

Individuelle Praktische Arbeit

Zeiterfassungssystem

Autor: Marc Keller
Berufsrichtung: Informatik, Schwerpunkt Applikationsentwicklung
Firma: Siemens Schweiz AG
Abteilung: LBDS-CEE-CH-MCS-SWP 1, Atos AG

Fachvorgesetzter: Marcel Vogt
Hauptexperte: Name
Zweit-Experte: Name

Startdatum: 12.01.2017
Abgabedatum: 01.02.2017

Präsentationstermin: XX.XX.XXXX, XX:XX

Version: 1.0

Änderungshistorie

Eine Übersicht aller Dokumentation Versionen und der Änderungen.

Version	Datum	Beschreibung	Autor
0.1	12.01.2017	Projektorganisation, Vorkenntnisse, Vorarbeiten, Firmenstandards, Ist- und Soll-Analyse, Zeitplan inkl. Meilensteine und Use-Cases, erster Arbeitsjournaleintrag	Marc Keller
0.2	13.01.2017	Use-Cases, Testkonzept, ERD, DB-Tabellen beschrieben, Evaluierung Versionverwaltungssystem, zweiter Arbeitsjournaleintrag	Marc Keller
0.3	18.01.2017	Klassendiagramm ergänzt, dritter Arbeitsjournaleintrag	Marc Keller
0.4	19.01.2017	Klassendiagramm angepasst, Klassen dokumentiert, vierter Arbeitsjournaleintrag	Marc Keller
0.5	20.01.2017	Fünfter Arbeitsjournaleintrag	Marc Keller
0.6	25.01.2017	Sechster Arbeitsjournaleintrag	Marc Keller
0.7	26.01.2017	Testprotokoll erstellt und Testresultate notiert, Testfazit geschrieben, siebter Arbeitsjournaleintrag	Marc Keller
0.8	27.01.2017	Diverse Anpassungen und Ergänzungen, achter Arbeitsjournaleintrag	Marc Keller

0.9	31.10.2017	Screenshots ergänzt und dokumentiert, Reflexion, Erweiterungspotenzial der Applikation, Schlusswort, neunter Arbeitsjournaleintrag	Marc Keller
1.0	01.02.2017	Systemarchitektur ergänzt und dokumentiert, Quellcode, diverse Anpassungen und Ergänzungen, zehnter Arbeitsjournaleintrag	Marc Keller

Tabelle 1: Änderungshistorie

Inhaltsverzeichnis

Teil 1: Umfeld und Ablauf	6
1 Aufgabenstellung	6
2 Projektorganisation.....	8
2.1 Beteiligte Personen	8
2.2 Projektmanagement-Methode	9
3 Vorkenntnisse	11
3.1 Allgemeine Kenntnisse.....	11
3.2 Tätigkeiten	11
4 Vorarbeiten	12
5 Firmenstandards	12
6 Zeitplan	13
6.1 Meilensteile.....	14
7 Arbeitsprotokoll	15
7.1 Tag 1 – Donnerstag, 12.01.2017	15
7.2 Tag 2 – Freitag, 13.01.2017.....	17
7.3 Tag 3 – Mittwoch, 18.01.2017	19
7.4 Tag 4 – Donnerstag, 19.01.2017	21
7.5 Tag 5 – Freitag, 20.01.2017.....	23
7.6 Tag 6 – Mittwoch, 25.01.2017	25
7.7 Tag 7 – Donnerstag, 26.01.2017	27
7.8 Tag 8 – Freitag, 27.01.2017.....	29
7.9 Tag 9 – Dienstag, 31.01.2017	31
7.10 Tag 10 – Mittwoch, 01.02.2017	33
Teil 2: Projekt	35
1 Zusammenfassung	35
1.1 Projektbeschreibung	35
1.2 System-Beschreibung.....	35
2 Informieren	36
2.1 Ist-Analyse.....	36
2.2 Soll-Analyse	37
2.3 Use-Cases	38

3	Planen	48
3.1	Testkonzept.....	48
3.2	Software-Architektur	56
3.3	Datenbankdesign	57
4	Entscheiden.....	58
4.1	Versionverwaltungssystem	58
5	Realisieren.....	60
5.1	Klassendiagramm	60
5.2	Screenshots	64
6	Kontrollieren	67
6.1	Testing	67
7	Auswerten	72
7.1	Reflexion	72
7.2	Erweiterungspotenzial der Applikation	72
7.3	Schlusswort	73
Anhang	74
1	Glossar.....	74
2	Abbildungsverzeichnis	75
3	Tabellenverzeichnis.....	76
4	Quellenverzeichnis.....	77
5	Literaturverzeichnis	77
6	Programmcode.....	78
6.1	Datenbank Export	78
6.2	ServiceConfig.java	80
6.3	DBConfig.java	80
6.4	IDBService.java.....	81
6.5	DBServiceImpl.java.....	81
6.6	User.java	84
6.7	WorkingHours.java.....	84
6.8	WebSocketConfiguration.java	85
6.9	WebSocketHandler.java.....	86
6.10	Service.java	87
6.11	ServiceTest.java.....	91
6.12	application.properties.....	92

6.13	log4j.properties.....	93
6.14	index.html	94
6.15	script.js	95
6.16	timetable.js	97
6.17	datepicker.js.....	98

Teil 1: Umfeld und Ablauf

In diesem ersten Teil der Dokumentation folgen allgemeine Informationen über das Projekt, die Anforderungen, das Arbeitsumfeld sowie über die Vorkenntnisse und Vorarbeiten. Des Weiteren ist auch das Arbeitsjournal in diesem Teil zu finden.

1 Aufgabenstellung

Die Aufgabenstellung beinhaltet den eingereichten Projektbeschreibung. Dieser beinhaltet alle Infos zum umzusetzenden Projekt, wie bspw. den Projekttitel, die eingesetzten Technologien, die Anforderungen, eine kurze Funktionsbeschreibung sowie einige Worte zur Motivation.

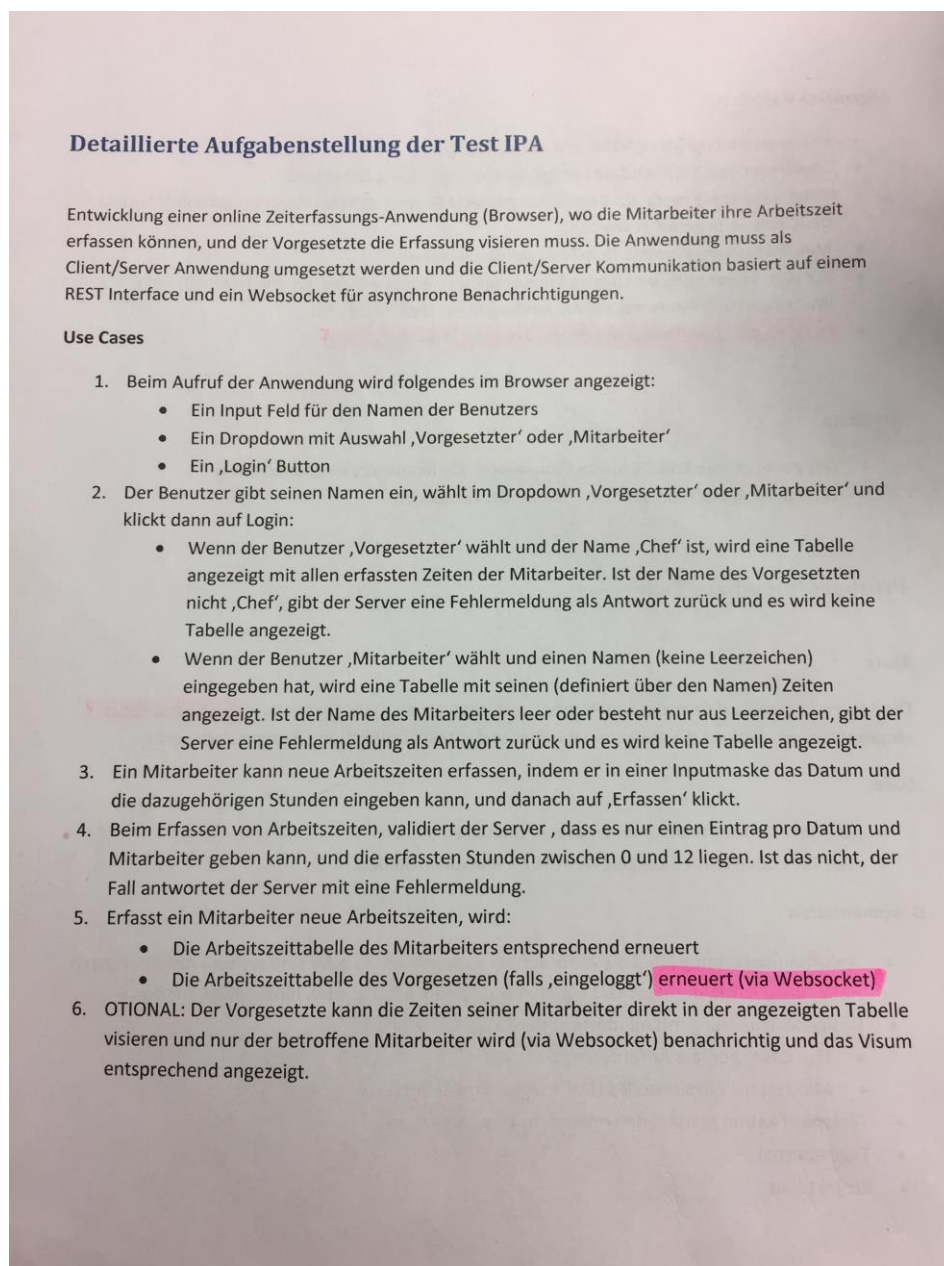


Abbildung 1: Aufgabenstellung Teil 1

Allgemeine Vorgaben

- Implementierung Client: HTML und Javascript/Jquery
- Implementierung Server: Java mit Spring Boot und MySQL Datenbank.
- Input Validierung muss auf dem Server gemacht werden. Optional kann sie zusätzlich auch auf der Client-Seite gemacht werden.
- Mindestens zwei sinnvolle Unit-Tests auf der Serverseite
- Auf dem Server muss ein Logging-Framework verwendet werden, welche Informationen, Warnungen und Fehler aus der Anwendung in ein Logfile schreibt.
- Es sollen die Code-Format Regeln des Polyalert Projektes gelten

Resultate

- Das ganze Eclipse Projekt soll im Subversion Code Repository eingchecked sein
- Lauffähige Applikation auf dem Rechner des Lernenden

Prüfbare/messbare Ziele**Tests**

Die Anwendung soll auf dem eigenen Rechner getestet werden. Dabei soll gemäss Testspezifikation vorgegangen werden und die gefundenen Fehler in der Testspezifikation dokumentiert werden.

Code

- Der Java Code enthält Kommentare an sinnvollen Stellen.
- Der Java Code enthält Logging Aufrufe an sinnvollen Stellen.

Dokumentation

- Anforderungskatalog (Sind die Anforderung zu wenig genau oder sind weitere Anforderungen nötig, dann sollen diese gemäss eigenen Gutdünken ergänzt bzw. hinzugefügt werden.)
- Designbeschreibung mit mindestens dem folgenden Inhalt:
 - Use Cases gemäss Anforderungskatalog
 - Mindestens ein sinnvolles UML Diagramm der Serverseite
- Testspezifikation gemäss den dokumentierten Use Cases.
- Tagesjournal
- Projektplan

Abbildung 2: Aufgabenstellung Teil 2

2 Projektorganisation

In der Projektorganisation sind die am Projekt beteiligten Personen sowie die verwendete Projektmanagement-Methode beschrieben.

2.1 Beteiligte Personen

Die am Projekt beteiligten Personen:

Position	Name	Telefon	E-Mail
Kandidat	Marc Keller	076 461 40 22	marc7keller@gmail.com
Fachvorgesetzter	Marcel Vogt	058 702 14 54	marcel.vogt@atos.net
Hauptexperte			
Zweit-Experte			
Validexperte			

Tabelle 2: Beteiligte Personen

2.2 Projektmanagement-Methode

Die Evaluierung, Entscheidung und Beschreibung der Projektmanagement-Methode.

Als mögliche Projektmanagement-Methoden kämen IPERKA und Scrum in Frage. Während bei IPERKA die Teilschritte sehr klar definiert sind, wird dies bei Scrum eher vernachlässigt.

2.2.1 Scrum

In Scrum arbeitet man in sogenannten Sprints. Das heisst, man definiert sich jeweils Tasks für eine gewisse Zeitspanne, bearbeitet diese in dieser Zeit und kontrolliert nach Abschluss des Sprints ob die Ziele erreicht wurden. Anschliessend wird wieder der nächste Sprint geplant usw.

Bei Scrum ist kein klarer Projektablauf gegeben und der Projektablauf muss somit vollständig selbst in die Hand genommen werden. Dennoch wird beispielsweise bei Atos sehr häufig auf diese Methode gesetzt, da Scrum vor allem auch für die Projektarbeit im Team sehr gut geeignet ist.

2.2.2 IPERKA

Im Gegensatz zu Scrum ist bei IPERKA der Projektablauf klar ersichtlich.

Folgende Phasen werden mit IPERKA durchlaufen:

Informieren	→	Der Auftrag wird geklärt und alle notwendigen Informationen werden gesammelt.
Planen	→	Das Ziel wird definiert, ein Lösungsweg bestimmt und ein Arbeits- und Zeitplan erstellt.
Entscheiden	→	Lösungs-Varianten werden besprochen, festgehalten und evaluiert.
Realisieren	→	Der Auftrag bzw. das Projekt wird umgesetzt.
Kontrollieren	→	Planung und Umsetzung wird verglichen, eine Qualitätskontrolle durchgeführt, Meilensteine überprüft und Tests durchgeführt.
Auswerten	→	Das Projekt wird reflektiert, wurde das Ziel erreicht, wie wurde gearbeitet, wie war die Zusammenarbeit bei der Arbeit im Team und welche neuen Erkenntnisse konnten gewonnen werden.

2.2.3 Gewählte Projektmanagement-Methode

Schliesslich habe ich mich für IPERKA entschieden, da Scrum vor allem für die Arbeit im Team gedacht ist und bei IPERKA der Projektablauf genau definiert ist. Da ich dieses Projekt alleine erarbeite und IPERKA grossen Wert auf die Struktur und die Planung eines Projekts legt, finde ich, dass IPERKA für dieses Projekt eher geeignet ist.

Um optimal nach IPERKA arbeiten zu können, wird das IPERKA-Prinzip auch in der Zeitplanung sowie im 2. Teil der Dokumentation angewandt.

3 Vorkenntnisse

Eine Auflistung der Vorkenntnisse anhand der getätigten Arbeiten auf Bezug der in diesem Projekt verwendeten Technologien.

3.1 Allgemeine Kenntnisse

Aktuelle Kenntnisse über die im Projekt verwendeten Technologien:

- Grundlagen Java
- Geringe Maven-Kenntnisse
- Grundlagen MySQL
- Grundlagen HTML/CSS
- Geringe jQuery-Kenntnisse
- Spring Framework
- Datenbankankbindung in Java
- Grundlagen Web-Socket in Java
- Grundlagen REST-Services in Java
- Grundlagen Unit-Tests mit JUnit und Mockito
- Geringe Kenntnisse mit Logging in Java

3.2 Tätigkeiten

Vorwärtige Tätigkeiten auf Bezug der im Projekt verwendeten Technologien:

- Auffrischen der Java-Kenntnisse
- Einlesen in die Web-Socket-Technologie
- Einlesen in die REST-Technologie
- Einlesen in Unit-Tests mit JUnit und Mockito
- Einlesen in Logging in Java
- Nutzen von Maven als Build-Management-Tool
- Implementation einer kleinen Java-Applikation mit REST-Service mithilfe des Spring-Frameworks
- Implementation einer kleinen Java-Applikation mit Web-Socket mithilfe des Spring-Frameworks
- Einbinden einer MySQL-Datenbank in Java mithilfe des Spring Frameworks
- Erstellen von einfachen Webseiten mit HTML/CSS
- Erstellen von kleineren Webapplikationen mit jQuery

4 Vorarbeiten

Eine kurze Beschreibung über die bereits getätigten Vorarbeiten für das Projekt.
--

Die Entwicklungsumgebung für das Projekt wurde eingerichtet, ich habe mich nochmals in die Thematik Web-Socket und REST-Service in Java mit dem Spring-Framework eingelesen und kleine Beispiel-Applikationen implementiert. Die Grundstruktur der Dokumentation habe ich ebenfalls bereits im Vorfeld erstellt.

5 Firmenstandards

Informationen über verwendete Firmenstandards.
--

Die Firma stellt keine besonderen Anforderungen an das Projekt. Die Dokumentation wurde eigens nach eigenen Vorstellungen umgesetzt und auch die eingesetzten Tools konnten selbst bestimmt werden.

Die einzige Anforderung an mein Projekt war es, nach den Code-Format Regeln des Polyalert-Projektes zu arbeiten.

6 Zeitplan

Die Zeitplanung beinhaltet den Zeitplan als GANTT-Diagramm sowie die gesetzten Meilensteine.

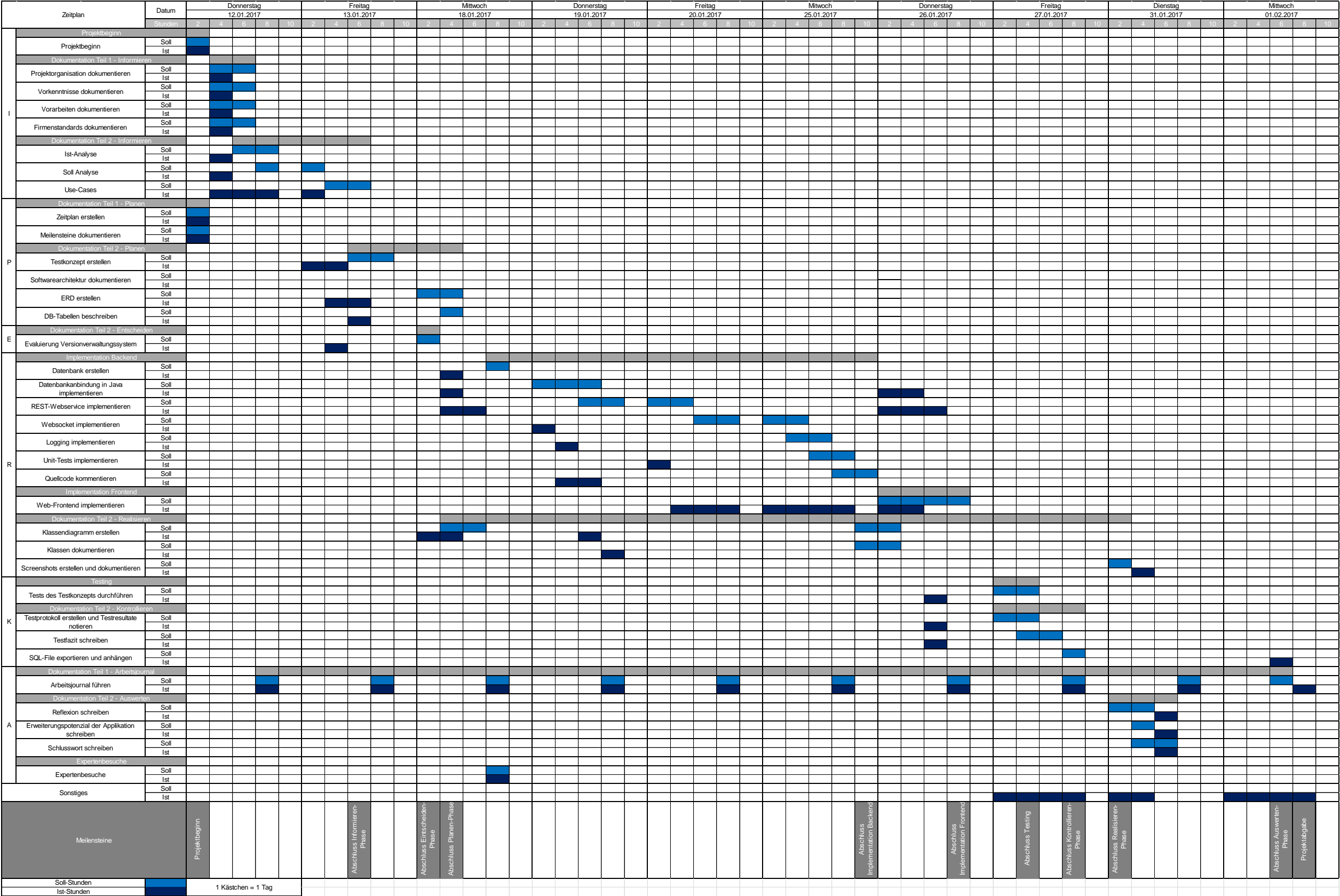


Abbildung 3: GANTT-Zeitplan

6.1 Meilensteile

Meilenstein	Beschreibung	Datum
Projektbeginn	Die letzten Dinge sind geklärt und das Projekt wurde begonnen.	Donnerstag, 12.01.2017
Projektübergabe	Das Projekt ist abgeschlossen und wird somit abgegeben.	Mittwoch, 01.02.2017
IPERKA-Phasen		
Abschluss Informieren-Phase	Die IPERKA Projektphase „Informieren“ ist abgeschlossen und alle zugehörigen Tasks sind erledigt.	Freitag, 13.01.2017
Abschluss Planen-Phase	Die IPERKA Projektphase „Planen“ ist abgeschlossen und alle zugehörigen Tasks sind erledigt.	Mittwoch, 18.01.2017
Abschluss Entscheiden-Phase	Die IPERKA Projektphase „Entscheiden“ ist abgeschlossen und alle zugehörigen Tasks sind erledigt.	Mittwoch, 18.01.2017
Abschluss Realisieren-Phase	Die IPERKA Projektphase „Realisieren“ ist abgeschlossen und alle zugehörigen Tasks sind erledigt.	Dienstag, 31.01.2017
Abschluss Kontrollieren-Phase	Die IPERKA Projektphase „Kontrollieren“ ist abgeschlossen und alle zugehörigen Tasks sind erledigt.	Freitag, 27.01.2017
Abschluss Auswerten-Phase	Die IPERKA Projektphase „Auswerten“ ist abgeschlossen und alle zugehörigen Tasks sind erledigt.	Mittwoch, 01.02.2017
Implementation		
Abschluss Implementation Backend	Die Implementation des Backends ist abgeschlossen.	Mittwoch, 25.01.2017
Abschluss Implementation Frontend	Die Implementation des Frontends ist abgeschlossen.	Donnerstag, 26.01.2017

Tabelle 3: Meilensteine

7 Arbeitsprotokoll

Dieses Kapitel enthält die Arbeitsprotokolle der 10 Arbeitstage.
Jedes Arbeitsprotokoll enthält die getätigten Arbeiten inkl. der Soll- /Ist-Zeit sowie eine kurze Beschreibung. Ausserdem enthält jedes Arbeitsprotokoll auch eine kurze Reflexion über den jeweiligen Arbeitstag.

7.1 Tag 1 – Donnerstag, 12.01.2017

Nr.	IPERKA-Phase	Arbeit	Soll-Zeit	Ist-Zeit	Abgeschlossen ?
Geplante Arbeiten					
1	Informieren	Projektbeginn	1h	15min	<input checked="" type="checkbox"/>
2	Planen	Zeitplan erstellen	1 1/2h	1h	<input checked="" type="checkbox"/>
3	Planen	Meilensteine dokumentieren	20min	15min	<input checked="" type="checkbox"/>
4	Informieren	Projektorganisation dokumentieren	20min	5min	<input checked="" type="checkbox"/>
5	Informieren	Vorkenntnisse dokumentieren	20min	10min	<input checked="" type="checkbox"/>
6	Informieren	Vorarbeiten dokumentieren	20min	10min	<input checked="" type="checkbox"/>
7	Informieren	Firmenstandards dokumentieren	20min	5min	<input checked="" type="checkbox"/>
8	Informieren	Ist-Analyse	30min	20min	<input checked="" type="checkbox"/>
9	Informieren	Soll-Analyse	45min	30min	<input checked="" type="checkbox"/>
10	Auswerten	Arbeitsjournal führen	1 1/2h	40min	<input checked="" type="checkbox"/>
Zusätzlich erledigte Arbeiten					
10	Planen	Use-Cases	4h	3 1/2h	<input checked="" type="checkbox"/>

Tabelle 4: Arbeitsprotokoll - Tag 1 - Donnerstag, 12.01.2017

7.1.1 Bemerkungen

Heute habe ich einiges geschafft, ich konnte alle für heute geplanten Arbeiten erledigen und konnte sogar mit den eigentlich erst für morgen geplanten Use-Cases beginnen. Dies aus diesem Grund, dass ich bei jedem Task etwas an Zeit einsparen konnte und diese deshalb dazugewonnen habe. Ein weiterer Grund ist wohl, dass ich bei der Zeitplanung wohl etwas grosszügig war. Mit den Use-Cases bin ich nun soweit, dass ich alle Use-Case-Diagramme erstellt habe und auch schon von den ersten zwei Use-Case-Diagrammen alle Fälle beschrieben habe, für das letzte werde ich wohl auch nicht mehr allzu viel Zeit aufwenden müssen. Dieser Task ist gleich für morgen früh vorgesehen.

