SIRUM

Dokumentation - Improvement der Darstellung der Zeiterfassung im Odoo Framework

Diese Dokumentation wurde erstellt von:
Jona Nitsch
Breiter Gang 7, 20355 Hamburg
E-Mail: Jona.Nitsch.JN@gmail.com
Ausbildung zum Fachinformatiker für Anwendungsentwicklung

Ausbildungsbetrieb:
BFW-Berufsförderungswerk Hamburg GmbH
Marie-Bautz-Weg 16, 22159 Hamburg
Ansprechpartner: Dr. Olaf Kubillus

Betrieb für die Praktische Ausbildung:
Sirum GmbH
Am Sandtorkai 32, 20457 Hamburg
Ansprechpartner: Dipl. Ing. (FH) Georg Notter
E-Mail: Georg.Notter@Sirum.de

Inhaltsverzeichnis
Seite

Einleitung

Sirum – Innovative Lösungen für die Transportlogistik

Das 2016 gegründete Unternehmen **Sirum** hat sich zum Ziel gesetzt, eine innovative Softwarelösung für die Transportlogistik zu entwickeln, die effizienter und umfassender ist als bestehende Ansätze. Während herkömmliche Lösungen oft nur einzelne Aspekte der Logistik abdecken, fokussiert sich Sirum auf eine ganzheitliche Optimierung, um Einschränkungen in der Gesamteffizienz zu überwinden.

Die **webbasierte Struktur** der Software ermöglicht eine flexible Anpassung und Erweiterung für unterschiedliche Anwendungsfälle. Mit etwa 30 Mitarbeitenden hat Sirum bereits eine breite Kundenbasis aufgebaut, die sowohl den deutschen Markt als auch internationale Märkte umfasst. Das Unternehmen verfügt über zwei Standorte, den **Hauptsitz Hamburg**, in der Speicherstadt im Digital Hub, und einen Standort in **Fellbach**, nahe Stuttgart.

Unternehmenswerte und Arbeitskultur
Sirum legt besonderen Wert auf Offenheit, Authentizität, Wertschätzung, Transparenz und Proaktivität. Diese Werte prägen den Arbeitsalltag und bilden die Grundlage für den Umgang mit Kunden und Mitarbeitenden.

Technologische Innovation: Das Sirum TMS
Mit dem **Sirum Transport Management System (TMS)** wurde eine umfassende Softwarelösung entwickelt, die auf dem Odoo-Framework basiert. Sie lässt sich flexibel an Kundenanforderungen anpassen und kombiniert verschiedene Module:

- **TMS** (Transport Management System)
- **DMS** (Document Management System)
- **ERP** (Enterprise Resource Planning)
- **WMS** (Warehouse Management System)

Auf Basis des **Docker-Prinzips** bietet die Software eine skalierbare, modulare und leistungsfähige Architektur, die optimal auf die Bedürfnisse von Unternehmen zugeschnitten ist.
Kundenstamm und Marktpotenzial
Der aktuelle Kundenstamm besteht überwiegend aus mittelständischen Unternehmen, die ihre Auftrags- und Transportverwaltung effizienter gestalten möchten. Gleichzeitig zeigen auch Großunternehmen Interesse und planen, ihre Systeme in den kommenden Jahren auf das Sirum TMS umzustellen.

SaaS-Architektur für Kostentransparenz
Sirum setzt auf eine **SaaS (Software as a Service)**-Architektur, die eine transparente und kalkulierbare Kostenstruktur ermöglicht. Im Vergleich zu herkömmlichen Serverlösungen, die mit Wartungs-, Strom- und anderen unvorhersehbaren Kosten verbunden sind, bietet SaaS eine höhere Planungssicherheit. Diese klare Kostenbasis hilft Kunden, ihre logistischen Prozesse stabil und effizient zu steuern, während unvorhersehbare Ausgaben reduziert werden.

Der Auftraggeber für das Projekt ist der Geschäftsführer von Sirum, hierbei handelt es sich um eine Interne Erweiterung die erstellt werden soll.

Problembeschreibung

Mitte 2024 kam es zur Kurzarbeit aufgrund einer knappen Auftragslage bei der Firma Sirum. Die Lage hat sich seitdem wieder entspannt doch bei der Retrospektive fiel auf das man schlecht auf die Herausforderungen vorbereitet war. Insbesondere gab es Probleme mit der Zeiterfassung, da Mitarbeitern eine klare Übersicht fehlte wie lange sie schon angemeldet waren. So kam es teilweise zu

Überstunden in der Kurzarbeit, welche nach Auflagen von der Agentur für Arbeit nicht zulässig sind. Somit entsteht bei solchen Fällen ein Schaden für das Unternehmen da der Antrag für Erstattung gänzlich oder in Teilen abgelehnt werden kann.

Projektbeschreibung

Das neue Modul soll die bereits vorhandenen Anmeldedaten der Mitarbeitenden verarbeiten, um den Anforderungen geänderter Arbeitsbedingungen – insbesondere in Zeiten von Kurzarbeit – gerecht zu werden. Ziel ist es, Szenarien wie diese in Zukunft einfacher bewältigen zu können. Eine der zentralen Neuerungen betrifft die visuelle Darstellung der Arbeitszeiten. Bisher fehlte eine Übersicht, was die eigenständige Einschätzung der Arbeitszeiten erschwerte und umständliche Berechnungen erforderte.

Die Überwachung der Anmeldezeiten ist ausschließlich für die Mitarbeitenden gedacht, damit diese ihre Arbeitszeiten eigenständig nachvollziehen können. Dabei wird ein System benötigt, das sowohl präzise als auch flexibel genug ist, um künftigen Herausforderungen gewachsen zu sein. Während Überstunden in der Vergangenheit unproblematisch waren, erfordert die veränderte Situation eine strukturierte und transparente Lösung.

Eine zentrale technische Herausforderung besteht in der korrekten Erfassung der Arbeitszeiten. Das Framework unterscheidet zwischen Server-Zeit und User-Zeit, wobei beide Werte in das System einfließen. Der Server verwendet eine fixe Zeitzone, die von seinem Standort abhängt, während für die User-Zeit die Zeitzone des aktuellen Standorts der Mitarbeitenden herangezogen wird. Dieser Aspekt ist besonders wichtig, da Mitarbeitende teilweise längere Zeit im Ausland arbeiten können. Beispielsweise arbeitete ein Mitarbeitender mehrere Monate in den USA. Dies ist durch den hohen Remote Anteil innerhalb des Unternehmens möglich. Um solche Fälle korrekt abzubilden, muss das System jeden Tag einzeln betrachten können – auch wenn Arbeitszeiten über Mitternacht hinausgehen. Eine Synchronisierung der Zeiten sowie die Berücksichtigung der unterschiedlichen Anforderungen sind daher unverzichtbar.

Für die Integration der vorhandenen Anmeldedaten ist eine korrekte Einbindung in die Logik des Frameworks notwendig. Dies kann über einen Depend- oder Inherit-Mechanismus erfolgen. Aufgrund der Vielzahl an Modulen, deren Funktionalitäten oder Werte genutzt werden, ist vermutlich eine Kombination dieser Ansätze sinnvoll.

Zusätzlich muss entschieden werden, ob das Modul auf einer bestehenden Grundlage entwickelt oder speziell zugeschnitten werden soll, um eine spätere Weiterverwendung zu erleichtern. Eine maßgeschneiderte Lösung kann flexiblere Anpassungen ermöglichen, während die Nutzung bestehender Komponenten Zeit und Ressourcen spart.

Ein weiteres wichtiges Feature ist die Implementierung einer AFK-Funktion (Away From Keyboard), welche die Systemsicherheit stärkt. Mithilfe eines JavaScript-Timers sollen alle möglichen Aktionen erfasst werden, um inaktive Sitzungen zu erkennen. Dabei erfolgt jedoch keine langfristige Speicherung der Daten, um Verstöße gegen arbeitsrechtliche Vorgaben zu vermeiden.

Ist-Analyse

Derzeit wird das Odoo-Framework zur Erfassung von Arbeitszeiten genutzt. Dieses Framework wurde als Webclient implementiert, wodurch Mitarbeitende ortsunabhängig auf das System zugreifen können.
Aktuell steht ein Standardmodul zur Verfügung, das die Erfassung von Anfangs- und Endzeiten ermöglicht. Diese Zeiten werden über einen Anmeldebutton im System registriert, der nach der Anmeldung automatisch in einen Logout-Button umgewandelt wird.

Das Odoo-Framework bietet eine integrierte Datenbank, die flexibel durch neue Module erweitert werden kann. Beim Erstellen eines Moduls generiert das System automatisch eine neue Tabelle mit

den erforderlichen Parametern, die durch Variablen in der Klassenstruktur definiert werden. Dabei ist lediglich der Datentyp der Variablen (z. B. Boolean, Float, String) festzulegen.

Die aktuell erfassten Daten beschränken sich auf die Grundfunktionen und sehen keine weitere Verarbeitung oder Auswertung vor. Eine visuelle Darstellung der Arbeitszeiten oder weiterführende Funktionen wie Analysen und Berichte sind bisher nicht Teil des Systems.
Derzeitige Daten die bereits vorhanden sind unter anderem die Serverzeiten, An- und Abmeldedaten. Hierzu kommen noch Personaldaten wovon die Mitarbeiter ID besonders interessant ist.

Daraus leiten sich derzeit folgende Probleme ab:
Es erfolgt keine aktuelle Anzeige der Arbeitszeit, was zu Problemen mit den Arbeitszeitgesetz in Bezug auf Überstunden oder aber Kurzarbeit führen kann.

Die fehlende Sichtbarkeit der aktuellen Arbeitszeit erschwert auch die Einhaltung von Pausen, die der Mitarbeitende aufgrund der lockeren Gleitzeitregelung eigenständig organisieren muss. Je nach Kundenterminen kann es zudem in seltenen Fällen vorkommen, dass die Arbeit spät begonnen wird. Was dazu führen kann, dass über Mitternacht hinaus gearbeitet wird. Diesen Terminen ist eben auch die Selbstverantwortlichkeit der Pausen geschuldet da ein Logout während eines Kundentermins ein absolutes No-Go ist.

Soll-Zustand

Durch die im vorausgehenden Text erläuterten Probleme, wurden die wichtigsten Punkte heraus analysiert und konkrete Ziele formuliert. Hierbei wurden folgende Kriterien als Kernpunkte herausgearbeitet deren Erfüllung es zu beachten gilt.

Pflichtkriterien:

- Darstellung der aktuellen Arbeitszeit (Ausgabe der Daten)

- Optionen zum Automatischen Ausloggen (Automatische Abmeldung)

- Anpassbarkeit der Logout Bedingungen (Automatische Abmeldung)

- Beachtung der Server-Zeiten und User-Zeiten

- Berechnung der Arbeitszeiten pro Tag

- Einhaltung der Coding Guidelines

- Erstellen von Unittests mit 100% Coverrage

Es soll eine Option zur manuellen Auswahl geben um ein automatisches Ausloggen nach 15-Minuten zu ermöglichen. Die Anpassbarkeit der Logout Bedingungen ist notwendig um verschiedene Arbeitszeitmodelle abbilden zu können.
Das Grund Modul welches es hierbei zu erweitern gilt ist das HR-Employee Modul. Welches grundlegende Strukturen für die Datenbank liefert, wie z.b. Mitarbeiter Nummern.

Wobei aber auch ein gewisser Part in das Modul Attendance verordnet werden muss um eine volle Funktionalität zu gewährleisten.

