Projektlaborbericht

HOCHSCHULE HEILBRONN
Reinhold-Würth-Hochschule
Campus Künzelsau
Fakultät für Technik und Wirtschaft

Mit dem Thema:

Programm für die Zeiterfassung von Kleinunternehmen

eingereicht durch:

Name:	David Kempf	Sean Woods	Lauritz Abel
Matrikelnummer:	207815	207820	207804
Studiengang:	AE		
	Hochschule Heilbronn – Campus Künzelsau		

betreut von: Prof. Dr.-Ing. Marcus Stolz
Hochschule Heilbronn – Campus Künzelsau

Künzelsau, 26.07.2023

Ehrenwörtliche Erklärung

Hiermit erklären wir, David Kempf, Sean Woods und Lauritz Abel, dass wir die vorliegende Arbeit selbstständig angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Es wurden keine anderen als die angegebenen Quellen und Hinweise verwendet.
Die vorliegende Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Künzelsau, 26.07.2023 David Kempf, Sean Woods, Lauritz Abel

Inhaltsverzeichnis

1. Ist-Zustand

Für kleine Handwerks- und Landschaftspflegebetriebe besteht aktuell das Problem, dass es keine mobile digitale Lösung für die Zeitdokumentation gibt. Momentan dokumentieren viele kleine Betriebe ihre Zeiten in kleinen Notizbüchern. Maschinenstunden werden wegen des Aufwandes oft nicht richtig dokumentiert und die Zeiten werden geschätzt. Es gibt die Möglichkeit, mittels von Softwareprodukten wie zum Beispiel von Stihl (Stihl Connector), Maschinenzeiten aufzunehmen und zu dokumentieren. Jedoch müssen diese wieder händisch in ein Rechnungstool übertragen werden. Ein weiteres Problem ist, dass die Büroarbeit von Handwerkern oft als lästig und unnötig betrachtet wird und somit wieder zu kurz kommt. Jedoch verdient ein Betrieb sein Geld nicht nur mit der verrichteten Arbeit, sondern mit dem Ausstellen von korrekten Rechnungen.

2. Kurzfassung

Um den aktuellen Zustand zu verbessern, ist das Ziel dieser Arbeit, mit Hilfe einer App die Zeiterfassung und Rechnungsstellung zu automatisieren. Das Programm soll in zwei Bereiche aufgeteilt werden. Eine Desktopanwendung, welche zur Aufgabenplanung und zum Controlling genutzt werden soll und eine mobile App welche als Anwenderapp, für die Zeiterfassung gedacht ist. Das Programm soll es dem Anwender erleichtern seine Arbeitszeiten und Maschinenzeiten zu dokumentieren und zu verwalten. Nachdem ein Auftrag abgearbeitet ist, soll es mittels eines Klicks möglich sein, eine Rechnung erstellen zu lassen. Somit wird die Zeit im Büro minimiert und die effektive Arbeitszeit erhöht.

3. Aktuell Verfügbare Systeme

TOGGL

Toggl ist ein großes Zeitverwaltungssystem, das von Unternehmen wie Amazon, SAP, Linkedin usw. genutzt wird. Das Programm besticht mit einer einfachen Bedienung. Somit kann auch ohne eine Schulung mit dem Programm gearbeitet werden. Die Software ist sehr offen aufgebaut und lässt sich mit anderen Softwareprodukten wie Jira, Salesforce usw. verbinden.

AtWork

AtWork ist eine kostenpflichtiges Zeiterfassungssystem. In diesem lässt sich das ganze Mitarbeiterteam verwalten. Das Programm hebt sich vor allem mit der Funktion der Überstundenberechnung und Auszahlung und der Funktion der Schichtplanung hervor.

Factro

Factro ist eine cloudbasierte App. Diese speichert die Arbeitsstunden projektbezogen ab. In der App ist es möglich, ein ganzes Team zu verwalten. Mithilfe der Controlling Funktion ist es möglich, die Auslastung des Teams auszuwerten und zu bewerten. Eine kleine Basis-Version ist kostenlos. Danach müssen kostenpflichtige Lizenzen gekauft werden.

Die bestehenden Anwendungen sind oft sehr teuer und somit nicht für kleine Betriebe rentabel oder geeignet. Die kostenlosen Apps sind oft unflexibel und eigenen sich nicht für den kommerziellen Gebrauch. Momentan fehlt auf dem Markt eine einfache und flexible Anwendung, welche sich auf die relevanten Funktionen begrenzt.

4. Entwicklung

GitHub

Was ist GitHub? GitHub ist eine cloudbasierte Software, welche die Datenverwaltung und den Projektablauf vereinfacht. Somit ist es möglich, dass mehrere Entwickler gleichzeitig an einem Sourcecode schreiben können oder Daten gemeinsam geteilt werden. Die neu entwickelten Inhalte können mit allen geteilt werden und werden versioniert abgespeichert. Somit ist für alle ersichtlich, was geändert oder hinzugefügt wurde. Es ist nicht nur möglich, Sourcecode zu verwalten, sondern auch andere Dateien wie Word, Excel, PowerPoint und vieles mehr.

JetBrains Rider

JetBrains Rider oder kurz nur Rider genannt, wurde als Entwicklungsumgebung (IDE) gewählt, da diese einige Vorteile mit sich bringt. Rider ähnelt sehr stark der Entwicklungsumgebung Visual Studios von Microsoft. Hier liegt auch schon ein großer Unterschied. Rider ist nicht von Microsoft. Rider ist eine komplett offene Entwicklungsumgebung und nicht an die von Microsoft gegebenen Einschränkungen gebunden. Anders als Visual Studio unterscheidet sich die IDE hierbei nicht bei der Installation auf verschiedenen Betriebssystemen. Refactoring- und Code-Fix-Tools sind kostenlos nutzbar. Der wohl größte Vorteil von Rider liegt jedoch in der Performance, da Rider im Gegensatz zu Visual Studio nicht an einen 32-Bit Prozessor gebunden ist. Außerdem ist die Entwicklungsumgebung sehr benutzerfreundlich, da diese eine XAML-Vorschau bietet, welche in Echtzeit die Xamarin XAML Seite zeigt. Bei einem Fehler im Sourcecode kann mithilfe der Codeanalyse der Fehler schnell ausgemacht und behoben werden. Tritt beim Ausführen ein Fehler auf, unterstützt der Debugger den Entwickler. Tritt während

der Bearbeitung ein Fehler auf, springt der Debugger im Programm an die jeweilige Stelle und zeigt dem Entwickler die Exception, wodurch Probleme schnell identifiziert und gelöst werden können.