7.1.2 Erfolge

Ein klarer Erfolg war es heute, dass ich alle für heute geplanten Tasks in weniger als der geplanten Zeit abschliessen konnte und aus diesem Grund sogar zusätzlich das Definieren aller Use-Cases schon fast abschliessen konnte.

7.1.3 Misserfolge

Wirkliche Misserfolge gab es heute glücklicherweise nicht, jedoch musste ich den Zeitplan mehrmals überarbeitet, da ich Mühe hatte die Zeiten einzuschätzen. Scheinbar war ich nun etwas grosszügig, jedoch hoffe ich, dass die für die Implementation vorgesehene Zeit ebenfalls eingehalten werden kann.

7.1.4 Ziele

Diesen grandiosen Start möchte ich nun ausnutzen und weiterhin so effizient arbeiten, denn so kann ich evtl. noch mehr Zeit einsparen die ich später noch zur Verfügung habe um die Zeit für unerwartete Probleme decken zu können.

7.2 Tag 2 – Freitag, 13.01.2017

Nr.	IPERKA-Phase	Arbeit	Soll-Zeit	Ist-Zeit	Abgeschlossen?
Geplante Arbeiten					
1	Informieren	Soll-Analyse	-	-	✓
2	Informieren	Use-Cases	1h	35min	✓
3	Planen	Testkonzept erstellen	1 1/2h	1h 40min	✓
4	Auswerten	Arbeitsjournal führen	1 1/2h	45min	✓
Zusätzlich erledigte Arbeiten					
5	Entscheiden	Evaluierung Versionverwaltungssystem	1h	45min	✓
6	Planen	ERD erstellen	1h	1h	✓
7	Planen	DB-Tabellen beschreiben	30min	20min	✓

Tabelle 5: Arbeitsprotokoll - Tag 2 - Freitag, 13.01.2017

7.2.1 Bemerkungen

Am heutigen Tag bin ich wieder ziemlich vorwärts gekommen und bin immer noch dem Zeitplan voraus. So konnte ich heute zusätzlich zu meinen geplanten Tasks das Versionverwaltungssystem evaluieren, das ERM erstellen und die zugehörigen Tabellen dokumentieren. Die Use-Cases musste ich heute nur noch kurz abschliessen, dafür benötigte ich nur noch rund eine halbe Stunde. Für das Testkonzept benötigte ich zwar etwas mehr Zeit als angenommen, jedoch ist das nicht weiter problematisch.

7.2.2 Erfolge

Ich konnte meinen Vorsprung zwar nicht gross weiter ausbauen, jedoch bin ich sehr gut im Zeitplan und konnte bereits einiges Tasks erledigen, die erst in der nächsten Woche geplant waren. Nun kann ich womöglich nächste Woche am Mittwoch bereits einen Tag früher mit der Implementation beginnen als geplant. Ausserdem habe nun ein gutes Zeitkontingent um unvorhersehbare Zeitaufwände decken zu können.

7.2.3 Misserfolge

Zwar bin ich weiterhin gut im Zeitplan, jedoch hätte ich heute etwas produktiver sein können, denn je länger ich arbeitete desto mehr liess meine Produktivität nach. Da ich feststellte wie weit ich bereits fortgeschritten war, nahm ich mir auch noch vor, die erste Version des Klassendiagramms umzusetzen. Ich zeichnete mir danach zwar ein Modell auf, wie die spätere Architektur in etwas aussehen könnte, jedoch konnte ich mich nicht mehr richtig konzentrieren und gab es aus diesem Grund schnell auf.

7.2.4 Ziele

Da ich heute das Klassendiagramm und die Softwarearchitektur nicht mehr schaffte oder besser gesagt erst einmal zur Seite schob, möchte ich nächsten Mittwoch direkt damit beginnen, mir die Software-Architektur genauer zu überlegen und anschliessend auch das Klassendiagramm umzusetzen. Danach kann ich sogar bereits mit der Implementation beginnen.

7.3 Tag 3 – Mittwoch, 18.01.2017

Nr.	IPERKA-Phase	Arbeit	Soll-Zeit	Ist-Zeit	Abgeschlossen?
Geplante Arbeiten					
1	Planen	ERD erstellen§	-	-	✓
2	Planen	DB-Tabellen beschreiben	-	-	✓
3	Entscheiden	Evaluierung Versionverwaltungssystem	-	-	✓
4	Realisieren	Klassendiagramm erstellen	1 1/2h	2h	✗
5	Planen	Datenbank erstellen	20min	5min	✓
6	Auswerten	Expertenbesuch	30min	10min	✓
7	Auswerten	Arbeitsjournal führen	1h	20min	✓
Zusätzlich erledigte Arbeiten					
8	Realisieren	Datenbankanbindung in Java implementieren	4h	1h 15min	✓
9	Realisieren	REST-Webservice implementieren	5h	1h 30min	✓

Tabelle 6: Arbeitsprotokoll - Tag 3 - Mittwoch, 18.01.2017

7.3.1 Bemerkungen

Heute konnte ich wie letzte Woche vorausgesehen, bereits mit der Implementation beginnen und bin weiterhin dem Zeitplan voraus. Am Morgen habe ich mir nochmals die geplante Architektur aufgezeichnet und ein erster Entwurf des Klassendiagramms erstellt und anschliessend die Datenbank erstellt. Danach habe ich mit der Implementation des Backends begonnen, die Datenbank angebunden und den REST-Service implementiert. Dies ist abgesehen von kleineren Problemen, die ziemlich schnell gelöst werden konnten, praktisch reibungslos abgelaufen.

7.3.2 Erfolge

Ich bin heute wieder sehr zufrieden mit meiner Leistung und konnte meinen Vorsprung bezüglich des Zeitplans weiter ausbauen. Die Implementation war eigentlich erst für Morgen geplant, doch aufgrund meines momentanen Vorsprungs konnte ich heute bereits mit der Implementation beginnen und die Realisierung der Datenbankanbindung und des REST-Services schon heute soweit abschliessen. Um das Ganze anschliessend auf die Funktion zu testen, habe ich einige Tests mit SoapUI durchgeführt und somit Webservice und Datenbankanbindung gleichzeitig getestet. Dies ist jedoch kein Ersatz für die Unit- und Blackboxtests, sondern war lediglich dazu gedacht, zu prüfen ob auch alles funktioniert, bevor ich mit der weiteren Implementation fortfahren.

7.3.3 Misserfolge

Am Morgen hatte ich etwas Anlaufschwierigkeiten und konnte nicht gleich mit vollem Einsatz wieder starten, da ich mich zuerst wieder ein wenig einarbeiten musste und deswegen ein wenig Zeit verloren habe.

7.3.4 Ziele

Meine Ziele sind es nun als nächstes mit der Implementation des WebSockets zu beginnen, das Klassendiagramm nochmals anzupassen und alle Klassen näher zu beschreiben. Zusätzlich will ich natürlich meinen Zeitvorsprung nach Möglichkeit weiter beibehalten, um für unvorhersehbare Probleme im späteren Projektverlauf gerüstet zu sein. Dazu möchte ich weiterhin schnell und effizient weiterarbeiten, damit mir später genügend Zeit bleibt, für die Implementation des Frontends und auch für weitere kleinere Anpassungen.

7.4 Tag 4 – Donnerstag, 19.01.2017

Nr.	IPERKA-Phase	Arbeit	Soll-Zeit	Ist-Zeit	Abgeschlossen?
Geplante Arbeiten					
1	Realisieren	Datenbankanbindung in Java	-	-	<input checked="" type="checkbox"/>
2	Realisieren	REST-Webservice implementieren	-	-	<input checked="" type="checkbox"/>
3	Auswerten	Arbeitsjournal führen	45min	30min	<input checked="" type="checkbox"/>
Zusätzlich erledigte Arbeiten					
4	Realisieren	Websocket implementieren	2h	45min	<input checked="" type="checkbox"/>
5	Realisieren	Logging implementieren	1 1/2h	1h	<input checked="" type="checkbox"/>
6	Realisieren	Quellcode kommentieren	45min	50min	<input checked="" type="checkbox"/>
7	Realisieren	Klassendiagramm erstellen	30min	10min	<input checked="" type="checkbox"/>
8	Realisieren	Klassen dokumentieren	45min	55min	<input checked="" type="checkbox"/>

Tabelle 7: Arbeitsprotokoll - Tag 4 - Donnerstag, 19.01.2017

7.4.1 Bemerkungen

Heute war wieder ein ziemlich erfolgreicher Tag, er hat damit begonnen, dass ich mit der Implementation vom Web-Socket begonnen habe. Diese verlief so reibungslos, dass ich sie nach 45min bereits soweit abgeschlossen habe. Da ich mir 2h dafür eingeplant hatte, hatte ich diese Zeit nun anderweitig zur Verfügung. Als nächsten Schritt ging ich dann das Logging an, dieses hatte ich nach einer Stunde statt den eineinhalb Stunden fertig implementiert. Danach kommentierte ich den bis jetzt stehenden Code vollständig, damit dieser auch leicht verständlich ist und da dies auch eine Anforderung an das Projekt ist. Zuletzt arbeitete ich noch am Klassendiagramm weiter und dokumentierte die einzelnen Klassen noch etwas ausführlicher.

7.4.2 Erfolge

Wie bereits gesagt, war der heutige Tag im grossen und ganzen wieder recht erfolgreich. Natürlich hatte ich gestern wieder einiges an Zeit dazugewonnen und habe sogar alle Tasks geschafft die eigentlich für heute geplant waren, jedoch konnte ich vor allem am Morgen wieder sehr schnell und effektiv arbeiten wodurch ich wieder viel erledigen konnte, auch wenn noch einige administrative Dinge dazwischen kamen.

7.4.3 Misserfolge

Etwas schade war, dass meine Effektivität und Produktivität am Nachmittag ziemlich nachliess, ich war ziemlich unkonzentriert und habe an vielen Dingen parallel und kreuz und quer weitergearbeitet. Dies ist nicht sehr arbeitsfördernd und raubte mir einiges an Zeit, welche ich sinnvoller hätte einsetzen können.

7.4.4 Ziele

Was meine nächsten Tasks betrifft, möchte ich morgen Vormittag gerne mit der Implementation der Unit-Tests beginnen und diese abschliessen. Danach möchte ich nochmals das Klassendiagramm studieren und falls nötig noch einige Anpassungen daran vornehmen. Anschliessend möchte ich ebenfalls nochmals durch die Dokumentation der einzelnen Klassen durchgehen und gegebenenfalls noch einige Dinge ergänzen. Im Anschluss, wenn alles glatt läuft, möchte ich bereits mit der Implementation des Frontends beginnen und damit möglichst weit kommen.

Abgesehen von den zu erledigenden Tasks, ist es mein Ziel meine Produktivität und Effektivität im Arbeiten über den ganzen Tag hinaus zu halten. Falls ich einmal Mühe habe mich zu konzentrieren oder falls ich sonst nicht weiterkomme, gehe ich lieber kurz Luft schnappen, um nachher wieder optimal und mit voller Energie weiterarbeiten zu können.

7.5 Tag 5 – Freitag, 20.01.2017

Nr.	IPERKA-Phase	Arbeit	Soll-Zeit	Ist-Zeit	Abgeschlossen?
Geplante Arbeiten					
1	Realisieren	REST-Webservice implementieren	-	-	<input checked="" type="checkbox"/>
2	Realisieren	Websocket implementieren	-	-	<input checked="" type="checkbox"/>
3	Auswerten	Arbeitsjournal führen	45min	40min	<input checked="" type="checkbox"/>
Zusätzlich erledigte Arbeiten					
4	Realisieren	Unit-Tests implementieren	2h	1 1/2h	<input checked="" type="checkbox"/>
5	Realisieren	Web-Frontend implementieren	4h	4 1/2h	<input type="checkbox"/>

Tabelle 8: Arbeitsprotokoll - Tag 5 - Freitag, 20.01.2017

7.5.1 Bemerkungen

Alle für den heutigen Tag nach Zeitplan geplanten Tasks konnte ich ja bereits gestern erledigen. Aus diesem Grund konnte ich heute bereits mit den Tasks vom nächsten Tag beginnen.

Dazu begann ich wie gestern vorgenommen, direkt heute Vormittag mit der Implementation der Unit-Tests. Zuerst überlegte ich mir welche Unit-Tests am meisten Sinn ergeben würden und habe mich schlussendlich dafür entschieden, die Login-Methode des Webservices mit allen möglichen Fällen durchzutesten. Dazu habe ich insgesamt 5 Testmethoden implementiert, die die Fälle abdecken, wenn sich der Vorgesetzte mit gültigen und ungültigen Daten anmeldet und wenn der Benutzer sich mit gültigen und ungültigen Daten anmeldet, wenn er bereits als User in der Datenbank existiert oder eben noch nicht existiert. Um diese Tests möglich zu machen, habe ich JUnit und das Mocking-Framework „Mockito“ verwendet, da ich die datenbankspezifischen Methoden mocken musste. Die restlichen Klassen und Methoden habe ich (noch) nicht getestet, da es in diesem Fall entweder nur wenig Sinn ergeben hätte bzw. ziemlich umständlich und problematisch gewesen wäre, da man bspw. um den Web-Socket zu testen, zusätzlich eine Web-Socket-Verbindung vom Server selber herstellen müsste.

Die restlichen freien Stunden des heutigen Tages, habe ich der Implementation des Frontends gewidmet. Diese ist momentan aber noch nicht abgeschlossen und wird nächste Woche noch einige Stunden in Anspruch nehmen.

7.5.2 Erfolge

Dadurch dass ich gestern bereits alle Tasks für heute erledigen konnte, konnte ich bereits mit den Tasks von nächster Woche beginnen. Nun bin ich einen ganzen Tag im Vorsprung und kann diese gewonnene Zeit in noch auftretende Probleme oder sinnvolle Erweiterungen und Anpassungen an der Applikation stecken.

7.5.3 Misserfolge

Zwar bin ich sehr zügig in der Umsetzung der Applikation jedoch ist mir schon mehrmals aufgefallen, dass ich teilweise etwas zu ungeduldig bin und tausende Dinge gleichzeitig implementieren möchte. Dies kann gelingen jedoch auch völlig in die Hose gehen. Und deshalb muss ich in den nächsten Tagen darauf achten, strukturierter vorzugehen und mich einem Task nach dem anderen zu widmen.

7.5.4 Ziele

Wie erwähnt muss ich in Zukunft etwas strukturierter vorgehen und mich auf eine Sache anstatt mehrere Sachen gleichzeitig konzentrieren. Dadurch arbeite ich konzentrierter und fokussierter und es wird auch der Qualität der Arbeit zu Gute kommen.

Was die nächsten Tasks betrifft, möchte ich nächste Woche zuerst die Implementation des Frontends abschliessen. Dies wird noch einige Zeit in Anspruch nehmen weshalb dies der einzige Task ist, den ich mir für nächsten Mittwoch vornehme. Wenn die Implementation abgeschlossen ist, werde ich mich dem Testing widmen.

7.6 Tag 6 – Mittwoch, 25.01.2017

Nr.	IPERKA-Phase	Arbeit	Soll-Zeit	Ist-Zeit	Abgeschlossen?
Geplante Arbeiten					
1	Realisieren	Web-Socket implementieren	-	-	<input checked="" type="checkbox"/>
2	Realisieren	Logging implementieren	-	-	<input checked="" type="checkbox"/>
3	Realisieren	Unit-Tests implementieren	-	-	<input checked="" type="checkbox"/>
4	Realisieren	Quellcode kommentieren	-	-	<input checked="" type="checkbox"/>
5	Realisieren	Klassendiagramm erstellen	-	-	<input checked="" type="checkbox"/>
6	Realisieren	Klassen dokumentieren	-	-	<input checked="" type="checkbox"/>
7	Auswerten	Arbeitsjournal führen	45min	30min	<input checked="" type="checkbox"/>
Zusätzlich erledigte Arbeiten					
8	Realisieren	Web-Frontend implementieren	7h	6h	<input checked="" type="checkbox"/>

Tabelle 9: Arbeitsprotokoll - Tag 6 - Mittwoch, 25.01.2017

7.6.1 Bemerkungen

Heute bin ich mit der Implementation des Frontends weitergefahren. Damit bin ich zwar wieder einiges weitergekommen, muss jedoch morgen wiederum noch einiges an Zeit darin investieren. Die Grundstruktur der Webseite wurde abgeschlossen, ebenfalls funktioniert das Login und das Laden der Daten. Das einzige was momentan noch einige Probleme macht, ist das Erfassen von neuen Arbeitszeiten, zwar wird der Webservice des Backends aufgerufen und die neuen Daten auch in die Datenbank gespeichert, jedoch lädt der Client die Daten nach der Erfassung nicht automatisch neu. Was jedoch auch funktioniert, ist die Benachrichtigung des Vorgesetzten sofern dieser angemeldet ist. Bei diesem werden bei Benachrichtigung auch die Daten automatisch aktualisiert.

7.6.2 Erfolge

Natürlich war ich heute schon einen Tag voraus was den Zeitplan betrifft und konnte somit bereits am Frontend arbeiten, obwohl dieses erst für Morgen geplant wäre. Diese gewonnene Zeit war sehr wichtig, wie sich im Moment immer mehr herauskristallisiert. So bin ich nun schon einige Zeit länger an der Implementation des Frontends als geplant. Dennoch sind bisher noch keine schwerwiegenden Probleme eingetreten und konnten falls es einmal Probleme gab ziemlich schnell gelöst werden.

7.6.3 Misserfolge

Zwar ist es positiv dass ich heute bereits mit dem geplanten Task von Morgen beginnen konnte, jedoch habe ich nun mein neu dazugewonnenes Zeitkontingent schon fast aufgebraucht und bin nun wieder gleich dem Zeitplan. Dies bedeutet nun natürlich, dass ich Morgen nochmals richtig Gas geben muss und die Implementation des Frontend möglichst schnell abschliessen muss, damit ich mit dem Testing beginnen kann und mir später auch noch etwas Zeit bleibt um mögliche Bugs zu beheben oder noch einige Anpassungen zu machen.

7.6.4 Ziele

Meine nächsten Ziele sind es nun, die Implementation des Frontends möglichst zeitnah abzuschliessen, damit ich evtl. bereits morgen mit dem Testing beginnen kann. Nichtsdestotrotz liegt mein Fokus momentan noch auf der Implementation.

7.7 Tag 7 – Donnerstag, 26.01.2017

Nr.	IPERKA-Phase	Arbeit	Soll-Zeit	Ist-Zeit	Abgeschlossen?
Geplante Arbeiten					
1	Realisieren	Web-Frontend implementieren	4h	1 1/2h	<input checked="" type="checkbox"/>
2	Auswerten	Arbeitsjournal führen	45min	30min	<input checked="" type="checkbox"/>
Zusätzlich erledigte Arbeiten					
3	Realisieren	Datenbankanbindung in Java implementieren	-	1h 30min	<input checked="" type="checkbox"/>
4	Realisieren	REST-Webservice implementieren	-	1h 45min	<input checked="" type="checkbox"/>
5	Kontrollieren	Tests des Testkonzepts durchführen	30min	35min	<input checked="" type="checkbox"/>
6	Kontrollieren	Testprotokoll erstellen und Testresultate notieren	30min	35min	<input checked="" type="checkbox"/>
7	Kontrollieren	Testfazit schreiben	15min	20min	<input checked="" type="checkbox"/>

Tabelle 10: Arbeitsprotokoll - Tag 7 - Donnerstag, 26.01.2017

7.7.1 Bemerkungen

Am heutigen Tag konnte ich wieder vieles erledigen und wieder einiges an Zeit gewinnen. In den ersten Stunden heute Morgen, konnte ich die Implementation des Frontends soweit abschliessen und die noch vorhandenen Fehler beheben. Und zwar hatte ich das Problem, dass der Client sich nach dem Erfassen von Arbeitszeiten nicht automatisch aktualisierte und stattdessen die Error-Methode des Ajax-Calls aufgerufen wurde. Dies konnte ich nun so beheben, dass ich auf der Serverseite einen Boolean zurückgab und nicht mehr wie vorher einen einfachen HTML-OK-Status. Was mir gestern ebenfalls noch aufgefallen ist, ist es, dass ich die Servervalidierung bei der Erfassung von Arbeitszeiten noch nicht umgesetzt hatte, weshalb ich dies heute noch nachholte.

Am Mittag war dann die gesamte Implementation im grossen und ganzen abgeschlossen und ich konnte nach dem Mittag bereits mit dem Testing beginnen. Die anschliessend durchgeführten Blackbox-Tests waren zum grössten Teil erfolgreich, in einigen wenigen Fällen, waren die Ergebnisse der Tests jedoch nicht wie gewollt und ich musste nochmals nachbessern. Und zwar war es so, dass ich bei der Erfassung neuer Arbeitszeiten bisher nur den Fall abgedeckt hatte, dass die Felder entweder leer waren oder nur Leerzeichen beinhalteten. Den Fall, dass der Benutzer Buchstaben und/oder Sonderzeichen eingibt habe ich jedoch noch nicht abgedeckt, weshalb diese Tests auch nicht erfolgreich waren. Ausserdem hatte ich noch den Fall, dass sich ein Benutzer mit dem Namen „Chef“ anmelden konnte und anschliessend zwar korrekt als Mitarbeiter angemeldet wurde, sich jedoch auch beim Web-Socket registrierte, da er den Namen „Chef“ besitzt, des Weiteren sollte es auch nicht so sein, dass ein Mitarbeiter den Namen „Chef“ haben kann. Ich implementierte also auf dem Server noch die fehlenden Validierungen und führte anschliessend die Tests nochmals durch, welche dann auch alle erfolgreich waren. Ein einziger Punkt, der noch nicht optimal gelöst ist, ist, dass zurzeit wenn der Benutzer ungültige Arbeitszeiten erfasst oder die Arbeitszeit für ein Datum erfasst, für welches bereits die Arbeitszeit erfasst wurde, dieselbe Fehlermeldung ausgegeben wird

und diese nicht spezifisch auf die Fehlereingabe ist. Diesem Punkt werde ich mich evtl. morgen oder in den nächsten Tagen noch widmen. Alles in allem funktioniert jedoch alles.

7.7.2 Erfolge

Mit dem heutigen Tag bin ich sehr zufrieden. Ich habe wieder einiges an Zeit hinzugewonnen, welche ich nun dafür nutzen kann, die Dokumentation nochmals zu überarbeiten, noch einige Verbesserungen und Erweiterungen am Code vorzunehmen und an den einen Ecken und Kanten nochmals etwas zu schleifen. Die Tests verliefen nun ebenfalls alle erfolgreich und falls am Ende noch Zeit bleibt können auch einige Dinge noch weiter auskuriert und optimiert werden.

7.7.3 Misserfolge

Wirkliche Misserfolge gab es heute nicht, ich bin heute sehr gut vorwärts gekommen und habe die zur Verfügung stehende Zeit auch sehr gut genutzt.

7.7.4 Ziele

Nun bin ich so weit, dass alle Funktionen implementiert sind und auch wie gewünscht funktionieren. Die für Morgen geplanten Tests habe ich ebenfalls schon alle durchgeführt und die letzten Fehler noch behoben. Dies bedeutet nun, dass ich die Zeit morgen für Optimierungen am Code und für die Überarbeitung der Dokumentation nutzen kann. Des Weiteren müssen noch die Screenshots der Applikation gemacht werden und das Klassendiagramm und evtl. auch andere Diagramme nochmals überarbeitet werden.

