

Studienarbeit

Integrierung einer Three-Tier Web Applikation für Lizenz Management System auf Microsoft Teams

im Studiengang Softwaretechnik und Medieninformatik Informatik
der Fakultät Informationstechnik Wintersemester 2023

Ertugrul Sevgili

Zeitraum:

Datum:

Prüfer: Prof. Dr. Dipl.-Ing. Jörg Friedrich

Inhaltsverzeichnis

1 EINLEITUNG	1
1.1 ABKÜRZUNGEN	1
2 GRUNDLAGEN	2
2.1 LABOREINHEIT- SOFTWAREARCHITEKTUR	2
2.2 FESTLEGUNG DER ANFORDERUNGEN.....	2
2.3 UML KLASSENDIAGRAMM.....	3
2.4 UML STATE MASCHINE DIAGRAMM.....	5
2.5 THREE-TIER (DREISCHICHTIGE) ARCHITEKTUR.....	8
2.6 RDBMS	9
2.7 JPA ORM.....	9
2.8 REST UND JAX RS.....	11
2.9 JUNIT TEST.....	13
3 REALISIERUNG	15
3.1 SOFTWARE-SETUP	15
3.1.1 MySQL RDBMS.....	15
3.1.2 Quarkus - Gradle	16
3.1.3 Java Development Kit (JDK)	18
3.1.4 MS Teams Template Projekt mit React.js.....	18
3.1.4.1 Visual Studio Code.....	18
3.1.4.2 Teams Toolkit	19
3.1.4.3 Erstellen eines Teams-Entwicklermandanten	19
3.1.4.4 Erstellen des Projektarbeitsbereichs.....	19
3.1.4.5 Ausführen der App	21
3.2 INTEGRIEREN DER SOFTWAREARCHITEKTUR LABOREINHEIT AUF TEAMS APP	22
3.3 ENVIRONMENT VARIABLES	24
3.4 STARTEN CLIENT UND BACKEND.....	25
3.5 STELLUNG DER APP ZUR VERFÜGUNG FÜR TEAMS	25
3.6 ANSICHTEN DER APP	28
3.6.1 Login.....	28
3.6.2 Home Page	28
3.6.2.1 Admin System.....	28
3.6.2.2 Admin Company	30
3.6.3 Users Page	31
3.6.3.1 Admin System.....	31
3.6.3.2 Admin Company	32
3.6.4 Contracts Page.....	33
3.6.4.1 Admin System.....	33
3.6.4.2 Admin Company	34
LITERATURVERZEICHNIS	35

1 Einleitung

Erläuterung der Aufgabenstellung und den Randbedingungen

1.1 Abkürzungen

RDBMS Relational Database Management System

ORM Object-Relational Mapping

JPA Jakarta Persistence API

JAX-RS Jakarta RESTful Web Services

JSON JavaScript Object Notation

UML Unified Modelling Language

DAO Data Access Object

CRUD Create-Read-Update-Delete

DML Data Manipulation Language

REST Representational State Transfer

HTTP Hypertext Transfer Protocol

MIME Multipurpose Internet Mail Extensions

URL Uniform Resource Locator

CORS Cross-Origin Resource Sharing

2 Grundlagen

Ziel dieser Studienarbeit ist es, die Software der Laboraufgabe des Softwarearchitektur-Moduls als eine ausführbare Registerkarte, die eine eingebettete Weboberfläche ist, auf Microsoft Teams zu integrieren.

Bei der Integrierung wurde die App in Microsoft Teams Store nicht veröffentlicht. Deswegen muss die App in Microsoft Teams Schnittstelle der Organisation manuell als ZIP-Datei hochgeladen und anschließend jeweiligem Team zur Verfügung gestellt werden.

2.1 Laboreinheit-Softwarearchitektur

Ziel der Laboreinheit ist es, dass die Studenten eine Three Tier Web Applikation, die folgende Technologien enthalten muss, implementieren, um ein Lizenz-Managementsystem anzubieten.

- Daten müssen in einem relationalen Datenbankenmanagementsystem gespeichert und bearbeitet werden. Im Projekt wird MySQL [RDBMS](#) verwendet, um diese Voraussetzung zu erfüllen.
- Objekte, die zu einem in einer objektorientierten Programmiersprache geschriebenes Anwendungsprogramm gehören, müssen in dieser relationalen Datenbank angelegt werden. Im Projekt wird [JPA ORM](#) verwendet, um diese Voraussetzung zu erfüllen.
- Die Kommunikation zwischen Server und Client muss durch [REST](#) Architekturstil realisiert werden. Im Projekt wird [JAX-RS](#) verwendet, um diese Voraussetzung zu erfüllen, indem Daten zwischen Client und Server in [JSON](#) gesendet werden. Außerdem anhand Quarkus Build Management System werden [JPA](#), [JAX-RS](#) und JUnit Tests zusammengebunden.
- Eine Benutzeroberfläche muss zur Verfügung gestellt werden. Im Projekt wird React.js verwendet, um diese Voraussetzung zu erfüllen.

2.2 Festlegung der Anforderungen

Die Applikation enthält 3 Benutzertypen:

Admin System:

- Als Produktbesitzer kann ein Admin System ein neues Kundenunternehmen anlegen.
- Beim Anlegen eines Kunden muss der Name und die Anschrift eingegeben werden. Ein optionales Eingabefeld für Adressdetails muss auch in Form angeboten werden.
- Ein Admin System kann alle Angaben inklusive Benutzer und Verträge aller Kunden sehen, bearbeiten oder entfernen.
- Ein Admin System kann seine oder ihre eigenen Profile bearbeiten.
- Ein Filter soll die angezeigten Kunden anhand der Anfangsbuchstaben des Kundennamens einschränken.
- Die Listenansicht soll alle Kundenattribute anzeigen und Button zum Bearbeiten, Löschen, Anzeigen aller zugehörigen Verträge und aller zugehörigen Benutzer enthalten.

Admin Company:

- Ein administrativer Benutzer eines Unternehmens kann alle Angaben aller Benutzer und Verträge des Unternehmens sehen, bearbeiten oder entfernen.
- Beim Anlegen eines Benutzers muss
 - Vor- und Nachname
 - E-Mail-Adresse
 - Handy- und Telefonnummer
 - Rolle (admin oder user) eingegeben werden.
- Beim Anlegen eines Vertrags muss
 - Start- und Ablaufdatum
 - 3 IP-Nummer. Falls die erste IP-Nummer eingegeben wird, muss andere 2 Eingabefelder für IP-Nummer dann aktiviert werden.
 - Der Vertrag muss höchstens zwei, mindestens einem Mitarbeiter zugewiesen werden.
 - Optionale numerische Eingabefelder für Portnummer, Serial Nummer und Issuer-Id müssen in Form angeboten werden.
- Ein Vertrag muss eine Versionsnummer haben und falls Start- und Ablaufdatum oder IP-Nummer geändert wurden, muss die Versionsnummer automatisch um 1 erhöht werden.
- Nach dem Anlegen wird vom Server automatisch erstellter Lizenzschlüssel unter Vertragsdetails erscheint. Admin Company kann vom Server einen neuen Schlüssel fordern.
- Falls Start- und Ablaufdatum oder eine IP-Adresse oder Lizenzschlüssel geändert wird, muss Version Nummer automatisch um 1 erhöht werden.
- Ein Admin Company kann seine oder ihre eigenen Profile bearbeiten, aber darf nicht seine oder ihre Rolle ändern.