Planung

Ressourcen:

Folgende Ressourcen werden für das Projekt bereitgestellt. Ein Laptop mit Funktastatur,Maus und Headset. Ein Internetzugang wie auch ein Zugang zu Gitlab zur Versionierung des Projektes und um Zwischenstände zu sichern.
Es wurde eine Zeitplanung für eine Kalkulation erstellt. Es sind insgesamt 80 Stunden vorgesehen und wie folgt verplant.

Projektphasen mit Zeitplanung in Stunden(Std.)	
1.Planung	18 Std.
2.Entwicklung	27 Std.
3. Testing	16 Std.
4. Refactoring	4 Std.
5. Soll-Ist-Vergleich	2 Std.
6. Dokumentation	13 Std.
Gesamtsumme der Stunden	80 Std.

Im Anhang unter Punkt 1 befindet sich eine feinere Gliederung.

Kosten-/Nutzen-Analyse

Im Gespräch mit dem Auftraggeber wurden die Ziele noch einmal klar definiert und welche Ziele konkret erreicht werden sollen. Die geplante Erweiterung dient in erster Linie der Einhaltung gesetzlicher Vorgaben und wird primär anhand ihres Nutzens bewertet. Da keine direkten Einnahmen generiert werden, sondern potenzielle Strafzahlungen und rechtliche Probleme vermieden werden sollen, lässt sich keine konkrete Rechnung zur Einsparung erstellen. Einsparungen können lediglich geschätzt werden und basieren nicht auf belastbaren, handfesten Daten.

- **Vermeidung von Strafen**: Die Erweiterung reduziert das Risiko rechtlicher Konsequenzen, insbesondere in Zusammenhang mit der Kurzarbeit.
- **Verbesserung der Arbeitszeitkontrolle**: Überstunden können besser überwacht und unnötige Mehrarbeit verhindert werden.

Eine genaue Quantifizierung der Einsparungen ist nicht möglich. Der Nutzen der Erweiterung liegt jedoch klar in der Risikominderung und der verbesserten Effizienz in der Arbeitszeitverwaltung. Weswegen eine beispielhafte Risikorechnung aufgeführt wird.

Risikoberechnung

Die Risikoberechnung erfolgt beispielhaft anhand von 25 Mitarbeitern da hier leitende Angestellte nicht mit gezählt werden.

Betrag des Kurzarbeitergeldes	
60,00 %	Vom Netto
2.750,00 €	Netto pro Mitarbeiter
2.750,00 € * 25 * 0,6 Mitarbeiter	
Summe = 41.250,00 €	
Wahrscheinlichkeit 10 %	
Risikosummer 4.125,00 €	
Stundensatz Entwickler = 60,00 € * 80 Stunden	
Kosten = 4.800,00 €	
Neue Wahrscheinlichkeit 5 %	
Neue Risikosumme 2.062,50 €	
Amortisierungszeit ab mehr als 2,3 Monaten	

Bußgeld bei Missachtung von Arbeitszeitgrenze	
80,00 €	1 Stunde
100,00 €	Je weitere halbe Stunde
ca. 10 Stunden = (9 * 2) * 100 + 80 * 25 Mitarbeiter	
Summe = 47.000,00 €	
Wahrscheinlichkeit 15 %	
Risikosumme 7.050,00 €	
Stundensatz Entwickler = 60,00 € * 80 Stunden	
Kosten = 4.800,00 €	
Neue Wahrscheinlichkeit 2 %	
Neue Risikosumme 940,00 €	
Amortisierungszeit gerundet ab mehr als 0,8 Monaten	

In der Rechnung werden die Entwicklerkosten von 4.800,00€ der Differenz der Risikosumme gegenübergestellt um somit den Zeitraum zu bestimmen ab wann eine Amortisierung möglich ist.
Die Fälle werden in dieser Risikoberechnung einzeln für sich geprüft und nicht vermischt.
Im Fall des Kurzarbeitergeldes ist eine Amortisierung nach 2,3 Monaten erfolgt. Während in der Beispielrechnung für die Bußgelder bereits eine Amortisierung nach weniger als einen Monat erfolgt.

SMART-Ziele

Mittels der Smart Methode wurde überprüft ob das Projekt diese Anforderungen erfüllt. Oder ob es ggf. Schwachstellen aufweist und vielleicht noch einmal genauer definiert werden muss.

(S)pezifisch	Die Einführung des Projektes zielt darauf ab, Arbeitszeitbeschränkungen sowie gesetzliche Vorgaben einzuhalten. Dadurch sollen potenzielle Konflikte mit rechtlichen Regelungen und anderen Vorschriften vermieden werden.
(M)essbar	Am Jahresende erfolgt eine Auswertung der Arbeitszeitverstöße. Feedback der Mitarbeitenden wird als weiter Messwert herangezogen
(A)kzeptiert	Das Projekt unterstützt die Einhaltung des Arbeitszeitgesetzes und reduziert das Risiko rechtlicher Konsequenzen. Um die Akzeptanz der Nutzer zu steigern ist der Quellcode für alle Nutzer einsehbar. Für maximale Transparenz können die Mitarbeiter auch ggf. fragen stellen.
(R)ealistisch	Das Ziel der Arbeitszeit Darstellung und bereitstellen von Logout Funktionen ist durchaus ein Ziel das sich umsetzten lässt.
(T)erminiert	Für die Implementierung des Moduls gibt es derzeit keinen festen Termin.

Durch diese Analyse ließ sich feststellen das es lediglich ein Problem mit Terminierung gab. Weswegen vereinbart wurde im Zeitraum der Entwicklung mehrere kleine Flash-Lights über den aktuellen Stand zu geben. Was sich ohne größere Probleme abbilden lässt, da diese Proaktiv während der Entwicklung je nach Zeitplanung und Fortschritt geplant werden können .

SWOT Analyse

Da eine präzise Kostenanalyse für dieses Projekt nicht aufgestellt werden kann, ist es wichtig, auch andere Aspekte zu berücksichtigen, um die Vor- und Nachteile des Projekts besser zu definieren.

Interne Analyse

Positive Aspekte für das Unternehmen:

- **Effektive Arbeitszeitverfolgung**: Mitarbeitende können ihre Arbeitszeiten effizient überwachen und verwalten.
- **Verhinderung unerlaubter Überstunden**: Das Modul hilft, Überstunden zu vermeiden, wenn diese nicht zulässig sind.
- **Erhöhung der Sicherheit**: Durch einen automatischen Logout wird die Systemsicherheit erhöht.
- **Erweiterbarkeit**: Das Modul kann bei Bedarf erweitert werden, um weitere Anforderungen zu erfüllen.

Mögliche negative Aspekte:

- **Reiner Kostenfaktor**: Da das Modul derzeit intern geplant ist, handelt es sich zunächst um eine Investition ohne direkten Umsatz.
- **Akzeptanzprobleme**: Es besteht die Möglichkeit, dass die Akzeptanz der Erweiterung bei den Mitarbeitenden geringer ausfällt als erwartet.

Externe Analyse

Die externe Analyse geht davon aus, dass das Modul zukünftig auch im Verkauf verfügbar sein wird.

Chancen:

- **Kundennutzen durch Übersichtlichkeit**: Kunden könnten die klare und einfache Anzeige der Arbeitszeiten zu schätzen wissen.
- **Skalierbarkeit für spezielle Anwendungsfälle**: Insbesondere für Lkw-Fahrer, die häufig unterwegs sind, bietet der automatische Logout einen praktischen Nutzen.
- **Erleichterung für die HR-Abteilung**: Weniger unerlaubte Überstunden führen zu einer Entlastung der Personalabteilung.

Risiken:

- **Bedenken bezüglich der Überwachung**: Mitarbeitende könnten sich durch das Modul überwacht fühlen, was zu Unzufriedenheit führen könnte.
- **Anpassungsaufwand für Kunden**: Das Modul erfordert möglicherweise umfangreiche Anpassungen, um den unterschiedlichen Bedürfnissen der Kunden gerecht zu werden.

Durchführung

Visualisierung des Vorgangs

Zur Visualisierung des Vorganges wurde ein Aktivitätsdiagramm (Anhang Punkt 2 und 3). Es beschreibt den geplanten Vorgang in einem beispielhaften Durchlauf. Für die Verarbeitung der Zeit wird zuerst geprüft ob eine Anmeldung vorliegt, sollte es keine Anmeldung vorliegen findet auch keine Berechnung statt. Danach soll eine Berechnung angestoßen werden die zuerst prüft ob es heute bereits andere Anmeldungen gab, dies wird mittels der Logout Daten ermittelt. Im Falle mehrerer Daten erfolgt eine weitere Prüfung ob die Daten von heute stammen. Stammen die Daten von heute werden sie zusammen addiert. Sollte ein Logout von heute sein aber der Login von gestern sollen die Zeiten vor 0 Uhr ignoriert werden. Da diese für die temporäre Berechnung nicht weiter relevant sind, hierbei würde die Berechnung dann von 0 Uhr bis Logout erfolgen. Die Daten bleiben unverändert vorhanden für die Zeitabrechnung an anderer Stelle. Danach werden die Werte für alle Daten, die die Kriterien erfüllen in einen Zwischenspeicher gesichert. Oder der Zwischenspeicher bleibt bei 0 sollte es keinen Treffer geben. Der Zwischenspeicher wird mit der aktuellen Berechnung, welche sich auf die aktuelle Anmeldung stützt, addiert und gelangt so zur Anzeige.

Hier soll nun im Hintergrund eine Prüfung erfolgen ob eine der folgenden Bedingungen, Reguläre Arbeitszeit-Logout oder Maximale Arbeitszeit-Logout, aktiv ist. Falls eine dieser Bedingungen aktiv ist ist, wird geprüft ob das Gesamtergebnis der Anzeige Zeit diesen Wert übersteigt. Sollte dies der Fall sein wird hier ein automatischer Logout ausgelöst.

Improvement der Darstellung der Zeiterfassung im Odoo Framework

Odoo Entwicklungsumgebung aufsetzen

Als Entwicklungsumgebung wird Odoo 11 genutzt, wobei intern vorbereitete Skripte für die verschiedenen Versionsstände beim erstellen des Grundgerüstes helfen. Der Grund warum sich für Odoo entschieden wurde, ist das Odoo die Grundlage für den Webservice von Sirum darstellt und dort später genutzt werden soll. Auch wenn Sirum intern auf einem der neueren Stände von Odoo arbeitet, steht ebenso der Gedanke im Raum es später ggf. für Kunden bereit stellen zu wollen. Dieses stand einer Entscheidung für die aktuelle Odoo Version die Sirum nutzt im Wege. Da Sirum Kunden im Spektrum von Odoo 11 bis Odoo 15 hat zur Zeit. Weswegen sich für die Version 11 entschieden wurde um hier eine Kompatibilität für alle nachfolge Versionen zu generieren. Der Aufbau der Entwicklungsumgebung erfolgte auf einem Linux-System (Linux Mint Cinnamon), das im Unternehmen von allen genutzt wird.

1. **Erstellung des Docker-Systems**:
   Mit dem folgenden Befehl wurde ein Docker-System aufgesetzt das die Grundlagen von Odoo beinhaltet:

   docker-compose run odoo -d devel

   Dieser Befehl sorgt dafür, dass Docker Odoo aufbaut und über die Konfigurationsdatei devel.yaml die benötigten Grundservices bereitstellt. Dadurch wird das Grundgerüst von Odoo konfiguriert und eine Verzeichnisstruktur mit den notwendigsten Grundfunktionen für Odoo 11 erstellt.