7.8 Tag 8 – Freitag, 27.01.2017

Nr.	IPERKA-Phase	Arbeit	Soll-Zeit	Ist-Zeit	Abgeschlossen?
Geplante Arbeiten					
1	Kontrollieren	Tests des Testkonzepts durchführen	-	-	<input checked="" type="checkbox"/>
2	Kontrollieren	Testprotokoll erstellen und Testresultate notieren	-	-	<input checked="" type="checkbox"/>
3	Kontrollieren	Testfazit schreiben	-	-	<input checked="" type="checkbox"/>
4	Auswerten	Arbeitsjournal führen	45min	30min	<input checked="" type="checkbox"/>
Zusätzlich erledigte Arbeiten					
5	Diverse	Diverse Arbeiten	7h	7h	<input checked="" type="checkbox"/>

Tabelle 11: Arbeitsprotokoll - Tag 8 - Freitag, 27.01.2017

7.8.1 Bemerkungen

Heute habe ich es ziemlich gemütlich genommen, da ich nur noch wenige offene Tasks habe und diese erst für nächste Woche geplant sind und es nur wenig Sinn ergibt mit diesen bereits jetzt zu beginnen. Aus diesem Grund habe ich wie gestern vorgenommen, die Dokumentation und sämtliche Diagramme nochmals überarbeitet und noch einige Anpassungen am Code vorgenommen. Beispielsweise wollte ich die Möglichkeit noch umsetzen, dass bei der Zeiterfassung dem Client spezifischere Fehlermeldungen ausgegeben werden können. Dazu habe ich zuerst versucht anstatt bei Server einen Boolean zurückzugeben, spezifische Strings zurückzugeben, welche der Client anschliessend nur noch ausgeben muss. Als dies jedoch nicht geklappt hatte und beim Client ständig nur die Error-Methode des Ajax-Requests aufgerufen wurde, habe ich mir überlegt, was auch die gängigste und gescheiteste Lösung von allen wäre, einfach eine Exception mit der zugehörigen Fehlermeldung zu werfen, doch als diese Methode wiederum und auch nach langem recherchieren nicht funktionierte, habe ich es aufgegeben und mich den weiteren Dingen gewidmet. Bspw. wollte ich nochmals die Kommentare im Code ergänzen und habe zusätzlich auch noch den Code des Frontends kommentiert. Des Weiteren habe ich noch einige Logs hinzugefügt.

7.8.2 Erfolge

Dass ich heute noch so viele Anpassungen und Verbesserungen an Dokumentation und am Code vornehmen konnte, war nur durch meinen gewonnenen Zeitvorsprung möglich. Zwar habe ich durch meine Problematik mit den Fehlermeldungen bei der Arbeitszeiterfassung viel Zeit verloren, vor allem auch weil ich es am Schluss nicht zum Laufen brachte. Dennoch war es ein grosser Vorteil, heute noch Zeit für solche Dinge gehabt zu haben.

7.8.3 Misserfolge

Ziemlich schade fand ich es, dass ich die Fehlermeldungen nicht zum Funktionieren brachte, denn es ist nicht wirklich von Vorteil, wenn einem Benutzer nur eine unspezifische nichts aussagende Fehlermeldung ausgegeben wird, mit welcher er wenig bis gar nichts anfangen kann. Leider kann ich es jedoch momentan auch nicht ändern und evtl. finde ich ja in den nächsten zwei Tagen nochmals ein wenig Zeit um mich damit auseinanderzusetzen.

7.8.4 Ziele

Meine Ziele sind es nun, alles abzuschliessen, noch die Screenshots der Applikation zu machen und zu beschreiben, die Reflexion und über das Erweiterungspotenzial der Applikation zu schreiben und schliesslich noch den Schlussteil zu schreiben. Falls mir noch Zeit bleibt, kann ich diese nochmals in das Problem mit den Fehlermeldungen investieren, noch eine Eingabvalidierung auf dem Client umsetzen oder was mir sonst noch in den Sinn kommt um die Applikation sinnvoll zu erweitern und zu verbessern.

7.9 Tag 9 – Dienstag, 31.01.2017

Nr.	IPERKA-Phase	Arbeit	Soll-Zeit	Ist-Zeit	Abgeschlossen?
Geplante Arbeiten					
1	Realisieren	Screenshots erstellen und dokumentieren	1h	1h	<input checked="" type="checkbox"/>
2	Auswerten	Reflexion schreiben	20min	10min	<input checked="" type="checkbox"/>
3	Auswerten	Erweiterungspotenzial der Applikation schreiben	20min	30min	<input checked="" type="checkbox"/>
4	Auswerten	Schlusswort schreiben	20min	10min	<input checked="" type="checkbox"/>
5	Auswerten	Arbeitsjournal führen	30min	15min	<input checked="" type="checkbox"/>
Zusätzlich erledigte Arbeiten					
6	Diverse	Diverse Arbeiten an der Dokumentation	4h	4h	

Tabelle 12: Arbeitsprotokoll - Tag 9 - Dienstag, 31.01.2017

7.9.1 Bemerkungen

Heute hatte ich noch die letzten Tasks geplant und zwar einige Screenshots der Applikation zu erstellen und diese näher zu beschreiben und schliesslich noch die Reflexion und das Schlusswort und über das Erweiterungspotenzial der Applikation zu schreiben.

Zu Beginn des heutigen Tages, bin ich zuerst noch einmal durch die Dokumentation gegangen und noch einige Anpassungen gemacht und einige Dinge ergänzt. Anschliessend habe ich mich den Screenshots gewidmet und diese näher dokumentiert. Am Nachmittag habe ich mich dann noch den letzten Punkten der Dokumentation gewidmet, die Reflexion, den Schlussteil und über das Erweiterungspotenzial der Applikation geschrieben.

7.9.2 Erfolge

Heute hatte ich nicht mehr viel zu tun, weshalb ich etwas zurücklehnen konnte und da nur noch kleinere Dinge zu tun waren, waren auch keine grossen Erfolge zu verzeichnen.

7.9.3 Misserfolge

Da ich heute nicht mehr viel zu tun hatte, habe ich es heute ziemlich gemütlich genommen, fast ein wenig zu gemütlich, denn ich hätte die restliche Zeit gut nutzen können, um mich nochmals der Zeiterfassung zu widmen oder vielleicht noch die Validierung auf dem Client oder sonstige Erweiterungen zu implementieren.

7.9.4 Ziele

Morgen habe ich keine Tasks mehr, ausser, noch den Programmcode in die Dokumentation zu übernehmen und das letzte Arbeitsprotokoll zu schreiben.

Die restliche Zeit möchte ich dafür nutzen nochmals die ganze Dokumentation durchzuschauen den ganzen Inhalt nochmals durchzulesen und gegeben falls noch ein wenig anzupassen oder zu ergänzen etc. Wenn mir dann noch Zeit bleibt, kann ich mich ja noch der Eingabevalidierung auf dem Client widmen oder noch etwas am Design des GUIs schrauben.

7.10 Tag 10 – Mittwoch, 01.02.2017

Nr.	IPERKA-Phase	Arbeit	Soll-Zeit	Ist-Zeit	Abgeschlossen?
Geplante Arbeiten					
1	Auswerten	Arbeitsjournal führen	30min	10min	<input checked="" type="checkbox"/>
Zusätzlich erledigte Arbeiten					
2	Diverse	Dokumentation nochmals durchschauen und die letzten Anpassungen machen	1h	1 1/2h	<input checked="" type="checkbox"/>
3	Planen	Software-Architektur in Grafik darstellen und dokumentieren	1 1/2h	2 1/2h	<input checked="" type="checkbox"/>
4	Realisieren	Quellcode in Doku exportieren und in Dokumentation übernehmen	1h	1h	<input checked="" type="checkbox"/>
5	Diverse	Diverse Arbeiten	-	3h	<input checked="" type="checkbox"/>

Tabelle 13: Arbeitsprotokoll - Tag 10 - Mittwoch, 01.02.2017

7.10.1 Bemerkungen

Heute war der letzte Tag und es war Zeit noch die letzten Dinge zu erledigen und das Projekt und die Dokumentation für die Abgabe vorzubereiten um sie dann schliesslich abzugeben. So bin ich am Vormittag nochmals durch die ganze Dokumentation durchgegangen und habe das ein oder andere noch angepasst, ausserdem habe ich nochmals die Rechtschreibung und Grammatik geprüft. Danach habe ich die Systemarchitektur welche ich mir aufgezeichnet hatte nochmals auf dem Computer erstellt, diese in die Dokumentation kopiert und noch etwas näher beschrieben. Anschliessend habe ich die finale Version der Applikation in das Remote-Repository gepusht, damit auf diesem die aktuelle und finale Version vorhanden ist. Im Anschluss habe ich dann den ganzen Quellcode in die Dokumentation kopiert. Schliesslich habe ich noch den Zeitplan fertiggestellt und ebenfalls der Dokumentation angefügt. Nun schreibe ich noch das letzte Arbeitsjournal und ergänze noch das Glossar.

7.10.2 Erfolge

Nun bin ich am Ende des Projektes und bin wie auch schon in der Reflexion beschrieben, sehr zufrieden damit, wie es gelaufen ist und welches Endprodukt daraus entstanden ist. Sehr zufrieden bin ich auch mit meinem Zeitmanagement, denn vor allem jetzt gegen den Schluss hatte ich nur noch wenig zu tun und war das ganze Projekt über meist dem Zeitplan voraus. Ebenfalls fand ich das Projekt sehr hilfreich um mir nochmals meine Stärken und Schwächen aufzuzeigen und mein ganzes Wissen in der Java-Programmierung, die Erstellung von Webservices und Websockets, weiter zu festigen.

7.10.3 Misserfolge

Leider hat meine Produktivität gegen den Schluss immer weiter abgenommen. Natürlich hatte ich auch nicht mehr sonderlich viel zu tun, jedoch gäbe es auch noch einige Erweiterungs- und oder Verbesserungsmöglichkeiten der Applikation und ich hätte diese Zeit definitiv besser nutzen können.

Teil 2: Projekt

Im 2. Teil der Dokumentation wird, nach IPERKA gegliedert, näher auf die Umsetzung des Projekts eingegangen.

In diesem Teil folgen eine technische Analyse des Projekts sowie Informationen über die Planung, die Umsetzung, und das Testing der Applikation. Am Ende dieses Teils ist dann ebenfalls noch die Reflexion über das Projekt zu finden.

1 Zusammenfassung

In diesem Abschnitt folgt eine erneute Kurzzusammenfassung über das Projekt.

1.1 Projektbeschreibung

Die Projektbeschreibung gibt eine allgemeine Übersicht über das Projekt.

Anhand von diesem Projekt, soll eine Webapplikation zur Zeiterfassung von Mitarbeitern umgesetzt werden. Einem Mitarbeiter soll es möglich sein, sich mit seinem Namen anzumelden und anschliessend die Zeit für ein bestimmtes Datum zu erfassen. Dem Vorgesetzten soll es möglich sein, sich als Chef anzumelden und die erfassten Zeiten der Mitarbeiter einzusehen. Wenn ein Mitarbeiter neue Arbeitszeiten erfasst, während der Vorgesetzte angemeldet ist, soll sich anschliessend die Tabelle beim Vorgesetzten automatisch aktualisieren.

1.2 System-Beschreibung

Die Systembeschreibung gibt Auskunft über den technischen Aufbau des Projekts.

Diese Webapplikation soll folgendermassen umgesetzt werden:

Die Applikation soll ein Java-Backend besitzen, welches an eine MySQL-Datenbank angebunden ist und einen REST-Service und einen Web-Socket zur Verfügung stellt, dazu soll das Spring-Framework verwendet werden. Als Frontend soll eine Weboberfläche mit HTML/CSS und JavaScript/jQuery umgesetzt werden auf welche die Benutzer zugreifen können.

2 Informieren

Als ersten Schritt nach IPERKA, sollen alle Informationen über das Projekt gesammelt werden. Dazu gehören, was ist bereits vorhanden, was wird benötigt, was sind die Anforderungen etc.

2.1 Ist-Analyse

In der Ist-Analyse wird das vorhandene System analysiert.

2.1.1 Hardware

Als Entwicklungsgerät und gleichzeitiger Testserver der Applikation, steht ein Lenovo P70 Notebook zur Verfügung welches mit Windows 7 läuft.

2.1.2 Entwicklungsumgebung

Als Entwicklungsumgebung wird Eclipse Neon eingesetzt, welches bereits installiert und konfiguriert ist.

2.1.3 Java JDK

Das Java Development Kit in Version 1.8.0(102) ist installiert und der Pfad in einer JAVA_HOME Variable gespeichert.

2.1.4 Maven

Eclipse beinhaltet ebenfalls mein angewandtes Build-Management-Tool Maven.

2.1.5 MySQL

MySQL ist als Bestandteil von XAMPP installiert und über phpmyadmin konfigurierbar.

2.1.6 Tomcat-Server

Ein Tomcat-Server zum «deployen» der Applikation ist Bestandteil von XAMPP sowie auch nativ in Eclipse vorhanden.

2.1.7 Internetbrowser

Als Internetbrowser sind neben dem vorinstallierten Internet Explorer, Google Chrome und Firefox installiert.

2.2 Soll-Analyse

In der Soll-Analyse werden die Anforderungen genauer analysiert.

2.2.1 GUI / Frontend

Das GUI bzw. Frontend der Web-Applikation, soll mit HTML/CSS, JavaScript und jQuery umgesetzt werden.

Wenn der Benutzer die Webseite aufruft, soll ein Eingabefeld für den Benutzernamen bzw. den Namen des Mitarbeiters, eine Dropdown-Auswahl mit den Optionen „Vorgesetzter“ und „Mitarbeiter“, sowie einen „Login“-Button angezeigt werden.

Wenn sich der Benutzer als Mitarbeiter und mit seinem Namen anmeldet, soll eine Tabelle mit den erfassten Zeiten dieses Mitarbeiters angezeigt werden. Ist das Eingabefeld mit dem Namen des Mitarbeiters leer oder besteht dieses nur aus Leerzeichen, sendet der Server eine Fehlermeldung zurück und es soll keine Tabelle angezeigt werden.

Wenn sich der Benutzer als Vorgesetzten und mit dem Namen „Chef“ anmeldet, soll eine Tabelle angezeigt werden mit den erfassten Zeiten aller Mitarbeiter. Ist der eingegebene Name des Vorgesetzten nicht „Chef“, sendet der Server eine Fehlermeldung zurück und es soll keine Tabelle angezeigt werden.

Dem Mitarbeiter soll ausserdem eine Eingabemaske zur Verfügung stehen, in welcher er neue Arbeitszeiten erfassen an. In dieser Maske sollen zwei Eingabefelder für das Datum und die Zeit, sowie ein „Erfassen“-Button angezeigt werden. Erfasst ein Mitarbeiter neue Zeiten, so soll die Tabelle dieses Mitarbeiters und des Vorgesetzten, falls dieser angemeldet ist, automatisch aktualisiert werden.

2.2.2 Java-Backend

Als Backend fungiert eine Java-Applikation. Diese soll zum einen einen REST-Service zur Verfügung stellen, über welchen sich der Mitarbeiter anmeldet und neue Zeiten erfasst. Um bei einer Zeiterfassung eines Mitarbeiters, die Tabelle des Vorgesetzten, falls dieser angemeldet ist, automatisch aktualisieren zu können, soll das Java-Backend ebenfalls einen Web-Socket bereitstellen. Ausserdem müssen die erhaltenen Daten des Frontends validiert werden.

Weitere Anforderungen an das Java-Backend sind es, mind. zwei sinnvolle Unit-Tests zu implementieren, Informationen, Warnungen und Fehler zu loggen, die Code-Format-Regeln des Polyalert Projekts zu verwenden und den Code sinnvoll zu kommentieren.

2.2.3 MySQL-Datenbank

Im Backend soll ebenfalls eine MySQL-Datenbank arbeiten, welche die Namen der Mitarbeiter und deren Zeiten beinhaltet.

2.3 Use-Cases

Die benötigten Teilfunktionen in Use-Cases dargestellt.

2.3.1 Use-Case 1: Login

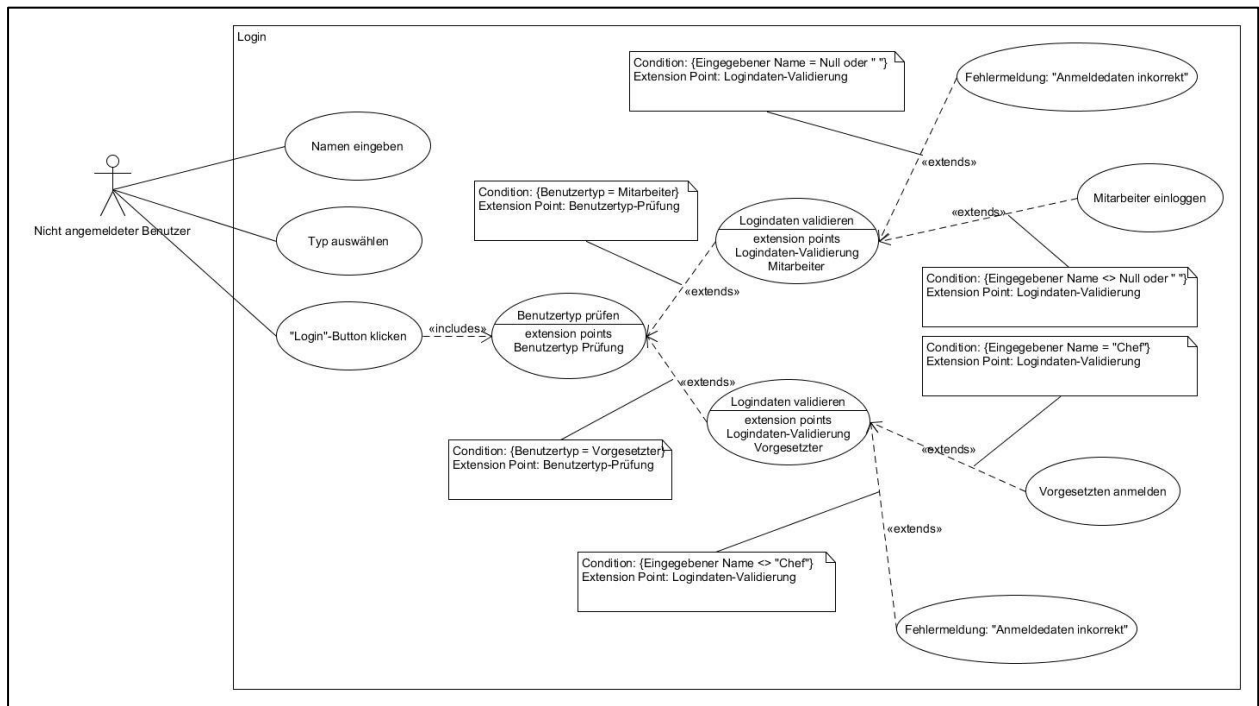


Abbildung 4: Use-Case-Diagramm 1 – Login

Use-Case 1.1: Login – Korrekte Eingaben, Vorgesetzter			
Beschreibung	Der Benutzer hat die Möglichkeit sich anzumelden.		
Akteure/Aktoren	Nicht angemeldeter Benutzer		
Eintrittsfall	Der Benutzer befindet sich auf der Login-Seite.		
Vorbedingungen	Der Benutzer ist ausgeloggt.		
Ergebnis / Nachbedingungen	Der Benutzer wird als Vorgesetzter angemeldet und landet in der Arbeitszeitenansicht, in welcher ihm die Arbeitszeiten aller Mitarbeiter angezeigt werden.		
Ablauf	Schritt	Akteur	Beschreibung
	1	Benutzer	Der Benutzer öffnet die Login-Seite.
	2	Benutzer	Der Benutzer wählt als Nutzertyp „Vorgesetzter“ aus.
	3	Benutzer	Der Benutzer gibt „Chef“ als Namen ein.
	4	Benutzer	Der Benutzer klickt den „Login“ Button.
	5	System	Der Benutzer wird als Vorgesetzter angemeldet.

Tabelle 14: Use-Case 1.1: Login - Korrekte Eingaben, Vorgesetzter

Use-Case 1.2: Login – Korrekter Nutzertyp, falscher Name, Vorgesetzter			
Beschreibung	Der Benutzer hat die Möglichkeit sich anzumelden.		
Akteure/Aktoren	Nicht angemeldeter Benutzer		
Eintrittsfall	Der Benutzer befindet sich auf der Login-Seite.		
Vorbedingungen	Der Benutzer ist ausgeloggt.		
Ergebnis / Nachbedingungen	Es wird eine Fehlermeldung ausgegeben, dass die Anmeldedaten inkorrekt seien.		
Ablauf	Schritt	Akteur	Beschreibung
	1	Benutzer	Der Benutzer öffnet die Login-Seite.
	2	Benutzer	Der Benutzer wählt als Nutzertyp „Vorgesetzter“ aus.
	3	Benutzer	Der Benutzer gibt einen falschen Namen ein (!= Chef).
	4	Benutzer	Der Benutzer klickt den „Login“ Button.
	5	System	Fehlermeldung: „Die Anmeldedaten sind inkorrekt!“

Tabelle 15: Use-Case 1.2: Login - Korrekter Nutzertyp, falscher Name, Vorgesetzter

Use-Case 1.3: Login – Falscher Nutzertyp, richtiger Name, Vorgesetzter			
Beschreibung	Der Benutzer hat die Möglichkeit sich anzumelden.		
Akteure/Aktoren	Nicht angemeldeter Benutzer		
Eintrittsfall	Der Benutzer befindet sich auf der Login-Seite.		
Vorbedingungen	Der Benutzer ist ausgeloggt.		
Ergebnis / Nachbedingungen	Es wird eine Fehlermeldung ausgegeben, dass die Anmeldedaten inkorrekt seien.		
Ablauf	Schritt	Akteur	Beschreibung
	1	Benutzer	Der Benutzer öffnet die Login-Seite.
	2	Benutzer	Der Benutzer wählt als Nutzertyp „Mitarbeiter“ aus.
	3	Benutzer	Der Benutzer gibt den Namen „Chef“ ein.
	4	Benutzer	Der Benutzer klickt den „Login“ Button.
	5	System	Fehlermeldung: „Die Anmeldedaten sind inkorrekt!“

Tabelle 16: Use-Case 1.3: Login - Falscher Nutzertyp, richtiger Name, Vorgesetzter

Use-Case 1.4: Login – Falscher Nutzertyp, falscher Name, Vorgesetzter			
Beschreibung	Der Benutzer hat die Möglichkeit sich anzumelden.		
Akteure/Aktoren	Nicht angemeldeter Benutzer		
Eintrittsfall	Der Benutzer befindet sich auf der Login-Seite.		
Vorbedingungen	Der Benutzer ist ausgeloggt.		
Ergebnis / Nachbedingungen	Der Benutzer wird fälschlicherweise als Mitarbeiter angemeldet.		
Ablauf	Schritt	Akteur	Beschreibung
	1	Benutzer	Der Benutzer öffnet die Login-Seite.
	2	Benutzer	Der Benutzer wählt als Nutzertyp „Mitarbeiter“ aus.
	3	Benutzer	Der Benutzer gibt einen falschen Namen ein (!= Chef).
	4	Benutzer	Der Benutzer klickt den „Login“ Button.
	5	System	Der Benutzer wird als Mitarbeiter angemeldet.

Tabelle 17: Use-Case 1.4: Login - Falscher Nutzertyp, falscher Name, Vorgesetzter

Use-Case 1.5: Login – Korrekte Eingaben, noch nicht vorhandener Mitarbeiter			
Beschreibung	Der Benutzer hat die Möglichkeit sich anzumelden.		
Akteure/Aktoren	Nicht angemeldeter Benutzer		
Eintrittsfall	Der Benutzer befindet sich auf der Login-Seite.		
Vorbedingungen	Der Benutzer ist nicht eingeloggt, der Mitarbeiter mit diesem Namen war bisher noch nie eingeloggt.		
Ergebnis / Nachbedingungen	Der Benutzer wird als Mitarbeiter angemeldet und landet in der Arbeitszeitenansicht, in welcher ihm zurzeit noch keine Arbeitszeiten angezeigt werden.		
Ablauf	Schritt	Akteur	Beschreibung
	1	Benutzer	Der Benutzer öffnet die Login-Seite.
	2	Benutzer	Der Benutzer wählt als Nutzertyp „Mitarbeiter“ aus.
	3	Benutzer	Der Benutzer gibt einen gültigen Namen (nicht leer und nicht nur Leerzeichen) ein, der noch nie zur Anmeldung verwendet wurde.
	4	Benutzer	Der Benutzer klickt den „Login“ Button.
	5	System	Der Benutzer wird als Mitarbeiter angemeldet.