.NET-Framework

Das .NET-Framework wurde von Microsoft entwickelt und bedeutet übersetzt so viel wie „Rahmenarbeit“. Hierbei dient das .Net Framework als Basisprogramm für die Programmierung von Desktop- und mobilen Anwendungen. Es besteht aus einer Laufzeitumgebung, Klassenbibliothek und einem Compiler. Der geschriebene Sourcecode, der im .Net Framework programmiert worden ist, wird in der Laufzeitumgebung ausgeführt. Die Klassenbibliothek dient als nützliche Quelle für jegliche Anwendungen, wie zum Beispiel für SQ Lite Datenbanken, PDF-Generatoren, Socket Server und so weiter. Um das .Net Framework nutzen zu können, wird ein Texteditor und ein Compiler benötigt. Hierfür wird in der folgenden Arbeit die IDE Rider von Jetbrains verwendet.

WPF .NET

Windows Presentation Foundation (WPF) ist eine Klassenbibliothek von .NET, welche zur Entwicklung von graphischen Oberflächen genutzt wird. WPF unterstützt verschiedenste Arten von GUIs: Typische Desktopanwendungen, 3D-Grafiken, Dokumente, Browser-basierte Anwendungen und Videos. WPF ist von Microsoft als Nachfolger für das aus .NET bekannte Windows Forms eingeführt worden. Eine WPF-Oberfläche kann als Programmcode programmiert, oder wie von Rider unterstützt, durch XAML-Language programmiert werden. Die Anwendung ist vektorbasiert aufgebaut. Daher ist die Auflösung unabhängig von der Bildschirmgröße. Dieser Vorteil wird deutlich beim Verwenden von Responsiv Designs. Beim Responsiv Design passt sich die grafische Oberfläche der Gesamtgröße an. Zieht man die Anwendung nun größer, leidet die Auflösung nicht darunter und der Inhalt passt sich an die Fenstergröße an. Vor allem ist dies aus

der Webentwicklung bekannt, wodurch mithilfe von HTML und JS äußerst schnell ansprechende Frontends für jede beliebige Displaygröße entwickelt werden können. (IT-Vision, 2023)

C#

C# wird in diesem Projekt als Programmiersprache verwendet. Wie auch Java ist C# eine objektorientierte Sprache. Ihren Ursprung hat diese auch bei Microsoft Anfang der 2000er Jahre. Die Sprache wurde plattformunabhängig im Rahmen der .NET-Strategie von Microsoft entwickelt. Sie sollte das Erstellen von Windowsanwendungen mit der .NET-Framework vereinfachen. Hierbei greift die Sprache Konzepte von Java, C++, Haskell, C sowie von Delphi auf.
Zeiger, welche in C++ als eher unsicher bekannt sind, werden in C# nur für den sogenannten „unsicheren Code“ zugelassen, so zum Beispiel Programme, die von einer Website ausgeführt werden ohne erweiterte Rechte. Da C# als .NET-Sprache gilt, gibt es auch die Möglichkeit einer Sprachunterstützung für Attribute und Delegaten. In Metadaten werden in C# Informationen über eine Klasse, ein Objekt oder eine Methode gespeichert. Diese werden zur Laufzeit ausgewertet.

MVVM

Das Model-View-ViewModel (MVVM) ist ein Architekturmuster, das häufig in der Entwicklung von WPF-Anwendungen verwendet wird. Es zielt darauf ab, die Trennung von Benutzeroberfläche „View“ und dem Anwendungsobjekt „Model“ zu erreichen, indem es eine separate Komponente (ViewModel) zwischen diesen beiden Schichten einführt.
Das Model repräsentiert die Daten und das Anwendungsobjekt der Anwendung. Es kann eine Datenbank, eine Datei oder jede andere Datenquelle sein. Das Model ist unabhängig von der Benutzeroberfläche und enthält normalerweise Klassen und Eigenschaften, die die zugrunde liegenden Datenstrukturen darstellen.

Die View ist die grafische Benutzeroberfläche, die dem Benutzer angezeigt wird. Sie enthält die XAML-Datei, die das Aussehen und das Layout der Anwendung definiert. Die View hat keine direkte Kenntnis von dem Anwendungsobjekt oder den Daten, sondern bindet stattdessen ihre Steuerelemente an Eigenschaften im ViewModel.

Das ViewModel dient als Vermittler zwischen der View und dem Model. Es enthält Eigenschaften, Befehle und andere Logik, die für die Interaktion mit der Benutzeroberfläche erforderlich sind. Wenn die View Benutzereingaben empfängt, werden diese Daten an das ViewModel weitergeleitet, das die entsprechenden Aktionen im Model auslöst. Ebenso aktualisiert das ViewModel die Eigenschaften, die von der View gebunden sind, um Änderungen im Model widerzuspiegeln.

Die Verwendung von MVVM in WPF bietet mehrere Vorteile. Erstens ermöglicht es eine klarere Trennung von Aufgaben, was die Wartbarkeit und Testbarkeit der Anwendung verbessert. Da das ViewModel keine Abhängigkeiten zur View hat, können Entwickler die Anwendungsobjekte isoliert testen. Zweitens ermöglicht das Datenbindungssystem von WPF eine effiziente Kommunikation zwischen View und ViewModel, wodurch sich der Code reduziert, und die Entwicklung beschleunigt. (Microsoft, 2023)

Entity-Relationship-Modell (ER-Modell)

Zu Beginn des Projektlabors wurde ein ER-Modell erarbeitet. Ein ER-Modell ist eine grafische Darstellung, um Beziehungen zwischen sogenannten Entitäten in einer Datenbank zu beschreiben, welches häufig in der Datenbankentwicklung verwendet wird. Ein ER-Modell umfasst und beinhaltet folgende Inhalte.

1. Entitäten (engl. Entities): Entitäten sind Objekte oder Konzepte, deren Inhalt in der Datenbank sein muss und in Beziehung zueinander stehen. Es können sowohl reale Objekte (z. B. Kunden, Projekte, Mitarbeiter) oder abstrakte Konzepte (z. B. Aufträge, Transaktionen) sein. Jede Entität wird durch verschiedene Attribute beschrieben, die ihre spezifischen Eigenschaften darstellen (Beim Kunden z. B. ID, Name, Adresse).
2. Beziehungen (engl. Relationships): Durch Beziehungen werden die Verbindungen zwischen verschiedenen Entitäten aufgezeigt. Sie beschreiben, wie die Entitäten miteinander in Beziehung stehen und verknüpft sind. Beispielsweise besteht zwischen einem Kunden und dem von ihm erteilten Auftrag eine direkte Beziehung. Hierbei können verschiedene Häufigkeiten zugeteilt werden. So kann ein Kunde mehrere Aufträge erteilen, pro Auftrag gibt es jedoch nur ein Angebot.
3. Attribute: Attribute sind Eigenschaften oder Merkmale, welche die Entitäten näher beschreiben. Jede Entität hat verschiedenste Attribute, die ihre spezifischen Informationen repräsentieren. Zum Beispiel könnte eine Entität "Kunde" Attribute wie ID, Name, Adresse, E-Mail und Telefonnummer haben.

