

Time Tracking System

Raffael Schäfer, Florian Wittmann

11. März 2024



Umwelt-Campus
Birkenfeld

H O C H
S C H U L E
T R I E R

Inhaltsverzeichnis

1	Grundidee	3
2	Systemübersicht	3
2.1	Hardware	3
2.1.1	ESP8266 und RC522	3
2.1.2	Zusätzliche Komponenten	3
2.2	Firmware	3
2.2.1	Datenerfassung	3
2.3	Server	3
2.3.1	Aufbau	3
3	Funktionen	4
3.1	Benutzerverwaltung	4
3.1.1	Anlegen	4
3.2	Authentifizierung	4
3.2.1	RFID Reader	4
3.3	Zeiterfassung (Erfassung des User-Verhaltens)	4
3.4	Datenanzeige	4
3.5	Sicherheit	5
4	Entwicklung	5
4.1	Hardware	5
4.1.1	Bridge	5
4.1.2	Hülle	5
4.2	API Design	6
4.3	Benutzeroberfläche	6
4.4	Datenverwaltung/Datenbearbeitung	6
4.5	Zeiterfassung	6
4.5.1	Unix Epoch Time	7
4.6	Sicherheit	7
4.7	Protokollierung	7
5	Optimierung und Zukunftsaussichten	7
5.1	Hardware	7
5.2	Sicherheit	7
5.3	Zeiterfassung	7
5.4	Server-Client-Verbindung	7
5.5	Server Architektur	7
5.6	Features	7
6	Appendix	8

Abbildungsverzeichnis

1	Leiterplattenplan aus dem Leiterplatteneditor der Software KiCad	7
2	Finale Version der Hülle	5
3	Entity Relationship Diagramm	6
4	Alle Vorgängerversionen des Hüllendesigns	8

1 Grundidee

Unser Ziel war es, ein benutzerfreundliches und effizientes System zu entwickeln, das es den Nutzern ermöglicht, ihre Arbeitszeit einfach und genau zu erfassen, sei es in einem Coworking-Space oder im Homeoffice. Wir haben uns darauf konzentriert, ein System zu schaffen, das sowohl für Einzelpersonen als auch für Teams geeignet ist und gleichzeitig flexibel genug ist, um den unterschiedlichen Arbeitsbedingungen gerecht zu werden.

Unsere Lösung präsentiert sich als ein handliches und platzsparendes Gerät, das nicht nur einfach zu transportieren und zu installieren ist, sondern auch mühelos zu bedienen. Bei der Entwicklung haben wir bewusst auf manuelle Dateneingaben verzichtet und stattdessen auf RFID-Technologie gesetzt, um das Ein- und Ausloggen ins System so einfach wie möglich zu gestalten.

2 Systemübersicht

2.1 Hardware

Unser Zeiterfassungssystem besteht aus zwei Hardware basierten Hauptkomponenten. Zum einen aus einem RFID RC522 NFC Reader, der mit einem Node MCU vom Typ ESP8266 Mod 12-F kommuniziert. Zum anderen NFC Chips, welche in diversen NFC Tags oder Karten verbaut sind.

2.1.1 ESP8266 und RC522

Wir haben uns für den ESP8266 entschieden, da der IOT-Oktopus keine freien Schnittstellen für den RC522 geboten hat, ohne dabei gewisse Schnittstellen doppelt zu belegen, was ein unvorhersehbares Verhalten der am Oktopus verbauten LED oder des RFID Readers zufolge hatte.

2.1.2 Zusätzliche Komponenten

Um die passende Kommunikation zwischen ESP8266 und RFID RC522 NFC Reader zu gewährleisten wurde eine Verbindungsplatine designt und im KI Labor mithilfe des dort vorhandenen Platinendruckers gefertigt. Zudem wird das System durch eine 3D gedruckte Hülle vor leichten Verschmutzungen geschützt und zusätzlich noch etwas ansehnlicher gestaltet.

2.2 Firmware

Die Firmware des ESP8266 steuert das Verhalten der Hardware. In dieser findet die Verbindung des ESPs mit einem geeigneten Router zur Herstellung einer aktiven Internetverbindung, der Zugriff auf einen geeigneten NTP Server und die Verarbeitung der vom RFID RC522 NFC Readers erhaltenen Signale statt. Essenzielle Daten, wie der API-Key oder die Verbindungsdaten zum gewünschten Router stehen hierbei nicht im Code selbst, sondern in den separat ausgelagerten Header-Dateien "api.h" und "credentials.h".