Tabelle 18: Use-Case 1.5: Login - Korrekte Eingaben, noch nicht vorhandener Mitarbeiter

Use-Case 1.6: Login – Korrekte Eingaben, bereits vorhandener Mitarbeiter			
Beschreibung	Der Benutzer hat die Möglichkeit sich anzumelden.		
Akteure/Aktoren	Nicht angemeldeter Benutzer		
Eintrittsfall	Der Benutzer befindet sich auf der Login-Seite.		
Vorbedingungen	Der Benutzer ist nicht eingeloggt, der Mitarbeiter mit diesem Namen war bereits einmal eingeloggt und hat bereits Arbeitszeiten erfasst.		
Ergebnis / Nachbedingungen	Der Benutzer wird als Mitarbeiter angemeldet und landet in der Arbeitszeitenansicht, in welcher ihm die eigenen Arbeitszeiten angezeigt werden.		
Ablauf	Schritt	Akteur	Beschreibung
	1	Benutzer	Der Benutzer öffnet die Login-Seite.
	2	Benutzer	Der Benutzer wählt als Nutzertyp „Mitarbeiter“ aus.
	3	Benutzer	Der Benutzer gibt einen gültigen Namen (nicht leer und nicht nur Leerzeichen) ein, der bereits einmal zur Anmeldung verwendet wurde.
	4	Benutzer	Der Benutzer klickt den „Login“ Button.
	5	System	Der Benutzer wird als Mitarbeiter angemeldet.

Tabelle 19: Use-Case 1.6: Login - Korrekte Eingaben, bereits vorhandener Mitarbeiter

Use-Case 1.7: Login – Falscher Nutzertyp, gültiger Name, Mitarbeiter			
Beschreibung	Der Benutzer hat die Möglichkeit sich anzumelden.		
Akteure/Aktoren	Nicht angemeldeter Benutzer		
Eintrittsfall	Der Benutzer befindet sich auf der Login-Seite.		
Vorbedingungen	Der Benutzer ist nicht eingeloggt.		
Ergebnis / Nachbedingungen	Es wird eine Fehlermeldung ausgegeben, dass die Anmeldedaten inkorrekt seien.		
Ablauf	Schritt	Akteur	Beschreibung
	1	Benutzer	Der Benutzer öffnet die Login-Seite.
	2	Benutzer	Der Benutzer wählt als Nutzertyp „Vorgesetzter“ aus.
	3	Benutzer	Der Benutzer gibt einen gültigen Namen (nicht leer und nicht nur Leerzeichen) ein.
	4	Benutzer	Der Benutzer klickt den „Login“ Button.
	5	System	Fehlermeldung: „Die Anmeldedaten sind inkorrekt!“

Tabelle 20: Use-Case 1.7: Login - Falscher Nutzertyp, gültiger Name, Mitarbeiter

Use-Case 1.8: Login – Richtiger Nutzertyp, ungültiger Name, Mitarbeiter			
Beschreibung	Der Benutzer hat die Möglichkeit sich anzumelden.		
Akteure/Aktoren	Nicht angemeldeter Benutzer		
Eintrittsfall	Der Benutzer befindet sich auf der Login-Seite.		
Vorbedingungen	Der Benutzer ist nicht eingeloggt.		
Ergebnis / Nachbedingungen	Es wird eine Fehlermeldung ausgegeben, dass die Anmeldedaten inkorrekt seien.		
Ablauf	Schritt	Akteur	Beschreibung
	1	Benutzer	Der Benutzer öffnet die Login-Seite.
	2	Benutzer	Der Benutzer wählt als Nutzertyp „Mitarbeiter“ aus.
	3	Benutzer	Der Benutzer gibt einen ungültigen Namen (leer oder ohne Leerzeichen) ein.
	4	Benutzer	Der Benutzer klickt den „Login“ Button.
	5	System	Fehlermeldung: „Die Anmeldedaten sind inkorrekt!“

Tabelle 21: Use-Case 1.8: Login - Richtiger Nutzertyp, ungültiger Name, Mitarbeiter

Use-Case 1.9: Login – Falscher Nutzertyp, ungültiger Name			
Beschreibung	Der Benutzer hat die Möglichkeit sich anzumelden.		
Akteure/Aktoren	Nicht angemeldeter Benutzer		
Eintrittsfall	Der Benutzer befindet sich auf der Login-Seite.		
Vorbedingungen	Der Benutzer ist nicht eingeloggt.		
Ergebnis / Nachbedingungen	Es wird eine Fehlermeldung ausgegeben, dass die Anmeldedaten inkorrekt seien.		
Ablauf	Schritt	Akteur	Beschreibung
	1	Benutzer	Der Benutzer öffnet die Login-Seite.
	2	Benutzer	Der Benutzer wählt als Nutzertyp „Vorgesetzter“ aus.
	3	Benutzer	Der Benutzer gibt einen ungültigen Namen (leer oder nur Leerzeichen) ein.
	4	Benutzer	Der Benutzer klickt den „Login“ Button.
	5	System	Fehlermeldung: „Die Anmeldedaten sind inkorrekt!“

Tabelle 22: Use-Case 1.9: Login - Falscher Nutzertyp, ungültiger Name, Mitarbeiter

2.3.2 Use-Case 2: Arbeitszeitenansicht

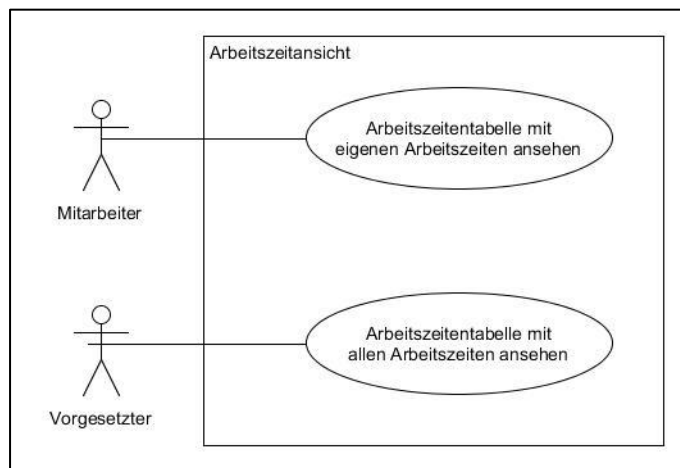


Abbildung 5: Use-Case-Diagramm 2 – Arbeitszeitenansicht

Use-Case 2.1: Arbeitszeitenansicht – Vorgesetzter, bereits erfasste Arbeitszeiten			
Beschreibung	Der Benutzer hat als Vorgesetzter die Möglichkeit die Arbeitszeiten aller Mitarbeiter anzusehen.		
Akteure/Aktoren	Als Vorgesetzter angemeldeter Benutzer		
Eintrittsfall	Der Benutzer befindet sich in der Arbeitszeitenansicht.		
Vorbedingungen	Der Benutzer ist als Vorgesetzter angemeldet, es sind bereits Arbeitszeiten erfasst.		
Ergebnis / Nachbedingungen	Es werden die Arbeitszeiten aller Mitarbeiter angezeigt.		
Ablauf	Schritt	Akteur	Beschreibung
	1	Benutzer	Der Benutzer öffnet die Arbeitszeitenansicht.
	2	System	Die Arbeitszeiten aller Mitarbeiter werden angezeigt.

Tabelle 23: Use-Case 2.1: Arbeitszeitenansicht - Vorgesetzter, bereits erfasste Arbeitszeiten

Use-Case 2.2: Arbeitszeitenansicht – Vorgesetzter, keine bereits erfassten Arbeitszeiten			
Beschreibung	Der Benutzer hat als Vorgesetzter die Möglichkeit die Arbeitszeiten aller Mitarbeiter anzusehen.		
Akteure/Aktoren	Als Vorgesetzter angemeldeter Benutzer		
Eintrittsfall	Der Benutzer befindet sich in der Arbeitszeitenansicht.		
Vorbedingungen	Der Benutzer ist als Vorgesetzter angemeldet, es sind bisher noch keine Arbeitszeiten erfasst.		
Ergebnis / Nachbedingungen	Es werden keine Arbeitszeiten angezeigt.		
Ablauf	Schritt	Akteur	Beschreibung
	1	Benutzer	Der Benutzer öffnet die Arbeitszeitenansicht.
	2	System	Es werden keine Arbeitszeiten angezeigt.

Tabelle 24: Use-Case 2.2: Arbeitszeitenansicht - Vorgesetzter, keine bereits erfassten Arbeitszeiten

Use-Case 2.3: Arbeitszeitenansicht – Mitarbeiter, bereits erfasste Arbeitszeiten			
Beschreibung	Der Benutzer hat als Mitarbeiter die Möglichkeit seine Arbeitszeiten anzusehen.		
Akteure/Aktoren	Als Mitarbeiter angemeldeter Benutzer		
Eintrittsfall	Der Benutzer befindet sich in der Arbeitszeitenansicht.		
Vorbedingungen	Der Benutzer ist als Mitarbeiter angemeldet, es sind bereits Arbeitszeiten erfasst.		
Ergebnis / Nachbedingungen	Es werden alle von diesem Mitarbeiter erfassten Arbeitszeiten angezeigt.		
Ablauf	Schritt	Akteur	Beschreibung
	1	Benutzer	Der Benutzer öffnet die Arbeitszeitenansicht.
	2	System	Es werden alle von diesem Mitarbeiter erfassten Arbeitszeiten angezeigt.

Tabelle 25: Use-Case 2.3: Arbeitszeitenansicht - Mitarbeiter, bereits erfasste Arbeitszeiten

Use-Case 2.4: Arbeitszeitenansicht – Mitarbeiter, keine bereits erfassten Arbeitszeiten			
Beschreibung	Der Benutzer hat als Mitarbeiter die Möglichkeit seine Arbeitszeiten anzusehen.		
Akteure/Aktoren	Als Mitarbeiter angemeldeter Benutzer		
Eintrittsfall	Der Benutzer befindet sich in der Arbeitszeitenansicht.		
Vorbedingungen	Der Benutzer ist als Mitarbeiter angemeldet, es sind bisher noch keine Arbeitszeiten erfasst.		
Ergebnis / Nachbedingungen	Es werden keine Arbeitszeiten angezeigt.		
Ablauf	Schritt	Akteur	Beschreibung
	1	Benutzer	Der Benutzer öffnet die Arbeitszeitenansicht.
	2	System	Es werden keine Arbeitszeiten angezeigt.

Tabelle 26: Use-Case 2.4: Arbeitszeitenansicht - Mitarbeiter, keine bereits erfassten Arbeitszeiten

2.3.3 Use-Case 3: Zeiterfassung

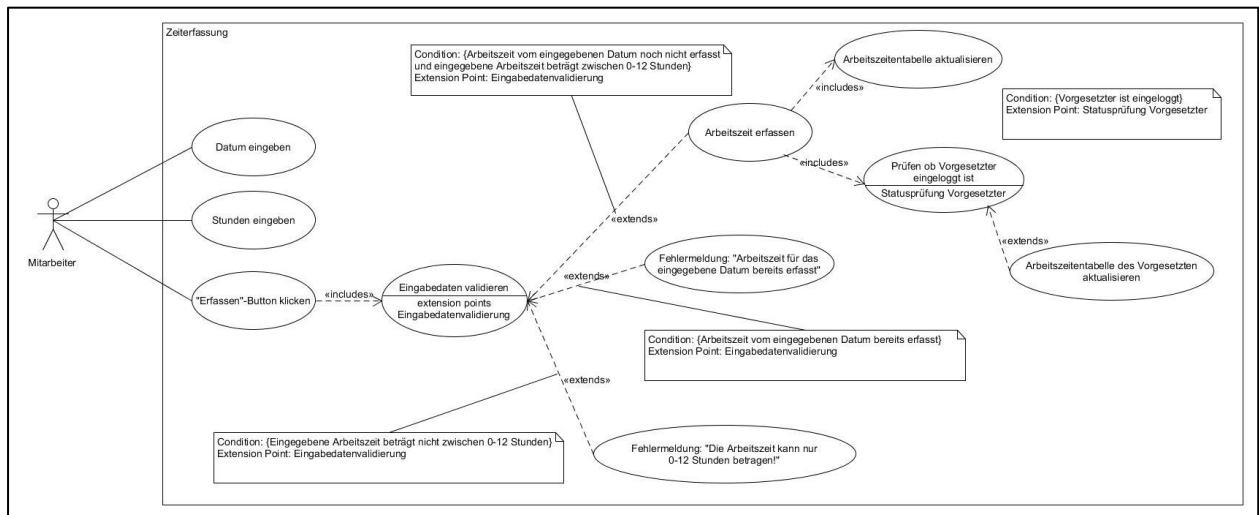


Abbildung 6: Use-Case-Diagramm 3 – Zeiterfassung

Use-Case 3.1: Zeiterfassung – Datum ohne bereits vorhandener Arbeitszeit, gültige Arbeitszeit, eingeloggter Vorgesetzter			
Beschreibung	Der Benutzer hat als Mitarbeiter die Möglichkeit, neue Arbeitszeiten zu erfassen.		
Akteure/Aktoren	Als Mitarbeiter angemeldeter Benutzer.		
Eintrittsfall	Der Benutzer befindet sich in der Zeiterfassungs-Ansicht.		
Vorbedingungen	Der Benutzer ist als Mitarbeiter angemeldet, es ist noch keine Arbeitszeit für das eingegebene Datum erfasst, der Vorgesetzte ist eingeloggt.		
Ergebnis / Nachbedingungen	Die Arbeitszeit wird erfasst und die Arbeitszeitenansicht beim Mitarbeiter und dem Vorgesetzten automatisch aktualisiert.		
Ablauf	Schritt	Akteur	Beschreibung
	1	Benutzer	Der Benutzer öffnet die Zeiterfassungs-Ansicht
	2	Benutzer	Der Benutzer gibt ein Datum ein, für welches noch keine Arbeitszeit erfasst wurde.
	3	Benutzer	Der Benutzer gibt eine gültige Arbeitszeit ein (zwischen 0 und 12h).
	4	Benutzer	Der Benutzer klickt den „Erfassen“-Button.
	5	System	Die Arbeitszeit wird erfasst und die Arbeitszeitenansicht des Mitarbeiters und des Vorgesetzten automatisch aktualisiert.

Tabelle 27: Use-Case 3.1: Zeiterfassung - Datum ohne bereits vorhandener Arbeitszeit, gültige Arbeitszeit, eingeloggter Vorgesetzter

Use-Case 3.2: Zeiterfassung – Datum ohne bereits vorhandener Arbeitszeit, gültige Arbeitszeit, ausgeloggtter Vorgesetzter			
Beschreibung	Der Benutzer hat als Mitarbeiter die Möglichkeit, neue Arbeitszeiten zu erfassen.		
Akteure/Aktoren	Als Mitarbeiter angemeldeter Benutzer.		
Eintrittsfall	Der Benutzer befindet sich in der Zeiterfassungs-Ansicht.		
Vorbedingungen	Der Benutzer ist als Mitarbeiter angemeldet, es ist noch keine Arbeitszeit für das eingegebene Datum erfasst, der Vorgesetzte ist ausgeloggt.		
Ergebnis / Nachbedingungen	Die Arbeitszeit wird erfasst und die Arbeitszeitenansicht des Mitarbeiters automatisch aktualisiert.		
Ablauf	Schritt	Akteur	Beschreibung
	1	Benutzer	Der Benutzer öffnet die Zeiterfassungs-Ansicht.
	2	Benutzer	Der Benutzer gibt ein Datum ein, für welches noch keine Arbeitszeit erfasst wurde.
	3	Benutzer	Der Benutzer gibt eine gültige Arbeitszeit ein (zwischen 0 und 12h).
	4	Benutzer	Der Benutzer klickt den „Erfassen“-Button.
	5	System	Die Arbeitszeit wird erfasst und die Arbeitszeitenansicht des Mitarbeiters automatisch aktualisiert.

Tabelle 28: Use-Case 3.2: Zeiterfassung - Datum ohne bereits vorhandener Arbeitszeit, gültige Arbeitszeit, ausgeloggtter Vorgesetzter

Use-Case 3.3: Zeiterfassung – Datum mit bereits vorhandener Arbeitszeit, gültige Arbeitszeit			
Beschreibung	Der Benutzer hat als Mitarbeiter die Möglichkeit, neue Arbeitszeiten zu erfassen.		
Akteure/Aktoren	Als Mitarbeiter angemeldeter Benutzer.		
Eintrittsfall	Der Benutzer befindet sich in der Zeiterfassungs-Ansicht.		
Vorbedingungen	Der Benutzer ist als Mitarbeiter angemeldet, es wurde bereits eine Arbeitszeit für das eingegebene Datum erfasst.		
Ergebnis / Nachbedingungen	Es wird eine Fehlermeldung ausgegeben, dass die Arbeitszeit dieses Datums bereits erfasst wurde.		
Ablauf	Schritt	Akteur	Beschreibung
	1	Benutzer	Der Benutzer öffnet die Zeiterfassungs-Ansicht.
	2	Benutzer	Der Benutzer gibt ein Datum ein, für welches bereits die Arbeitszeit erfasst wurde.
	3	Benutzer	Der Benutzer gibt eine gültige Arbeitszeit ein (zwischen 0 und 12h).
	4	Benutzer	Der Benutzer klickt den „Erfassen“-Button.
	5	System	Fehlermeldung: „Arbeitszeit für das eingegebene Datum bereits erfasst!“

Tabelle 29: Use-Case 3.3: Zeiterfassung - Datum mit bereits vorhandener Arbeitszeit, gültige Arbeitszeit

Use-Case 3.4: Zeiterfassung – Datum ohne bereits vorhandener Arbeitszeit, ungültige Arbeitszeit			
Beschreibung	Der Benutzer hat als Mitarbeiter die Möglichkeit, neue Arbeitszeiten zu erfassen.		
Akteure/Aktoren	Als Mitarbeiter angemeldeter Benutzer.		
Eintrittsfall	Der Benutzer befindet sich in der Zeiterfassungs-Ansicht.		
Vorbedingungen	Der Benutzer ist als Mitarbeiter angemeldet, es wurde noch keine Arbeitszeit für das eingegebene Datum erfasst.		
Ergebnis / Nachbedingungen	Es wird eine Fehlermeldung ausgegeben, dass die Arbeitszeit nur zwischen 0 und 12 Stunden betragen darf.		
Ablauf	Schritt	Akteur	Beschreibung
	1	Benutzer	Der Benutzer öffnet die Zeiterfassungs-Ansicht.
	2	Benutzer	Der Benutzer gibt ein Datum ein, für welches noch keine Arbeitszeit erfasst wurde.
	3	Benutzer	Der Benutzer gibt eine ungültige Arbeitszeit ein (nicht zwischen 0 und 12h).
	4	Benutzer	Der Benutzer klickt den „Erfassen“-Button.
	5	System	Fehlermeldung: „Die Arbeitszeit kann nur zwischen 0 und 12 Stunden betragen!“

Tabelle 30: Use-Case 3.4: Zeiterfassung - Datum ohne bereits vorhandener Arbeitszeit, ungültige Arbeitszeit

Use-Case 3.5: Zeiterfassung – Datum mit bereits vorhandener Arbeitszeit, ungültige Arbeitszeit			
Beschreibung	Der Benutzer hat als Mitarbeiter die Möglichkeit, neue Arbeitszeiten zu erfassen.		
Akteure/Aktoren	Als Mitarbeiter angemeldeter Benutzer.		
Eintrittsfall	Der Benutzer befindet sich in der Zeiterfassungs-Ansicht.		
Vorbedingungen	Der Benutzer ist als Mitarbeiter angemeldet, es wurde bereits eine Arbeitszeit für das eingegebene Datum erfasst.		
Ergebnis / Nachbedingungen	Es wird eine Fehlermeldung ausgegeben, dass die Arbeitszeit nur zwischen 0 und 12 Stunden betragen darf und für das eingegebene Datum bereits eine Arbeitszeit erfasst wurde.		
Ablauf	Schritt	Akteur	Beschreibung
	1	Benutzer	Der Benutzer öffnet die Zeiterfassungs-Ansicht
	2	Benutzer	Der Benutzer gibt ein Datum ein, für welches bereits eine Arbeitszeit erfasst wurde.
	3	Benutzer	Der Benutzer gibt eine ungültige Arbeitszeit ein (nicht zwischen 0 und 12h).
	4	Benutzer	Der Benutzer klickt den „Erfassen“-Button.
	5	System	Fehlermeldung: „Die Arbeitszeit für das angegebene Datum wurde bereits erfasst, ausserdem kann die Arbeitszeit nur zwischen 0 und 12 Stunden betragen!“

Tabelle 31: Use-Case 3.5: Zeiterfassung - Datum mit bereits vorhandener Arbeitszeit, ungültige Arbeitszeit

3 Planen

In der Planungsphase des IPERKA-Prinzips wird die eigentliche Planung des Projekts vorgenommen. Dies beinhaltet bspw. die Zeitplanung, das Testkonzept usw.

3.1 Testkonzept

Im Testkonzept wird ein erstes Konzept für die nach der Implementation der Applikation durchzuführenden Tests erstellt. Ausserdem sind die Randbedingungen und die eingesetzten Testmittel- und Methoden genauer beschrieben.

3.1.1 Randbedingungen

Die Randbedingungen beschreiben die genaue Testumgebung wie bspw. Informationen über das zum Test verwendete Gerät, Betriebssystem, Browser etc.

Die Tests werden unter folgenden Bedingungen durchgeführt:

- Gerät: Als Testgerät wird ein Lenovo P70 Notebook mit einer Displayauflösung von 1920x1080 Pixel verwendet. Zusätzlich werden noch zwei externe Monitore mit einer Displayauflösung von je 1920x1200 Pixel verwendet.
- Betriebssystem: Als Betriebssystem ist Windows 7 installiert.
- Browser: Als Browser wird die momentan aktuelle Version (55.0.2883.87) von Google Chrome in der 64-bit Version verwendet.

3.1.2 Eingesetzte Testmittel und –Methoden

Informationen zum geplanten Testvorgang und die zum Test verwendeten Tools.

Um die Applikation zu testen, werden nach Abschluss der Implementation Blackbox-Tests durchgeführt. Dabei werden normale Abläufe der Applikation praxisgetreu getestet. Eine Anforderung an die Applikation ist es ebenfalls mind. 2 sinnvolle Unit-Tests zu implementieren, diese sind jedoch im Code ersichtlich und werden nicht näher dokumentiert. Um den REST-Service des Java-Backends zu testen, wird das Tool „SoapUI“ verwendet, welches neben anderen Webservices auch die Möglichkeit bietet REST-Services zu testen. Diese Tests werden bei Bedarf zwischendurch durchgeführt, sind jedoch auch kein eigentlicher Teil des Testings.

3.1.3 Testfälle

Alle Testfälle basierend auf den zuvor definierten Use-Cases.

Testfall 1 – Login - Korrekte Eingaben, Vorgesetzter		
Vorbedingungen:	Der Benutzer ist nicht eingeloggt.	
Testmittel:	Google Chrome Browser	
Testablauf:	Schritt	Beschreibung
	1	Die „Login“-Seite wird geöffnet.
	2	„Vorgesetzter“ wird als Nutzertyp ausgewählt.
	3	„Chef“ wird als Name eingegeben.
	4	Der „Login“-Button wird geklickt.
Erwartetes Resultat:	Man wird als Vorgesetzter angemeldet und landet in der Arbeitszeitenansicht, in welcher die Arbeitszeiten aller Mitarbeiter angezeigt werden.	

Tabelle 32: Testfall 1 - Login - Korrekte Eingaben, Vorgesetzter

Testfall 2 – Login – Korrekter Nutzertyp, falscher Name, Vorgesetzter		
Vorbedingungen:	Der Benutzer ist nicht eingeloggt.	
Testmittel:	Google Chrome Browser	
Testablauf:	Schritt	Beschreibung
	1	Die „Login“-Seite wird geöffnet.
	2	„Vorgesetzter“ wird als Nutzertyp ausgewählt.
	3	Es wird ein falscher Name eingegeben (!= Chef).
	4	Der „Login“-Button wird geklickt.
Erwartetes Resultat:	Es wird eine Fehlermeldung ausgegeben, dass die Anmeldedaten inkorrekt seien.	