Installierte Module

Die Entwicklungsumgebung enthält die folgenden Module, die bereits vorinstalliert sind:

- **Base** (Kernel von Odoo)
- **Attendance**
- **Signup**
- **Initial Setup Tools**
- **Employee Directory**
- **Resource**
- **Web**

Diese sieben Module sind Teil der 25 Standardmodule, die in jeder Entwicklungsumgebung mit implementiert werden. Weitere Module wie **Discuss**, ein internes Chat-Programm, oder **Mail Create Activity Wizard**, ein früheres Projekt, das eine einfache Aufgabenverteilung ermöglichte, sind ebenfalls enthalten.

Planung der Darstellung

Zunächst wurde das **Attendance-Modul**, das erweitert werden soll, genauer untersucht, um eine geeignete Stelle für die Anzeige der Arbeitszeit und weiterer Optionen zu finden.

Im ersten Konzept wurde entschieden, die Zeitansicht direkt im **Attendance-Login** zu integrieren, sodass die Mitarbeitenden beim Login sofort sehen können, wie viel Zeit bereits gearbeitet wurde. Die **Logout-Funktion** sollte hingegen separat platziert werden. Hierfür wurde die **Employee-Darstellung** als geeigneter Ort gewählt, um den Logout-Button zu integrieren.

Erstellen des Verzeichnes:
Für eine korrekte Funktionalität muss die Erweiterung in das für Odoo typische Modulschema integriert werden. Zunächst wird ein Ordner im entsprechenden Verzeichnis angelegt, der den Modulnamen präzise beschreibt. Dieser Name sollte sowohl den Bereich als auch die Funktion des Moduls klar definieren. In diesem Fall wurde der Ordnername auf **hr_attendance_compute_time** festgelegt.

Die Struktur sieht hierbei beispielhaft wie in Schaubild 1 dargestellt aus.

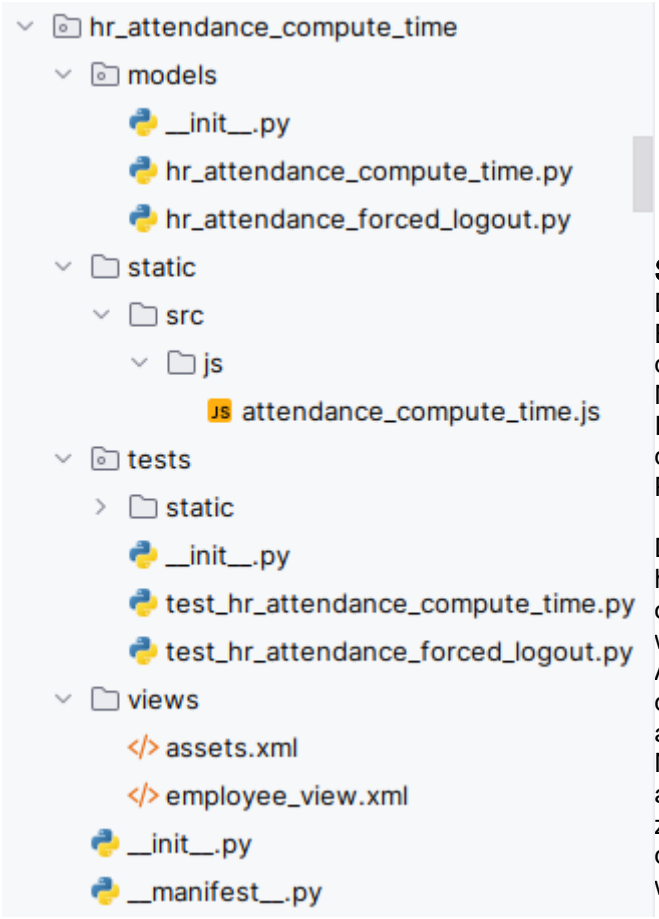


Schaubild 1: Ordner Struktur

- **hr** verweist auf den Bereich „Human Resources“,
- **attendance** beschreibt den Bezug zum Attendance-Modul,
- **compute_time** zeigt an, dass es sich um die Berechnung der Arbeitszeit handelt.

Schreiben des Manifests:

Das Manifest spielt eine zentrale Rolle bei der Entwicklung eines Odoo-Moduls, da ohne dieses kein Modul installiert werden kann. Im Manifest werden mehrere wichtige Informationen und Einstellungen für das Modul definiert. Diese sind näher im Anhang unter Punkt 4 beschrieben.

Die __init__.py-Dateien sind einfach zu handhaben und verweisen lediglich auf Ordner oder Python-Dateien innerhalb eines Ordners, wie z.B. Models. Sie regeln die Aufrufhierarchie des Moduls. Es ist wichtig, auf die richtige Reihenfolge beim Schreiben zu achten, da das, was zuerst in den Init- oder Manifest-Dateien aufgeführt ist, auch zuerst ausgeführt wird. Falls Abhängigkeiten zwischen Codebestandteilen bestehen, muss die Reihenfolge der Aufrufe korrekt eingehalten werden, um Fehler zu vermeiden.

Erstellen der Funktionen und Klassen:

Zunächst wird eine Klasse **hr.employee** erstellt. Dabei wurde sich entschieden, die Klasse von **hr.employee** zu vererben, da viele der benötigten Werte bereits in diesem Core-Modul gespeichert sind und somit wiederverwendet werden können.

Im nächsten Schritt werden innerhalb der Klasse einige Variablen definiert. Dank der Flexibilität von Odoo werden diese Variablen direkt als Felder in der Datenbank deklariert. So können sie automatisch in der Datenbank gespeichert und verwaltet werden.

Falls erforderlich, werden für bestimmte Felder auch Standardwerte gesetzt. Dies kann beispielsweise wie in Schaubild 2 gezeigt aussehen:

```
class HrAttendance(models.Model):
    _inherit = "hr.attendance"

    attendance_ids = fields.One2many(
        comodel_name: "hr.attendance", inverse_name: "employee_id", help="list of the attendances of the employee"
    )
```

Schaubild 2: Datenbank Erweiterung

Zunächst wird ein Feld vorbereitet, das alle Anmeldedaten in Bezug auf die Mitarbeiter-ID sammelt. Dadurch wird eine einfache Zuordnung ermöglicht, die später zur Berechnung der Arbeitszeiten basierend auf den Anmelde- und Abmeldedaten dient.

Eine wichtige technische Grundlage bietet Odoo, das die Integration von APIs in Funktionen unterstützt. Dies geschieht mithilfe des @api.depends(args)-Dekorators, der vor einer Funktion platziert wird. Der Dekorator definiert die notwendigen Abhängigkeiten und ermöglicht so den Zugriff auf Daten aus anderen Modulen, auch wenn diese nicht direkt vererbt wurden.

Die zentrale Funktion des Moduls berechnet die aktuelle Arbeitszeit auf Basis von Check-in- und Check-out-Daten. Dabei wird zunächst der Zeitunterschied zwischen der User-Zeit und der Server-Zeit ausgeglichen, um Berechnungsfehler zu vermeiden. Solche Fehler könnten auftreten, wenn die Zeitzonen des Users und des Servers voneinander abweichen. Ohne eine Anpassung der Zeitzonen enstehen massive Ungenauigkeiten in der Arbeitszeitberechnung entstehen.

Ein weiterer essenzieller Schritt ist die Sicherstellung, dass nur der aktuelle Tag für die Berechnung berücksichtigt wird. Dies wird durch den Einsatz von Lambda-Funktionen gewährleistet, die das heutige Datum überprüfen. Das Datum des aktuellen Tages wird in einer Variablen gespeichert, und ein künstlicher Zeitwert wird erstellt, der den jeweiligen Tag mit dem Zeitstempel 00:00:00 repräsentiert.

Falls der Check-in vor dem heutigen Datum liegt, kommen verschiedene Abfragen durch Lambda-Funktionen zum Einsatz. Diese prüfen zunächst, ob es am aktuellen Tag eine Anmeldung gab oder ob lediglich ein Logout am heutigen Tag stattgefunden hat. Die entsprechenden Anmeldedaten werden dann in einer Variablen gespeichert. Obwohl es Fälle geben kann, in denen Arbeitszeiten über Mitternacht hinausgehen, wird dennoch jeder Tag für sich betrachtet, um die Berechnungen konsistent und nachvollziehbar zu gestalten.

Berechnung der Arbeitszeit

Sobald die relevanten Daten identifiziert sind, wird die **Arbeitszeit** berechnet. Die Schritte beinhalten:

1. **Abgleich der Anmeldungen und Abmeldungen**: Hier wird geprüft, ob aktuell eine Anmeldung vorliegt, um die Zeit zwischen Anmeldung und der aktuellen Zeit zu berechnen.
2. **Korrektur von Zeitwerten**: Für Anmeldungen, die am Vortag stattfanden, wird der Login-Wert durch den künstlich erzeugten Wert (00:00:00) ersetzt, um sicherzustellen, dass immer nur der jeweilige Tag berücksichtigt wird.

3. **Speicherung der Ergebnisse**: Alle berechneten Daten werden in Variablen gespeichert, die später weiterverarbeitet werden können.

Überprüfung der Arbeitszeiten

Die erfassten Zeiten müssen anschließend überprüft werden. Zunächst wird festgelegt, was passieren soll, wenn bestimmte Bedingungen erfüllt sind. Dazu gehört:

- **Erzwingen des Logouts**: Eine Funktion wird geladen, die den **Check-out** erzwingt, wenn bestimmte Schwellenwerte überschritten sind. Diese werden vorher händisch definiert von einem entsprechenden HR Mitarbeiter. Damit hier eine Skalierbarkeit besteht war es nötig den Mitarbeitern diese Möglichkeit an die Hand zu geben.
- **User-Alert**: Ein **Sticky-Note**-Alert wird für den Benutzer ausgelöst, um ihn darüber zu informieren, dass er ausgeloggt wurde. Diese wird innerhalb der vorgefertigten Funktion des Logouts erzeugt und mittels einer zuvor erzeugten Nachricht befüllt.

Beides wird durch eine Nested Funktion gewährleistet die innerhalb der Funktion zur Berechnung der Arbeitszeit steckt. Es wurde sich hierfür entschieden da die Funktion lediglich innerhalb der Berechnung benötigt wird.

Nun beginnt die Prüfung der Bedingungen:

1. **Überprüfung der offenen Anmeldung**: Wenn keine offene Anmeldung vorhanden ist, werden die restlichen Abfragen übersprungen und direkt mit der Berechnung der aktuellen Arbeitszeit fortgefahren.

2. **Arbeitszeitgrenzen**: Falls eine Anmeldung vorhanden ist, müssen bestimmte Abfragen auf **False** auslaufen, damit der Mitarbeiter weiterhin angemeldet bleibt.
   - Zunächst wird geprüft, ob die **Standard-Arbeitszeit** von 8 Stunden überschritten wird (Checkbox für Standard-Arbeitszeit aktiv). Wenn diese aktiv ist, wird die Arbeitszeit mit dem festgelegten Limit verglichen und ein Logout wird ggf. ausgelöst.
   - Zusätzlich kann ein **prozentualer Puffer** für die Arbeitszeit hinzugefügt werden, um kleine Überschreitungen zu erlauben.

3. **Maximale Arbeitszeit**: Eine ähnliche Prüfung erfolgt für die maximale Arbeitszeit von 10 Stunden pro Tag. Auch hier kann ein Puffer hinzugefügt werden, allerdings müssen rechtliche Vorgaben beachtet werden. So das keine Verletzungen des Arbeitszeitgesetz erfolgen wonach die Regelarbeitszeit 8 Stunden beträgt und die zulässige Überstunden Zahl von 10 Stunden Gesamtzeit nicht überschritten werden sollen. Dies hat jedoch der zuständige HR-Mitarbeiter zu überwachen.