User:

- Ein Benutzer kann ihm zugewiesene Verträge sehen.
- Ein Benutzer kann nur die IP-Adresse jeweiligen Vertrags ändern, falls der Vertrag nicht abgelaufen ist.
- Ein Benutzer kann seine eigenen Profile bearbeiten, aber darf nicht seine Rolle ändern.

2.3 UML Klassendiagramm

Die statistischen strukturellen Aspekte der Software-Architektur kann aus dem folgenden [UML](#) Klassendiagramm ersehen werden. In diesem Projekt befinden sich 3 Entitäten:

- Company : Ein Unternehmen muss mindestens eine/-n Mitarbeiter/-in und einen Vertrag haben
- User : Ein Benutzer kann für keinen Vertrag oder viele Verträge zuständig sein.
- Contract : Ein Vertrag muss mindestens einem Benutzer oder kann höchstens 2 Mitarbeitern zugewiesen werden.

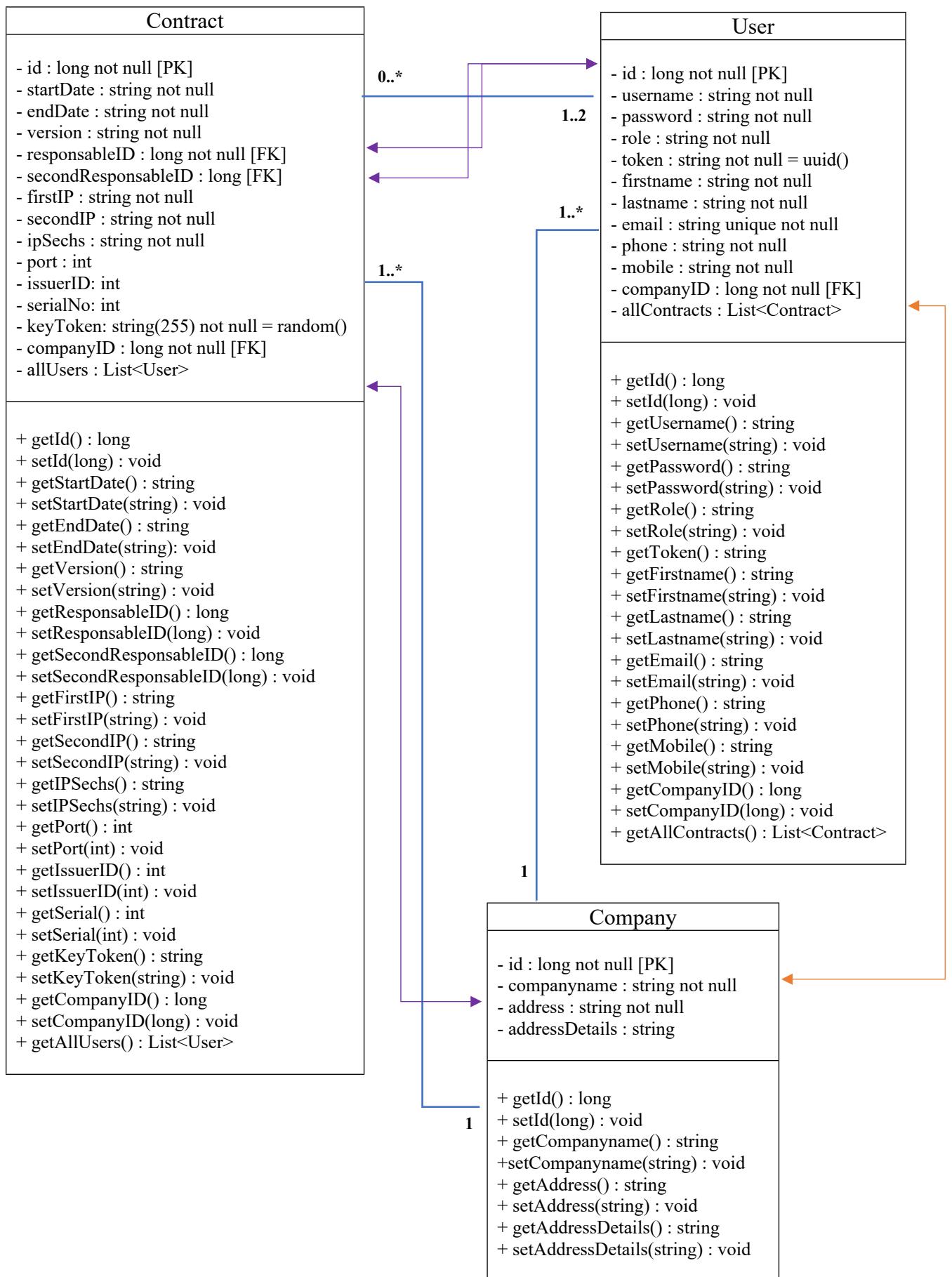


Abbildung 1: UML Klassendiagramm

- (-) **private** : Nur Operationen der Klasse haben Zugriff auf ein privates Attribut.
- (+) **public** : Es erlaubt den Zugriff auf Objekte aller anderen Klassen
- [PK]** : Primary Key (Primärschlüssel). Es dient zur eindeutigen Identifizierung einer Entität in einer Tabelle.
- [FK]** : Foreign Key (Fremdschlüssel). Es bezieht sich auf den Primärschlüssel einer anderen Entität in einer anderen Tabelle.
- (0..*) Assoziation** : Es gibt entweder keine Beziehung oder viele Beziehungen zwischen Entitäten.
- (1..*) Assoziation** : Es gibt mindestens eine Beziehung zwischen Entitäten.
- (1..2) Assoziation** : Es gibt entweder eine Beziehung oder zwei Beziehungen zwischen Entitäten.

2.4 UML State Maschine Diagramm

Das dynamische Verhalten eines einzelnen Objekts können anhand UML State Maschine Diagramm modelliert werden. Diese Darstellung hilft uns über den Arbeitslauf in dem System einen Überblick zu verschaffen, somit die Entwicklung der Benutzeroberfläche konkretisiert werden kann.

