Building an extensible framework for automatic employee time logging using physical markers

Master Thesis

Aleksei Kosozhikhin

Thesis supervisors:
Prof. Dr. Jacques PASQUIER-ROCHA
Pascal GREMAUD

Philosophisch-naturwissenschaftliche Fakultät
der Universität Bern

July 2018

Abstract

Tracking of employee hours historically has been a cornerstone metric for determining the hired force's pay. In the modern times it is becoming increasingly important to provide precision and automation to this process. The current state of technology includes several methods of location and proximity detection, allowing us to use physical markers for automatic employee time tracking.

In this thesis we construct a concept of an abstract extensible framework for manual and automatic employee hours tracking by covering the whole time tracking workflow. We also build an implementation of the proposed concept suited for all participants of the employee hours tracking process and using multiple tracking technologies.

Keywords: Bluetooth beacons, QR Codes, RESTful API, Xamarin, ASP.NET Core

Acknowledgements

I want to thank my colleagues at Cure-IT AG and especially Christian Wienholz for providing resources and immeasurable help on this project.

I also want to thank Pascal Gremaud for supervising my project and guiding me.

Last but not least, I want to thank my girlfriend Lisa for being my guiding star in this life and always helping me to keep my goals in front of me.

Table of Contents

List of Figures

1 Introduction

1.1. Motivation

Employee hours tracking as an important worker's productivity metric appeared as early as the nineteenth century[1]. Classic examples of the tools used to track worker's time are punch cards and timesheets. Precision and automation in this process become increasingly important, especially with employees who are being paid by the hour, such as freelancers and consultants.

Disadvantages of classic manual time logging include:

- **Human error**
  Employees inaccurately enter their time, making HR departments spend a lot of time fixing these errors.
- **Time theft**
  Employees might intentionally enter incorrect time or punch in for their absent colleagues.
- **Time consuming process**
  Manual time logging takes much more time than expected, which might lead to adding time records later in bulk, causing even more lost time or additional errors.

The American Payroll Association estimated that automation reduces costs related to payroll management by up to 80%[7], which is influenced by reducing the aforementioned risks. Modern technology provides several methods of tracking employee working hours. However, tying yourself to a single technology might not be the best decision, as we show in the following chapters.

1.1.1. Goals

Following the necessity of modernization and the current state of technology, we formulate the goals of this work as follows:

- Analyze existing processes and solutions.
- Propose a concept for an extensible and customizable framework, encompassing the entire workflow of employee hours tracking.
- Implement a running prototype of proposed concept.

1.2. Organization

- **Introduction**

  The introduction describes the motivation and goals of this work, gives an overview of the following chapters and provides a description of the formatting conventions used by the author.
- **Proximity and location**

  This chapter introduces several popular solutions for proximity and location tracking along with their pros and cons.
- **Existing solutions**

  The second chapter provides a critical review of the existing employee time tracking solutions.
- **Framework proposal**

  Here we reiterate over requirements for our system and propose a framework architecture for further implementation.
- **Implementation**

  This chapter describes our motivational implementation of the proposed framework, describing used technologies and libraries.
- **Future work**

  In this chapter we explain our ideas on improvement of the framework concept as well as the implementation project.
- **Conclusion**

  Finally, in conclusion we summarize our achievements made as part of this thesis.
- **Appendix**

  Contains extracts of artifacts or service messages, abbreviations and references used throughout this work.

1.3. Notations and Conventions

- Formatting conventions:
  - Abbreviations and acronyms as follows Uniform Resource Locator (URL) for the first usage and URL for any further usage;

  - `http://localhost:9090/api` is used for web addresses;
  - Code is formatted as follows:

```
public double division(int _x, int _y) {
    double result;
    result = _x / _y;
    return result;
}
```

- The work is divided into six chapters that are formatted in sections and subsections. Every section or subsection is organized into paragraphs, signalling logical breaks.
- Figures, Tables and Listings are numbered inside a chapter. For example, a reference to Figure j of Chapter i will be noted *Figure i.j*.
- As far as gender is concerned, I systematically select the masculine form due to simplicity. Both genders are meant equally.

2
Proximity and location

2.1. Introduction

The problem of logging employee hours can be reduced to the problem of attributing the worker to his workplace. This means that as long as the worker is at his workplace (e.g. his computer, a conference room or another designated location), the system should count the time towards his working hours. This prompts us to use different *proximity* and *location* technologies to connect the worker to his workplace. The former allows a device to provide a reaction (e.g. start logging working time) based on a distance to a specific physical marker, while the latter determines an absolute location, which can be compared then to a known location of a workplace to initiate time logging. This section will give a short overview of the existing technologies and analyze their advantages and disadvantages.

2.2. Technology overview

2.2.1. QR codes and barcodes

Barcode is an optical, machine-readable, representation of data[8]. Since its introduction in 1966 and to this date it remains the most popular solution for Automatic Identification

and Data Capture (AIDC)[9]. Introduced in 1994, QR Code technology became popular outside of industrial use for its faster readability and greater storage capacity, compared to original barcodes. However, 2012 Inc magazine study showed[10], that the technology was unknown to 97% of consumers to the point where a target user wouldn't know how to scan the QR Code or how to find an app able to do that. Another valid point mentioned in the article is that interaction with a QR Code takes too much steps and can hardly get automated. However, it still remains an extremely cheap solution, since it can be simply printed on a surface or displayed from a device's screen. It also requires close proximity (under 1m) to get recognized, which, depending on circumstances, can be viewed as an advantage or a disadvantage.

Fig. 2.1.: Example of a QR Code containing a link to the Software Engineering Group website

2.2.2. Radio-frequency identification

Radio-Frequency Identification (RFID) technology uses radio waves to read and capture information stored on a tag attached to an object[11]. As demonstrated on Figure 2.2, the main disadvantage of using an RFID system is its complexity in terms of the amount of different components: RFID tags, readers, reader control and apps. Near-Field Communication (NFC) builds on top of RFID technology and can be presented as a branch of high-frequency RFID. The unique feature of NFC is the ability to act as both a reader and an NFC tag. NFC is widely used in smartphones: according to an IHS Technology study[12], it will be included in 64% of the phones shipped in 2018, which partly resolves the infrastructure complexity problem of RFID, even though dedicated NFC tags still need to be deployed. Furthermore, the frequency range used by NFC presents a distance limitation (addressed even in the name of the technology), with optimal range of NFC being around 10cm. Advantages of using NFC include security (it was designed with security and range limitations to prevent hacking attempts) and low power consumption (tags operate in passive mode, meaning that they only activate when a reader is present and don't require their own power source).

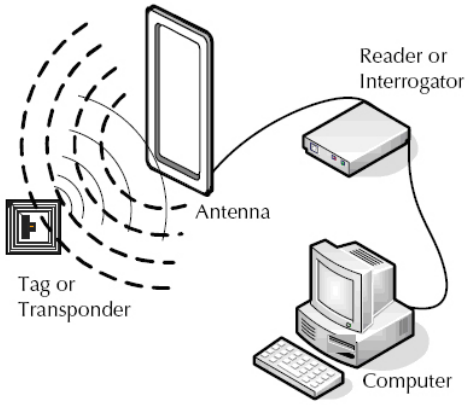

Fig. 2.2.: Typical RFID system

2.2.3. Wi-Fi

Wi-Fi is a popular wireless networking technology, but it can be successfully used for proximity detection[3] as well as location tracking[13]. Due to the complexity of its technology, Wi-Fi location tracking solutions are generally more expensive than RFID with tags priced at around $50 on average and requiring more power consumption. While most of the smartphones are Wi-Fi-enabled and therefore don't require specific tag attached to them, constant usage of Wi-Fi can drain the device's battery significantly and therefore requires compromises. However, Wi-Fi tracking can operate in a logically passive mode, meaning that any Wi-Fi-enabled device periodically sends a handshake-message to all access points in range, which allows the system to track devices based on their MAC addresses and signal power, although this might be considered a privacy issue. Wi-Fi tracking also requires a serious investment into router infrastructure and its subsequent maintenance, which brings the price of the solution even higher.

