Maximilian Weiß-Reinthaler, 00835113

Implementierung und Evaluierung einer http-basierten Webanwendung nach Vorbild eines RESTful Web Services für ein industrielles Warendepot

Masterarbeit

von

Maximilian Weiß-Reinthaler, BSc

Technische Universität Graz

Fakultät für Informatik und Biomedizinische Technik

Institute of Interactive Systems and Data Science

Ao. Univ.-Prof. Dipl.-Ing. Dr.techn. Nikolai Scerbakov

Graz, Januar 2018

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich und inhaltlichen entnommen Stellen als solche kenntlich gemacht habe.

Graz, am _____ _____

Datum Unterschrift

STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Graz, am _____ _____

Date Signature

KURZFASSUNG

Diese Arbeit beschäftigt sich mit der Planung und Implementierung einer Softwarelösung für ein industrielles Warendepot. Die Aufgabe der Softwarelösung ist hierbei das Verarbeiten von Anforderungen zur Ein- und Auslagerung von Lagergütern und deren Abarbeitung innerhalb des Systems. Dies beinhaltet die durchgehende Begleitung eines Lagerguts durch den Prozess der Ein- und Auslagerung, sowie während der Dienstleistungen.

Die Komponenten dieser Softwarelösung sind ein Webservice der Daten verwaltet und für die anderen Komponenten zur Verfügung stellt, ein Webclient mit dessen Hilfe der Benutzer Daten überwachen und bearbeiten kann, sowie ein Client auf einem Handheld Computer mit dem sich Tätigkeiten innerhalb des Lagers protokollieren lassen.

Die Arbeit beschäftigt sich mit der Implementierung der Logik des Backends und beleuchtet außerdem Sicherheitsaspekte, die dabei zu berücksichtigen sind. Es werden typische Bedrohungen ausgemacht und erläutert wie das implementierte System davor geschützt wird. Es wird dargestellt wie das Backend mit Hilfe des PHP-Frameworks *CodeIgniter* konstruiert wurde und wie dessen Struktur genutzt wurde um bestimmte Funktionen des Backends als öffentliches API bereitzustellen.

Die Implementierung der Software für den mobilen Client zeigt auf wie das offene API benutzt werden kann. Die Testphase, die zusammen mit dem Kunden durchgeführt wurde, hat aufgezeigt, dass Effizienz und Usability der mobilen Applikation maßgeblich für die Akzeptanz des Systems verantwortlich sind. Ein weiteres Lager dieser Art wird vom Kunden bereits erbaut.

ABSTRACT

This thesis deals with the planning and implementation of a software solution for an industrial warehouse. It is the systems task to process all incoming requests. To store or outhouse goods and to organize and document all the further steps, a good has to take in the internal system. The components of this software solution are a backend server which provides data for the other components, a web client which allows users to monitor and maintain data and at last a client on a handheld computer which should be used for operations on the goods inside the warehouse.

One part of the thesis deals with the implementation of the backend and aspects of security, one has to consider. Typical threads will be explained. It will be shown how the implemented system deals with those threads. The php-Framework *CodeIgniter* was used to design the Backend. The architecture of the framework was adapted to deploy all the functions the web client needs and also an open API for the mobile devices.

The implementation of the mobile client software shows how the open API can be used by other systems. The period of testing, were developers and employees of the customer worked together, showed how important efficiency and usability are to such an interactive system. The customer is planning yet another warehouse of this kind.

INHALTSVERZEICHNIS

1 EINLEITUNG

Die Firma MOTIONDATA Software GmbH in Seiersberg bietet Software Komplett Lösungen für KFZ-Betriebe an. Das Kernprodukt der Firma ist das MOTIONDATA Dealer-Management-System. Um dieses DMS herum bietet die Firma außerdem eine breite Palette an Software und Hardware Lösungen für alle möglichen Anforderungen an Autohändler, Werkstätten und Teilehändler. Eine der Hauptaufgaben der Entwicklungsabteilung ist die stetige Integration von Schnittstellen, Webservices und Output von Fremdsystemen (oft Konkurrenzsystemen) in MOTIONDATA.

Im Laufe dieser Diplomarbeit soll die Anforderung eines Kunden der Firma MOTIONDATA behandelt werden. Bei dem Kunden handelt es sich um einen größeren KFZ-Dienstleister. Die Anforderung betrifft im konkreten ein Areal dieses Kunden auf dem eine größere Zahl von Werkstätten und Verkaufszentren, auf engstem Raum, nebeneinanderliegen. Der Kunde will Räder und Reifen von Kundenfahrzeugen die in den umliegenden Werkstätten und Verkaufszentren bearbeitet und verkauft werden in ein zentrales Reifenlager einlagern können. Das Gebäude, in dem sich das Reifenlager befindet, wurde bereits errichtet und steht einsatzbereit am Firmengelände. Die Aufgabe der MOTIONDATA Software GmbH ist die Realisierung der kompletten Lagerlogistik sowie die Anbindung an die Dealer-Management-Systeme der Werkstätten und Verkaufszentren. Für die Lagersoftware hat sich der Kunde den Namen ‚*Reifenhotel*' ausgesucht. In diesem Kapitel wird die wirtschaftliche Motivation, die konkrete Anforderung an die Logistik und die Software, sowie die damit verbundenen Problemstellungen behandelt.

1.1 WIRTSCHAFTLICHE ÜBERLEGUNG

Derzeit hat der Kunde 2.951 Garnituren Kundenräder in einem externen Lager eingelagert. Dabei handelt es sich um 98 Prozent Räder (Reifen auf Felgen) und 2 Prozent Reifen (Reifen ohne Felgen). Die Unterscheidung zwischen Reifen und Rädern spielt in den späteren Unterkapiteln insofern eine Rolle, als dass Reifen auf Felgen auch gewaschen und gewuchtet werden können, während diese Dienstleistungen für Reifen ohne Felgen wegfallen.

Die Einlagerungsquote auf alle Betriebe des Kunden im Jahr 2014 betrug 17,8 Prozent. Diese Quote wird anhand der aktiven Fahrzeuge gemessen, daher Fahrzeuge die in den letzten zwölf Monaten einen Werkstattaufenthalt hatten. Auf die Betriebe des Referenzgeländes gerechnet liegt die Einlagerungsquote allerdings weit höher, bei ca. 35 Prozent pro Jahr. Tendenz stark steigend. Dies liegt daran, dass das Referenzgelände in einem Ballungszentrum am Rande einer Großstadt liegt. Hier lagern Kunden ihre Winter- oder Sommerräder aus Platzgründen tendenziell lieber in der Werkstatt, in der sie sie wechseln lassen, ein.

Bei ca. 23.400 Werkstattkunden im Jahr 2014 ergäbe bereits eine Einlagerungsquote von 30 Prozent einen Bedarf von 7000 Stellplätzen. Dieser Bedarf kann zurzeit aber nicht annähernd gedeckt werden. Durch die Errichtung des zentralen Reifenlagers können ab Oktober 2015 4.588 Stellplätze angeboten werden.

Diese Überlegungen sind auch für die Firma MOTIONDATA relevant. Man kann anhand der derzeitigen Lage und der prognostizierten Entwicklung für diesen Markt davon ausgehen, dass der Kunde dieses Geschäftsmodell in Zukunft eventuell ausbauen wird und auch andere Standorte mit der Reifenhotel Logistik ausrüsten will. Es ist daher von großer Bedeutung für die Firma sich hier als Anbieter zu positionieren und ein erfolgreiches Referenzprojekt für diesen Markt umzusetzen.

1.2 ANFORDERUNG

Im Folgenden werden die konkreten Anforderungen, wie sie der Kunden formuliert hat, von der *Einlagerungsanforderung* über das Abarbeiten von *Dienstleistungen* an den Reifen, die Terminisierung von Auslagerungen, deren Stornierung, *Just-In-Time-Auslagerungen* und *ad-hoc-Auslagerungen* bis hin zur endgültigen Auslagerung an den Kunden, Mechaniker oder das zuständige Kundencenter, erläutert.

1.2.1 EINLAGERUNG

Die Anforderung zur Einlagerung eines Lagerguts muss aus den Dealer Management Systemen der Werkstätten und Verkaufszentren kommen. Diese werden in weiterer Folge nur noch KC (Kundencenter) genannt. Die Einlagerungsnummern sollen von den Systemen automatisch vergeben werden und müssen innerhalb des Systems eindeutig sein. Die Einlagerungsnummer ist der Schlüsselbegriff für den ganzen Ein- und Auslagerungsprozess. Die ersten beiden Stellen der Einlagerungsnummer identifizieren das Kundencenter, von dem die Anforderung kommt. Bei der Erstellung der Einlagerungsanforderung wird eine Banderole gedruckt auf der sich ein Barcode mit der Einlagerungsnummer befindet. Dieser Barcode wird in weitere Folge zur Identifizierung des Lagerguts benötigt.

Sobald die Einlagerung vom DMS an die Reifenhotel-Software übermittelt wurde erhält das Lagergut den Status ‚*angefordert*' und wird an die nächste Sammelstelle oder an einen Routenzug übergeben. An der Sammelstelle wechselt der Status des Guts auf ‚*angenommen*'. An diesem Punkt werden die Mitarbeiter des Reifenlagers aktiv. Sie holen die einzulagernden Räder bei den Sammelstellen ab und bringen sie zum Reifenlager. Im Parterre des Reifenlagers werden die Lagergüter aller Kundencenter gesammelt und an den dortigen Checkpoints registriert. Dies geschieht per Handscanner. Der Handscanner scannt die Banderole am Lagergut und den Code des Checkpoints und liefert diese Daten an die zentrale Reifenhotel Software. Diese prüft ob für dieses Lagergut Dienstleistungen, wie Waschen oder Wuchten, vorgesehen sind. Der weitere Status des Lagerguts hängt davon ab ob der Lagermitarbeiter die Dienstleistung gleich durchführen will (Status = ‚*Zur Dienstleistungsdurchführung ausgelagert*') oder doch erst später (Status = ‚*vorbereitet*'). Dies entscheidet er per Handscanner. Räder mit dem Status ‚*Zur Dienstleistungsdurchführung ausgelagert*' werden dann an die Wasch- beziehungsweise Wuchtmaschine weitergeben, Räder mit Status ‚*vorbereitet*' kommen auf die Lagerebenen des Reifenlagers. Der Lagerplatz für jedes einzelne Lagergut wird beim Einlagern auf den Checkpoint berechnet. Hier soll je nach benötigtem Lagerplatz, der sich aus den Reifendimensionen ergibt, das Lager von unten nach oben effizient befüllt werden. Der Lagerplatz den die Reifenhotel Software errechnet, wird ohne Barcode ausgedruckt und an die Rückseite der Banderole geklebt. Trifft das Lagergut am vorgesehenen Lagerplatz ein, muss der Barcode mit der Einlagerungsnummer und der Barcode am Lagerplatz abermals vom Mitarbeiter gescannt werden und die Einlagerung wird per Handscanner an die Reifenhotel-Software gesendet. Jetzt hat das Lagergut den Status ‚*eingelagert*'.

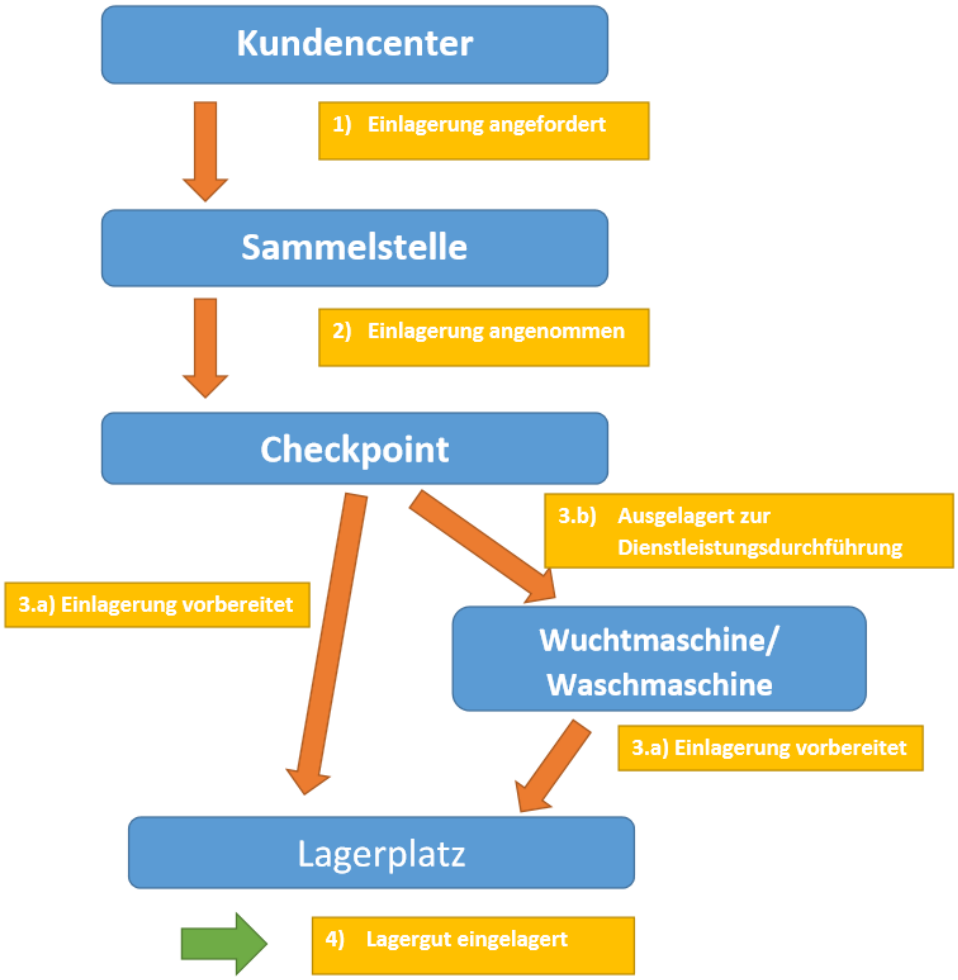

Abbildung 1.1 - Einlagerungsprozess

Der Einlagerungsprozess. Die in blau markierten physischen Orte müssen von einem Lagergut bis zur Einlagerung durchlaufen werden. Währenddessen durchlaufen sie die gelb markierten Zustände.

1.2.2 AUSLAGERUNG

Die Auslagerungsanforderung an das Reifenhotel erfolgt ebenfalls über die Dealer-Management-Systeme. Schlüsselbegriff ist auch hier die Einlagerungsnummer. Generell sollten sich Auslagerungsanforderungen aus Terminen ergeben, die auf das Lagergut mit der betroffenen Einlagerungsnummer referenzieren. Unter Berücksichtigung der KC spezifischen Vorlaufzeit werden Auslagerungsanforderungen an das Reifenhotel gesandt und erhalten dort den Status ‚*terminisiert*'.

Zusätzlich zur Auslagerung auf Termin, können auch sogenannte *ad-hoc-Auslagerungen* vorgenommen werden. Das sind beschleunigte Auslagerungsanforderungen die sofortige Maßnahmen erfordern. Die

Anforderung geht daher sofort an das Reifenhotel und scheint am Display des Lagermitarbeiters, der am nähersten zum Lagergut positioniert ist, auf.

Die Auslagerung mit der höchsten Priorität ist die *just-in-time-Auslagerung*. Diese kommt direkt vom Leitstand der bearbeitenden Werkstatt und muss direkt einem Techniker oder einer Arbeitsbühne zugeordnet werden. Diesem wird das Lagergut dann am Ende des Auslagerungsprozesses übergeben. Die Auslagerung kann aber auch durch eine Übergabe direkt an den Kunden erfolgen.

Innerhalb der Reifenhotel Software werden die Auslagerungen nach diesen Prioritäten gereiht und den Mitarbeitern direkt auf den Handheld Scannern angezeigt. Nimmt der Lagerarbeiter die Auslagerung an, wird der Status des Lagerguts auf ‚*angenommen*' gesetzt und der Mitarbeiter muss den Vorgang durch Scannen des Lagerregals und des Lagerguts bestätigen. Selbiges geschieht am Checkpoint. Hier wird außerdem geprüft ob an dem Lagergut noch etwaige Dienstleistungen offen sind. Sollte das der Fall muss die aus Verrechnungsgründen an das jeweilige KC kommuniziert werden.

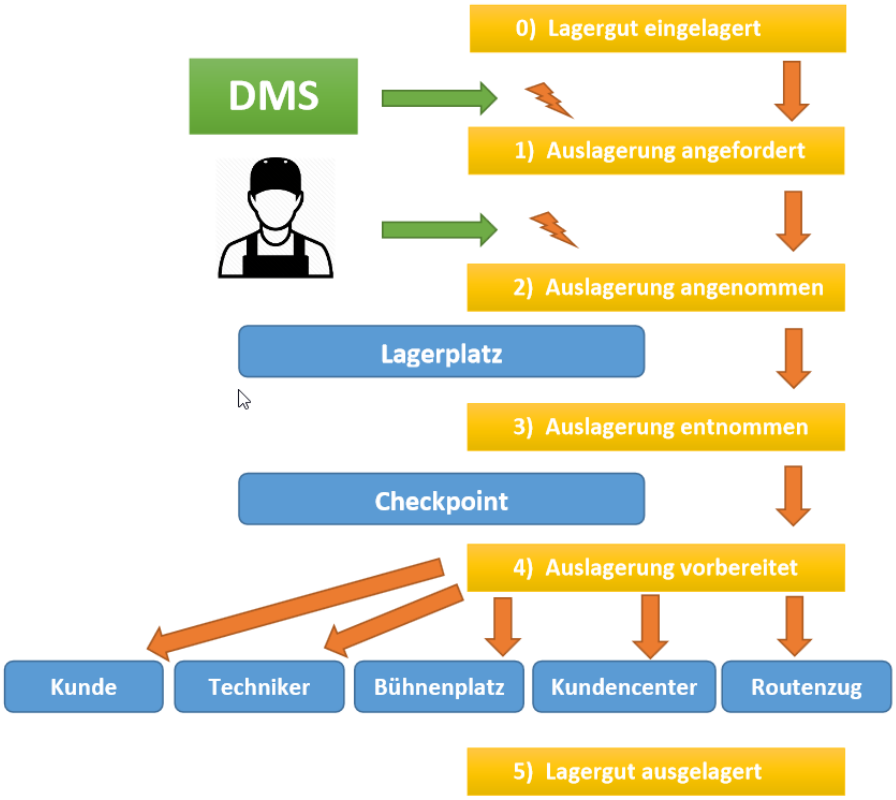

Abbildung 1.2 - Auslagerungsprozess

Der Auslagerungsprozess. Das DMS gib den Anstoß zur Auslagerung eines Lagerguts, der Lagermitarbeiter entscheidet wann er sie annimmt.

1.2.3 STORNIERUNG

Storniert werden kann eine Auslagerungsanforderung nur über das DMS des betroffen KCs. Geschieht das noch während der Status des Lagerguts ‚*Auslagerungsanforderung terminisiert*' lautet wird diese von der Reifenhotel-Software einfach verworfen. Passiert die Stornierung während des Auslagerungsprozesses, muss das Einlagerungsprozedere wieder von dem Status an dem sich das Lagergut befindet, bis zum Status ‚*eingelagert*', durchlaufen werden.

1.2.4 DIENSTLEISTUNGEN

Bei Dienstleistungen handelt es sich um Aktivitäten die an den Lagergütern durchgeführt werden können. Zurzeit hat der Kunde hier den Nachwucht-Service und den Räderwasch-Service bestimmt. Besteht der Bedarf nach einer dieser Dienstleistungen muss dies auch im Rahmen der Einlagerungsanforderung über das DMS bekannt gegeben werden. Der Lagermitarbeiter hat die Möglichkeit diese bereits beim Einlagern am Checkpoint abzuhandeln oder sie per Handscanner auf später zu verschieben. Diese verschobenen Dienstleistungen können von dem Mitarbeiter zu einem späteren Zeitpunkt abgefragt und erledigt werde. Durch Bekanntgabe seiner Position kann der Mitarbeiter die nächste offene Dienstleistung auf seiner Ebene abfragen. Durch das Scannen des Lagerguts und des Lagerorts bestätigt er, dass er die Dienstleistung angenommen hat und der Status des Lagerguts wechselt auf ‚*Zur Dienstleistungsdurchführung ausgelagert*'. An den verschiedenen Wasch- und Wuchtanlagen befinden sich ebenfalls Barcodes, die zur Bestätigung der jeweiligen Dienstleistung gescannt werden müssen. Danach wird das Lagergut gemäß dem Einlagerungsprozedere wieder zurück auf seinen reservierten Lagerort gebucht.

1.2.5 MONITORING

Sämtliche Änderungen des Auftragsstatus werden in einer Lagerbewegungsdatei zur jeweiligen Einlagerungsnummer mit Datum, Uhrzeit und Sachbearbeiter mitgeschrieben. Diese Daten können über den Webclient jederzeit eingesehen werden und von berechtigten Mitarbeitern mit Daten wie Technikernummer/ -name, Kundenname bei Direktauslagerungen oder Fahrziel bei Auslagerung über den Routenzug, ergänzt werden.

In einer weiteren Ansicht sollen relevante Daten auch für Kunden angezeigt werden. Über Monitore in den jeweiligen KCs sollen diese den wartenden Kunden angezeigt werden. So soll die Wartezeit bei *ad-hoc* und *just-in-time-Auslagerungen* nachvollziehbar gemacht werden.

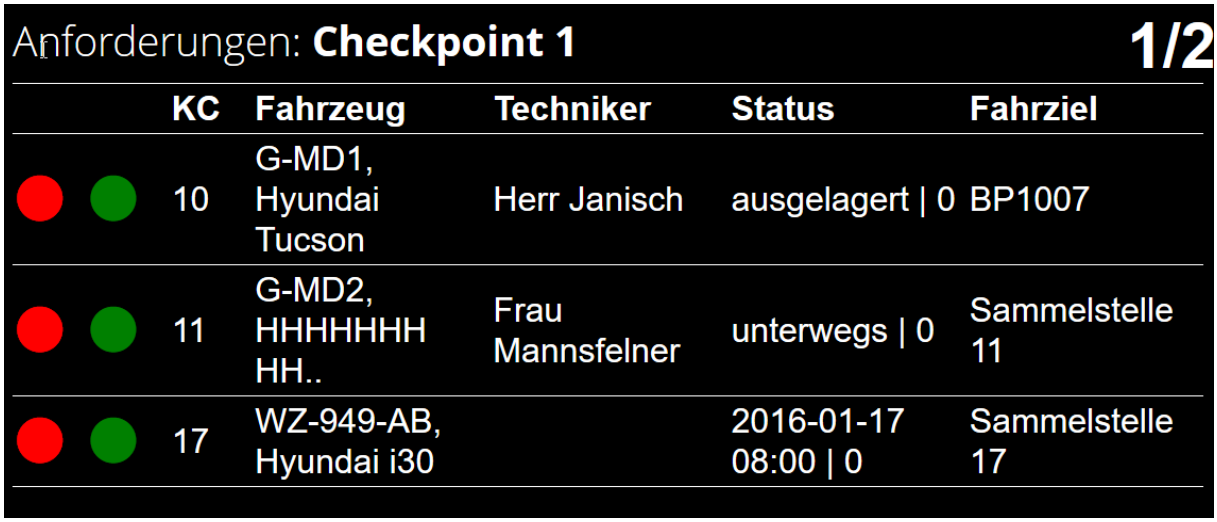

Abbildung 1.3 - Monitoring Tool

Das Monitoring Tool. Auf Bildschirmen in den KC's sollen Kunden den Status ihrer Auslagerungen verfolgen können.

1.2.6 WEITERE ANFORDERUNGSPUNKTE

Die oben genannten Punkte können als Kernaufgaben der Reifenhotel Software bezeichnet werden. Die gesamte Anforderung an die Reifenhotel Gesamtlösung beinhaltet außerdem noch die Punkte

- Umlagerung von Lagergütern,
- Inventur,
- Lagercheck

die zu einem späteren Zeitpunkt noch kurz erläutert werden. Die konkreten Anforderungen an die Handheld Scanner wird ebenfalls in einem eigenen Kapitel, das sich mit der Implementierung diese Geräte beschäftigt, genauer erläutert.

1.3 TECHNISCHE UND LOGISTISCHE HERAUSFORDERUNGEN UND UMSETZUNG

Zur Umsetzung dieser Anforderungen wird eine HTTP-basierte Web Applikation nach dem Vorbild eines RESTful Services implementiert die die Logik der Reifenhotel Software abbilden soll. Diese Applikation bietet eine Schnittstelle für alle anderen Komponenten der Reifenhotel Software, indem sie die Daten des Reifenhotels über die HTTP-Methoden GET, POST und PUT zugänglich macht. Der Server auf dem die Daten des Reifenhotels gespeichert werden befindet sich im Netzwerk der Firma MOTIONDATA Software GmbH und somit nicht im Netzwerk des Kunden. Der unter 1.2.5 erwähnte Webclient wird ebenfalls über diesen Server gehostet und kann daher direkt auf die Daten zugreifen. Die Bedienung der Web Maske zur Bearbeitung von Einlagerungen muss daher passwortgeschützt sein und die üblichen Verschlüsselungsstandards erfüllen.

Für die Zugriffe über das DMS und den Handheld Scanner stehen die HTTP-Methoden der Web Applikation zur Verfügung. Auch hier müssen aus Sicherheitsgründen zusätzliche Daten wie eine

Userkennung und ein Sicherheitstoken übermittelt werden. Wie dies innerhalb des Services und des offenen API umgesetzt wurde wird in den Kapiteln 3.1.4.5 und 4 näher erläutert. Die Web Applikation sendet und empfängt Daten in JSON Format.

Die Kombination aus HTTP-Methoden und JSON-Objekten wurde gewählt um die Kommunikation möglichst schnell und effizient zu gestalten. Bis auf eine Ausnahme, nämlich die Methode zum Anlegen einer Einlagerung die vom jeweiligen DMS aufgerufen wird, ist es gelungen Methoden zu definieren die nur sehr kleine Datenobjekte entgegennehmen. Für den Kunden ist es unerlässlich, dass die Interaktion der Web Applikation und der Handheld Scanner möglichst zeitnah abläuft.

Bei den Handheld Scannern hat sich der Kunde für das Model *Morphic* der Firma *NordicID* entschieden. Hierbei handelt es sich um einen äußerst robusten Kompaktcomputer der mit einem Laser zum Lesen von Barcodes und QR-Codes ausgestattet ist. Wie viele Konkurrenzprodukte verwendet auch dieses Gerät das Betriebssystem *Windows Embedded CE 6.0.*.[1] Daraus resultiert, dass die Implementierung einer mobilen Applikation im Microsoft .Net Framework 3.5 für Windows CE Teil der Kundenanforderung ist.

Windows Embedded CE ist als Betriebssystem für derartige Geräte sehr beliebt da es, basierend auf der herkömmlichen Windows Architektur, vom Hardware Hersteller sehr stark an die Anforderungen des jeweiligen Geräts angepasst werden kann. Daraus sollte sich, je nach Hersteller, ein möglichst effizientes und außerdem robustes Betriebssystem ergeben. So ist es auch durchaus üblich, dass der Hersteller eigene Dynamic Link Libraries (DLLs) zur Verfügung stellt um es Benutzern zu ermöglichen Hardware Komponenten anzusprechen.

Was beim Implementieren der clientseitigen Software allerdings genauso berücksichtigt werden muss ist, dass der Mainstream Support für dieses Betriebssystem bereits am 09.04.2013 abgelaufen ist und der Extended Support am 10.04.2018 abläuft.[2] Da unklar ist wie der Hardware Hersteller damit umgeht sollte beim Planen der clientseitigen Architektur bereits an spätere, alternative Applikationen gedacht werden.

Abgesehen von der eigentlichen Kundenanforderung ist also eine Anforderung der Firma MOTIONDATA an sich selbst, eine Architektur zu schaffen die es in weitere Zukunft möglich macht den HTTP-basierten Webservice von möglichst vielen unterschiedlichen Plattformen anzusprechen. Hierzu soll eine Metaebene geschaffen werden die den Webservice für weitere Implementierungen möglichst gut abstrahiert und beschreibt. Dies ist schon alleine deshalb lohnend da bereits in der ersten Implementierungsphase der Reifenhotel Software mehrere unterschiedliche Komponenten auf den Service zugreifen müssen. Denn neben dem Zugriff via Handheld Scanner müssen auch das MOTIONDATA DMS und andere Managementsystemen, die beim Kunden im Einsatz sind, auf den Service zugreifen. Da der Wille sich an Konkurrenzprodukte anzubinden bei DMS Herstellern

[1] Vgl. https://www.nordicid.com/en/home/products-barcode-uhf-rfid-reader-writer/mobile-readers/nordic-id-morphic_uhf-rfid_barcode_reader-writer-scanner/, letzter Aufruf am 02.01.2018

[2] Vgl. https://support.microsoft.com/de-de/lifecycle/search?alpha=Windows%20Embedded%20CE%206.0, letzter Aufruf am 02.01.2018

verständlicherweise eher gering ist, die Kooperation aller sich im Einsatz befindenden Systeme aber unabdinglich für die Sinnhaftigkeit dieses Projekts ist, muss ein offenes API geschaffen werden, dass für externe Entwickler nachvollziehbar und gut dokumentiert ist. Dieses API muss außerdem von der restlichen Logik des Backends getrennt werden. Damit ist die Logik gemeint die vom Webportal benötigt wird. Dieses Portal ermöglicht dem Benutzer Einlagerungen anzulegen und zu verwalten. Weitere administrative Befugnisse werden durch Benutzerhierarchien geschützt. Dieser Teil des Systems soll nicht in dem offenen API enthalten sein.

