um Login Daten einzugeben
<b>Beschreibung</b>	Der User möchte sich einloggen, um auf das Ticket System zugreifen zu können. Mit Hilfe des Login Formulars kann der User seine Daten eingeben.
<b>Aktion</b>	Der User gibt seinen registrierten Usernamen und sein registriertes Passwort an. Der User klickt auf den Button „Login“
<b>Erwarteter Ergebniszustand</b>	Die Daten werden in der DB überprüft und der User wird weitergeleitet auf index.php

#### Testfälle 1

Testfall 02	<b>Registrierung mit korrekten Daten</b>
<b>Ausgangszustand</b>	Registrierungsformular wird angezeigt
<b>Beschreibung</b>	Ein User möchte sich in das Ticket System einloggen und will sich nun registrieren. Mit Hilfe des Registrierungsformulars gibt er seine Daten an.
<b>Aktion</b>	Der User gibt seinen Usernamen und sein Passwort ein. Er wiederholt sein Passwort und klickt auf den Button „registrieren“.
<b>Erwarteter Ergebniszustand</b>	Die Daten werden nach Klick auf Button „registrieren“ in der DB gespeichert.

Testfall 03	
<b>Ausgangszustand</b>	<b>Ticket erstellen</b> Formular wird angezeigt, um ein neues Ticket zu erstellen. Spalten; Titel Ticket, Category, Departement, Priority, Description
<b>Beschreibung</b>	Ein User möchte ein neues Ticket erstellen, dazu gibt er alle notwendigen Informationen in das Formular ein.
<b>Aktion</b>	Der User füllt die Textfelder mit Daten und klickt auf den Submit Button.
<b>Erwarteter Ergebniszustand</b>	Die Daten werden nach Klick auf Submit Button in der DB gespeichert.
Testfall 04	
<b>Ausgangszustand</b>	<b>Ticket speichern</b> Der User gibt alle notwendigen Informationen in das Formular ein.
<b>Beschreibung</b>	Der User möchte sein neues Ticket speichern.
<b>Aktion</b>	Die Daten werden nach Klick auf den Submit Button in der DB gespeichert. Das Ticket wird in der View angezeigt.
<b>Erwarteter Ergebniszustand</b>	Die Daten werden in der DB gespeichert und das Ticket wird ausgegeben.
Testfall 05	
<b>Ausgangszustand</b>	<b>Ticket löschen</b> Der User kann die Tabelle mit allen Tickets ansehen, es existiert ein Button „delete“
<b>Beschreibung</b>	Ein Ticket soll gelöscht werden
<b>Aktion</b>	Der User löscht das Ticket seiner Wahl, in dem er auf den „delete“ Button klickt.
<b>Erwarteter Ergebniszustand</b>	Das ausgewählte Ticket wird gelöscht.
Testfall 06	
<b>Ausgangszustand</b>	<b>Ticket bearbeiten</b> Der User kann die Tabelle mit allen Tickets ansehen, es existiert ein Button „update“
<b>Beschreibung</b>	Ein Ticket soll bearbeitet werden
<b>Aktion</b>	Ein Ticket kann bearbeitet werden, wenn der User auf den Button „update“ klickt.
<b>Erwarteter Ergebniszustand</b>	Der User klickt auf den Button „update“ und es besteht die Möglichkeit das Ticket anhand des Formulars zu bearbeiten. Anschliessend wird das Ticket wieder gespeichert.

Testfall 07	<b>Login mit falschen Daten</b>
<b>Ausgangszustand</b>	Loginformular wird angezeigt
<b>Beschreibung</b>	Der User möchte sich einloggen, um auf das Ticket System zugreifen zu können. Mit Hilfe des Login Formulars kann der User seine Daten eingeben.
<b>Aktion</b>	User gibt falschen Benutzernamen und falsches Passwort ein.
<b>Erwarteter Ergebniszustand</b>	Fehlermeldung „Username oder Passwort ist falsch“

Testfall 08	<b>Registrierung mit falschen Daten</b>
<b>Ausgangszustand</b>	Registrierungsformular wird angezeigt
<b>Beschreibung</b>	Ein User möchte sich in das Ticket System einloggen und will sich nun registrieren. Mit Hilfe des Registrierungsformulars gibt er seine Daten an.
<b>Aktion</b>	User gibt schon vorhandenen Benutzernamen ein.
<b>Erwarteter Ergebniszustand</b>	Fehlermeldung „Dieser Benutzername ist bereits vergeben“