4. **HR-Funktionalitäten**: Während der Erprobung wurde entschieden, diese Checkboxen in den Bereich zu verschieben, der nur für **HR-Mitarbeiter** zugänglich ist. Das Tool dient dazu, eine einfache und effiziente Verwaltung von Arbeitszeiten zu ermöglichen, ohne viel organisatorischen Aufwand.

Berechnung der aktuellen Arbeitszeit

Am Ende der Überprüfung wird die **aktuelle Arbeitszeit** berechnet. Diese wird jedoch nur ausgegeben, wenn der Mitarbeiter tatsächlich angemeldet ist. Sollte keine Anmeldung vorliegen, wird keine Arbeitszeit berechnet.

Um die Ausgabe der Arbeitszeit in einem lesbaren und nutzbaren Format darzustellen, wurde das ursprüngliche **Time-Format** so angepasst, dass es in einem **String** ausgegeben werden kann, der Stunden und Minuten korrekt trennt.

1. **Stundenformatierung**: Zunächst wurden die Stunden des Datensatzes extrahiert und in einen **Integer** umgewandelt. Die Minuten, die im ursprünglichen Format als Nachkommastellen angezeigt wurden, fallen dabei weg, da sie keinen weiteren Einfluss auf die Stundenangabe haben.
2. **Minutenberechnung**: Weil Stunden und Minuten getrennt voneinander angezeigt werden sollen, muss hier eine Änderung der Berechnung erfolgen. Dies führte zu einem anfänglichen Problem und es hat gedauert bis eine entsprechende Formel zur genauen Berechnung der Zeit vorlag.

3. **String-Formatierung**: Nachdem Stunden und Minuten korrekt berechnet wurden, werden diese Werte in einem **vorbereiteten String** eingefügt, der die finale Anzeige der Arbeitszeit darstellt. So wird der Benutzer eine klare Darstellung der geleisteten Arbeitszeit erhalten.

4. **Varianz der Anzeige**: Um die Arbeitszeit visuell hervorzuheben und monotone Anzeigen zu vermeiden, wurde entschieden, einen zusätzlichen Text in der Anzeige zu integrieren, der je nach der bereits absolvierten Arbeitszeit variiert. Dies sorgt dafür, dass Veränderungen in der Arbeitszeit sofort ins Auge fallen, anstatt eine statische Anzeige zu haben, die keine dynamische Rückmeldung gibt. Ein Beispiel zu dieser Anzeige ist in Schaubild 3 mittels Auszug aus dem Quellcode aufgezeigt.

```
elif record.actual_time > self.normal_work_time_per_day:
    record.time_message = _("You're working overtime now")
```

Schaubild 3: Beispiel Ausgabe

Im ersten Konzept wurde darauf verzichtet, Hinweise oder Notizen zu Pausenzeiten anzuzeigen. Diese wären technisch möglich, werden jedoch zum aktuellen Zeitpunkt nicht als zielführend angesehen. Sie könnten in einer späteren Erweiterung integriert werden, um die Einhaltung von Pausenregelungen zu unterstützen und zusätzliche Funktionen zu bieten.

Ein weiteres zentrales Feature des Moduls ist der automatische **Checkout**. Dies wird durch eine separate Klasse ermöglicht, die das **HR Attendance-Modul** erweitert. Diese Klasse übernimmt die Rolle, automatisch den **Check-out** von Mitarbeitern zu realisieren, wenn bestimmte Bedingungen erfüllt sind.

1. **Automatisches Ausloggen**: Die Funktion zum automatischen Ausloggen prüft, ob der Mitarbeiter in der **Attendance-Liste** korrekt eingetragen ist und ob ein **Check-out** fehlt. Falls beide Bedingungen zutreffen, wird der aktuelle **Zeitstempel** als **Checkout-Zeit** für den Mitarbeiter eingetragen.

2. **Inaktivitätsprüfung**: Eine weitere Funktion zur Überprüfung der **Inaktivität** wurde ebenfalls implementiert. Weil das System keine standardisierten Werte für die Inaktivität von Nutzern bereitstellt, musste eine eigene Funktion entwickelt werden, die aktiv die nötigen Daten abruft und prüft, ob der Mitarbeiter längere Zeit inaktiv war. Falls eine bestimmte Inaktivitätsdauer überschritten wird, erfolgt ebenfalls ein automatischer Checkout.

Die Kombination dieser beiden Funktionen sorgt dafür, dass die Arbeitszeit der Mitarbeiter korrekt erfasst wird und auch bei inaktiven oder vergessenen Logouts automatisch ein **Check-out** vorgenommen wird, um die Arbeitszeit richtig zu erfassen.

Platzieren und Gestalten der Ansicht

Ursprünglich war geplant, die Anzeige der Arbeitszeit direkt unterhalb des Logout-Buttons innerhalb des Attendance-Moduls zu platzieren (Anhang Punkt 5). Diese Platzierung brachte jedoch einige unerwartete Schwierigkeiten mit sich, insbesondere im Hinblick auf die Modellierung der Ansicht. Um die Anzeige der Arbeitszeit korrekt zu integrieren, mussten mehrere Modulansichten angepasst werden, was zu Konflikten mit den Coding Guidelines von Sirum führte. Laut diesen Guidelines sollte jeder Abschnitt des Codes weitestgehend autark und für sich selbstständig sein, was durch die nötigen Änderungen an anderen Modulen beeinträchtigt werden würde. Deshalb wurde die anfängliche Platzierungsidee schnell wieder verworfen.
Stattdessen wurde entschieden, die Anzeige der Arbeitszeit an einer zentraleren Stelle in der Weboberfläche unterzubringen (Anhang Punkt 6). Diese Entscheidung brachte folgende Vorteile mit sich:
Zum einen mussten keine massiven Änderungen an bestehenden Modulen vorgenommen werden, da die Platzierung nun außerhalb des Attendance-Moduls und damit flexibler gehandhabt werden konnte. Und die Arbeitszeit ist nun immer sichtbar, ohne dass Mitarbeiter in ein anderes Modul springen müssen, um diese einzusehen.

Die Arbeitszeit wird nun in der Topbar der Weboberfläche angezeigt. Dies ist eine prominente und leicht zugängliche Position, die den Mitarbeitern einen ständigen Überblick ermöglicht. Über die XML-Dateien wurde die Platzierung der Checkboxen gesteuert. Diese Checkboxen sind erforderlich, um unterschiedliche Funktionen wie das Setzen von Standardarbeitszeiten oder maximalen Arbeitszeiten zu aktivieren. Zusätzlich wurde der JavaScript-Aufruf in den XML-Dateien integriert. Dieser sorgt dafür, dass die benötigten Daten dynamisch geladen und die Arbeitszeit korrekt angezeigt wird.

```
<?xml version="1.0" encoding="utf-8" ?>
<odoo>
        <template id="assets_backend" name="hr_attendance_compute_time_assets" inherit_id="web.assets_backend">
            <xpath expr="." position="inside">
                <script
            type="text/javascript"
            src="/hr_attendance_compute_time/static/src/js/attendance_compute_time.js"
          />
            </xpath>
        </template>
</odoo>
```

Schaubild 4: XML Aufruf

Die Entscheidung, die Zeit in der Topbar darzustellen, führt zu einer praktischen, einfachen und effektiven Lösung, die sowohl den Anforderungen der Benutzerfreundlichkeit als auch den technischen Richtlinien entspricht. Der Aufruf innerhalb der XML Datei kann z.b. wie in Schaubild 4 gezeigt aussehen.

Probleme mit der neuen Darstellung:

Da die Darstellung nun auf die Weboberfläche ausgelagert wurde, trat ein neues Problem auf. Zuvor wurde bei jedem Aufruf der Funktion die Berechnung in Echtzeit durchgeführt. Nun jedoch erfolgt die Berechnung nur noch beim ersten Aufruf der Seite, was zu einer Einschränkung in der Funktionalität führte. Um dieses Problem zu lösen, wurde die Idee eines automatischen Refreshs eingebracht. Anfangs war eine enge Taktung für das Neuladen der Seite geplant, doch dies erwies sich als störend. Bei jedem neuen Aufruf erschien ein „Loading“-Pop-up, das sich relativ mittig auf der oberen Seite befand, was schnell als unangenehm empfunden wurde. Daher wurde entschieden, auf ein zu häufiges Neuladen der Seite zu verzichten. Stattdessen wurde eine Zeitspanne von 50 Sekunden als akzeptabel festgelegt, um die Seite neu zu laden.
Die Neuberechnung muss mindestens alle 60 Sekunden erfolgen, um sicherzustellen, dass ein Logout rechtzeitig erkannt wird. 60 Sekunden wurden jedoch als zu lang angesehen, weshalb sich für 50 Sekunden entschieden wurde.
Hierbei erscheint das „Loading“ zwar weiterhin sollte aber als weniger störend oder verwirrend wahrgenommen werden als eine Anzeige alle paar Sekunden.

Ebenso kam es zu Problemen mit der konkreten Anzeige selber (Anhang 7) ,weil das gewünschte Element vor dem es platziert werden sollte nicht einzigartig benannt wurde, der Name war doppelt vergeben. Was einen doppelten Aufruf zufolge hatte und unerwartete Fehler verursachte. Auch wenn dieses Problem gelöst werden konnte. Sollte das Problem mit der doppelten Namensgebung in der Weboberfläche,möglicherweise in späteren Reviews noch einmal genauer betrachtet werden. Da es so sonst immer wieder bei neuen Projekten zu Fehlern führen kann.

Qualitätssicherung:
Zur Sicherung der Qualität des Codes gibt es mehrere Richtlinien, die während des gesamten Entwicklungsprozesses eingehalten werden müssen. Zu diesem zählen unter anderem Coding-Guidelines, Linting-Guidelines, Namensvorgaben für die Module und Tests welche sowohl in automatisierter Form (Test-Pipelines und Unittests) aber auch in manueller Form stattfinden. Wobei die manuellen Tests an zwei Stellen stattfinden, zu einem in der Entwicklungsumgebung begleitend zum Coding, zum anderen in einer Server-Seitigen Umgebung die eine entsprechende Testumgebung simuliert. Die beim Coding manuell durchgeführten Tests werden durch ein im Framework vorhandenes Debugging-Tool unterstützt. Zur Überprüfung des Quellcodes kommen unter anderem Logger zum Einsatz, sowie das wdb Tool, das es ermöglicht, innerhalb des Programms Debugging-Punkte zu setzen. Dadurch können die einzelnen Schritte überprüft werden oder die in Variablen aktuell gespeicherte Werte, geprüft werden. So lassen sich Fehler lokalisieren und auf bestimmte Codezeilen zurückführen.

Die Anforderungen an die Qualitätssicherung beinhalten auch die Struktur und Namensgebung der Ordner. Die Namen folgen einer vorgeschriebenen Logik diese Ordner eindeutig zu beschreiben, damit diese schnell ihrer Kategorie und nutzen verordnet werden können.
In den Dokumentationen für diese Ordner wird genau beschrieben, welche Dateien wo abgelegt werden müssen und welche Namensregeln dafür gelten. Die Namen der Dateien sollten immer sprechend und an den Moduldaten orientiert sein.