In Kapitel 4 soll erläutert werden wie die Architektur des Backends umgesetzt wurde. Dazu wird das gewählte php-Framework *CodeIgniter* näher beleuchtet und auf seine Schwächen und Stärken geprüft. Die Architektur des Frameworks wird dann so adaptiert, dass sie den Anforderungen an das *Reifenhotel*-Backend entspricht.

Die Implementierung der Applikation für die mobilen Clients ist Thema von Kapitel 5. Hier wird das zuvor designte API verwendet. Dazu werden die einzelnen Funktionen der Applikation erläutert und mit der dazu verwendeten Schnittstelle zum Backend in Verbindung gebracht.

In einem weiteren Kapitel soll das Thema Sicherheit behandelt werden. Hierzu soll erörtert werden welche Probleme es bei RESTful Services in Bezug auf Sicherheit gibt. Wie garantiert man, zum Beispiel, dass es sich bei der Aufrufenden Komponente um einen gültigen User handelt und wie sollen Userberechtigungen transferiert werden? Wie werden User Sessions verwaltet mit einem an sich zustandslosen Service? Es sollen existierende Standards evaluiert werden und auf den konkreten Fall der Reifenhotelsoftware umgemünzt werden. Es soll außerdem aufgezeigt werden welche allgemeinen Sicherheitsvorkehrungen noch getroffen werden müssen.

2 ANFORDERUNG DES KUNDEN AN DAS SYSTEM

Die Anforderung des Kunden wurde in Kapitel 1 bereits erläutert. In diesem Kapitel soll erörtert werden wie die einzelnen Prozessschritte, die der Kunde in seiner Anforderung formuliert hat, umgesetzt werden sollen. Die die genaue Implementierung der jeweiligen Komponenten wird in den späteren Kapiteln erfolgen.

Um die Abläufe innerhalb der Reifehotelarchitektur zu veranschaulichen soll die Anforderung des Kunden in mehrere separate Punkte unterteilt und als Usecase formuliert werden. Für jeden dieser Usecases soll aufgezeigt werden welche Funktionen die einzelnen Komponenten der Reifenhotelsoftware bereitstellen müssen um diesen zu bedienen.

2.1 REIFENHOTEL BROWSERANWENDUNG UND MONITORING

Alle Daten die mit der Reifenhotelsoftware verwaltet werden, müssen für die administrativen Benutzer einsehbar sein. Hierzu soll die Softwarelösung eine Browsermaske anbieten die es dem Benutzer erlaubt sich anzumelden und Daten abzufragen. Je nach Berechtigung kann ein Benutzer auch Daten bearbeiten und Auslagerungen anfordern. Es soll daher eine benutzerfreundliche und ansprechende Oberfläche geschaffen werden die auch von weniger technikaffinen Mitarbeitern, schnell bedient werden kann.

In einer weiteren Ansicht sollen Kundenrelevante Daten sichtbar gemacht werden. Da diese auf Fernsehbildschirmen in den KCs gezeigt werden soll diese simpel und ansprechend im Design sein und gut leserlich nur die wichtigsten Daten veranschaulichen. Die Ansicht muss sich selbst aktualisieren und hat eine rein informative Funktion. Sie soll daher keinerlei Userinteraktion erwarten.

2.2 ANLEGEN EINER EINLAGERUNG

Die Einlagerungsdaten zu einem Lagergut müssen an den die Webapplikation des Reifenhotels gesandt werden. Dazu muss die Webapplikation eine Funktion anbieten, die alle Daten entgegennimmt, die der Kunde, zu einem Lagergut, gespeichert haben will. Des Weiteren muss die Webapplikation jede der eingehenden Einlagerungen abarbeiten und die angegebenen Daten auf ihre Korrektheit überprüfen. Um welche Daten es sich hierbei handelt wird in Tabelle 2.2.1 aufgezeigt.

Parameter	Typ	Typ	Beschreibung / Beispiel
partnertoken	String(40)	Benötigt	Eindeutiger Partner-Token, der von MOTIONDATA bekanntgegeben wird.
einlagerungstermin	DateTime	Benötigt	Format: YYYY-MM-DD HH:MM:SS

einlagerungsnummer	String (12)	Benötigt	11000015 (Beispiel: Nummer 15 aus Kundencenter 11)
betriebsnummer	Integer (3)	Benötigt	Betriebsnummer des DMS-Betriebes (Kundencenter-ID 10,11,12,13,17,…)
kunden_id	String(10)	Benötigt	Kundennummer: 1234, A1234
anrede	String(30)	Optional	Herr,Frau,Herr Doktor, Frau DI
nachname	String (80)	Optional	Nachname oder Firmenbezeichnung
nameszusatz	String (50)	Optional	
vorname	String (80)	Optional	Vorname oder Ansprechpartner
strasse	String (80)	optional	
plz	Integer(5)	Optional	
ort	String (80)	Optional	
bundesland	String (40)	Optional	
email	String (80)	Optional	
telefon_buero	String (30)	Optional	
telefon_privat	String (30)	Optional	
telefon_mobil	String (30)	Optional	
fax	String (30)	Optional	
fabrikat	String (80)	Optional	Hyundai
modell	String (80)	Optional	i30
baureihe	String (80)	Optional	
amtl_kennzeichen	String (10)	Benötigt	
fahrgestellnummer	String (17)	Benötigt	
lagergut	String (50)	Optional	Sommerräder (Reifen+Felge), Winterräder (Reifen+Felge), Sommerreifen, Winterreifen
reifenhersteller_vorne	String (20)	Optional	
felgenhersteller_vorne	String (20)	Optional	
felgenart_vorne	String (20)	Optional	Stahl / Alu / Lose
geschwindigkeitssymbol_vorne	Char (1)	Optional	V, W, Q,
stueckzahl_vorne	Integer	Optional	
reifenbreite_vl	Integer	Optional	185
reifenhoehe_vl	Integer	Optional	65
felgendurchmesser_vl	Integer	Optional	15 (in Zoll)
dot_vl	String (4)	Optional	(Vorne Links)
tragfaehigkeit_vl	Integer	Optional	

profiltiefe_vl	Integer	Optional	(Vorne Links) in MM
reifenbreite_vr	Integer	Optional	185
reifenhoehe_vr	Integer	Optional	65
felgendurchmesser_vr	Integer	Optional	15 (in Zoll)
dot_vr	String (4)	Optional	Vorne Rechts)
tragfaehigkeit_vr	Integer	Optional	
profiltiefe _vr	Integer	Optional	(Vorne Rechts) in MM
reifenhersteller_hinten	String (20)	Optional	
felgenhersteller_hinten	String (20)	Optional	
felgenart_hinten	String (20)	Optional	Stahl / Alu / Lose
geschwindigkeitssymbol_hinten	Char (1)	Optional	V, W, Q,
stueckzahl_hinten	Integer	Optional	
tragfaehigkeit_hinten	Integer	Optional	
reifenbreite_hl	Integer	Optional	185
reifenhoehe_hl	Integer	Optional	65
felgendurchmesser_hl	Integer	Optional	15 (in Zoll)
dot_hl	String (4)	Optional	(Hinten Links)
tragfaehigkeit_hl	Integer	Optional	
profiltiefe_hl	Integer	Optional	(Hinten Links) in MM
reifenbreite_hr	Integer	Optional	185
reifenhoehe_hr	Integer	Optional	65
felgendurchmesser_hr	Integer	Optional	15 (in Zoll)
dot_hr	String (4)	Optional	(Hinten Rechts)
tragfaehigkeit_hr	Integer	Optional	
profiltiefe_hr	Integer	Optional	(Hinten Rechts) in MM
reifenhersteller_reserverad	String (20)	Optional	
felgenhersteller_reserverad	String (20)	Optional	
felgenart_reserverad	String (20)	Optional	Stahl / Alu / Lose
geschwindigkeitssymbol_reserverad	Char (1)	Optional	V, W, Q,
stueckzahl_reserverad	Integer	Optional	
tragfaehigkeit_reserverad	Integer	Optional	
reifenbreite_reserverad	Integer	Optional	185
reifenhoehe_hl	Integer	Optional	65
felgendurchmesser_reserverad	Integer	Optional	15 (in Zoll)
dot_reserverad	String (4)	Optional	(Hinten Links)
profiltiefe_reserverad	Integer	Optional	(Hinten Links) in MM

radzierblende_vl	Bool	Optional	true / false
radzierblende_vr	Bool	Optional	true / false
radzierblende_hl	Bool	Optional	true / false
radzierblende_hr	Bool	Optional	true / false
radzierblende_reserverad	Bool	Optional	true / false
radschraube_vl	Bool	Optional	true / false
radschraube_vr	Bool	Optional	true / false
radschraube_hl	Bool	Optional	true / false
radschraube_hr	Bool	Optional	true / false
radschraube_reserverad	Bool	Optional	true / false
dienstleistungen	Array	Optional	Array von Dienstleistungen; Folgende Dienstleistungen sind erlaubt und werden textlich erwartet: Waschen Wuchten
bemerkung	Text	Optional	Bemerkungen zur Einlagerung
benutzer_id	String (10)	Benötigt	Benutzer-ID der Person die die Einlagerung vorgenommen hat
benutzer_name	String (100)	Optional	Benutzername der Person die die Einlagerung vorgenommen hat (Zur Anzeige in der Anwendung)

Tabelle 2.2.1: Parameter die nötig sind um eine Einlagerung in der Reifenhotelsoftware anzulegen.

Die Daten werden unterschieden, zwischen den Feldern ohne die die Einlagerung gar nicht erst angenommen wird und die dazu führen, dass die Funktion der Webapplikation einen Fehler zurückgibt und solchen die zwar auch zu den Pflichtfeldern gehören aber manuell nachbearbeitet werden können. Dies gilt sowohl für unkorrekte und fehlerhaft eingegebene Werte als auch für fehlende Werte.
Entspricht eine Einlagerungsnummer nicht den Namenkonventionen oder die Nummer des übergebenen Kundencenters entspricht nicht den ersten beiden Stellen der Einlagerungsnummer, dann muss die Webapplikation ein Fehlerobjekt zurückgeben.
Weicht jedoch, zum Beispiel, das Feld „*Reifenprofil links vorne*“ zu stark von dem Wert in dem Feld ‚*Reifenprofil rechts vorne*‘ ab, dann soll die Webapplikation trotzdem eine Erfolgsmeldung an das DMS zurückgeben. Innerhalb der Reifenhotelsoftware soll der Fehler aber kommuniziert werden. Die Browseranwendung mit der die Reifenhotelmitarbeiter unter anderem arbeiten soll die Fehler anzeigen und eine Maske bereitstellen in der diese Daten nachbearbeitet werden können.

2.3 EINLOGGEN ALS BENUTZER VIA HANDSCANNER

Das Benutzen des Handscanners darf nur registrierten Benutzern möglich sein. Hierzu muss die Applikation auf dem Handscanner eine Maske anbieten die es dem User erlaubt seine Benutzer-ID zu übergeben. Diese Maske soll keine weiteren Aktionen, außer dem Beenden der Applikation, erlauben. Der Benutzer soll über diese Maske den Laser des Handscanner anstoßen können und damit seine ID, die als Barcode auf seinem Ausweis platziert sein wird, scannen. Die Applikation soll diesen Barcode auswerten und die darin enthaltene ID zusammen mit einem Sicherheitstoken an die Webapplikation schicken. Zusätzlich ist auf jedem Gerät eine *DeviceID* hinterlegt die ebenfalls an die Webapplikation übergeben werden muss. Jeder Benutzer(UserID) kann einem Gerät(DeviceID) zugeordnet werde. Die Zuordnung soll durch Administrator innerhalb der in Punkt 2.1 beschriebenen Browseranwendung erfolgen.

Die Webapplikation muss eine Funktion anbieten die eine UserID, DeviceID und einen Sicherheitstoken entgegennimmt und diese Daten auf ihre Gültigkeit überprüft. Sie muss überprüfen

- ob die UserID existiert,
- ob der User diesem Gerät zugeteilt ist und
- ob der Sicherheitstoken gültig ist.

Sind alle diese Punkte erfüllt muss die Webapplikation eine Erfolgsmeldung an den Client zurücksenden. Ist einer der drei Punkte nicht erfüllt muss eine entsprechende Fehlermeldung an den Client gesandt werden.

Bekommt der Client die Erfolgsmeldung von der Webapplikation kann er die restlichen Funktionen der Clientapplikation für diesen Benutzer freigeben.

Ist der Benutzer für einen Zeitraum von X Minuten untätig, soll er von der Clientapplikation abgemeldet werden. Will er die Arbeit wieder aufnehmen muss er sich erneut anmelden. Der Wert X soll vom Administrator innerhalb der Browseranwendung definiert werden können. Nach der erfolgreichen Anmeldung soll die Clientapplikation automatisch eine Abfrage der gültigen Systemparameter an die Webapplikation senden. Der Zeitraum X soll unteranderem in diesen Systemparametern enthalten sein. Die Webapplikation soll daher eine Funktion anbieten die keine Parameter erwartet (UserID und Sicherheitstoken müssen immer mitgesandt werden) und die gültigen Systemparameter zurückgibt.

2.4 ÜBERPRÜFEN VON LAGERGÜTERN MIT DEM HANDSCANNER

Der Reifenhotelmitarbeiter sowie der angestellte des Kundencenters muss jederzeit in der Lage sein Informationen zu einem Lagergut via Handscanner einzuholen. Die Applikation auf dem Handscanner muss dem Benutzer daher eine Maske anbieten die den Scanner auslöst und die eingescannte Nummer des Lagerguts auswertet. Die Webapplikation muss eine Funktion bereitstellen die eine Einlagerungsnummer entgegennimmt und alle verfügbaren Informationen zu diesem Lagergut

zurückliefert. Diese müssen dem Benutzer übersichtlich angezeigt werden. Bei den Benötigten Informationen handelt es sich um folgende:

- Status,
- falls vorhanden, den vorgesehenen Lagerort,
- falls eingelagert, den Lagerort,
- offene Dienstleistungen

2.5 DURCHFÜHREN VON DIENSTLEISTUNGEN

Zum Abarbeiten von offenen Dienstleistungen muss der Handscanner dem Mitarbeiter eine Maske anbieten auf der ihm die offenen Dienstleistungen auf einer gewissen Ebene des Reifenhotels angezeigt werden. Das physische Reifenlager, das vom Kunden ja bereits erbaut und fertiggestellt wurde, hat vier Lagerebenen. Bevor der Mitarbeiter Dienstleistungen abfragt muss ihm die Applikation auf dem Handscanner die Möglichkeit geben eine Ebene auszuwählen. Sobald der Mitarbeiter eine Ebene ausgewählt hat, soll im Hintergrund eine Abfrage an die Webapplikation getätigt werden die die Nummer der Ebene entgegennimmt und entweder das erste Lagergut, das offene Dienstleistung hat, zurückliefert oder eine Fehlermeldung zurückgibt, weil auf dieser Ebene keine Dienstleistungen vorhanden sind. Wird ein Lagergut zurückgeliefert soll dem Benutzer die Einlagerungsnummer, der Lagerort und alle offenen Dienstleistungen angezeigt werden. Der Benutzer soll hier die Möglichkeit haben die Bearbeitung anzunehmen oder abzulehnen. Lehnt er die Bearbeitung ab, so bleibt der Status dieses Lagerguts unverändert. Versucht der Mitarbeiter danach noch einmal nach Dienstleistungen auf diese Ebene zu fragen, soll er wieder dasselbe Lagergut angezeigt bekommen, sofern dieses inzwischen nicht von einem anderen Mitarbeiter bearbeitet wurde. Es soll dem Mitarbeiter also nicht möglich sein, Lagergüter mit offenen Dienstleistungen zu übergehen.

Nimmt der Mitarbeiter die Bearbeitung an, so muss der Status des Lagerguts auf ‚Ausgelagert zur Dienstleistungsdurchführung' gesetzt werden. Hierzu muss die Webapplikation dem Client eine Funktion anbieten die Lagergut, Lagerort und die betroffenen Dienstleistungen entgegennimmt und im Erfolgsfall die Statusanpassung durchführt und eine entsprechende Meldung an den Client zurückgibt.

2.6 ABARBEITEN VON EINLAGERUNGEN (EINLAGERN VON LAGERGÜTERN)

Um ein Lagergut einzulagern muss es mit dem Handscanner erfasst werden und an die Webapplikation gesandt werden. Der Handscanner muss dem Mitarbeiter daher eine Input-Maske anbieten die den Laser des Scanners auslöst und den eingelesenen Barcode auswertet.

Der Benutzer soll über das Display zum Scannen des Lagerguts aufgefordert werden. Sofort danach muss dem Benutzer angezeigt werden ob dieses Lagergut bereits einen vorgesehenen Lagerort hat. Dazu kann im Hintergrund die Funktion, die in Punkt 2.4 definiert wurde, verwendet werden. Ist der Status des Lagerguts ‚*Einlagerung vorbereitet*' und ist ein vorgesehener Lagerort vorhanden soll dieser dem Benutzer eindeutig angezeigt werden. Sind für dieses Lagergut Dienstleistungen vorhanden, sollen auch diese angezeigt werden. Aus dem Scannen des Lagerguts resultiert die Aufforderung zum Scannen des Lagerorts. Diese Aufforderung muss den vorhandenen Informationen zum Lagergut angepasst sein. Ist weder ein Lagerort noch eine Dienstleistung vorhanden wird dem Mitarbeiter nur die Aufforderung zum Lagerort Scannen angezeigt. Ist ein vorgesehener Lagerort vorhanden darf der Scanner auch nur mehr diesen als Input akzeptieren. Dies soll in erster Linie Zeit sparen und unnötigen Aufrufen an die Webapplikation vorbeugen.

Sind das gescannte Lagergut und der gescannte Lagerort valide, müssen diese an die Webapplikation gesandt werden. Die Webapplikation muss daher eine Funktion anbieten die Lagergut und Lagerort entgegennimmt und gegen die ihr bekannten Daten validiert. Der Ablauf des Einlagerungsprozesses ist in Abbildung 1.1 dargestellt und wird in Absatz 1.2.1 erklärt. Die Einlagerungsfunktion der Webapplikation muss diesen Ablauf implementieren. Dies bedeutet, dass sie den empfangenen Lagerort gegen den Status des empfangenen Lagerguts validieren muss und im Falle einer Nichtübereinstimmung eine Fehlermeldung inklusive Erklärung zurückgibt. Entspricht der übergebene Lagerort dem Satus des Lagerguts, ist der Status auf die nächste Stufe gemäß dem Einlagerungsprozess zu heben und eine Erfolgsmeldung inklusive neuem Status an den Client zurückzugeben.

Ist der Status eines Lagerguts ‚*Einlagerung angenommen*', bei dem übergebenen Lagerort handelt es sich um einen Checkpoint und die Einlagerungsanforderung beinhaltet Dienstleistungen so müssen auch diese in der Erfolgsmeldung enthalten sein. Dem Benutzer muss dann über Buttons in der Einlagerungsmaske die Möglichkeit gegeben werden die Dienstleistungen anzunehmen oder abzulehnen. Um diese Information an die Webapplikation zu senden kann die Funktion die in Punkt 2.5 definiert wurde verwendet werden. Nimmt der Benutzer die Bearbeitung der Dienstleistungen an erfolgt die Status Anpassung des Lagerguts analog zu Punkt 2.5. Nimmt er die Bearbeitung nicht an so soll der Status auf ‚*Einlagerung vorbereitet*' aktualisiert werden.

Die Einlagerungsmaske der Applikation für den Scanner stellt eines der zentralen Elemente der Reifenhotelsoftware da. In Kombination mit den zugehörigen Funktionen der Webapplikation stellt sie das wichtigste Werkzeug für die Mitarbeiter des Reifenhotels dar. Eine zuverlässige, robuste Inputmaske ist für den reibungslosen Ablauf der Einlagerung daher genauso unerlässlich wie eine zeitnahe Verarbeitung aller Anfragen durch die Webapplikation.

2.7 ANLEGEN UND TERMINISIEREN VON AUSLAGERUNGEN

Den an das Reifenhotel angebundenen KC's muss die Möglichkeit gegeben werden eingelagerte Räder und Reifen auch wieder auszulagern. Dazu muss die Webapplikation eine Funktion anbieten die die Einlagerungsnummer und ein Datum entgegennimmt. Anhand der übergebenen Einlagerungsnummer soll dem passenden Lagergut nun das übergebene Datum als Auslagerungsdatum zugeordnet werden. Ein gültiges Auslagerungsdatum kann aus Sicht des Funktionsaufrufes in der Zukunft liegen (unbeschränkt) oder bis zu fünf Minuten in der Vergangenheit. Auslagerungsdaten die in der Zukunft liegen werden sollen von der Webapplikation als terminisierte Auslagerungen interpretiert werden. Auslagerungsdaten die zeitlich gesehen bereits vor dem Zeitpunkt des Funktionsaufrufs liegen, sollen von der Webapplikation als *ad-hoc-Auslagerungen* interpretiert werden. Diese Unterscheidung ist wichtig für die Verteilung von Prioritäten. Aufgerufen können diese Funktionen vom jeweiligen DMS des ein- beziehungsweise auslagernden KC's werden. Im Falle von MOTIONDATA funktioniert dieser Aufruf über einen Dienst der Veränderungen in der Einlagerungsdatenbank überwacht und Termine auf referenzierte Einlagerungsobjekte überprüft. Es ist dem jeweiligen DMS Hersteller allerdings freigestellt wann und wie diese Funktionen aufgerufen werden. Die Aufgabe der Reifenhotel Software ist hier lediglich die Bereitstellung der Webservicefunktionen. Die höchste Priorität sollen *just-in-time-Auslagerungen* haben. Das Anlegen einer solchen obliegt nicht dem DMS sondern soll nur den Benutzern der Reifenhotel-Browserlösung möglich sein. Die Browsermaske muss daher eine Inputmaske anbieten in der ein berechtigter Mitarbeiter die nötigen Daten für eine *just-in-time-Auslagerung* eingeben kann. Diese Daten sollen direkt ins System übernommen werden. Ist einem Lagergut ein Auslagerungsdatum zugeordnet muss in allen der genannten Fälle der Status des Lagerguts auf *‚Auslagerung angefordert'* gesetzt werden. Zusätzlich soll es möglich sein eine Auslagerung wieder zu stornieren. Hierfür muss die Webapplikation eine Funktion anbieten die die Einlagerungsnummer eines Lagerguts entgegennimmt und die Auslagerungsanforderung im System wieder entfernt. Ist das Lagergut bereits unterwegs muss diese Funktion dafür sorgen, dass das Lagergut seine Schritte wieder gemäß dem Einlagerungsprozess, also in die Gegenrichtung, durchläuft.

2.8 AUSLAGERUNGEN ANFORDERN, AUSLAGERN VON LAGERGÜTERN

Um tagesaktuell auslagern zu können muss dem Lagerpersonal die Möglichkeit gegeben werden fällige Auslagerungen einzusehen. Hierzu muss die in Punkt 2.1 beschriebene Browseranwendung eine Ansicht anbieten in der alle anstehenden Auslagerungen eingesehen, gefiltert und gesucht werden können. Des Weiteren muss es den Lagermitarbeitern möglich sein Auslagerungen über den Handscanner abzurufen. Hierzu muss die Applikation auf dem Handscanner eine Maske zur Verfügung stellen auf der die nächste Auslagerung für den jeweiligen Mitarbeiter angezeigt wird. Dem Benutzer soll beim Öffnen dieser Auslagerungsmaske die Möglichkeit geboten werden eine Auslagerung anzufordern oder eine bereits angenommen Auslagerung durch Scannen des Lagerorts zu bestätigen.

Zum Anfordern einer Auslagerung durch den Client muss die Webapplikation eine Funktion anbieten die die Benutzeridentifikation des Mitarbeiters übergibt und die Auslagerung in seiner Nähe mit der höchsten Priorität zurückgibt. Bei der Vergabe der Prioritäten geht hier *just-in-time-Auslagerung* vor *ad-hoc-Auslagerung*. Danach kommen terminisierte Auslagerungen in chronologischer Reihenfolge. Um den Standort des Mitarbeiters zu bestimmen soll diese Funktion die letzte Interaktion dieses Benutzers heranziehen und deren geographische Lage auswerten. Liefert die Webapplikation dem Mitarbeiter eine Auslagerung auf den Handscanner muss der Status des Lagerguts von ‚*Auslagerung angefordert*' auf ‚*Auslagerung angenommen*' gesetzt werden und die Auslagerung muss dem jeweiligen Benutzer zugeordnet werden. Dieser soll nicht in der Lage sein die Reihenfolge der Auslagerungen zu ändern. Eine weitere Abfrage an die Webapplikation soll ihm immer wieder die gleiche Auslagerung zurückgeben. Erst wenn er das Lagergut tatsächlich (physisch) aus dem Lagerort entfernt und dies, durch Scannen des Barcodes auf dem Lagergut, bestätigt, soll ihm die nächste Auslagerung angezeigt werden. Hierzu muss die Webapplikation eine Funktion anbieten die die Einlagerungsnummer des Lagerguts entgegennimmt und deren Status auf ‚*Auslagerung entnommen*' setzt, sofern sich dieser zuvor im Status ‚*Auslagerung angenommen*' befindet. Dieselbe Funktion soll in weiterer Folge auch dafür genutzt werden alle weiteren Schritte, die im Auslagerungsprozess in Punkt 1.2.2 beschrieben sind, entgegenzunehmen.

2.9 INVENTUR

Der Administrator des Reifenhotels soll in der Lage sein Inventurlisten anzulegen. Hierzu soll die Browseranwendung eine Maske anbieten in der eine solche eröffnet werden kann. Eine Inventurliste kann für das gesamte Reifenhotel angelegt werden oder nur für gewisse Abschnitte. Es können daher auch mehrere Inventurlisten angelegt werden. Dem Lagermitarbeiter muss eine Maske angeboten werden mit der er Lagergüter inventarisieren kann. Er muss in der Lage sein auf dieser Maske die Einlagerungsnummer und den Lagerort eines Lagerguts einzuscannen und diese mit einer Bestätigungstaste an den Webservice zusenden. Die Webapplikation muss hierfür eine Funktion anbieten, die Einlagerungsnummer und Lagerort entgegennimmt und einer Inventurliste zuordnet. Die Zuordnung zu einer Inventurliste erfolgt anhand des übergebenen Lagerorts. Ein Lagerort soll immer nur einem Abschnitt zugeteilt sein. Die Funktion muss dem Client lediglich eine Bestätigung zurückliefern, dass sie die Daten erhalten hat. Im Hintergrund muss geprüft werden ob die übermittelte Lagergut/Lagerort Kombination mit den Daten, die im System gespeichert sind, übereinstimmt. Ist dem nicht so muss dem Administrator eine Liste mit nicht übereinstimmenden Datensätzen angezeigt werden.

2.10 LAGERGÜTER UMLAGERN

Der Mitarbeiter im Reifenlager muss in der Lage sein, Lagergüter unkompliziert umlagern zu können. Hierfür soll ihm am Handscanner eine Maske angeboten werden, die es ihm ermöglicht die Einlagerungsnummer eines Lagerguts zu scannen. Im Hintergrund soll die Applikation am Handscanner

die, in Punkt 2.4 definierte, Funktion dazu benutzen den aktuellen Lagerort dieses Lagerguts abzufragen. Dieser soll dem Benutzer dann am Display des Handscanners angezeigt werden. Danach soll der neue Lagerort, auf den das Lagergut umgelagert werden soll, gescannt werden und mit einer Bestätigungstaste an die Webapplikation gesandt werden. Die Webapplikation muss hierzu eine Funktion anbieten die die Einlagerungsnummer des Lagerguts und den neuen Lagerort entgegennimmt und diese auf ihre Korrektheit überprüft. Im Erfolgsfall muss eine Meldung an den Client gesandt werden die die Umlagerung bestätigt. Im Fehlerfall ist eine entsprechende Fehlermeldung zu senden.

2.11 BENUTZERWECHSEL AM HANDSCANNER

Den Mitarbeitern im Lager muss es möglich sein sich von der Applikation am Handscanner manuell abzumelden. Hierzu muss die Applikation eine Funktion anbieten die die Abmeldung eines Benutzers erlaubt. Die Applikation soll dann in den Zustand vor der Benutzeridentifizierung wechseln. Danach soll wieder eine Identifizierung wie sie in Punkt 2.3 beschrieben ist nötig sein um die Funktionen des Handscanners zu nutzen.

3 WEBSECURITY

Sicherheitsrelevante Überlegungen für Webseiten unterscheiden sich im Großen und Ganzen nicht von denen für Webservices. Es gibt keinen wesentlichen Unterschied zwischen dem Web das für die Darstellung in einem Browser bestimmt ist und von Menschen ‚konsumiert' wird und dem ‚programmierbaren' Web das von einer Software angesprochen wird.[3] Über die genaue Bedeutung von REST und die Eigenschaften von RESTful Services wird später noch näher eingegangen. In Sicherheitsbelangen reicht es zu wissen das REST für „Representational State Transfer" steht und eigentlich nur eine Beschreibung oder Ableitung der Architektur des Webs und seinen Einschränkungen selbst ist.[4] Ein Service der als RESTful bezeichnet wird weist daher die gleichen Eigenschaften und auch Sicherheitslücken auf die jede beliebige Webseite und Webanwendung die mit HTTP-Requests arbeitet und die Methoden GET, POST, PUT etc. unterstützt. Es liegt daher auf der Hand, dass sich diese Arbeit mit dem Thema Websecurity ganz allgemein beschäftigen muss. Durch seine Effizienz und seinen geringen Overhead sind REST-Architekturen mittlerweile sehr weit verbreitet. Folglich muss jemand der einen RESTful Service bereitstellt auch gewährleisten, dass die nötigen Sicherheitsstandards eingehalten werden. In dieser Arbeit wird bewusst nicht von einem RESTful-Service sondern von einer HTTP-basierten Webapplikation nach Vorbild eines RESTful-Services gesprochen, da die Grundidee eines RESTful-Services an die Anforderungen des Kunden und die Abläufe innerhalb der Firma MOTIONDATA angepasst wurde.