Tabelle 33: Testfall 2 - Login - Korrekter Nutzertyp, falscher Name, Vorgesetzter

Testfall 3 – Login – Falscher Nutzertyp, richtiger Name, Vorgesetzter		
Vorbedingungen:	Der Benutzer ist nicht eingeloggt.	
Testmittel:	Google Chrome Browser	
Testablauf:	Schritt	Beschreibung
	1	Die „Login“-Seite wird geöffnet.
	2	„Mitarbeiter“ wird als Nutzertyp ausgewählt.
	3	„Chef“ wird als Name eingegeben.
	4	Der „Login“-Button wird geklickt.
Erwartetes Resultat:	Es wird eine Fehlermeldung ausgegeben, dass die Anmeldedaten inkorrekt seien.	

Tabelle 34: Testfall 3 - Login - Falscher Nutzertyp, richtiger Name, Vorgesetzter

Testfall 4 – Login – Falscher Nutzertyp, falscher Name, Vorgesetzter		
Vorbedingungen:	Der Benutzer ist nicht eingeloggt.	
Testmittel:	Google Chrome Browser	
Testablauf:	Schritt	Beschreibung
	1	Die „Login“-Seite wird geöffnet.
	2	„Mitarbeiter“ wird als Nutzertyp ausgewählt.
	3	Es wird ein falscher Name eingegeben (!= Chef).
	4	Der „Login“-Button wird geklickt.
Erwartetes Resultat:	Man wird fälschlicherweise als Mitarbeiter angemeldet.	

Tabelle 35: Testfall 4 - Login - Falscher Nutzertyp, falscher Name, Vorgesetzter

Testfall 5 – Login – Korrekte Eingaben, noch nicht vorhandener Mitarbeiter		
Vorbedingungen:	Der Benutzer ist nicht eingeloggt, der Mitarbeiter mit diesem Namen war bisher noch nie eingeloggt.	
Testmittel:	Google Chrome Browser	
Testablauf:	Schritt	Beschreibung
	1	Die „Login“-Seite wird geöffnet.
	2	„Mitarbeiter“ wird als Nutzertyp ausgewählt.
	3	Es wird ein gültiger Name eingegeben (nicht leer und nicht nur Leerzeichen), der noch nie zur Anmeldung verwendet wurde.
	4	Der „Login“-Button wird geklickt.
Erwartetes Resultat:	Man wird als Mitarbeiter angemeldet und landet in der Arbeitszeitenansicht, in welcher momentan noch keine Arbeitszeiten angezeigt werden.	

Tabelle 36: Testfall 5 - Login - Korrekte Eingaben, noch nicht vorhandener Mitarbeiter

Testfall 6 – Login – Korrekte Eingaben, bereits vorhandener Mitarbeiter		
Vorbedingungen:	Der Benutzer ist nicht eingeloggt, der Mitarbeiter mit diesem Namen war bereits einmal eingeloggt und hat bereits Arbeitszeiten erfasst.	
Testmittel:	Google Chrome Browser	
Testablauf:	Schritt	Beschreibung
	1	Die „Login“-Seite wird geöffnet.
	2	„Mitarbeiter“ wird als Nutzertyp ausgewählt.
	3	Es wird ein gültiger Name eingegeben (nicht leer und nicht nur Leerzeichen), der bereits einmal zur Anmeldung verwendet wurde.
	4	Der „Login“-Button wird geklickt.
Erwartetes Resultat:	Man wird als Mitarbeiter angemeldet und landet in der Arbeitszeitenansicht, in welcher die Arbeitszeiten dieses Mitarbeiters angezeigt werden.	

Tabelle 37: Testfall 6 - Login - Korrekte Eingaben, bereits vorhandener Mitarbeiter

Testfall 7 – Login – Falscher Nutzertyp, gültiger Name, Mitarbeiter		
Vorbedingungen:	Der Benutzer ist nicht eingeloggt.	
Testmittel:	Google Chrome Browser	
Testablauf:	Schritt	Beschreibung
	1	Die „Login“-Seite wird geöffnet.
	2	„Vorgesetzter“ wird als Nutzertyp ausgewählt.
	3	Es wird ein gültiger Name eingegeben (nicht leer und nicht nur Leerzeichen).
	4	Der „Login“-Button wird geklickt.
Erwartetes Resultat:	Es wird eine Fehlermeldung ausgegeben, dass die Anmeldedaten inkorrekt seien.	

Tabelle 38: Testfall 7 - Login - Falscher Nutzertyp, gültiger Name, Mitarbeiter

Testfall 8 – Login – Richtiger Nutzertyp, ungültiger Name, Mitarbeiter		
Vorbedingungen:	Der Benutzer ist nicht eingeloggt.	
Testmittel:	Google Chrome Browser	
Testablauf:	Schritt	Beschreibung
	1	Die „Login“-Seite wird geöffnet.
	2	„Mitarbeiter“ wird als Nutzertyp ausgewählt.
	3	Es wird ein ungültiger Name eingegeben (leer oder nur Leerzeichen).
	4	Der „Login“-Button wird geklickt.
Erwartetes Resultat:	Es wird eine Fehlermeldung ausgegeben, dass die Anmeldedaten inkorrekt seien.	

Tabelle 39: Testfall 8 - Login - Richtiger Nutzertyp, ungültiger Name, Mitarbeiter

Testfall 9 – Login – Falscher Nutzertyp, ungültiger Name, Mitarbeiter		
Vorbedingungen:	Der Benutzer ist nicht eingeloggt.	
Testmittel:	Google Chrome Browser	
Testablauf:	Schritt	Beschreibung
	1	Die „Login“-Seite wird geöffnet.
	2	„Vorgesetzter“ wird als Nutzertyp ausgewählt.
	3	Es wird ein ungültiger Name eingegeben (leer oder nur Leerzeichen).
	4	Der „Login“-Button wird geklickt.
Erwartetes Resultat:	Es wird eine Fehlermeldung ausgegeben, dass die Anmeldedaten inkorrekt seien.	

Tabelle 40: Testfall 9 - Login - Falscher Nutzertyp, ungültiger Name, Mitarbeiter

Testfall 10 – Arbeitszeitenansicht – Vorgesetzter, bereits erfasste Arbeitszeiten		
Vorbedingungen:	Der Benutzer ist als Vorgesetzter angemeldet, es sind bereits Arbeitszeiten erfasst.	
Testmittel:	Google Chrome Browser	
Testablauf:	Schritt	Beschreibung
	1	Die Arbeitszeitenansicht wird geöffnet.
Erwartetes Resultat:	Es werden die Arbeitszeiten aller Mitarbeiter angezeigt.	

Tabelle 41: Testfall 10 - Arbeitszeitenansicht - Vorgesetzter, bereits erfasste Arbeitszeiten

Testfall 11 – Arbeitszeitenansicht – Vorgesetzter, keine bereits erfassten Arbeitszeiten		
Vorbedingungen:	Der Benutzer ist als Vorgesetzter angemeldet, es sind bisher noch keine Arbeitszeiten erfasst.	
Testmittel:	Google Chrome Browser	
Testablauf:	Schritt	Beschreibung
	1	Die Arbeitszeitenansicht wird geöffnet.
Erwartetes Resultat:	Es werden keine Arbeitszeiten angezeigt.	

Tabelle 42: Testfall 11 - Arbeitszeitenansicht - Vorgesetzter, keine bereits erfassten Arbeitszeiten

Testfall 12 – Arbeitszeitenansicht – Mitarbeiter, bereits erfasste Arbeitszeiten		
Vorbedingungen:	Der Benutzer ist als Mitarbeiter angemeldet, es sind bereits Arbeitszeiten erfasst.	
Testmittel:	Google Chrome Browser	
Testablauf:	Schritt	Beschreibung
	1	Die Arbeitszeitenansicht wird geöffnet.
Erwartetes Resultat:	Es werden alle von diesem Mitarbeiter erfassten Arbeitszeiten angezeigt.	

Tabelle 43: Testfall 12 - Arbeitszeitenansicht - Mitarbeiter, bereits erfasste Arbeitszeiten

Testfall 13 – Arbeitszeitenansicht – Mitarbeiter, keine bereits erfassten Arbeitszeiten		
Vorbedingungen:	Der Benutzer ist als Mitarbeiter angemeldet, es sind bisher noch keine Arbeitszeiten erfasst.	
Testmittel:	Google Chrome Browser	
Testablauf:	Schritt	Beschreibung
	1	Die Arbeitszeitenansicht wird geöffnet.
Erwartetes Resultat:	Es werden keine Arbeitszeiten angezeigt.	

Tabelle 44: Testfall 13 - Arbeitszeitenansicht - Mitarbeiter, keine bereits erfassten Arbeitszeiten

Testfall 14 – Zeiterfassung – Datum ohne bereits vorhandener Arbeitszeit, gültige Arbeitszeit, eingeloggter Vorgesetzter		
Vorbedingungen:	Der Benutzer ist als Mitarbeiter angemeldet, es ist noch keine Arbeitszeit für das eingegebene Datum erfasst, der Vorgesetzte ist eingeloggt.	
Testmittel:	Google Chrome Browser	
Testablauf:	Schritt	Beschreibung
	1	Die Zeiterfassungs-Ansicht wird geöffnet.
	2	Es wird ein Datum eingegeben, für welches noch keine Arbeitszeit erfasst wurde.
	3	Es wird eine gültige Arbeitszeit (zwischen 0-12h) eingegeben.
	4	Es wird der „Erfassen“-Button geklickt.
Erwartetes Resultat:	Die Arbeitszeit wird erfasst und die Arbeitszeitenansicht des angemeldeten Benutzers und des Vorgesetzten automatisch aktualisiert.	

Tabelle 45: Testfall 14 - Zeiterfassung - Datum ohne bereits vorhandener Arbeitszeit, gültige Arbeitszeit, eingeloggter Vorgesetzter

Testfall 15 – Zeiterfassung – Datum ohne bereits vorhandener Arbeitszeit, gültige Arbeitszeit, ausgeloggtter Vorgesetzter		
Vorbedingungen:	Der Benutzer ist als Mitarbeiter angemeldet, es ist noch keine Arbeitszeit für das eingegebene Datum erfasst, der Vorgesetzte ist ausgeloggt.	
Testmittel:	Google Chrome Browser	
Testablauf:	Schritt	Beschreibung
	1	Die Zeiterfassungs-Ansicht wird geöffnet.
	2	Es wird ein Datum eingegeben, für welches noch keine Arbeitszeit erfasst wurde.
	3	Es wird eine gültige Arbeitszeit (zwischen 0-12h) eingegeben.
	4	Es wird der „Erfassen“-Button geklickt.
Erwartetes Resultat:	Die Arbeitszeit wird erfasst und die Arbeitszeitenansicht des angemeldeten Benutzers automatisch aktualisiert.	

Tabelle 46: Testfall 15 - Zeiterfassung - Datum ohne bereits vorhandener Arbeitszeit, gültige Arbeitszeit, ausgeloggtter Vorgesetzter

Testfall 16 – Zeiterfassung – Datum mit bereits vorhandener Arbeitszeit, gültige Arbeitszeit		
Vorbedingungen:	Der Benutzer ist als Mitarbeiter angemeldet, es wurde bereits eine Arbeitszeit für das eingegebene Datum erfasst.	
Testmittel:	Google Chrome Browser	
Testablauf:	Schritt	Beschreibung
	1	Die Zeiterfassungs-Ansicht wird geöffnet.
	2	Es wird ein Datum eingegeben, für welches bereits eine Arbeitszeit erfasst wurde.
	3	Es wird eine gültige Arbeitszeit (zwischen 0-12h) eingegeben.
	4	Es wird der „Erfassen“-Button geklickt.
Erwartetes Resultat:	Es wird eine Fehlermeldung ausgegeben, dass die Arbeitszeit dieses Datums bereits erfasst wurde.	

Tabelle 47: Testfall 16 - Zeiterfassung - Datum mit bereits vorhandener Arbeitszeit, gültige Arbeitszeit

Testfall 17 – Zeiterfassung – Datum ohne bereits vorhandener Arbeitszeit, ungültige Arbeitszeit		
Vorbedingungen:	Der Benutzer ist als Mitarbeiter angemeldet, es wurde noch keine Arbeitszeit für das eingegebene Datum erfasst.	
Testmittel:	Google Chrome Browser	
Testablauf:	Schritt	Beschreibung
	1	Die Zeiterfassungs-Ansicht wird geöffnet.
	2	Es wird ein Datum eingegeben, für welches noch keine Arbeitszeit erfasst wurde.
	3	Es wird eine ungültige Arbeitszeit (nicht zwischen 0-12h) eingegeben.
	4	Es wird der „Erfassen“-Button geklickt.
Erwartetes Resultat:	Es wird eine Fehlermeldung ausgegeben, dass die Arbeitszeit nur zwischen 0 und 12 Stunden betragen darf.	

Tabelle 48: Testfall 17 - Zeiterfassung - Datum ohne bereits vorhandener Arbeitszeit, ungültige Arbeitszeit

Testfall 18 – Zeiterfassung – Datum mit bereits vorhandener Arbeitszeit, ungültige Arbeitszeit		
Vorbedingungen:	Der Benutzer ist als Mitarbeiter angemeldet, es wurde bereits eine Arbeitszeit für das eingegebene Datum erfasst.	
Testmittel:	Google Chrome Browser	
Testablauf:	Schritt	Beschreibung
	1	Die Zeiterfassungs-Ansicht wird geöffnet.
	2	Es wird ein Datum eingegeben, für welches bereits eine Arbeitszeit erfasst wurde.
	3	Es wird eine ungültige Arbeitszeit (nicht zwischen 0-12h) eingegeben.
	4	Es wird der „Erfassen“-Button geklickt.
Erwartetes Resultat:	Es wird eine Fehlermeldung ausgegeben, dass die Arbeitszeit nur zwischen 0 und 12 Stunden betragen darf und für das eingegebene Datum bereits eine Arbeitszeit erfasst wurde.	

Tabelle 49: Testfall 18 - Zeiterfassung - Datum mit bereits vorhandener Arbeitszeit, ungültige Arbeitszeit

3.2 Software-Architektur

Informationen über die Architektur und den technischen Aufbau der umgesetzten Applikation.

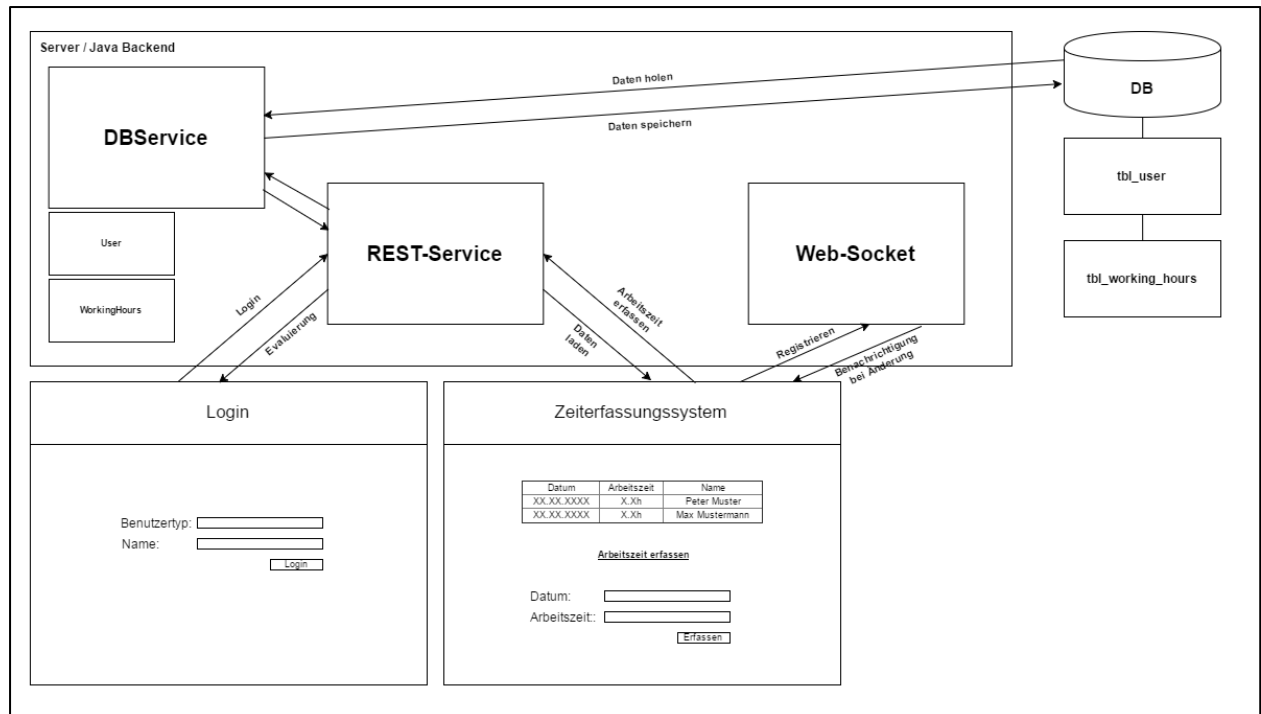


Abbildung 7: Software-Architektur

Die Architektur der Software ist folgendermassen aufgebaut. Und zwar besteht die Applikation aus dem Server, oben links im Bild, dem Client, unten im Bild und der Datenbank, oben rechts im Bild. Das Login Formular auf dem Client ruft beim Klick auf den „Login“-Button, den zugehörigen Webservice auf dem Server auf und erhält eine Antwort vom Server, ob die Daten korrekt waren und er nun eingeloggt ist oder nicht.

In der zweiten Ansicht werden die Daten zuerst geladen, dies geschieht ebenfalls wieder über den Webservice. Um die Daten anschliessend zu laden ruft der Webservice die zugehörige Methode des DBService Interfaces auf. Die Klasse DBServiceImpl implementiert diese Methoden und baut die Verbindung zur Datenbank auf und setzt anschliessend die Querys ab. Wird auf dem Client eine neue Arbeitszeit erfasst, wird diese wiederum an eine Webservice-Methode weitergeleitet, welcher die Daten zuerst validiert und bei korrekter Eingabe die Daten über das DBService-Interface in der Datenbank speichert.

Falls sich ein Vorgesetzter anmeldet, registriert sich dieser automatisch beim Web-Socket, welcher den Client anschliessend automatisch benachrichtigt, wenn ein Mitarbeiter neue Arbeitszeiten erfasst hat.

3.3 Datenbankdesign

In diesem Abschnitt sind das Datenbankdesign, das ERD und die einzelnen Tabellen der Datenbank näher beschrieben.

Die Struktur der Datenbank ist sehr einfach. Die Tabelle besteht aus lediglich 2 Tabellen, eine Tabelle mit allen Benutzern und eine Tabelle mit allen Arbeitszeiten. Einem Benutzer können keine, eine oder mehrere Arbeitszeiten zugewiesen werden, eine Arbeitszeit muss jedoch zwingend einem Benutzer zugewiesen werden aus diesem Grund existiert zwischen der Benutzer-Tabelle und der Arbeitszeiten-Tabelle eine „1 zu m“-Beziehung.

3.3.1 Entity-Relationship-Model

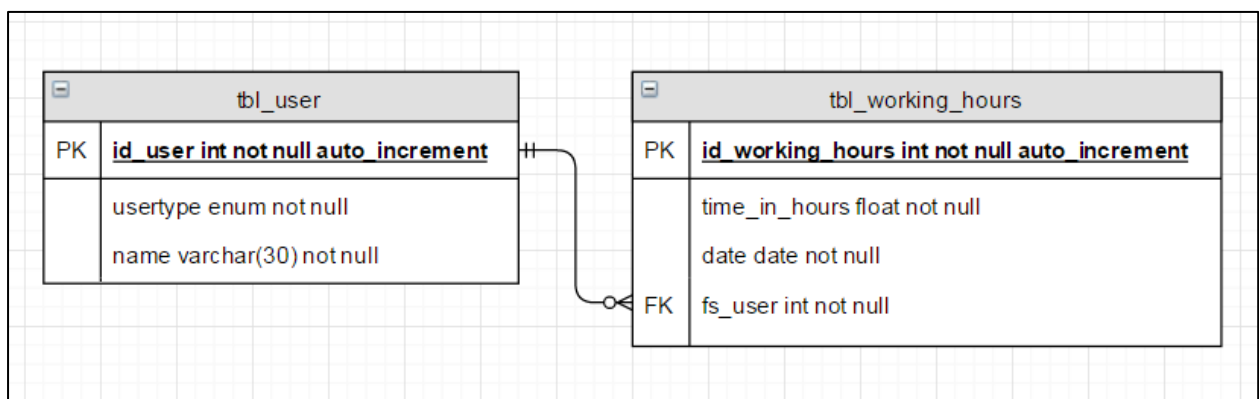


Abbildung 8: ERM

3.3.2 Tabellen

3.3.2.1 tbl_user

In dieser Tabelle befinden sich alle Benutzer. Die Tabelle beinhaltet die ID des Users, den „usertype“ als Enum, also ob der Benutzer ein Mitarbeiter oder ein Vorgesetzter ist, und den Namen des Benutzers, dieser wird als varchar mit 30 Stellen gespeichert. Alle Attribute dürfen nicht leer sein, sind also „not null“.

3.3.2.2 tbl_working_hours

In dieser Tabelle befinden sich die erfassten Arbeitszeiten der Mitarbeiter. Die Tabelle beinhaltet die ID der Arbeitszeit, die Arbeitszeit als Dezimalzahl, das Datum der erfassten Zeit als Datum und den Fremdschlüssel des Users, um der Arbeitszeit den zugehörigen User zuzuweisen. Auch in dieser Tabelle sind alle Attribute „not null“, müssen also zwingend einen Wert beinhalten.

4 Entscheiden

In der 3. Phase des IPERKA-Prinzips, der Entscheidung, wird genauer entschieden, wie vorgegangen wird bzw. eine gezielte Entscheidung getroffen.

In diesem Fall wird in dieser Phase die Nutzung eines Versionverwaltungssystems evaluiert.

4.1 Versionverwaltungssystem

In diesem Abschnitt wird darüber entschieden, welches Versionverwaltungssystem in diesem Projekt zur Anwendung kommen soll.

Ein Versionverwaltungssystem ist heutzutage ein sehr wichtiges Instrument in der Softwareentwicklung. Dabei werden ausgewählte Dateien und Ordner vom System beobachtet und neue Dateiversionen und Änderungen festgehalten. Dies ermöglicht es, frühere Versionen von Dateien wiederherzustellen und allfällige Fehler oder Änderungen rückgängig zu machen. Ebenfalls ein grosser Vorteil von Versionverwaltungssystemen ist es, dass leichter im Team gearbeitet werden kann. Dabei arbeiten die zusammenarbeitenden Personen in einem gemeinsamen Repository und können ihre aktuellen Versionen gemeinsam verwalten.

Der entscheidende Vorteil eines Versionverwaltungssystems in diesem Projekt ist jedoch die eigentliche Versionverwaltung, die es mir ermöglicht, zwischen verschiedenen Versionen zu wechseln und vorherige Versionen wiederherzustellen. Dies hat entscheidende Vorteile, wenn sich in einer neuen Version einmal ein Fehler einschleicht oder ich mit verschiedenen Versionen arbeiten möchte.

Es gibt viele verschiedene Versionverwaltungssysteme, die weitverbreitetsten Versionverwaltungssysteme sind jedoch GIT und SVN. Während GIT sehr oft bei Open-Source-Projekten angewandt wird, wird SVN oft bei Softwareprojekten in grösseren Unternehmen eingesetzt.

Der Hauptunterschied zwischen den beiden Systemen ist dabei das Arbeiten im Team in einem gemeinsamen Repository.

SVN setzt dabei auf ein zentrales Repository, das von allen Benutzern gemeinsam genutzt wird. Der Nachteil dabei ist, dass kein eigenes Repository erstellt werden kann, in dem die einzelnen User ihre Versionen verwalten können. Jedoch kann es auch ein Vorteil sein, dass das Comitten bzw. Pushen in ein geteiltes Repository weniger Schritte erfordert.