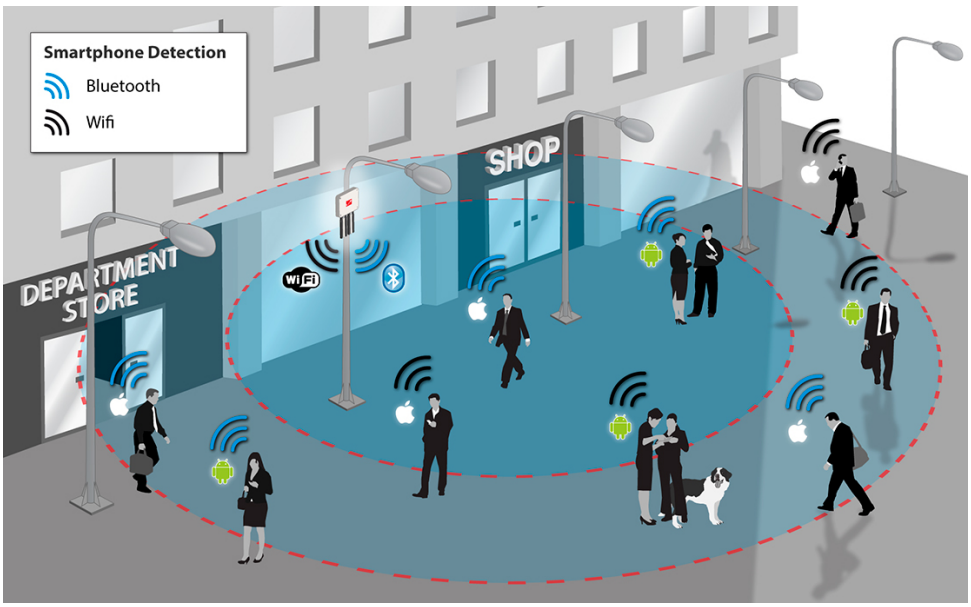

Fig. 2.3.: Combined Wi-Fi and Bluetooth tracking system

2.2.4. Bluetooth beacons

Bluetooth beacons are a class of Bluetooth Low Energy (BLE) devices that periodically broadcast their identifier or other specific message to the nearby devices. By placing such device at a known location, a developer of positioning system might pinpoint the location of a receiving device (such as a smartphone) when it recognizes the broadcasted identifier. Usage of BLE technology allows to improve the beacon lifetime up to 2 years without recharging it and at the same time decrease the size of its battery (e.g. many beacon devices operate on "coin" batteries). Due to wide spread of Bluetooth technology, it is supported on most of the smartphones. Key platform holders invest in their own BLE beacon standards, such as **iBeacon** by Apple and **Eddystone** by Google. Besides trivial proximity detection, Bluetooth beacons can be used for indoor positioning, as proposed by Rida *et al*[5]. Different standards allow to send different payload with every *frame* of a beacon broadcast, e.g. a Uniform Resource Locator (URL) or a Universally Unique Identifier (UUID).

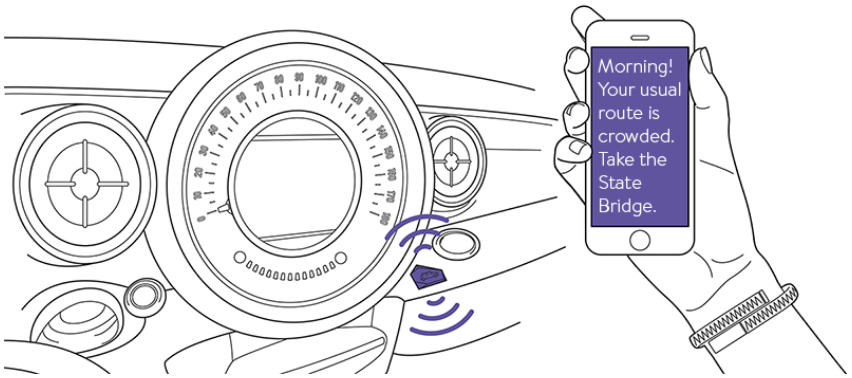

Fig. 2.4.: *Estimote* sticker beacon

2.2.5. Global Positioning System

Global Positioning System (GPS) is a global navigation satellite system that provides geolocation and time information to any receiver with unobstructed line of sight to four or more GPS satellites. *Geofencing* uses location data provided by GPS to match it against a virtual perimeter assigned to a real-world area. While the GPS technology is widespread among modern smartphones, its constant usage results in major power drain, and indoor location becomes increasingly inaccurate due to the satellite signal being obstructed by the building's walls. Thus, using the GPS technology for geofencing would require some implementation compromises, such as reduced activation frequency and wide area definition, which would affect precision.

2.2.6. Biometrics authentication

Biometrics describe human body's characteristics, allowing computer systems to identify a person by an image of his face, audio of his voice or other measures [2]. Biometric identifiers are unique to individuals, thus making them more reliable in identification

Fig. 2.5.: Example of geofencing areas

compared to other technical means, especially when using multiple modalities at once. However, using biometric data often causes privacy and discrimination concerns by providing information about a person without his consent[4]. Nevertheless, use of biometrics authentication in attendance tracking is predicted to grow in the next years[21].

2.3. Conclusion

State of the art technology provides many solutions to the proximity and location problem, differing in their parameters. Some technologies guarantee close proximity (QR Codes, NFC) while others work in big or arbitrary range (Bluetooth beacons, GPS). Another important charateristic is battery use, which limits the applicability of several technologies, like Wi-Fi and GPS. This diversity allows for great customization for individual customer's needs, based on price, maintainability, complexity, reliance and other parameters.

3
Existing solutions

3.1. Introduction

Due to the high demand for a modern solution of the time tracking problem, several products were developed to try to meet that demand. In this chapter we provide an overview of a few popular solutions and try to analyze their advantages and disadvantages.

3.2. Timedock

`https://timedock.com/`

Timedock provides a hosted solution connected to wall-mounted NFC card readers or special apps installed on work supervisors' mobile devices (Android and iOS). Every worker gets equipped with an ID card containing an NFC chip and a QR Code that can be used interchangeably for "badging" in or out. Timesheet data can be accessed in a hosted Web portal through a simplistic interface. Any additional reporting has to be done through external tools with data exported in Comma-Separated Values (CSV) format. Pricing model is another downside — customer has to pay a fixed rate per employee, which becomes complicated and costly.

Fig. 3.1.: Timedock punch clock machine

3.3. TimeTac

`https://www.timetac.com/en/`

TimeTac covers employee personal working time tracking, time tracking per project and absence management. Time logging is done through a web portal or mobile app manually. Alternatively, **TimeTac** provides wall-mounted terminals for manual and NFC-based logging, as well as NFC stickers for location-based logging through mobile app. Detailed timesheets and calendars are available for manager personnel through the same web portal. However, it provides an outdated user experience with all of its featured crammed on the same portal and sometimes the same page. Furthermore, as in **Timedock**'s example, pay-per-user subscription becomes bloated and unmanageable for bigger companies.

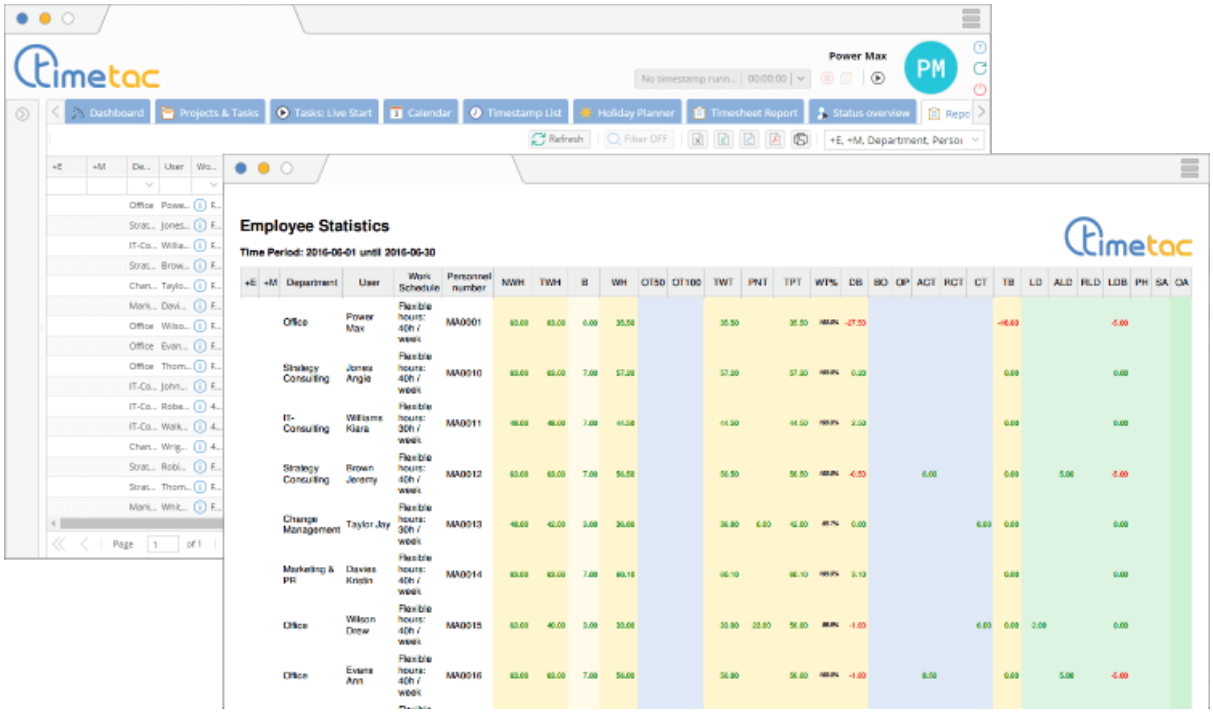

Fig. 3.2.: TimeTac web portal

3.4. Calamari

`https://calamari.io/`

Calamari offers a plethora of integrations and solutions for employee time tracking, from manual time logging through a mobile app to automatic tracking with BLE Beacons and QR Codes. Software integrations are also possible, with several pre-existing ones like **Slack**[1] or **Toggl**[2] and ability to create customized integrations through their documented Application Programming Interface (API). However, it doesn't go beyond simple "check in"/"check out" workflow and doesn't allow any metadata with time logs on top of user's name. On the other hand, Calamari provides a more flexible tier-based pricing scheme.