Die Suche nach Mitteln und Wegen eine Anwendung sicher zu machen führt Entwickler immer wieder in Foren und alle möglich Blogs in denen die neuesten Erkenntnisse zum Thema Websecurity diskutiert werden. Die rasante Entwicklung auf diesem Terrain ist dafür verantwortlich, dass sich auch erfahrene Entwickler immer wieder gern austauschen um keine Neuigkeiten zu verpassen. Jede neue Entwicklung bietet eventuell auch eine Möglichkeit etablierte Sicherheitsmechanismen zu umgehen.

Die Frage was eine Architektur gewährleisten muss um allgemeinhin als sicher zu gelten zieht sich dabei durch alle Expertenbeiträge. Wörter die immer wiederkehren sind:

- ***Reliability*** – Glaubwürdigkeit, Zuverlässigkeit einer Anwendung
- ***Security*** – allgemein die Sicherheit eines Vorgangs
- ***Confidentiality*** – Vertraulichkeit der Daten (Verschlüsselung)
- ***Integrity*** – Vollständigkeit und Richtigkeit der Daten
- ***Atomicity*** – Atomare Vorgänge (kritische, heikle Vorgänge sollen in nicht unterbrochen bzw. pausiert werden)

Diese Begriffe, in unterschiedlicher Interpretation und Priorisierung, dienen zur Messbarkeit des erreichten oder angestrebten Sicherheitslevels. Sie werden herangezogen um die möglichen

[3] Vlg. Leonard Richardson & Sam Ruby; RESTful Web Services; O'Reilly 2007; (siehe Preface XV)

[4] Vlg. Mark Massé; REST API Design Rulebook; O'Reilly 2012; (siehe Seite 5 – '*Code-on-Demand*')

Vorgehensweisen zu vergleichen. Ein häufiger Vergleich auf den man stößt, wenn man sich mit diesem Thema beschäftigt ist der Vergleich zwischen RESTful Services und ihrer Security und der klassischen Webservice Security die mit SOAP-Nachrichten (Envelopes) arbeitet. Die einhellige Meinung zu diesem Vergleich ist, dass Webservices mit SOAP mehr Sicherheit liefern und ein weit mächtigeres API haben als das für REST Services der Fall ist. Allerdings ist diese Sicherheit auch mit viel Overhead verbunden. RESTful Services können wesentlich simpler implementiert werden, bringen in ihrer Rohform aber quasi überhaupt keine Sicherheitsvorkehrungen von selbst mit. Einige Kommentatoren gehen sogar so weit und sagen, dass es gar keine REST Security gibt. [5]Daher stellt sich gerade wenn es um Websecurity für HTTP-basiert Webservices geht, dem Entwickler oft die Frage welche Sicherheitsvorkehrungen denn überhaupt getroffen werden müssen und welche Implementierungsvariante diese Sicherheitsmaßnahmen unterstützt. Hier ist es sinnvoll Prioritäten für die geplante Anwendung zusetzen, um eine effiziente aber auch sichere Architektur konstruieren zu können. Um diese Prioritäten setzen zu können muss man aber verstehen warum die oben genannten Begriffe so wichtig für ein sicheres System sind. Man muss verstehen welche Bedrohungen für Anwendungen die mit einem Service kommunizieren und für den Service selbst bestehen.

Um die Bedrohungen, denen ein System ausgesetzt ist auszumachen versetzt man sich am besten in die Lage eines potentiellen Angreifers. Einem Angreifer reichen oft kleine Unachtsamkeiten seitens der Entwickler um entscheidende Informationen zu erlangen und in weiterer Folge erheblichen Schaden anzurichten. Um ein Gefühl dafür zu erlangen wie Kleinigkeiten, die ein gutgläubiger Entwickler oft nicht bedenkt, zu kapitalen Sicherheitslücken führen können, ist im Vorfeld zu evaluieren auf welche Weise ein Angreifer vorgehen könnte.

[5] Vlg. csoonline.com - Why REST Security Doesn't Exist (and what to do about it); Chris Comerford; Pete Soderling; Feb 24, 2010 7:00 AM; http://www.csoonline.com/article/2124905/identity-management/why-rest-security-doesn-t-exist--and-what-to-do-about-it-.html; letzter Aufruf am 29.08.2016

3.1 BEDROHUNGEN

Im weiteren Kapitel werden nun klassische Angriffspunkte einer Web Applikation aus der Sicht eines Angreifers beleuchtet und in den Kontext der Reifenhotel Anwendung gestellt.

3.1.1 BESCHAFFUNEN VON INFORMATIONEN

Um ein API sicher zu machen muss man vorgehen wie ein potentieller Angreifer. Wenn man sich in die Lage von jemandem versetzt, der dem Produkt Schaden zufügen will oder einen persönlichen Vorteil ziehen will, kann eine andere Perspektive auf die Eigenschaften eines API eingenommen werden. Im ersten Schritt, der nötig ist um ein API anzugreifen, gilt es möglichst umfangreiche Informationen darüber zu erlangen. Öffentliche APIs verfügen meist über eine sehr gute Dokumentation. Dies ist notwendig um sie außenstehenden Entwicklern zugänglich zu machen und es diesen so zu ermöglichen ein API in ihr System zu integrieren. Auf Basis einer Dokumentation können Schwachstellen von Systemen identifiziert werden und potentiellen Angreifern somit zukünftige Angriffspunkte offengelegt werden. Daher gilt es bereits im Designprozess besonderes Augenmerk auf eine sinnvolle Architektur zu legen. Man sollte sich gut überlegen wer welche Daten beziehen darf, welche Information geliefert wird und auch welche Fehlermeldungen ausgegeben werden.

Neben den offenen APIs gibt es auch solche die im Hintergrund einer Anwendung arbeiten und deren Verwendung nur für bestimmte Entwickler gedacht ist. Diese APIs sind in der Regel minimal, nur intern oder manchmal auch gar nicht dokumentiert. Dies bedeutet allerdings auf keinen Fall, dass es ist nicht möglich ist, die URLs, und damit die Einstiegspunkte, die dieses API anbietet, herauszufinden. Der Traffic zwischen einer Anwendung und einem API kann sehr leicht mitgelesen werden. Ein Angreifer müsste nur eine entsprechende Software z.B Wireshark installieren und dann die besagte Anwendung verwenden. Einen Anmeldevorgang durchspielen, eine simple Datenabfrage oder vielleicht sogar Inhalte hochladen, wenn es möglich ist. In einer Umgebung die nicht SSL-verschlüsselt ist hat der Angreifer nun sämtliche URLs und die Parameter die erwartet werden. Aber auch wenn die Verbindung SSL verwendet gibt es die Möglichkeit mit Proxy-Rekordern heraus zu finden wohin die Anfragen einer Anwendung gehen. Wenn die Domain einer API bekannt ist kann es ein Angreifer auch mit einem Brute-Force-Angriff versuchen. API Architekturen folgen oft einem gewissen Muster. Der Angreifer könnte es daher zum Beispiel mit ***[Domainname] + "/api"*** oder ***[Domainname] + "/api/v1"*** versuchen.

Mit dieser Information kann er beginnen seine eigenen Requests zu versenden. Dazu muss er nicht einmal Code schreiben. Im einfachsten Fall können die Requests in die Adressleiste eines herkömmlichen Browsers eingetippt werden. Oder man bedient sich eines REST-Clients. Mit einem REST-Client Tool (z.B. ARC - Advanced Rest Client von Google) kann man auch komplexere Requests abschicken. Selbst wenn der Angreifer noch keine Schwächen in der Architektur des API entdeckt hat, kann er versuchen mehr über den Code, der dahinter liegt, heraus zu finden, indem er zufälligen Input in seine Requests packt, um zu sehen wie das System reagiert. Wenn der Angreifer weiß welche Parameter das System erwartet kann er herausfinden wie mit unerwartetem Input umgegangen wird. Was passiert, wenn ein Zahlen-Wert erwartet wird und ein String-Wert versandt wird? Was passiert, wenn der Request ohne Parameter versandt wird? Wie sehen die Fehlermeldungen aus die

zurückkommen? Für den Entwickler gilt es Fehlermeldungen so schlicht wie möglich zu halten und dem Benutzer trotzdem eine sinnvolle Rückmeldung zugeben. Wichtig ist es hier keine Informationen über die Abläufe preiszugeben die hinter der API-Funktion liegen. Zum Beispiel dürfen keine Fehler die beim Parsen der Nachricht entstehen direkt zurückgeben. Der Angreifer kann sich so durch eine Vielzahl an verschiedenen Inputs ein Bild über die Funktionsweise des Parsers machen und etwaige Schwächen ausmachen. Selbiges gilt auch für SQL-Abfragen. Jede Information über die Struktur der Datenbank hilft dem Angreifer einen Injection-Angriff vorzubereiten. Dieser Punkt soll ein Verständnis dafür schaffen warum oft kleine Unachtsamkeiten in der Entwicklung entscheidend sein können, wenn es um die Sicherheit eines Systems geht. Der Technologie-Blog http://blog.smartbear.com gliedert die hier beschriebene Vorgehensweise in drei Stufen. In einem 2014 erschienen Artikel über das Hacken von APIs[6] definiert der Autor Ole Lensmar für das Sammeln von hilfreichen Informationen über das Backend einer Anwendung die folgenden Begriffe:

1.) **Fuzzing:**

Immer wieder zufälligen Input an ein API schicken bis man auf etwas stößt das einen Fehler auslöst.

Der Entwickler muss daher sichergehen, dass alle ausgelösten Fehlermeldungen einheitlich sind und nur die wirklich nötigen Informationen preisgegeben werden. Heikle Informationen zum dahinterliegenden System dürfen nicht in Fehlermeldungen wiedergegeben werden.

2.) **Invalid Input Attacks**

Hier versucht der Angreifer die mit Fuzzing gefunden Fehler näher zu beleuchten und mit verschiedenen Variationen möglichst viel heraus zu finden. Interessant ist hier unter anderem wie das System mit Input umgeht den es nicht erwartet. Dies können Parameter mit dem falschen Typ sowie ungültige Header sein.

Wie auch beim Fuzzing gilt für den Entwickler alle möglichen Fehlerfälle zu betrachten und zu hinterfragen.

3.) **Malicious Input**

Mit „schadhaftem“ Input wird in diesem Schritt versucht den zuvor gefundenen, unbehandelten Fehler so auszunutzen, dass ein Schaden am System entsteht.

---

[6] Vlg. http://blog.smartbear.com - SmartBear Blog; Ole Lensmar; November 2014;
http://blog.smartbear.com/readyapi/api-security-testing-how-to-hack-an-api-and-get-away-with-it-part-1-of-3/;
http://blog.smartbear.com/readyapi/api-security-testing-how-to-hack-an-api-and-get-away-with-it-part-2-of-3/
http://blog.smartbear.com/readyapi/api-security-testing-how-to-hack-an-api-and-get-away-with-it-part-3-of-3/
abgerufen am 19.04.2017

Der Entwickler muss sich hier vergewissern, dass es nicht möglich ist das System mit unerwartetem Input zum Absturz zubringen. Dazu müssen alle Eventualitäten bedacht werden die die Userinteraktion betreffen.

Die folgenden Unterpunkte werden sich mit klassischen Attacken auseinandersetzen die oft nur möglich sind weil ein Angreifer zuvor, mit den in Kapitel 3.1.1 beschriebenen Methoden, Informationen sammeln konnte.

3.1.2 SQL-INJECTION

Injection–Angriffe sind der Überbegriff für alle Angriffe bei denen Daten in eine Anwendung injiziert werden die die ursprüngliche Funktionsweise der Anwendung außer Kraft setzen, umgehen oder sogar beschädigen.

SQL-Injection Angriffe sind eine spezielle Untergruppe der Injection Angriffe. Sie richten sich gegen das nachgelagerte Datenbanksystem einer Anwendung. Ihre Relevanz verdeutlicht eine Studie des Ponemon Instituts im Jahr 2014.[7]Hier gaben 65 Prozent der untersuchten Organisationen an innerhalb der letzten zwölf Monate Opfer eines SQL–Injection Angriffs geworden zu sein. SQL-Injection Angriffe nutzen SQL–Anfragen einer Anwendung aus bei denen Userinput als Parameter an die dahinterliegende Datenbank weitergegeben wird.

Das Open Web Application Security Project (OWASP) liefert für eine SQL-Injection folgende Definition:

> "*A SQL injection attack consists of insertion or "injection" of a SQL query via the input data from the client to the application. A successful SQL injection exploit can read sensitive data from the database, modify database data (Insert/Update/Delete), execute administration operations on the database (such as shutdown the DBMS), recover the content of a given file present on the DBMS file system and in some cases issue commands to the operating system. SQL injection attacks are a type of injection attack, in which SQL commands are injected into data-plane input in order to effect the execution of predefined SQL commands*." [8]

Wie bereits erwähnt kann sich ein Angreifer mit den vorhin besprochenen Methoden Wissen über die einzelnen Komponenten eines API aneignen. Dieses wissen kann er nun also nutzen um beispielsweise über die Inputparameter einer Anwendung SQL - Abfragen abzusetzen. Dies soll anhand eines Beispiels

[7] Vgl. The SQL Injection Threat Study; Ponemon Institute LLC; April 2014
https://www.ponemon.org/local/upload/file/DB%20Networks%20Research%20Report%20FINAL5.pdf

[8] OWASP am 04/10/2016; https://www.owasp.org/index.php/SQL_Injection; abgerufen am 27.09.2016

aus der Reifenhotelsoftware demonstriert werden. Angenommen der Funktionsaufruf aus Kapitel 2.3 zur Registrierung eines Benutzers würde folgendermaßen aussehen:

https://dev-rh.motiondata.at/Service/V1/Benutzer?[BenutzerID]&[DeviceID]

Mit dem Handscanner soll der Benutzer die Benutzer-ID auf seinem Ausweis scannen die dem Request als ID angefügt wird. Der Benutzer hat die Benutzer-ID ‚4711', der Scanner den er benutzt hat die Device-ID ‚3326'. Der Request den der Reifenhotel-Service erhält würde wie folgt aussehen:

https://dev-rh.motiondata.at/Service/V1/Benutzer?benutzerId=4711&deciceId=3326

Mit diesen Parametern stellt die Reifenhotelsoftware folgende Abfrage an die Datenbank:

```
SELECT * FROM Benutzer where BenutzerID = '4711' and DeviceID = '3326'
```

Dies sollte nur ein Ergebnis zurück liefern, wenn es tatsächlich einen Benutzer gibt, der diese beiden Bedingungen erfüllt. Was passiert aber, wenn jemand der keine gültige Benutzer-ID besitzt folgenden Barcode einscannt:

Abbildung 3.1 - Injection Barcode
Barcodes können in beliebigem Format jederzeit online generiert werden z.B: unter
http://barcode.tec-it.com/de

Der daraus folgende Request:

https://dev-rh.motiondata.at/Service/V1/Benutzer?'%20or%20'1'=1;--&3326

würde daher diese Abfrage an die Datenbank stellen:

```
SELECT * FROM Benutzer where BenutzerID = '' or 1=1;-- and DeviceID = '3326'
```

Da 1=1 immer wahr ist und die Frage nach der Device-ID gleich auskommentiert wurde liefert diese Anfrage alle Benutzer des Systems als Antwort. Das kann natürlich nicht im Sinne des Entwicklers sein. Ein Angreifer könnte so alle Userdaten auslesen oder sich mit dem Handscanner unbefugt Zugang zu den weiteren Funktionen des API verschaffen. Dieser Input ist nur eine von mehreren Zeichenketten die das Umgehen eines klassischen Logins ermöglichen würde.

Hat der Angreifer Wissen über die Benennung einzelner Datenbank-Tabellen, könnte er auch folgenden Parameter als Benutzer-ID verwenden:

```
1111';Drop Table Benutzer;--
```

Diese Abfrage würde zwar eher keinen Benutzer mit der ID 1111 finden, jedoch würde der nächste SQL-Command die gesamte Tabelle Benutzer löschen, was unweigerlich zu einem Zusammenbruch des

Systems führen würde, da sich kein User mehr einloggen könnte und auch alle anderen Abfragen, auf die Tabelle Benutzer, eine Fehlermeldung produzieren würden. Dies sind nur zwei von vielen Beispielen für Dinge die ein Angreifer mit SQL-Injections machen kann. Die Frage warum, wozu und vor allem durch wen so etwas geschehen sollte ist dabei nebensächlich. Als Entwickler wäre es grob fahrlässig diese potentiellen Sicherheitslücken nicht zu schließen. Dabei sei an dieser Stelle darauf hingewiesen, dass es noch viele weitere Varianten und Szenarios gibt wie man ungefilterte SQL-Inputs ausnutzen kann. Die genaue Erläuterung dieser Angriffsvektoren und den Szenarios beziehungsweise Rahmenbedingungen die für deren Erfolg nötig sind würde allerdings den Rahmen dieses Kapitels überschreiten. Zur Veranschaulichung der Auswirkungen von SQL-Injections sollten die beiden genannten Beispiele ausreichen. Sie sollten jedem Entwickler klarmachen, dass validieren von SQL-Inputs ein absolutes Muss für jede Webapplikation ist.

3.1.2.1 VORKEHRUNGEN

Als adäquates Mittel gegen solche Attacken sind folgende Maßnahmen zu nennen:

1.) **Datenbank Zugriffsrechte sinnvoll vergeben**

Als erste Maßnahme sollte man genau regeln wer welche SQL-Befehle ausführen darf. Befehle wie 'Drop Table...' oder 'Alter Table...' sollte prinzipiell nur von Benutzern mit Administratorrechten ausgeführt werden dürfen. Der Datenbankzugriff aus einer Webapplikation heraus sollte nie über den Zugang des Administrators oder eines anderen privilegierten Datenbankusers geschehen. Sinnvoll ist es hier das sogenannte 'Least Privilige Priciple' [9] anzuwenden. Dabei hat jeder Zugriff nur die Rechte die er braucht um genau die Aktionen auszuführen für die er zur Verfügung gestellt wird. Schreibende und lesende Zugriffe können so getrennt werden in dem man dafür jeweils eigene Datenbankbenutzer mit den jeweiligen Privilegien zur Verfügung stellt.

2.) **Input validieren:**

Als Entwickler sollte man bei jedem Zugriff auf die Datenbank bedenken woher die jeweiligen Parameter kommen und welchen Restriktionen man diese unterziehen kann bevor sie weitergegeben werden. Des Weiteren gibt es mittlerweile in fast jeder Programmiersprache Bibliotheken, die gute und zuverlässige Funktionen zur Validierung von SQL-Parametern anbieten. So kann verhindert werden das fragwürdiger Input überhaupt bis zur Datenbank vordringt.

3.1.2.2 BEISPIEL AUS DER REIFENHOTEL SOFTWARE

Im konkreten Fall des Reifenhotels lässt sich der Input der über den Scanner zur Webapplikation gelangt sehr stark eingrenzen. Die verwendeten Einlagerungsnummern sind immer 10 Zeichen lang,

[9] Vlg. http://phpsecurity.readthedocs.io/en/latest/Injection-Attacks.html; Letzter Aufruf am 10.10.2016

während die Lagerplatz Bezeichnungen immer sieben Zeichen lang sind. Lagerplatz Bezeichnungen müssen außerdem immer dem definierten Muster entsprechen:

[0] Großbuchstabe
[1] Zahl
[2] Zahl
[3] Großbuchstabe
[4] Zahl
[5] Zahl
[6] Großbuchstabe

Hier lässt sich zumindest im Falle des Zugriffs über den Scanner sehr genau definieren wie Input aussehen muss. Das Resultat daraus ist, dass der Scanner schon beim Auswerten eines gescannten Barcodes entscheidet ob ein Input überhaupt zugelassen ist. Eine SQL-Injection über den Scanner wie sie im Beispiel oben dargestellt ist, wäre also gar nicht möglich.

Innerhalb des serverseitigen Codes werden SQL – Parameter mithilfe der im ***CodeIgniter-*** Framework enthaltenen ‚Prepared Statements' und der Funktionen ‚`escape_str`' validiert oder ‚sanitized', wie der Fachbegriff aus dem Englischen lautet.

```
1.  public function get_standort($standort_id) {
2.          $this->db->from("rho_standort");
3.          $this->db->where("barcode", $standort_id);
4.          $query = $this->db->get();
5.          if ($query->num_rows() == 1) {
6.              return $query->row();
7.          }
8.          return FALSE;
9.      }
```

Codesegment 3.1: Beispiel aus dem Reifenhotel: Erstellungen einer SQL – Abfrage zum Abfragen eines Standorts. Dabei werden die von Codeigniter zur Verfügung gestellten Funktionen db->from und db->where verwendet.

```
1.  function escape_str($str, $like = FALSE){
2.
3.      if (is_array($str)){
4.
5.          foreach ($str as $key => $val) {
6.                  $str[$key] = $this->escape_str($val, $like);
7.              }
8.              return $str;
9.          }
10.
11.     if (function_exists('mysql_real_escape_string') AND  is_resource($this-
    >conn_id)){
12.
13.         $str = mysql_real_escape_string($str, $this->conn_id);
14.     }
15.     elseif (function_exists('mysql_escape_string')){
16.         $str = mysql_escape_string($str);
17.     }
18.     else{
19.         $str = addslashes($str);
20.     }
21.     // escape LIKE condition wildcards
22.     if ($like === TRUE){
23.         $str = str_replace(array('%', '_'), array('\\%', '\\_'), $str);
24.     }
25.     return $str;
26. }
```

Codesegmenet 3.2: Die Funktion escape_str() aus der CI_DB_mysql_driver Klasse die das Codeigniter Framework zur Verfügung stellt.

Verwendet wird die Funktion *escape_str* tatsächlich in der Zeile *$query = $this->db->get();* von Codesegement 3.2. Hier wird der zuvor geformte SQL-String validiert bevor die Abfrage tatsächlich abgeschickt wird. Daher kann kein unzulässiger Input bis zur Datenbank durchdringen.

Zusätzlich dazu werden die Datenbankrechte aller Reifenhotel-User eingeschränkt. Alle Benutzer die im Reifenhotel arbeiten, haben bezüglich der dahinterliegenden Datenbank die gleichen Rechte. Ihnen sind die SQL-Befehle SELECT, INSERT, UPDATE erlaubt. Auch die Administratoren, die über einen weit größeren Funktionsumfang innerhalb der Browseranwendung verfügen und haben nicht mehr Datenbankrechte. Das Erstellen, Erweitern und Löschen von Tabellen ist nur den Backendentwicklern erlaubt.

3.1.3 CROSS-SITE SCRIPTING

Bei Cross-Site-Scripting Angriffen wird schadhafter Code in eine Anwendung eingeschleust. Bei der Weiterverarbeitung der Ressource die den schadhaften Code enthält kann es dann zum eigentlichen Angriff kommen. Somit könnte sowohl das verarbeitende Backend von dem Angriff betroffen sein, als auch der Benutzer der die schadhafte Ressource aufruft. Cross-Site-Scripting Angriffe, auch XSS–Attacken genannt, machen sich den Umstand zu Nutze, dass viele Webbrowser Skriptsprachen unterstützen die in den HTML–Code einer Webseite eingebettet sind, damit diese vom Client lokal ausgeführt werden. Das Einbetten von Skripten zum dynamischen Aufbau von Webseiten macht diese flexibler und erweitert deren Funktionalität. Dies ist auch der Grund warum diese optionale Vorgabe der Webarchitektur, auch Code-on-Demand genannt, trotz der offensichtlichen Gefahr die davon ausgeht, soweit verbreitet ist. Bedingung für die Nutzung solcher Skripte ist eine gemeinsame Skriptsprache die vom Browser auch interpretiert werden kann. Die wohl bekanntesten Vertreter davon sind JavaScript, Java Applets und Flash.[10]

Erste Voraussetzung für einen Erfolgreichen Angriff ist, dass die schadhafte Ressource von der Webanwendung als valider Input angesehen wird. Wenn eine Webseite ihren Benutzern zum Beispiel erlaubt Dateien hochzuladen ohne den Input zu überprüfen, könnten Benutzer auch Dateien hochladen die schadhafte Skripte enthalten. Diese könnten den Server auf dem der Service läuft ausspionieren oder beschädigen. Natürlich kann es auch sein, dass der schadhafte Inhalt nicht den Server angreift, sondern einfach darauf auf ist, dass er am Server gespeichert wird und später von anderen Usern aufgerufen wird. Die gängige Fachliteratur zum Thema Cross-Site-Scripting unterscheidet zwischen ‚***Reflected XSS***' oder zu Deutsch ‚***Reflexives XSS'*** und ‚***Stored XSS***', zu Deutsch ‚***persistentes XSS'***.

Ein simples Beispiel für reflexives XSS wäre eine Webseite die dem Benutzer ein Suchfeld anbietet mit dem er nach Lagergütern im Reifenhotel suchen kann. Der Request den diese Seite absendet würde dann wie folgt aussehen:

https://dev-rh.at/index/reifenhotel/einlagerungen.php?suche=10001691

Und das Ergebnis der Suche könnte so angezeigt werden:

```
<p>Kein Ergebnis für 10001691 gefunden</p>
```

Wird allerdings nicht überprüft was der Benutzer ins Suchfeld eingibt, könnte dieser auch folgenden Request absetzen:

https://dev-rh.at/index/reifenhotel/einlagerungen.php?suche=<script>alert('XSS');</script>

Das würde zu folgender Darstellung im Browser führen:

[10] Vlg. Mark Massé; REST API Design Rulebook; O'Reilly 2012; (siehe Seite 5 – '*Code-on-Demand*')

```
<p>Kein Ergebnis für <script>alert('XSS');</script>gefunden</p>
```

Der Browser würde das Skript sofort beim Laden der Seite ausführen. Um zu verstehen wo die Gefahr beim reflexiven XSS liegt wird zunächst die Herangehensweise von Snyder, Myer und Southwell betrachtet. Diese kategorisieren 2010 in Ihrem Buch Pro PHP Security[11] XSS Attacken als

1.) ‚Remote Side to Application Side'
oder
2.) ‚Application Site to Same- or Remote Site'.

Bei der ersten Variante erfolgt der Angriff von außen über eine manipulierte Ressource in einem E-Mail oder einer anderen Webseite. Das Opfer soll dazu gebracht werden auf einen Link zu klicken, ein Bild zu laden oder ein Formular abzusenden. Eine solche Aktion könnte, eventuell sogar im Hintergrund, ohne dass es das Opfer merkt, einen reflexiven XSS–Angriff durchführen. Der Hintergedanke bei diesem Vorgehen ist meistens, dass das Opfer zur Zeit des Angriffs über eine gültige Session in der angegriffenen Anwendung verfügt. Der Charakter des Angriffs hängt dann natürlich vom Inhalt des Skripts ab das in den Link eingebettet wurde.

3.1.3.1 THEORETISCHES BESIPIEL AUS DER REIFENHOTELSOFTWARE:

Um die Möglichkeiten einer reflexiven XSS Attacke zu veranschaulichen wurde folgendes Beispiel von Stuttard und Pinto[12] für die Reifenhotel Software adaptiert:

1.) Das Opfer loggt sich mit seinen Zugangsdaten in die Reifenhotel Webapplikation ein.
2.) Der Angreifer hat eine reflexive XSS-Lücke gefunden und einen manipulierten Link erstellt, der wie folgt aussieht:
https://dev-rh.at/index#/reifenhotel/einlagerungen/suche.php?=Lagernummer"><script>document.location=http://angreifer.seite/steal.php?cookie= + document.cookie </script>
3.) Das Opfer klickt auf den Link oder ruft ihn unwissentlich (versteckte Einbindung als IFrame etc.) auf.
4.) Der Sever sendet die Seite mit Schadcode an das Opfer.
5.) Der Browser des Opfers führt folgenden Schadcode aus:
<script>document.location=http://angreifer.seite/steal.php?cookie= + document.cookie </script>
6.) Dadurch wird die Session-ID an den Angreifer gesendet

[11] Vlg. Snyder, Myer, Southwell; Pro PHP Security – From Application Security Principles to the Implementation of XSS Defense; Apress 2010; (siehe Seite 45)

[12] Vlg. Stuttard, Pinto; The Web Application Hackers Handbook – Finding and Exploiting Security Flaws; Wiley Publishing, Inc. 2010; (siehe Seite 436)

7.) Durch die Kenntnis der Session-ID kann Angreifer die Session übernehmen („Cookie-Replay-Attacke“)

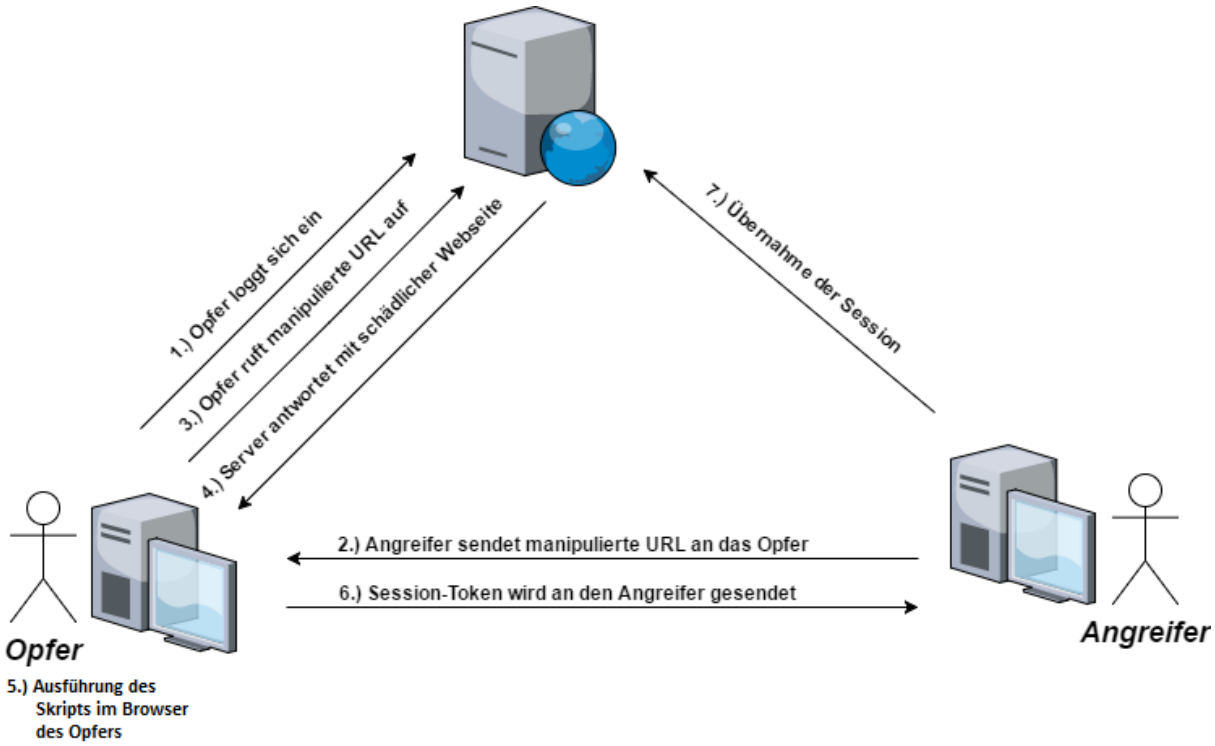

Abbildung 3.2 - Reflexives XSS
Grafische Darstellung eines reflexiven XSS – Angriffes aus dem Buch Hacking im Web von Tim Schäfers, adaptiert für die Reifenhotelanwendung der Firma MOTIONDATA

Die zweite Variante, nämlich die Variante ***‚Application Site to Same or Remote Site‘*** entspricht dem was auch als ***‚Stored Cross-Site-Scripting‘*** oder persistentes XSS bezeichnet wird. Beim persistenten XSS wird der schadhafte Code irgendwo innerhalb der Anwendung hinterlegt. Sie nutzt so zu sagen das Vertrauen des Benutzers in die Anwendung aus. Voraussetzung dafür ist, dass das schadhafte Skript dauerhaft, zum Beispiel in einer dahinterliegenden Datenbank, gespeichert werden kann.