Bei GIT besitzt jeder User ein eigenes lokales Repository und man arbeitet dann zusätzlich mit sogenannten Remote-Repositories. Ein Remote-Repository ist dabei ein weiteres zentrales Repository, das auf einem Server stationiert ist. Dieses Remote-Repository kann anschliessend einem lokalen Repository hinzugefügt werden um eine aktuelle Version vom Remote Repository zu pullen/herunterzuladen oder eine neue Version in das Remote Repository zu pushen/hochzuladen. Da bei GIT mit lokalen und externen Repositories gearbeitet wird, benötigt das pushen/hochladen mehrere Schritte, da die aktuelle Version zuerst in das lokale Repository committet/gespeichert und erst anschliessend auf das Remote-Repository gepusht/hochgeladen werden kann.

Da ich mit SVN gar keine Erfahrungen habe und in der Vergangenheit ausschliesslich mit GIT gearbeitet habe, werde ich dieses auch in diesem Projekt verwenden.

5 Realisieren

In der Realisierungsphase eines Projekts, wird das jeweilige Projekt schliesslich umgesetzt. In diesem Fall ist das die Realisierung der Applikation, welche in diesem Kapitel genauer dokumentiert wird.

5.1 Klassendiagramm

Die Übersicht aller Klassen in einem Klassendiagramm.

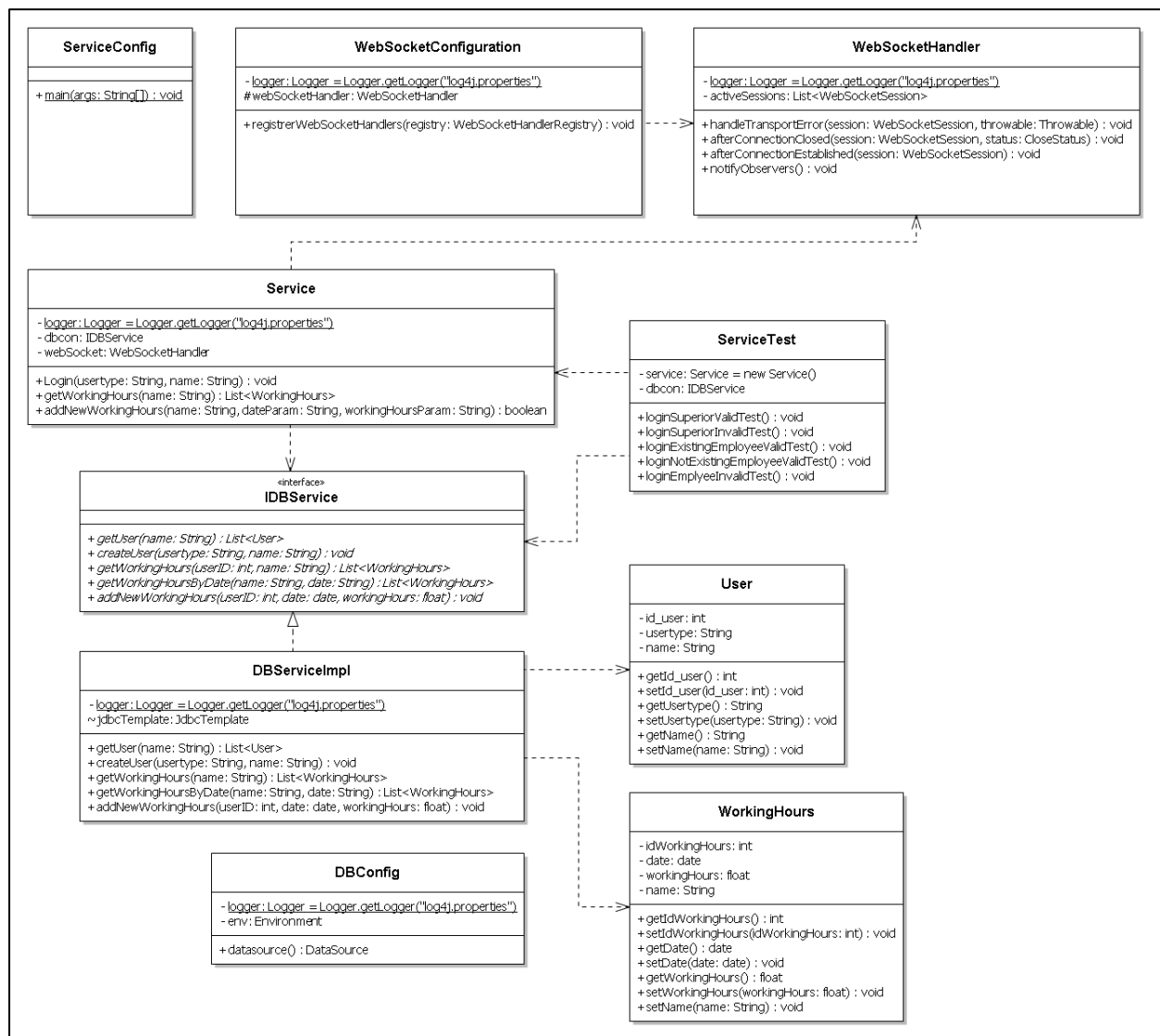


Abbildung 9: Klassendiagramm

5.1.1 Klassendokumentation

Eine genauere Beschreibung jeder Klasse.

5.1.1.1 ServiceConfig

Diese ServiceConfig-Klasse wird von Spring benötigt, um die Applikation inkl. aller Komponenten zu starten.

5.1.1.2 User & WorkingHours

Dies sind zwei reine Datenklassen, die von der DBServiceImpl-Klasse verwendet werden, um mithilfe des RowMappers die Daten aus der Datenbank an ein Objekt zu binden und diese später in einer Liste am Client zurückzugeben.

5.1.1.3 DBConfig

Die DBConfig Klasse ist dafür zuständig, die Datenbankverbindung zu konfigurieren. Dazu lädt und registriert er die Eigenschaften aus dem „application.properties“-File, welches alle Datenbankdaten wie Datenbanknamen, Benutzer etc. beinhaltet.

5.1.1.4 IDBService

Bei IDBService handelt es sich um ein Interface, welches vier datenbankspezifische Methoden beinhaltet. Es dient als Schnittstelle für den Webservice um datenbankspezifische Methoden aufzurufen um Daten aus der Datenbank zu laden oder um neue Daten in der Datenbank abzuspeichern.

5.1.1.5 DBServiceImpl

„DBServiceImpl“ ist die Klasse, welche die vier Methoden des Interfaces „IDBService“ implementiert.

Sie implementiert folgende Methoden:

getUser(String name):

Diese Methode sucht in der Datenbank nach einem Benutzer mit einem spezifischen Namen.

createUser(String usertype, String name):

Diese Methode speichert einen neuen Benutzer(Mitarbeiter) in der Datenbank, falls dieser noch nicht vorhanden ist.

getWorkingHours(int userID, String name):

Diese Methode lädt die Arbeitszeiten aller Mitarbeiter, falls die Anfrage vom Vorgesetzten kommt, ansonsten lädt sie die Arbeitszeiten des jeweiligen Mitarbeiters.

addNewWorkingHours(int userID, Date date, float workingHours):

Diese Methode ist dafür zuständig, neu erfasste Arbeitszeiten in die Datenbank zu speichern.

5.1.1.6 Service

Diese Klasse stellt den Web-Service dar, der dem Client insgesamt 3 Methoden zur Verfügung stellt:

login(String usertype, String name):

Diese Methode ruft der Client auf, um einen Benutzer anzumelden.

Dazu wird als erstes geprüft, ob es sich beim Benutzer um den Vorgesetzten oder um einen Mitarbeiter handelt.

Handelt es sich um den Vorgesetzten, werden die Eingabedaten direkt in dieser Methode geprüft. Sind die Anmeldedaten valid und korrekt, wird ein „True“ zurückgegeben, ansonsten ein „False“. Handelt es sich beim Anmelder um einen Mitarbeiter, wird zuerst geprüft, ob die eingegebenen Daten überhaupt valid sind. Wenn die eingegebenen Daten valid sind, wird der Mitarbeiter in der Datenbank gesucht, ist der Mitarbeiter bereits in der Datenbank erfasst, wird ein „True“ zurückgegeben, falls nicht, wird ein neuer Benutzer in der Datenbank angelegt und anschliessend ebenfalls ein „True“ zurückgegeben. Falls die Eingabedaten nicht valid sind, wird ein „False“ zurückgegeben.

getWorkingHours(String name):

Diese Methode ruft einzig die gleichnamige Methode der DBServiceImpl-Klasse auf und gibt die erhaltene Liste an „workingHours“-Objekten zurück.

addNewWorkingHours(String name, Date date, float workingHours):

Diese Methode ruft als erstes die „getUser()“-Methode der DBServiceImpl-Klasse auf um an die ID des Benutzers zu gelangen. Anschliessend ruft sie die „addNewWorkingHours()“-Methode der DBServiceImpl-Klasse auf um die neu erfassten Arbeitszeiten in die Datenbank zu speichern.

5.1.1.7 WebSocketConfiguration

Die WebSocketConfiguration-Klasse ist für die Konfiguration des Web-Sockets zuständig. Sie implementiert das Spring-Interface „WebSocketConfigurer“ und die Methode „registerWebSocketHandlers()“, welche dafür zuständig ist die Handlerklasse für den Web-Socket zu registrieren. Dabei handelt es sich um die im nächsten Punkt folgende Klasse.

5.1.1.8 WebSocketHandler

Diese Klasse erbt von der Spring-Klasse TextWebSocketHandler und ist für die Bereitstellung des Web-Services zuständig.

Sie beinhaltet folgende vier Methoden:

handleTransportError():

Diese Methode wird aufgerufen, wenn während der Web-Socket-Verbindung ein Fehler auftritt. Danach loggt sie den aufgetretenen Fehler in ein Log-File.

afterConnectionClosed():

Diese Methode wird aufgerufen, wenn eine Web-Socket-Session geschlossen wurde. Anschliessend entfernt sie die jeweilige Session aus der Liste „activeSessions“, in welcher alle Sessions der mit dem Web-Socket verbundenen Clients gespeichert sind.

afterConnectionEstablished():

Diese Methode wird aufgerufen, wenn eine neue Web-Socket-Verbindung hergestellt wurde, danach speichert sie die jeweilige Session in die „activeSessions“-List, in welcher alle Sessions der mit dem Web-Socket verbundenen Clients gespeichert sind.

notifyObservers():

Diese Methode wird vom REST-Service aus aufgerufen, wenn ein Mitarbeiter neue Arbeitszeiten erfasst hat. Danach benachrichtigt diese Methode die in der „activeSessions“-Liste gespeicherten Sessions, sofern sich eine Session darin befindet.

5.2 Screenshots

Einige Screenshots der Applikation.

5.2.1 Login

Zeiterfassungssystem

Login

Benutzertyp:

Name:

Abbildung 10: Screenshot 1 – Login

Im Screenshot oben, ist das Login-Formular zu sehen, mit welchem sich ein Benutzer mit dem jeweiligen Benutzertyp und seinem Namen anmeldet.

5.2.2 Login – Fehlermeldung bei Falscheingabe

Unter localhost:8080 wird Folgendes angezeigt:

Die Anmeldedaten sind inkorrekt!

Abbildung 11: Screenshot 2 - Login - Fehlermeldung bei Falscheingabe

Diese Fehlermeldung wird angezeigt, wenn sich ein Vorgesetzter mit einem anderen Namen als „Chef“ anmeldet oder das „Name“-Feld leer ist oder nur aus Leerzeichen besteht.

5.2.3 Arbeitszeitenansicht – Vorgesetzter – Erfasste Daten

Zeiterfassungssystem

Arbeitszeitenansicht

Show entries

Search:

Datum	Arbeitszeit in Stunden	Name
2017-01-25	7.5	Testuser2
2017-01-26	8.75	Testuser2
2017-01-30	8	Testuser
2017-01-31	8.5	Testuser

Showing 1 to 4 of 4 entries

Previous Next

Abbildung 12: Screenshot 3 - Arbeitszeitenansicht - Vorgesetzter - Erfasste Daten

In diesem Screenshot ist die Arbeitszeitenansicht des Vorgesetzten ersichtlich. Ihm werden darin in einer Tabelle die Zeiten aller Mitarbeiter angezeigt. Bei der Tabelle handelt es sich um die von jQuery mitgelieferte Tabelle „Datatable“. Sie ermöglicht es, nach einer bestimmten Zeile zu ordnen oder bspw. auch nach einem Eintrag zu suchen.

5.2.4 Arbeitszeitenansicht – Vorgesetzter – keine erfassten Daten

Zeiterfassungssystem

Arbeitszeitenansicht

Show

10

 entries

Search:

Datum	Arbeitszeit in Stunden	Name
No data available in table		

Showing 0 to 0 of 0 entries

Previous

Next

Abbildung 13: Screenshot 4 - Arbeitszeitenansicht - Vorgesetzter - keine erfassten Daten

Fall noch keine Arbeitszeiten erfasst wurden, wird dem Vorgesetzten eine leere Tabelle angezeigt.

5.2.5 Arbeitszeitenansicht – Mitarbeiter – Erfasste Daten

Zeiterfassungssystem			
Arbeitszeitenansicht			
Show <input type="text" value="10"/> entries		Search: <input type="text"/>	
Datum	Arbeitszeit in Stunden	Name	
2017-01-30	8	Testuser	
2017-01-31	8.5	Testuser	
Showing 1 to 2 of 2 entries		Previous	1 Next

Abbildung 14: Screenshot 5 - Arbeitszeitenansicht – Mitarbeiter – Erfasste Daten

In der Arbeitszeitenansicht eines Mitarbeiters, werden diesem nur seine eigenen Arbeitszeiten in der Tabelle angezeigt.

5.2.6 Arbeitszeitenansicht – Mitarbeiter – Keine erfassten Daten

Zeiterfassungssystem

Arbeitszeitenansicht

Show

10

 entries

Search:

Datum	Arbeitszeit in Stunden	Name
No data available in table		

Showing 0 to 0 of 0 entries

Previous

Next

Abbildung 15: Screenshot 6 - Arbeitszeitenansicht – Mitarbeiter – keine erfassten Daten

Wenn der Mitarbeiter bisher noch keine Arbeitszeit erfasst hatte, wird diesem wiederum eine leere Tabelle angezeigt.

5.2.7 Neue Arbeitszeit erfassen

Neue Arbeitszeit erfassen

Datum:

Arbeitszeit in h:

Abbildung 16: Screenshot 7 - Neue Arbeitszeit erfassen

Um eine neue Arbeitszeit zu erfassen, steht dem Mitarbeiter das obig abgebildete Formular zur Verfügung.

5.2.8 Neue Arbeitszeit erfassen – Datumsauswahl



Neue Arbeitszeit erfassen

Datum:

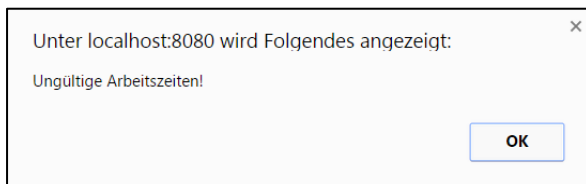
January 2017

Su	Mo	Tu	We	Th	Fr	Sa
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

Abbildung 17: Screenshot 8 - Neue Arbeitszeit erfassen – Datumsauswahl

Um eine einfache Datumseingabe mit dem richtigen Format zu ermöglichen, habe ich bei der Datumseingabe auf den „Datepicker“ von jQuery gesetzt, dieser bietet den Vorteil, dass das Datumsformat manuell festgelegt werden kann.

5.2.9 Neue Arbeitszeit erfassen – Fehlermeldung bei ungültiger Eingabe



Unter localhost:8080 wird Folgendes angezeigt:

Ungültige Arbeitszeiten!

OK

Abbildung 18: Screenshot 9 - Neue Arbeitszeit erfassen - Fehlermeldung bei ungültiger Eingabe

Wenn der Mitarbeiter ungültige Arbeitszeiten erfasst, das heisst bspw. ein Feld leer lässt, ein Datum auswählt für welches bereits eine Arbeitszeit erfasst wurde, eine Arbeitszeit von weniger als 0h oder grösser als 10h einträgt oder sonstige invalide Daten eingibt, wird die obig abgebildete Fehlermeldung ausgegeben.

6 Kontrollieren

In der Kontrollieren-Phase des Projekts wird nun die Umsetzung genauer kontrolliert und Tests durchgeführt.

In diesem Kapitel findet man das Testing inkl. dem Testprotokoll mit den tatsächlichen Testergebnissen.

6.1 Testing

In diesem Abschnitt werden die vorwärtig definierten Tests durchgeführt und in einem Testprotokoll anschaulich dargestellt. Schliesslich wird noch ein Testfazit gezogen.

6.1.1 Testprotokoll

Testprotokoll	
Testfall 1 – Login - Korrekte Eingaben, Vorgesetzter	
Tatsächliches Ergebnis:	Man wird als Vorgesetzter angemeldet und landet in der Arbeitszeitenansicht, in welcher die Arbeitszeiten aller Mitarbeiter angezeigt werden.
Testergebnis:	Erfolgreich
Testfall 2 – Login – Korrekter Nutzertyp, falscher Name, Vorgesetzter	
Tatsächliches Ergebnis:	Es wird eine Fehlermeldung ausgegeben, dass die Anmeldedaten inkorrekt seien.
Testergebnis:	Erfolgreich
Testfall 3 – Login – Falscher Nutzertyp, richtiger Name, Vorgesetzter	
Tatsächliches Ergebnis:	Man wird fälschlicherweise als Mitarbeiter eingeloggt.
Testergebnis:	Nicht erfolgreich
Wiederholter Test	
Tatsächliches Ergebnis:	Es wird eine Fehlermeldung ausgegeben, dass die Anmeldedaten inkorrekt seien.
Testergebnis:	Erfolgreich

Testfall 4 – Login – Falscher Nutzertyp, falscher Name, Vorgesetzter	
Tatsächliches Ergebnis:	Man wird fälschlicherweise als Mitarbeiter eingeloggt.
Testergebnis:	Erfolgreich
Testfall 5 – Login – Korrekte Eingaben, noch nicht vorhandener Mitarbeiter	
Tatsächliches Ergebnis:	Man wird als Mitarbeiter angemeldet und landet in der Arbeitszeitenansicht, in welcher momentan noch keine Arbeitszeiten angezeigt werden. Der neue Benutzer wird ausserdem in der Datenbank abgespeichert.
Testergebnis:	Erfolgreich
Testfall 6 – Login – Korrekte Eingaben, bereits vorhandener Mitarbeiter	
Tatsächliches Ergebnis:	Man wird als Mitarbeiter angemeldet und landet in der Arbeitszeitenansicht, in welcher die Arbeitszeiten dieses Mitarbeiters angezeigt werden.
Testergebnis:	Erfolgreich
Testfall 7 – Login – Falscher Nutzertyp, gültiger Name, Mitarbeiter	
Tatsächliches Ergebnis:	Es wird eine Fehlermeldung ausgegeben, dass die Anmeldedaten inkorrekt seien.
Testergebnis:	Erfolgreich
Testfall 8 – Login – Richtiger Nutzertyp, ungültiger Name, Mitarbeiter	
Tatsächliches Ergebnis:	Es wird eine Fehlermeldung ausgegeben, dass die Anmeldedaten inkorrekt seien.
Testergebnis:	Erfolgreich
Testfall 9 – Login – Falscher Nutzertyp, ungültiger Name, Mitarbeiter	
Tatsächliches Ergebnis:	Es wird eine Fehlermeldung ausgegeben, dass die Anmeldedaten inkorrekt seien.
Testergebnis:	Erfolgreich

Testfall 10 – Arbeitszeitenansicht – Vorgesetzter, bereits erfasste Arbeitszeiten	
Tatsächliches Ergebnis:	Es werden die Arbeitszeiten aller Mitarbeiter angezeigt.
Testergebnis:	Erfolgreich
Testfall 11 – Arbeitszeitenansicht – Vorgesetzter, keine bereits erfassten Arbeitszeiten	
Tatsächliches Ergebnis:	Es werden keine Arbeitszeiten angezeigt, nur eine leere Tabelle.
Testergebnis:	Erfolgreich
Testfall 12 – Arbeitszeitenansicht – Mitarbeiter, bereits erfasste Arbeitszeiten	
Tatsächliches Ergebnis:	Es werden alle von diesem Mitarbeiter erfassten Arbeitszeiten angezeigt.
Testergebnis:	Erfolgreich
Testfall 13 – Arbeitszeitenansicht – Mitarbeiter, keine bereits erfassten Arbeitszeiten	
Tatsächliches Ergebnis:	Es werden keine Arbeitszeiten angezeigt, nur eine leere Tabelle.
Testergebnis:	Erfolgreich
Testfall 14 – Zeiterfassung – Datum ohne bereits vorhandener Arbeitszeit, gültige Arbeitszeit, eingeloggter Vorgesetzter	
Tatsächliches Ergebnis:	Die Arbeitszeit wird erfasst und die Arbeitszeitenansicht des angemeldeten Benutzers und des Vorgesetzten automatisch aktualisiert.
Testergebnis:	Erfolgreich
Testfall 15 – Zeiterfassung – Datum ohne bereits vorhandener Arbeitszeit, gültige Arbeitszeit, ausgeloggtter Vorgesetzter	
Tatsächliches Ergebnis:	Die Arbeitszeit wird erfasst und die Arbeitszeitenansicht des angemeldeten Benutzers automatisch aktualisiert.
Testergebnis:	Erfolgreich

Testfall 16 – Zeiterfassung – Datum mit bereits vorhandener Arbeitszeit, gültige Arbeitszeit	
Tatsächliches Ergebnis:	Es wird eine Fehlermeldung ausgegeben, dass die Arbeitszeiten ungültig seien.
Testergebnis:	(Nicht) erfolgreich
Testfall 17 – Zeiterfassung – Datum ohne bereits vorhandener Arbeitszeit, ungültige Arbeitszeit	
Tatsächliches Ergebnis:	Der Server wirft einen Error und der Client gibt einen “alert” mit dem Error aus.
Testergebnis:	Nicht erfolgreich
Wiederholter Test	
Tatsächliches Ergebnis:	Es wird eine Fehlermeldung ausgegeben, dass die Arbeitszeiten ungültig seien.
Testergebnis:	(Nicht) erfolgreich
Testfall 18 – Zeiterfassung – Datum mit bereits vorhandener Arbeitszeit, ungültige Arbeitszeit	
Tatsächliches Ergebnis:	Der Server wirft einen Error und der Client gibt einen “alert” mit dem Error aus.
Testergebnis:	Nicht erfolgreich
Wiederholter Test	
Tatsächliches Ergebnis:	Es wird eine Fehlermeldung ausgegeben, dass die Arbeitszeiten ungültig seien.
Testergebnis:	(Nicht) erfolgreich

Tabelle 50: Testprotokoll

6.1.2 Testfazit

Die meisten Tests waren erfolgreich, jedoch muss bei einigen Punkten noch nachgebessert werden. So wird bspw. bei der Erfassung der Arbeitszeit, wenn an diesem Datum bereits eine Zeit erfasst wurde, zwar eine Fehlermeldung ausgegeben, jedoch nicht spezifisch auf den Fehler. Das heisst, dem Benutzer ist nicht klar, ob nun die Arbeitszeit ungültig ist, oder ob er für dieses Datum bereits eine Arbeitszeit erfasst ist. Des Weiteren wird im Moment die Arbeitszeit nur so weit überprüft, dass die Arbeitszeit zwischen 0-12 Stunden beträgt. Doch der Fall, dass der Benutzer Buchstaben oder Sonderzeichen in das Textfeld eingibt, wird momentan nicht gehandelt und der Server wirft zurzeit einen Fehler, welche vom Client anschliessend angezeigt wird und für den User nichts aussagend ist. Ein weiterer Fehler der noch behoben werden muss, ist es, dass wenn sich ein neuer Benutzer mit dem Namen „Chef“ anmeldet, er zwar als Mitarbeiter mit dem Namen „Chef“ angemeldet wird, sich jedoch fälschlicherweise beim Web-Socket registriert, was nicht sein sollte. Das heisst, der Name „Chef“ sollte abgefangen werden, falls es sich bei dem sich einloggenden Benutzer um einen Mitarbeiter handelt.

Nach wiederholtem Test:

Schliesslich konnte ich die noch auftretenden Fehler und Probleme auch noch beheben.