Schreiben von Unittests:

Zum Erstellen der Unittest-Datei muss zunächst die entsprechende Bibliothek importiert werden. Dann wird eine Testklasse als Umgebung erzeugt, in der mittels einer Funktion setUp die Rahmenbedingungen definiert werden. In diesem Fall wird ein künstlicher Mitarbeiter erzeugt, der später für die Testfälle verwendet wird. Mittels einer weiteren Funktion werden künstliche Logindaten erzeugt, bei denen vorab die Mitarbeiter-ID sowie Login- und Logout-Zeiten übermittelt werden, um die Ergebnisse gezielt steuern zu können. Da in der Testumgebung auf die tatsächlichen Funktionen zugegriffen wird, können so verschiedene Szenarien getestet werden.

```
10  class TestHrEmployee(TransactionCase):
        jona
11      def setUp(self):
12          super(TestHrEmployee, self).setUp()
13          self.employee = self.env["hr.employee"].create(
14              {
15                  "name": "John Doe",
16                  "forced_checkout_normal_work_hour_check_box": False,
17                  "forced_checkout_max_work_hour_check_box": False,
18                  "max_work_time_per_day": 10,
19                  "normal_work_time_per_day": 8,
20              }
21          )
22          self.attendance_model = self.env["hr.attendance"]
```

Schaubild 5: Initialisierung von Unittests

Im Anschluss werden die ersten Testfälle formuliert, um alle Optionen des Codes abzufragen. Zuerst erfolgt eine Abfrage, bei der keinerlei Anmeldung vorliegt. In diesem Fall wird ein Wert in Sekunden zurückerwartet, sowie eine Nachricht für den Fall, dass es zu einem Fehler kommt. Danach folgt ein Test mit einer einfachen Anmeldung von heute, bei dem der erwartete Wert für die Berechnung überprüft wird. Ein weiterer Test behandelt die Situation, dass eine Anmeldung von heute und eine vom Vortag vorliegen, die sich in den heutigen Tag hinein erstreckt. Danach wird die Funktion zur Berechnung des Zeitstrings überprüft. Als letzter Test wird die Ausgabe der Nachrichten kontrolliert, sowohl wenn keine Anmeldung vorliegt, als auch wenn eine Anmeldung mit einer bestimmten Arbeitszeit vorhanden ist.
Der Grund, warum bei den forced_checkouts beide Werte auf False gesetzt wurden (siehe Schaubild 5) , liegt darin, dass diese Funktionen in einem separaten Test geprüft werden. Dieser Test behandelt vorrangig die Funktionalität der Berechnungsfunktionen. Anfangs kam es jedoch zu größeren Abweichungen, da die Serverzeit und die Userzeit nicht synchron waren. Dies führte dazu, dass in den Testfällen die Ergebnisse verschoben wurden. Die Ursache dafür wurde erkannt, was eine Nachbesserung im Code zur Handhabung der Zeiten erforderlich machte.
Der Aufbau der nächsten Testdatei erfolgt ähnlich wie bei der ersten Datei. In der setUp-Funktion muss diesmal jedoch ein User erzeugt werden. Der übliche Weg über den Environment-User hat in diesem Fall nicht funktioniert und Herausforderungen geschaffen, weshalb ein neues Objekt der Klasse hr_employee erstellt werden musst, um erfolgreiche Tests durchzuführen. Die forced-Objekte werden hier nicht in der setUp-Funktion definiert, da sich diese gegenseitig ausschließen. Deshalb müssen die beiden Bedingungen in den Testfällen angepasst werden. Der erste Test prüft, ob die Funktion „Force Checkout“ funktioniert. Hierfür wird ein Login von vor drei Stunden ohne Logout gesetzt, während die normalen Arbeitsstunden auf zwei Stunden eingestellt werden und die entsprechende Checkbox aktiviert wird. In diesem Fall wird ein Logout erwartet.

Soll-Ist-Vergleich

Sämtliche Soll-Kriterien wurden erfüllt. Es wurde ein funktionierendes Modul entwickelt, das die Arbeitszeit automatisiert berechnen kann. Zudem bietet das Modul die Möglichkeit, durch Checkboxen, die unter Berücksichtigung der Zugriffsrechte platziert wurden, einen automatischen

Logout durchzuführen. Darüber hinaus wurden die Coding Guidelines eingehalten, was eine schnelle Erweiterbarkeit des Moduls gewährleistet.
Während der Dokumentation fiel in den manuellen Reviews jedoch auf, dass bei den Unittests nicht alle Anwendungsfälle berücksichtigt wurden.

Projektphasen SOLL-/Ist-Vergleich in Stunden	
SOLL	IST
Planung	
18 Stunden	15 Stunden
Der Part der Planung lief besser als geplant und an einzelnen Stellen konnte Zeit eingespart werden	
Entwicklung/Coding	
27 Stunden	28 Stunden
Durch ungenaue Beschreibungen waren mehrere Änderungen nötig was mehr Zeit gekostet hat	
Testing	
16 Stunden	15 Stunden
Das Testing war schneller erfolgreich als geplant	
Refactoring	
4 Stunden	2 Stunden
Durch vorhandene Übung war weniger Refactoring nötig als geplant	
Soll-Ist-Vergleich	
2 Stunden	2 Stunden
Die Zeit hierfür war angemessen berechnet.	
Dokumentation	
13 Stunden	17 Stunden
Die Dokumentation hat deutlich Länger gedauert als angenommen	
Gesamtsumme SOLL= 80 Stunden	Gesamtsumme IST = 79 Stunden

Fazit

Für mich war dieses Projekt an einigen Stellen sehr herausfordernd. Die Schwierigkeiten, die sich im Projekt selbst ergaben, waren stellenweise sehr überraschend. Durch regelmäßige interne Gespräche wurden die Zwischenstände betrachtet und teilweise neue Impulse in das Projekt eingebracht.

Durch das Berücksichtigen der neuen Impulse änderte sich teilweise der Blickwinkel auf das Projekt, was an einigen Stellen Hilfreich war.

Das als Ergebnis eine fertige Version herauskam, war sehr zufriedenstellend. Das ein Testfall vergessen wurde, war persönlicher Nachlässigkeit und dem Vertrauen in Kontrollmechanismen geschuldet. Leider waren die Testpiplines während der Tests nicht erreichbar. Eine bessere Kontrolle der Testfälle hätte diese Fehler vermeiden können.

Für mich nehme ich aus dem Projekt folgenden Punkte besonders mit:
- Bessere Kontrolle der Tests
- Erkenntnisse zum Handling von Javascript im Framework
- Wechseln des Blickwinkels

Glossar

Odoo ist eine integrierte ERP-Software-Lösung mit dualen Lizenzmodell. Das Geschäftsmodell von Odoo entspricht hierbei dem Open-Core Prinzip. Odoo S.A. selber fokussiert ihre Entwicklung auf ERP,CRM,Buchhaltung,CMS und E-Commerce.

Docker ist eine Möglichkeit Software Betriebssystem unabhängig zu gestalten. Und in kleine Bausteine sogenannte Container zu zerlegen. Durch diese Option lassen sich einzelne Container beliebig zusammenstellen.

Unittest sind eine Möglichkeit Code in einer abgeschirmten Testumgebung zu prüfen und auf Funktionalität hin zu testen. Wobei die Eingabe und Ausgabewerte dem Unittest mit gegeben werden.

Coverrage oder zu Deutsch Testabdeckung beschreibt das ein Testverfahren bestimmte Anforderungen erfüllt, hier wird zwischen Funktionsabdeckung, Zweigabdeckung und Pfadabdeckung unterschieden.

Tool zu deutsch Werkzeug, beschreibt Tool Hilfsmittel die einem Software zur Erreichung der Ziele bereitstellt.

Variablen sind Behälter den ein Wert zugeordnet werden kann und damit ans Programm übergeben wird, diese können statisch definiert sein oder per Eingabe erfolgen, ggf. zieht sich das Programm die Werte auch ohne Eingabe.

Kernel ist der Zentrale Bestandteil eines Systems, in ihm werden die Prozess und Datenorganisation bestimmt, auf dem die restliche Struktur aufbaut.

Trigger ist ein bestimmter Punkt,Wert,Schalter etc. der platziert wird. Wenn die Bedingungen erfüllt werden wird eine abfolge von Schritten automatisiert ausgeführt.

User ist der Endnutzer eines Produktes.

Refactoring ist das Anpassen des Codes nach bestimmten Vorgaben wie z.b. Zeichenlänge pro Zeile, Einrückungen oder auch das Verwenden von „ oder , an bestimmten Stellen können gefordert sein.

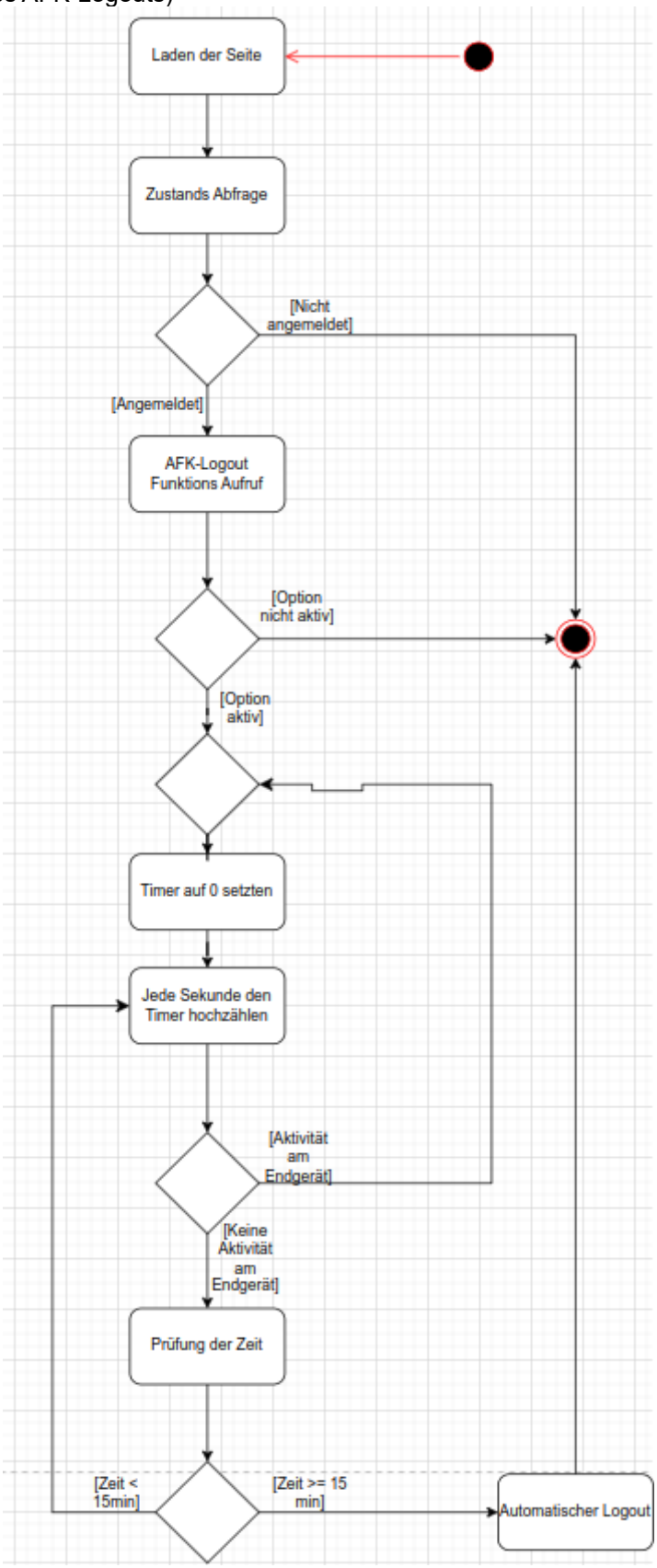
String,Integer und Boolean definieren die Werte innerhalb einer Variablen und ordnen diese so String = Text, Intger = Ganzzahlen oder Boolean = Wahr oder Falsch zu. Dies ist wichtig um Daten mit einander verarbeiten oder abgleichen zu können. Neben diesen 3 Werten gibt es noch einige andere wie Date =Datum, Float = Fließkommazahlen und noch viele mehr.