Persistentes XSS kommt häufig in Webanwendungen vor die die Interaktion zwischen Endbenutzern unterstützen. Es tritt auf wenn die Eingabe eines Endbenutzers innerhalb der Anwendung gespeichert wird und dann einem anderen Benutzer angezeigt wird, ohne dass dazwischen entsprechend auf unzulässigen Inhalte gefiltert wird.[13]

[13] Vlg. Stuttard, Pinto; The Web Application Hackers Handbook – Finding and Exploiting Security Flaws; Wiley Publishing, Inc. 201; (siehe Seite 439)

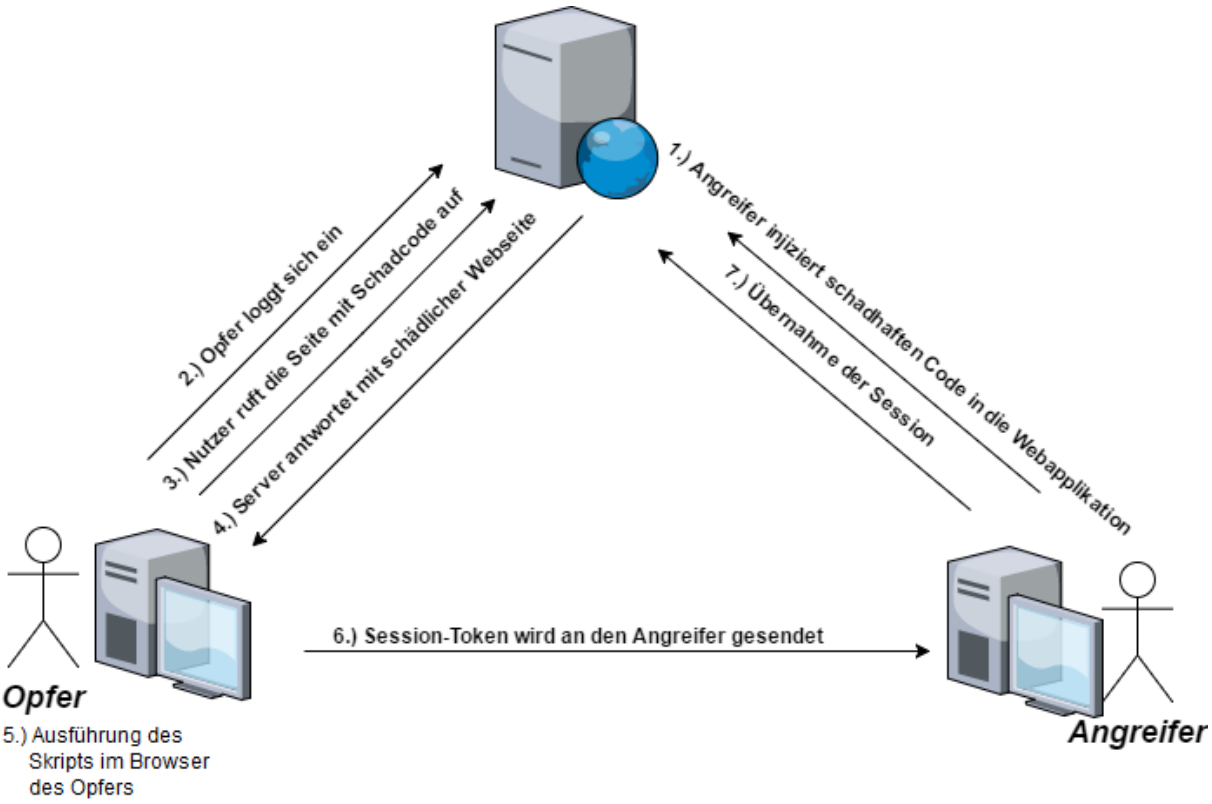

Abbildung 3.3 - Persistentes XSS

Grafische Darstellung eines persistenten XSS – Angriffes aus dem Buch Hacking im Web von Tim Schäfers, adaptiert für die Reifenhotelanwendung der Firma MOTIONDATA

3.1.3.2 THEORETISCHES BESIPIEL AUS DER REIFENHOTELSOFTWARE:

Ein Angreifer terminisiert eine Einlagerung über die Webanwendung des Reifenhotels oder über Schnittstelle die Ein- und Auslagerungen an das DMS sendet. Bei diesem Vorgang wird das Feld Bemerkung mitübergeben. Dies ist ein Textfeld mit 2000 Zeichen. In dieses Feld fügt der Angreifer ein Skript ein das von einem Browser interpretiert werden kann. Ein anderer Lagermitarbeiter betrachtet über die Webanwendung die offenen Einlagerungen und klickt auf genau die Einlagerung die als Bemerkung das schadhafte Skript enthaltet. Beim Anzeigen der entsprechenden Webseite würde der Browser den schadhaften Code bereits ausführen. Dies würde man als ***Stored Cross-Site Scripting*** oder ***persistentes XSS*** bezeichnen.

Die Gefahr die von beiden Varianten, egal ob sie in ***reflexiv*** oder ***persistent***, ***Remote Side to Application Side*** oder ***Application Site to Same or Remote Site*** unterteilt werden ist, dass ein Skript oder Code-on-Demand vom Client ausgeführt wird das weder von der Anwendung noch vom Benutzer, sondern von einem unbekannten Dritten kommt.

3.1.4 SESSION ANGRIFFE

Ein gängiges Szenario bei Webanwendungen ist, dass sich der Benutzer zu allererst einmal einloggen muss um gewisse Vorgänge durchführen zu können. Wie kann die Anwendung allerdings feststellen ob sich ein Benutzer bereits eingeloggt hat, wenn doch HTTP ein zustandsloses Protokoll ist und jede Anfrage für sich gesondert abgehandelt wird? Der Client müsste eigentlich bei jeder Anfrage die gesamten Benutzerdaten mitsenden, damit die Anwendung ihn identifizieren kann. Um dies zu vermeiden werden Sessions verwendet.

Über eine Session-ID kann die Anwendung eine Anfrage einem Benutzer zuordnen. Sie ist quasi eine temporäre Bestätigung, dass es sich bei dem Absender um den Benutzer XY handelt und dieser sich zuvor mit seinem Passwort authentifiziert hat. Je nach Implementierung, beinhaltet eine Session auch mehrere Daten. Zum Beispiel Daten zu einer laufenden Interaktion die über mehrere Schritte des Benutzers abgehandelt wird. Diese Informationen werden normalerweise in einer temporären Datei am Server abgespeichert. Diese Datei bekommt eine Session-ID, bestehend aus einer Zufallszeichenkette und dem Initialisierungszeitpunkt der Sitzung. Diese ID wird dem Client bei der Authentifizierung übermittelt und in weiterer Folge vom Client wiederum bei jeder weiteren Anfrage mitgesandt. Durch das Mitsenden der ID durch den Client weiß der Server, dass er die Anfrage einer solchen temporären Datei und damit auch vorangegangenen Anfragen zuordnen kann.[14]

Die wohl gängigste Methode die Session-ID zu übermitteln ist Cookies zu setzen und diese über das HTTP-Header Feld zu übermitteln. Des Weiteren nennt Schäfer[15] noch die Möglichkeit die Session-ID innerhalb von Formularen über die POST-Methode (eventuell auch mit einem versteckten Feld) zu senden. Außerdem gibt es die Möglichkeit die Session-ID als URL–Parameter zu übergeben. Diese Variante ist zwar simpel aber auch unsicher. Sie ermöglicht unter anderem Session-Fixation (siehe Unterpunkt 3.2.4.3)

Diese Einleitung soll veranschaulichen warum Session-Management für moderne Webanwendungen nahezu unumgänglich ist und welche Sicherheitsrisiken eine fehlerhafte Implementierung des Session-Managements mit sich bringen kann. In den folgenden Unterpunkten sollen populäre Session-Angriffe kurz beschrieben werden.

3.1.4.1 MAN-IN-THE-MIDDLE-ANGRIFF

[14] Vlg. Snyder, Myer, Southwell; Pro PHP Security – From Application Security Principles to the Implementation of XSS Defense; Apress 2010; (siehe Seite 93 ff.)
[15] Hacking im Web; Tim Philipp Schäfers; Franzis Verlag; 2016; (siehe 103 ff.)

Für einen Man-in-the-Middle-Angriff braucht der Angreifer Zugang zum Netzwerk des Opfers. Dies kann der Fall sein, wenn es sich um ein öffentliches Netzwerk handelt oder ein Netzwerk das nicht ausreichend gesichert ist. Welche Gegebenheiten herrschen müssen um diesen Zugang zu haben soll hier nicht weiter erläutert werden, da dies in erster Linie ein Thema der Netzwerksicherheit ist und nicht der Anwendung selbst. Dem Entwickler der Anwendung muss aber bewusst sein, dass es durchaus möglich ist, dass einer seiner Benutzer über ein solches Netzwerk kommuniziert und dementsprechende Vorkehrungen zu treffen sind. In der unten angeführten Grafik wird gezeigt wie eine POST Anfrage an die Reifenhotelanwendung aussieht, wenn man sie mit dem Sniffing-Tool Wireshark abfängt:

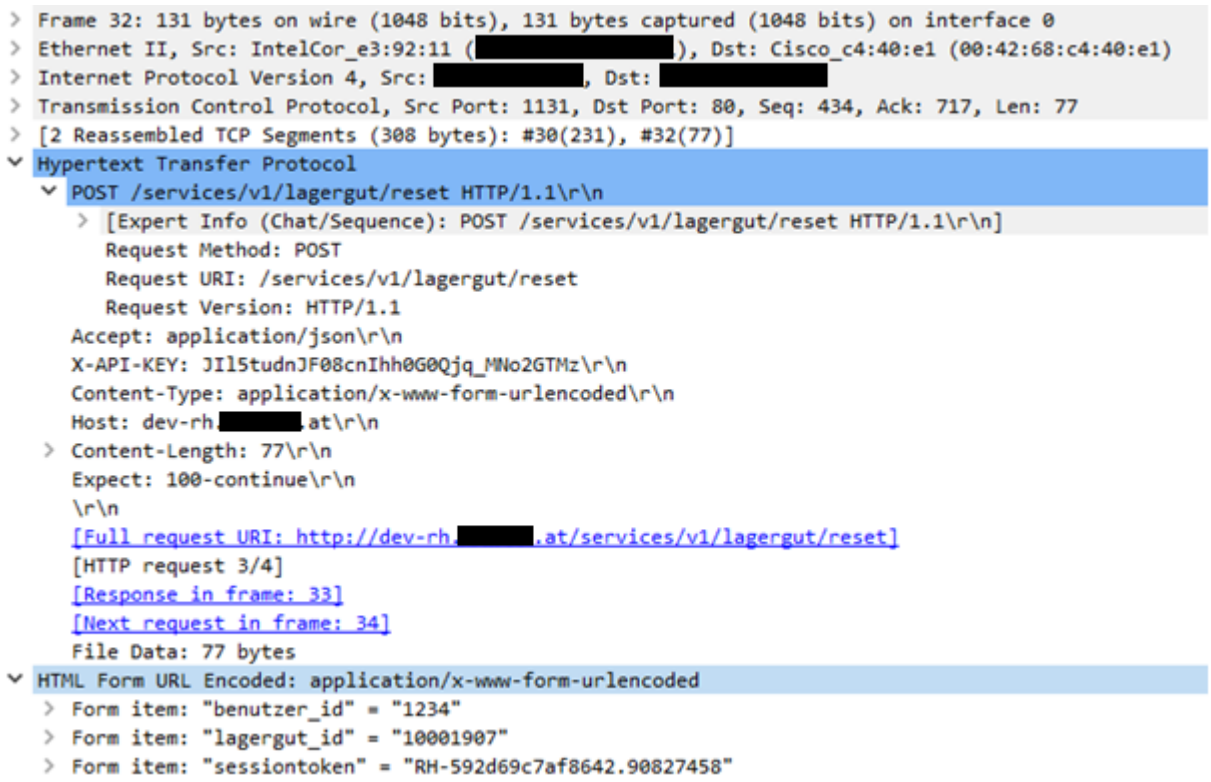

Abbildung 3.4 - Kommunikation zwischen Client und dem Webserver

Unverschlüsselte Kommunikation zwischen Client und dem Webserver der Reifenhotelanwendung, aufgezeichnet mit der Analysesoftware Wireshark [https://www.wireshark.org/]

Wie man sehen kann handelt es sich hier um keine verschlüsselte Kommunikation. Der Angreifer sieht welcher URL angesprochen wird, welche Methode verwendet wird und er sieht alle Header und alle Parameter inklusive des Sessiontokens. Somit hat der Angreifer alle Information um sich gegenüber der Anwendung auszuweisen und könnte einen sogenannten Cookie-Replay-Angriff versuchen.[16]

Prinzipiell lässt sich das Mitlesen und Manipulieren von Daten verhindern, wenn die Kommunikation zwischen Client und Server verschlüsselt wird. Voraussetzung dafür ist, dass ein sicheres

[16] Vlg. Hacking im Web; Tim Philipp Schäfers; Franzis Verlag; 2016; (siehe 92 ff.)

Verschlüsselungsprotokoll verwendet wird und darauf geachtet wird die aktuellste Version davon zu benutzen.[17]

Um die Gefahr eines solchen Cookie-Replay-Angriffs zu reduzieren sollten Sessions immer sofort ungültig sein sobald sich ihr Inhaber von der Anwendung abmeldet. Aber auch wenn sich der Benutzer nicht abmeldet sollte eine Session nach einer gewissen Zeit der Inaktivität für ungültig erklärt werden.

Im Falle der Reifenhotelanwendung gilt für den Handscanner, sowie für die Browseranwendung ein Timeout von 15 Minuten. Ist der Mitarbeiter für diese Zeit untätig läuft seine Session aus und er muss sich erneut mit seinem Benutzernamen und Passwort anmelden, wodurch eine neue Session-ID generiert wird. Auf Wunsch des Kunden wurde dieses Timeout allerdings so eingerichtet, dass sie vom Administrator einstellbar ist. Es liegt daher in dessen Ermessen einen guten Kompromiss zwischen Sicherheit und Usability zu finden. Es muss aber davor gewarnt sein sich als Entwickler in so einer Situation komplett aus der Verantwortung zu ziehen. Wenn nötig muss ein solcher Entscheidungsträger hier erst über die etwaigen Sicherheitsrisiken aufgeklärt werden, da sonst aus Gründen der Benutzerfreundlichkeit wahrscheinlich ein möglichst langes Timeout gewählt werden würde.

3.1.4.2 SESSION-HIJACKING

Wenn ein Angreifer es schafft einer gültigen Session-ID habhaft zu werden, während das Opfer diese noch benutzt bezeichnet man diesen Vorgang als Session-Hijacking. Dies kann auf mehrere Arten passieren. Eine Möglichkeit wäre es die Session-ID mittels JavaScript zu stehlen das über Cross-Site-Scripting auf die Seite eingeschleust wurde[18] (siehe Unterpunkt 3.1.3). Ein weiterer Angriffspunkt wäre ein schlechter Algorithmus zur Generierung der Session-ID. Ein Generierungsalgorithmus muss für jede neue Session eine eindeutige Session-ID generieren. Dabei darf der Algorithmus für den Angreifer aber auf keinen Fall vorhersehbar sein. Ein Angreifer könnte ja durch Analysieren von gültigen Session-IDs (zum Beispiel seinen eigenen) herausfinden wie sie generiert werden. Wurde der Generierungsalgorithmus einmal durchschaut und ist jemand in der Lage den Vorgang der Generierung zu rekonstruieren, ist das Session Management nicht mehr sicher.

In der Reifenhotelanwendung werden Session-IDs mithilfe von GUIDs (Globally Unique Identifier), auch UUIDs (Universally Unique Identifier) genannt, erstellt.[19]

3.1.4.3 SESSION-FIXATION

[17] Vlg. Sebastian Kübeck; Web-Sicherheit - Wie Sie Ihre Webanwendung sicher vor Angriffen schützen; MITP München; 2011; (siehe Seite 94 ff.)
[18] Vlg. Stefan Kunz, Stefan Esser; PHP – Sicherheit: PHP/MySQL-Webanwendung sicher programmieren; dPunkt Verlag 2008 (siehe Seite 180 ff.)
[19] Network Working Group; Juli 2005; A Universally Unique IDentifier (UUID) URN Namespace; http://www.ietf.org/rfc/rfc4122.txt; letzter Aufruf am 02.01.2018

Im Gegensatz zum Session-Hijacking versucht der Angreifer bei der Session-Fixation nicht die Session-ID seines Opfers zu erraten, sondern ihm eine Session-ID aufzuzwingen. Eine Session-ID die er bereits kennt. Entweder weil er sie sich selbst bei einer Anwendung eingeloggt hat und nun eine gültige Session kennt oder eine zufällig generierte Session-ID von der er weiß, dass sie die dem Muster der Session-IDs einer Anwendung entspricht. Angenommen diese bekannte Session-ID wäre 1234. Mit dieser ID bastelt sich der Angreifer eine Anfrage die wie folgt aussehen könnte:

http://randomonlinestore.com/login.php?PHPSESSIONID=1234

Den Link mit dieser Anfrage lässt der Angreifer seinem Opfer zukommen. Per Email oder einem Forums Eintrag mit folgendem Text: „Sonderangebote für Bestandskunden! Einfach anmelden und sparen.“ (nur als Beispiel wie ein gutgläubiger Benutzer dazu gebracht werden könnte so einen Link an zu klicken). Der Benutzer klickt auf den Link und meldet sich an. Er übermittelt daher seiner Benutzernamen und sein Passwort und verfügt damit über eine gültige Session. Die Session-ID kennt der Angreifer aber schon. Er kann die Session damit übernehmen und sich als das Opfer ausgeben.

Dies Vorgehensweise kann erfolgreich sein, wenn eine Anwendung die Session-ID als ***$_GET*** – Parameter erwartet und diese daher in die URL eingebaut werden kann. Konkret kann es sein, dass eine Anwendung Cookies verwendet, User die Cookies ablehnen aber nicht ausschließen möchte und daher zusätzlich auch den ***$_GET*** - Parameter überprüft und gegebenenfalls verarbeitet sollte die Verwendung von Cookies unterbunden worden sein.

Um Session-Fixation zu vermeiden ist es zu empfehlen die Session-ID nicht in der URL zu übergeben, sondern als Parameter in der Payload einer POST – Anfrage. Da dies nicht immer möglich ist, wird in der Reifenhotel Anwendung bei jedem Anmeldevorgang eine neue Session-ID generiert. Die Methode zum Anmelden des Benutzers ist jedoch eine POST – Anfrage. Eine eventuelle Session-ID in der Payload eben dieser Methode würde von der Anwendung außerdem einfach ignoriert werden.

3.1.4.4 CROSS-SITE REQUEST FORGERY

Beim Cross-Site Request Forgery wird versucht gültige Sessions und aktive Logins eines anderen Benutzers zu nutzen um Schaden anzurichten oder einen Vorteil zu erzielen. Der Angreifer platziert den gewünschten (schadhaften) Request so, dass das Opfer potentiell dann darüber stolpert, wenn er gerade über eine aktive Session in der Zielanwendung verfügt. Dies könnte über ein Werbebanner innerhalb derselben Webseite passieren oder über einen manipulierten Link in einem Userpost. Der Aufruf muss aber nicht innerhalb der Anwendung passieren, es würde durchaus ausreichen, wenn der Angreifer den manipulierten Link in einem Forum postet, dass sich mit relevanten Themen zur Zielanwendung befasst. Verwendet das Opfer zum Lesen des Forums den gleichen Browser reicht der

Klick auf den Link während es in der Zielanwendung eingeloggt ist und der schadhafte Request kann mit den Zugangsdaten und der Session-ID des Opfers ausgeführt werden.

Für diesen Angriff sind zwar sehr genaue Informationen über das Opfer nötig, ist er jedoch erfolgreich kann damit erheblicher Schaden angerichtet werden. Man stelle sich nur vor was möglich ist, wenn jemand einen beliebigen Befehl innerhalb einer Netbanking Anwendung ausführen kann und das nicht nur mit den gültigen Zugangsdaten eines Anderen, sondern auch über dessen IP-Adresse.

Für das Webportal des Reifenhotels ist diese Art der Bedrohung durchaus ein Thema. Verhindert wird dies durch die Verwendung von sogenannten CSFR-Tokens. CSFR steht hier natürlich für Cross-Site Request Forgery und diese Tokens sind genau dazu da dieses zu vermeiden. Die Idee dahinter ist, dass jeder Request nur aus einem bestimmten Kontext heraus verschickt werden darf. Damit das Backend der Anwendung weiß, dass dieser Kontext gegeben ist werden CSFR-Tokens verwendet. Dieser Kontext ist im Falle eines Webportals eine bestimmte Form, durch deren Verwendung ein Request versandt werden soll. Beim Laden dieser Form wird daher ein solcher Token generiert und als Input vom Typ „*hidden*“ zur Form hinzugefügt.

```
1.  $data['token'] = $this->auth->token();
2.  $this->load->view('myView', $data);
```

Zusätzlich wird der Token beim Generieren innerhalb der Session gespeichert.

```
1.  function token()  {
2.    $token = bin2hex(openssl_random_pseudo_bytes(16));
3.    this->session->set_userdata('token', $token);
4.    return $token;
5.  }
```

Der Controller der den Request bearbeitet muss dann überprüfen ob der Token der innerhalb der Session gespeichert ist, dem entspricht der über den Input der Form hereinkommt.

```
1.  if($this->input->post('token') == $this->session->userdata('token'))
2.  {
3.    //...
4.  }
```

3.1.4.5 SESSION-MANAGEMENT UND AUTHENTIFIZEIRUNG IN DER REIFENHOTEL ANWENDUNG

Wie bereits erwähnt werden zur Generierung von Session-IDs im Reifenhotel GUIDs herangezogen. Für jeden Anmeldevorgang wird eine neue Session generiert. Beim Logout wird die Session zerstört. Ist ein Benutzer 15 Minuten untätig wird er automatische von der Webanwendung abgemeldet und seine Session wird ebenfalls zerstört. Will er die Arbeit fortsetzen muss er sich erneut anmelden und bekommt eine neue Session-ID.

Die Applikation am Scanner verfügt ebenfalls über eine Abmeldefunktion. Außerdem meldet sich die Applikation selbst ab wenn der Benutzer des Scanners 15 Minuten untätig ist. Diese Leerlaufzeit kann vom Administrator allerdings angepasst werden. Die Applikation lädt sich diese Information bei einer erfolgreichen Anmeldung herunter. Diese Leerlaufzeit bezieht sich allerdings nur auf Nutzeraktivität am Scanner wie zum Beispiel Tasten- und Bildschirmaktivitäten des Arbeiters. Ist die Session am Server inzwischen abgelaufen bekommt der Benutzer eine dementsprechende Fehlermeldung und die Applikation wechselt auf die Anmeldemaske, damit sich der Benutzer erneut anmelden kann.

Die gesamte Kommunikation inklusive des Logins nutzt die ‚Secure Socket Layer' – Technologie. Sie ist daher umgangssprachlich formuliert SSL-Verschlüsselt. Die Webanwendung übermittelt dem Client die Session über den Header-Wert ‚Set-Cookie'.

Die Scanner-Applikation bekommt den Session-Token innerhalb des JSON – Objects das bei einer erfolgreichen Anmeldung zurückgesandt wird übermittelt. Die Applikation muss sich diesen Wert als globale Variable zur Laufzeit speichern und mit jedem weiteren Request wieder übergeben. Die Session-ID wird nicht dauerhaft am Gerät gespeichert. Stürzt die Applikation ab oder wird sie beendet (mit oder ohne Abmeldung) dann geht die Session-ID verloren und der Benutzer muss sich per Login eine neue holen.

Für die Webanwendung benötigt der Benutzer einen Benutzernamen und ein Passwort das mindestens 10 Zeichen lang sein muss. Auf Kundenwunsch wurde von der Notwendigkeit von Zahlen, Groß- und Kleinbuchstaben und Sonderzeichen in den gewählten Passwörtern abgesehen obwohl dies von der Firma MOTIONDATA empfohlen wurde. Die Verwaltung der Benutzerkonten liegt beim Reifenhotel-Administrator. Das Anlegen und Löschen eines Administrators liegt zum Zeitpunkt da diese Arbeit verfasst wird noch beim zuständigen Entwickler der Firma MOTIONDATA.

Die Benutzer die mit dem Handscanner arbeiten wurden in eine andere Benutzergruppe, in weiterer Folge ‚Gruppe Lager' genannt, unterteilt. Sie sind daher auch nur zum Absetzen einer reduzierten und klar geregelten Auswahl an API–Anfragen berechtigt.

Der Ursprünglich von der Firma MOTIONDATA entworfene Anmeldmechanismus für die Gruppe Lager wurde vom Kunden nach einer ersten Testphase abgelehnt. Die Ursprüngliche Implementierung sah vor, dass sich ein Mitarbeiter der Gruppe Lager anmeldet indem er seine Benutzerkennung die als Barcode auf seinem Mitarbeiterausweiß angebracht ist scannt und die Anmeldung mit einem zumindest 4-stelligen Pincode bestätigt. Die POST Methode zum Anmelden eines Mitarbeiters übermittelt dann die Benutzerkennung, den Pincode und außerdem die Device-ID des Gerätes in einem JSON Objekt an das API. Diese überprüft

1.) ob es den Benutzer gibt,
2.) ob der Pincode zum Benutzer passt und
3.) ob der Benutzer diesem Gerät zugewiesen ist.

Das Eintippen eines zusätzlichen Pincodes wurde vom Kunden als unpraktisch im Arbeitsalltag bezeichnet.

Der Pincode war als zusätzliche Sicherheitsvorkehrung gedacht. Die Benutzerkennung eines Mitarbeiters wäre sehr leicht von seinem Ausweis auszulesen, da es sich um einen gewöhnlichen Barcode handelt. Dies allein als Authentifizierung um eine Session-ID zu erhalten ist nicht ausreichend. Der Benutzer muss nun zwar keinen Pincode mehr eintippen, die Benutzerkennung wird jedoch vom Scanner gescannt und mit einem geheimen API-Token ergänzt. Der API-Token ist im Code der Applikation enthalten und kann so von niemanden außer den Entwicklern eingesehen werden. Der Token kann auch nicht ausgelesen werden da die Kommunikation SSL verschlüsselt ist. Der Token wird für jedes Softwareupdate neu generiert. Der Server ignoriert jeden Anmeldeversuch der diesen Token nicht beinhaltet. Der Server kennt die aktuellen Tokens da die Serverimplementierung auch bei der Firma MOTIONDATA liegt. Diese Implementierung gewährleistet, dass Anfragen an das API nur von der Applikation am Scanner kommen und nicht von irgendwelchen Tools oder selbstgeschriebenen Clients. Diese Applikation ist sehr stark an die fixierten Abläufe der Reifenhotelorganisation gebunden und bietet in den Augen des Kunden wenig Spielraum um Schaden anzurichten, da jeder API-Zugriff einem Gerät zugeordnet werden kann. Wird ein Scanner entwendet oder ist nicht mehr auffindbar kann dessen Device-ID außerdem vom Administrator gesperrt werden. Anfragen von diesem Gerät sind danach ungültig.