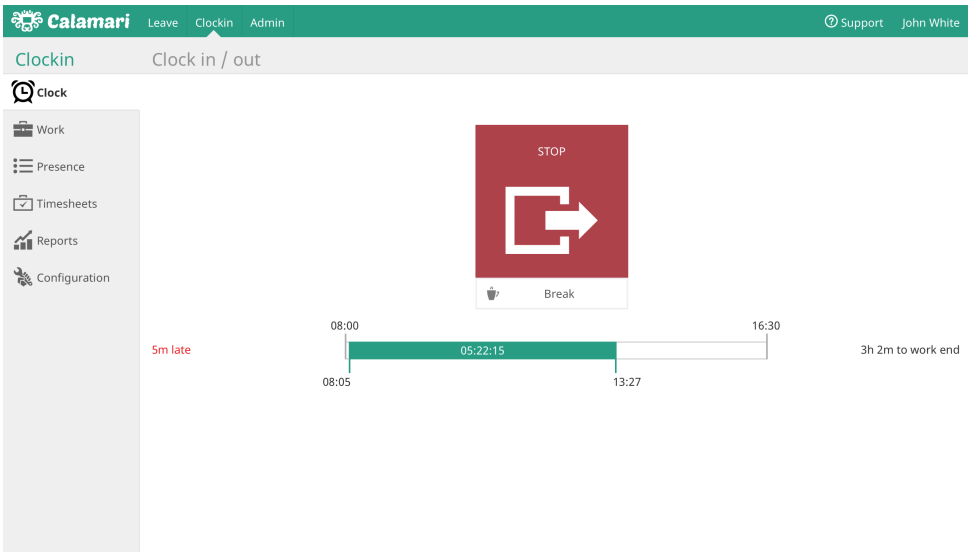

Fig. 3.3.: Calamari "Clock In" through web portal

3.5. TimeCamp

`https://www.timecamp.com/`

TimeCamp provides manual time logging with project specification, as well as automatic project tracking based on user's activity. As with most of the solutions, it gives pay-per-user subscription options and hosted solution for bigger companies (as opposed to cloud service in the basic package). It also offers many app and service integrations, e.g. **GitHub** and **Atlassian Jira**. However, its activity tracking technology raises privacy concerns: it determines what the user has spent his time on by tracking automatically in the background the running processes and browser navigation history, as well as taking screenshots of the desktop.

[1] A popular cloud-based set of team collaboration tools
[2] A time-tracking application

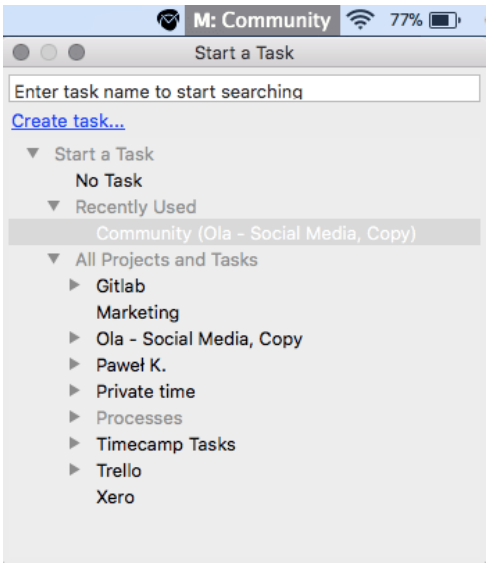

Fig. 3.4.: TimeCamp manual task management

3.6. Conclusion

There are several solutions on the market solving time tracking problem in some way, however they all focus on a specific target audience:

- **Single developers**: those solutions tend to concentrate on many projects that the developer works on in parallel. They often implement a ToDo mechanism as well as some rudimentary billing functionality.
- **External workforce**: the main focus is tracking work on a remote site.
- **Project managers**: focusing on the task management within projects and tracking planned vs spent hours.

4
Framework proposal

4.1. Introduction

Following the state of technology and market research, as well as our personal professional experience with time tracking, we accumulate the features of existing solutions, but design our workflow based on employer/workforce relationship. From the employer side, the main focus will be on tracking time that the employee spent on a project, while from the employee side the expected result is his working time approved before billing the customer or his representative.

4.2. Requirements

According to these goals, we design the framework concept with the following requirements:

- The system has to allow tracking working hours.
- Tracking has to follow the ”badging” logic (i.e. ”badge in” to signify start of work and ”badge out” for end of work).

- The system has to be technology-agnostic, i.e. providing a framework able to encompass many different proximity and location technologies based on the customer's needs.
- Logged time can and has to be attributed with project details.
- Digital process has to include all stakeholders from actual workers to persons responsible for signing off the time records.
- Attempts of forging time records have to be prevented by the system.

4.3. Components

We subsequently propose a framework design based on the following three main components.

4.3.1. Markers

A worker's workplace is associated with a logical marker. Depending on the chosen technology, this logical marker can in turn be represented by a physical marker, i.e. a device providing information to a client, or existing solely as a model of workplace. As an example, a map area corresponding to the company campus could be used for detecting user's attendance through GPS geofencing.

4.3.2. Clients

End-users of the system work with it through its clients. The framework is designed to separate backend from frontend to support any number of different clients for different use cases. A client application has to unload from the backend as much logic pertaining to generation of time records (including worker authentication, marker detection and so forth) as possible before supplying the complete time record. An example of such client would be a software integration to a badging system: when a user unlocks the doors to his office building by using his badge, the software would recognize his identifier and create a time record in the system. The same would be done on user leaving the building, thus creating a pair of "check in" and "check out" time records.

4.3.3. Backend

Framework's backend provides functionality of the system to the clients through a common API. It is completely client-agnostic: the set of features, provided by client to a user, depends only on the client implementation. However, access to the features is limited by role-based authorization policies based on the authenticated user's session.

4.3.4. Authentication

To achieve modularity, the framework is designed to use an external authentication service. It is accessed independently by clients and the backend. It is designed only to

provide authentication, with authorization defined and implemented on the backend for fuller control.

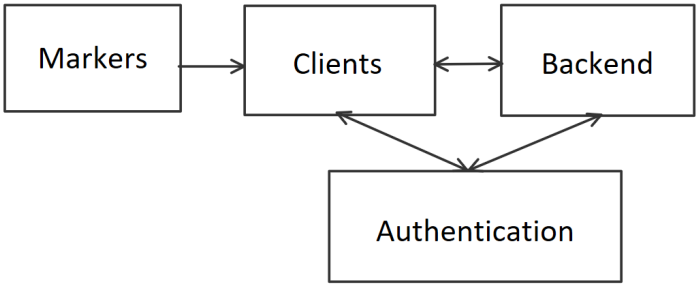

Fig. 4.1.: Framework's main components and data flow

4.4. User types

Following the decision to authorize users' access to the system's features based on their roles, we propose the following roles and their responsibilities:

- **Worker**
  Responsible for time record creation. Can provide additional metadata with a time record, browse his recently created records and fix recent inconsistencies.
- **Time manager**
  Gets an overview of workers and their time records. Assigns projects to workers. On special occasions can act as a proxy for a specific worker to add time records on his behalf. Responsible for overviewing the correctness of time records before sending them for signing off.
- **Signee**
  Signs off on time records for specific worker's month of work and locks them from further changes.
- **User manager**
  Adds new users and deletes them from the system. Manages users' roles. Creates user proxy rules for time managers.
- **System manager**
  Manages markers and projects in the system. Can act as a user manager.

The system allows to combine multiple roles for a single user, which is handy in smaller companies where employees might have more responsibilities.

4.5. Typical workflow scenario

1. System manager adds user manager.
2. System manager adds project and marker.
3. User manager adds time manager, worker and signee.
4. Time manager assigns worker to the project.

5. Worker uses marker to create time records for the project.
6. Time manager checks if time records are valid and sends them to signee.
    a) If time records are invalid, time manager requests proxy permissions from user manager.
    b) User manager grants time manager proxy permissions on the worker.
    c) Time manager adds missing time records and/or removes extra records.
7. Signee checks the provided time sheet and signs off on it.

5 Implementation

5.1. Time-Tracking Tool

The Time-Tracking Tool (TTT) is an implementation of the proposed employee time tracking framework we developed as a commercial product prototype for **Cure-IT AG**. The idea and requirements for TTT stemmed from the consulting work that employees of Cure-IT AG did for other companies. This also helped to specify target audience of the project as IT specialists: official employees, consultants or freelancers. Initial meeting with my supervisor in the company resulted in the list of requirements, architecture outline and other details described in the previous chapter.

5.2. Backend API

The Backend API is the core component of the framework, providing data access and manipulation functionality to the clients. Following the idea of decoupling the client and server components, providing a uniform interface and making the client software handle state on its own, we decided to develop the Backend API as RESTful. It also provides predictable and logical API specification, which helps client software developers.