2.2.1 Datenerfassung

Erhält der ESP8266 ein Signal vom RFID RC522, so wird mit diesem Signal die ID des NFC Tags übermittelt. Anschließend wird ein Update der Zeit vom NTP Server angefordert. Folglich werden die beiden erhaltenen Daten "ID" und "Zeit" zusammen mit dem API-Key in eine temporäre JSON-Datei verpackt und an den in der "api.h" eingetragenen Server übermittelt.

2.3 Server

Der Server läuft über einen Docker Container und kann somit flexibel und überall gehostet werden. Die verwendeten Ports müssten aber noch manuell freigegeben werden, damit der Node MCU darauf zu greifen kann. Sonst kann auf die Software zu Testzwecken via Kommandozeile gestartet werden. Dabei ist wichtig, dass die Adresse des Server auf 0.0.0.0 gesetzt wird, damit er auch im Ganzen Netz verfügbar ist.

2.3.1 Aufbau

Der Server besteht aus drei wichtigen Komponenten, sie bilden die Kernfunktionalität der Software

- Pug.py (Rendert das Frontend)
- Flask (API)
- SQLite (Datenbank)

3 Funktionen

3.1 Benutzerverwaltung

3.1.1 Anlegen

Das Anlegen von Nutzern passiert über die Web-Benutzeroberfläche. Ein Nutzer wird über den Knopf “Create new User” in der Nutzerauflistung angelegt. Dafür werden folgende Daten benötigt:

- Vorname
- Nachname
- Adresse:
 - Straßename
 - Hausnummer
 - Ortsname
 - Postleitzahl
 - Land
- Position

Falls eine Position fehlt, die der neue Nutzer erfüllen soll, kann sie innerhalb der Positionsanlegung erstellt werden. Man findet innerhalb des Formulars zu Erstellung eines Nutzers einen Link auf die Positionserstellung. Eine neue Karte kann auch per Weboberfläche angelegt werden. Dies geschieht ähnlich wie bei der Anlegung eines Nutzers, eine Karte wird unter dem Knopf “Create new Card” angelegt. Zur Erstellung einer Karte wird nur die UID gebraucht. Um eine Karte einem Nutzer zuzuweisen, muss man dann auf den Knopf namens “Grant ownership” drücken. Danach kann man eine verfügbare Karte einem beliebigen Nutzer zuweisen. Das Aktualisieren und Entfernen von Daten kann unter den jeweiligen Detailansichten passieren.

3.2 Authentifizierung

3.2.1 RFID Reader

Der User kann sich im System anmelden, via dem RFID Reader (ESP32F), dieser teilt dem Server mit welche NFC Karte sich eingeloggt hat und gibt gleichzeitig auch noch die Unix epoch time mit.

3.3 Zeiterfassung (Erfassung des User-Verhaltens)

In der jetzigen Version des Projektes fällt die Zeiterfassung leider nur minimal aus. Das System speichert, wenn ein Nutzer sich an oder abmeldet und zeigt das in zwei Ansichten im Web-UI. Einmal gibt es eine große Auflistung auf der Hauptseite, welche Karten sich zuletzt an-/abgemeldet haben. Des weiteren gibt es noch das Userprofil, welches grade den Status eines Nutzers zeigt und seine letzten An-/Abmeldungen. Folgende Daten werden in den jeweiligen Listen gezeigt:

- Uhrzeit
- Karten UID (nur in der Gesamtauflistung auf der Hauptseite)
- Status(online/offline)

Die Zeiterfassung erfolgt über einen NTP Server, der in der Firmware des Node MCUs angesteuert wird. (siehe 4.5) Dazu wurde die “NTPClient.h” Bibliothek eingebunden und verwendet. Der in der Firmware angegebene “utcOffset” muss dabei nach aktuellem Stand der Firmware je nach Winter oder Sommerzeit manuell geändert werden.

3.4 Datenanzeige

Alle gesammelten Daten werden in der Weboberfläche dargestellt:

- User Daten (Name, Wohnort, etc.)
- RFID Karten
- Positionen
- Generierte Logs

Für die meisten Daten gibt es zwei Ansichten, ein mal eine Auflistung aller Daten und einer Detailansicht, bei der man noch mehr Informationen bekommt und die Relationen zwischen den Daten sehen kann (zum Beispiel wem gehört welche Karte). Von dem Ansichtsmodell sind die generierten Logs ausgeschlossen (siehe 3.3).