Das ER-Modell wird meistens als Diagramm dargestellt, das als Entity-Relationship-Diagramme (ER-Diagramme) bekannt ist. In diesen Diagrammen werden die Entitäten üblicherweise als Rechtecke, die Attribute als Ovale und die Beziehungen als Verbindungslinien mit Rauten zwischen den Entitäten dargestellt.
Das ER-Modell, half bei der ersten Ideenfindung welche Objekte das Projekt umfasst und wie diese anschließend bei der Programmierung in Beziehung gebracht werden sollten. Zudem diente es als Überblick, was noch zu machen, beziehungsweise einzubinden ist.

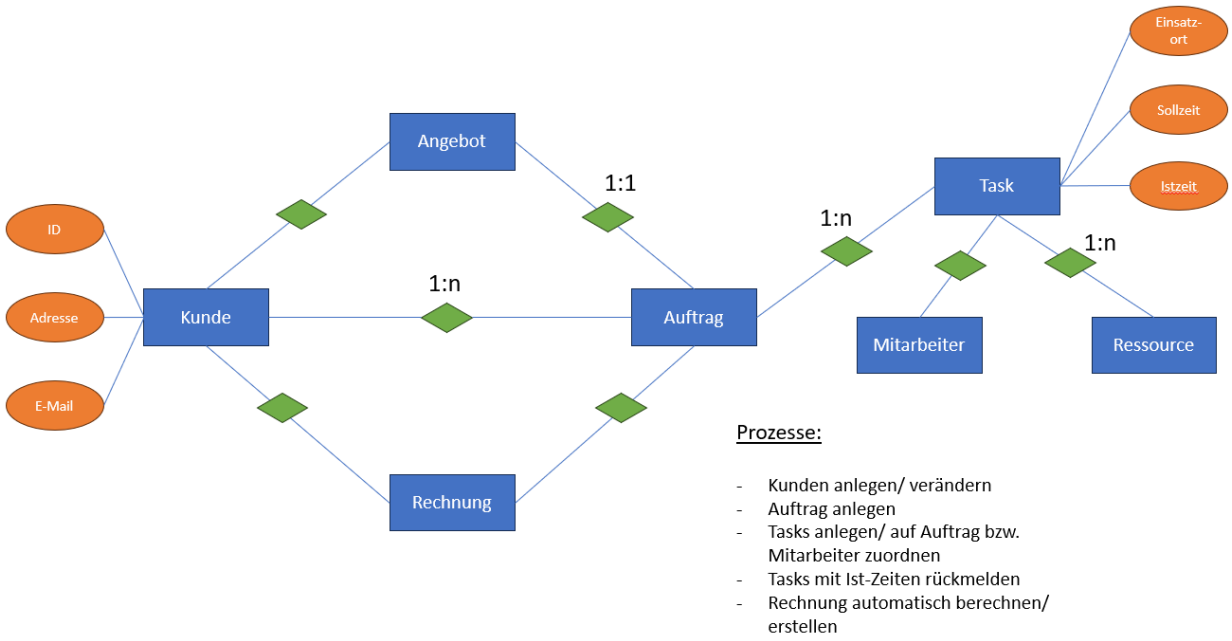

Abbildung 1: ER-Modell

Aufbau Datenbank

Für die Datenspeicherung und Datenverwaltung wurde eine SQ-Lite Datenbank verwendet. SQ-Lite bietet die Möglichkeit, direkt in die Anwendung eine Datenbank zu implementieren, ohne dass eine weitere Server-Software benötigt wird. Bei dem Erstellen der Datenbank ist es wichtig, keinen absoluten Pfad zu verwenden. Ein indirekter Pfad hat den Vorteil, dass dieser, egal auf welchem PC die Anwendung ausgeführt wird, funktioniert. Ein absoluter Pfad birgt die Gefahr in sich, dass der Pfad nicht existiert und die Anwendung nicht ausgeführt werden kann. Als Pfad empfiehlt sich der Debug Ordner zu verwenden, denn dieser ist unabhängig vom Installationsort. In Rider können Datenbanken direkt eingefügt werden und deren Inhalt betrachtet werden.

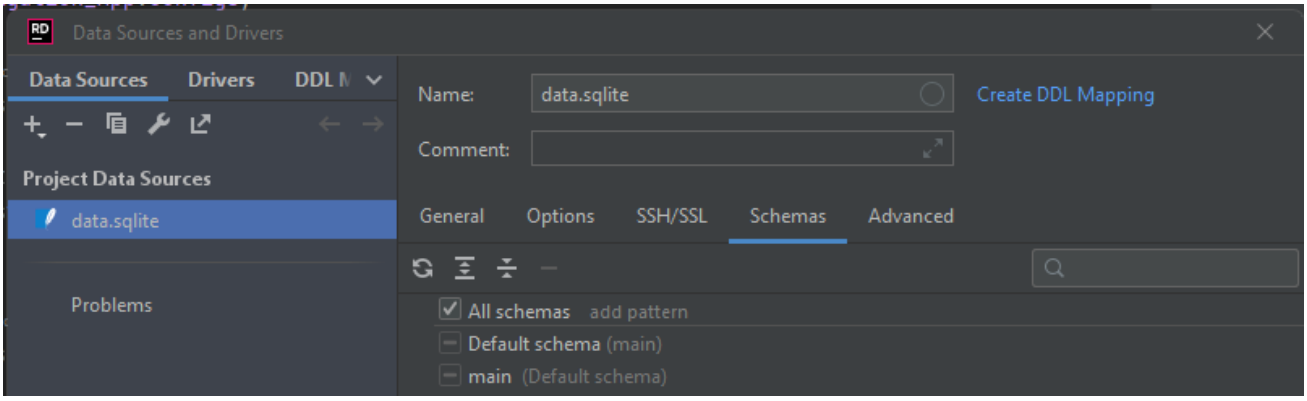

Abbildung 2: Datenbank in Rider einfügen

Zum Betrachten muss die Funktion „All schemas“ aktiviert werden. Mittels dieser Funktion werden alle später erstellten Tabellen angezeigt. Für jede Datenbank wird in dem Ordner DB einen Klasse erstellt. Diese wird mit einem Rechtsklick und dem Befehl „Add“ hinzugefügt. In der Public Class, wie zum Beispiel „Db_Customer“, werden sämtliche „Eigenschaften“ für die Tabellenerstellung definiert. Hierfür beschreibt jede Zeile eine Spalte der Tabelle. In jeder Zeile muss der Spaltennamen und der Datentyp definiert sein.

```
public string Character { get; set; }
public string BgColor { get; set; }
public string ID { get; set; }
public string Name { get; set; }
public string Adress { get; set; }
public string Mail { get; set; }
public string Phone { get; set; }
```

Mit diesen Informationen kann nun eine Tabelle in der Datenbank erstellt werden. Hierfür wird eine Methode erstellt. Diese funktioniert nach dem try catch Prinzip. Dieses versucht die Funktion, die im try steht, auszuführen solange die Bedingungen für try erfüllt sind. Werden diese nicht erfüllt geht das Programm automatisch in catch und führt die Funktionen in catch aus. In folgendem Sourcecodeabschnitt wird im try eine neue Tabelle erstellt solange diese noch nicht vorhanden ist. Falls eine vorhanden ist, wird eine Fehlermeldung ausgegeben.

```
public static Error CreateTable(string dataSource)
{
    SQLiteConnection conn = null;

    try
    {
        conn = new SQLiteConnection(dataSource);
        conn.CreateTable<Db_Customer>();

        return null;
    }
    catch (Exception ex)
    {
        return new Error(ex.ToString());
    }
    finally
    {
        conn?.Close();
    }
}
```