Testfall 09	<b>Ticket Description mit mehr 50+ Zeichen befüllen</b>
<b>Ausgangszustand</b>	Formular wird angezeigt um neues Ticket zu eröffnen
<b>Beschreibung</b>	Der User gibt möchte eine Ticket Beschreibung eingeben
<b>Aktion</b>	User gibt 50+ Zeichen in das Textfeld ein.
<b>Erwarteter Ergebniszustand</b>	Das Ticket kann nicht gespeichert werden, da nur 50 Zeichen in der Beschreibung erlaubt sind.

Testfall 10	<b>Registrierung mit falschen Daten/Sonderzeichen</b>
Ausgangszustand	Registrierungsformular wird angezeigt
Beschreibung	Ein User möchte sich in das Ticket System einloggen und will sich nun registrieren. Mit Hilfe des Registrierungsformulars gibt er seine Daten an.
Aktion	User gibt Sonderzeichen ein
Erwarteter Ergebniszustand	Fehlermeldung „Bitte geben Sie einen korrekten Benutzernamen, Passwort etc. ein.“

Testfall 11	<b>Login mit falschen Daten/Sonderzeichen</b>
Ausgangszustand	Login Formular wird angezeigt
Beschreibung	Ein User möchte sich in das Ticket System einloggen. Mit Hilfe des Login Formulars gibt er seine Daten an.
Aktion	User gibt Sonderzeichen ein
Erwarteter Ergebniszustand	Fehlermeldung „Dieser Username bzw. dieses Passwort ist nicht vorhanden.“

## Testprotokoll

Name der Testperson	Olivia Pawlowitz
Datum der Durchführung	10.11.2017

### Testprotokoll 1

Testfall 01	<b>Einloggen</b>
Ausgangszustand	Login Formular wird angezeigt
Beschreibung	Der User wird aufgefordert sich einzuloggen mit seinen Benutzerdaten
Aktion	User gibt korrekte Login Daten ein
Erwarteter Ergebniszustand	Der User wird weitergeleitet auf Ticket System, nachdem die Daten überprüft wurden
Ergebnis / Abweichung	✓

Testfall 02	<b>Registrierung mit korrekten Daten</b>
Ausgangszustand	Registrierungsformular wird angezeigt
Beschreibung	Der User wird aufgefordert sich mit korrekten Daten zu registrieren. (valide Mailadresse etc.)
Aktion	User gibt korrekte Daten ein.
Erwarteter Ergebniszustand	Der User wird in der DB gespeichert und kann sich nun einloggen mit dem Login Formular
Ergebnis / Abweichung	✓

Testfall 03	<b>Ticket erstellen</b>
Ausgangszustand	User ist eingeloggt und kann auf das Ticket System zugreifen
Beschreibung	User möchte ein Ticket erstellen
Aktion	User gibt im Formular „Ticket Details“ alle notwendigen Informationen bezüglich seines Tickets ein
Erwarteter Ergebniszustand	User kann Daten in Formular abfüllen und speichern
Ergebnis / Abweichung	✓

Testfall 04	<b>Ticket speichern</b>
Ausgangszustand	User kann Ticket erstellen
Beschreibung	User möchte das Ticket nun abspeichern
Aktion	Der Button „save“ wird betätigt
Erwarteter Ergebniszustand	Das Ticket wird gespeichert in DB und in der unteren Tabelle angezeigt
Ergebnis / Abweichung	✓

Testfall 05	<b>Ticket löschen</b>
Ausgangszustand	Ticket ist in Tabelle vorhanden
Beschreibung	Der User möchte das erstellte Ticket löschen
Aktion	User klickt auf den Button „delete“ um das bereits vorhandene Ticket zu löschen
Erwarteter Ergebniszustand	Das Ticket wird gelöscht
Ergebnis / Abweichung	✓

Testfall 06	<b>Ticket bearbeiten</b>
Ausgangszustand	Ticket ist in Tabelle vorhanden
Beschreibung	Der User möchte ein schon vorhandenes Ticket aus der Tabelle bearbeiten
Aktion	Der User klickt auf den Button „update“
Erwarteter Ergebniszustand	Der User kann das Ticket im Formular „Ticket Details“ bearbeiten
Ergebnis / Abweichung	✓