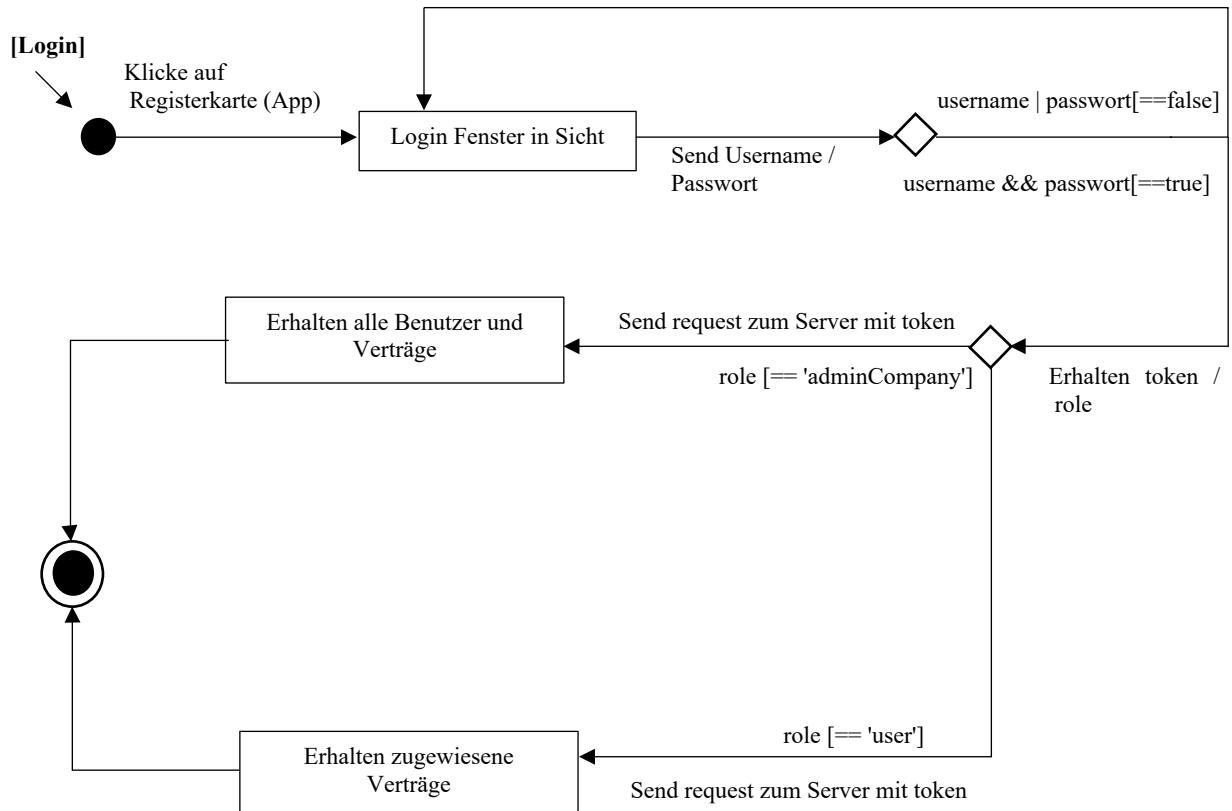


Abbildung 2: UML State Maschine Diagramm für Login

[Neuer Benutzer Hinzufügen]

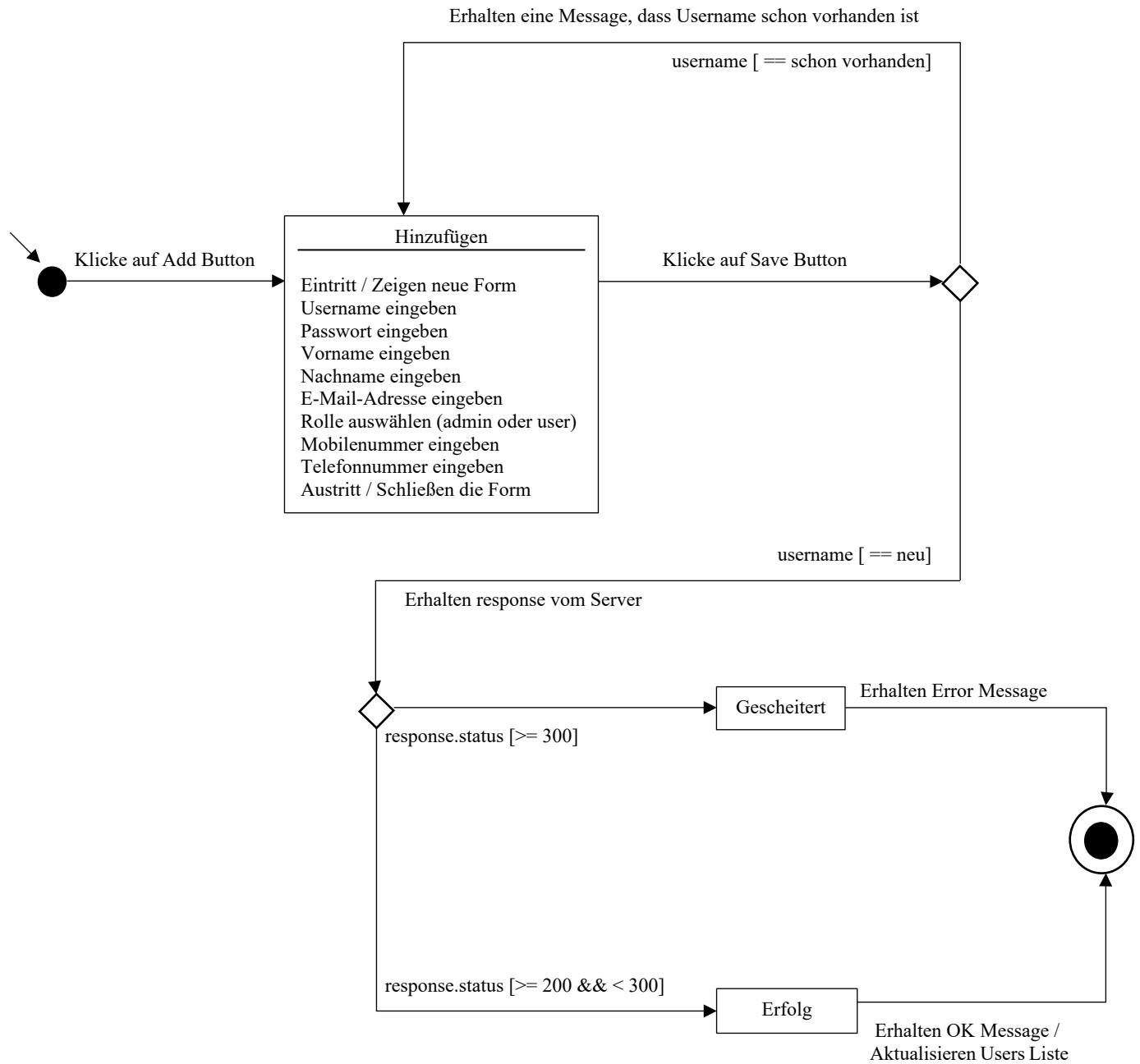


Abbildung 3: UML State Maschine Diagramm für neuen Benutzer hinzufügen

[Neuer Vertrag Hinzufügen]