Anhang

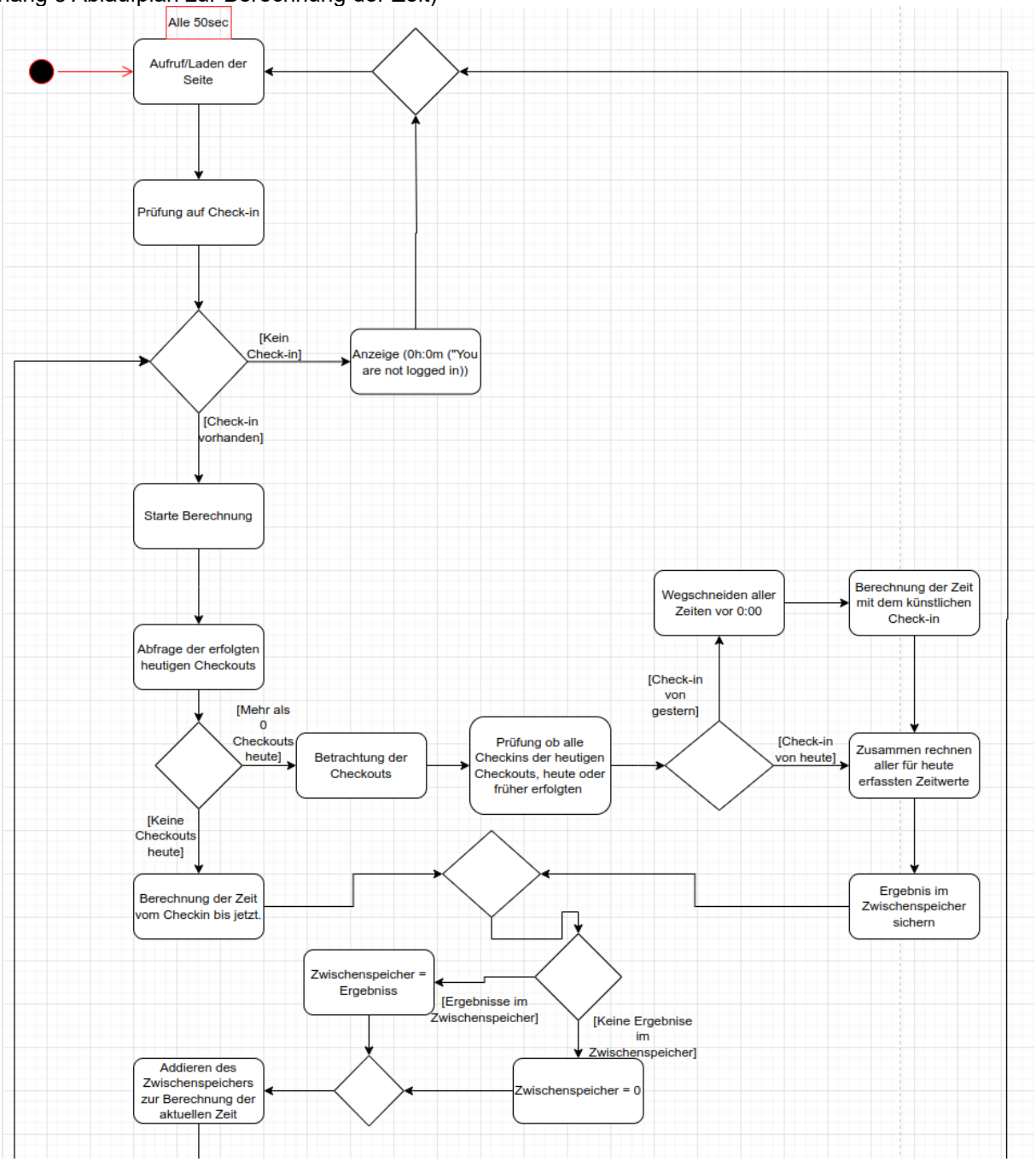
Anhang 1. Zeitplanung)

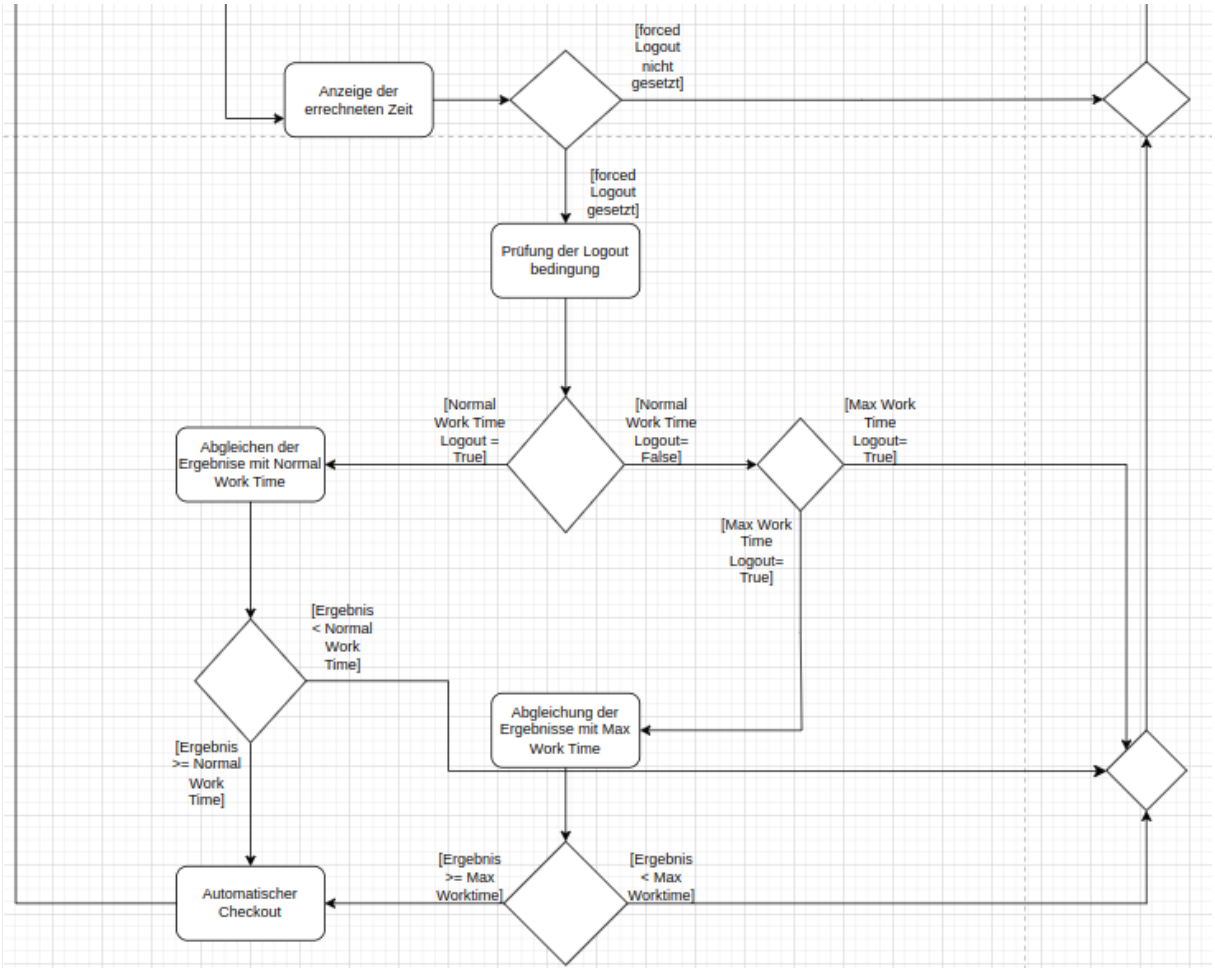
Projektphasen mit Zeitplanung in Stunden(Std.)	
1.Planung	18 Std.
1.1 Besprechung des Projektes und	
der einzelnen Phasen	
1.2 Kosten-Nutzenanalyse	
1.3 Vorgehen strukturieren	
1.4 Erstellen eines Entwurfes	
2.Entwicklung	27 Std.
2.1 Planung notwendiger Datenbank zugriffe	
2.2 Auswahl der passenden Veerbungen	
2.3 Anpassen der Oberfläche	
2.4 Integrieren der Erweiterung	
3. Testing	16 Std.
3.1 Manuelles Testing	
3.2 Unittests erstellen u. Ausführen	
4. Refactoring	4 Std.
5. Soll-Ist-Vergleich	2 Std.
6. Dokumentation	13 Std.
Gesamtsumme der Stunden	80 Std.

Anhang 2. Ablaufplan des AFK-Logouts)



Anhang 3 Ablaufplan zur Berechnung der Zeit)



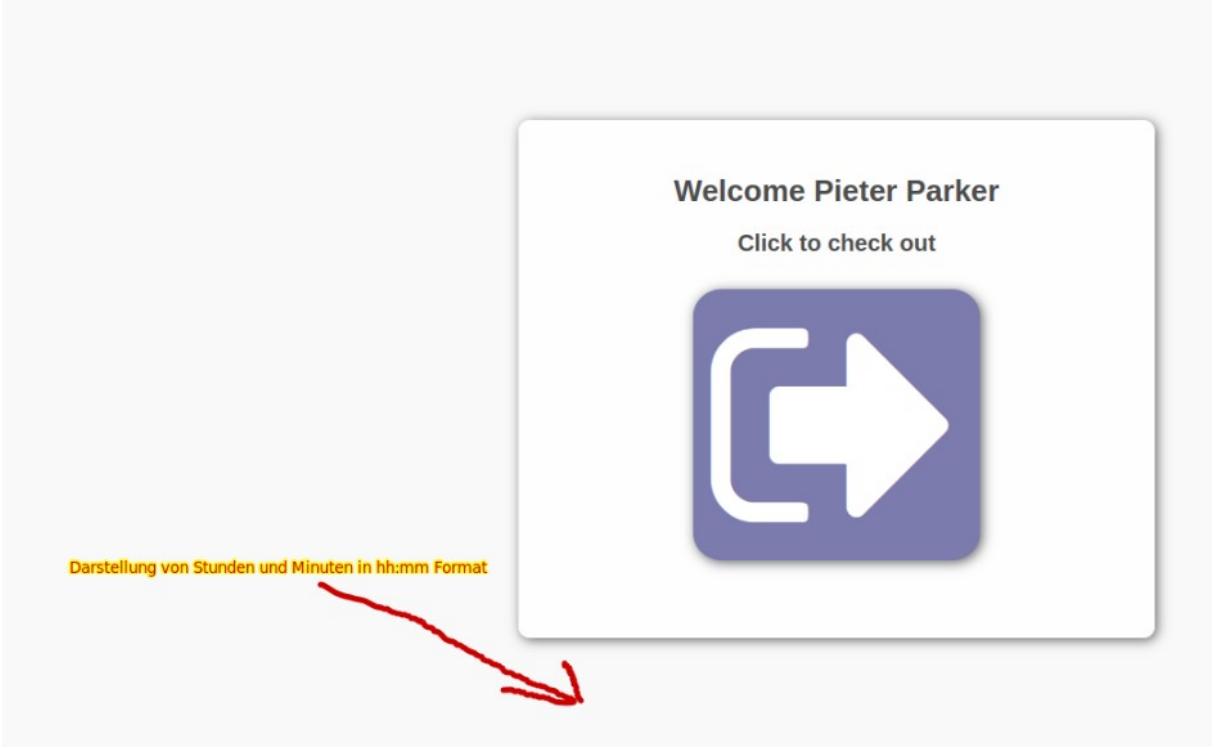


Anhang 4)