In dem nächsten Schritt muss nun die gerade erstellte Tabelle noch mit Zeilen und Inhalt gefüllt werden. Hierfür wird eine neue Klasse für das Schreiben und Lesen der Tabelle erstellt, zum Beispiel: „RW_Customer.cs“.

```
/// Reihen in DB schreiben
/// </summary>
public static Error Write(List<Db_Customer> rows, string
dataSource)
{
    SQLiteConnection conn = null;

    try
    {
        //Aufbau Verbindung zur Datenbank
        conn = new SQLiteConnection(dataSource);
        //Alle Reihen durchsuchen ob ID bereits vorhanden
        foreach (var row in rows)
        {
            string where = $"WHERE ID='{row.ID}'";
```

```
            // pruefen ob Item/Reihe vorhanden ist
            var item = conn.Query<Db_Customer>($"SELECT * FROM
{nameof(Db_Customer)} {where}").FirstOrDefault();
            if (item != null)
            {
                // schon vorhanden, loeschen
                conn.Execute($"DELETE FROM {nameof(Db_Customer)}
{where}");
            }
            //neue Reihe einfügen
            conn.Insert(row);
        }
```

Das Einfügen oder Beschreiben einer Zeile folgt immer der gleichen Methode. Zuerst wird eine Verbindung zu der Datenbank aufgebaut. Danach wird die jeweilige Tabelle danach durchsucht, ob die dementsprechende Reihe schon vorhanden ist und ob in dieser der zu ändernde Wert vorhanden ist. Ist dieser vorhanden, wird die ganze Reihe durch den Befehl „conn.Execute($"DELETE FROM {nameof(Db_Customer)} {where}");“ gelöscht. Der Befehl beinhaltet folgende Informationen: es sollen Daten aus der Tabelle DB_Customer gelöscht werden und „where“, welche Reihe gelöscht werden soll. Nachdem die Reihe gelöscht ist, kann die neue Reihe mit den korrigierten oder neuen Werten eingefügt werden. Bei der Funktion, eine Reihe zu löschen, wird die gleiche Methode ausgeführt wie beim Schreiben. Jedoch wird der Befehl „conn.Insert(row);“ nicht ausgeführt. Somit wird nur die Reihe gelöscht und keine neue eingefügt.
Die dritte grundlegende Funktion ist das Lesen aus den Tabellen. Hierfür muss wieder eine Verbindung zu der Datenbank SQ Lite aufgebaut werden. Query heißt die „Lesefunktion“ in C#. Query hat seinen Ursprung aus dem Lateinischen (quaerere) und bedeutetet übersetzt „fragen/suchen“. (centron, 2023) In dieser „query“ muss definiert werden, „wo“ gesucht werden soll. Wenn die ganze Tabelle ausgegeben werden soll, wird „where“ kein Inhalt zugewiesen where=““. Anhand dem definierten“query“ kann die Suchfunktion ausgeführt werden. Ist diese fertiggestellt, wird die Verbindung zur Datenbank wieder getrennt.

```
//Initialisierung verbindung
SQLiteConnection conn = null;

try
{
    //Aufbau Verbindung mit Datenbank
    conn = new SQLiteConnection(dataSource);
    //Zusammensetzung des Querys für die Suche
    //Name der Tabelle in Query aufnehmen
    string query = $"SELECT * FROM {nameof(Db_Customer)} ";
    if (where != "")
    {
        //Suchoption
        query += "WHERE " + where;
    }
    //Suchen anhand des gegebenen Parameters
    return (conn.Query<Db_Customer>(query), null);
}
catch (Exception ex)
{
    return (new List<Db_Customer>(), new Error(ex.ToString()));
}
finally
{
    conn?.Close();
}
```

Wenn dem „query“ ein zu suchender Wert zugewiesen werden soll, wird dies über eine weiter Methode „Readwith“ getan, zum Beispiel der „ReadwithID“ Methode. In dieser Methode wird „where“ den Wert der zu suchenden ID zugewiesen. Nachdem der Wert zugewiesen wurde, springt das Programm mit dem „return“ Befehl zu der Methode „Read“ und führt diese aus. In dieser wird dann „query“ den Wert von „where“ zugewiesen.

```
public static (List<Db_Customer>, Error) ReadwithID(string ID,
string dataSource)
{
    var where = $"ID='{ID}'";
    return Read(where, dataSource);
}
```

Der Anwender kann zum Beispiel einen neuen Kunden anlegen mit dem Button „+Add New Customer“. Daraufhin öffnet sich ein Popup Fenster, in dem alle relevanten Informationen zu dem Kunden eingetragen werden. Sind nicht alle Felder ausgefüllt, ploppt eine Fehlermeldung auf, die den Anwender auffordert die restlichen Daten abzulegen. Das Design ist immer dasselbe zum Erstellen von neuen Aufträgen, Kunden, Ressourcen oder Bestellungen. Somit findet sich der Anwender schnell in der Handhabung zurecht.