Ein Mitarbeiter kann sich nun nicht mehr unter dem Namen „Chef“ anmelden und es wird stattdessen eine Fehlermeldung ausgegeben.

Bei der Erfassung neuer Arbeitszeiten, werden nun ungültige Arbeitszeiteingaben ebenfalls abgefangen und eine Fehlermeldung ausgegeben.

Der einzige Punkt der noch offen ist, ist die nicht spezifische Fehlerausgabe bei der Zeiterfassung, dies ist aber kein Fehler sondern eher eine Verbesserungsmöglichkeit.

7 Auswerten

In der letzten Phase nach IPERKA, wird das gesamte Projekt nochmals reflexiert und ausgewertet.

7.1 Reflexion

In diesem Abschnitt wird das gesamte Projekt nochmals reflexiert. Was konnte umgesetzt werden, was fehlt allenfalls noch, wie sah es mit dem Zeitmanagement aus usw.

Das Projekt ist nun abgeschlossen und es wird Zeit ein Fazit zu ziehen.

Ich bin sehr zufrieden damit, wie das Projekt verlaufen ist und welches Endprodukt daraus entstanden ist. Alle Funktionen und Anforderungen konnten ordnungsmässig implementiert werden und auch alle zugehörigen Tests waren erfolgreich. Besonders mit meinem Zeitmanagement bin ich ausserordentlich zufrieden, denn ich habe von der ersten Minute an, richtig Gas gegeben und habe dadurch bereits zu Beginn des Projekts einiges an Zeit gewonnen. Dadurch war ich dem Zeitplan die meiste Zeit voraus und arbeitete meist schon an den Tasks vom nächsten Tag. Diese gewonnene Zeit konnte ich anschliessend dafür aufwenden, nochmals die Dokumentation durchzugehen oder noch einige Anpassungen am Code zu machen.

7.2 Erweiterungspotenzial der Applikation

Wie könnte die Applikation in Zukunft noch erweitert werden?

Die Applikation könnte noch beliebig erweitert werden. Der nächste Punkt der wohl nochmals angegangen werden müsste, sind die Fehlermeldungen bei der Arbeitszeiterfassung, denn diese sind momentan noch nicht spezifisch.

Ein weiterer Punkt wäre das Design des GUIs, denn der Fokus wurde in diesem Projekt auf die Funktionalität und nicht auf die Darstellung im GUI gesetzt.

Des Weiteren ist momentan die Validierung der Daten ausschliesslich auf dem Server umgesetzt. Eine zusätzliche Validierung auf dem Client wäre diesbezüglich ebenfalls noch von Vorteil. Dieser könnte anschliessend dafür sorgen, dass erst gar keine Requests an den Server gesendet werden, wenn die Eingaben invalide sind, das heisst wenn das Eingabeformat falsch ist oder ein Feld leer gelassen wird. Um dies zu verhindern könnte man bspw. den zugehörigen Button deaktivieren, die betroffenen Felder rot markieren und eine Meldung oder Notiz ausgeben, welchen den Nutzer darüber informiert, welche Felder nicht korrekt ausgefüllt wurden. Dies würde auch die Benutzerführung um einiges verbessern. Eine weitere naheliegende Funktion welche die Applikation noch sinnvoll erweitern würde, wäre die Möglichkeit eines Vorgesetzten zur Visierung der Arbeitszeiten der Mitarbeiter. Somit könnte ein Vorgesetzter bspw. einen Haken setzen um zu bestätigen, dass er die Arbeitszeit eingesehen hat und damit einverstanden ist.

Momentan existiert erst ein Vorgesetzter namens „Chef“. Sinnvoll wäre es noch, wenn mehrere Vorgesetzte existieren könnten und diese anschliessend ihre unterstellten Mitarbeiter einsehen könnten usw. Dementsprechend müsste natürlich eine Beziehung bestehen zwischen dem Mitarbeiter und dem Vorgesetzten. Um sich dann auch neu registrieren zu können wäre es auch

noch vorteilhaft, ein neues Registrierungsformular und ein separates Login-Formular zu implementieren.

Man merkt also, hinter diesem Projekt steckt noch viel Potenzial und die Erweiterungsmöglichkeiten sind praktisch unbegrenzt.

7.3 Schlusswort

Einige letzte Worte zum Projekt.

Nun ist das Projekt zu Ende und ich schaue auf ein erfolgreiches Projekt zurück. Es hat mir sehr geholfen, meine Kenntnisse und mein Know-How in der Java-Programmierung und in der Implementierung von REST-Webservices und Web-Sockets weiter zu vertiefen und zu verbessern und vor allem auch noch mehr Praxis darin zu bekommen. Mein ganzer Workflow konnte sich dadurch verbessern und verschnellern und ich kann nun viele Tools noch gezielter einsetzen. Ebenfalls hat mir dieses Projekt auch meine Stärken und Schwächen näher aufgezeigt und mir somit dabei geholfen zu erkennen wo ich mich noch verbessern muss. Schliesslich war es auch ein sehr interessantes Projekt und es hat Spass gemacht es umzusetzen. Dementsprechend hat dies natürlich auch meine Motivation gesteigert.

Anhang

Im Anhang findet man diverse Verzeichnisse wie das Abbildungs-, Tabellen- und Literaturverzeichnis, das Glossar sowie die Angaben der Quellen. Schliesslich befindet sich auch noch der Programmcode darin.

1 Glossar

Hier sind einige technische Bezeichnungen und mögliche Fremdwörter usw. näher erläutert.

Begriff, Abkürzung	Erklärung
(Remote-) Repository	Ein lokales oder entferntes (remote) von einem Dateiverwaltungssystem zu beobachtendes Dateiverzeichnis.
Backend	Die Logik einer Applikation im Hintergrund.
Boolean	Ein Datentyp welcher true oder false beinhaltet.
Commit	Dateiänderungen im Repository speichern.
CSS	CSS ausgeschrieben Cascading Style Sheets ist eine Stylesheet-Sprache, welche meist zur Gestaltung von Webseiten verwendet wird.
Datatable	Eine von jQuery bereitgestellte Tabelle mit zahlreichen Funktionen.
Daterangepicker	Eine von jQuery bereitgestellte Datumsauswahl.
Deployen	Eine Applikation auf einem Server installieren, starten und bereitstellen
Enum	Ein Datentyp welcher nur festgelegte Werte annehmen kann.
Exception	Ausgabe von Fehlerzuständen
Frontend	Die Benutzeroberfläche, das GUI einer Applikation.
GIT/SVN	Zwei gängige Versionverwaltungssysteme.
HTML	HTML ausgeschrieben Hypertext Markup Language ist eine webbasierte Scriptsprache.
Java	Eine objektorientierte Programmiersprache von Oracle
JavaScript	JavaScript (kurz JS) ist eine Skriptsprache welche für dynamisches HTML in Webbrowser entwickelt wurde um bspw. Benutzerinteraktionen auszuwerten, Inhalte zu verändern etc.
jQuery	jQuery ist eine freie JavaScript-Bibliothek, welche zahlreiche Funktionen bereitstellt.
JUnit	JUnit ist ein Framework zum Testen von Java-Programmen, insbesondere von Klassen und Methoden.
Logging/Log	Automatisches Speichern von Prozessen, Datenänderungen und Fehlern
Maven	Ein Build-Management-Tool der Apache Software Foundation basierend auf Java
Mockito	Mockito ist eine freie Programmbibliothek zum Erstellen von Mock-Objekten für Unit-Tests in Java.
MySQL	Datenbankverwaltungssystem
Open-Source	Quelloffen
Push	Hochladen von aktuellen Dateiversionen auf ein Remote-Repository
REST	REST ausgeschrieben Representational State Transfer ist eine Webservice-Technologie welche auf der Struktur und dem Verhalten des World Wide Web aufbaut.
RowMapper	Eine bereitgestelltes Interface des Spring Frameworks welche

	Datenbanktabellen in Objekte umwandelt.
SoapUI	Eine Software zum Testen von Webservices.
Spring Framework	Quelloffenes Framework für Java
Varchar	Eine Kette an chars bzw. einzelnen Zeichen

Tabelle 51: Glossar

2 Abbildungsverzeichnis

Hier sind nochmals alle in der Dokumentation vorhandenen Abbildungen aufgelistet.

Abbildung 1: Aufgabenstellung Teil 1	6
Abbildung 2: Aufgabenstellung Teil 2	7
Abbildung 3: GANTT-Zeitplan	13
Abbildung 4: Use-Case-Diagramm 1 – Login.....	38
Abbildung 5: Use-Case-Diagramm 2 – Arbeitszeitenansicht	43
Abbildung 6: Use-Case-Diagramm 3 – Zeiterfassung.....	45
Abbildung 7: Software-Architektur	56
Abbildung 8: ERM.....	57
Abbildung 9: Klassendiagramm.....	60
Abbildung 10: Screenshot 1 – Login.....	64
Abbildung 11: Screenshot 2 - Login - Fehlermeldung bei Falscheingabe	64
Abbildung 12: Screenshot 3 - Arbeitszeitenansicht - Vorgesetzter - Erfasste Daten.....	64
Abbildung 13: Screenshot 4 - Arbeitszeitenansicht - Vorgesetzter - keine erfassten Daten.....	65
Abbildung 14: Screenshot 5 - Arbeitszeitenansicht – Mitarbeiter – Erfasste Daten	65
Abbildung 15: Screenshot 6 - Arbeitszeitenansicht – Mitarbeiter – keine erfassten Daten	65
Abbildung 16: Screenshot 7 - Neue Arbeitszeit erfassen.....	65
Abbildung 17: Screenshot 8 - Neue Arbeitszeit erfassen – Datumsauswahl	66
Abbildung 18: Screenshot 9 - Neue Arbeitszeit erfassen - Fehlermeldung bei ungültiger Eingabe	66

3 Tabellenverzeichnis

Hier sind nochmals alle in der Dokumentation vorhandenen Tabellen aufgelistet.

Tabelle 1: Änderungshistorie	2
Tabelle 2: Beteiligte Personen	8
Tabelle 3: Meilensteine.....	14
Tabelle 4: Arbeitsprotokoll - Tag 1 - Donnerstag, 12.01.2017	15
Tabelle 5: Arbeitsprotokoll - Tag 2 - Freitag, 13.01.2017	17
Tabelle 6: Arbeitsprotokoll - Tag 3 - Mittwoch, 18.01.2017	19
Tabelle 7: Arbeitsprotokoll - Tag 4 - Donnerstag, 19.01.2017	21
Tabelle 8: Arbeitsprotokoll - Tag 5 - Freitag, 20.01.2017	23
Tabelle 9: Arbeitsprotokoll - Tag 6 - Mittwoch, 25.01.2017	25
Tabelle 10: Arbeitsprotokoll - Tag 7 - Donnerstag, 26.01.2017	27
Tabelle 11: Arbeitsprotokoll - Tag 8 - Freitag, 27.01.2017	29
Tabelle 12: Arbeitsprotokoll - Tag 9 - Dienstag, 31.01.2017.....	31
Tabelle 13: Arbeitsprotokoll - Tag 10 - Mittwoch, 01.02.2017	33
Tabelle 13: Use-Case 1.1: Login - Korrekte Eingaben, Vorgesetzter	38
Tabelle 14: Use-Case 1.2: Login - Korrekter Nutzertyp, falscher Name, Vorgesetzter	39
Tabelle 15: Use-Case 1.3: Login - Falscher Nutzertyp, richtiger Name, Vorgesetzter	39
Tabelle 16: Use-Case 1.4: Login - Falscher Nutzertyp, falscher Name, Vorgesetzter	40
Tabelle 17: Use-Case 1.5: Login - Korrekte Eingaben, noch nicht vorhandener Mitarbeiter	40
Tabelle 18: Use-Case 1.6: Login - Korrekte Eingaben, bereits vorhandener Mitarbeiter	41
Tabelle 19: Use-Case 1.7: Login - Falscher Nutzertyp, gültiger Name, Mitarbeiter	41
Tabelle 20: Use-Case 1.8: Login - Richtiger Nutzertyp, ungültiger Name, Mitarbeiter	42
Tabelle 21: Use-Case 1.9: Login - Falscher Nutzertyp, ungültiger Name, Mitarbeiter	42
Tabelle 22: Use-Case 2.1: Arbeitszeitenansicht - Vorgesetzter, bereits erfasste Arbeitszeiten.....	43
Tabelle 23: Use-Case 2.2: Arbeitszeitenansicht - Vorgesetzter, keine bereits erfassten Arbeitszeiten	43
Tabelle 24: Use-Case 2.3: Arbeitszeitenansicht - Mitarbeiter, bereits erfasste Arbeitszeiten	44
Tabelle 25: Use-Case 2.4: Arbeitszeitenansicht - Mitarbeiter, keine bereits erfassten Arbeitszeiten .	44
Tabelle 26: Use-Case 3.1: Zeiterfassung - Datum ohne bereits vorhandener Arbeitszeit, gültige Arbeitszeit, eingeloggter Vorgesetzter	45
Tabelle 27: Use-Case 3.2: Zeiterfassung - Datum ohne bereits vorhandener Arbeitszeit, gültige Arbeitszeit, ausgeloggtter Vorgesetzter.....	46
Tabelle 28: Use-Case 3.3: Zeiterfassung - Datum mit bereits vorhandener Arbeitszeit, gültige Arbeitszeit	46
Tabelle 29: Use-Case 3.4: Zeiterfassung - Datum ohne bereits vorhandener Arbeitszeit, ungültige Arbeitszeit	47
Tabelle 30: Use-Case 3.5: Zeiterfassung - Datum mit bereits vorhandener Arbeitszeit, ungültige Arbeitszeit	47
Tabelle 31: Testfall 1 - Login - Korrekte Eingaben, Vorgesetzter	49
Tabelle 32: Testfall 2 - Login - Korrekter Nutzertyp, falscher Name, Vorgesetzter	49
Tabelle 33: Testfall 3 - Login - Falscher Nutzertyp, richtiger Name, Vorgesetzter	49
Tabelle 34: Testfall 4 - Login - Falscher Nutzertyp, falscher Name, Vorgesetzter	50

Tabelle 35: Testfall 5 - Login - Korrekte Eingaben, noch nicht vorhandener Mitarbeiter	50
Tabelle 36: Testfall 6 - Login - Korrekte Eingaben, bereits vorhandener Mitarbeiter	51
Tabelle 37: Testfall 7 - Login - Falscher Nutzertyp, gültiger Name, Mitarbeiter.....	51
Tabelle 38: Testfall 8 - Login - Richtiger Nutzertyp, ungültiger Name, Mitarbeiter	51
Tabelle 39: Testfall 9 - Login - Falscher Nutzertyp, ungültiger Name, Mitarbeiter	52
Tabelle 40: Testfall 10 - Arbeitszeitenansicht - Vorgesetzter, bereits erfasste Arbeitszeiten	52
Tabelle 41: Testfall 11 - Arbeitszeitenansicht - Vorgesetzter, keine bereits erfassten Arbeitszeiten ..	52
Tabelle 42: Testfall 12 - Arbeitszeitenansicht - Mitarbeiter, bereits erfasste Arbeitszeiten	52
Tabelle 43: Testfall 13 - Arbeitszeitenansicht - Mitarbeiter, keine bereits erfassten Arbeitszeiten	52
Tabelle 44: Testfall 14 - Zeiterfassung - Datum ohne bereits vorhandener Arbeitszeit, gültige Arbeitszeit, eingeloggter Vorgesetzter	53
Tabelle 45: Testfall 15 - Zeiterfassung - Datum ohne bereits vorhandener Arbeitszeit, gültige Arbeitszeit, ausgeloggtter Vorgesetzter.....	53
Tabelle 46: Testfall 16 - Zeiterfassung - Datum mit bereits vorhandener Arbeitszeit, gültige Arbeitszeit	54
Tabelle 47: Testfall 17 - Zeiterfassung - Datum ohne bereits vorhandener Arbeitszeit, ungültige Arbeitszeit	54
Tabelle 48: Testfall 18 - Zeiterfassung - Datum mit bereits vorhandener Arbeitszeit, ungültige Arbeitszeit	55
Tabelle 49: Testprotokoll	70
Tabelle 50: Glossar	75

4 Quellenverzeichnis

Hier werden alle verwendeten Quellen zur Informationsbeschaffung dargelegt.

5 Literaturverzeichnis

Hier werden alle verwendeten Literaturquellen zur Informationsbeschaffung dargelegt.

6 Programmcode

Der Code der umgesetzten Applikation.

6.1 Datenbank Export

```
001 -- phpMyAdmin SQL Dump
002 -- version 4.5.1
003 -- http://www.phpmyadmin.net
004 --
005 -- Host: 127.0.0.1
006 -- Erstellungszeit: 01. Feb 2017 um 13:37
007 -- Server-Version: 10.1.16-MariaDB
008 -- PHP-Version: 7.0.9
009
010 SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";
011 SET time_zone = "+00:00";
012
013
014 /*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
015 /*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
016 /*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
017 /*!40101 SET NAMES utf8mb4 */;
018
019 --
020 -- Datenbank: `db_time_recording_system`
021 --
022
023 DROP DATABASE IF EXISTS `db_time_recording_system`;
024 CREATE DATABASE `db_time_recording_system`;
025
026 -- -----
027
028 --
029 -- Tabellenstruktur für Tabelle `tbl_user`
030 --
031
032 CREATE TABLE `tbl_user` (
033   `id_user` int(11) NOT NULL,
034   `usertype` enum('Vorgesetzter','Mitarbeiter','','','') NOT NULL,
035   `name` varchar(30) NOT NULL
036 ) ENGINE=InnoDB DEFAULT CHARSET=latin1;
037
038 --
039 -- Daten für Tabelle `tbl_user`
040 --
041
042 INSERT INTO `tbl_user` (`id_user`, `usertype`, `name`) VALUES
043 (1, 'Vorgesetzter', 'Chef');
044
045 -- -----
046
047 --
048 -- Tabellenstruktur für Tabelle `tbl_working_hours`
049 --
050
051 CREATE TABLE `tbl_working_hours` (
052   `id_working_hours` int(11) NOT NULL,
053   `time_in_hours` float(10,2) NOT NULL,
```

```
054     `date` date NOT NULL,
055     `fs_user` int(11) NOT NULL
056 ) ENGINE=InnoDB DEFAULT CHARSET=latin1;
057
058 --
059 -- Indizes der exportierten Tabellen
060 --
061
062 --
063 -- Indizes für die Tabelle `tbl_user`
064 --
065 ALTER TABLE `tbl_user`
066     ADD PRIMARY KEY (`id_user`);
067
068 --
069 -- Indizes für die Tabelle `tbl_working_hours`
070 --
071 ALTER TABLE `tbl_working_hours`
072     ADD PRIMARY KEY (`id_working_hours`);
073
074 --
075 -- AUTO_INCREMENT für exportierte Tabellen
076 --
077
078 --
079 -- AUTO_INCREMENT für Tabelle `tbl_user`
080 --
081 ALTER TABLE `tbl_user`
082     MODIFY `id_user` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=26;
083 --
084 -- AUTO_INCREMENT für Tabelle `tbl_working_hours`
085 --
086 ALTER TABLE `tbl_working_hours`
087     MODIFY `id_working_hours` int(11) NOT NULL AUTO_INCREMENT;
088 /*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
089 /*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
090 /*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
```

6.2 ServiceConfig.java

```
001 package net.atos.zeiterfassungssystem;
002
003 import org.springframework.boot.SpringApplication;
004 import org.springframework.boot.autoconfigure.SpringBootApplication;
005
006 //Die Main-Klasse welche von Spring dazu verwendet wird um die gesamte
    Applikation inkl. aller Komponenten zu starten
007 @SpringBootApplication
008 public class ServiceConfig {
009     public static void main(String[] args) {
010         SpringApplication.run(ServiceConfig.class, args);
011     }
012
013 }
```

6.3 DBConfig.java

```
001 package net.atos.zeiterfassungssystem.data;
002
003 import javax.sql.DataSource;
004
005 import org.apache.log4j.Logger;
006 import org.springframework.beans.factory.annotation.Autowired;
007 import org.springframework.context.annotation.Bean;
008 import org.springframework.context.annotation.Configuration;
009 import org.springframework.context.annotation.PropertySource;
010 import org.springframework.core.env.Environment;
011 import org.springframework.jdbc.datasource.DriverManagerDataSource;
012
013 // Diese Klasse ist für die Konfiguration der Datenbank zuständig
014
015 @PropertySource(value = {
016     "classpath:application.properties" })
017 @Configuration
018 public class DBConfig {
019     private static final Logger logger = Logger.getLogger("log4j.properties");
020
021     @Autowired
022     private Environment env;
023
024     // Die DB-Eigenschaften aus dem "application.properties"-File werden
    geladen
025     // und in der DataSource gespeichert
026     @Bean
027     public DataSource dataSource() {
028         logger.info("Get database properties");
029
030         DriverManagerDataSource dataSource = new DriverManagerDataSource();
031
032         dataSource.setDriverClassName(env.getProperty("spring.datasource.driver-
    class-name"));
033         dataSource.setUrl(env.getProperty("spring.datasource.url"));
034
035         dataSource.setUsername(env.getProperty("spring.datasource.username"));
036
037         dataSource.setPassword(env.getProperty("spring.datasource.password"));
038         return dataSource;
039     }
040 }
```


037 }

6.4 IDBService.java

```
001 package net.atos.zeiterfassungssystem.data;
002
003 import java.util.Date;
004 import java.util.List;
005
006 import net.atos.zeiterfassungssystem.javaclasses.User;
007 import net.atos.zeiterfassungssystem.javaclasses.WorkingHours;
008
009 /*
010  * Das DB-Interface, welches durch die Klasse "DBServiceImpl" implementiert
011  * wird und für sämtliche
012  * Datenbank-Verbindungen verwendet wird.
013  */
014 public interface IDBService {
015     public List<User> getUser(String name);
016
017     public void createUser(String usertype, String name);
018
019     public List<WorkingHours> getWorkingHours(String name);
020
021     public List<WorkingHours> getWorkingHoursByDate(String name, String date);
022
023     public void addNewWorkingHours(int userID, Date date, float workingHours);
024 }
```

6.5 DBServiceImpl.java

```
001 package net.atos.zeiterfassungssystem.data;
002
003 import java.sql.ResultSet;
004 import java.sql.SQLException;
005 import java.util.Date;
006 import java.util.List;
007
008 import org.apache.log4j.Logger;
009 import org.springframework.beans.factory.annotation.Autowired;
010 import org.springframework.jdbc.core.JdbcTemplate;
011 import org.springframework.jdbc.core.RowMapper;
012 import org.springframework.stereotype.Service;
013
014 import net.atos.zeiterfassungssystem.javaclasses.User;
015 import net.atos.zeiterfassungssystem.javaclasses.WorkingHours;
016
017 /*
018  * Diese Klasse implementiert das Interface "IDBService" und stellt Methoden
019  * zur Verfügung, welche eine Verbindung
020  * zur Datenbank herstellen und bestimmte Querys absetzen
021  */
022 @Service
023 public class DBServiceImpl implements IDBService {
024     private static final Logger logger = Logger.getLogger("log4j.properties");
025
026     @Autowired
027     JdbcTemplate jdbcTemplate;
```

```

028  /*
029  * Diese Methode sucht in der Datenbank nach einem User, der einen
030  * bestimmten Namen hat Den gefundenen User gibt sie in einer Objektliste
031  * von Usern zurück
032  */
033  @Override
034  public List<User> getUser(String name) {
035      logger.info(String.format("Get user with the name %s from database",
name));
036
037      return jdbcTemplate.query("Select * from tbl_user Where name = ?",
new RowMapper<User>() {
038          public User mapRow(ResultSet rs, int rowNum) throws
SQLException {
039              User user = new User();
040              user.setId_user(rs.getInt("id_user"));
041              user.setUserType(rs.getString("usertype"));
042              user.setName(rs.getString("name"));
043              return user;
044          }
045      }, name);
046
047  }
048
049  /*
050  * Diese Methode speichert einen neuen Benutzer in der Datenbank ab
051  */
052  @Override
053  public void createUser(String usertype, String name) {
054      jdbcTemplate.update("Insert into tbl_user (usertype,name) values
(?,?)", usertype, name);
055
056      logger.info(String.format("User named %s with usertype %s created",
name, usertype));
057  }
058
059  /*
060  * Diese Methode lädt alle Arbeitszeiten, im Falle dass der Benutzer ein
061  * Vorgesetzter ist, oder lädt nur die Arbeitszeiten des jeweiligen
062  * Benutzers aus der Datenbank, und gibt diese in einer Objektenlisten von
063  * WorkingHours-Objekten zurück
064  */
065  @Override
066  public List<WorkingHours> getWorkingHours(String name) {
067      logger.info(String.format("Select working hours for user %s",
name));
068
069      // im Falle des Vorgesetzten..
070      if (name.contains("Chef")) {
071          return jdbcTemplate.query("Select * from tbl working hours,
tbl user Where tbl user.id user = tbl working hours.fs user",
new RowMapper<WorkingHours>() {
072              public WorkingHours mapRow(ResultSet rs,
int rowNum) throws SQLException {
073                  WorkingHours workingHours = new
WorkingHours();
074                  workingHours.setIdWorkingHours(rs.getInt("id user"));