Testfall 07	<b>Login mit falschen Daten</b>
Ausgangszustand	User hat sich bereits registriert und möchte sich anmelden
Beschreibung	Es wird überprüft ob der User in der DB vorhanden ist oder nicht
Aktion	Der User gibt einen falschen Usernamen oder ein falsches Passwort ein
Erwarteter Ergebniszustand	„Wrong username or password“
Ergebnis / Abweichung	✓



Testfall 08	<b>Registrierung mit falschen Daten</b>
Ausgangszustand	User möchte sich registrieren
Beschreibung	User gibt falsche Daten ein in das Formular ein.
Aktion	Der User gibt eine unzulässige Emailadresse ein.
Erwarteter Ergebniszustand	Fehlermeldung „Wrong emailadress“
Ergebnis / Abweichung	✓

Testfall 09	<b>Ticket Description mit mehr 50+ Zeichen befüllen</b>
Ausgangszustand	Ticket Formular wird angezeigt und User möchte Beschreibung einfügen
Beschreibung	Ticket Beschreibung soll mehr als 50 Zeichen enthalten.
Aktion	User verwendet bei der Ticket Beschreibung mehr als 50 Zeichen
Erwarteter Ergebniszustand	Fehlermeldung „Only 50 chars“
Ergebnis / Abweichung	Es gibt keine Fehlermeldung, der Text wird nur 50 Zeichen lang angezeigt.

## Testbericht

Testfall	Problem	Weiteres Vorgehen
Testfall 09	Es wird keine Fehlermeldung angezeigt	

### Testbericht 1

## Testergebnis

Die Testergebnisse sind grösstenteils erfolgreich verlaufen. Es gab nur eine Abweichung von den erwarteten Ergebnissen. Bis zur Abgabe kann das Problem jedoch nicht behoben werden.

## 7 Auswertung

### 7.1 Möglichkeiten zur Weiterentwicklung

#### -Kontobestätigung per Email

Als nächstes könnte ich hinzufügen, dass der User sich erst dann registrieren kann, wenn die Email Adresse von ihm bestätigt wurde.

#### -Funktion hochladen von Bilddateien

Es wäre eigentlich gedacht, dass ich einen File Uploader implementiere, jedoch hat dies aus zeitlichen Gründen keinen Sinn gemacht. Dies wäre eine Möglichkeit zur Weiterentwicklung, da heutzutage jedes gute Ticket System so eine Funktion aufweist.

#### -Admin Bereich zum Bearbeiten der Tickets

Eine gute und wichtige Funktion wäre auch ein spezifischer Admin Bereich. Nur der Admin kann dann alle Tickets bearbeiten oder löschen.

### 7.2 Wie ich die Arbeit empfunden habe

Wenn ich rückblickend auf dieses Modul schaue, konnte ich meinen Rucksack voll mit neuen Erfahrungen packen. Die Arbeit habe ich als sehr interessant empfunden. Die meisten Techniken, welche ich eingesetzt habe waren mir bereits bekannt, jedoch konnte ich mich nie richtig anfreunden mit der Objekt orientierten Programmierung. Bevor dieses Modul gestartet habe, musste bzw. wollte ich auch viel Zeit in die Objekt orientierte Programmierung investieren. Ich habe mir aufgrund dessen etliche Bücher durchgelesen und aus eigenen Projekten schlussendlich Erfahrungen gesammelt. Es hat mir sehr Spass gemacht mit dem CSS-Framework Bootstrap zu arbeiten und ich werde dies sicher auch in der IPA verwenden. Es erleichtert einem so viel und man gewinnt nur Zeit, was in diesem Projekt sehr wichtig war. So konnte ich mich ganz auf die Funktionen meiner Applikation konzentrieren.

Während des Moduls verlief nicht alles wie erwartet, doch aus diesen Fehlern kann ich nur profitieren. Es wäre komisch, wenn ich keine Fehler machen würde und auf anhin alles geklappt hätte.

### 7.3 Was ich das nächste Mal verbessern könnte

Ich habe zwischenzeitlich gemerkt, dass ich ab und zu meinem Zeitplan ausweiche und an einer Sache länger benötige als gedacht. Ich hatte einfach noch kein Gefühl für das richtige Zeitmanagement, bei so einem Projekt. Nächstes Mal werde ich mehr Zeit für die relevanten Dinge aufwenden als zum Beispiel für eine SOLL-IST-Analyse.