5.2.1. Choosing a framework

Streamlining the development of the Backend API implementation would require the selected framework to use a common language of development and be well-documented or have a large user-generated know-how database. The selected framework also has to support third-party extensions and have a large database of such extensions. It has to provide built-in authentication and authorization mechanisms or provide easy integration of third-party services. It also has to provide flexibility in hosting solutions and database integration based on customer's needs: from centralized cloud hosting to deployment in customer's infrastructure.

This allows us to formulate the following set of requirements:

- Familiar codebase
- Large developers community
- Extensibility through third-party plugins
- Integrated user authentication and authorization
- Simple deployment
- Database connectors

We selected **ASP.NET Core 2.0** for implementation of the Backend API as it follows all of the requirements. Its main programming language is C#, which takes a 5th spot in TIOBE's programming language popularity index for June 2018[16]. ASP.NET Core is highly extensible through official and unofficial NuGet repositories: NuGet Gallery faces around 150 million downloads a week[17]. It provides a built-in *Identity* mechanism[18], as well as integration of external auth services. It is also one of the fastest Web application platforms, according to TechEmpower's benchmarks[15].

5.2.2. Software pattern

For the implementation of the Backend API we are using the Model-View-Controller (MVC) pattern, with models represented by database entities, views being JSON representations of a relevant Data Transfer Object (DTO) and controllers — the code connecting the two. ASP.NET Core provides very good MVC integration out of the box utilizing file naming conventions, making development very easy. We grouped controllers by the resources they provide, each controller providing functionality from simple set of Create/Read/Update/Delete (CRUD) operations to grouping, filtering and combining different models.

5.2.3. Database

For the database engine we chose *Microsoft SQL Server* with Entity Framework (EF) as an Object-Relational Mapping (ORM) framework on top of it. The choice was made for the sake of the development simplicity, since EF provides handy data annotations in model classes and Migrations mechanism for incremental database schema updates. The solution can be updated to use different other *SQL* database engines through *Database Providers*, while *NoSQL* databases have to be used through external libraries. The simplicity of *SQL* connection allowed us to use a local database for debugging and an *Azure*-hosted one in release.

5.2.4. Entities

In our data scheme we propose the following entities as part of our framework's workflow:

- **Time record** is a basic unit of user's work activity. It signifies either start or end of work, either holding this flag explicitly or deriving it from surrounding time records. Each time record is associated with a worker user, a customer project, a type of time record's source and (optionally) a marker. To follow the main purpose of the framework (tracking working time), time record records a timestamp of its creation.
- **Marker** is a logical representation of a workplace. It contains a displayable name, an identifier in form of a UUID and it is always associated with a customer project. Upon marker's recognition in a client, it is supposed to check against the Backend API to find the marker by its ID. In addition to this, it can store metadata (serialized in the database in form of a JSON object) with client-specific schema.
- **Source type** represents a single type of time record's originating source. It can optionally require a marker to be associated with every time record using the source type. In this prototype we use two source types *without* markers (for regular manual and automatic conditional time record creation) and two *with* markers (QR Code and Beacon).
- **Customer project** represents a work project that a worker user is assigned to. It is used for filtering and categorization of logged time;
- **Project assignment** is a link between a worker user and a project. It always has a starting and ending date. Without a project assignment a worker user cannot create a time record for a project.
- **User** represents any user of the framework, from worker to management personnel.
- **Roles** help the system to authorize users' access to certain features. Every user can be member of one or many roles.
- **User roles** serve to link users and roles in a many-to-many relationship.
- **User proxy** provides time managers temporary rights to add a time record on behalf of a worker user.
- **Sign-off requests** are a final step in time management workflow. They get created when time manager is ready to provide worker's timesheet to responsible signee user. After signee accepts the request, worker user's work gets locked in the request's time period, make final any changes made before.

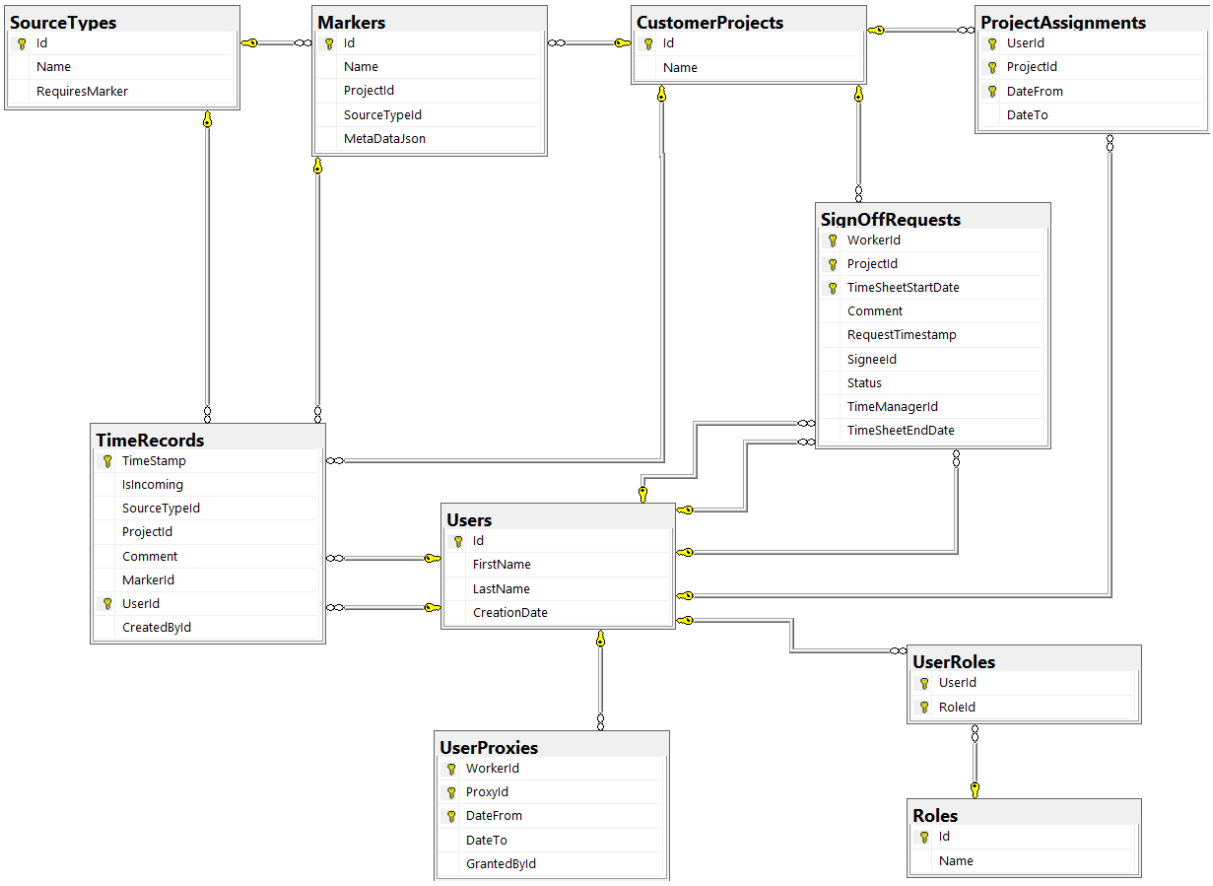

Fig. 5.1.: Database diagram in Microsoft SQL Server notation

5.2.5. Documentation and hosting

The API documentation is done with *Swagger*, which provides an automatically generated Web UI. While we use Internet Information Services (IIS) Express to deploy the Backend API for debugging, ASP.NET Core supports (and promotes) lightweight *Kestrel* server, which can be run on all platforms supported by ASP.NET Core, including Windows and Linux. This prototype is hosted on Azure services and can be easily deployed directly through the Integrated Development Environment (IDE), but using Kestrel would even allow running the web server from a console application on a regular PC.