4 DIE WEBAPPLIKATION – IMPLEMENTIERUNG DER SERVER LOGIK

Was ist REST? REST steht für *Representational State Transfer*. REST ist keine Programmiersprache, kein Protokoll oder Framework. Es ist viel eher eine Idee oder Ansammlung von Regeln wie ein Protokoll oder eine Programmiersprache verwendet werden soll. In der englischen Fachliteratur taucht immer wieder der Begriff *REST Architectural Style* auf, also ein Architekturstil für eine Anwendung. Man spricht daher vom REST–Architekturstil wann immer Daten, über ein standardisiertes Interface (wie zum Beispiel HTTP), transferiert werden und das ohne eine zusätzliche Protokollschicht (Messaging Layer), wie zum Beispiel Simple Object Access Layer (SOAP).[20] REST kann auch als Alternative zur Webserviceschiene die auf SOAP, WSDL und den WS-* Spezifikationen basiert gesehen werden. Diese Art der *application-to-application* Kommunikation war lange Zeit die vorherrschende Methode APIs bereitzustellen und zu integrieren.

Ein RESTful Service sollte daher dazu verwendet werden die Repräsentation einer Ressource oder einem Set an Ressourcen, nicht die Ressourcen selbst, für einen konsumierenden Client zur Verfügung zu stellen. Der Client kann aber auch selbst Ressourcen verändern oder hinzufügen. Dazu sendet er eine Repräsentation davon an den Service, der daraus wiederum eine neue Ressource generiert oder eine bestehende aktualisiert. Da jede Ressource über eine eindeutige Adresse auch URI (*Uniform Resource Identifier*) genannt, verfügen muss, können Ressourcen auch gelöscht werden. Diese Funktionen werden auch als CRUD-Befehle (*Create, Read, Update, Delete*) bezeichnet.

Die Serverlogik der Reifenhotel Anwendung orientiert sich stark an diesem Konzept. Eine komplexe Businesslogik auf diese vier Befehle herunter zu brechen und dabei die oben genannten Restriktionen einzuhalten ist in einem iterativen Entwicklungsprozess, unter dem allgemein herrschenden Kostendruck der Privatwirtschaft, allerdings nur bedingt möglich. Wie bereits erwähnt handelt es sich beim Backend der Reifenhotel Anwendung um eine http-basierten Webapplikation nach Vorbild eines RESTful Services und nicht um einen RESTful Service. Wo die Philosophie von REST aufgegeben wurde, Regeln, die als Best-Practice gelten, durchbrochen werden und wie man dies vermeiden hätte können soll zum Abschluss dieser Arbeit in Kapitel 6 noch näher beleuchtet werden.

Dieses Kapitel beschäftigt sich mit dem Webframework CodeIgniter das zur Umsetzung der Webapplikation der Reifenhotelsoftware ausgewählte wurde und mit der Webapplikation selbst.

4.1 RESTFUL SERVICES MIT DEM CODEIGNITER FRAMEWORK

Eine REST – Architektur von Grund auf zu gestalten lässt dem Entwickler zwar viele Freiheiten ist aber mit erheblichem Aufwand verbunden. Ein Framework kann dem Entwickler einiges dieser Arbeit abnehmen, indem es einerseits das nötige Grundgerüst, wie Ordnerstruktur und Routing, zur Verfügung stellt und andererseits Strukturen schafft in denen der Entwickler gängige Komponenten einer Webanwendung, zum Beispiel Datenbankzugriffe, Sessions etc. einfach nach seinen

[20] Vlg. https://docs.oracle.com/cd/E24329_01/web.1211/e24983/overview.htm#RESTF105; Oracle – Help Center; Letzter Aufruf am 26.08.2017

Vorstellungen definieren kann. Kurzgesagt sollen sich wiederholende Tätigkeiten vereinfacht werden und die Wiederwendung von Code gefördert werden.[21]

Das CodeIgniter Framework ist ein Open Source Webframework der US-amerikanische Software-Firma **EllisLab,** das auf der Skriptsprache PHP basiert. Als Open Source Projekt wird sein Quellcode unter der MIT–Lizenz [22] auf GitHub [23] bereitgestellt. CodeIgniter wird immer wieder als leichtgewichtiges, schnelles Framework bezeichnet. Rasmus Lerdorf, der Erfinder von PHP, sagte zum Thema PHP-basierte Frameworks unter anderem, er möge CodeIgniter denn „es ist schneller, leichter und am wenigsten wie ein Framework". Lerdorf sieht den Gebrauch von Frameworks allgemein aber kritisch, da sie weit weniger Performant sind und mit ihren vorgegebenen Strukturen Entwickler manchmal in die falsche Richtung locken.[24]

4.1.1 DATENSTRUKTUREN – MVC

Das Konzept des Frameworks orientiert sich an der Model-View-Controller-Architektur (MVC) und lässt auch eine Erweiterung auf eine hierarchische Model-View-Controller-Architektur (HMVC) zu. Dies bedeutet, dass sich Controller, Model und Views in mehreren Unterverzeichnissen gruppieren lassen. Die strikte Trennung von Logik und Darstellung der MVC-Architektur lässt es zu, dass ein systemübergreifend verwendbares Backend entwickelt wird, die Weboberfläche aber unabhängig davon im Corporate Design der Firma MOTIONDATA gehalten ist.[25] Der Entwickler ist allerdings nicht gänzlich an die MVC-Architektur gebunden. Es wird ausdrücklich darauf hingewiesen, dass es dem Entwickler offen steht seine Applikation minimalistischer anzulegen und nur Controller und Views zu verwenden.

Eine der ersten Fragestellungen, die es beim Implementieren eines RESTful API zu beantworten gilt, lautet, wie eine Ressource überhaupt über einen URI gefunden werden kann. Als Beispiel soll her folgende URL aufgerufen werden:

https://test-rh.motiondata.at/services/v1/lagergut

diese soll offensichtlich eine Repräsentation aller Ressourcen vom Typ Lagergut liefern. Was soll aber folgende URL liefern?

https://test-rh.motiondata.at/services/v1/lagergut/10000175

offensichtlich eine Repräsentation der Lagergut-Ressource mit der ID 10000175. Nun existiert diese Ressource zwar in der Datenbank aber es existiert keine Datei oder Verzeichnis am Server mit diesem Namen. Hier muss also veranlasst werden, dass der Request nicht nach dieser speziellen Adresse sucht, sondern schon zuvor in seine Einzelteile zerlegt wird und dem Controller die weitere Logik überlassen

[21] Vlg. https://de.wikipedia.org/wiki/Webframework; Wikipedia – Webframework; Letzter Aufruf 26.08.2017
[22] Vlg. https://www.codeigniter.com/user_guide/license.html; CodeIgniter – The MIT License (MIT); Letzter Aufruf am 26.08.2017
[23] Vlg. https://github.com/bcit-ci/CodeIgniter; GitHub - bcit-ci/CodeIgniter; Letzter Aufruf am 26.08.2017
[24] Vlg. https://www.sitepoint.com/rasmus-lerdorf-php-frameworks-think-again/; Sitepoint; Letzter Aufruf am 26.08.2017
[25] Vlg. https://www.codeigniter.com/user_guide/overview/mvc.html; CodeIgniter; Letzter Aufruf am 27.08.2017

wird. Dazu kann ein zentraler Einstiegspunkt für alle Anfragen gewählt werden. Auf diesen zentralen Einstiegspunkt müssen alle Anfragen weitergeleitet werden.

Dazu kann im Root-Verzeichnis des CodeIgniter Frameworks in der Datei *index.php* ein *Basepath* hinterlegt werden. Per Default zeigt dieser auf das File ‚*CodeIgniter.php*' im Verzeichnis ‚*root/system/core*'. Hierher gelangen alle Anfragen und hier passiert das Weiterrouten. Grundsätzlich sollten URLs wie folgt aufgebaut sein:

```
example.com/class/function/ID
```

Dabei repräsentiert *class* die Controller-Klasse nach der gesucht werden soll und *function* die Funktion oder Methode die aufgerufen werden soll. Aller weiteren Teile der URL werden dem Controller übergeben in dem die weitere Logik implementiert ist.[26]

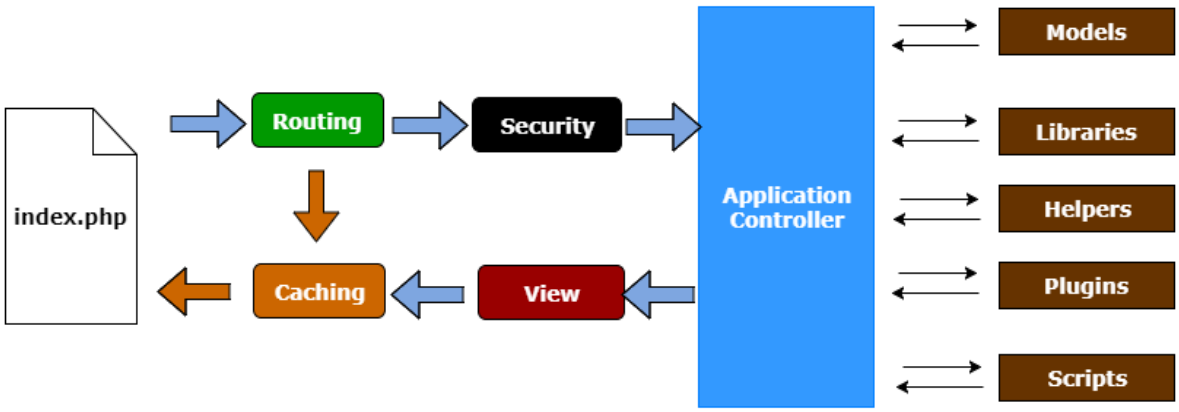

Abbildung 4.1 - CodeIgniter Framework
Application Flow Chart des CodeIgniter Frameworks nach British Columbia Institute of Technology[27]

Um zu vermeiden, dass Benutzer sich direkt per URL in die Ordnerstruktur des Frameworks navigieren können, wird in jedem Ordner per Default eine Datei *index.html* mit der Statusnachricht „***403 - Forbidden***" hinterlegt. Grundsätzlich sollte das Framework aber so konfiguriert werden, dass alle Anfragen auf die Datei *index.php* im Root-Verzeichnis umgeleitet werden. Dazu kann eine Datei *.htaccess* im Root-Verzeichnis hinterlegt werden. Zusätzlich erlaubt CodeIgniter auch sogenannte *Querry-Strings* bei denen man den Namen des gewünschten Controllers und der gewünschten Funktion als Parameter übergibt. Dies wird ermöglicht, in dem man in der Datei *application/config.php* den Controller-Trigger und den Funktions-Trigger setzt.[28]

[26] Vlg. https://www.codeigniter.com/user_guide/general/urls.html; CodeIgniter - CodeIgniter URLs; Letzter Aufruf am 27.08.2017
[27] Vlg. https://www.codeigniter.com/user_guide/overview/appflow.html CodeIgniter; Letzter Aufruf am 27.08.2017
[28] Vlg. https://www.codeigniter.com/user_guide/general/urls.html; CodeIgniter - CodeIgniter URLs; Letzter Aufruf am 27.08.2017

```
1. $config['enable_query_strings'] = FALSE;
2. $config['controller_trigger'] = 'c';
3. $config['function_trigger'] = 'm';
```

Setzt man die Variable *'enable_query_strings'* auf *TRUE*, könnte die Suche nach einem Lagergut mit der ID 345 wie folgt aussehen:

```
index.php?c=Lagergut&m=view&id=345h
```

4.1.2 BIBLIOTHEKEN

Das CodeIgniter Framework bringt ein breites Spektrum an Bibliotheken mit, die einige immer wieder benötigte Helferklassen beinhalten. Diese sind im Verzeichnis *‚root/system/libraries‘* platziert. Darunter die FTP Klasse, die Unit-Testing Klasse, die Emailklasse, eine Verschlüsselungsbibliothek, eine Session Bibliothek und auch Caching-Treiber um nur ein paar wenige aufzuzählen. Eine vollständige Liste aller zur Verfügung gestellten Bibliotheken und Helferklassen ist in der CodeIgniter Dokumentation einsehbar.[29] Sämtliche Bibliotheken können vom Benutzer zum Eigengebrauch erweitert und bearbeitet werden.[30]

Prinzipiell werden Ressourcen bei CodeIgniter erst bei Bedarf geladen. Dieser Vorgang wird erst durch einen konkreten Request ausgelöst. Wenn es aber im Sinne der Anwendung ist, dass eine Ressource automatisch geladen wird kann diese in der Datei *‚application/config/autoload.php‘* hinzugefügt werden.[31]

4.1.2.1 DATENBANK ZUGRIFF

CodeIgniter bietet Datenbanktreiber für viele gängige Datenbankensystem an die die Interaktion mit Datenbanken für den Entwickler stark vereinfachen und vereinheitlichen.[32] Im konkreten beinhaltet das Framework Datenbankklassen für die folgenden Datenbank-Management-Systeme:

- Cubrid
- ibase
- Mysql
- Mysqli
- Mssql

[29] Vlg. https://www.codeigniter.com/user_guide/libraries/index.html; CodeIgniter – Libraries; Letzter Aufruf am 27.08.2017
[30] Vlg. https://www.codeigniter.com/user_guide/overview/mvc.html; CodeIgniter; Letzter Aufruf am 27.08.2017
[31] Vlg. https://www.codeigniter.com/user_guide/general/autoloader.html; CodeIgniter - Auto-loading Resources; Letzter Aufruf am 27.08.2017
[32] Vlg. https://www.codeigniter.com/user_guide/database/index.html; CodeIgniter - Database Reference; Letzter Aufruf am 27.08.2017

- oci8
- odbc
- pdo
- postgre
- sqlite
- sqlite3
- sqlsrv

4.1.2.2 ZUSÄTZLICHE HELFERKLASSEN

Weitere Helferklassen wie zum Beispiel *directory_helper.php*, *language_helper.php*, *string_helper.php*, *text_helper.php* finden sich außerdem im Verzeichnis ‚*root/system/helpers*'. Eine volle Liste aller Helper inklusive Erklärungen und Beispielen findet sich in der CodeIgniter Dokumentation.[33]

4.2 ADAPTIERUNG EINES RESTFUL SERVICES FÜR DIE REIFENHOTEL SOFTWARE

Die Reifenhotelanwendung verfügt über drei Umgebungen die die gleiche Verzeichnisstruktur beinhalten. Die Development-Umgebung, eine Testumgebung und eine Produktivumgebung. Die drei Umgebungen werden über Subdomains abgebildet und sind über die folgenden URLs erreichbar.

Development: https://www.dev-rh.motiondata.at/index.php

Test: https://www.test-rh.motiondata.at/index.php

Produktivumgebung: https://www.rh.motiondata.at/index.php

Zusätzlich zu den unterschiedlichen Umgebungen wurde auch die ursprüngliche CodeIgniter Verzeichnisstruktur (Abbildung 4.2) an die Anforderungen der Reifenhotelarchitektur angepasst.

[33] Vlg. https://www.codeigniter.com/user_guide/helpers/index.html; CodeIgniter – Helpers; Letzter Aufruf am 27.08.2017

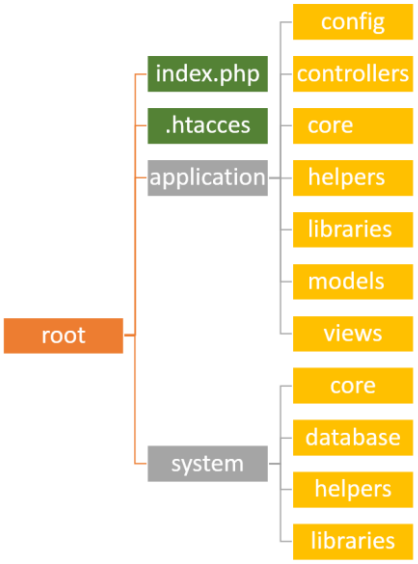

Abbildung 4.2 - Verzeichnisstruktur des CodeIgniter Frameworks per Default

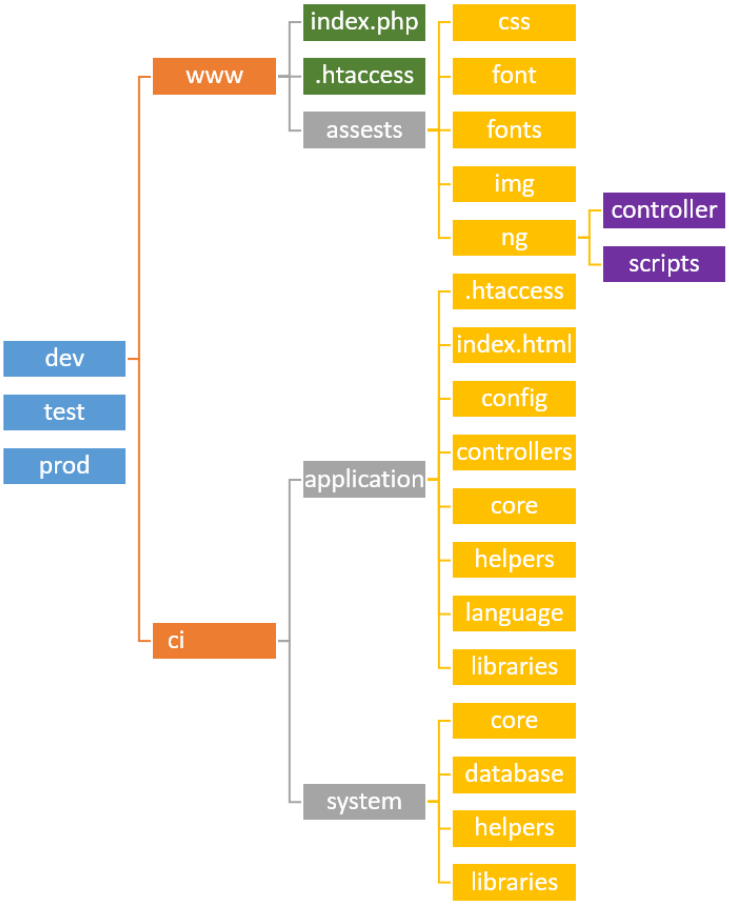

Abbildung 4.3 - Verzeichnisstruktur des CodeIgniter Frameworks angepasst für die Reifenhotelanwendung

Wie in Abbildung 4.3 zu sehen ist wurde eine zusätzliche Ebene geführt durch die die Weboberfläche von der implementierten Logik getrennt wird. Dieses Vorgehen ist immer dann sinnvoll wenn kein reiner Webservice angeboten wird, sondern auch eine grafische Oberfläche, in Form einer Webseite, Teil der Architektur ist. Das Verzeichnis *www* enthält hier die Datei *index.php,* die als zentraler Einstiegspunkt für die gesamte Anwendung dient. Darin wird die Variable ***application_folder***, die den Pfad zum Verzeichnis *application* festlegt und die Variable ***system_path***, die den Pfad zum Verzeichnis *system* festlegt, definiert. Des Weiteren enthält der Ordner die Datei *.htaccess,* die dafür verantwortlich ist, dass alle Anfragen auf die Datei *index.php* in genau diesen Ordner weitergeleitet werden. Innerhalb dieser Datei wird daraufhin der entsprechende Controller instanziiert, der für die jeweilige Anfrage zuständig ist.

4.3 CONTROLLER

Wie in Abbildung 4.4 erkennbar ist wurde im Verzeichnis *controller* innerhalb des *application* Verzeichnisses eine Ordnerstruktur eingeführt die unteranderem die Unterordner Services, Administration und Reifenhotel beinhaltet. Diese Gruppierung teilt die Sammlung von Controllern die für die gesamte Anwendung nötig sind in ihre Einsatzgebiete ein. Das Verzeichnis *administration* enthält dabei Controller wie *benutzer.php, basisdaten.php* und *rollen.php* die Aktionen verwalten die nur von Systemadministratoren durchgeführt werden dürfen. Das Verzeichnis Reifenhotel beinhaltet sämtlich Controller die die Logik innerhalb der Webanwendung abbilden. Genau wie die Controller im Verzeichnis *administration* werde sie alle vom Core-Controller *MD_Controller* abgeleitet. Dabei handelt es sich um eine, für die Reifenhotelanforderung angepasste Ableitung des *CI_Controllers*, der wiederum der Basis Controller des Frameworks ist und von dem alle anderen Controller ableiten sollten.

Wie bereits erwähnt werden alle Anfragen über die Datei *index.php* geleitet. Diese wiederum lädt die Core–Datei *CodeIgniter.php,* innerhalb derer der URI der Anfrage in seine Einzelteile zerlegt wird. Mit Hilfe der php-Funktion *class_existes* wird nach dem benötigten Controller gesucht und dieser wird gegebenenfalls instanziiert. Der nun instanziierte Controller ist nun dafür verantwortlich die weiteren benötigten Schritte zur korrekten Abarbeitung der Anfrage zu tätigen. So wird als Beispiel in der *index* Funktion des Controllers *einlagerungen.php* (die *index()* Funktion eines Controllers wird immer dann aufgerufen, wenn die URL der Anfrage keinen Methodennamen übergeben bekommt) das Model *lagergut_model* geladen. Aus den Daten die das Model zur Verfügung stellt wird ein Datenfeld zusammengestellt. Dieses wird an die View *list_std* übergeben und diese View wird geladen.

```
1. $this->load->view("master/list_std", $data);
```

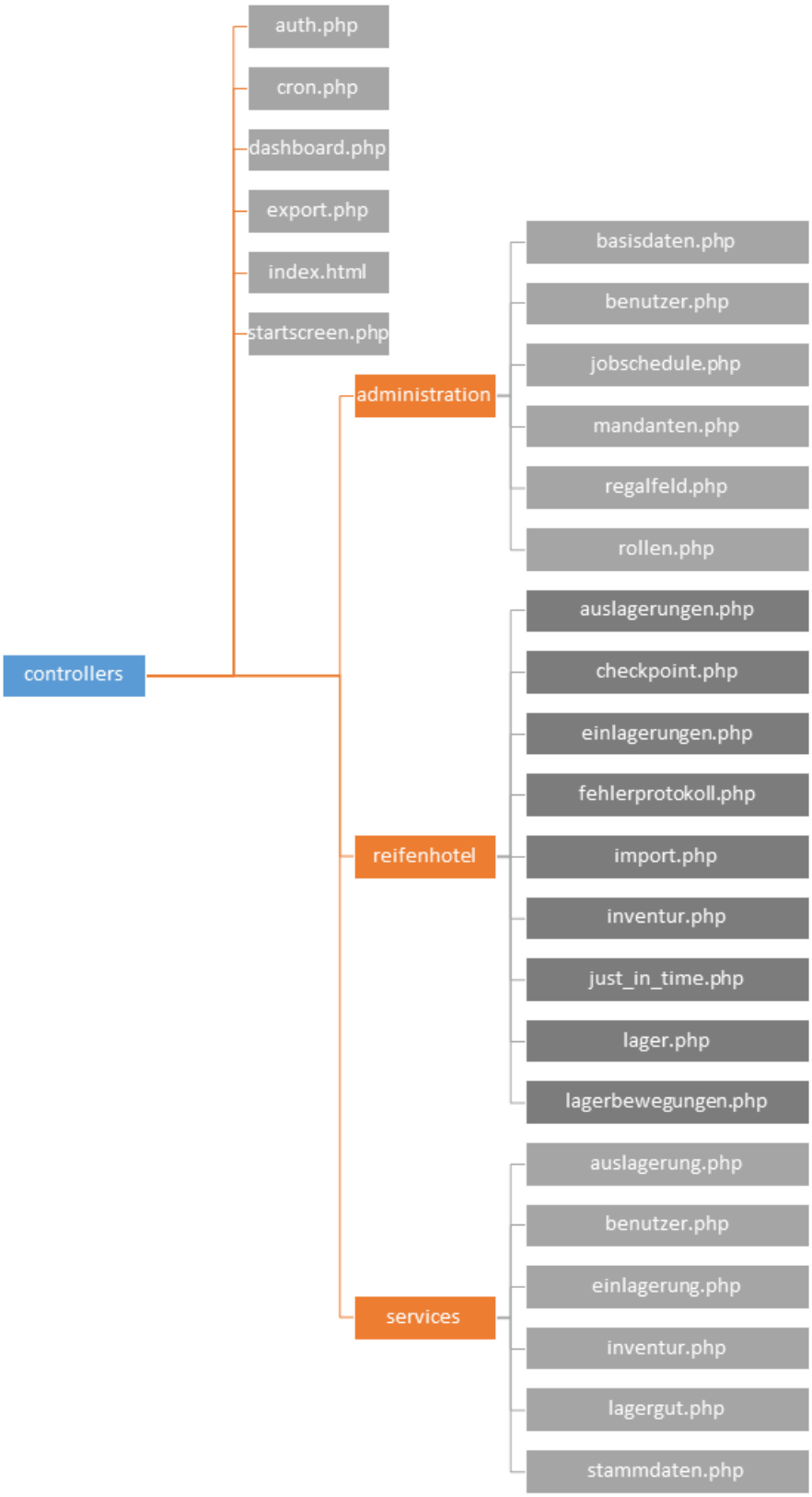

Abbildung 4.4 - Verzeichnisstruktur Controller

Das Verzeichnis controller innerhalb des application Verzeichnisses

Wie in Abbildung 4.4 zu erkennen ist gibt es zusätzlich zu den Unterordnern *administration* und *Reifenhotel* den Unterordner *services*. Dieses Verzeichnis bildet das öffentliche API des Reifenhotels ab. Im Gegensatz zu den Controllern in den Verzeichnissen Administration und Reifenhotel sind die Controller innerhalb dieses Verzeichnisses nicht von der Klasse *MD_Controller* sondern von der Klasse *REST_Controller* abgeleitet. Dabei handelt es sich um eine Open-Source php-Klasse die an die Anforderungen der Reifenhotel Architektur angepasst wurde. Bereitgestellt wird diese via Git-Hub.[34] Beim Autor handelt es sich um **Chris Kacerguis**, der auch schon am CodeIgniter Projekt beteiligt war.[35] Sie bietet nützliche Funktionen wie zum Beispiel

_detect_*method()* – Methode zum Eruieren der http-Methode einer Anfrage.

_detect_api_key() – Boolean-Methode die herausfindet ob ein API-Key übergeben wurde.

_log_request() – Methode zum Loggen der Anfrage.

_log_response() – Methode zum Loggen der Antwort.

4.4 MODEL

Innerhalb des Frameworks ist die Funktion des Models wie folgt beschrieben:

*„The **Model** represents your data structures. Typically your model classes will contain functions that help you retrieve, insert, and update information in your database.”*[36]

Das Model soll also laut Definition Datenstrukturen abbilden. Typischerweise sollte das Model Funktionen bereitstellen die dem aufrufenden Controller genau die Daten liefert die in einem gewissen Anwendungsfall benötigt werden. Je nach Anwendungsfall werden die bereitgestellten Daten dann an eine View weitergegeben, in eine http-Response verpackt oder sie sind ausschlaggebend dafür welche weiteren Schritte innerhalb der implementierten Logik folgen. Kurzum jegliche Interaktion mit einer Datenbank sollte über das Datenmodel geregelt sein.

Betrachten wir hierzu ein Beispiel aus dem Reifenhotel. Der Anwendungsfall ist eine hereinkommende Einlagerung. Die URL der Anfrage lautet *‚…/api/V1/Einlagerung*‘. Die Methode der Anfrage ist ***POST***. Eine JSON-Repräsentation der Einlagerung befindet sich in der Payload. Wie bereits erläutert wird diese Anfrage zur *index_post()*-Funktion der Controllers *Einlagerung.php* durchgeroutet. Diese Funktion validiert zuerst die Daten in der Payload gegen die geltenden Restriktionen und übergibt sie bei erfolgreicher Validierung an die Funktion *_do_einlagerung($arr_post)*. Wobei es sich bei *$arr_post* um den Inhalt der Payload in Form eines Datenfeldes handelt. Dann muss geprüft werden ob die

[34] Vlg. https://github.com/chriskacerguis/codeigniter-restserver; GitHub; Letzter Aufruf am 17.10.2017
[35] Vlg. https://github.com/chriskacerguis; GitHub - Chris Kacerguis; Letzter Aufruf am 17.10.2017
[36] Vlg. https://www.codeigniter.com/user_guide/overview/mvc.html; CodeIgniter – MVC; Letzter Aufruf am 19.10.2017

Einlagerung mit diese ID bereits vorhanden ist und wenn ja, welchen Satus sie hat. Dies geschieht mit folgendem Befehl:

```
1. $o_einlagerung=$this->lagergut_model-
   >get_lagergut_by_einlagerungsnummer($arr_post["einlagerungsnummer"]);
```

Da im Konstruktor des Controllers bereits das Lagergutmodel geladen wurde

```
1. function __construct()  {
2.     parent::__construct();
3.     $this->load->model("services/operatives_system_model");
4.     $this->load->model("reifenhotel/lagergut_model");
5.     $this->load->model("reifenhotel/lagerplatz_model");
6.     $this->load->library("MD_Fehlerprotokoll");
7. }
```

kann in weiterer Folge die Funktion *get_Lagergut_by_einlagerungsnummer* aus dem Lagergutmodel verwendet werden. Innerhalb dieser Funktion passiert der eigentliche Datenbankzugriff. Dieser bedient sich der bereitgestellten Helferklassen:

```
1. $this->db->from("rho_lagergut lagergut");
2. $this->db->where("lagergut.einlagerungsnummer", $einlagerungsnummer)
3. $query = $this->db->get();
```

Im weiteren Verlauf dieses Vorgangs wird unteranderem geprüft ob der Kunde, der in der JSON-Repräsentation der Einlagerung enthalten ist, bereits existiert. Dazu wird das Model des operativen Systems herangezogen.

```
1. $kunden_id = $this->operatives_system_model-
   >insert_kunde($arr_post);
```

Die Funktion *insert_kunde* prüft ob ein Kunde bereits im System existiert, aktualisiert ihn, falls ja und legt ihn an, falls nicht.