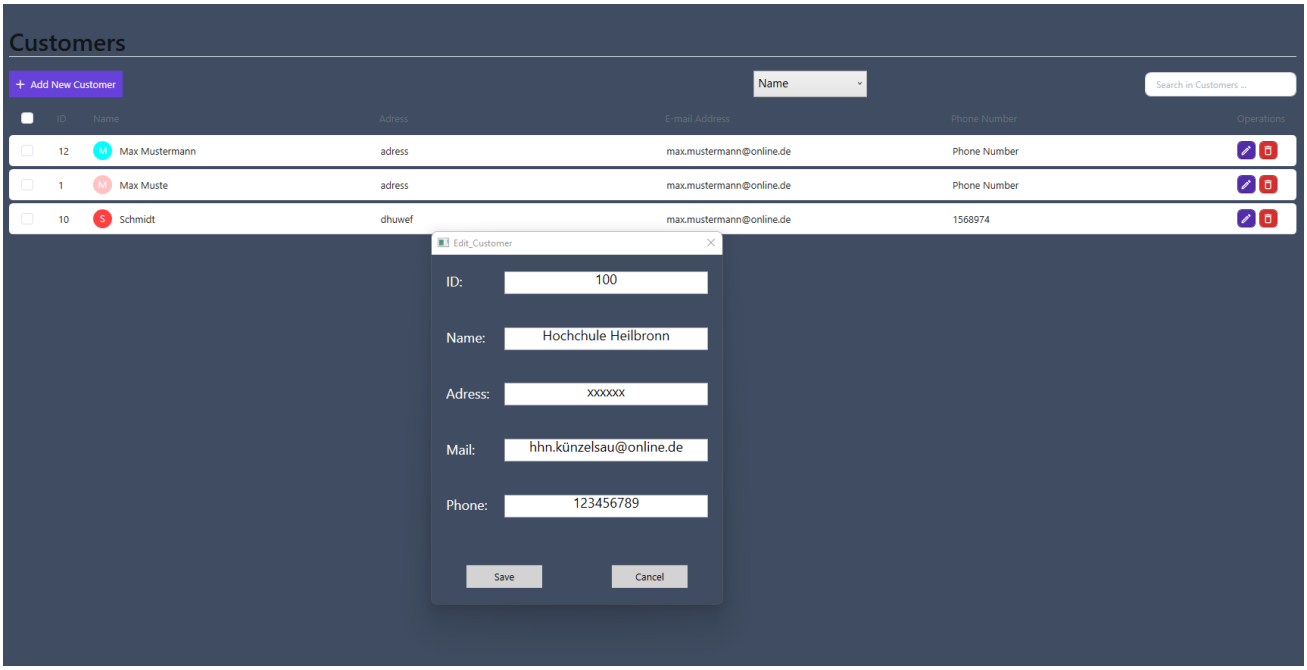

Abbildung 3: Tabellenansicht WPF

Benutzerverwaltung

Die Benutzerverwaltung spielt bei der Timetracking App eine wichtige Rolle. Für jede Aktion in der Anwendung ist eine Anmeldung notwendig. Erfolgt diese nicht, erscheint eine Fehlermeldung und fordert den Benutzer, auf sich anzumelden.

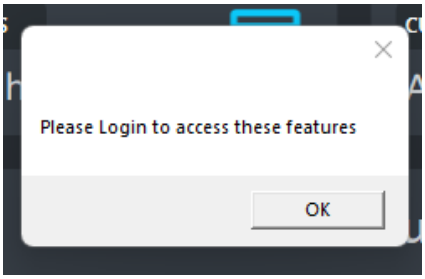

Abbildung 4: Meldungsfenster

Durch eine konsequente Anmeldung ist immer gewährleistet, dass nur die Funktionen, die für den Benutzer bestimmt sind, ausgeführt werden können. Nur ein Administrator kann einen neuen Benutzer anlegen oder löschen. Für das Anlegen wird dem neuen Benutzer eine ID und eine Rolle zugeteilt. Des Weiteren werden Benutzern auch deren „Rechte“ in der App zugewiesen: „Read only Permission“, „Write Permission“ und „Full Acess“.

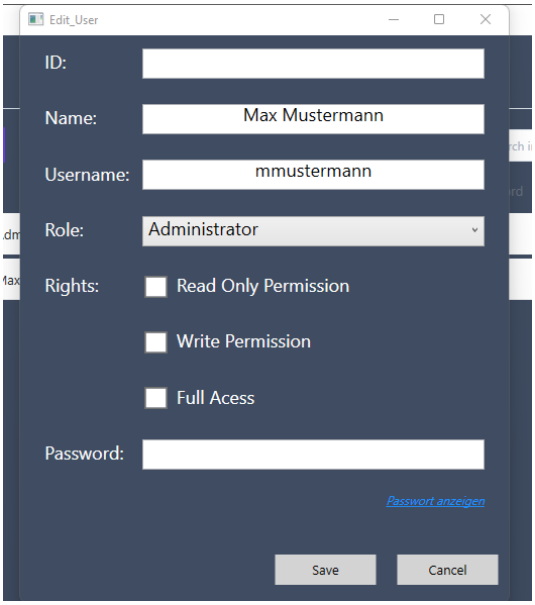

Abbildung 5: Benutzer einfügen

Die dem Benutzer zugeordnete ID wird als eindeutiges Erkennungsmerkmal genutzt und ist im ganzen Programm einmalig. Es kann jeder Benutzer gelöscht werden. Daher könnte der Fehlerfall auftreten, dass alle Benutzer gelöscht wurden und kein Benutzer mehr vorhanden ist. Wäre dies der Fall, könnte das Programm nicht mehr

genutzt werden, da kein Administrator vorhanden wäre, um einen neuen Benutzer zu erstellen. Ist im Programm kein Administrator mehr vorhanden, wird automatisch ein Standardnutzer mit den Standardanmeldedaten erstellt (Benutzername: „Admin“ und Passwort: „root“).

Socket Verbindung

Allgemeines

Die Verbindung zwischen der mobilen Handy-App und der „Master-Software“ auf dem Desktop Rechner wurde über eine Socket Verbindung realisiert.
Eine Socket Verbindung bietet die Möglichkeit, dass zwei Geräte, die sich im selben Netzwerk befinden, miteinander kommunizieren können. Über diese Verbindung können dann Datenpakete zwischen beiden Teilnehmern hin und her geschickt werden.
Eine Socket Verbindung ist üblicherweise wie folgt aufgebaut. Sie besteht aus zwei Teilnehmern, wobei einer als Server und der andere als Client fungiert. In unserem Fall übernimmt die Desktopanwendung die Aufgabe als Server. Hierfür wird zunächst ein TCP/IP-Socket geöffnet. Dieser hat eine spezielle IP, je nachdem auf welchem Gerät die Anwendung läuft, und eine Portnummer. Dieser erstellte TCP Listener wartet dann unter dieser Portnummer so lange, bis ein Client die Anfrage auf Verbindung stellt.

```
// Erstelle einen TCP/IP-Socket
TcpListener listener = new TcpListener(ipAddress, Port);
listener.Start();
// Warte auf eine eingehende Verbindung
TcpClient client = await listener.AcceptTcpClientAsync();
```

Die mobile App ist der Client, der eine Verbindung mit dem Server aufbauen möchte. Hierfür wird die Server IP-Adresse, sowie die Portnummer benötigt. Nachdem der Server die Anfrage des Clients akzeptiert hat, können Datenpakete ausgetauscht werden, auf die der Server antworten kann.

```
// Erstelle eine TCP/IP-Verbindung
TcpClient client = new TcpClient();
client.Connect(serverIpAddress, Port);
```

Innerhalb der Socket Verbindung können nur bestimmt Datentypen übertragen werden. Die zu übertragenden Daten müssen daher vor Versenden über den Socket serialisiert werden. Dies wird über den JsonSerializer realisiert. Dieser wandelt die zu versendende Nachricht in einen JSON-String um, der anschließend versendet werden kann.

```
string jsonString = JsonSerializer.Serialize(message);
```

Funktion im Programm

Da bei erstmaliger Installation der App auf einem mobilen Endgerät noch keine Daten, weder die Anmelde- noch die Arbeitsdaten, vorhanden sind, muss zunächst über einen Einmalcode die User Datenbank, sowie die Tasks Datenbank übermittelt werden. Dieser Einmalcode kann in der Desktop Anwendung generiert werden und muss anschließend über die Socket Verbindung an den Server geschickt werden. Ist dieser Code gültig, werden die Datenbanken vom Server versendet.