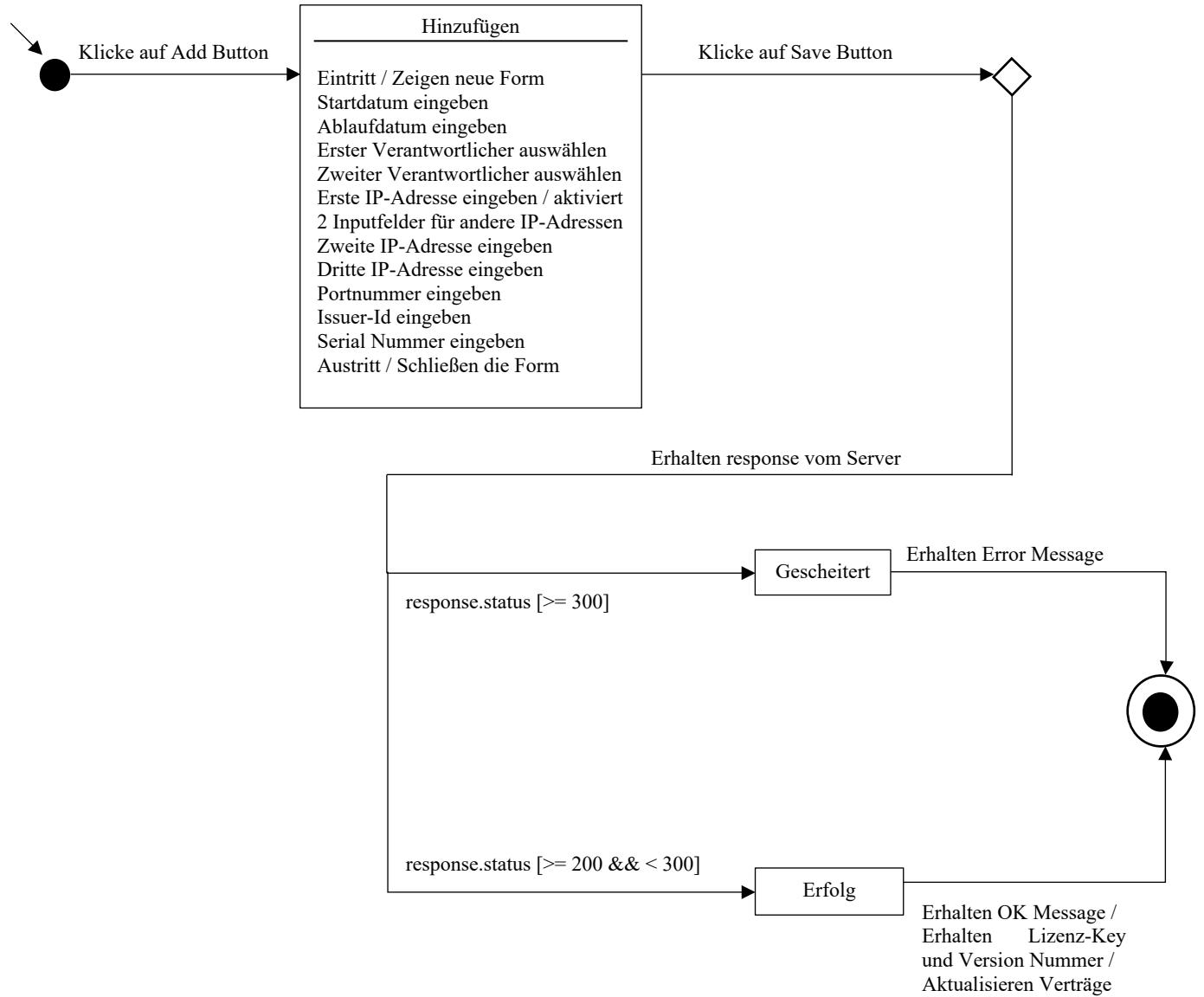


Abbildung 4: UML State Maschine Diagramm für neuen Vertrag hinzufügen

2.5 Three-Tier (Dreischichtige) Architektur

Die dreischichtige Architektur ist eine etablierte Softwareanwendungsarchitektur, die Anwendungen in drei logischen und physischen Datenverarbeitungsschichten organisiert:

- **Darstellungsschicht (Web-Server):** Die Darstellungsschicht ist die Benutzerschnittstelle und die Kommunikationsschicht der Anwendung, in der der Endbenutzer mit der Anwendung interagiert. Diese Schicht kann beispielsweise in einem Webbrowser, als Desktopanwendung oder als grafische Benutzerschnittstelle ausgeführt werden. In diesem Projekt wird diese Schicht durch React.js entwickelt und auf MS Teams als Registerkarte integriert.
- **Anwendungsschicht (Anwendungsserver):** Diese Schicht, die auch als logische Schicht oder Mittelschicht bezeichnet wird, ist das Herz der Anwendung. In dieser Schicht werden die in der Darstellungsschicht erfassten Informationen verarbeitet. Die Anwendungsschicht kann auch Daten in der Datenschicht hinzufügen, löschen oder ändern. In diesem Projekt wird diese Schicht durch [JPA ORM](#) und [JAX-RS](#) entwickelt und anhand Quarkus Gradle konfiguriert.
- **Datenschicht (Datenbankserver):** In der Datenschicht, die manchmal als Datenbankschicht, Datenzugriffsebene oder Back-End bezeichnet wird, werden die von der Anwendung verarbeiteten Informationen gespeichert und verwaltet. In diesem Projekt wird diese Schicht durch MySQL [RDBMS](#) entwickelt.

Der Hauptvorteil dieser Architektur besteht darin, dass jede Schicht auf ihrer eigenen Infrastruktur ausgeführt wird.

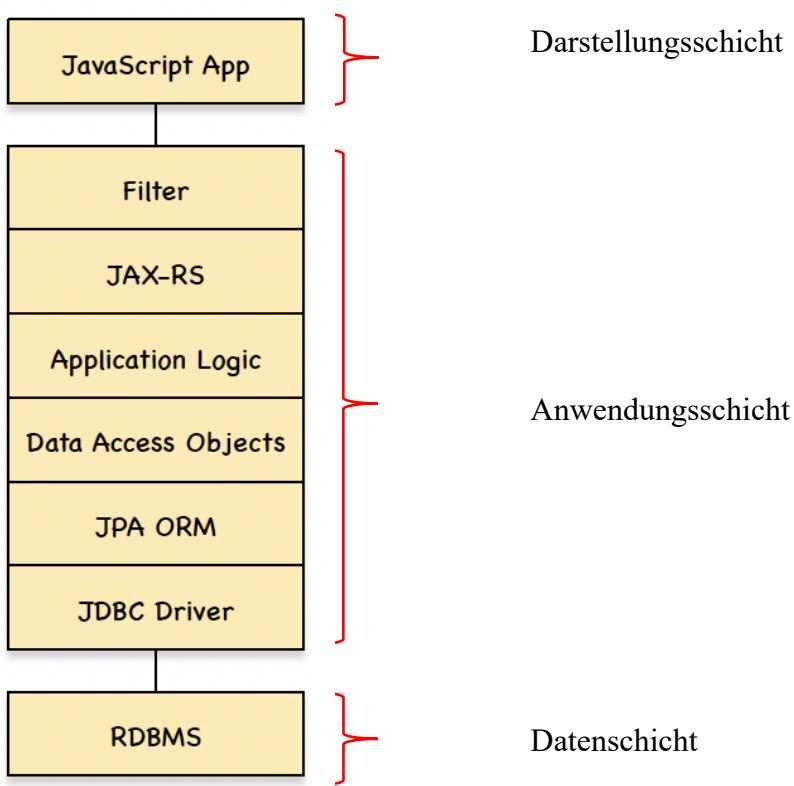


Abbildung 5: Three-Tier Architektur

2.6 RDBMS

Mit einem Relational Database Management System ([RDBMS](#)) lassen sich relationale Datenbanken erstellen, pflegen und administrieren. Daten sind in strukturierten Tabellen abgelegt und stehen in eindeutigen Beziehungen zueinander.

Eine Tabelle, die auch Relation genannt wird, besteht aus Columns, die auch Attributes genannt werden, und Zeilen, die auch Tupels oder Entitys genannt werden. Innerhalb einer relationalen Datenbank muss jede Entität eindeutig zu identifizieren sein. Dies erfolgt über so genannte Primärschlüssel. Alle 3 Entitäten in dieser Projektarbeit haben einen eindeutigen Primärschlüssel (PK) "id" mit dem Typ *long*.