Das dritte Model das im Konstruktor geladen wird ist das Lagerplatzmodel. Dieses Model stellt unter anderem die Funktionen *insert_lager, get_lagergutbreite* und *get_lagerguthoehe* zur Verfügung. Für den Fall das eine Einlagerung alle Kriterien des definierten Prozesses erfüllt kann sie eingelagert werden und es muss ein passender physischer Lagerplatz im Gebäude gefunden werden. Dieser komplexe Vorgang kann im Controller durch das Datenmodel *lagerplatz_model* innerhalb eines Funktionsaufrufs abgebildet werden.

```
1. $this->lagerplatz_model->insert_lager(array(
2.     "einlagerungsnummer" => $arr_post["einlagerungsnummer"],
```

```
3.      "lagergut_breite" => $this->lagerplatz_model->get_lagergutbreite($o_lagergut),
4.      "lagergut_hoehe" => $this->lagerplatz_model->get_lagerguthoehe($o_lagergut),
5.      "benutzer_bearbeitung" => NULL,
6.      "datum_bearbeitung" => date("Y-m-d H:i:s"),
7.  ));
```

4.5 WEB MASKE - VIEWS

Views sind eine Möglichkeit dem Benutzer die angefragten Daten zu präsentieren. Die Teile der Architektur, die das öffentliche Reifenhotel-API abbilden, kommen ohne Views aus. Sie dienen als Schnittstelle zu anderen Anwendungen und liefern Datenrepräsentationen im JSON Format. Wie diese vom anfragenden Client grafisch dargestellt werden ist dem Entwickler des API egal. Für den Teil der Architektur der die Logik der Webanwendung abbildet spielt es allerdings sehr wohl eine Rolle wie die Daten dargestellt werden. Das CodeIgniter Framework bietet dem Entwickler dafür die Funktion *view*() aus der Klasse *Loader.php* an. Mit dieser Funktion kann die benötigte View geladen werden und die darzustellenden Daten an diese übergeben werden.

```
1. $this->load->view('master/list_std', $data["data_main"]);
```

Die View selbst kann dabei alles Mögliche sein. Im Falle des Reifenhotels handelt es sich um **php**-Dateien die aus **html-**, **css-**, **JavaScript-** und **php-Code** bestehen. Bei einer View kann es sich auch nur um ein Fragment der darzustellenden Seite handeln. Es können nämlich auch mehrere Teile einer Webseite geladen werden. z.B:

```
1. $this->load->view('templates/header',$data);
2. $this->load->view('news/index', $data);
3. $this->load->view('templates/footer');
```

Beispiel aus der CodeIgniter Dokumentation.[37] Hier wird die Seite index.php, unter Einbeziehung der Daten aus dem Objekt $data, geladen. Zusätzlich werden ein Header und ein Footer aus dem Verzeichnis geladen.

Als Beispiel aus dem Reifenhotel soll hier die Einlagerungsansicht aus dem Webportal betrachtet werden.

Navigiert der Mitarbeiter im Webportal des Reifenhotels zu folgender Adresse:

https://dev-rh.motiondata.at/index#/reifenhotel/einlagerungen/index/

wird ihm eine Liste mit ausstehenden Einlagerungen angezeigt, wie man sie in Abbildung 4.5 sieht. Durch die in Kapitel 4.3 erwähnte Funktionsweise des CodeIgniter Frameworks wird daher der Controller *einlagerung.php* aus dem Verzeichnis *controllers\reifenhotel* aufgerufen. Mangels einer

[37] Vlg. https://www.codeigniter.com/user_guide/tutorial/static_pages.html; CodeIgniter – Tutorial; Letzter Aufruf am 19.10.2017

übergebenen Methode wird in diesem Szenario die Methode *index* des Controllers aufgerufen. Diese lädt das Model *lagergut_model.php* aus dem Verzeichnis *models\reifenhotel*.

```
1.  $this->load->model("reifenhotel/lagergut_model");
```

Über dieses Model werden die aktuell offenen Einlagerungen abgerufen:

```
1.  $this->lagergut_model->get_einlagerungen_list(FALSE, $einstellungen)-
    >result_array()
```

Diese Information wird zusammen mit anderer Information in das Daten Objekt *$data["data_main"]* verpackt, um damit die entsprechende View zu laden.

```
1.  $this->load->view('master/list_std', $data["data_main"]);
```

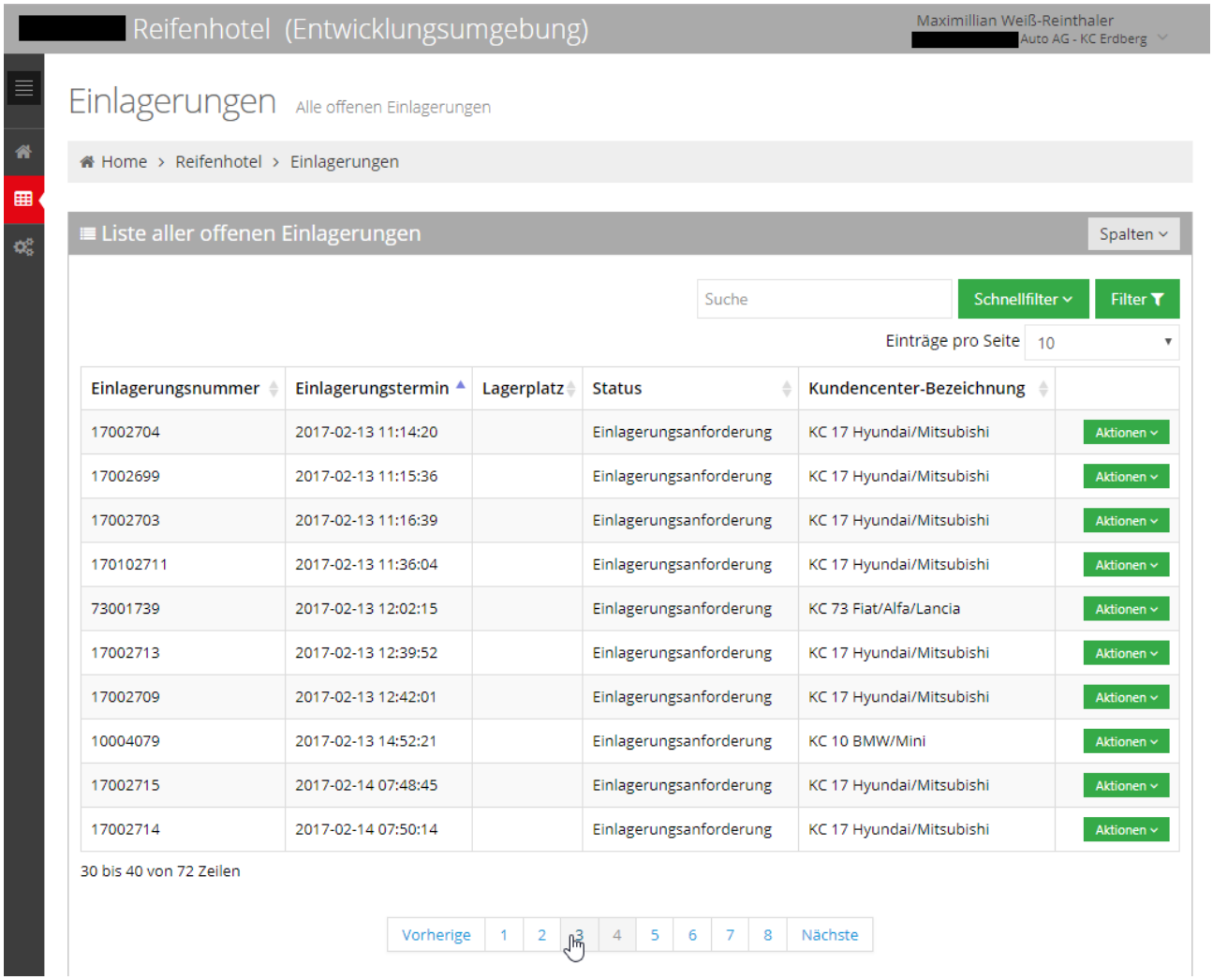

Abbildung 4.5 - Einlagerungsansicht
Liste mit offenen Einlagerungen. Im Webportals des Reifenhotels einsehbar.

5 DER HANDHELDSCANNER – IMPLEMENTIERUNG UND LOGIK

Ein wesentlicher Punkt der in der Analyse des Projekts Reifenhotel war die Frage nach den Endgeräten mit denen die Mitarbeiter des Reifenhotels die einzelnen Arbeitsschritte an das Backend übermitteln. Aus Sicht der Backendentwickler kann es von Vorteil sein, wenn man schon in der Designphase eines API von die Limitierungen der konsumierenden Clients weiß. Daher war es wichtig sich bereits früh im Planungsprozess, zusammen mit dem Kunden, auf ein Endgerät zu einigen an dessen Stärken und Schwächen die Clientarchitektur angepasst wird. Aus der Problemstellung geht hervor, dass mobile Geräte benötigt werden, die über eine kabellose Internetverbindung verfügen, möglichst robust sind, daher stoßfest, spritzwasserfest und außerdem über ein Hardwaremodul zum Auslesen von Barcodes verfügen, sei es eine Kamera oder ein Laser.

Die Firma MOTIONDATA hat bereits Erfahrung mit mobilen Geräten zur Unterstützung von Lagersoftware. Die Clientsoftware MOTIONDATA – Lagermann ist eine Software für Handheldscanner die über einen Webservice mit dem MOTIONDATA DMS verbunden ist. Die Zielgruppe dieser Software sind Kunden mit großen Ersatzteillagern oder Kunden die sich auf den Teilehandel spezialisiert haben. Bei der dafür verwendeten Hardware handelt es sich um einen Handheldscanner der Firma Nordic ID aus Finnland. Dieses Gerät wurde auch im Falle des Projekts Reifenhotel, sowohl vom Kunden als auch von der Firma MOTIONDATA, als am besten geeignet angesehen. In Unterkapitel 5.1 soll dieses Gerät kurz beschrieben werden.

5.1 NORDIC ID MORPHIC HANDHELD SCANNER

Abbildung 5.1 - NordicID Morphic
Der Handheldscanner Typ Morphic der Firma Nordic ID[38]

[38] https://www.nordicid.com/en/home/products/mobile-readers/nordic-id-morphic/; Nordic ID – Morphic; Letzter Aufruf am 04.11.2017

Das Gerät verfügt in der einfachsten und billigsten Ausführung über einen Barcodelaser, eine Frontkamera, eine kabellose Netzwerkverbindung, GPS– und 3G-Verbindung und Bluetooth. Wenn man den Scanner in Abbildung 5.1a betrachtet fällt sofort der gelbe Knopf in der Mitte des Geräts auf. Dieser Knopf löst den Laser zum Auslesen von Barcodes aus. Durch seine Position ist er mit dem Daumen erreichbar wenn das Gerät flach in der Hand des Benutzers liegt. Der Laser und die Frontkamera befinden sich an der nach vorne Gerichteten Seite des Geräts. Der Laser kann zum Einlesen von 1d Barcodes verwendet werden. Die Kamera kann auch 2D – Barcodes (auch QR-Codes genannt) einlesen. Durch diese Beschaffenheit hebt sich der Handheldscanner in Sachen Usability ganz klar von einem gewöhnlichen Smartphone ab und gewährleistet schnelles und effizientes Arbeiten im Lagerbetrieb. Das Gerät verfügt außerdem über einen Touchscreen.

Als Nachteil dieser Geräte gegenüber Smartphones ist das Betriebssystem zu sehen. Die Geräte werden mit dem Betriebssystem Windows Embedded 6.0[39] ausgeliefert. Wie bereits in Kapitel 1 erwähnt, wurde der grundlegende Support für dieses Betriebssystem von Microsoft bereits 2013 eingestellte. Das Enddatum für den erweiterten Support ist auf 10.04.2018 festgelegt.[40]

Für den Kunden überwog das Argument der Usability klar. Aus Entwicklersicht schwebt in diesem Fall allerdings ein zukünftiges Problem im Raum. Es ist sehr wahrscheinlich, dass die die Firma Nordic ID keine neuen Handscanner mit Windows Embbeded 6.0 ausliefert, wenn der Support für dieses Betriebssystem von Microsoft erst einmal gänzlich eingestellt wurde. Mit welchem Betriebssystem die Geräte stattdessen ausgestattet werden ist zurzeit noch nicht klar. Dies bedeutet aber in weiterer Folge, dass die Client Applikation früher oder später auch für ein anderes Betriebssystem bereit gestellt werden muss.

[39] Vlg. https://msdn.microsoft.com/en-us/library/dn600283(v=winembedded.60).aspx; Microsoft – Windows Embedded CE 6.0; Letzter Aufruf am 15.11.2017

[40] Vlg. https://support.microsoft.com/de-de/lifecycle/search?alpha=Windows%20Embedded%206.0; Microsoft Lifecycle-Richtlinie; Letzter Aufruf am 14.11.2014

5.2 NORDIC ID DLLS

Um die Hardware des Scanners anzusprechen stellt die Firma Nordic ID einen eigenen Service zur Verfügung. Mit Hilfe dieses Services kann eine Applikation mit den Hardwaretreibern des Geräts kommunizieren. Der Service nennt sich Nordic ID Multiple Hardware Layers, kurz MHL. Verpackt ist der Service in eine Bibliothek die ähnlich wie ein API zu verwenden ist.[41]

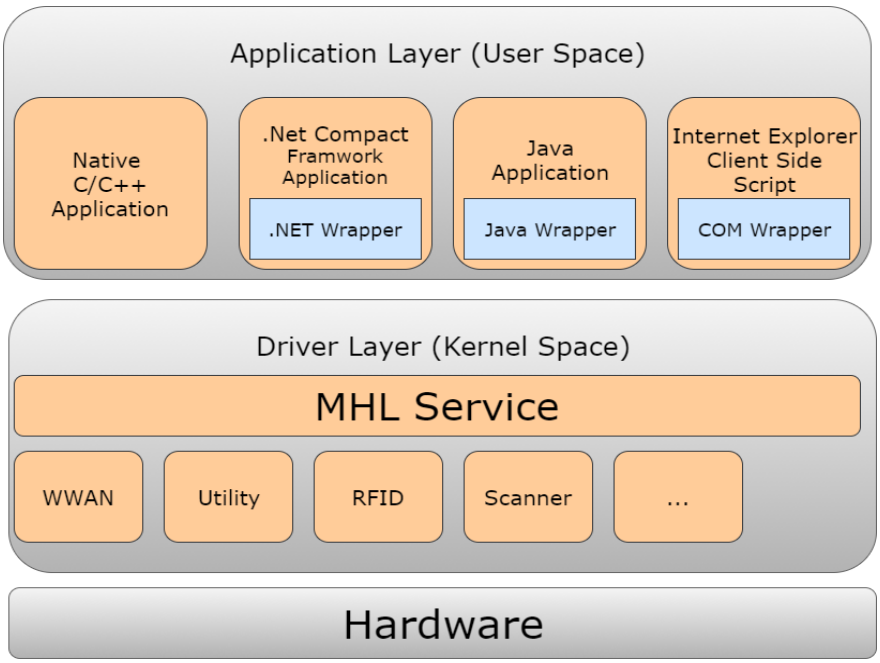

Abbildung 5.2 - NordicID MHL
Architektur des Services Nordic ID MHL [42]

Der MHL Wrapper kann als C#-DLL, Java-Bibliothek oder als C-Header heruntergeladen werden.

In das Projekt eingebunden sieht der bereitgestellte Service, *DOTNET_MHL_V3.dll*, dann aus wie in Listing 6.1 dargestellt.

```
1.  namespace NordicId
2.  {
3.      public class MHL
4.      {
5.          public MHL();
6.
7.          public int CloseDrv(int handle);
8.          public int Execute(int handle, string name);
9.          public byte[] GetBin(int handle, string name);
10.         public byte[] GetBin(int handle, string name, int length);
11.         public bool GetBool(int handle, string name);
```

[41] Vlg. https://www.nordicid.com/en/home/support/mhl; Nordic ID – MHL + Download; Letzter Aufruf am 15.11.2017
[42] Vlg. https://www.nordicid.com/en/home/support/mhl; Nordic ID – MHL + Download; Letzter Aufruf am 15.11.2017

```
12.         public uint GetDword(int handle, string name);
13.         public int GetInt(int handle, string name);
14.         public int GetLastError();
15.         public string GetString(int handle, string name);
16.         public int LoadProfile(int handle, string name);
17.         public int OpenDrv(string name);
18.         public int SaveProfile(int handle, string name);
19.         public int SetBin(int handle, string name, byte[] val);
20.         public int SetBool(int handle, string name, bool val);
21.         public int SetDword(int handle, string name, uint val);
22.         public int SetInt(int handle, string name, int val);
23.         public int SetString(int handle, string name, string val);
24.     }
25. }
```

Listing 5.1

Im Fall der Reifenhotel Applikation wird Zugriff auf Scanner und Tastatur benötigt. Dazu wird beim Aktivieren der jeweiligen Clientmaske der Service instanziiert und mit Hilfe der Funktion *OpenDrv(…)* wird der jeweilige Hardwaretreiber geladen. Siehe Listing 6.2.

```
1.  private const Int32 VK_ESC = 27;
2.  private const Int32 VK_ENT = 13;
3.  private const Int32 VK_F12 = 123;
4.  private HotkeyWindow HotKeyWnd = new HotkeyWindow();
5.  private Int32 hKeyb = 0;
6.  private Int32 hScan = 0;
7.  private MHL mhlCore = new MHL();
8.
9.  void FormEinlagern_Activated(object sender, System.EventArgs e)
10. {
11.     hScan = mhlCore.OpenDrv("Scanner");
12.     hKeyb = mhlCore.OpenDrv("Keyboard");
13.     if (hKeyb > 0)
14.     {
15.         // Save current keyboard map
16.         mhlCore.SaveProfile(hKeyb, "MHL_Reifenhotel");
17.
18.         mhlCore.SetDword(hKeyb, "SpecialKey.Esc.All.VK", VK_ESC);
19.         mhlCore.SetDword(hKeyb, "SpecialKey.Ok.All.VK", VK_ENT);
20.         mhlCore.SetDword(hKeyb, "SpecialKey.Scan.All.VK", VK_F12);
21.
22.         mhlCore.SetDword(hKeyb, "Keyboard.Reload", 1);
23.
24.         // Assign HotKeyWnd callback
25.         HotKeyWnd.callback = new HotkeyCallbackFunc(HotkeyCallback);
26.         HotKeyWnd.RegisterKey(VK_ESC, KeyModifiers.None);
27.         HotKeyWnd.RegisterKey(VK_ENT, KeyModifiers.None);
28.         HotKeyWnd.RegisterKey(VK_F12, KeyModifiers.None);
29.     }
30. }
```

Listing 5.2

Zusätzlich wird die Tastaturbelegung gespeichert und die gespeicherten Tasten werden für das Event *HotkeyCallback* registriert in der in weiterer Folge ihre jeweilige Funktion implementiert wird. Das Aktivieren des Lasers zum Einlesen von Barcodes und verarbeiten des Barcodes ist in Listing 6.3 dargestellt.

```
1.  string result = string.Empty;
2.  if (hScan > 0)
3.  {
4.      mhlCore.SetDword(hScan, "Scanner.Scan", 2);
5.      result = mhlCore.GetString(hScan, "Scanner.ScanResultString");
6.  }
7.  return result;
```

Listing 5.3

5.3 BEREITSTELLEN UND INSTALLATION DER APPLIKATION

Zusätzlich zu dem Service MHL stellt die Firma Nordic ID auf ihrer Homepage auch ein Programm namens Customizer zu Verfügung.[43] Eine Software die dem Entwickler beim Erstellen und Verwalten der fertigen Softwarepakete behilflich ist. Das erstellte Executable wird zusammen mit allen Abhängigkeiten, darunter auch die benötigt Bibliothek DOTNET_MHL_V3.dll, in eine .PAK-Datei verpackt. Wird diese Datei in den Flashspeicher des Geräts gelegt, wird sie beim Starten des Geräts von Windows CE installiert. Die gewünschten Dateipfade müssen daher beim Erstellen der .PAK Datei bereits berücksichtigt werden.

5.4 APPLIKATION - USER INTERFACE

Die Oberfläche der Clientsoftware besteht aus einer vorgeschalteten Anmeldemaske, einem Hauptmenü mit vier Menüpunkten und einem Untermenü mit ebenfalls vier Menüpunkten. Die Möglichkeiten und Funktionen der Oberfläche sind auf Wunsch des Kunden sehr starr auf die Abläufe und Prozesse, die in Kapitel 2 beschrieben wurden, beschränkt. Abbildung 5.3 zeigt eine Sitemap durch die einzelnen Masken der Applikation.

[43] Vlg. https://www.nordicid.com/en/downloads/file-preview/downloads/development-tools/apps/nordic_id_customizer_2_v.1.0.0.59.exe ; Nordic ID – Customizer 2, Download; Letzter Aufruf am 16.11.2017

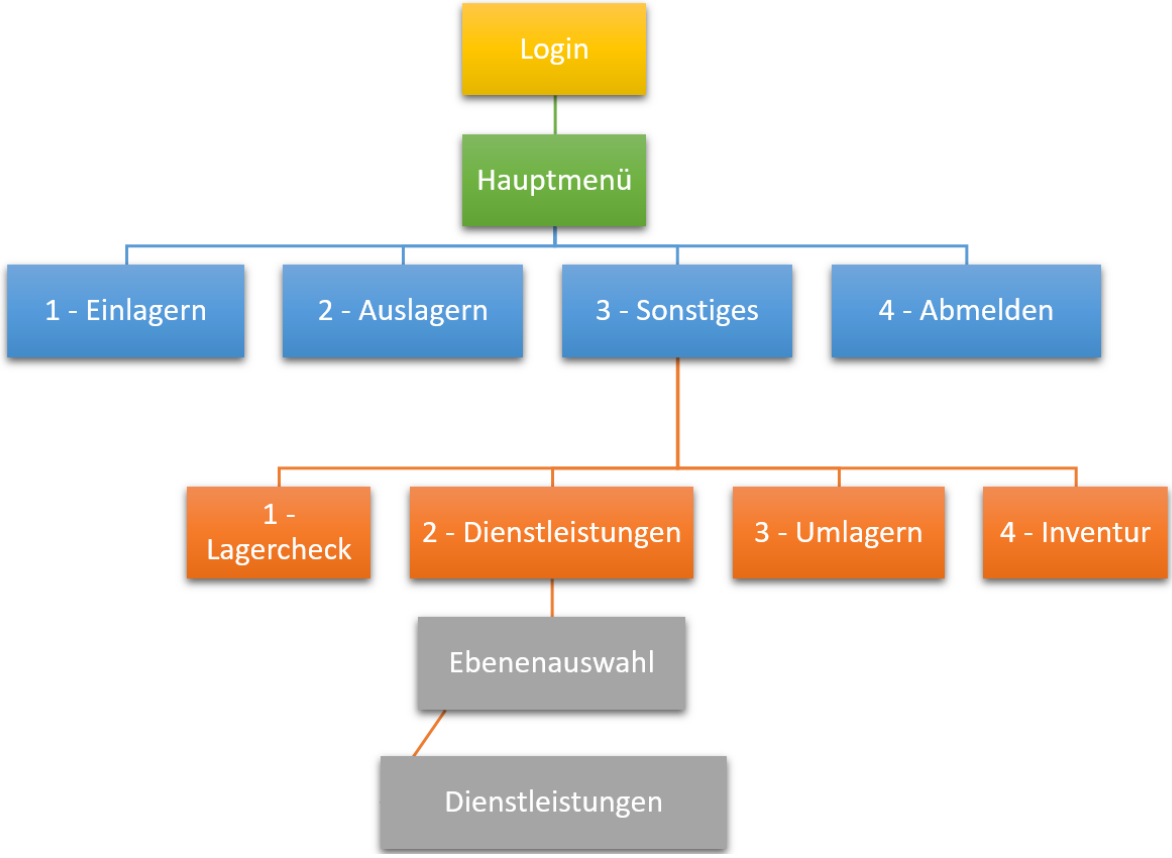

Abbildung 5.3 - Aufbau der Clientsoftware

5.4.1 LOGIN

Wird die Applikation gestartet, erscheint als Erstes eine Anmeldemaske wie sie in Abbildung 5.4 a zu sehen ist. Hier wird der Benutzer aufgefordert seine Benutzer ID zu scannen. Der Scanner kann durch Drücken des roten Buttons am Display oder durch Drucken des gelben Scanner-Buttons auf der Tastatur ausgelöst werden. Ist die Benutzer ID valide und die Anmeldung (Kapitel 5.6.1) erfolgreich, färbt sich der Button am Display grün und der Benutzer wird zum Hauptmenü weitergeleitet (Abbildung 5.4b). Ist der Anmeldevorgang nicht erfolgreich wird eine entsprechende Fehlermeldung angezeigt (Abbildung 5.4c). Die Leiste am oberen Bildschirmrand beherbergt drei weitere Steuerelemente. Bei dem ersten Element ganz links handelt es sich um einen Button der, wie sein Icon verrät, zum Beenden der Applikation dient. Auf Wunsch des Kunden erscheint dem Benutzer beim Drücken des Buttons ein zusätzlicher Dialog, inklusive eines schrillen Warntons, mit dem das Beenden der Applikation bestätigt werden muss. Es gibt aus Sicht des Kunden, im normalen Betrieb, für den Mitarbeiter wenige Gründe die Applikation zu beenden. Die beiden weiteren Elemente sind jeweils an Timer gebunden, die in regelmäßigen Abständen den Akkustand und die Signalstärke der kabellosen Internetverbindung abfragen und diese mit den entsprechenden Icons darstellen. Sie sind außerdem auf der Basismaske implementiert von der alle anderen Masken abgeleitet werden und daher auch auf allen weiteren Masken vorhanden.

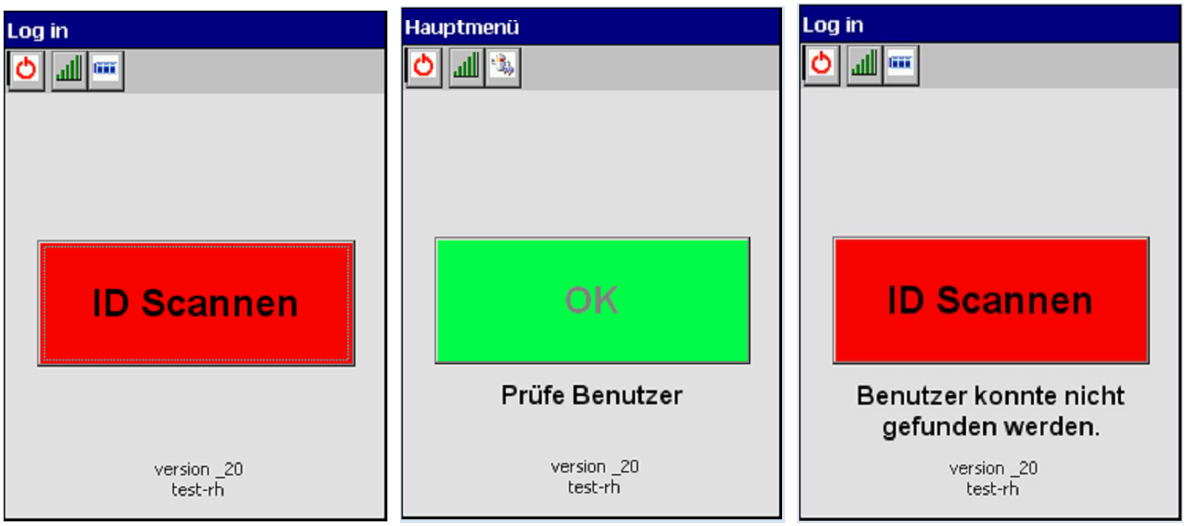

Abbildung 5.4 - Anmeldemaske
a) Anmeldemaske beim Start der Applikation. b) Bei erfolgreicher Anmeldung färbt sich der Button am Display grün. c) Fehlerhafte Anmeldung

5.4.2 HAUPTMENÜ

Das Hauptmenü besteht aus vier rechteckigen Buttons am Display (Abbildung 5.5a). Jeder dieser Buttons steht für einen Menüpunkt. Auf Wunsch des Kunden wurden diese Menüpunkte farblich getrennt. Es war dabei durchaus erwünscht, dass hier grelle Farbtöne gewählt werden. Jeder der Buttons kann per Berührung am Display aber auch durch Drücken der zugeordneten Zahl auf der Tastatur gewählt werden. Das Menü am linken, oberen Bildrand beherbergt zwei Optionen. Erstens kann die Applikation hier direkt beendet werden (inklusive Bestätigungsdialog), zweitens kann sich der Benutzer analog zu Menüpunkt 4 abmelden (Abbildung 5.5b). Wird der gelbe Scanner-Button auf der Tastatur betätigt während sich der Benutzer im Hauptmenü befindet, wird ein Lagercheck (Kapitel 5.6.10) für den gescannten Barcode durchgeführt und der Benutzer wird zur Maske Lagercheck (Abbildung 5.10b) weitergeleitet auf dem ihm das Ergebnis der Anfrage angezeigt wird.