Fig. 5.2.: Overview of the API in the Swagger UI (part 1 of 2)

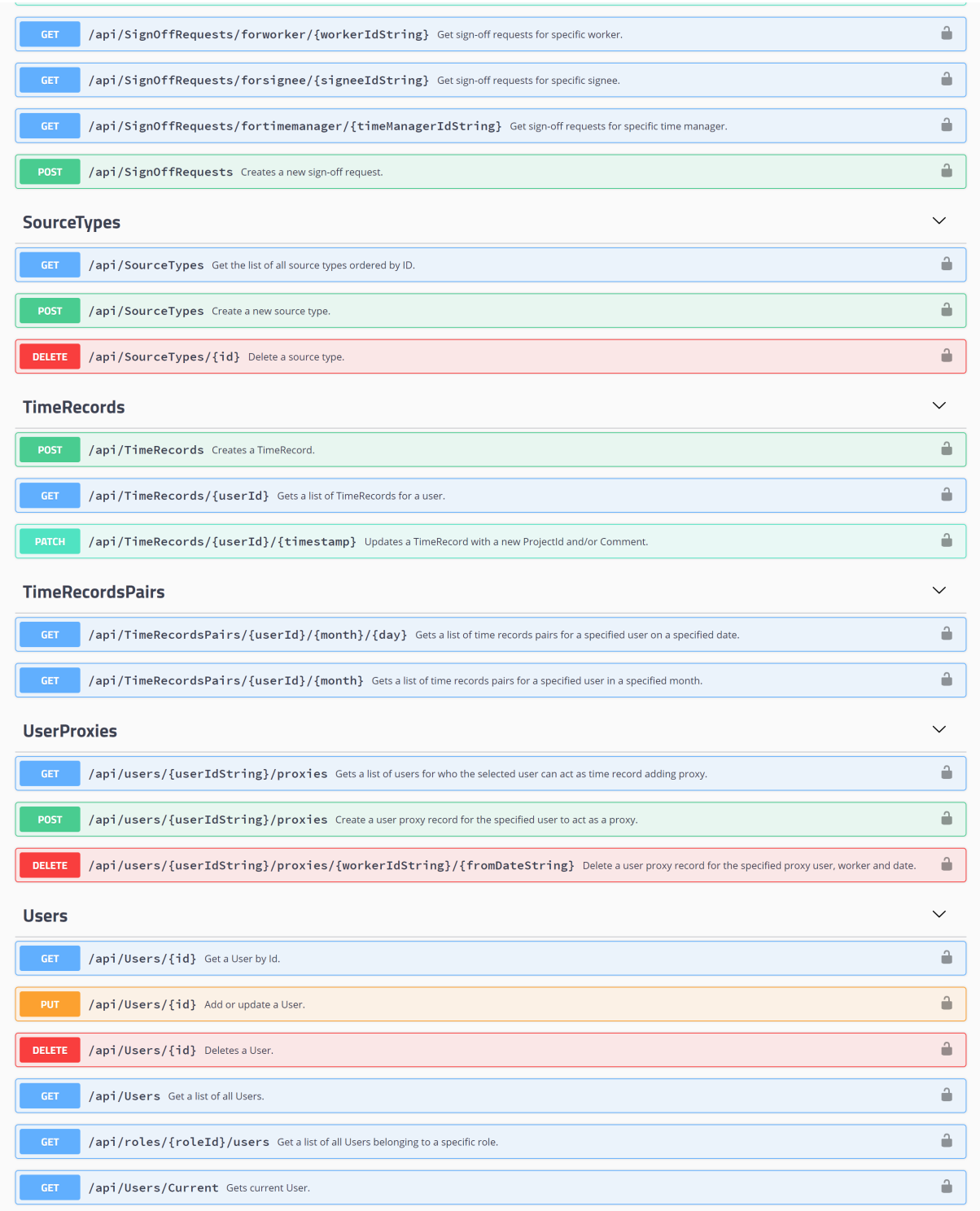

Fig. 5.3.: Overview of the API in the Swagger UI (part 2 of 2)

5.2.6. Authorization

While *Authentication* is a different component of the framework, represented by an external service, the *Authorization* mechanism is implemented within the Backend API. Every user has one or many roles associated with him, which limits the methods he can call on the API. Although ASP.NET Core supports role-based authorization through *Identity*

mechanism, implementation of *Identity* is not straightforward, therefore we decided to exclude it from this prototype. Instead we apply a policy-based authorization, where every policy checks for one or several roles of a user.

Policy specification is very simple and can be done in the main services configuration method of the project:

```
services.AddAuthorization(options =>
{
        options.AddPolicy("IsSystemManager", policy =>
        {
                policy.RequireAuthenticatedUser();
                policy.RequireAssertion(async context => await IsOfAnyRoles(
                        new[] {"systemmanager"}, services, context));
                policy.Build();
        });
}
```

List. 5.1: Policy specification

In Listing 5.1 `policy.RequireAssertion()` calls a custom-written function that retrieves the set of the current user's roles and checks them against the set of roles provided in the function's argument.

5.3. Web UI

As part of this prototype we developed a Web Interface as a possible client interface to the Backend API. Its audience is any type of management user. For its implementation we again decided to use **ASP.NET Core 2.0**, following the list of its advantages we described in the previous section.

5.3.1. Authorization

The set of features the Web UI provides to every user is limited by his roles. We're using the same policy-based authorization mechanism as in the Backend API. However, since authorization checks can be (and are) used on controller level, controller's method level and anywhere else in the code or on the view, calls to `IsOfAnyRoles()` tend to be very frequent. Every call requires acquiring current user's roles through an expensive API call. Thus, we implemented caching of user's role in the session storage via cookie, which is very well supported by ASP.NET Core. The storage uses *lazy initialization* to save current user's roles on first request.

Policy-based authorization mechanism in ASP.NET Core allows conditional View rendering based on user's policy. Listing 5.2 demonstrates its implementation.

```
@if ((await AuthorizationService.AuthorizeAsync(User, this.ViewContext, "
    CanWorkWithProjects")).Succeeded)
{
        <a class="btn btn-primary" asp-controller="Projects" asp-action="Index">
                Projects
        </a>
```

```
6 }
```

List. 5.2: Policy specification

ASP.NET Core uses the *Razor* engine for creating dynamic pages by running arbitrary .NET code snippets. As seen in this example, a method is called on `AuthorizationService` instance that was injected to the page's code before, running an authorization check against a policy "CanWorkWithProjects". Based on the result of that check, a button will be displayed or hidden on this page. This mechanism provides us with great flexibility and code reuse in View development.

5.4. Mobile UI

For a worker user we implemented a separate client application in the form of a mobile application. While this prototype focuses on an Android application, we developed it with cross-platform capability in mind. With that requirement, we chose **Xamarin Forms** as a framework for Mobile UI. Like ASP.NET Core for other components, Xamarin uses C# as development language and has a plethora of third-party libraries accessible through NuGet repositories. It allows to reuse most of the code across different platforms (including currently iOS, Android and Windows 10 Universal Windows Platform), resorting to platform-specific code only to call specific platform APIs.

Like Windows Presentation Foundation (WPF) on desktop Windows systems, Xamarin supports binding View Model classes to Views, allowing extensive use of Model-View-View Model (MVVM) software design pattern. We use the *FreshMVVM* library that adds further support of MVVM with naming conventions, built-in dependency injection containers and page navigation stack improvements.

Mobile UI provides two main groups of functionality: creating time records and displaying time record history.

5.4.1. Creating time records

The main page allows the user to quickly create a time record by choosing a project and pressing a button. The projects selection is limited by project assignments created by time manager through Web UI. The time record gets created using the current time as a timestamp and sent to the Backend API.

Alternatively, the user can activate the QR Code-scanning functionality. This will display a camera view, which the user can use to point the device to a QR Code on his workplace. After the code is scanned and recognized, the Mobile UI checks it against the Backend API to determine if it is a valid Marker ID and is corresponding to a project that the user is assigned to. If everything is correct, the user can apply the project and marker data to a new time record.

Finally, a background task is constantly scanning via Bluetooth for new Beacons. Whenever a Beacon is detected, it decodes the Beacon frame payload to get its identifier and,

like with the QR Code, sends it to the Backend API in order to receive marker and project information. However, the main difference is that it does not prompt the user before creating a time record, making this method completely automatic. When the beacon leaves the receiving range of the device, after a specific timeout (e.g. 60 seconds) another time record gets created to tell the system that the worker left his workplace.

5.4.2. Managing existing time records

On a different page the worker user gets an overview of his previous time records. This list gets updated every time he creates a new time record and provides filtering by month and year. In case of a missing time record, the application will display a warning message, prompting the user to take action. We designed the system to give worker an opportunity to fix inconsistencies in a previous day. Any other dates would require involvement of his time manager.

5.5. Authentication service

While ASP.NET Core can provide authentication service on its own through *Identity* mechanism, proper cross-component authentication might be more easily achieved through an external authentication service. ASP.NET Core has built-in support for Google, Facebook and other popular means of authentication. We decided to use *Azure Active Directory (AD)* since it provides flexible authentication mechanisms, integration to ASP.NET Core and Xamarin through official and third-party libraries, allowing all components of our solution to use a single Azure AD tenant. It is also well-connected to regular AD and Office 365, which are widely used in the corporate sector[19].

5.6. Example scenarios

The following steps describe a typical scenario for the implemented prototype's users. The company "Small Company" was recently signed up for a new project with their customer "Wealthy Customer, Inc". The example workflow includes a member of the "Wealthy Customer, Inc" management team George Manager, who's responsible for managing the new project, two software developers from "Small Company": Bob Skilled and John Newguy, as well as their colleague Tim Finalsay, who's a manager in "Small Company", and Bill Admin, the system administrator.