## 7.4 Glossar

Begriff	Erklärung
Repository	Ablage eines Versionierungssystems
Framework	Rahmenwerk für Softwarelösung

Glossar 1

## 7.5 Quellenverzeichnis

Tabellendesign :	<a href="https://v4-alpha.getbootstrap.com/components/forms/">https://v4-alpha.getbootstrap.com/components/forms/</a>
Glyphicons:	<a href="https://www.w3schools.com/bootstrap/bootstrap_ref_comp_glyphs.asp">https://www.w3schools.com/bootstrap/bootstrap_ref_comp_glyphs.asp</a>
MySQL:	<a href="https://stackoverflow.com/questions/8243503/mysql-insert-id-returns-0">https://stackoverflow.com/questions/8243503/mysql-insert-id-returns-0</a> <a href="https://stackoverflow.com/questions/5665571/auto-increment-in-phpmyadmin">https://stackoverflow.com/questions/5665571/auto-increment-in-phpmyadmin</a>
Conditions :	<a href="https://stackoverflow.com/questions/7229745/how-can-i-store-a-php-conditional-statement-in-the-database-for-later-use">https://stackoverflow.com/questions/7229745/how-can-i-store-a-php-conditional-statement-in-the-database-for-later-use</a>
DB Connection :	<a href="http://php.net/manual/de/function.mysql-select-db.php">http://php.net/manual/de/function.mysql-select-db.php</a>

## 8 Anhang

Es werden einzelne Codeausschnitte aus der Applikation gezeigt. Diese dienen der Veranschaulichung der Programmierung der Applikation.

### 8.1 Codeausschnitte

#### 8.1.1 Ticketsystem : index.php

```
<?php

include "action.php";

?>
<!DOCTYPE html>
<html>
    <head>
        <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
        <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.1.1/jquery.min.js"></script>
        <script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
        <meta name="viewport" content="width=device-width, initial-
scale=1.0"/>

        <title>Ticket System</title>

        <link rel="stylesheet" href="" type="text/css" />
        <script type="text/javascript"></script>
    </head>
    <body>

        <div class="container">
            <div class="jumbotron">

                <h1>Ticket System</h1>

                <a href="http://localhost/ticket/login/Login.php" class="btn
pull-right name="logout" class="btn btn-primary">Logout</a>
            </div>
        </div>
        <div class="container">
            <div class="row">
                <div class="col-xs-6 col-md-10"></div>
                <div class="col-xs-6 col-md-10">
                    <div class="panel panel-primary">
                        <div class="panel-heading"> Ticket Details</div>
                        <div class="panel-body">
```

&lt;?php

```

        if(isset($_GET["update"])){
            $id = $_GET["id"];
            $where = array("id"=>$id,);
            $row = $obj->select_record("tickets",$where);

        ?>
        <!-- Tabelle für Ticket System-->

        <form method="post" action="action.php">
        <table class="table table-hover">
        <tr>

        <td><input type="hidden" name="id" value="<?php echo $id; ?>"></td>

        </tr>

        <tr>

        <td>Name</td>

        <td><input type="text" class="form-control" value="<?php echo
$row["t_name"]; ?>" name="name" placeholder=" Ticket name"></td>

        </tr>

        <tr>

        <td>Category</td>

        <td><input type="text" class="form-control" name="category" value="<?php
echo $row["category"]; ?>" placeholder=" Category"></td>

        </tr>

        <tr>

        <td>Departement</td>

        <td><input type="text" class="form-control" name="departement" value="<?php
echo $row["departement"]; ?>" placeholder=" Departement"></td>

        </tr>

        <tr>

        <td>Priority</td>

        <td><input type="text" class="form-control" name="priority" value="<?php
echo $row["priority"]; ?>" placeholder=" Priority"></td>

        </tr>

```

```

<tr>

    <td>Description</td>

    <td><input type="text" class="form-control" name="description" value="<?php
echo $row["description"]; ?>" placeholder=" Description"></td>

</tr>

<tr>

    <td colspan="2" align="c"><input type="submit" class="btn btn-primary"
name="edit" value="Update"></td>

</tr>

</table>

</form>

<?php
}else{

?>

<!-- Formular fuer Ticket System-->

<form method="post" action="action.php">

    <table class="table table-hover">

        <tr>
            <td>Ticket Name</td>

            <td><input type="text" class="form-control" name="name" placeholder=" Ticket
name"></td>

        </tr>

        <tr>

            <td>Category</td>

            <td><input type="text" class="form-control" name="category" placeholder="
Category"></td>

        </tr><tr>

            <td>Departement</td>

            <td><input type="text" class="form-control" name="departement" placeholder="
Departement"></td>

        </tr>