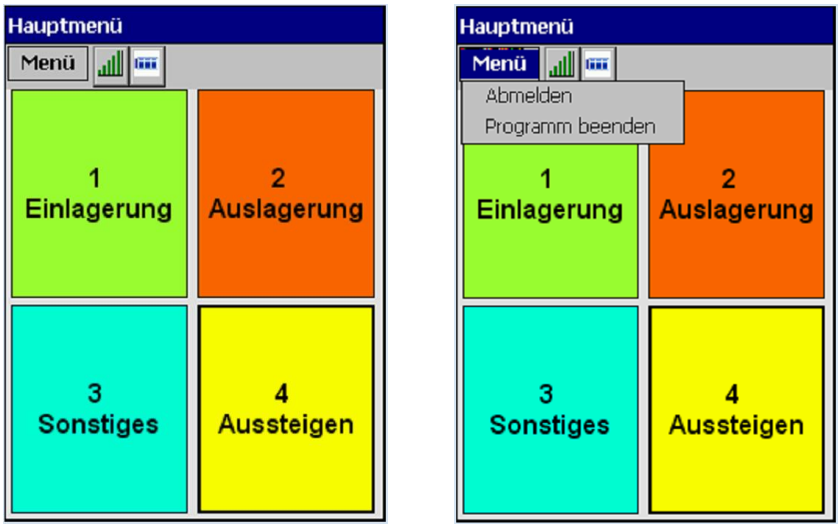

Abbildung 5.5 - Hauptmenü
a) Hauptmenü b) Zusätzliche Optionen des Untermenüs.

5.4.3 EINLAGERUNSGMASKE

Wählt der Benutzer ‚Menüpunkt 1 – Einlagerung' wird er zur Einlagerungsmaske, die in Abbildung 5.6a dargestellt ist, weitergeleitet. Wie bei allen anderen Masken, die über die beiden rot und grün gefärbten Buttons am unteren Bildschirmrand verfügen, ist auch hier das Drücken des grünen bzw. des roten Buttons am Display äquivalent zum Betätigen des grünen bzw. des roten Knopfes auf der Tastatur. Hier wird der Benutzer aufgefordert das Lagergut zu scannen das er einlagern möchte. Unmittelbar nach dem Scan erfolgt im Hintergrund ein Lagercheck (Kapitel 5.6.10) auf den gescannten Barcode. Entspricht der Barcode einem validen Lagergut wird dem Benutzer hier bereits der vorgesehene Lagerplatz angezeigt und er wird aufgefordert diesen ebenfalls zu scannen (Abbildung 5.6b). Der gescannte Barcode wird gegen das Schema der allgemein gültigen Lagerplatzbezeichnungen validiert und sollte es sich um eine solche Lagerplatzbezeichnung handeln wird diese zusammen mit der ID des Lagerguts in einen Request gepackt und an das API gesandt (Kapitel 5.6.3) ohne das es noch weiterer Interaktion des Benutzers bedarf. Ist die Anfrage zur Einlagerung erfolgreich wird dem Benutzer die entsprechende Meldung am Display angezeigt (Abbildung 5.6c). Ist der Lagerplatz auf den Eingelagert wird ein sogenannter Checkpoint, also ein Sammelpunkt am Eingang des Reifenhotels, werden dem Benutzer auch eventuelle Dienstleistungen, die an dem Lagergut vorzunehmen sind, angezeigt (Abbildung 5.6d). Diese kann der Benutzer annehmen oder auf später verschieben. Ist der Vorgang nicht zulässig wird der Benutzer auf die Fehlermaske (Abbildung 5.6e) weitergeleitet. Alle Masken leiten im Fehlerfall auf diese Maske weiter. Die entsprechende Fehlermeldung wird der Maske beim Aufruf übergeben und in der Box in der Mitte angezeigt. Die Textgröße passt sich an die Länge der Meldung an. Durch betätigen des roten Buttons am Display kommt der Benutzer wieder zur ursprünglichen Maske zurück.

Abbildung 5.6 - Einlagerungsmaske

a) Einlagerungsmaske zu Beginn des Vorgangs. b) Ein Lagergut wurde gescannt. Der Lagercheck hat im Hintergrund bereits den vorgesehen Lagerort heraus gefunden. c) Das Lagergut wurde erfolgreich eingelagert. d) Ein anderes Lagergut wurde am Checkpoint des Reifenhotels abgelegt. Dem Mitarbeiter wird dabei angezeigt welche Dienstleistungen für dieses Lagergut offen sind. e) Es wurde ein ungültiger Lagerort gescannt. Die Applikation navigiert zur Fehlermaske.

5.4.4 AUSLAGERUNGSMASKE

Menüpunkt „2 –Auslagern“ beinhaltet die Auslagerungsmaske. Hier hat der Benutzer die Möglichkeit Auslagerungen anzufordern und anzunehmen oder bereits angenommene Auslagerungen auszulagern (Abbildung 5.7a). Durch das Drücken des grünen Buttons auf Display oder Tastatur wird eine Anfrage an den Server geschickt, der daraufhin nach offenen Auslagerungen sucht, die in der nächsten Zukunft zu bearbeiten sind (Kapitel 5.6.9). Mit einer erfolgreichen Anfrage wird der Status des betroffenen Lagerguts von ‚*Auslagerung angefordert*‘ auf ‚*Auslagerung angenommen*‘ gesetzt und dem anfragenden Mitarbeiter zugeordnet (Abbildung 5.7b). Danach kann ein Auslagerungsvorgang analog zum Einlagerungsvorgang durchgeführt werden. Der Mitarbeiter muss daher Lagergut und Lagerort an den jeweiligen Punkten, gemäß dem in Kapitel 2 definiert Prozesses, scannen. Der Mitarbeiter kann die Auslagerung aber auch ablehnen. Wird der rote Button auf Tastatur oder Display gedrückt wechselt die Maske in ihren Ursprungszustand zurück, der in Abbildung 5.7a dargestellt ist. Im Hintergrund wird aber eine Anfrage an den Server geschickt, die im Erfolgsfall den Status des Lagerguts wieder auf ‚*Auslagerung angefordert*‘ zurücksetzt (Kapitel 5.6.8).

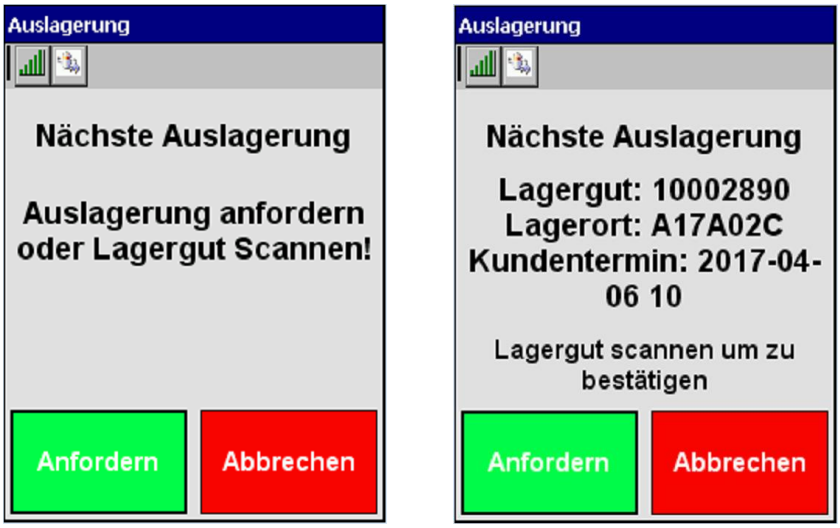

Abbildung 5.7 - Auslagerungsmaske

a) Startzustand der Auslagerungsmaske mit der Aufforderung an den Benutzer, entweder eine Auslagerung anzufordern oder eine bereits angefangene Auslagerung zu bearbeiten, in dem er das betroffene Lagergut scannt. b) Eine neue Auslagerung wurde angenommen. Das Display sagt dem Mitarbeiten wo sie liegt und wohin sie muss.

5.4.5 ABMELDEMASKE

Über Menüpunkt „4 – Abmelden“ kann sich der Benutzer abmelden. Die Applikation navigiert ihn dann wieder zur Anmeldemaske zurück. Im Hintergrund wird die Funktion des API (Kapitel 5.6.2) aufgerufen die die Session des Benutzers am Server für ungültig erklärt.

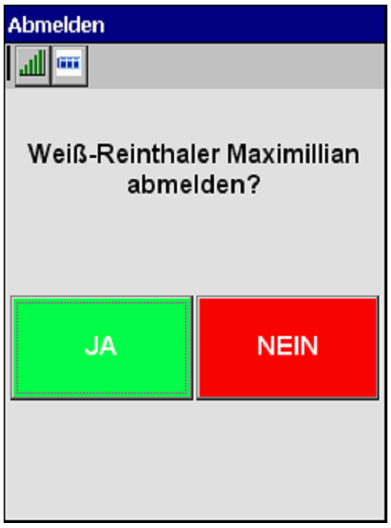

Abbildung 5.8 - Abmeldemaske

5.4.6 UNTERMENÜ – SONSTIGES

Menüpunkt „3 – Sonstiges“ beherbergt vier weiter Menüpunkte.

1. Lagercheck
2. Dienstleistungen
3. Umlagern
4. Inventur

Analog zum Hauptmenü kann auch hier über das Display oder über die Tastatur ein Menüpunkt gewählt werden. Die Scanner-Taste hat auf dieser Maske keine Funktion. Zurücknavigieren ist nur mit roten Knopf auf der Tastatur möglich.

Abbildung 5.9 - Untermenü mit den vier Auswahlmöglichkeiten

5.4.7 LAGERCHECK

Diese Maske ist über Menüpunkt „1 – Lagercheck“ des Untermenüs erreichbar oder aber wie in Punkt 5.4.2 erwähnt, durch betätigen der Scanner-Taste innerhalb des Hauptmenüs. Wird sie über das Untermenü geöffnet, wird der Benutzer aufgefordert den Barcode zu scannen den er abfragen möchte (Abbildung 5.10a). Es kann sich dabei um ein Lagergut aber natürlich auch um einen Lagerort handeln. Das API sendet dann eine Anfrage an den Server die alle Informationen zu einem Objekt holt (Kapitel 5.6.10). Wird die Maske per Scann aus dem Hauptmenü geöffnet, wird sie bereits mit den Informationen aus der Anfrage geöffnet und angezeigt (Abbildung 5.10b).

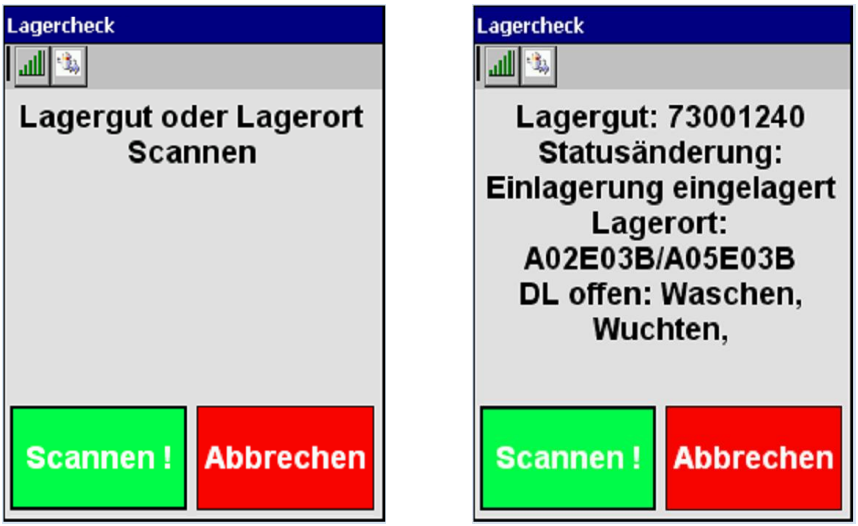

Abbildung 5.10 - Lagercheckmaske

a) Lagercheckmaske bei Start aus dem Untermenü. b) Lagercheckmaske nach dem Verarbeiten der Anfrage

Findet die Anfrage keine Information wird dem Benutzer statt der Information über das Objekt eine entsprechende Meldung angezeigt.

5.4.8 DIENSTLEISTUNGEN

Menüpunkt „2 – Dienstleistungen“ wählt der Mitarbeiter des Reifenhotels wenn er offene Dienstleistungen abarbeiten will. Dazu muss er allerdings zuvor die Ebene auswählen auf der er sich befindet. Die Anzahl der Ebenen ist eine Einstellung die der Administrator hinterlegt und beim Maskenaufruf vom Client abgerufen wird (Kapitel 5.6.13). Je nach Anzahl der Ebenen ändert sich die Form der Auswahlmaske. Im ursprünglichen Reifenhotel gibt es vier Ebenen (Stockwerke). Dies wird sich auch in näherer Zukunft nicht ändern. Allerdings wurde vom Kunden bereits 2017, ein zweites Reifenhotel eingerichtet das 2018 in Betreib gehen soll. Dabei handelt es sich um ein weit kleineres Gebäude mit nur einer Ebene. Aus Sicht des Entwicklers bleibt die Tatsache, dass die Anzahl der Ebenen variieren kann und die Auswahlmaske dementsprechend anzuzeigen ist. Gibt es nur eine Ebene kommt der Mitarbeiter direkt auf die Bearbeitungsmaske (Abbildung 5.11c). Die Abfrage nach offenen Dienstleistungen wird bereits beim Öffnen der Maske im Hintergrund abgesetzt (Kapitel 5.6.4). Als Ebenen Parameter wird automatisch die Erste Ebenen übergeben. Für zwei bis vier Ebenen wird die Auswahlmaske wie in Abbildung 5.11a dargestellt angezeigt. Ab fünf Ebenen kann die gewünschte Ebene in einem Drop-Down Menü ausgewählt werden (Abbildung 5.11b). Der Benutzer kann beim Navigieren durch das Menü die Steuertasten auf der Tastatur verwenden, die um den gelben Button herum in der Mitte des Geräts angebracht sind. Mit dem grünen Button auf der Tastatur bestätigt er seine Auswahl und fragt die offenen Dienstleistungen für diese Ebene ab. Ist für eine Ebene keine Dienstleistung vorhanden wird der Benutzer auf die Fehlermaske weitergeleitet (Abbildung 5.6e). Durch Bestätigen des Fehlers kommt er wieder zurück zur Auswahlmaske. Wird ein Lagergut mit offenen Dienstleistungen gefunden wird es dem Mitarbeiter wie in Abbildung 5.11c dargestellt, angezeigt. Zu diesem Zeitpunkt wurde der Status des Lagerguts bereits auf ‚Dienstleistung angenommen‘ gesetzt. Durch scannen des Lagerorts sendet der Scanner eine entsprechende Anfrage an den Server und der Status des Lagerguts wird auf ‚Ausgelagert zur Dienstleistungsdurchführung‘ gesetzt. Durch Betätigung des roten Buttons am Display oder der Tastatur kommt der Benutzer wieder zur Auswahlmaske zurück. Analog zur Auslagerungsmaske wird auch hier im Hintergrund eine entsprechende Meldung an den Server geschickt, sollte der Mitarbeiter die Dienstleistung doch nicht annehmen.

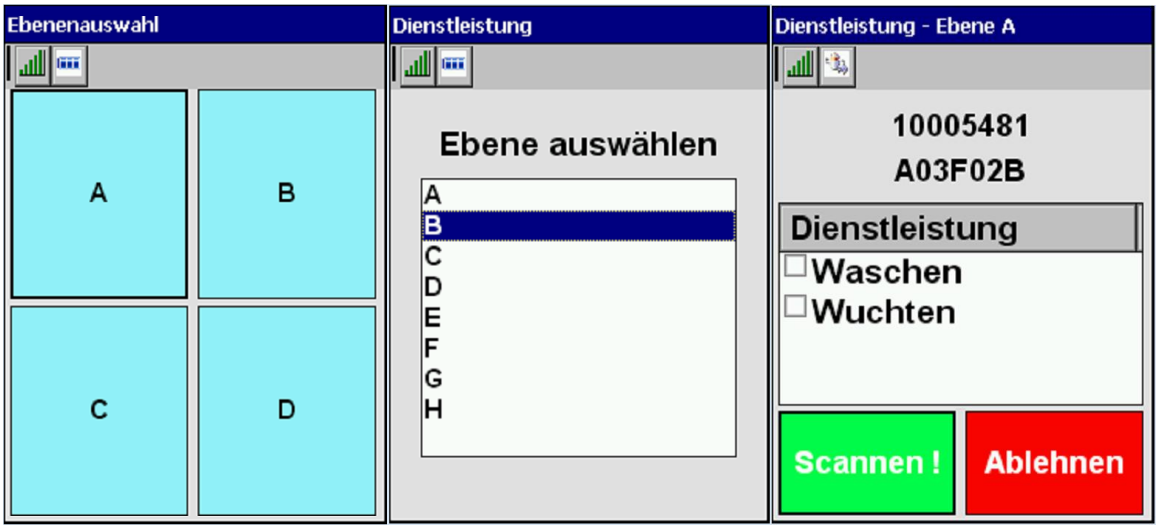

Abbildung 5.11 - Ebenen- und Dienstleistungsmaske

a) Auswahlmaske bei vier Ebenen. b) Auswahlmaske mit Drop-Down Menü bei fünf oder mehr Ebenen. c) Bearbeitungsmaske mir List der offenen Lagergut Id, Lagerort, Dienstleistungen.

5.4.9 UMLAGERN

Menüpunkt „3 – Umlagern“ bietet die Möglichkeit Lagergüter von einem Lagerort zum anderen Umzulagern. Der Mitarbeiter muss dazu nur das betroffenen Lagergut scannen und den neuen Lagerort. Der Client verschickt daraufhin die Anfrage zur Umlagerung an den Server (Kapitel 5.6.7). Erfolg wird mit einer entsprechenden Meldung zum neuen Lagerort des Lagerguts kommuniziert. Schlägt die Anfrage fehl, beziehungsweise ist sie unzulässig, wird der Benutzer zur Fehlermaske weitergeleitet. Ob der Benutzer zuerst das Lagergut oder den Lagerort scannt ist dabei unwichtig. Die Applikation erkennt von selbst ob es sich bei dem gescannten Barcode um ein Lagergut oder einen Lagerort handelt.

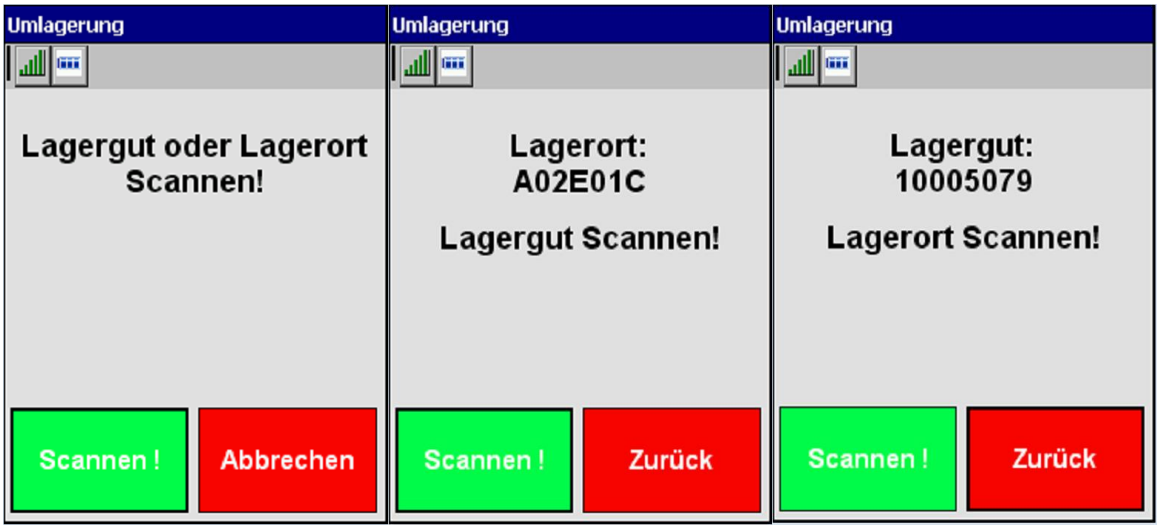

Abbildung 5.12 - Umlagerungsmaske

a)Maske zum Umlagern von Lagergütern. b) Ein Lagerort wurde bereits gescannt, das Lagergut fehlt noch. c) Ein Lagergut wurde bereits gescannt, der Lagerort fehlt noch. d) Die Umlagerung war erfolgreich. e) Die Umlagerung verursachte einen Fehler.

5.4.10 INVENTUR

Die Maske unter Menüpunkt „4 - Inventur“ unterscheidet sich optisch nicht von der Maske unter Menüpunkt 3. Auch hier benötigt eine erfolgreiche Anfrage ein gescanntes Lagergut und einen Lagerort. Die URL der Anfrage ist jedoch eine andere (Kapitel 5.6.5). Inventuren werden vom Administrator über die Weboberfläche gestartet. Ist zum Zeitpunkt der Anfrage keine Inventur gestartet bekommt der Benutzer eine entsprechende Fehlermeldung auf der Fehlermaske angezeigt.

5.5 APPLIKATIONSARCHITEKTUR (MOBILER CLIENT)

Herzstück der Applikation ist die Klasse *AppController*. Sie ist als Singleton implementiert und wird beim Programmstart instanziiert. Die Klasse *AppController* verwaltet globale Variablen wie die Benutzerdaten des eingeloggten Benutzers und dessen Session, Stammdaten wie die Anzahl der hinterlegten Lagerebenen und das Timeout nach dem ein Benutzer abgemeldet werden soll. Zusätzlich bietet sie Funktionen an die entweder immer wieder verwendet werden oder einen globalen Kontext haben, wie zum Beispiel die Funktionen zum ein- und ausloggen eines Benutzers oder zum Speichern der Systemvariablen.

Eine ähnliche Funktion haben die Klassen *Logger*, *FormController, JSONParser* und *API*. Aus Gründen der Übersicht und Wartbarkeit wurden diese Funktionen aber nach ihren jeweiligen Aufgabenbereichen aufgegliedert und in eigene Klassen gepackt. Nachdem der App Controller instanziiert wurde erzeugt sich dieser jeweils eine Instanz jeder der Helferklassen.

Die Klasse *Logger* bietet Funktionen zum mitprotokollieren der Tätigkeit der Applikation an. Als globale Variablen besitzt das Objekt den Pfad der Logdatei und das erwünschte Log Level, daher welche Einträge letztendlich wirklich protokolliert werden sollen. Die Werte dieser Variablen liest sich die

Klasse aus der, im Flashspeicher hinterlegten, Datei ‚*webconfig.txt*' aus. Diese Datei dient in erster Linie dazu die korrekte URL des Reifenhotel Services zu hinterlegen. Sie wurde um die Einstellungen für die Klasse *Logger* erweitert. Das ursprüngliche Konzept diese Einstellungen über die Web-Maske einstellbar zumachen und beim Start über das API einzulesen wurde verworfen, da auf Wunsch des Kunden das Logging unabhängig von der Verbindung zum Service betrieben werden soll.

Die Klasse *FormController* verwaltet alle Masken die innerhalb der Applikation zum Einsatz kommen und ist für das Navigieren zwischen diesen verantwortlich. Beim Erzeugen des Form Controllers werden alle Masken erzeugt und als Member des Objekts gespeichert. Die Navigationsfunktion blendet dann die gewünschte Maske ein und die bisher angezeigte aus. Die Funktion *Start* übergibt die als Einstiegspunkt definierte Maske als Rückgabewert. Die Funktion *CloseAllForms* sorgt dafür, dass beim Beenden der Applikation alle Masken korrekt geschlossen werden.

Die Klasse API implementiert die Zugriffe auf den Webservice. Beim Erstellen einer Instanz liest der Konstruktor die hinterlegte URL aus der Konfigurationsdatei. Sie hält außerdem die statische Variable *API_Token* die in Kapitel 3.1.4.5 erwähnt wird. Ändert sich dieser Token im System muss er direkt in dieser Klasse geändert werden und der Code muss neu kompiliert und ausgeliefert werden. Dieser Token kann nicht in der Konfigurationsdatei hinterlegt werden, da er sonst ja für Benutzer sichtbar wäre und ausgelesen werden könnte. Die einzelnen Funktionen der Klasse sind eine Art Adapter für die HTTP-Anfragen an den Reifenhotel Service zu sehen. Sie werden aus einem gewissen Kontext aufgerufen und bekommen Parameter wie zum Beispiel eine Lagergut-ID und einen Lagerort übergeben. Die Funktion baut daraus einen gültigen Request mit Session-Token, Benutzer-ID und API-Token und sucht die richtige URL für die benötigte Anfrage heraus. Die aufrufende Maske bekommt davon nichts mit. Sie ruft lediglich die API–Funktion, mit den ihr bekannten Parametern, auf und bekommt ein fertiges Objekt zurück, in dem alle Informationen enthalten sind die sie benötigt.

Dass aus der Antwort vom Server ein Objekt wird ist der Verdienst der Klasse *JSONParser*. Sie prüft die empfangenen Zeichenketten und zerlegt sie in Objekte der Klasse *ResponseObject*. Ein solches Objekt besteht immer aus den folgenden Komponenten:

- string Lagergut – Die Id des Lagerguts.
- string StatusAenderung – Der Status der durch die Anfrage erzeugt wurde.
- string Lagerort – Aktueller Lagerort, wenn sich das Lagergut im Lager befindet.
- string Lagerplatz – Aktueller Lagerort außerhalb des Lagers (Checkpoints, Sammelplätze, Transportwägen, etc)
- bool DienstLeistungenOffen – sind für dieses Lagergut noch Dienstleistungen offen.
- string Zustellung – Wo hin muss das Lagergut ausgelagert werden.
- string Kundentermin – Wann muss das Lagergut ausgelagert werden.
- string ErrorCode – Fehlerkategorie, falls die Anfrage einen solchen verurschte.
- List<TaskObject> Dienstleistungen – List mit den offenen Dienstleistungen

Die einzelen Member werden je nach Kontext der Anfrage befüllt. Die Aufrufende Maske holt sich die Informationen die sie braucht. Die Liste der offenen Dienstleistungen ist eine Liste von Ojekten des Typs *TaskObject* das wiederum aus einem Code, einer Bezeichnung und einem Status besteht.

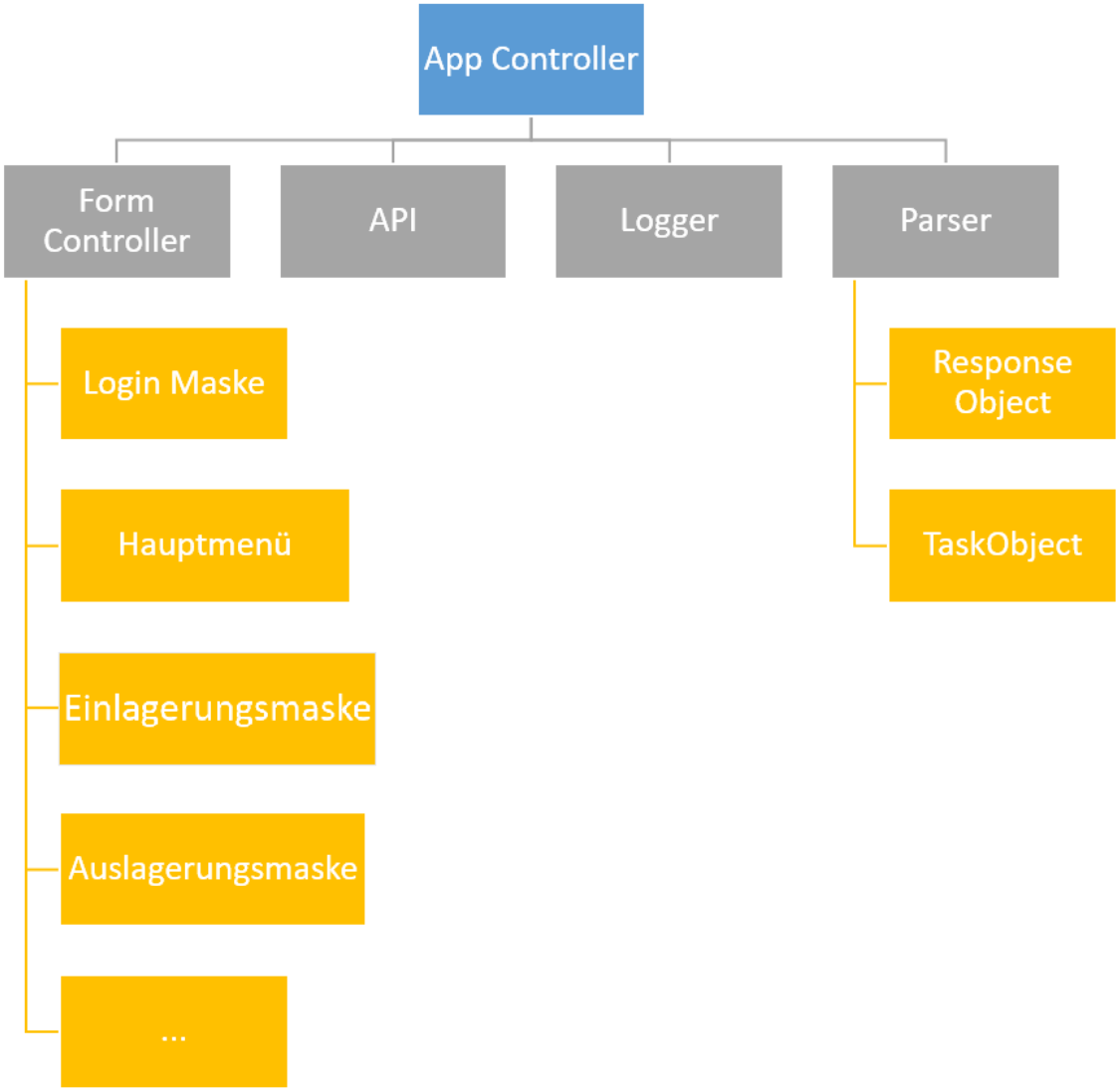

Abbildung 5.13 - Hierarchie der Klassen innerhalb der Client Software

Der Programmstart erfolgt wie in Listing 5.4 dargestellt, indem der *AppController* instanziiert wird. Dieser registriert seine Helfer (**JSONParser**,**API**,…) und der Form Controller holt sich die Maske mit der die Applikation starten soll und übergibt sie der Funktion *Application.Run*.

```
1.  static class Program
2.  {
3.      static void Main()
4.      {
5.          AppController Controller = AppController.Instance;
6.          Controller.RegisterControllers();
7.          Application.Run(Controller.mFormController.Start());
8.      }
9.  }
```

Listing 5.4

Die Masken sind so implementiert, dass jeder Arbeitsvorgang in einer eigenen Windows Form[44] abgebildet ist. Es gibt daher eine Windows Form für den Einlagerungsprozess, den Auslagerungsprozess, um Dienstleistungen abzuarbeiten, den Umlagerungsprozess und die Inventur. Dazu kommen noch die Masken für den Lagercheck, die beiden Menümasken und die Anmeldemaske, die keinen Arbeitsvorgang abbilden aber notwendig sind um eine funktionierende Applikation zu bilden.