1. Bill Admin signs in to the Web UI in order to create the new project in the system.

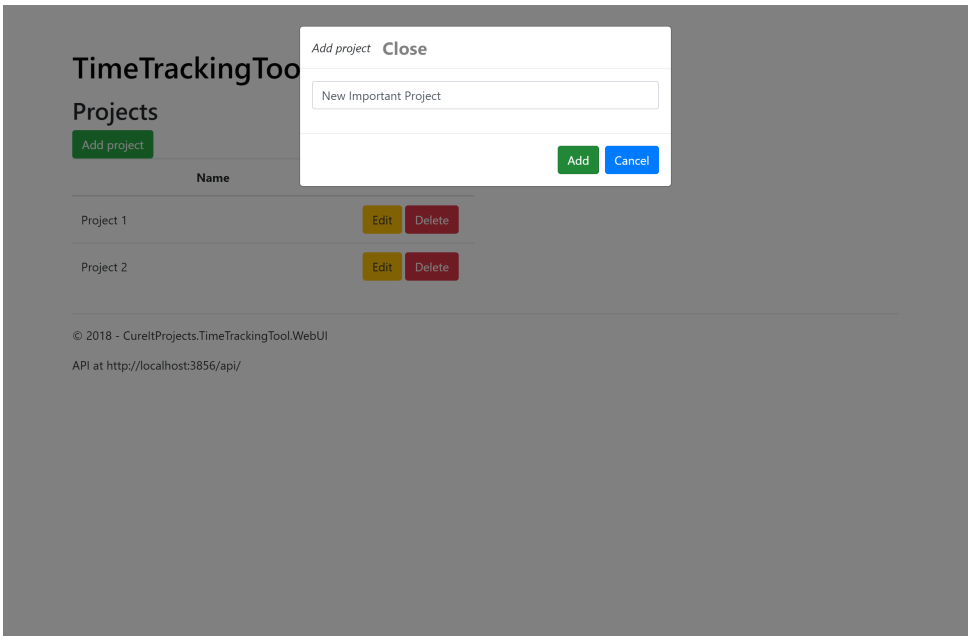

Fig. 5.4.: Creating a new project

2. After Bill is done, George Manager assigns John Newguy to the project by creating a project assignment for him.

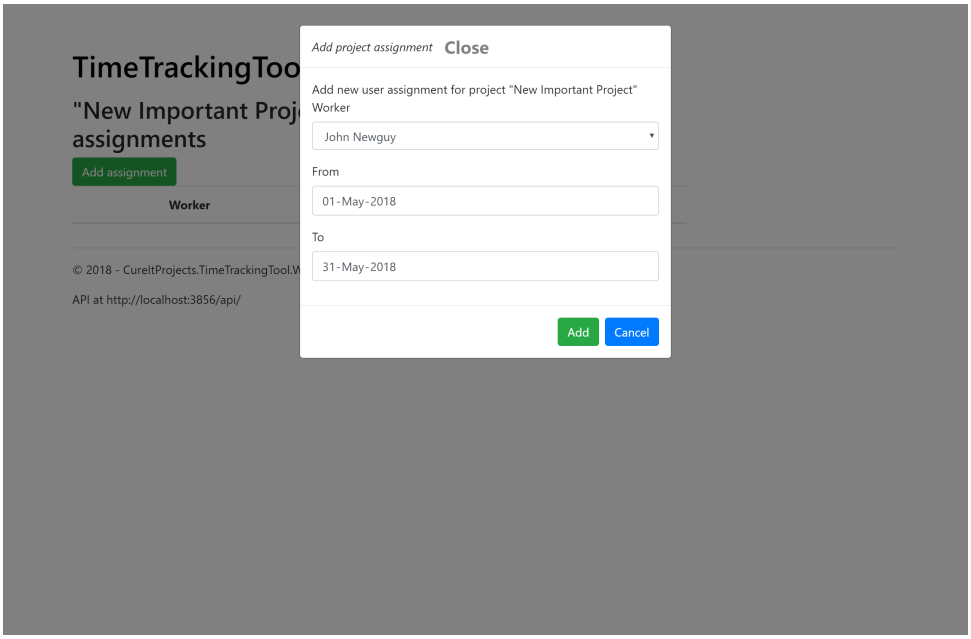

Fig. 5.5.: Creating a new project assignment

3. Since ”Wealthy Customer, Inc” is an old customer of ”Small Company”, they have installed a licensed beacon device in the developers’ room. George wants to use it to track John’s work and asks Bill Admin to add the beacon to the system, linking it to the new project.

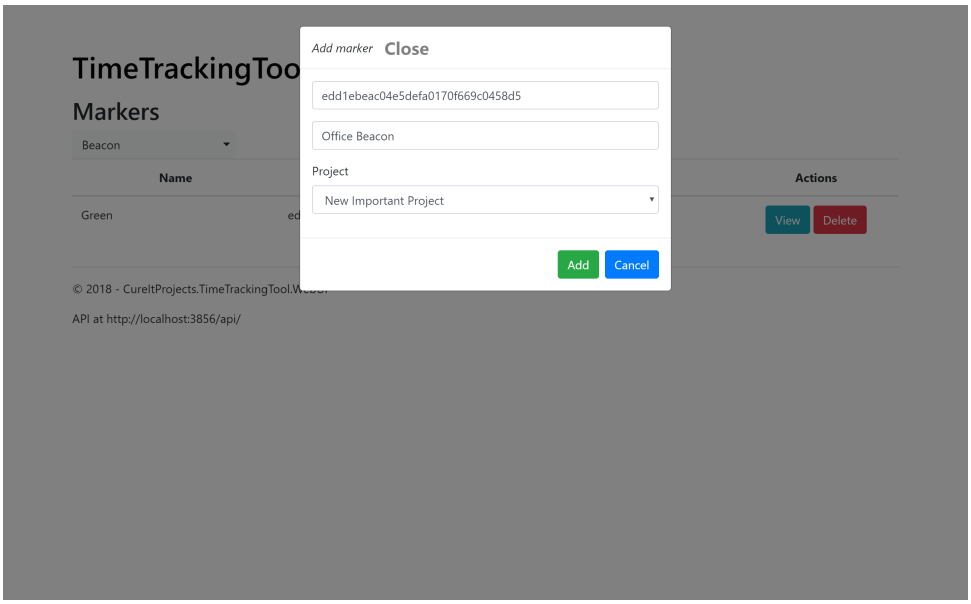

Fig. 5.6.: Adding a marker

4. Before going to work, John Newguy signs in to the Mobile UI on his Android device.
5. When he arrives to the "Wealthy Customer, Inc" office, the Mobile UI detects the beacon and "checks in" John.

Fig. 5.7.: A smartphone with a Mobile UI app next to a beacon device

6. At the end of the day, John leaves the office. After his phone loses the beacon signal, it "checks out" John from work.
7. John checks the timesheet in the Mobile UI to see if everything got created correctly.

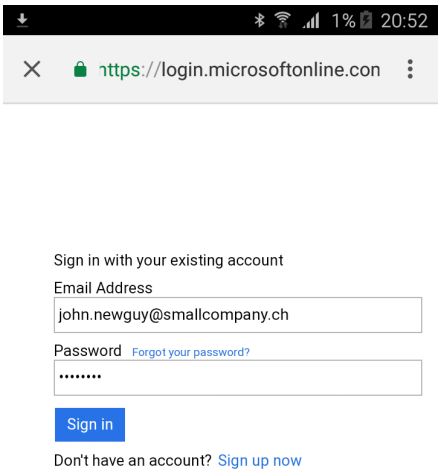

Fig. 5.8.: Signing in to the Mobile UI

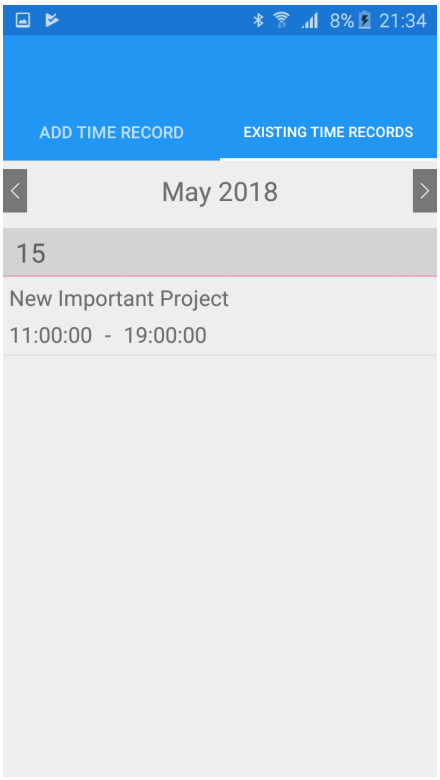

Fig. 5.9.: Timesheet on the Mobile UI

8. On the 1st of the next month George Manager from "Wealthy Customer, Inc" decides to check on John's work. He signs in to the Web UI, selects John as a worker and the previous month to have an overview of his logged time.

Fig. 5.10.: Timesheet view on the Web UI

9. Since everything seems correct to George, he sends a sign-off request to Tim Finalsay so that John can get paid for his work.