Wichtige Aspekte im Manifest:

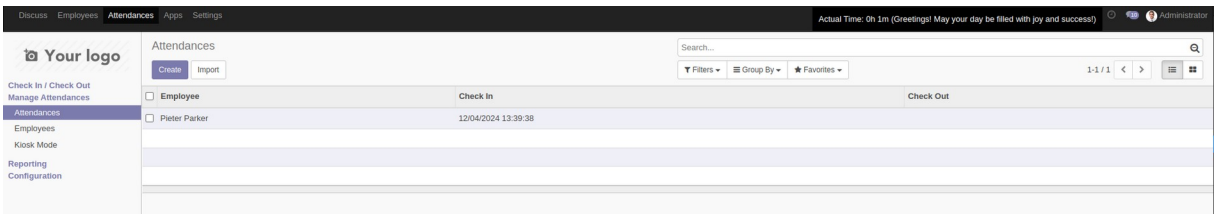
- **Name des Moduls**: Hier wird der Name des Moduls festgelegt, unter dem es später im Odoo App-Store zu finden ist.
- **Autor**: Der Entwickler oder die Firma, die das Modul erstellt hat, wird angegeben. Es kann auch eine Verlinkung zur eigenen Website hinzugefügt werden.
- **Lizenz**: Die Lizenz, unter der das Modul vertrieben wird, wird hier definiert.
- **Kategorie**: Die Kategorie, in die das Modul im Odoo App-Store einsortiert werden soll, wird angegeben.
- **Abhängigkeiten (depends)**: Hier werden die Module aufgelistet, von denen das Modul abhängt, um vollständig funktionsfähig zu sein. Diese Abhängigkeiten müssen erfüllt sein, damit das Modul korrekt funktioniert.
- **Daten (data)**: Im Manifest wird auch festgelegt, welche Views (Ansichten) geladen werden sollen. Nur die hier angegebenen Views werden letztlich im System angezeigt.
- **Boolean-Werte**: Es können Einstellungen wie installable und auto_install vorgenommen werden. Mit installable wird bestimmt, ob das Modul überhaupt installiert werden kann. Mit auto_install wird festgelegt, dass das Modul automatisch installiert wird, sobald alle Abhängigkeiten erfüllt sind.

Anhang 5)



Anhang 6)

Anhang 7)



Quellcode:

Manifest.py

```
{
    "name": "Attendances Work Time",
    "author": "Sirum GmbH",
    "website": "https://www.sirum.de",
```

```
    "license": "LGPL-3",
    "category": "Human Resources",
    "depends": ["hr", "web", "web_notify", "web_notify_extension", "hr_attendance"],
    "data": [
        "views/assets.xml",
        "views/employee_view.xml",
    ],
    "installable": True,
    "auto_install": False,
}
```

init.py

```
from . import models
```

models/__init__.py

```
from . import hr_attendance_compute_time
from . import hr_attendance_forced_logout
```

models/hr_attendance_compute_time.py

```
import random
from datetime import datetime

import pytz

from odoo import _, api, fields, models
from odoo.tools import DEFAULT_SERVER_DATETIME_FORMAT


class HrEmployee(models.Model):
    _inherit = "hr.employee"

    actual_time = fields.Float(
        compute="_compute_actual_time",
        compute_sudo=True,
        digits=(12, 2),
        string="Actual Time (Hours)",
    )

    automatic_logout = fields.Boolean(default=False)

    actual_time_str = fields.Char(
        compute="_compute_actual_time_str", compute_sudo=True, string="Actual Time"
    )

    time_message = fields.Char(
        compute="_compute_time_message", compute_sudo=True, string="Time Message"
    )

    attendance_ids = fields.One2many(
        "hr.attendance", "employee_id", help="list of the attendances of the employee"
    )

    max_work_time_per_day = fields.Float(default=10)
```

```
normal_work_time_per_day = fields.Float(default=8)

forced_checkout_normal_work_hour_check_box = fields.Boolean(default=False)

forced_checkout_max_work_hour_check_box = fields.Boolean(default=False)

forced_checkout_buffering_checkbox = fields.Boolean(default=False)

forced_checkout_buffer = fields.Float(default=0)

@api.depends("attendance_ids.check_in", "attendance_ids.check_out")
def _compute_actual_time(self):
    for record in self:
        user_tz = self.env.user.tz or "UTC"
        local_tz = pytz.timezone(user_tz)
        utc_tz = pytz.timezone("UTC")

        today = datetime.today().date()
        start_of_day = datetime(today.year, today.month, today.day, 0, 0, 0)
        attendance_of_today = record.attendance_ids.filtered(
            lambda att: att.check_in
            and datetime.strptime(
                att.check_in, DEFAULT_SERVER_DATETIME_FORMAT
            ).date()
            == today
        )
        logout_today = record.attendance_ids.filtered(
            lambda att: att.check_out
            and datetime.strptime(
                att.check_out, DEFAULT_SERVER_DATETIME_FORMAT
            ).date()
            == today
            and not datetime.strptime(
                att.check_in, DEFAULT_SERVER_DATETIME_FORMAT
            ).date()
            == today
        )

        total_hours = (
            sum(attendance_of_today.mapped("worked_hours"))
            if attendance_of_today
            else 0
        )

        hours_from_yesterday = 0
        check_in_recent = record.attendance_ids.filtered(
            lambda att: att.check_in and not att.check_out
        )

        if logout_today:
            replaced_check_in_time = local_tz.localize(start_of_day)
            check_out_time_utc = datetime.strptime(
                logout_today[0].check_out, DEFAULT_SERVER_DATETIME_FORMAT
            ).replace(tzinfo=utc_tz)
            check_out_time_local = check_out_time_utc.astimezone(local_tz)
```

```
        delta = check_out_time_local - replaced_check_in_time
        hours_from_yesterday += delta.total_seconds() / 3600.0

    if check_in_recent:
        check_in_time_utc = datetime.strptime(
            check_in_recent[0].check_in, DEFAULT_SERVER_DATETIME_FORMAT
        ).replace(tzinfo=utc_tz)
        check_in_time_local = check_in_time_utc.astimezone(local_tz)
        current_time_local = datetime.now(local_tz)
        delta = current_time_local - check_in_time_local
        total_hours += delta.total_seconds() / 3600.0

    work_time = total_hours

    info_message = _(
        "You have been logged out because you have "
        "exceeded your maximum working hours."
        " Please contact the HR department if you "
        "have any questions. Have a nice day!"
    )

    def force_checkout_if_needed(condition, message):
        if condition:
            self.env["hr.attendance"]._force_checkout(record)
            self.env.user.alert(
                title=_("Logout Information"),
                message=message,
                alert_type="info",
                sticky=True,
            )

    if check_in_recent:
        if record.forced_checkout_normal_work_hour_check_box:
            force_checkout_if_needed(
                record.normal_work_time_per_day <= work_time
                or record.normal_work_time_per_day
                <= work_time + hours_from_yesterday,
                info_message,
            )
        if record.forced_checkout_buffering_checkbox:
            buffer_time = (
                record.normal_work_time_per_day / 100
            ) * record.forced_checkout_buffer
            force_checkout_if_needed(
                buffer_time <= work_time
                or buffer_time <= work_time + hours_from_yesterday,
                info_message,
            )
        if record.forced_checkout_max_work_hour_check_box:
            force_checkout_if_needed(
                record.max_work_time_per_day <= work_time
                or record.max_work_time_per_day
                <= work_time + hours_from_yesterday,
                info_message,
            )
        if record.forced_checkout_buffering_checkbox:
```

```
                buffer_time = (
                    record.max_work_time_per_day / 100
                ) * record.forced_checkout_buffer
                force_checkout_if_needed(
                    buffer_time <= work_time
                    or buffer_time <= work_time + hours_from_yesterday,
                    info_message,
                )

            record.actual_time = (
                total_hours + hours_from_yesterday if check_in_recent else 0
            )

    @api.depends("actual_time")
    def _compute_actual_time_str(self):
        for record in self:
            hours = int(record.actual_time)
            minutes = int((record.actual_time * 60) % 60)
            record.actual_time_str = "{}h {}m".format(hours, minutes)

    @api.depends("actual_time")
    def _compute_time_message(self):

        for record in self:
            if record.actual_time > self.max_work_time_per_day:
                record.time_message = _("Check out immediately")
            elif record.actual_time > (self.max_work_time_per_day / 100 * 90):
                record.time_message = _(
                    "Check out soon, you will hit maximum work time soon!"
                )
            elif record.actual_time > self.normal_work_time_per_day:
                record.time_message = _("You're working overtime now")
            elif record.actual_time > (self.normal_work_time_per_day / 100 * 90):
                record.time_message = _("End of workday is near! :) ")
            elif record.actual_time > (self.normal_work_time_per_day / 2):
                record.time_message = _("Half time, keep it going")
            elif record.actual_time == 0:
                record.time_message = _("You are not logged in.")
            else:
                random_messages = {
                    1: _("Welcome, have a nice day!"),
                    2: _("Hello there! Wishing you a wonderful day ahead!"),
                    3: _("Greetings! May your day be filled with joy and success!"),
                    4: _("Hi! Hope your day is as amazing as you are!"),
                    5: _("Welcome! Have a fantastic day!"),
                }
                random_message = random.randint(1, 5)
                record.time_message = random_messages.get(random_message)
```

models/hr_attendance_forced_logout.py

```
from datetime import datetime

from odoo import fields, models
```

```
class HrAttendance(models.Model):
    _inherit = "hr.attendance"

    attendance_ids = fields.One2many(
        "hr.attendance", "employee_id", help="list of the attendances of the employee"
    )

    def _force_checkout(self, employee):
        attendance_records = self.search(
            [("employee_id", "=", employee.id), ("check_out", "=", False)]
        )
        for attendance in attendance_records:
            attendance.write({"check_out": datetime.now()})

    def mark_inactive(self):
        employee = self.env["hr.employee"].search(
            [("user_id", "=", self.env.uid)], limit=1
        )
        if employee and employee.automatic_logout:
            attendance = self.env["hr.attendance"].search(
                [("employee_id", "=", employee.id), ("check_out", "=", False)], limit=1
            )
            if attendance:
                attendance.write({"check_out": fields.Datetime.now()})
```