Jede Maske verfügt über einen initialen Status der Größe, Standort, Farbe und Beschriftung der einzelnen Steuerelemente, beim Aktivieren der Maske, definiert. Je nach Benutzerinteraktion verändert sich der Status und damit eventuell auch die Position und Beschriftung einzelner Steuerelemente. Als Beispiel für diesen Vorgang sollen hier zwei Beispiele angeführt werden.

Als erstes Beispiel dient die Auslagerungsmaske. Sie besteht im Ursprungszustand aus zwei Buttons und drei Labels. Wurde eine Auslagerung gefunden, werden aller drei Labels benötigt und angezeigt. (Abbildung 5.14a). Wurde die Auslagerung erfolgreich durchgeführt werden die Labels *Header* und *Footer* nicht benötigt. Das Label *Body* soll aber möglichst viel Platz haben um alle Informationen anzeigen zu können (Abbildung 5.14b). Daher werden für diesen Status die Labels *Footer* und *Header* ausgeblendet, die Position von Label *Body* wird nach oben verschoben und die Größe wird angepasst.

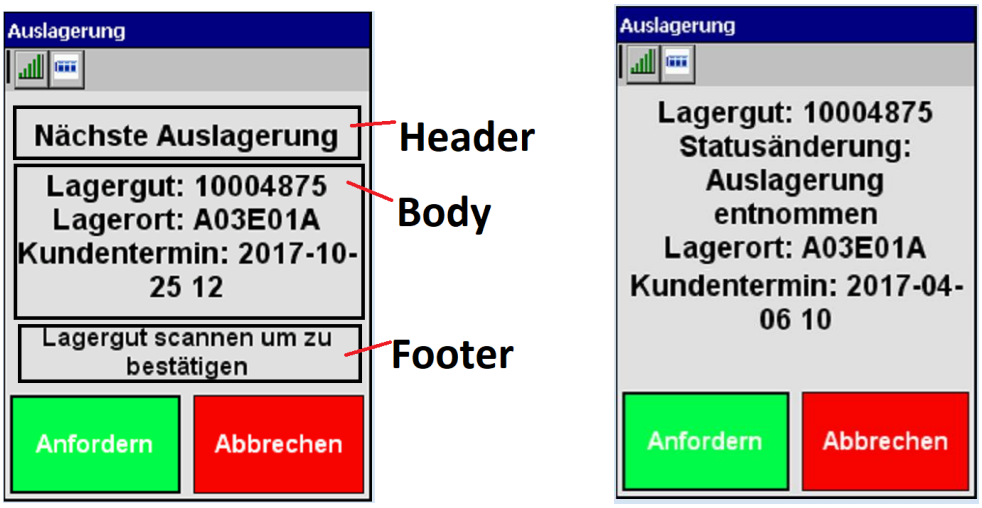

Abbildung 5.14 - Dynamische Labels
a) Auslagerungsmaske bei eingegangener Auslagerung. b) Erfolgsmeldung

Bei der Einlagerungsmaske sollen dem Benutzer offene Dienstleistungen, gleich beim Einlagern des Lagerguts, angezeigt werden. Daher muss für diesen Maskenzustand das Label Header etwas nach oben verschoben werden und das Steuerelement *ListView* wird eingeblendet. In allen anderen Zuständen der Maske ist dieses Element unsichtbar (Abbildung 5.15a und b).

[44] Vlg. https://docs.microsoft.com/en-us/dotnet/framework/winforms/; Microsoft – WinForms; Letzter Aufruf am 22.11.2017

Abbildung 5.15 - Eingeblendete Liste

a) Einlagerungsmaske nach dem ein Lagergut gescannt wurde. b) Die Einlagerung war erfolgreich. Es wurden Dienstleistungen für dieses Lagergut gefunden.

Durch diese Vorgehensweise muss nicht für jeden einzelnen Arbeitsschritt eines Vorgangs eine eigene Maske erstellt werde. Dies wäre übersichtlicher, allerdings müssten temporäre Informationen wie gescannte Barcodes dann ebenfalls global gespeichert werden um von der jeweils nächsten Maske wieder verwendet werden zu können. Außerdem müssten diese Masken dann während des Arbeitsvorgangs geladen werden was sich wiederum auf die Performance auswirken würde. Alle Masken bei Programmstart zu laden würde diesem Problem nur bedingt abhelfen da auch das ein und ausblenden von bereits geladenen Masken mit einer gewissen Reaktionszeit verbunden ist. Zusätzlich würde es den Programmstart erheblich verlangsamen, wenn über 30 Masken geladen werden müssten. Das Verschieben der Steuerelemente, je nach Arbeitsschritt und Situation, hat sich hier als weitaus perfomanter erwiesen. Der Source Code der Applikation wird dadurch aber auch unübersichtlicher.

5.6 API ZUGRIFFE

Es sollen nun die Zugriffe auf den Reifenhotelservice aus Sicht des Clients betrachtet werden. Dazu werden die einzelnen Funktionen der Klasse API betrachtet, die Anfragen an den Service die sie implementieren, die Parameter die diese Anfragen benötigen und der Kontext in dem sie aufgerufen werden.

5.6.1 REGISTER USER

Zweck und Anwendung

Benutzer identifizieren und authentifizieren. Wird auf der Anmeldemaske aufgerufen um das Hauptmenü zu betreten. Der hier empfangene Session-Token wird für die weiteren Funktionen benötigt.

URL

*„https://dev-rh.motiondata.at/services/v1/**benutzer/login**“*

http-Methode: POST

Parameter

- ‚user_id' – Eindeutige Benutzerkennung
- ‚device_id' – Eindeutige Gerätekennung

Payload-Format: x-www-form-urlencoded

Antwort: application/JSON

5.6.2 LOGOUT USER

Zweck und Anwendung

Laufendende Session beenden. Wird auf der Abmeldemaske und vom Hauptmenü aus aufgerufen um den Benutzer abzumelden. Wird zusätzlich auf der Anmeldemaske verwendet wenn der Benutzer auf einem anderen Gerät noch eine laufende Session besitzt.

URL

*„https://dev-rh.motiondata.at/services/v1/**benutzer/logout**“*

http-Methode: POST

Parameter

- ‚user_id' – Eindeutige Benutzerkennung
- ‚cancel_current' – "true" wenn andere, laufende Session beendet werden soll.

Payload-Format: x-www-form-urlencoded

Antwort: application/JSON

5.6.3 EDIT LAGERGUT

Zweck und Anwendung

Wird benutzt um die Bewegung eines Lagerguts zu kommunizieren. Wird von der Einlagerungsmaske, der Auslagerungsmaske und der Dienstleistungsmaske verwendet, da das Abarbeiten einer Dienstleistung durch Einlagern auf dem jeweiligen Gerät (z.B Waschmaschine, Wuchtmaschine, …) passiert.

URL

*„https://dev-rh.motiondata.at/services/v1/**lagergut/edit**"*

http-Methode: POST

Parameter

- ‚user_id' – Eindeutige Benutzerkennung
- ‚sessiontoken ' – Session
- ‚lagergut_id' – Lagergut das bewegt werden soll
- ‚standort_id ' – Lagerort an den das Lagergut bewegt werden soll

Payload-Format: x-www-form-urlencoded

Antwort: application/JSON

5.6.4 EDIT DIENSTLEISTUNGEN

Zweck und Anwendung

Lagert ein Lagergut zur Durchführung der Dienstleistung(en) aus. Wird auf der Einlagerungsmaske und auf der Dienstleistungsmaske verwendet.

URL

*„https://dev-rh.motiondata.at/services/v1/ **lagergut /status_dienstleistungsdurchfuehrung**"*

http-Methode: POST

Parameter

- ‚user_id' – Eindeutige Benutzerkennung
- ‚sessiontoken ' – Session
- ‚lagergut_id' – Lagergut das bewegt werden soll
- ‚standort_id' – Lagerort/Standort an dem sich das Lagergut gerade befindet
- ‚annehmen' – „true" wenn der Benutzer die Bearbeitung der Dienstleistung annimmt.

Payload-Format: application/JSON

Antwort: application/JSON

5.6.5 INVENTUR

Zweck und Anwendung

Wird verwendet um ein Lagergut und seinen aktuellen Lagerort für den Zweck einer Inventur zu erfassen. Der Aufruf erfolgt über die Maske „Inventur".

URL

*„https://dev-rh.motiondata.at/services/v1/**inventur**"*

http-Methode: POST

Parameter

- ‚user_id' – Eindeutige Benutzerkennung
- ‚sessiontoken ' – Session
- ‚obj_id1' – Lagergut das inventarisiert werden soll
- ‚obj_id2' – Lagerort an dem sich das Lagergut aktuell befindet

Payload-Format: x-www-form-urlencoded

Antwort: application/JSON

5.6.6 INVENTUR UPDATE

Zweck und Anwendung

Wurde ein Lagergut im Zuge einer Inventur bereits an einem anderen Lagerort erfasst bietet diese Funktion dem Benutzer die Möglichkeit diesen Eintrag zu überschreiben. Der Aufruf erfolgt über die Maske Inventur.

URL

*„https://dev-rh.motiondata.at/services/v1/**inventur/update**"*

http-Methode: POST

Parameter

- ‚user_id' – Eindeutige Benutzerkennung
- ‚sessiontoken ' – Session
- ‚obj_id1' – Lagergut das inventarisiert werden soll
- ‚obj_id2' – Lagerort an dem sich das Lagergut aktuell befindet

Payload-Format: x-www-form-urlencoded

Antwort: application/JSON

5.6.7 UMLAGERUNG

Zweck und Anwendung

Wird benutzt um ein Lagergut von seinem aktuellen Lagerort an einen anderen, freien Lagerort um zu lagern.

URL

*„https://dev-rh.motiondata.at/services/v1/**lagergut/umlagerung**"*

http-Methode: POST

Parameter

- ‚user_id' – Eindeutige Benutzerkennung
- ‚sessiontoken ' – Session
- ‚obj_id1' – Lagergut das inventarisiert werden soll
- ‚obj_id2' – Lagerort an den das Lagergut umgelagert werden soll

Payload-Format: x-www-form-urlencoded

Antwort: application/JSON

5.6.8 RESET AUSLAGERUNG

Zweck und Anwendung

Fordert ein Mitarbeiter eine neue Auslagerung an, wird ihm die gefundene Auslagerung zugewiesen und der Status des Lagerguts ändert sich auf ‚Auslagerung angenommen'. Will der Mitarbeiter die Auslagerung nicht annehmen kann er sie mit Hilfe dieser Funktion zurücklegen. Die Anforderung ist ihm dann nicht mehr zugewiesen und der Satus des Lagerguts wird auf ‚Auslagerung angefordert' zurückgesetzt. Der Aufruf erfolgt über die Auslagerungsmaske.

URL

„https://dev-rh.motiondata.at/services/v1/ ***lagergut/reset****“*

http-Methode: POST

Parameter

- ‚user_id' – Eindeutige Benutzerkennung
- ‚sessiontoken ' – Session
- ‚lagergut_id' – Lagergut das inventarisiert werden soll

Payload-Format: x-www-form-urlencoded

Antwort: application/JSON

5.6.9 GET AUSLAGERUNGEN

Zweck und Anwendung

Wird vom Mitarbeiter aufgerufen wenn er wissen will ob in nächster Zeit Auslagerungen zu tätigen sind. Aufruf erfolgt über die Auslagerungsmaske.

URL

*„https://dev-rh.motiondata.at/services/v1/****lagergut/auslagerung****“*

http-Methode: GET

Parameter

- ‚user_id' – Eindeutige Benutzerkennung
- ‚sessiontoken ' – Session

Antwort: application/JSON

5.6.10 LAGERCHECK

Zweck und Anwendung

Wird vom Mitarbeiter aufgerufen wenn er Informationen zu einem Lagergut, Lagerort oder Lagerplatz benötigt. Wird außerdem beim Scannen eines Lagerguts auf der Einlagerungsmaske verwendet um dem Mitarbeiter während des Einlagerungsvorgangs den vorgesehen Lagerort anzeigen zu können.

URL

*„https://dev-rh.motiondata.at/services/v1/**lagergut/info**“*

http-Methode: GET

Parameter

- ‚user_id‘ – Eindeutige Benutzerkennung
- ‚sessiontoken ‘ – Session
- ‚obj_id1‘ – Lagergut, Lagerort oder Lagerplatz über den Informationen abgefragt werden sollen.

Antwort: application/JSON

5.6.11 GET DIENSTLEISTUNGEN

Zweck und Anwendung

Sucht nach offenen Dienstleistungen auf eine Lagerebene. Wird von der Maske zur Ebenen Auswahl aus aufgerufen, wen eine Ebene ausgewählt wurde und sich die Maske zur Bearbeitung von Dienstleistungen öffnet.

URL

*„https://dev-rh.motiondata.at/services/v1/**lagergut/naechste_Dienstleistung**“*

http-Methode: GET

Parameter

- ‚user_id‘ – Eindeutige Benutzerkennung
- ‚sessiontoken ‘ – Session
- ‚ebene‘ – Ebene auf der nach offenen Dienstleistungen gesucht werden soll

Antwort: application/JSON

5.6.12 GET STAMMDATEN

Zweck und Anwendung

Wird direkt nach dem Anmeldevorgang aufgerufen um die Einstellungen für den angemeldeten Benutzer abzurufen.

URL

*„https://dev-rh.motiondata.at/services/v1/**stammdaten/parameter**“*

http-Methode: GET

Parameter

- ‚user_id' – Eindeutige Benutzerkennung
- ‚sessiontoken ' – Session

Antwort: application/JSON

5.6.13 GET EBENEN

Zweck und Anwendung

Wird beim ersten Aktivieren der Maske zur Auswahl der Ebene aufgerufen. Liefert der Applikation die Anzahl der vorhandenen Lagerebenen.

URL

*„https://dev-rh.motiondata.at/services/v1/**lagergut/auslagerung**"*

http-Methode: GET

Parameter

- ‚user_id' – Eindeutige Benutzerkennung
- ‚sessiontoken ' – Session
- ‚obj_id1' – Lagergut das inventarisiert werden soll

Antwort: application/JSON

6 PROJEKTABLAUF, DISKUSSION UND AUSBLICK

Für die Projektabnahme wurde mit dem Kunden eine Testphase vereinbart in der gemeinsam etwaige Fehler ausgebessert und Anpassungen vorgenommen werden sollen. Zusätzlich zu den zuvor durchgeführten, internen Tests wurde dem Kunden, beziehungsweise einem Ansprechpartner innerhalb der IT-Abteilung des Kunden, Benutzerdaten mit Administratorrechten für das Webportal zugewiesen und ein Prototyp der Client Software ausgehändigt. So konnte schon einige Zeit vor dem Echtstart einiges an Feedback gesammelt werden.

Die Dauer der Testphase und die zusätzlich angefallenen Entwicklungsstunden, die sich aus nachträglichen Änderungen der Anforderungen ergaben, waren weder von Kundenseite noch von der Geschäftsführung der Firma MOTIONDATA eingeplant. Tatsächlich wäre es in diesem Fall ratsam gewesen eine weit längere, beziehungsweise interaktivere Analysephase einzuplanen. Die Änderungen die sich ergaben als die Entwicklung bereits weit fortgeschritten war, hätten sich so deutlich reduzieren lassen.

Die fertige Liste der Anforderungen wie sie in Kapitel 1.2 aufgelistet sind, ergab sich allerdings erst während der laufenden Entwicklung. Die saubere Planung eines API erschwert sich dadurch erheblich, da einzelne Funktionen erst im Nachhinein dazukommen und sich an das bereits geschaffene anpassen müssen, anstatt von Anfang an eingeplant werden zu können. Dadurch entstehen oft unschöne Kompromisse und Redundanzen. Konventionen werden gebrochen und die Wartbarkeit des Codes sinkt.

Was das Erscheinungsbild und die Handhabung der Clientsoftware auf den Handheldscannern angeht wurde bereits früh in der Entwicklung ein Prototyp der Applikation, zusammen mit dem Kunden, getestet. Dies hat dazu geführt, dass hier später nur noch kleinere, vergleichsweise 'billige', Änderungen an der Oberfläche vorzunehmen waren.

Als Beispiel soll hier die mehrfache Anpassung der Schriftgröße innerhalb der Clientapplikation genannt werden oder der Umstand, dass eine gescannte Einlagerung, plus Lagerort, in der ursprünglichen Definition immer noch per Knopfdruck bestätigt werden musste, um an den Service gesandt zu werden. Dies wurde im Laufe der Testphase vom Kunden als Umständlich kritisiert und dementsprechend entfernt. Aus Entwicklersicht werden solche Anliegen des Kunden oft als Banalitäten abgetan. Aus Sicht des Benutzers, der täglich damit arbeiten soll, gilt es hier aber eine ideale Lösung zu finden. Derlei Probleme kommen auch erst durch den regelmäßigen Gebrauch einer Software, durch unterschiedliche Benutzer zu Tage. Solche Anpassungen müssen vom Entwickler daher immer eingeplant werden, wenn man ein Produkt schaffen will mit dem auch die Benutzer zufrieden sind.

Kritischer sind hingegen Anpassungen an der Logik und den Abläufen innerhalb einer Applikation. Hier liegt es auch an der Kulanz der entwickelnden Firma, ob vorab festgeschriebene Abläufe im Nachhinein angepasst werden. Fakt ist, dass hier Kosten entstehen und sich der Zeitpunkt Fertigstellung nach hinten verschiebt.

Im konkreten Fall des Reifenhotelprojekts, gab es mehrere solcher kritischer Anpassungen der Definition, die erst im Laufe des Projekts an die Entwickler heran getragen wurden. Dies erklärt sich durch die organisatorischen Gegebenheiten des Projekts. Es handelt sich um eine individual Implementierung für einen einzelnen Kunden. Die Funktionsweise des Systems wurde vom Kunden, der kein technisches Know-how innehat, erdacht. Der Kunde hat noch keine Erfahrungen mit einem System dieser Art. Das System muss in keine bestehende Infrastruktur integriert werden und kann quasi auf der grünen Wiese geplant werden. Das Resultat all dieser Gegebenheiten ist, dass die Anforderungen an das System, mit denen der Kunde das erste Mal an die Firma MOTIONDATA herantrat, sehr vage formuliert waren.

In der Praxis lassen sich Umstände dieser Art kaum vermeiden. Aus Entwicklersicht ist es daher wichtig früh auf den Kunden einzugehen und ihn bei der Formulierung seine Anforderungen zu unterstützen. Dies kann durch gemeinsames Erstellen von Usecases oder auch Prototypen geschehen. Es ist auch die Verantwortung des Entwicklers früh kritische Fragen zu stellen und bei formulierten Abläufen ins Detail zu gehen. Aus Sicht der Projektleitung ist es wichtig diese Umstände finanziell und auch zeitlich ein zu planen und Kunden und anderen Entscheidungsträgern dies von Anfang an verständlich zu machen.

Die Erste Entwicklungsphase des Projekts Reifenhotel dauerte von Oktober 2015 bis Juli 2016 und endete mit der Auslieferung der ersten fünf Handheldscanner an den Kunden. Die Scanner enthielten die erste Testversion der Clientapplikation. Ausgewählte Mitarbeiter des Kunden erhielten Zugangsdaten zum Webportal.

Der Kunde hatte bereits in vorangegangene Reifensaison (Frühjahr 2016) angefangen Reifen und Räder in das bereits erbaute Lagergebäude einzulagern. Mit diesen Lagergütern wurde die gesamte Anwendung ab diesem Zeitpunkt getestet. In dieser Phase wurden einige Lücken im organisatorischen Ablauf geschlossen und viele Anpassungen am Inhalt und Erscheinung des Webportals sowie auch der Clientsoftware durchgeführt.

Zusätzlich zum manuellen Test wurde von der Firma MOTIONDATA eine Schnittstelle vom DMS zum Reifenhotel zur Verfügung gestellt. Dabei handelt es sich um einen Dienst der am Datenbank Server des jeweiligen Kundencenters läuft und in regelmäßigen Abständen prüft ob über das DMS neue Ein- oder Auslagerungen getätigt wurden. Somit waren die Kundencenter, die mit dem MOTIONDATA DMS verwaltet werden bereits an das Reifenhotel angeschlossen. Die so eingelangten Einlagerungen konnten dann im Webportal weiter bearbeitet werden. Nach einer zwei wöchigen Testphase wurden die beiden API-Funktionen, die die Schnittstelle benötigt, an zwei weitere DMS-Hersteller, kommuniziert. So konnten auch die weiteren Kundencenter an die Reifenhotel Infrastruktur angeschlossen werden.

Im Herbst 2016 wurden die bisher gesammelten Daten von der Testumgebung in die Produktivumgebung übernommen und die Schnittstellen von den Kundencentern wurden auf die produktive URL umgestellt. Im Falle der von Motiondata bereitgestellten Schnittstelle passiert dies über einen Parameter in der Konfiguration des Diensts. Auch die Anfragen über den Handscanner wurden über die Konfigurationsdatei im Flashspeicher auf die produktive URL geleitet. Zudem wurden drei weitere Scanner für den Lagerbetrieb ausgeliefert. Softwareupdates am Scanner wurden zu

diesem Zeitpunkt nicht mehr über die Fernwartung durchgeführt, sondern bereits von dem Vorort zuständigen Mitarbeiter des Kunden. So wurde pünktlich zum Echtstart Version 16 (!) der mobilen Clientsoftware auf allen Scannern installiert.

Auch nach dem Echtstart blieb die Entwicklung der Firma MOTIONDATA in engem Kontakt mit dem zuständigen Projektleiter des Kunden. So wurden über das Bug-Tracker-Tool Mantis (https://www.mantisbt.org/) über 70 Punkte mit Bezug zur Reifenhotel Software erfasst. Dabei handelt es sich um Fehler und Erweiterungen. Erfasst wurden diese Punkte sowohl von den Entwicklern selbst als auch vom Projektleiter des Kunden.

Auch für die Zukunft (Stand Oktober 2017) stehen bereits weitere Anforderungen an. Eine davon ist die sogenannte Mandantenfähigkeit des Systems. Diese Anforderung entstand im Rahmen der Planung eines weiteren Reifenhotels, beziehungsweise eines weiteren Standorts. Dieses soll organisatorisch von dem zuerst erbauten Reifenhotel und natürlich auch von weiteren, getrennt sein. Dies bedeutet, dass das ganze System in verschiedene Mandanten (Standorte) unterteilt wird, die miteinander nichts zu tun haben. Zum Beispiel existiert ein Benutzer der im Reifenhotel Erdberg angelegt ist im Reifenhotel Floridsdorf gar nicht. Dies ist mit erheblichen Umstrukturierungen im Backend und der Datenbank verbunden.

Eine weitere Anforderung die bereits beauftragt wurde ist die Einlagerung von Lagergütern auf zwei Lagerplätzen. Hier soll im Falle eines vollen Lagers errechnet werden ob ein Lagerplatz, auf Grund der Dimensionen der eingelagerten Räder, nicht voll genutzt wird. So kann auch in einem vermeintlich vollen Lager noch Platz gefunden werden. Diese Änderung betrifft nicht nur die Einlagerungslogik sondern auch die Verarbeitung am Scanner. Hier müssen in Zukunft zwei Lagerplätze angezeigt und natürlich auch gescannt werden. Dies ist nicht ohne grobe Eingriffe in die bereits implementierte Logik der Clientsoftware möglich.

Ein weiterer wichtiger Punkt ist die Zukunft der mobilen Clients. Wie bereits mehrfach erwähnt endet der Support für Windows CE 6.0 im Jahr 2018. Konkret auf dieses Problem angesprochen, gab der Technische Support der Firma Nordic ID folgende Antwort:

"Hi Maximilian,

Microsoft has stated that they will continue maintenance for Windows CE 6.0 OS until April 10th 2018. Nordic ID provides these fixes to customers by incorporating them into device firmware releases bundled together with device hardware specific software fixes and improvements. Although Microsoft's OS support will end in 2018, Nordic ID will keep supporting the device OS with regards to hardware. Usually, the software support will continue for a minimum of three years after the device EOL statement. For the time being, no EOL date has been set to any of Nordic ID Windows CE 6.0 based devices.

In any case, Android seems to be our next operating system so if you want to jump on it before our new device is ready, then please take a look our EXA51 and EXA31 products:

https://www.nordicid.com/en/home/products-barcode-uhf-rfid-reader-writer/extensions-for-smart-devices/nordic-id-exa51/

If you have any further questions, please don't hesitate to contact us.

Best Regards, ***Turo Rantanen | Technical Support Engineer****"*

Bei dem Gerät das unter dem angehängten Link vorgestellt wird handelt es sich um den Nordic ID EXA51, einen Scanner auf dem ein Smartphone direkt aufgesetzt werden kann. Dabei muss es sich laut Beschreibung um kein bestimmtes Smartphone handeln. Als Betriebssystem für diese Geräte werden außerdem Android und iOS angeführt. Da der Kunde das Gerät Nordic ID Morphic vor allem deshalb ausgewählt hat weil das Gerät sehr handlich ist, kommen diese Geräte eher nicht in Frage.

Wie aus dem Email hervor geht, wird die Firma Nordic ID in Zukunft sehr wahrscheinlich auf das Betriebssystem Android setzen. Konkret bedeutet dies, dass ab einem gewissen Zeitpunkt, in nicht allzu fernen Zukunft, neue Geräte nur noch mit dem Betriebssystem Android ausgeliefert werden. Für diesen Zeitpunkt muss die Firma MOTIONDATA ein Konzept parat halten.

Eine Möglichkeit wäre die Applikation selbst zu entwickeln sobald mehr Informationen zur verwendeten Android Version vorhanden sind. Die zweite Möglichkeit wäre diese Aufgabe einer externen Firma zu übertragen, die auf diesem Gebiet Erfahrung hat.

Eine weitere Problemstellung des Projekts Reifenhotel die es zu lösen gilt ist die Bereitstellung des entwickelten Backends nicht nur für die angebundene, interaktive Weboberfläche, sondern auch für andere Systeme die mit dem Backend kommunizieren müssen. Wie in Kapitel 1 erwähnt handelt es sich bei dem Areal für das diese Softwarelösung bereitgestellt wird um ein größeres Gelände auf dem mehrere Werkstätten und Verkaufszentren nebeneinanderliegen. Diese sind in unabhängige Geschäftseinheiten unterteilt, da sie zwar alle zum selben Konzern gehören aber unterschiedliche Marken vertreten. Da jeder Markenhersteller seine ganz eigenen isolierten Systeme betreibt, deckt ein Dealer Management System immer nur eine gewisse Bandbreite an Marken und Schnittstellen zu deren Systemen ab. Dieser Umstand führt dazu, dass auf dem Areal des Kunden drei verschiedene Dealer Management Systeme im Einsatz sind. Alle diese Systeme müssen aber mit dem Backend kommunizieren. Hierfür braucht es ein transparentes API das auch für außenstehende nach zu vollziehen ist.

Einen weit größeren Umfang an Funktionen aus dem Backend benutzen die Handheldscanner, die von den Lagermitarbeitern verwendet werden. Dieses API muss in Zukunft von einer Android Applikation angesprochen werden die eventuell von einer Gruppe externer Entwickler verwendet werden. Für diese Entwickler sind die Abläufe und die Logik des Systems neu und vermutlich nicht gleich nachvollziehbar.

Zusammen gefasst kann also gesagt werden, dass die Anforderung darin besteht ein Backend zur Verfügung zu stellen, dass nicht nur auf verschiedenen Betriebssystemen laufen muss sondern auch von unterschiedlichen Entwicklern außerhalb der Firma MOTIONDATA angesprochen werden muss.

Hierfür eignet sich das derzeitige API nur bedingt. Wenn es längerfristig relevant bleiben soll muss es hier in Zukunft Anpassungen geben.

7 LITERATURVERZEICHNIS

British Columbia Institute of Technology (2017): CodeIgniter – User Guide;

Kübeck, Sebastian (2011): Web-Sicherheit - Wie Sie Ihre Webanwendung sicher vor Angriffen schützen; MITP München(siehe Seite 94 ff.)

Kunz, Stafan & Stafan Esser (2008): PHP – Sicherheit: PHP/MySQL-Webanwendung sicher programmieren; dPunkt Verlag; (siehe Seite 180 ff.)

Massé, Mark (2102): REST API Design Rulebook; O'Reilly (siehe Seite 5)

Ponemon Institute LLC (April 2014): The SQL Injection Threat Study;

Richardson, Leonard & Sam Ruby (2007): RESTful Web Services; O'Reilly (siehe Preface Seite XV)

Schäfers, Tim Philipp (2016): Hacking im Web; Franzis Verlag; (siehe Seite 92 ff. und 103 ff.)

Snyder, Myer, Southwell (2010): Pro PHP Security – From Application Security Principles to the Implementation of XSS Defense; Apress; (siehe Seite 45, 93 ff.)

Stuttard, Dafydd & Marcus Pinto (2011): The Web Application Hackers Handbook – Finding and Exploiting Security Flaws; Wiley Publishing, Inc. (siehe Seite 436, 439)

8 ABBILDUNGSVERZEICHNISS