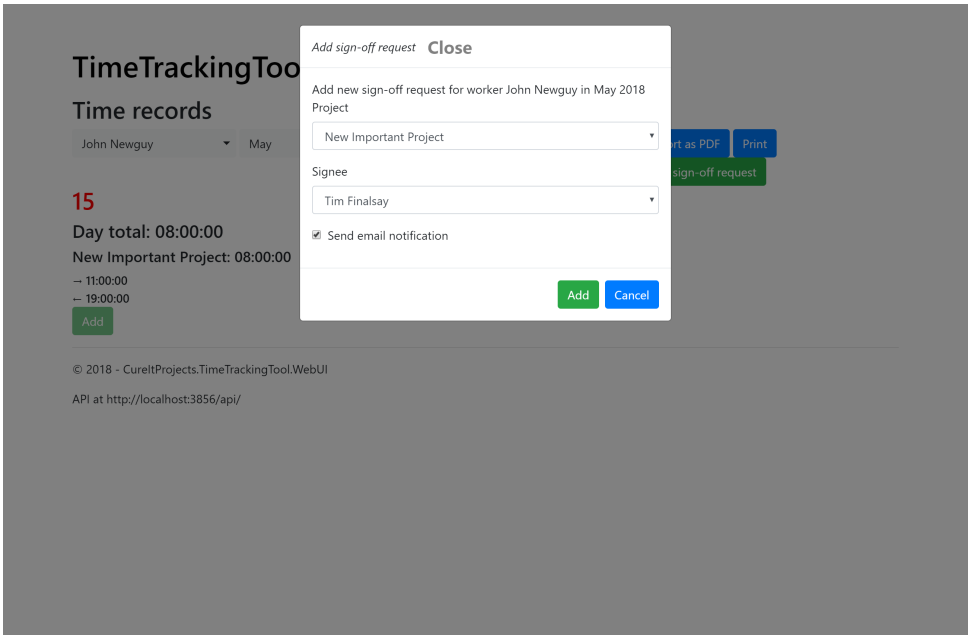

Fig. 5.11.: Creating a new sign-off request

10. Tim receives a notification email and uses the included URL to navigate to the sign-off request details.
11. After reviewing the time records, Tim accepts the sign-off request, locking the project for John on that month from any further changes.

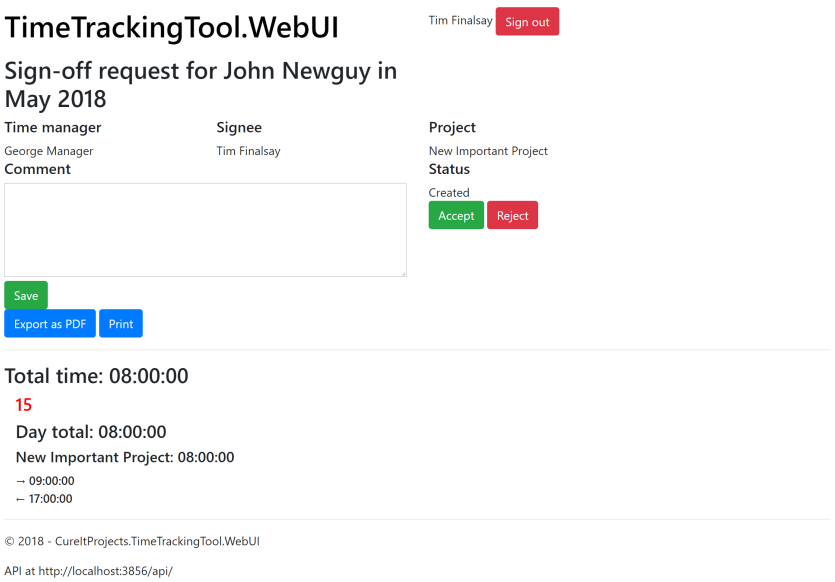

Fig. 5.12.: Sign-off request details view on the Web UI

12. Tim also prints out the timesheet for his archive.

The next scenario demonstrates how a different method of creating time records ties in to the system and shows additional management features.

1. Tim decides to send another employee, Bob Skilled, to an off-site location to work on the project. When Bob arrives to work, he signs in to the Mobile UI and creates a time record manually, since there's no beacon there.
2. When he comes to work next day the app notifies him that he forgot to check out yesterday.

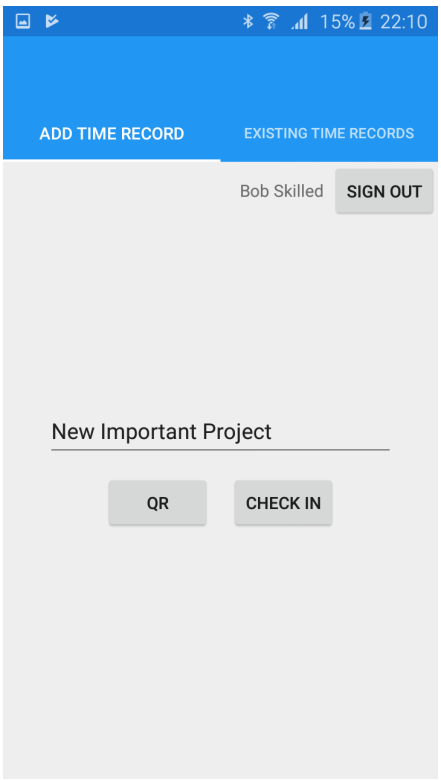

Fig. 5.13.: Adding a time record manually on the Mobile UI

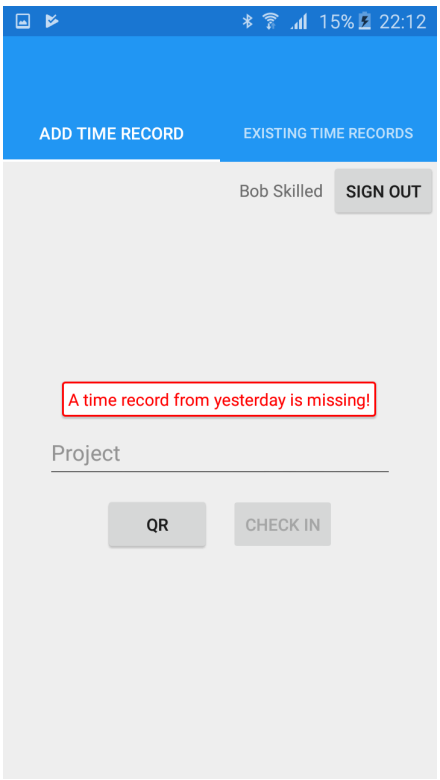

Fig. 5.14.: Missing time record notification on the Mobile UI

3. He's able to fix his mistake by adding a missing time record.
4. Next month George Manager sees that Bob forgot to check out once more and wants to fix that.