static/src/js/attendance_compute_time.js

```
odoo.define("hr_attendance_compute_time.MenuComputeDisplay", function (require) {
  "use strict";

  var WebClient = require("web.WebClient");
  var rpc = require("web.rpc");
  var session = require("web.session");
  var REFRESH_INTERVAL = 50000;
  var INACTIVITY_TIMEOUT = 900000;

  WebClient.include({
    show_application: function () {
      var self = this;
      this._super.apply(this, arguments);

      if (session.is_bound && $(".o_actual_time_display").length === 0) {
        var $my_compute_display = $("<li>", {
          html: '<a href="#" style="color: #FFFFFF; padding: 10px; background: #000000;">Actual Time:
Loading...</a>',
          class: "o_actual_time_display",
        });

        var $navbar = $("#oe_main_menu_navbar");
        var $mail_navbar_item = $navbar.find(".o_mail_navbar_item").first();

        if ($mail_navbar_item.length) {
          $mail_navbar_item.before($my_compute_display);
        } else {
```

```
        $navbar.append($my_compute_display);
      }

      setInterval(function () {
        self._compute_value().then(
          function (result) {
            var actual_time_str = result.actual_time_str;
            var time_message = result.time_message;
            var display_text = "Actual Time: " + actual_time_str;
            if (time_message) {
              display_text += " (" + time_message + ")";
            }
            $my_compute_display.find("a").text(display_text);
          },
          function (error) {
            console.error("Error computing value", error);
          }
        );
      }, REFRESH_INTERVAL);

      this._setupInactivityMonitor();
    }
  },

  _setupInactivityMonitor: function () {
    var self = this;
    var timeout = null;

    function markInactive() {
      self._mark_user_inactive();
    }

    function resetTimer() {
      clearTimeout(timeout);
      timeout = setTimeout(markInactive, INACTIVITY_TIMEOUT);
    }

    window.onload = resetTimer;
    window.onmousemove = resetTimer;
    window.onmousedown = resetTimer;
    window.ontouchstart = resetTimer;
    window.onclick = resetTimer;
    window.onkeypress = resetTimer;
  },

  _mark_user_inactive: function () {
    rpc
      .query({
        model: "hr.attendance",
        method: "mark_inactive",
        args: [session.uid],
      })
      .then(
        function () {
          console.log("User marked as inactive");
        },
```

```
      function (error) {
        console.error("Error marking user as inactive", error);
      }
    );
  },

  _compute_value: function () {
    var self = this;
    return self._get_employee_id().then(
      function (employee_id) {
        if (employee_id) {
          return rpc
            .query({
              model: "hr.employee",
              method: "read",
              args: [employee_id],
              kwargs: { fields: ["actual_time_str", "time_message"] },
            })
            .then(
              function (result) {
                return result[0];
              },
              function (error) {
                console.error("Error in hr.attendance read", error);
                return { actual_time_str: "Error reading actual time", time_message: "" };
              }
            );
        }
        return { actual_time_str: "Employee ID not found", time_message: "" };
      },
      function (error) {
        console.error("Error getting employee ID", error);
        return { actual_time_str: "Error getting employee ID", time_message: "" };
      }
    );
  },

  _get_employee_id: function () {
    return rpc
      .query({
        model: "hr.employee",
        method: "search_read",
        domain: [["user_id", "=", session.uid]],
        fields: ["id"],
        limit: 1,
      })
      .then(
        function (employees) {
          return employees.length > 0 ? employees[0].id : false;
        },
        function (error) {
          console.error("Error in search_read", error);
          return false;
        }
      );
  },
```

```
  });
});
```

views/assets.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<odoo>
      <template id="assets_backend" name="hr_attendance_compute_time_assets"
inherit_id="web.assets_backend">
         <xpath expr="." position="inside">
            <script
      type="text/javascript"
      src="/hr_attendance_compute_time/static/src/js/attendance_compute_time.js"
    />
         </xpath>
      </template>
</odoo>
```

views/employee_view.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<odoo>
   <data>
      <!-- Vererbung der hr.employee Ansicht zur Hinzufügung der erzwungenen Checkout-Felder -->
      <record id="view_employee_form_inherit_combined" model="ir.ui.view">
         <field name="name">hr.view_employee_form.inherit.combined</field>
         <field name="model">hr.employee</field>
         <field name="inherit_id" ref="hr.view_employee_form" />
         <field name="arch" type="xml">
            <!-- Hinzufügen der Automatic Logout Option nach resource_calendar_id -->
            <xpath expr="//field[@name='resource_calendar_id']" position="after">
               <field name="automatic_logout" string="Logout if inactive (15min)" />
            </xpath>
            <!-- Hinzufügen der erzwungenen Checkout-Felder nach user_id -->
            <xpath expr="//field[@name='user_id']" position="after">
               <field name="max_work_time_per_day" />
               <field name="forced_checkout_max_work_hour_check_box" string="Forced Checkout" />
               <field name="normal_work_time_per_day" />
               <field name="forced_checkout_normal_work_hour_check_box" string="Forced Checkout" />
               <field name="forced_checkout_buffering_checkbox" string="Buffer for Forced Checkout" />
               <field name="forced_checkout_buffer" string="in Percent" />
            </xpath>
         </field>
      </record>
   </data>
</odoo>
```

tests/static/__init__.py

```
from . import test_hr_attendance_compute_time
from . import test_hr_attendance_forced_logout
```

tests/static/test_hr_attendance_compute_time.py

```
import unittest
from datetime import datetime, timedelta
```

```
import pytz

from odoo.tests.common import TransactionCase
from odoo.tools import DEFAULT_SERVER_DATETIME_FORMAT


class TestHrEmployee(TransactionCase):
    def setUp(self):
        super(TestHrEmployee, self).setUp()
        self.employee = self.env["hr.employee"].create(
            {
                "name": "John Doe",
                "forced_checkout_normal_work_hour_check_box": False,
                "forced_checkout_max_work_hour_check_box": False,
                "max_work_time_per_day": 10,
                "normal_work_time_per_day": 8,
            }
        )
        self.attendance_model = self.env["hr.attendance"]

    def create_attendance(self, employee, check_in, check_out=None):
        vals = {
            "employee_id": employee.id,
            "check_in": check_in,
            "check_out": check_out,
        }
        return self.attendance_model.create(vals)

    def test_compute_actual_time_no_attendance(self):
        self.employee._compute_actual_time()
        self.assertEqual(
            self.employee.actual_time,
            0,
            "Actual time should be 0 when no attendances are recorded.",
        )

    def test_compute_actual_time_with_attendance(self):
        check_in = (datetime.now() - timedelta(hours=2)).strftime(
            DEFAULT_SERVER_DATETIME_FORMAT
        )
        self.create_attendance(self.employee, check_in)
        self.employee._compute_actual_time()
        self.assertAlmostEqual(
            self.employee.actual_time,
            2,
            places=2,
            msg="Actual time should be calculated correctly with attendance records.",
        )

    def test_compute_actual_time_with_attendance_and_work_time_from_yesterday(self):
        local_tz = pytz.timezone(self.env.user.tz or "UTC")
        utc_tz = pytz.timezone("UTC")

        today = datetime.now(local_tz).date()
        now = datetime.now(utc_tz)
```

```
        check_in_yesterday = (now - timedelta(days=1)).strftime(
            DEFAULT_SERVER_DATETIME_FORMAT
        )
        check_out_from_yesterday = (
            local_tz.localize(datetime(today.year, today.month, today.day, 3, 0, 0))
            .astimezone(utc_tz)
            .strftime(DEFAULT_SERVER_DATETIME_FORMAT)
        )
        check_in = (now - timedelta(hours=2)).strftime(DEFAULT_SERVER_DATETIME_FORMAT)

        self.create_attendance(
            self.employee, check_in_yesterday, check_out_from_yesterday
        )
        self.create_attendance(self.employee, check_in)
        self.employee._compute_actual_time()

        self.assertAlmostEqual(
            self.employee.actual_time,
            5,
            places=2,
            msg="Actual time should be calculated correctly with attendance records.",
        )

    def test_compute_actual_time_str(self):
        self.employee.actual_time = 3.5
        self.employee._compute_actual_time_str()
        self.assertEqual(
            self.employee.actual_time_str,
            "3h 30m",
            "Actual time string should be formatted correctly.",
        )

    def test_compute_time_message(self):
        self.employee.actual_time = 0
        self.employee._compute_time_message()
        self.assertEqual(
            self.employee.time_message,
            "You are not logged in.",
            "Message should be 'You are not logged in.' when actual time is 0.",
        )

        self.employee.actual_time = 7.5
        self.employee._compute_time_message()
        self.assertEqual(
            self.employee.time_message,
            "End of workday is near! :) ",
            "Message should be 'End of workday is near! :) ' "
            "when actual time is near max work hours.",
        )


if __name__ == "__main__":
    unittest.main()
```

tests/static/test_hr_attendance_forced_logout.py

```python
import unittest
from datetime import datetime, timedelta

from odoo import fields
from odoo.tests.common import SavepointCase
from odoo.tools import DEFAULT_SERVER_DATETIME_FORMAT


class TestHrAttendance(SavepointCase):
    @classmethod
    def setUpClass(cls):
        super(TestHrAttendance, cls).setUpClass()
        cls.hr_employee = cls.env["hr.employee"].create(
            {"name": "Test Employee", "user_id": cls.env.ref("base.user_demo").id}
        )

    def test_force_checkout(self):
        check_in_time = datetime.now() - timedelta(hours=3)
        attendance = self.env["hr.attendance"].create(
            {
                "employee_id": self.hr_employee.id,
                "check_in": check_in_time.strftime(DEFAULT_SERVER_DATETIME_FORMAT),
                "check_out": False,
            }
        )

        self.hr_employee.forced_checkout_normal_work_hour_check_box = True
        self.hr_employee.normal_work_time_per_day = (
            2.0  # Normalarbeitszeit zum Testen reduzieren
        )
        self.env["hr.attendance"]._force_checkout(self.hr_employee)

        attendance.refresh()
        self.assertIsNotNone(attendance.check_out)

    def test_mark_inactive(self):
        check_in_time = datetime.now() - timedelta(hours=1)
        attendance = self.env["hr.attendance"].create(
            {
                "employee_id": self.hr_employee.id,
                "check_in": check_in_time.strftime(DEFAULT_SERVER_DATETIME_FORMAT),
                "check_out": False,
            }
        )

        # Benutzereinstellung für den Mitarbeiter simulieren
        self.env.user.write({"employee_ids": [(4, self.hr_employee.id)]})

        self.env["hr.attendance"].mark_inactive()

        attendance.refresh()
        self.assertIsNotNone(attendance.check_out, "Check-out wurde nicht gesetzt")
        if attendance.check_out:
            self.assertAlmostEqual(
```

```
            fields.Datetime.from_string(attendance.check_out),
            datetime.now(),
            delta=timedelta(minutes=1),
        )


if __name__ == "__main__":
    unittest.main()
```

test/static/test_menu_compute_time.js (Wurde aus der Finalen Version gelöscht da kein Javascript Test nötig ist)

```
odoo.define('hr_attendance_compute_time.MenuComputeDisplayTests', function (require) {
    "use strict";

    var QUnit = require('web.qunit');
    var WebClient = require('web.WebClient');
    var rpc = require('web.rpc');
    var session = require('web.session');
    var sinon = require('sinon');

    QUnit.module('MenuComputeDisplay', {
        beforeEach: function() {
            this.webClient = new WebClient();
            this.rpcStub = sinon.stub(rpc, 'query');
        },
        afterEach: function() {
            this.rpcStub.restore();
        }
    });

    QUnit.test('show_application should add Actual Time display', async function (assert) {
        assert.expect(2);

        session.is_bound = true;
        session.uid = 1;

        this.rpcStub.withArgs(sinon.match({ method: 'search_read' })).resolves([{ id: 1 }]);
        this.rpcStub.withArgs(sinon.match({ method: 'read' })).resolves([{ actual_time_str: '4h 30m' }]);

        await this.webClient.show_application();
        var $computeDisplay = $('.o_actual_time_display');
        assert.equal($computeDisplay.length, 1, 'Actual Time display is added to the DOM');
        assert.equal($computeDisplay.find('a').text(), 'Actual Time: Loading...', 'Actual Time display shows
loading text initially');
    });

    QUnit.test('_compute_value should return actual time string', async function (assert) {
        assert.expect(1);

        this.rpcStub.withArgs(sinon.match({ method: 'read' })).resolves([{ actual_time_str: '4h 30m' }]);

        var actual_time_str = await this.webClient._compute_value();
        assert.equal(actual_time_str, '4h 30m', 'Actual time string is correctly computed');
    });
```

```
    QUnit.test('_get_employee_id should return employee id', async function (assert) {
        assert.expect(1);

        this.rpcStub.withArgs(sinon.match({ method: 'search_read' })).resolves([{ id: 1 }]);

        var employee_id = await this.webClient._get_employee_id();
        assert.equal(employee_id, 1, 'Employee ID is correctly retrieved');
    });
});
```