```

```
<tr>

    <td>Priority</td>

    <td><input type="text" class="form-control" name="priority" placeholder="
Priority"></td>

</tr>

<tr>

<tr>

    <td>Description</td>

    <td><input type="text" class="form-control" name="description" placeholder="
Description"></td>

</tr>

<tr>

    <td colspan="2" align="c"><input type="submit" class="btn btn-primary"
name="submit" value="Save"></td>

</tr>

</table>

</form>

<?php
}

?>

</div>

</div>

</div>

<div class="col-xs-6 col-md-10"></div>

</div>

</div>
```



```

<div class="container">
    <div class="row">
        <div class="col-xs-6 col-md-10"></div>
        <div class="col-xs-6 col-md-10">

            <table class="table">
                <thead class="thead-inverse">

                    <tr>
                        <th>#</th>
                        <th>Ticket Name</th>
                        <th>Category</th>
                        <th>Departement</th>
                        <th>Priority</th>
                        <th>Description</th>

                    <th>&nbsp;</th>
                    <th>&nbsp;</th>

                </tr>

            <?php
                $myrow = $obj->fetch_record("tickets");
                foreach ($myrow as $row) {

                    ?>

                    <tr>
                        <td><?php echo $row["id"]; ?></td>
                        <td><?php echo $row["t_name"]; ?></td>
                        <td><b><?php echo $row["category"]; ?></b></td>
                        <td><b><?php echo $row["departement"]; ?></b></td>
                        <td><b><?php echo $row["priority"]; ?></b></td>
                        <td><b><?php echo $row["description"]; ?></b></td>
                        <td><a class="glyphicon glyphicon-
pencil"href="index.php?update=1&id=<?php echo $row["id"]; ?>" </a></td>
                        <td><a class="glyphicon glyphicon-trash"
href="action.php?delete=1&id=<?php echo $row["id"]; ?>" </a></td>
                    </tr>

                <?php
                }

                ?>

            </table>

        </div>
    <div class="col-xs-6 col-md-10"></div>
</div>

</body>
</html>

```

### 8.1.2 Ticketsystem : dbconfig.php

```
<?php
```

```
class Database
{
public $con;
public function __construct(){
$this->con = mysqli_connect("localhost","root","", "ticket_system");
if (!$this->con) { // if connection fails the system returns an error
echo "Error in Connecting ".mysqli_connect_error();
}
}
}

?>
```