Innerhalb einer relationalen Datenbank können Verknüpfungen mithilfe eines Fremdschlüssels eingereicht werden. Aus Abbildung-1 kann ersehen werden, dass 3 Entitäten Verbindungen miteinander verknüpfen.

Es gibt unterschiedliche Arten von Constraints (Beschränkungen) innerhalb einer relationalen Datenbank. Der Zweck von Constraints besteht darin, die Datenintegrität zu bewahren, wenn Daten bearbeitet, aus der Tabelle entfernt oder neue Daten in die Tabelle gespeichert werden. Nachfolgend sind einige der am häufigsten verwendeten Constraints:

- **NOT NULL** : Es stellt sicher, dass ein Attribut keinen NULL-Wert haben kann. In dieser Projektarbeit wird es verwendet.
- **DEFAULT** : Es stellt einen Standardwert für ein Attribut bereit, wenn keine angegeben ist.
- **UNIQUE** : Es stellt sicher, dass alle Werte in einer Column unterschiedlich sind. In dieser Projektarbeit wird es verwendet.
- **PRIMARY KEY** : In dieser Projektarbeit wird es verwendet.
- **FOREIGN KEY** : In dieser Projektarbeit wird es verwendet.
- **CHECK** : Es gewährleistet, dass alle Werte in einer Column bestimmte Bedingungen erfüllen.

2.7 JPA ORM

Anhand der Objektrelationale Abbildung (englisch [ORM](#)) Technik können in einer objektorientierten Programmiersprache, beispielsweise in diesem Projekt verwendete Sprache Java, angelegte Objekte in einer relationalen Datenbank abgespeichert werden, indem Objekte auf die Strukturen der Datenbank abgebildet werden.

Klassen werden auf Tabellen abgebildet, da jedes Objekt entspricht einer Tabellenzeile und für jedes Attribut der Klasse wird eine Tabellenspalte reserviert. Die Identität eines Objekts entspricht dem Primärschlüssel der Tabelle.

[JPA](#) ist ein Standard für Java Anwendungen, um sowohl die Daten zu persistieren als auch [ORM](#) zu realisieren. Um die Objekte einer Klasse in Datenbank zu persistieren, wird `@Entity` Komponente von [JPA](#) verwendet. Jede JPA-Entität muss einen Primärschlüssel haben, der sie eindeutig identifiziert. Diese Identifizierung wird durch `@Id` Annotation realisiert.

```

import javax.persistence.Table;

@Entity
@Table(name = "Tcompany")
public class Company implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @SequenceGenerator(name = "tcompanySeq", sequenceName = "ZSEQ_TCOMPANY_ID",
    allocationSize = 1, initialValue = 10)
    @GeneratedValue(generator = "tcompanySeq")

    @Column(name = "ID")
    private Long id;

    @Basic(optional=false)
    @Column(name = "companynname", length=64)
    private String companynname;

    @Basic(optional=false)
    @Column(name = "address", length=64)
    private String address;

    @Basic(optional=true)
    @Column(name = "addressDetails", length=64)
    private String addressDetails;
}

```

ID	address	addressDetails	companynname
10	Dornhaldestraße 6, 70199 Stuttgart	Baden Württemberg	entrecode GmbH
11	Berlinerstraße 43, 73776 Esslingen a.N.	BW	MyCode GmbH
NULL	NULL	NULL	NULL

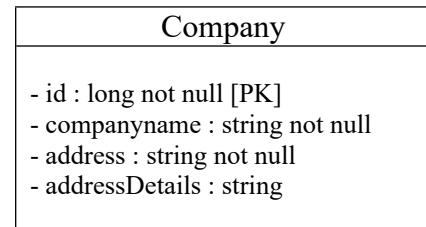
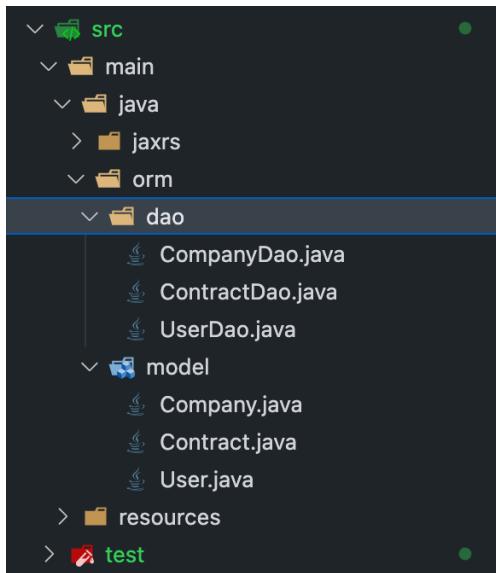


Abbildung 6: Entity Klasse, UML und Tabellenansicht

Um das Projekt zu realisieren, werden 3 Entity Klassen, also 3 Tabellen verwendet. In Datenstruktur befinden sich diese Models in /orm/model Verzeichnis als Entity Java Klassen.

Für jegliche Entity Klasse werden entsprechende [DAO](#) Klasse definiert. Alle [CRUD](#) Operationen, Zugriff auf die Datenbank und [DML](#) Operationen werden von [DAO](#) Klasse durchgeführt. In Datenstruktur befinden sich diese Models in /orm/dao Verzeichnis als [DAO](#) Java Klassen.

```
import orm.model.*;

@ApplicationScoped
public class CompanyDao {

    @Inject
    EntityManager em;
    public Company getCompany(Long id) {
        return em.find(Company.class, id);
    }

    @Transactional
    public List<Company> getCompanies(String token) {
        User template = new User();
        try {
            template = (User) em.createQuery("SELECT u FROM User u WHERE u.token=:token")
                .setParameter("token", token).getSingleResult();
        } catch(NoResultException e) {
            template.setRole("");
        }
        if( template.getRole().equalsIgnoreCase("adminSystem") ) {
            Query q = em.createQuery("SELECT c FROM Company c");
            @SuppressWarnings("unchecked")
            List<Company> companies = q.getResultList();
            return companies;
        } else {
            return new ArrayList<>();
        }
    }
}
```

Abbildung 7: Durchführung einer GET Operation anhand SQL Query, um die Daten aller Unternehmen zu erhalten.

2.8 [REST](#) und [JAX RS](#)

[REST](#) ist ein Architekturstil, der auf WEB-Standards und [HTTP](#) Protokolle basiert. [REST](#) beschreibt, wie verteilte Systeme miteinander kommunizieren können. [REST](#) setzt folgende sechs Architekturprinzipien voraus:

- **Client-Server:** Das client-server design pattern erzwingt die "seperation of concerns ", was dazu hilft, dass die Client- und die Serverkomponenten unabhängig voneinander weiterentwickelt werden zu können. Durch die Trennung der Benutzeroberfläche und Server können Clients leichter auf verschiedene Plattformen portiert werden und die Vereinfachung der Serverkomponenten verbessert die Skalierbarkeit.
- **Zustandslosigkeit:** Client und Server müssen miteinander zustandslos kommunizieren. Es bedeutet, dass jede Anfrage vom Client alle Informationen, die Server benötigt, um die Anfrage zu erfüllen, beinhalten muss. Server kann auf keinen gespeicherten Kontext zurückgreifen.