3.5 Sicherheit

Der Server wird vor falschen Log Einträgen via eines API-Keys geschützt, der bei jeder Anfrage mitgeschickt werden muss (siehe 4.2). Sonstige Schutzmaßnahmen sprengten leider den Scope des Projektes.

4 Entwicklung

4.1 Hardware

4.1.1 Bridge

Zur Entwicklung der Verbindungsplatine wurde die Hardware zunächst auf einem Breadboard aufgesteckt und eine geeignete Verbindung der notwendigen Pins mithilfe ausreichender Steckbrückenkabel hergestellt und umfangreich getestet. Anschließend wurde der Schaltplan nach Vorlage des Breadboards in KiCad Version 7.0 erstellt, woraus sich im weiteren Verlaufe ein Leiterplattenentwurf erstellen ließ. Der Druck der Leiterplatte erfolgte auf Anfrage im KI-Labor am Umwelt-Campus Birkenfeld.

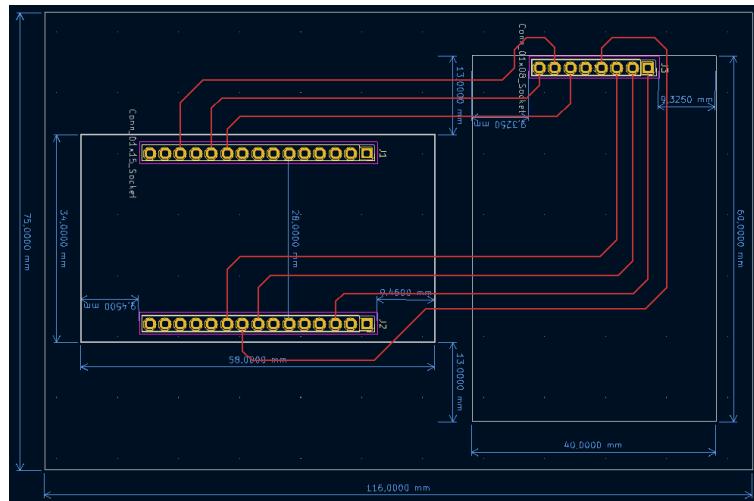


Abbildung 1: Leiterplattenplan aus dem Leiterplatteneditor der Software KiCad 7

4.1.2 Hülle

Das Gehäuse wurde innerhalb von Fusion 360 designt und dann mit PLA Filament 3D gedruckt. Es hat mehrere Anläufe gebraucht, um das richtige Design hinzubekommen.

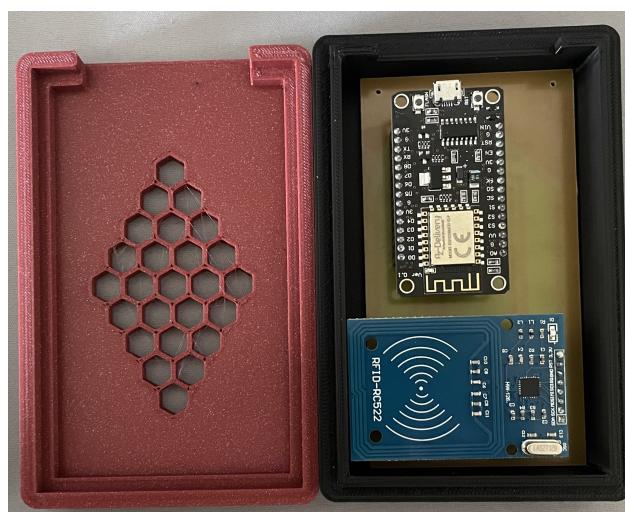


Abbildung 2: Finale Version der Hülle

4.2 API Design

Zur Erstellung unseres Systems haben wir uns auf folgende JSON Struktur geeinigt, um die Daten zwischen den verschiedenen Softwaresystemen zu kommunizieren.

```
{
  "key": "api key",
  "data": {
    "request_data": "Some Data"
  },
  "error": "Error Message"
}
```

- **Key:** Das Key Attribut enthält den API-Schlüssel, welcher das System schützt vor nicht autorisierten Zugriffen.
- **Data:** Ist ein weiteres JSON Objekt innerhalb der Abfrage, welches Abfragen spezifische Daten enthält.
- **Error:** Error enthält den Text einer entstandenen Fehlermeldung.