Abbildung 6: Einmalcode der Desktopanwendung

Zur anschließenden Anmeldung wird der im Login-Fenster eingegebene User an den Server geschickt und in der dort bestehenden Datenbank gesucht. Gibt es den User dort so antwortet der Server mit einer Freigabe, auf welche hin der Client Daten senden kann.

Diese Daten sind beispielsweise dazu da, um die über die App geschriebenen Zeiten an den Server zu senden. Hierfür wird die Datenbank der App, in der die Zeiten gespeichert werden, eingelesen, über den Serializer in einen JSON-String umgewandelt und anschließend über den Socket geschickt. Der Server erhält die

Nachricht deserialisiert diese und speichert die Zeiten anschließend ab. Die in der Datenbank bis dahin neu angelegten Tasks, werden dann im zweiten Schritt wieder an die App übertragen.

PDF-Rechnungserstellung

Als abschließender Prozess eines Auftrages steht die Rechnungserstellung. In diesem Abschnitt soll eine fertige Rechnungsvorlage mit den gesammelten Stunden und Preisen gefüllt werden. Für das Erstellen und Editieren von PDF- Dokumenten stellt die .NET Bibliothek das NuGet „PDF Sharp“ zur Verfügung. Hierfür muss ein Ablageort für die Rechnung definiert werden und ein Pfad, wo das Programm die Vorlage findet.

```
//einlesen des Ausgabepfads
var (data, err) = Rw_Settings.ReadwithID("1", Paths.sqlite_path);
if (err != null)
{
    MessageBox.Show(err.GetException().Message);
}
else if (data.Count == 0)
{
    MessageBox.Show("No Directory for Pdf Files set");
}
//wenn Eintrag vorhanden und Pfad existiert dann starten
else if (data.Count == 1 && Directory.Exists(data[0].Ressource))
{
    //Vorlagenpdf Pfad einlesen
    var (pdfmodel, err1) = Rw_Settings.ReadwithID("2",
Paths.sqlite_path);
    if (err1 != null)
    {
        MessageBox.Show(err1.GetException().Message);
    }
    else if (pdfmodel.Count == 0)
    {
        MessageBox.Show("No File for Pdf presentation set");
    }
```

Da die Vorlage nicht verändert werden soll, wird diese geöffnet und eine Kopie erstellt. Die Kopie wird unter der Variable „outputDocument“ gespeichert.

```
//Dokument öffnen und kopie erstellen
PdfDocument document = PdfReader.Open(pdfmodel[0].Ressource,
    PdfDocumentOpenMode.Import);
PdfDocument outputDocument = new PdfDocument();
```

Dem Programm ist bei dem neuen Dokument nicht bekannt, wie viele Seiten es hat. Daher muss im nächsten Schritt die Seitenanzahl aufgenommen werden, um später jede Seite genau ansprechen zu können.

```
int count = Math.Max(document.PageCount, document.PageCount);
for (int idx = 0; idx < count; idx++)
{
    // Get page from 1st document
    PdfPage page1 = document.PageCount > idx ? document.Pages[idx]
: new PdfPage();
    page1 = outputDocument.AddPage(page1);
```

Wie bei anderen Office Produkten bekannt, muss für einen Text eine Schriftart und Größe definiert werden.

```
// Create a font
XFont font = new XFont("Verdana", 10);
XFont fontgroß = new XFont("Verdana", 20);
```

Damit die Vorlage mit Daten gefüllt werden kann, müssen die Daten aus der Datenbank an lokale Variablen übergeben werden. Dies hat den Vorteil, dass kein großer Aufwand nötig ist, um verschiedene Werte in der Rechnung auszugeben. Bei dem Rechnungsdatum wird die DateTime Funktion genutzt. Somit wird immer das aktuelle Datum ausgegeben, an dem die Rechnung erstellt wurde. Im Gegensatz hierzu steht das „Lieferdatum“. Dies wird beim Fertigmelden des Auftrags gespeichert. Dies erfolgt über eine Eingabe des Operators.

```
string Position = "Position";
string Menge = "Menge";
string Einzelpreis = "Einzelpreis";
string Positionpreis = "Positionpreis";
string Gesamtpreis = "Gesamtpreis";
string Rechnungsnummer = "Rechnungsnummer";
```

```
string Lieferdatum = deliverydate;
string Rechnungsdatum = DateTime.Now.ToString("dd.MM.yyyy");
string[] Adresse = { "Name", "Straße", "PLZ+Ort" };
int pos_Anzahl = 13;
float Preis = 0;
```

Positionen wie zum Beispiel „Menge“, „Position“, „Einzelpreis“ und so weiter sind jeweils abhängig von ihrem „Task“. Daher müssen diese in einer Rechnung öfters beschrieben werden. Hierzu werden alle Werte über eine Liste eingelesen. Gibt es keine Position, wird eine Fehlermeldung ausgegeben. Werden mehr als eine Position gezählt, werden die Werte an die jeweilige Variable weitergegeben. Pro Position wird in der Schleife „i“+1 addiert. Die jeweilige Position kommt immer in Zeile „i“. Somit füllt sich die Tabelle mit jeder Position weiter. Auf einer Rechnung ist Platz für 13 Positionen. Soll dies in der Zukunft erweitert werden, muss eine Rechnungsvorlage mit 2 Seiten erstellt werden.

```
List<Db_Ressources> ressource = new List<Db_Ressources>();
//für jeden Task eine Position anlegen
for (int i = 0; i < pos_Anzahl; i++)
{
    if (orderID != "" && i < tasklist.Count)
    {
        (ressource, var err3) =
            Rw_Ressources.ReadwithID(tasklist[i].Ressource,
Paths.sqlite_path);
        if (err3 != null)
        {
            MessageBox.Show(err3.GetException().Message);
        }
        if (ressource.Count == 1)
        {
            Position = ressource[0].Name;
            Menge = tasklist[i].ActualHours.ToString();
            Einzelpreis = ressource[0].Costs.ToString();
            Positionpreis = (tasklist[i].ActualHours *
ressource[0].Costs).ToString();
            Preis += (tasklist[i].ActualHours *
ressource[0].Costs);
        }
    }
```

Die einzelnen Variablen werden immer mit der gleichen Methode in die PDF geschrieben. In diesem Schritt wird auch die Position auf der Seite definiert, auf der die Variable geschrieben werden soll. Hierbei ist die Y-Koordinate absolut angegeben und die X-Koordinate relativ. Dadurch passt sich die X-Koordinate der Seitengröße an. Der Zeilenabstand verändert sich mit der Seitengröße nicht, daher wird diese auch absolut angegeben. Die Y-Koordinate berechnet sich aus einem absoluten Wert plus den Zeilenabstand mal „i“, „i“ wird wieder durch die If-Schleife definiert. Mit dem Code „XStringFormats.BottomLeft“ wird definiert, wo sich der absolute Nullpunkt der Seite befindet.

```
// Position
gfx.DrawString(Position, font, XBrushes.Blue,
    new XRect(page1.Width / 12, -490 + (i * 15), page1.Width,
page1.Height),
    XStringFormats.BottomLeft);
```