- **Caching:** Eine Response des Servers sollte implizit oder explizit als cachefähig oder nicht cachefähig gekennzeichnet sein. Falls sie cachefähig ist, können Clients diese Informationen speichern und weiterhin erneut verwenden.
- **Einheitliche Schnittstelle:** Die Komponenten REST-konformer Services nutzen eine einheitliche, allgemeine und von implementiertem Dienst entkoppelte Schnittstelle.
- **Layered System:** In diesem Stil wird das Verhalten der Komponenten eingeschränkt. Beispielsweise jede Komponente kann nicht über die unmittelbare Ebene, mit der sie interagiert, hinaussehen.
- **Code on Demand(optional):** REST ermöglicht auch die Erweiterung der Client-Funktionalität durch Herunterladen und Ausführen von Code in Form von Applets oder Skripten

In einer REST-basierten Architektur befindet sich einer REST-Server, der den Zugriff auf Ressourcen bereitstellt und einer REST-Client, der auf die Ressourcen zugreift und sie modifiziert. Zugriff auf Ressourcen wird von HTTP-Standartmethoden verwirklicht.

HTTP Standartmethoden:

- **GET:** Die angegebenen Ressourcen vom Server gefordert. GET weist keine Nebeneffekte auf. Der Zustand am Server wird nicht verändert.
- **POST:** Eine neue Ressource wird angelegt.
- **PUT:** Entweder eine vorhandene Ressource wird bearbeitet oder eine neue Ressource wird angelegt.
- **DELETE:** Eine vorhandene Ressource wird gelöscht.

JAX-RS, die die Spezifikation von Java ist, wird verwendet, um die Verwendung des REST-Architekturstils zu ermöglichen. JAX-RS benutzt Annotationen, um die REST-Relevanz so wie gelieferte und konsumierte MIME-Types, URLs, HTTP-Methoden von Java Klassen zu definieren.

```
@PUT
@Path("updateuser")
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.APPLICATION_JSON)
public String updateUser(User user) {
    String result = userDao.save(user);
    if(result.equalsIgnoreCase("Persistence Exception")) {
        throw new WebApplicationException(Response.Status.INTERNAL_SERVER_ERROR);
    } else if(result.equalsIgnoreCase("Forbidden")) {
        throw new WebApplicationException(Response.Status.FORBIDDEN);
    }
    else {
        throw new WebApplicationException(Response.Status.OK);
    }
}
```

Abbildung 8: JAX-RS Klasse

Annotationen der Abbildung 8:

- **@Path("updatuser")**: Es setzt den Pfad auf base [URL](#) + /updateuser
- **@POST**: Es definiert, dass eine [HTTP](#) PUT Request auf oben definierten PATH von der folgenden Methode updateUser beantwortet wird.
- **@Consumes (MediaType.APPLICATION_JSON)**: Es definiert, dass als [MIME](#)-Type beliebter Typ [JSON](#) von der Methode konsumiert wird.
- **@Produces (MediaType.APPLICATION_JSON)**: Es definiert, dass als [MIME](#)-Type [JSON](#) zum Client geliefert wird.

2.9 JUnit Test

Bei Entwicklung oder Änderung einer Software sollte es sichergestellt werden, dass die Software in allen möglichen Situationen erfolgreich funktioniert. Durch verschiedene Tests kann Funktionen, Benutzerinteraktion sowie Datenübertragung und -bearbeitung in den implementierten Modulen getestet werden. Somit wird die Qualitätssicherung des Codes sichergestellt. Der Hauptzweck des Testens ist es, Fehler bei Implementierung zu finden. Aber es bedeutet nicht, dass die Software fehlerfrei ist, falls bei einem Test kein Fehler aufgetreten ist.

JUnit ist ein verbreitetes Framework zum Testen von Java-Programmen, das besonders für automatisierte Unit-Tests einzelner Units (Klassen oder Methoden) geeignet ist. Ein Test kann entweder erfolgreich, diese Situation mit grüner Farbe dargestellt wird, oder nicht erfolgreich, diese Situation mit roter Farbe dargestellt wird.

In dieser Projektarbeit wird Junit 5 Framework verwendet. In build.gradle muss dependency(Abhängigkeit) von Junit festgestellt werden. (Siehe Abbildung 9) Außerdem kann ein Visual Studio Code Erweiterung verwendet werden, um Tests durchführen zu können. (Siehe Abbildung 10)

```
dependencies {  
    implementation 'io.quarkus:quarkus-resteasy-jsonb'  
    implementation 'io.quarkus:quarkus-jdbc-mysql'  
    implementation 'io.quarkus:quarkus-hibernate-orm'  
    implementation enforcedPlatform("${quarkusPlatformGroupId} : ${quarkusPlatformArtifactId} : test")  
    implementation 'io.quarkus:quarkus-resteasy'  
  
    testImplementation 'io.quarkus:quarkus-junit5'  
    testImplementation 'io.rest-assured:rest-assured'  
}
```

Abbildung 9: build.gradle Konfiguration für Junit

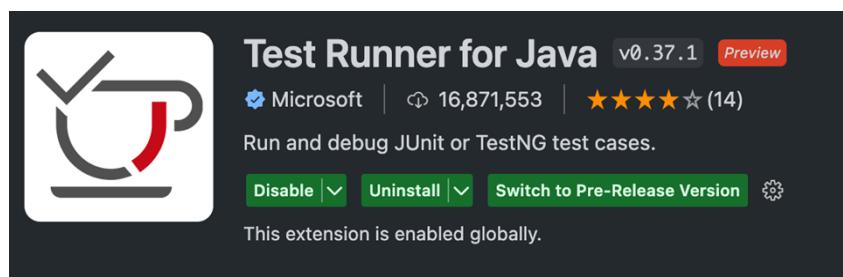


Abbildung 10: Erweiterung für Durchführung der Tests

In dieser Projektarbeit wird folgende Methoden aus DAO Klassen mit 100 % Erfolg Prozent getestet (Siehe Abbildung 11):

1-) UserDao:

- checkCreateDeleteUser() :
 - Anlegen eines neuen Users
 - Versuchen einen neuen User mit gleichem Username, der sich schon in Database befindet, anzulegen und Fehlermeldung zu bekommen
 - Entfernen eines Users
- checkUpdateUser() :
 - Bearbeiten einiger Angaben sowie Username und Nachname eines Users
- checkLogin():
 - Vergleichen Username und Password eines Users mit der Daten in Database und Testen der Funktionalität von Login

2-) CompanyDao:

- checkCreateUpdateDeleteCompany() :
 - Anlegen eines neuen Unternehmens
 - Bearbeiten einer Angabe sowie der Name des Unternehmens
 - Versuchen ein neues Unternehmen mit gleichem Namen und gleicher Adresse, die sich schon in Database befinden, anzulegen und Fehlermeldung zu bekommen
 - Entfernen eines Unternehmens

3-) ContractDao:

- checkCreateDeleteContract() :
 - Anlegen eines neuen Vertrags
 - Entfernen eines Vertrags
- checkUpdateContract() :
 - Bearbeiten einiger Angaben sowie Ablaufdatum und IP-Adresse eines Vertrags
- checkUpdateKey() :
 - Austauschen eines Token Keys eines Vertrags mit einem neuen Token Key