4.3 Benutzeroberfläche

Die Umsetzung der Benutzeroberfläche für unser Zeitverwaltungssystem wurde durch die Anwendung der Template Engine Pug realisiert, welche eine effiziente und elegante Möglichkeit bietet, HTML-Dateien zu generieren. Zur Gestaltung der Bedienelemente griffen wir auf Bootstrap zurück, eine bewährte Bibliothek für responsives Design und eine Vielzahl von vorgefertigten Komponenten. Darüber hinaus ermöglichte uns Alpine.js, dynamische Inhalte auf den Seiten nahtlos zu ändern und eine interaktive Benutzererfahrung zu schaffen. Wir haben versucht, ein einheitliches Design über die Seiten verteilt zu verwenden. Auf der Startseite hat man eine Übersicht über alle Ein-/Abmeldungen. Für die Punkte, Nutzern, Karten und Positionen gibt es jeweilige Listen und Detailansichten. Um Daten anzulegen oder zu verändern, gibt es die passenden Formulare. Um Daten zu löschen, gibt es eine einheitliche Seite zur Bestätigung der Aktion. Das Farbschema ist bewusst irreführend gewählt, damit ein Nutzer länger braucht, um die Aktion zu vollenden, um fehl Entscheidungen zu minimieren.

4.4 Datenverwaltung/Datenbearbeitung

Nutzerdaten werden innerhalb der Webformulare erstellt und dann via den Control Schnittstellen in eine SQLite Datenbank eingetragen. Die Formulardaten werden via POST Request an die Route zurückgegeben und dort werden die Daten weiter gegeben an den Controller, der letztlich die richtige Funktion auf unserem Modell ausführt. Dieses Modell ist eine Abstraktion von unserer SQL Datenbank, welche die Entwicklung erleichtert und durch seine einheitlichen Schnittstellen Fehler verhindert. Die Modelle können alle CRUD Operationen ausführen auf dem Modell, Funktionsweisen, welche komplexer sind, sollten von Controller übernommen werden, um unnötige Komplexität zu verhindern.

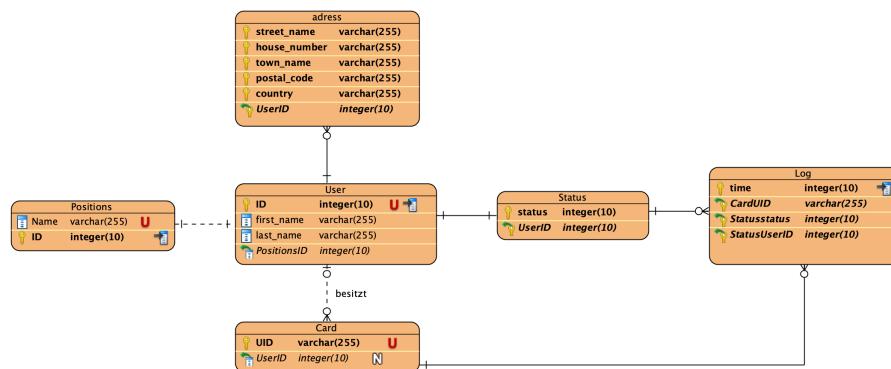


Abbildung 3: Entity Relationship Diagramm

4.5 Zeiterfassung

Zunächst wurden Nachforschungen betrieben, welche Möglichkeiten es gibt, um eine zuverlässige Erfassung der Zeit zu gewährleisten. Neben diversen hardwarebasierten Erfassungsmethoden fiel Wahl schließlich auf eine softwarebasierte Erfassung. Dies umfasst die Zeitanfrage über eine Anfrage an einen Network Time Protokoll Server. Der Vorteile, der sich hieraus ergibt, ist, dass keine weitere Hardware benötigt wird, um die Zeit zu erfassen. Der größte Nachteil dieser Methode ist jedoch, dass zwangsläufig eine Netzwerkverbindung bestehen muss, um die Zeit vom NTP Server erhalten zu können.

4.5.1 Unix Epoch Time

Die Zeit wird hierbei lediglich als Unix Timestamp, also der Anzahl an vergangenen Sekunden seit dem 1. Januar 1970, 00:00 Uhr UTC, erfasst und an den Server weitergegeben, da die Zeit dadurch einfach als Integer in der Datenbank hinterlegt werden kann. Hier kommt es jedoch zu Komplikationen der Zeiterfassung am 19. Januar 2038, da unsere Datenbank lediglich 32-Bit Integer speichert.