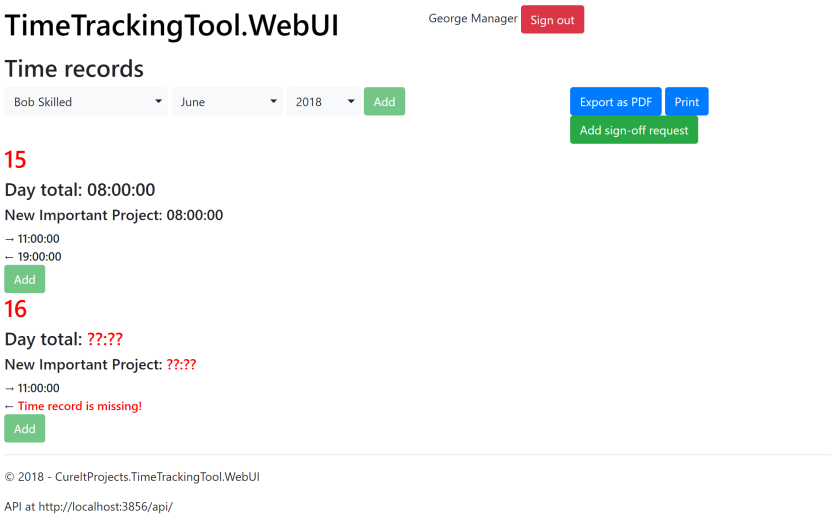

Fig. 5.15.: Timesheet view on the Web UI with missing time records

5. Bill Admin grants him the temporary rights to create time records for Bob.

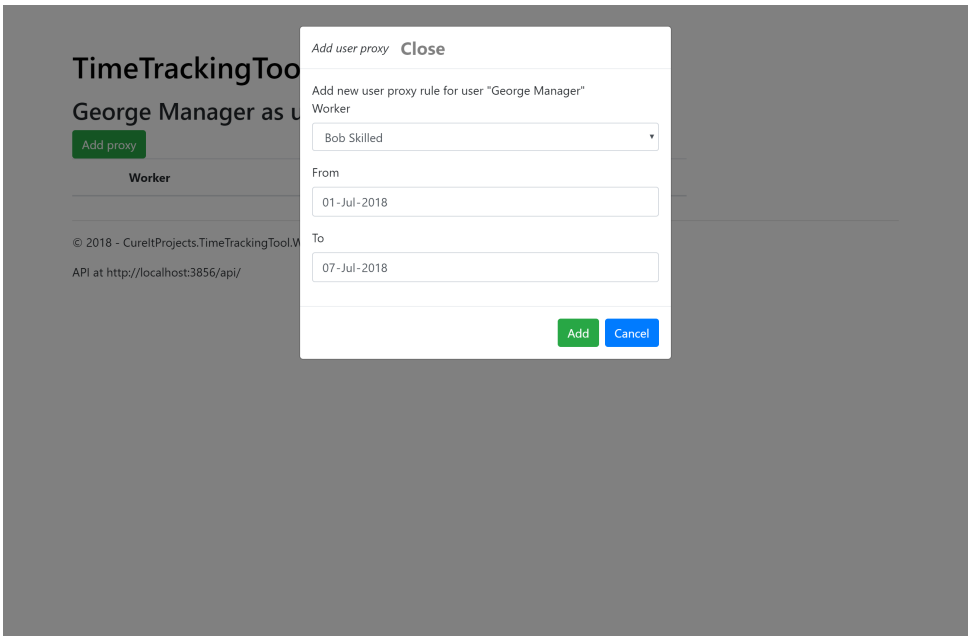

Fig. 5.16.: Adding a user proxy permission

6. George adds the missing time record and proceeds by sending the sign-off request to Tim.

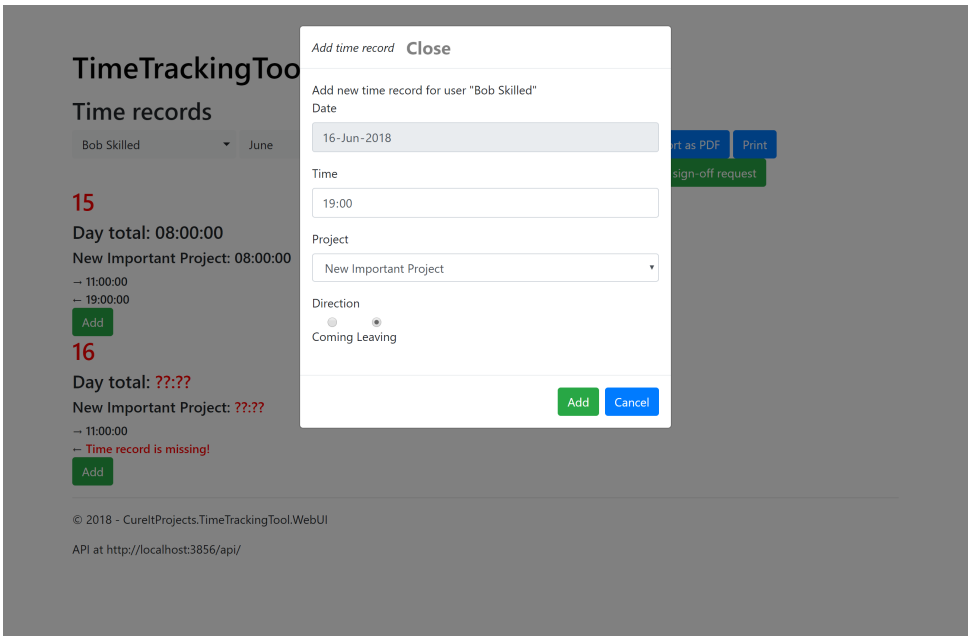

Fig. 5.17.: Adding a time record as a user proxy on the Web UI

These scenarios show how multiple technologies and clients can be combined within the same framework, complementing each other and helping the customer to gather the required working time data in different environments.

6 Future work

6.1. As a concept

Our current implementation only supports detalization on the project level. However, often it is important to manage working time on a lower level, e.g. tasks within the same project. We will follow up with adding this level of detail, making it possible to specify task on creating the time record. This will also help further integration with task-management solutions like *Atlassian Jira* or *GitHub*: logging time spent on a task on those platforms would automatically create time records in our system and vice versa, specifying work item number in the time record comment would update it in the task management system.

We also plan to investigate how well other positioning technologies fit into our model. Two top-priority directions of research are geofencing with GPS technology and precise positioning with multiple Bluetooth Beacons.

6.2. As a product

Since the project is developed with support of *Cure-IT AG*, we are planning to release it as a marketable product by the end of the year. This will require many quality of life improvements, from better thought-through user interface to localization.

A crucial feature missing from the current implementation of the Mobile UI is work in background. Major mobile operating systems like Android and iOS put applications into background after a period of no user interaction, practically freezing their operations. This will stop scanning for Beacons and prevent them from creating new time records. Thus, it is required to implement periodic scanning in a background task through platform-specific mechanisms.

Another important step is the implementation of an iOS Mobile UI. According to *StatCounter*, iOS is a third most popular operating system in the world[20], making it a good decision to expand the framework there.

7 Conclusion

Employee hour tracking is a problem that doesn't have a "silver bullet" technology. However, modern state of technology provides a wide choice of technologies that can be selected or combined based on specific customer's needs.

In this thesis we propose a framework concept able to incorporate different technologies of proximity and location tracking to log employees' working time. We define the core backend with universal API and an abstract client registering time records and communicating them to the API.

Furthermore, we developed a prototype implementation of the proposed framework, showing the possibility and feasibility of combining several proximity and location technologies for employee time tracking. Using am extensible framework as a basis for a time tracking system can be very useful for customer-specific customization.

A

Common Acronyms

AD	Active Directory
AIDC	Automatic Identification and Data Capture
API	Application Programming Interface
BLE	Bluetooth Low Energy
CRUD	Create/Read/Update/Delete
CSV	Comma-Separated Values
DTO	Data Transfer Object
EF	Entity Framework
GPS	Global Positioning System
IDE	Integrated Development Environment
IIS	Internet Information Services
MVC	Model-View-Controller
MVVM	Model-View-View Model
ORM	Object-Relational Mapping
NFC	Near-Field Communication
QR Code	Quick Response Code
RFID	Radio-Frequency Identification
TTT	Time-Tracking Tool
UI	User Interface
URL	Uniform Resource Locator
UUID	Universally Unique Identifier
WPF	Windows Presentation Foundation
XAML	Extensible Application Markup Language

B

License of the Documentation

References

[1] Willard Le Grand Bundy. Workman's time-recorder., May 1891. US Patent 452,894. 3

[2] Anil K. Jain, Patrick Flynn, and Arun A. Ross. *Handbook of Biometrics*. Springer-Verlag, Berlin, Heidelberg, 2007. 9

[3] Clemens Nylandsted Klokmose, Matthias Korn, and Henrik Blunck. Wifi proximity detection in mobile web applications. In *EICS*, 2014. 8

[4] Emilio Mordini and Holly Ashton. The transparent body: Medical information, physical privacy and respect for body integrity. In *Second generation biometrics: the ethical, legal and social context*, pages 257–283. Springer, 2012. 10

[5] M. E. Rida, F. Liu, Y. Jadi, A. A. A. Algawhari, and A. Askourih. Indoor location position based on bluetooth signal strength. In *2015 2nd International Conference on Information Science and Control Engineering*, pages 769–773, April 2015. 9

Referenced Web Resources

[6] Free Documentation Licence (GNU FDL). http://www.gnu.org/licenses/fdl.txt (accessed July 30, 2005).

[7] How to Automate Payroll. https://www.inc.com/guides/2010/12/how-to-automate-payroll.html (accessed June 19, 2018). 3

[8] Barcode. https://en.wikipedia.org/wiki/Barcode (accessed June 21, 2018). 6

[9] Bar code technology remains most popular AIDC. https://searcherp.techtarget.com/feature/Bar-code-technology-remains-most-popular-AIDC (accessed June 21, 2018). 7

[10] QR Codes? Don't Bother. 5 Reasons. https://www.inc.com/eric-v-holtzclaw/qr-codes-dont-bother-five-reasons.html (accessed June 21, 2018). 7

[11] What is RFID? https://www.epc-rfid.info/rfid (accessed June 23, 2018). 7

[12] Two in three phones to come with NFC in 2018. https://www.nfcworld.com/2014/02/12/327790/two-three-phones-come-nfc-2018/ (accessed June 23, 2018). 7

[13] WiFi Location Tracking: Is It The Right Technology For Your Application? https://www.airfinder.com/blog/wifi-location-tracking (accessed June 23, 2018). 8

[14] What is geofencing? Putting location to work. https://www.cio.com/article/2383123/mobile/geofencing-explained.html (accessed June 21, 2018).

[15] Web Framework Benchmarks. https://www.techempower.com/benchmarks/ (accessed June 27, 2018). 20

[16] TIOBE Index for June 2018. https://www.tiobe.com/tiobe-index/ (accessed June 27, 2018). 20

[17] NuGet Gallery - Statistics. https://www.nuget.org/stats (accessed July 01, 2018). 20

[18] Introduction to Identity on ASP.NET Core. https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-2.1&tabs=visual-studio%2Caspnetcore2x (accessed July 01, 2018). 20

[19] Microsoft Azure Active Directory - Review 2017. http://uk.pcmag.com/microsoft-azure-active-directory/71365/review/microsoft-azure-active-directory (accessed July 02, 2018). 27

[20] Operating System Market Share Worldwide - June 2018. http://gs.statcounter.com/os-market-share (accessed July 03, 2018). 36

[21] Technavio Predicts Strong Growth in Biometric Workforce Management. https://findbiometrics.com/technavio-biometric-workforce-management-303317/ (accessed July 05, 2018). 10