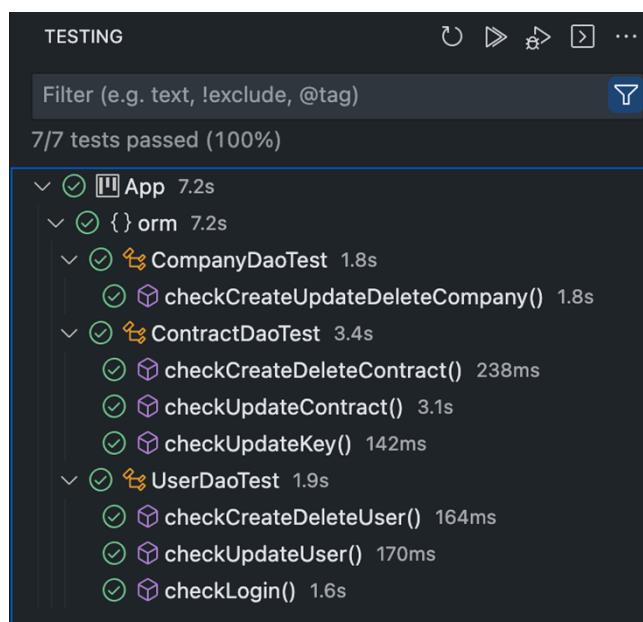


Abbildung 11: Testergebnis

3 Realisierung

3.1 Software-Setup

Vor der Realisierung der App müssen einige allgemeine Vorbereitungen und Konfigurationen getroffen werden. Alle diese Schritte werden in den folgenden Abschnitten gezeigt und erklärt.

3.1.1 MySQL RDBMS

Um die Daten, die der App gehören, erfolgreich und leicht verwalten zu können, wird eine open-source Software [MySQL Community Server App](#) entsprechend dem Betriebssystem des Users installiert.

Um die Datenbank leicht zu verwalten, anstelle Command-Tools des Betriebssystems wird eine open-source Benutzeroberfläche [MySQL Workbench](#) installiert.

Das Password des Users 'root' muss vermerkt werden. Dieses Password kann später anhand folgenden Kommandos ausgetauscht werden.

```
mysql> SET PASSWORD FOR 'root'@'localhost' = 'new_password';
```

Nach der Installation muss eine Verbindung zum Database anhand MySQL Workbench angelegt werden (Siehe Abbildung 12). Falls eine erfolgreiche Verbindung zum Database angelegt wird, dann muss eine instance, die Tabellen enthalten wird, angelegt werden. (Siehe Abbildung 13)