4.6 Sicherheit

Um sicherzustellen, ob ein Zeiterfassungsgerät (ESP) auf den Server zu greifen darf, wird bei jeder Anfrage ein Schlüssel mitgegeben, der bei jeder „log“ Anfrage überprüft wird.

4.7 Protokollierung

Wenn ein Nutzer eine RFID Karte an das Lesegerät hält, erfasst der ESP die UID in der Karte. Danach macht der ESP eine POST Anfrage an den Server. Anschließend trägt der Server den erzeugten Eintrag in die Log-Tabelle ein. Ein Eintrag besteht aus der Epoch Time, der Karten UID, dem zu gehörigen User und dem veränderten Status.

5 Optimierung und Zukunftsaussichten

5.1 Hardware

- Kleinere Platine
- Kleineres Gehäuse
- USB-C als Stromversorgung
- Wandhalterung für Gebäude-Eingänge (als Add-on)
- Geräte Feedback, um dem Nutzer zu signalisieren, dass der An- und Abmeldevorgang geklappt hat.
- Hardware basierte Uhrzeit (bessere Systemzeit)
- Outdoor geeignete Hardware

5.2 Sicherheit

- Login System für das Webinterface
- Besserer API Schutz

5.3 Zeiterfassung

- automatisierte Winter/Sommerzeit
- offline Zeiterfassung

5.4 Server-Client-Verbindung

- Queue als Zwischenspeicher für Serveranfragen, falls Server mal nicht erreichbar
- Server Cluster für Hochverfügbarkeit

5.5 Server Architektur

- Postgres statt SQLite als Datenbank (Docker Container)
- Aufteilung der Logischeneinheiten in Microservices (db, api, frontend)
- Verwendung eines ORM statt selbst geschriebenen SQL Befehlen
- Frontend basierend auf einem modernen Web Framework (React, Vue, Svelte) (Client-Side Rendering)

5.6 Features

- Chat Funktion via LLM (z.B. Llama2 oder Gemma)
- Support für Microsoft Teams
- Login System
- Bessere Arbeitszeit Analyse und Visualisierung

6 Appendix

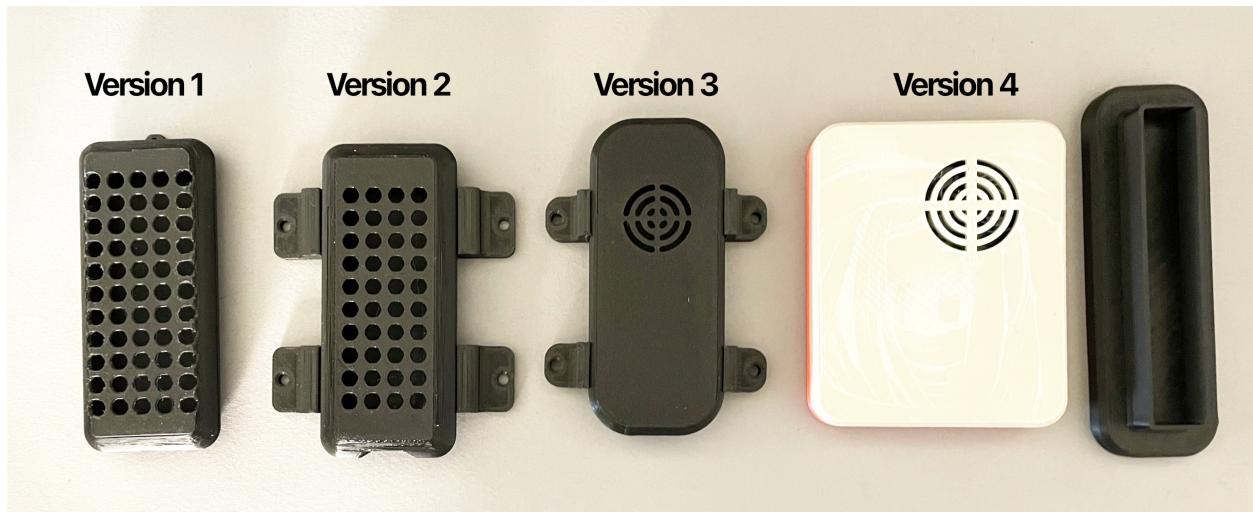


Abbildung 4: Alle Vorgängerversionen des Hüllendesigns