### 8.1.3 Ticketsystem : classAction.php

```
<?php
```

```
include "dbconfig.php";
```

```
class Data extends Database
{
    public function insert_record($table,$fields){ // where I want to
insert my content
        $sql = "";
        $sql .= "INSERT INTO ".$table; // here I will receive my input
variables
        $sql .= " (" .implode(",", array_keys($fields)).") VALUES ";
        $sql .= "(" .implode("'", array_values($fields)).")";
        $query = mysqli_query($this->con,$sql);

        if($query){
            return true;
        }
    }

    public function fetch_record($table){
        $sql = "SELECT * FROM ".$table;
        $array = array();
        $query = mysqli_query($this->con,$sql);

        while($row = mysqli_fetch_assoc($query)){

            $array[] = $row;

        }

        return $array;
    }
}
```

```
public function select_record($table,$where){
    $sql = "";
    $condition = "";

    foreach ($where as $key => $value) {

        $condition .= $key . "=" . $value . " AND ";

    }
    //AND chars must be removed with substract
    $condition = substr($condition, 0, -5);
    $sql .= "SELECT * FROM ".$table." WHERE ".$condition;
    $query = mysqli_query($this->con,$sql);
    $row = mysqli_fetch_array($query);

    return $row;
}

public function update_record($table,$where,$fields){
    $sql = "";
    $condition = "";

    foreach ($where as $key => $value) {

        $condition .= $key . "=" . $value . " AND ";

    }

    //AND chars must be removed with substract
    $condition = substr($condition, 0, -5);
    foreach ($fields as $key => $value) {

        $sql .= $key . "=" . $value . ", ";

    }

    $sql = substr($sql, 0,-2);
    $sql = "UPDATE ".$table." SET ".$sql." WHERE ".$condition;

    if(mysqli_query($this->con,$sql)){
        return true;
    }
}

public function delete_record($table,$where){
    $sql = "";
    $condition = "";

    foreach ($where as $key => $value) {

        $condition .= $key . "=" . $value . " AND ";

    }
}
```

```
//AND chars must be removed with substract
    $condition = substr($condition, 0, -5);
    $sql = "DELETE FROM ".$table." WHERE ".$condition;

    if(mysqli_query($this->con,$sql)){
        return true;
    }
}

}

$obj = new Data;

//The button save is pressed and the form can be saved the data will be
insert in the current fields
if(isset($_POST["submit"])){

    $myArray = array(
        "t_name" => $_POST["name"],
        "category" => $_POST["category"],
        "departement" => $_POST["departement"],
        "priority" => $_POST["priority"],
        "description" => $_POST["description"]
    );

    //call the function insert_record
    if($obj->insert_record("tickets",$myArray)){
        header("location:index.php?msg=Record Inserted");
        //message: Record inserted
    }
}

//The button edit is pressed and the form can be edited
if(isset($_POST["edit"])){
    $id = $_POST["id"];
    $where = array("id"=>$id);
    $myArray = array(
        "t_name" => $_POST["name"],
        "category" => $_POST["category"],
        "departement" => $_POST["departement"],
        "priority" => $_POST["priority"],
        "description" => $_POST["description"]
    );

    //call the function update_record
    if($obj->update_record("tickets",$where,$myArray)){
        header("location:index.php?msg=Record Updated Successfully");
        //message: Update Successfully
    }
}
}
```

```
//The button delete is pressed and the id will be deleted
if(isset($_GET["delete"])){
    $id = $_GET["id"] ?? null;
    $where = array("id"=>$id);
    //call the function delete_record
    if($obj->delete_record("tickets",$where)){
        header("location:index.php?msg=Record Deleted Successfully");
        //message: Deleted Successfully
    }
}
?>
```

#### 8.1.4 Login: login.php

```
<?php

session_start();

include_once 'include/ClassUser.php';

$user = new User();

if (isset($_POST['submit'])) {

    extract($_POST);
    //check user in db
    $login = $user->check_login($emailusername, $password);

    if ($login) {

        //forwarding to the ticket system
        header("location:http://localhost/ticket/index.php");

    } else {

        echo 'Wrong username or password';

    }

}

?>
```

```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="utf-8">

  <title>Login </title>
  <link rel="stylesheet" href="assets/css/bootstrap.min.css" />

</head>
<body>

<div id="container" class="container">

  <h1>Login</h1>

  <form action="" method="post" name="Login">

    <table class="table " width="400">

      <tr>

        <th>Username or Email:</th>

        <td>

          <input type="text" name="emailUsername" required>

        </td>

      </tr>

      <tr>

        <th>Password:</th>

        <td>

          <input type="password" name="password" required>

        </td>

      </tr>

      <tr>

        <td>&nbsp;</td>

        <td>

          <input class="btn" type="submit" name="submit" value="Login"
onclick="return(submitlogin());">

        </td>

      </tr>

      <tr>

        <td>&nbsp;</td>


```

```
<td><a class="btn btn-primary"href="registration.php">New user</a></td>
    </tr>
</table>
</form>
</div>

<script>

    function submitlogin() { //Form validation; make sure information comes
through to the user.

        var form = document.login;

        if (form.emailusername.value == "") {

            alert("Enter email or username.");

            return false;

        } else if (form.password.value == "") {

            alert("Enter password.");

            return false;

        }

    }

}

</script>

</body>
</html>
```