```

```
076         workingHours.setDate(rs.getDate("date"));
077         workingHours.setWorkingHours(rs.getFloat("time_in_hours"));
078         workingHours.setName(rs.getString("name"));
079         return workingHours;
080     }
081 }
082 }
083 // im Falle eines Mitarbeiters..
084 else {
085     return jdbcTemplate.query(
086         "Select * from tbl_working_hours, tbl_user Where",
087         new RowMapper<WorkingHours>() {
088             public WorkingHours mapRow(ResultSet rs,
089 int rowNum) throws SQLException {
089                 WorkingHours workingHours = new
WorkingHours();
090                 workingHours.setIdWorkingHours(rs.getInt("id_user"));
091                 workingHours.setDate(rs.getDate("date"));
092                 workingHours.setWorkingHours(rs.getFloat("time_in_hours"));
093                 workingHours.setName(rs.getString("name"));
094                 return workingHours;
095             }
096         }, name);
097     }
098 }
099
100 /*
101  * Diese Methode sucht in der Datenbank nach einer Arbeitszeit mit dem
102  * angegebenen Namen und Datum
103  */
104 @Override
105 public List<WorkingHours> getWorkingHoursByDate(String name, String date) {
106     return jdbcTemplate.query(
107         "Select * from tbl_working_hours, tbl_user Where",
108         new RowMapper<WorkingHours>() {
109             public WorkingHours mapRow(ResultSet rs, int
rowNum) throws SQLException {
110                 WorkingHours workingHours = new
WorkingHours();
111                 workingHours.setIdWorkingHours(rs.getInt("id_user"));
112                 workingHours.setDate(rs.getDate("date"));
113                 workingHours.setWorkingHours(rs.getFloat("time_in_hours"));
114                 workingHours.setName(rs.getString("name"));
115                 return workingHours;
116             }
117         }, name, date);
```

```
118     }
119
120     /*
121     * Diese Methode speichert die neu erfasste Arbeitszeit in der Datenbank ab
122     */
123     @Override
124     public void addNewWorkingHours(int userID, Date date, float workingHours) {
125         jdbcTemplate.update("Insert into tbl_working_hours
126         (time_in_hours,date,fs_user) values (?,?,?)", workingHours, date, userID);
127         logger.info(String.format("New working hours added for %s from user
128         with id %s", date.toString(), String.valueOf(userID)));
129     }
130 }
```

6.6 User.java

```
001 package net.atos.zeiterfassungssystem.javaclasses;
002
003 // Die User-Klasse die vom RowMapper verwendet wird
004
005 public class User {
006     int id_user;
007
008     String usertype;
009
010     String name;
011
012     public int getId_user() {
013         return id_user;
014     }
015
016     public void setId_user(int id_user) {
017         this.id_user = id_user;
018     }
019
020     public String getUsertype() {
021         return usertype;
022     }
023
024     public void setUsertype(String usertype) {
025         this.usertype = usertype;
026     }
027
028     public String getName() {
029         return name;
030     }
031
032     public void setName(String name) {
033         this.name = name;
034     }
035 }
```

6.7 WorkingHours.java

```
001 package net.atos.zeiterfassungssystem.javaclasses;
002
003 import java.util.Date;
```

```
004
005 // Die WorkingHours-Klasse welche vom RowMapper verwendet wird
006
007 public class WorkingHours {
008     int idWorkingHours;
009
010     Date date;
011
012     float workingHours;
013
014     String name;
015
016     public int getIdWorkingHours() {
017         return idWorkingHours;
018     }
019
020     public void setIdWorkingHours(int idWorkingHours) {
021         this.idWorkingHours = idWorkingHours;
022     }
023
024     public Date getDate() {
025         return date;
026     }
027
028     public void setDate(Date date) {
029         this.date = date;
030     }
031
032     public float getWorkingHours() {
033         return workingHours;
034     }
035
036     public void setWorkingHours(float workingHours) {
037         this.workingHours = workingHours;
038     }
039
040     public String getName() {
041         return name;
042     }
043
044     public void setName(String name) {
045         this.name = name;
046     }
047 }
```

6.8 WebSocketConfiguration.java

```
001 package net.atos.zeiterfassungssystem.web;
002
003 import org.apache.log4j.Logger;
004 import org.springframework.beans.factory.annotation.Autowired;
005 import org.springframework.context.annotation.Configuration;
006 import org.springframework.web.socket.config.annotation.EnableWebSocket;
007 import org.springframework.web.socket.config.annotation.WebSocketConfigurer;
008 import
009 org.springframework.web.socket.config.annotation.WebSocketHandlerRegistry;
010 /*
```

```

011  * Die WebSocketConfiguration-Klasse, welche das Spring-Interface
012  * und den zugehörigen Handler registriert.
013  */
014
015  @Configuration
016  @EnableWebSocket
017  public class WebSocketConfiguration implements WebSocketConfigurer {
018      private static final Logger logger = Logger.getLogger("log4j.properties");
019
020      @Autowired
021      protected WebSocketHandler websocketHandler;
022
023      @Override
024      public void registerWebSocketHandlers(WebSocketHandlerRegistry registry) {
025          registry.addHandler(websocketHandler, "/zeiterfassungssystem");
026
027          logger.info("WebSocket-Handler registered");
028      }
029
030  }

```

6.9 WebSocketHandler.java

```

001  package net.atos.zeiterfassungssystem.web;
002
003  import java.util.ArrayList;
004  import java.util.List;
005
006  import org.apache.log4j.Logger;
007  import org.springframework.stereotype.Component;
008  import org.springframework.web.socket.CloseStatus;
009  import org.springframework.web.socket.TextMessage;
010  import org.springframework.web.socket.WebSocketSession;
011  import org.springframework.web.socket.handler.TextWebSocketHandler;
012
013  /*
014  * Die WebSocketHandler-Klasse, welche von der Spring-Klasse
015  * "TextWebSocketHandler" erbt und sämtliche für den
016  * WebSocket benötigten Methoden implementiert
017  */
018  @Component
019  public class WebSocketHandler extends TextWebSocketHandler {
020      private static final Logger logger = Logger.getLogger("log4j.properties");
021
022      private List<WebSocketSession> activeSessions = new ArrayList<>();
023
024      /*
025      * Falls während der WebSocket-Verbindung ein Fehler auftritt, wird diese
026      * Methode aufgerufen Sie loggt den dabei aufgetretenen Fehler in ein
027      * Log-File
028      */
029      @Override
030      public void handleTransportError(WebSocketSession session, Throwable
031      throwable) throws Exception {
032          logger.error("error occurred at sender " + session, throwable);
033      }
034  }

```

```

034     * Diese Methode wird aufgerufen, wenn eine WebSocket-Session geschlossen
035     * wurde Sie entfernt die geschlossene Session von der activeSessions-Liste
036     */
037     @Override
038     public void afterConnectionClosed(WebSocketSession session, CloseStatus
status) throws Exception {
039         activeSessions.remove(session);
040
041         logger.info(String.format("Session %s closed because of %s",
session.getId(), status.getReason()));
042     }
043
044     /*
045     * Diese Methode wird aufgerufen, wenn eine neue WebSocket-Verbindung
046     * hergestellt wurde Sie fügt die neu Session der activeSessions-Liste
047     * hinzu
048     */
049     @Override
050     public void afterConnectionEstablished(WebSocketSession session) throws
Exception {
051         activeSessions.add(session);
052
053         logger.info("Connected ... " + session.getId());
054     }
055
056     /*
057     * Diese Methode wird von einer REST-Webservice-Methode aufgerufen, wenn
ein
058     * Benutzer neue Arbeitszeiten erfasst hat und der Vorgesetzte angemeldet
059     * ist Sie benachrichtigt die sich in der activeSessions-Liste befindenden
060     * Sessions, sofern sich eine Session darin befindet
061     */
062     public void notifyObservers() throws Exception {
063         if (!activeSessions.isEmpty()) {
064             for (WebSocketSession webSocketSession : activeSessions) {
065                 TextMessage message = new TextMessage("New update");
066                 webSocketSession.sendMessage(message);
067
068                 logger.info(String.format("Session %s notified because
of changes", webSocketSession.getId()));
069             }
070         }
071     }
072 }

```

6.10 Service.java

```

001 package net.atos.zeiterfassungssystem.webservices;
002
003 import java.text.SimpleDateFormat;
004 import java.util.Date;
005 import java.util.List;
006
007 import org.apache.log4j.Logger;
008 import org.springframework.beans.factory.annotation.Autowired;
009 import org.springframework.stereotype.Controller;
010 import org.springframework.web.bind.annotation.RequestMapping;
011 import org.springframework.web.bind.annotation.RequestMethod;
012 import org.springframework.web.bind.annotation.RequestParam;

```

```
013 import org.springframework.web.bind.annotation.ResponseBody;
014
015 import net.atos.zeiterfassungssystem.data.IDBService;
016 import net.atos.zeiterfassungssystem.javaclasses.User;
017 import net.atos.zeiterfassungssystem.javaclasses.WorkingHours;
018 import net.atos.zeiterfassungssystem.web.WebSocketHandler;
019
020 /*
021  * Die Service-Klasse stellt den REST-Service dar und stellt die benötigten
022  * Methoden zur Verfügung,
023  * welche vom Client aufgerufen werden können
024  */
025 @Controller
026 @RequestMapping("/zeiterfassungssystem")
027 public class Service {
028     @Autowired
029     private IDBService dbcon;
030
031     @Autowired
032     private WebSocketHandler websocket;
033
034     private static final Logger logger = Logger.getLogger("log4j.properties");
035
036     /*
037     * Diese Methode loggt den Benutzer ein und antwortet dem Client mit einem
038     * "true" oder "false" je nachdem, ob die Anmeldedaten korrekt waren oder
039     * nicht
040     */
041     @RequestMapping(path = "/login", method = RequestMethod.POST)
042     public @ResponseBody boolean login(@RequestParam(value = "usertype") String
043     usertype, @RequestParam(value = "name") String name) {
044         System.out.println(name);
045
046         name = name.trim();
047
048         System.out.println(name);
049
050         if (usertype.equals("Vorgesetzter")) {
051             if (name.equals("Chef")) {
052                 return true;
053             } else {
054                 return false;
055             }
056         } else {
057             if (name.equals("")) {
058                 return false;
059             } else {
060                 if (name.equals("Chef")) {
061                     return false;
062                 } else {
063                     if (dbcon.getUser(name).isEmpty()) {
064                         dbcon.createUser(usertype, name);
065                         return true;
066                     } else {
067                         return true;
068                     }
069                 }
070             }
071         }
072     }
073 }
```



```
070     }
071
072     /*
073     * Diese Methode lädt die Arbeitszeiten dieses Benutzers aus der Datenbank
074     * und gibt sie als Objektliste zurück
075     */
076     @RequestMapping(path = "/getworkinghours", method = RequestMethod.GET)
077     public @ResponseBody List<WorkingHours> getWorkingHours(@RequestParam(value = "name") String name) {
078         return dbcon.getWorkingHours(name);
079     }
080
081     /*
082     * Diese Methode speichert die neuen Arbeitszeiten in der Datenbank ab und
083     * gibt dem Client ein "true" oder "false" zurück je nachdem ob die Eingaben
084     * gültig sind oder nicht
085     */
086     @RequestMapping(path = "/addnewworkinghours", method = RequestMethod.POST)
087     public @ResponseBody boolean addNewWorkingHours(@RequestParam(value = "name") String name, @RequestParam(value = "date") String dateParam,
088         @RequestParam(value = "workinghours") String
089         workingHoursParam) throws Exception {
090         if (name.isEmpty()) {
091             return false;
092         } else {
093             if (dateParam.isEmpty()) {
094                 return false;
095             } else {
096                 if (workingHoursParam.isEmpty()) {
097                     return false;
098                 } else {
099                     List<User> user = dbcon.getUser(name);
100                     int id_user = user.get(0).getId_user();
101
102                     SimpleDateFormat dateFormatter = new
103                     SimpleDateFormat("yyyy-mm-dd");
104
105                     Date date;
106
107                     try {
108                         date =
109                         dateFormatter.parse(dateParam.trim());
110                     } catch (Exception e) {
111                         return false;
112                     }
113
114                     Float workingHours;
115
116                     try {
117                         workingHours =
118                         Float.parseFloat(workingHoursParam);
119                     } catch (Exception e) {
120                         return false;
121                     }
122
123                     if (workingHours >= 0 && workingHours <= 12) {
124                         if (dbcon.getWorkingHoursByDate(name,
125                             dateParam).isEmpty()) {
```

```
121         dbcon.addNewWorkingHours(id_user,  
122         date, workingHours);  
123  
124         websocket.notifyObservers();  
125         return true;  
126     } else {  
127         return false;  
128     }  
129 } else {  
130     return false;  
131 }  
132 }  
133 }  
134 }  
135 }  
136 }
```

6.11 ServiceTest.java

```
001 import static org.junit.Assert.assertEquals;
002 import static org.mockito.Matchers.anyString;
003 import static org.mockito.Mockito.doNothing;
004 import static org.mockito.Mockito.doReturn;
005 import static org.mockito.Mockito.verify;
006
007 import java.util.ArrayList;
008 import java.util.List;
009
010 import org.junit.Test;
011 import org.junit.runner.RunWith;
012 import org.mockito.InjectMocks;
013 import org.mockito.Mock;
014 import org.mockito.runners.MockitoJUnitRunner;
015
016 import net.atos.zeiterfassungssystem.data.IDBService;
017 import net.atos.zeiterfassungssystem.javaclasses.User;
018 import net.atos.zeiterfassungssystem.webservices.Service;
019
020 /*
021  * Die zugehörige Testklasse zur "Service"-Klasse
022  * Sie testet die Login-Methode mit einigen Szenarien
023  */
024 @RunWith(MockitoJUnitRunner.class)
025 public class ServiceTest {
026     @InjectMocks
027     private Service service = new Service();
028
029     @Mock
030     private IDBService dbcon;
031
032     /*
033      * Wenn sich ein Vorgesetzter mit gültigen Daten anmeldet
034      */
035     @Test
036     public void loginSuperiorValidTest() {
037         String usertype = "Vorgesetzter";
038         String name = "Chef";
039
040         assertEquals(true, service.login(usertype, name));
041     }
042
043     /*
044      * Wenn sich ein Vorgesetzter mit dem falschen Namen anmeldet
045      */
046     @Test
047     public void loginSuperiorInvalidTest() {
048         String usertype = "Vorgesetzter";
049         String name = "Chief";
050
051         assertEquals(false, service.login(usertype, name));
052     }
053
054     /*
055      * Wenn sich ein bereits existierender Mitarbeiter mit gültigen Daten
056      * anmeldet
057      */
058     @Test
```

```

059     public void loginExistingEmployeeValidTest() {
060         String usertype = "Mitarbeiter";
061         String name = "Testuser";
062
063         List<User> userList = new ArrayList<User>();
064         User user = new User();
065         user.setId_user(1);
066         user.setName("Testuser");
067         user.setUsertype("Mitarbeiter");
068
069         userList.add(user);
070
071         doReturn(userList).when(dbcon).getUser(anyString());
072
073         assertEquals(true, service.login(usertype, name));
074
075         verify(dbcon).getUser(anyString());
076     }
077
078     /*
079     * Wenn sich ein noch nicht existierender Mitarbeiter mit gültigen Daten
080     * anmeldet
081     */
082     @Test
083     public void loginNotExistingEmployeeValidTest() {
084         String usertype = "Mitarbeiter";
085         String name = "Testuser";
086
087         List<User> userList = new ArrayList<User>();
088
089         doReturn(userList).when(dbcon).getUser(anyString());
090         doNothing().when(dbcon).createUser(anyString(), anyString());
091
092         assertEquals(true, service.login(usertype, name));
093
094         verify(dbcon).getUser(anyString());
095         verify(dbcon).createUser(anyString(), anyString());
096     }
097
098     /*
099     * Wenn sich ein Mitarbeiter ohne Namen anmelden möchte
100     */
101     @Test
102     public void loginEmployeeInvalidTest() {
103         String usertype = "Mitarbeiter";
104         String name = "";
105
106         assertEquals(false, service.login(usertype, name));
107     }
108

```

6.12 application.properties

```

001 spring.datasource.platform=mysql
002 spring.datasource.url=jdbc:mysql://localhost:3306/db_time_recording_system
003 spring.datasource.username=root
004 spring.datasource.password=
005 spring.datasource.driver-class-name=com.mysql.jdbc.Driver

```

6.13 log4j.properties

```
001 log4j.rootLogger=INFO, R
002 log4j.appender.R=org.apache.log4j.RollingFileAppender
003 log4j.appender.R.File=log/zeiterfassungssystem.log
004 log4j.appender.R.MaxFileSize=100KB log4j.appender.R.MaxBackupIndex=2
005 log4j.appender.R.layout=org.apache.log4j.PatternLayout
006 log4j.appender.R.layout.ConversionPattern=%r [%t] %5p (%F:%L) - %m%n
```

6.14 index.html

```

001 <!--Zeiterfassungssystem - Webfrontend-->
002
003 <!DOCTYPE html>
004 <html lang="de">
005     <head>
006         <meta charset="utf-8"/>
007
008         <title>Zeiterfassungssystem</title>
009
010         <link rel="stylesheet"
href="//cdn.datatables.net/1.10.12/css/jquery.dataTables.min.css"></link>
011         <link rel="stylesheet"
href="//code.jquery.com/ui/1.12.1/themes/base/jquery-ui.css">
012         <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.1.0/jquery.min.js"></sc
ript>
013         <script
src="//cdn.datatables.net/1.10.12/js/jquery.dataTables.min.js"></script>
014         <script src="https://code.jquery.com/ui/1.12.1/jquery-
ui.js"></script>
015
016         <script src="_js/timetable.js"></script>
017         <script src="_js/datepicker.js"></script>
018         <script src="_js/script.js"></script>
019     </head>
020
021     <body onload="showAndHideElements();">
022
023         <div id="header">
024             <h1>Zeiterfassungssystem</h1>
025         </div>
026
027         <div id="content">
028
029             <!--Das Login-Formular-->
030             <div id="loginView">
031
032                 <h2>Login</h2>
033                 <p>Benutzertyp:</p>
034                 <select id="usertype" name="usertype" size="1">
035                     <option>Vorgesetzter</option>
036                     <option>Mitarbeiter</option>
037                 </select>
038                 <p>Name:</p>
039                 <input type="text" id="name" name="name"/><br/><br/>
040                 <input type="button" id="loginButton"
name="loginButton" value="Login" onclick="login();" />
041
042             </div>
043
044             <!--Die Arbeitszeitentabelle-->
045             <div id="timetableView">
046
047                 <h2>Arbeitszeitenansicht</h2>
048                 <table id="timetable" name="timetable" class="display"
cellspacing="0" width="100%"></table>
049
050             </div>
051

```

```
052         <!--Das Formular um neue Arbeitszeiten zu erfassen-->
053         <div id="newWorkingHoursView">
054
055             <h2>Neue Arbeitszeit erfassen</h2>
056             <p>Datum:</p>
057             <input type="text" id="datepicker" name="date">
058             <p>Arbeitszeit in h:</p>
059             <input type="text" id="workingHours"
name="workingHours"/></p>
060             <input type="button" id="saveButton" name="saveButton"
value="Speichern" onclick="saveData();" />
061
062         </div>
063
064     </div>
065 </body>
066 </html>
```

6.15 script.js

```
001 /*
002 * Das Haupt-Javascript-File, welches allgemeine Funktionen beinhaltet
003 */
004
005 var setName;
006 var wsconnection;
007
008 var tableData;
009 var table;
010
011 /*
012 * Funktion welche die die DIV-Container ein- und/oder ausblendet
013 */
014 function showAndHideElements ()
015 {
016     if(setName == null)
017     {
018         $('#loginView').show();
019         $('#timetableView').hide();
020         $('#newWorkingHoursView').hide();
021     }
022     else
023     {
024         $('#loginView').hide();
025         $('#timetableView').show();
026
027         if(setName == "Chef")
028         {
029             $('#newWorkingHoursView').hide();
030         }
031         else
032         {
033             $('#newWorkingHoursView').show();
034         }
035     }
036 }
037
038 /*
039 * Funktion welche den Benutzer einloggt
040 */
041 function login()
```

```
042 {
043     var name = $('#name').val();
044     var usertype = $('#usertype :selected').text();
045
046     $.ajax
047     ({
048         url:
"http://localhost:8080/zeiterfassungssystem/login?usertype="+usertype+"&name="+name,
049         method: 'POST',
050         dataType: "json",
051         success: function(resultData)
052         {
053             if(JSON.parse(resultData) == true)
054             {
055                 setName = name;
056                 showAndHideElements();
057                 if(setName == "Chef")
058                 {
059                     startWebSocket();
060                 }
061                 loadData();
062             }
063             else
064             {
065                 alert("Die Anmeldedaten sind inkorrekt!");
066             }
067         },
068         error: function(errMsg)
069         {
070             alert(errMsg);
071         }
072     })
073 }
074
075 /*
076 * Funktion welche beim Vorgesetzten den Web-Socket startet bzw. zu
077 * diesem eine Verbindung aufbaut
078 */
079 function startWebSocket()
080 {
081     wsconnection = new
WebSocket('ws://localhost:8080/zeiterfassungssystem');
082     wsconnection.onmessage = function(){loadData()};
083 }
084
085 /*
086 * Funktion welche die neu erfassten Arbeitszeiten an den Server sendet
087 */
088 function saveData()
089 {
090     var date = $('#datepicker').val();
091     var workingHours = $('#workingHours').val();
092
093     $.ajax
094     ({
095         url:
"http://localhost:8080/zeiterfassungssystem/addnewworkinghours?name="+setName+"&date="+date+"&workinghours="+workingHours,
096         method: 'POST',
```



```
097     dataType: 'json',
098     success: function(resultData)
099     {
100         if(JSON.parse(resultData) == true)
101         {
102             loadData();
103         }
104         else
105         {
106             alert("Ungültige Arbeitszeiten!");
107         }
108     },
109     error: function(errMsg)
110     {
111         alert(errMsg);
112     }
113 })
114 }
115
116 /*
117 * Funktion welche die Arbeitszeiten vom Server abfragt und die jQuery-
118 * Datatable mit den Daten befüllt
119 */
120 function loadData()
121 {
122     $.ajax
123     ({
124         url:
125         'http://localhost:8080/zeiterfassungssystem/getworkinghours?name='+setName,
126         method: 'get',
127         dataType: 'json',
128         success: function(data)
129         {
130             tableData = data;
131             table.clear();
132             table.rows.add(tableData).draw();
133         },
134         error: function(errMsg)
135         {
136             alert(errMsg);
137         }
138     })
139 }
```

6.16 timetable.js

```
001 /*
002 * Wenn die Seite geladen wurde, wird die jQuery-Datatable initialisiert
003 */
004 $(document).ready(function()
005 {
006     table = $('#timetable').DataTable
007     ({
008         columns :
009         [
010             {
011                 title : "Datum",
012                 data : "date"
013             },
014             {
015                 title : "Arbeitszeit in Stunden",
```

```
016         data : "workingHours"
017     },
018     {
019         title : "Name",
020         data : "name"
021     }
022 ]
023 });
024 });
```

6.17 datepicker.js

```
001 /*
002 * Die Funktion welcher den jQuery-Datepicker initialisiert
003 */
004 $( function()
005 {
006     $( "#datepicker" ).datepicker({dateFormat: 'yy-mm-dd'});
007 });
```