Alle weiteren Informationen werden mit derselben Methode in das PDF-Dokument eingetragen. Nachdem alle Wert in das PDF-Dokument übertragen wurden, wird noch das endgültige Dokument abgespeichert. Hierfür versucht das Programm, das Dokument unter dem endgültigen Dokumentennamen unter dem hinterlegten Pfad abzuspeichern. Ist dies nicht möglich, wird wieder eine Fehlermeldung ausgegeben. Außerdem wird das Dokument direkt dem Anwender geöffnet und wiedergegeben.

```
try
{
    if (dboutputPath == "")
    {
        outputDocument.Save(data[0].Ressource + "\\" + PDFname +
".pdf");
    }
    else
    {
        outputDocument.Save(dboutputPath + "\\" + PDFname +
".pdf");
```

```
    }
    //PDf als Bytearray zurückgeben
    using (var ms = new MemoryStream())
    {
        outputDocument.Save(ms);
        return ms.ToArray();
    }
}
catch (Exception e)
{
    MessageBox.Show(e.Message);
}
```

Error-handling

Für das Error-handling in C# gibt es im Wesentlichen zwei Vorgaben. Einmal müssen Fehler von Codeblöcken, in denen eine Ausnahme (Exception) auftreten könnte, abgefangen werden, sodass das Programm weiter ausgeführt wird. Dasselbe gilt für den Inhalt von Variablen/Objekten, wie z.B. Arrays. Diese müssen immer auf Inhalt geprüft werden, bevor diese abgefragt werden können, um verhindern zu können, dass das Programm in einem nicht vorhandenen Index nach Inhalten sucht. Im Projekt wurden für das Abfangen von Fehlern drei verschiedene Vorgehensweisen verwendet.

Exception-Handling über Klasse:

Für die Rückgabe eines Wertes kann die Klasse Error direkt in einer Methode eingebunden werden. Die Klasse sieht wie folgt aus:

```
public class Error
{
    private readonly System.Exception _ex;

    /// <summary>
    /// Neuer Error aus string generieren
    /// </summary>
    /// <param name="err"></param>
    public Error(string err)
    {
```

```
            _ex = new System.InvalidOperationException(err);
        }

/// <summary>
/// Error als Exception zurueckgeben
/// </summary>
/// <returns></returns>
public System.Exception GetException()
{
    return _ex;
}
```

Aufgerufen und ausgewertet werden Ausnahmen wie folgt:

Die Methode mit Anwendung der Errorklasse:

```
public static Error Write(List<Db_Tasks> rows, string dataSource)
```

Wird die Methode nun verwendet, gibt diese bei einem Error den Fehlertext zurück. Zur Auswertung muss der Rückgabewert in eine Variable kopiert und abgefragt werden. Enthält diese einen Wert, bedeutet das, dass ein Fehler aufgetreten ist. Der Fehler wird in der Oberfläche über eine Messagebox ausgegeben.

```
var err = Rw_Tasks.Write(new List<Db_Tasks> { data },
Paths.sqlite_path);
if (err != null)
{
    MessageBox.Show(err.GetException().Message);
}
```

Exception-Handling mit try-catch:

Das klassische Error-handling in C# erfolgt über einen try-catch Block. Hierbei wird der Code, der eine Ausnahme liefern könnte, mit einem try-Block umschlossen. Taucht ein Fehler auf wird der catch-Block aufgerufen und der Fehler wird in der Oberfläche ausgegeben.

```
try
{
    actualhours =
float.Parse(Actual_Field.Text,CultureInfo.InvariantCulture.NumberFo
rmat);
```

```
    estimatehours =
float.Parse(Estimate_Field.Text,CultureInfo.InvariantCulture.Number
Format);
    costs =
float.Parse(Costs_Field.Text,CultureInfo.InvariantCulture.NumberFor
mat);
}
catch (Exception exception)
{
    MessageBox.Show(exception.Message);
}
```

Überprüfen von Inhalten:

Beim Überprüfen von Inhalten kann entweder über eine IF-Funktionalität oder bei Arrays mit einer For-Schleife der Inhalt validiert werden. Beim IF wird hierbei abgefragt, ob der Wert ungleich null etc. ist, sodass nur dann der gewünschte Code ausgeführt werden kann. Falls ein Array keinen Inhalt besitzt, kann durch eine For-Schleife gewährleistet werden, dass der Code wirklich nur dann ausgeführt wird, wenn Inhalt vorhanden ist.

Grundsätzlich gibt es noch andere Möglichkeiten, Fehler vorzubeugen oder auszuwerten. Die oben genannten sind aber die am häufigsten verwendeten.

Oberfläche App

In Abbildung 7 kann man die Benutzeroberfläche der mobilen App für Android Geräte sehen. In der oberen linken Ecke befindet sich der Anmelde-Button. Über diesen kann ein Benutzer angemeldet werden (siehe Abbildung 8), um die Funktionen der App nutzen zu können. Für alle Funktionen außer „Connecten“ und „Settings“ wird eine Anmeldung benötigt.