### 8.1.5 Login: registration.php

```
<?php

include_once 'include/classUser.php';

$user = new User();

if (isset($_POST['submit'])){

    extract($_POST);

    $register = $user->reg_user($fullname, $username, $password, $email);

    if ($register) {

        echo "<div style='text-align:center'>Registration successful <a
href='login.php'>Click here</a> to login</div>";

    } else {

        echo "<div style='text-align:center'>Registration failed. Email or
Username already exists please try again.</div>";

    }

}

?>

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="utf-8">

    <title>Register</title>

    <link rel="stylesheet" href="assets/css/bootstrap.min.css" />

</head>

<body>
    <div id="container" class="container">

        <h1>Registration</h1>

        <form action="" method="post" name="reg">
```



```
<table class="table">
<tr>
    <th>Name:</th>
    <td>
        <input type="text" name="fullname" required>
    </td>
</tr>
<tr>
    <th>Username:</th>
    <td>
        <input type="text" name="username" required>
    </td>
</tr>
<tr>
    <th>Email:</th>
    <td>
        <input type="email" name="email" required>
    </td>
</tr>
<tr>
    <th>Password:</th>
    <td>
        <input type="password" name="password" required>
    </td>
</tr>
<tr>
    <td>&nbsp;</td>
    <td>
        <input class="btn" type="submit" name="submit" value="Register"
onclick="return(submitreg());">
    </td>
</tr>
```

```
        </td>
    </tr>
</tr>

        <td>&nbsp;</td>
        <td><a class="btn btn-primary" href="login.php">Login</a></td>
    </tr>
</table>

</form>

</div>

<script> //Form validation; make sure information comes through to the user.
    function submitreg() {
        var form = document.reg;
        if (form.name.value == "") {
            alert("Enter name.");
            return false;
        } else if (form.username.value == "") {
            alert("Enter username.");
            return false;
        } else if (form.pass.value == "") {
            alert("Enter password.");
            return false;
        } else if (form.email.value == "") {
            alert("Enter email.");
            return false;
        }
    }
</script>

</body>

</html>
```

### 8.1.6 Login: include/classUser.php

```
<?php
```

```
include "dbconfig.php";
```

```
class User{

    public $db;

    public function __construct(){
        $this->db = new mysqli(DB_SERVER, DB_USERNAME, DB_PASSWORD,
DB_DATABASE);

        if(mysqli_connect_errno()) {
            echo "Error: Could not connect to database.";
            exit;
        }
    }
    //for registration process
    public function reg_user($name,$username,$password,$email){

        $password = md5($password);
        $sql="SELECT * FROM users WHERE username='$username' OR
email='$email'";

        //checking if the username or email is available in db
        $check = $this->db->query($sql) ;
        $count_row = $check->num_rows;

        //if the username is not in db then insert to the table
        if ($count_row == 0){
            $sql1="INSERT INTO users SET username='$username',
password='$password', fullname='$name', email='$email'";
            $result = mysqli_query($this->db,$sql1) or
die(mysqli_connect_errno(). "Data cannot inserted");
            return $result;
        }
        else { return false;}
    }
}
```

```
//for login process
public function check_login($emailusername, $password){

    $password = md5($password);
    $sql2="SELECT id from users WHERE email='$emailusername' or
username='$emailusername' and password='$password'";

    //checking if the username is available in the table
    $result = mysqli_query($this->db,$sql2);
    $user_data = mysqli_fetch_array($result);
    $count_row = $result->num_rows;

    if ($count_row == 1) {
        // this login var will use for the session thing
        $_SESSION['login'] = true;
        $_SESSION['id'] = $user_data['id'];
        return true;
    }
    else{
        return false;
    }
}

//for showing the username or fullname
public function get_fullname($id){
    $sql3="SELECT fullname FROM users WHERE id = $id";
    $result = mysqli_query($this->db,$sql3);
    $user_data = mysqli_fetch_array($result);
    echo $user_data['fullname'];
}

//starting the session
public function get_session(){
    return $_SESSION['login'];
}

public function user_logout() {
    $_SESSION['login'] = FALSE;
    session_destroy();
}

}

?>
```

### 8.1.7 Login: include/dbconfig.php

<?php

```
define('DB_SERVER', 'localhost');
```

```
define('DB_USERNAME', 'root');
```

```
define('DB_PASSWORD', '');
```

```
define('DB_DATABASE', 'ticket_system');
```

```
class DB_con {
```

```
    public $connection;
```

```
    function __construct(){
```

```
        $this->connection = new mysqli(DB_SERVER, DB_USERNAME,
DB_PASSWORD,DB_DATABASE);
```

```
        //if connection is not working it returns database error
```

```
        if ($this->connection->connect_error) die('Database error -> ' .
$this->connection->connect_error);
```

```
    }
```

```
    function ret_obj(){
```

```
        return $this->connection;
```

```
    }
```

```
}
```