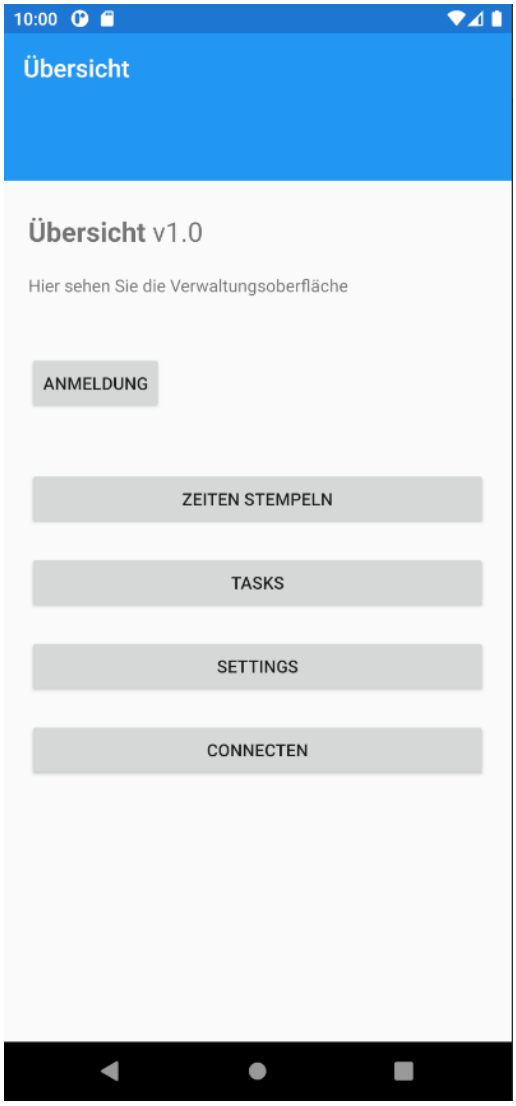

Abbildung 7: Homebildschirm App

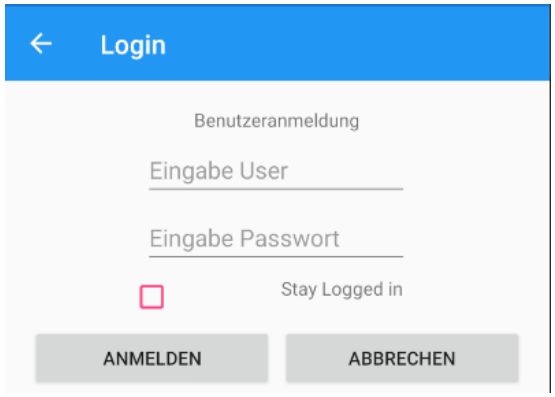

Abbildung 8: Login Oberfläche

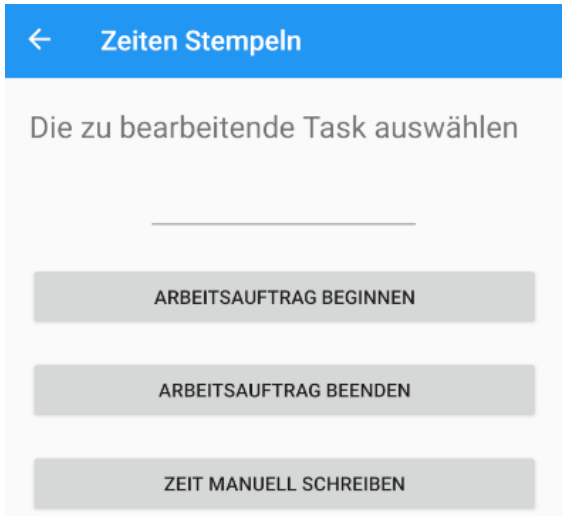

Abbildung 9: Zeiten Stempeln

Zudem gibt es weitere Buttons auf der Übersichtsseite, diese wären „Zeiten Stempeln“, „Tasks“, „Settings“ und „Connecten“.

Im Untermenü „Zeiten stempeln“ kann aus der Taskliste ein bestimmter Arbeitsauftrag ausgewählt und mit einem Klick auf „Arbeitsauftrag beginnen“ die Zeiterfassung gestartet werden. Hat man die zu erledigenden Arbeiten abgeschlossen, kann man die Zeit wieder gestoppt werden und die bis dahin

vergangene Zeit wird dem ausgewählten Arbeitsauftrag zugeschrieben und in der Datenbank gespeichert.
Möchte man unabhängig davon Zeiten nachtragen, weil man beispielsweise vergessen hat, die Zeit zu erfassen, kann über die Schaltfläche „Zeit manuell schreiben“, händisch eine Zeit nachgetragen werden. Hierfür muss wie zuvor auch, zuerst eine Task ausgewählt werden und anschließend die gewünschte Zeit eingetragen werden. Diese Funktion steht jedoch nur Benutzern mit Admin Rechten zur Verfügung.
Schaut man in der Übersicht auf die nächste Schaltfläche, so kommt man mit einem Klick auf „Tasks“ in eine Übersicht, in der man die aktuell anstehenden Arbeitsaufträge mit deren Beschreibung, deren eingeplanten Stunden und deren bis zu diesem Zeitpunkt tatsächlich erfolgten Arbeitsstunden sehen kann.
In den Settings können verschiedene Einstellungen getroffen werden, welche der Bedienung der App dienen.
Im Fenster „Connecten“ kann, der in der Desktopanwendung generierte Einmalcode eingegeben werden, um die App bei erstmaligem Start mit dem Server zu verbinden, sowie die User- und Task Datenbank herunterzuladen.

5. Ausblick

Der aktuelle Entwicklungsstand der App bietet ein gutes Fundament, um darauf aufbauen zu können. Hierbei umfasst die App gerade die Funktionen des Stempelns von Arbeitszeiten auf eine „Task“, das Erstellen von Kunden, Aufträgen, Tasks, Ressourcen und vieles mehr. In Zukunft kann sich die App weiter in Richtung Controlling und Human Ressource Management entwickeln. Im Bereich des Controllings kann die Angebotserstellung und Kostenkalkulation ergänzt werden. Dies würde den Unternehmer unterstützten und würde Fehlkalkulationen reduzieren. Eine weitere interessante Erweiterung wäre die Lohnabrechnung. Da die Arbeitszeit pro Mitarbeiter für jeden Auftrag sowieso erfasst werden, können diese Daten direkt für die Lohnabrechnung genutzt werden. Ein weiterer wichtiger Schritt wäre die Vermarktung der Software. Bevor die Software vermarktet werden kann, muss diese jedoch noch qualifiziert werden. Hierfür müssten sämtliche Grenzen und Fehler ausgetestet werden, damit noch vorhandene Bugs behoben werden können. Ebenfalls müssten ein Lizenzierungsprinzip und eine Vertriebsplattform gewählt und erarbeitet werden.
Nach diesem Projekt kann die Aussage getroffen werden, dass die Entwicklung von Anwendungen einen äußerst komplexen Vorgang darstellt. Kleine Funktionalitäten sind zwar schnell erarbeitet, jedoch äußerst komplex in der Gesamteinbindung. Durch gutes Error-handling, Oberflächendesign und viele Kleinigkeiten im Backend gestaltet es sich oftmals langwieriger als gedacht, diese angemessen in das Projekt einzubinden.

6. Abkürzungsverzeichnis

GUI Graphical User Interface
HTML Hypertext Markup Language
ID Identifier
IDE Integrated Development Environment
IP Internet Protocol
JS Javascript
JSON Java Script Object Notation
PC Personal Computer
PDF Portable Document Format
TCP Transmission Control Protocol
XAML Extensible Application Markup Language

7. Abbildungsverzeichnis

8. Literaturverzeichnis

centron. (18. 7 2023). Von https://www.centron.de/glossary/query/#:~:text=Der%20Begriff%20Query%20beschreibt%20die,%E2%80%9C%20oder%20%E2%80%9Esuchen%E2%80%9C%20bedeutet. abgerufen

IT-Vision. (25. 7 2023). Von Windows Presentation Foundation (WPF) - Begriffserklärung im Entwickler-Lexikon/Glossar auf www.IT-Visions.de abgerufen

it-visions. (25. Juli 2023). Von https://www.it-visions.de/glossar/alle/3718/Windows_Presentation_Foundation.aspx abgerufen

Microsoft. (25. 7 2023). *Microsoft .NET*. Von https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm abgerufen

sos-Software. (25. Juli 2023). Von https://sos-software.com/jetbrains-rider-eine-alternative-zu-visual-studio/#:~:text=4%20Gr%C3%BCnde%20wieso%20JetBrains%20Rider%20besser%20ist%20als%20Visual%20Studio%3A&text=Somit%20kann%20mit%20Rider%20schneller,denselben%20Funktionen%20zu%20finden%20is abgerufen