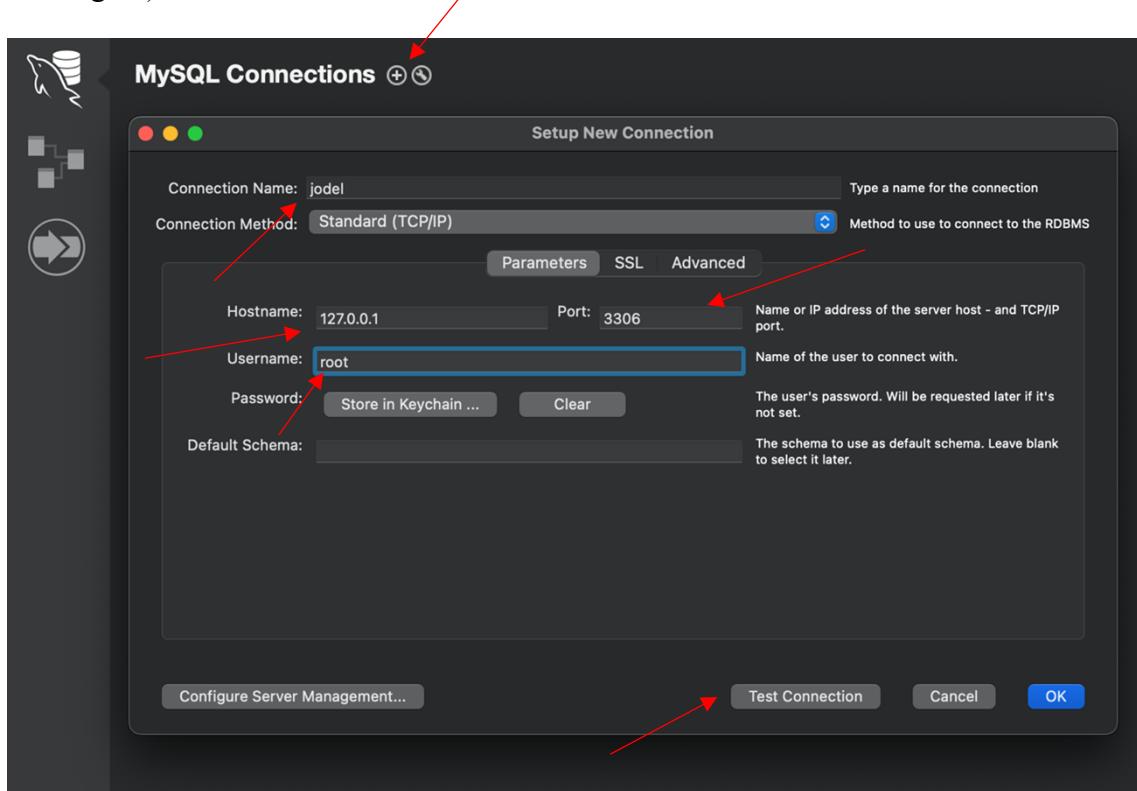


Abbildung 12: Anlegen einer Verbindung zum Database

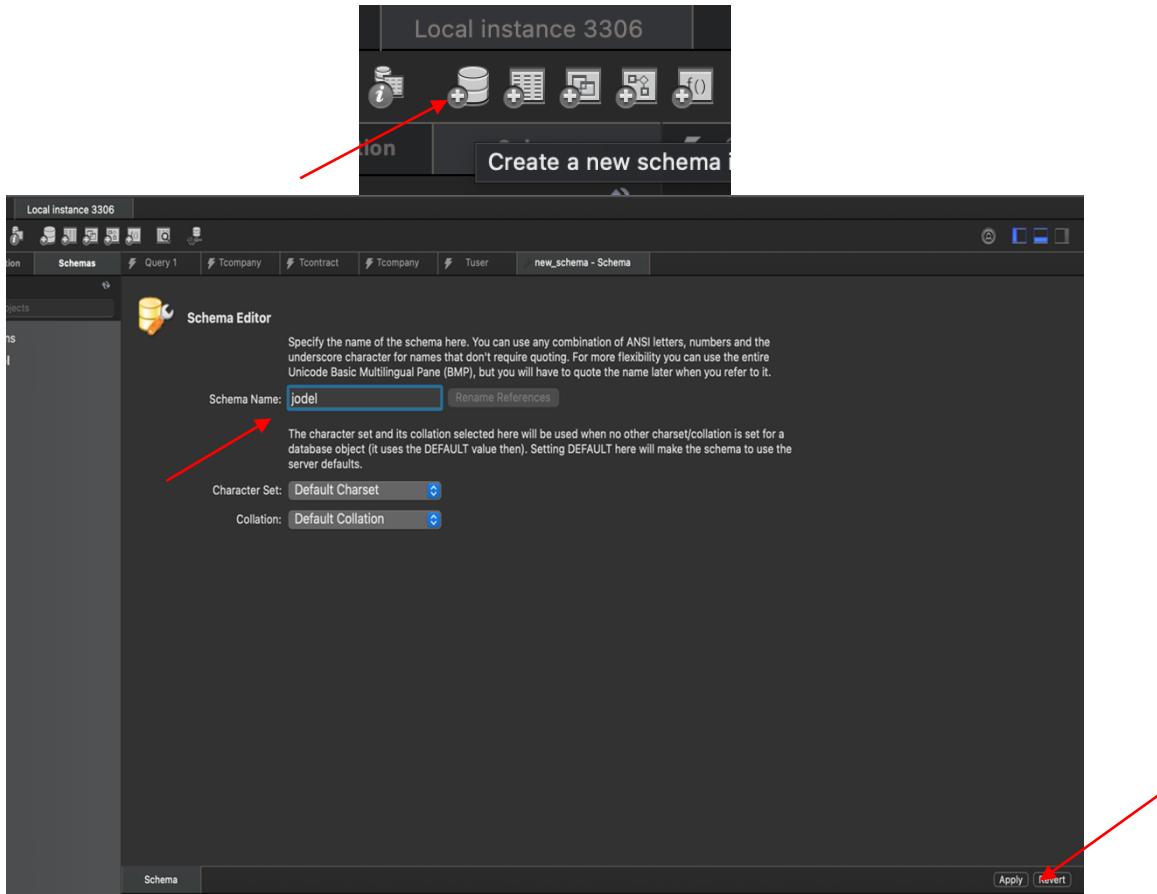


Abbildung 13: Anlegen einer instance

Anhand der Datei, die import.sql heißt und sich unter dem Pfad /src/main/resources befindet, kann ein initialer User in der Users Tabelle beim Build angelegt werden. (Siehe Abbildung 14) Man kann auch anhand MySQL Workbench diese Aktion durchführen.

```
-- Some initial data
INSERT INTO TUSER (id, role, token, firstname, lastname, email, phone, mobile, companyID)
VALUES ("1", "adminSystem", "19645f04-6197-11ed-861e-3fdb8d325102",
        "Ertugrul", "Sevgili", "erseit01@hs-esslingen.de",
        "+491782952014", "+4915256939090", "0");
```

Abbildung 14: Initial Data für Users Tabelle Hinzufügen

3.1.2 Quarkus - Gradle

Um Build Management zu verwalten, werden [gradle](#) und [quarkus](#) framework installiert. Es gibt verschiedene Installationsmöglichkeiten. Man kann entweder manuell oder anhand Command-Tools des Betriebssystems diese Softwares installieren.

Alle erforderliche build Informationen sowie Abhängigkeiten, Libraries, Java Versions, usw. befinden sich in der File *build.gradle*, die sich in root Ordner des Projekts befinden soll.

Nach der Installation von quarkus und gradle muss eine Datei *application.properties* (Siehe Abbildung 15) angelegt werden, um die Konfiguration des Projekts festzulegen. In dieser Datei werden alle Konfigurationsangaben des Projekts zusammengesetzt. Die Datei soll sich unter /src/main/resources befinden.

```
# Configuration file
quarkus.banner.path=banner-swa
# Step 3 -----
quarkus.datasource.db-kind = mysql
quarkus.datasource.username = root
quarkus.datasource.password = password
quarkus.datasource.jdbc.url = jdbc:mysql://localhost:3306/jodel?serverTimezone=UTC

# drop and create the database at startup (use `update` to only update the schema
# or drop-and-create )
quarkus.hibernate-orm.database.generation=drop-and-create|  
# Schema creation not supported for MySQL
# quarkus.hibernate-orm.database.generation.create-schemas=true

quarkus.hibernate-orm.log.sql=false

# quarkus.log.level=DEBUG
# quarkus.log.category."org.hibernate".level=DEBUG
%dev.quarkus.http.port=8811
quarkus.http.cors=true
quarkus.http.cors.origins=https://localhost:53000
quarkus.http.cors.headers=accept, origin, authorization, content-type, x-requested-with
quarkus.http.cors.methods=GET,PUT,POST,DELETE,OPTIONS
```

Abbildung 15: Konfigurationsdatei – *application.properties*

Folgende Konfigurationen müssen angepasst werden:

- quarkus.datasource.username:** Username für Database, also root.
quarkus.datasource.password: Das Passwort des Users root, das bei der Installation von MySQL Community Server vermerkt wurde.
quarkus.datasource.jdbc.url: Der Pfad des Databases. Hostname (localhost), portnummer (3306), instance (jodel) werden in Abbildung 8 und 9 definiert.
- quarkus.hibernate-orm.database-generation :** Beim ersten Start muss es als *drop-and-create* definiert werden. Das entfernt alle Tabellen aus instance, falls sie vorhanden sind, und erzeugt diese Tabellen wieder beim Build. Diese Konfiguration muss beim nächsten Start als *update* definiert werden, um vorhandene Daten nicht zu verlieren.
- %dev.quarkus.http.port :** Die Portnummer des Servers. Default ist 8080. In diesem Projekt wird es 8811 festgelegt.
- quarkus.http.cors.origins :** URL von Client. Wegen [CORS](#) Policy von Browser kann nicht Datentransfer zwischen Client und Server durchgeführt werden, wenn beide Seite auf der gleichen Resource gestartet werden. Deswegen muss diese Konfiguration angepasst werden, um [CORS](#) Policy für

definierte [URL](#) deaktiviert zu werden. In Debugger Mode Microsoft Teams verwendet <https://localhost:53000>, um den Client zu starten. Diese Konfiguration kann in `/tabs/.env.teams.fx` File geändert werden.

3.1.3 Java Development Kit (JDK)

Das Java Development Kit ist ein Softwarepaket, die bei der Entwicklung in der Programmiersprache Java benötigt wird. In diesem Projekt wird Java Version 11 verwendet. In der Date `build.gradle` steht die benutzte Version von java (Siehe Abbildung 16)

```
compileJava {  
    options.encoding = 'UTF-8'  
    options.compilerArgs << '-parameters'  
}  
  
compileTestJava {  
    options.encoding = 'UTF-8'  
}  
  
java {  
    sourceCompatibility = JavaVersion.VERSION_11  
    targetCompatibility = JavaVersion.VERSION_11  
}
```

Abbildung 16: Version Nummer von java in `build.gradle`

3.1.4 MS Teams Template Projekt mit React.js

Microsoft Teams bietet eine Sammlung von Apps, die von Microsoft oder externen Diensten bereitgestellt werden. Bei Microsoft Teams-Apps kann es sich um Registerkarten, Bots oder Nachrichtenerweiterungen oder eine beliebige Kombination dieser drei handeln. Wenn man bereits eine Web-App, eine SharePoint-Site, eine PowerApp oder eine andere webbasierte Anwendung besitzt, kann es sein, einige oder alle Funktionen dieser App in Microsoft Teams zu aktivieren. In diesem Projekt wird das Laborprojekt des Moduls Softwarearchitektur als Registerkarte in Microsoft Teams App integriert.

Um ein neues Projekt mit dem Teams-Toolkit einzurichten, werden folgende Technologien benötigt:

3.1.4.1 Visual Studio Code

[Visual Studio Code](#) ist eine universelle Entwicklungsumgebung, die für verschiedene Programmiersprachen verwendet werden kann.

3.1.4.2 Teams Toolkit

In Visual Studio Code unter Erweiterungen kann man unterschiedliche Erweiterungen kostenlos installieren. Das Teams Toolkit vereinfacht den Entwicklungsprozess mit Tools zum Bereitstellen.

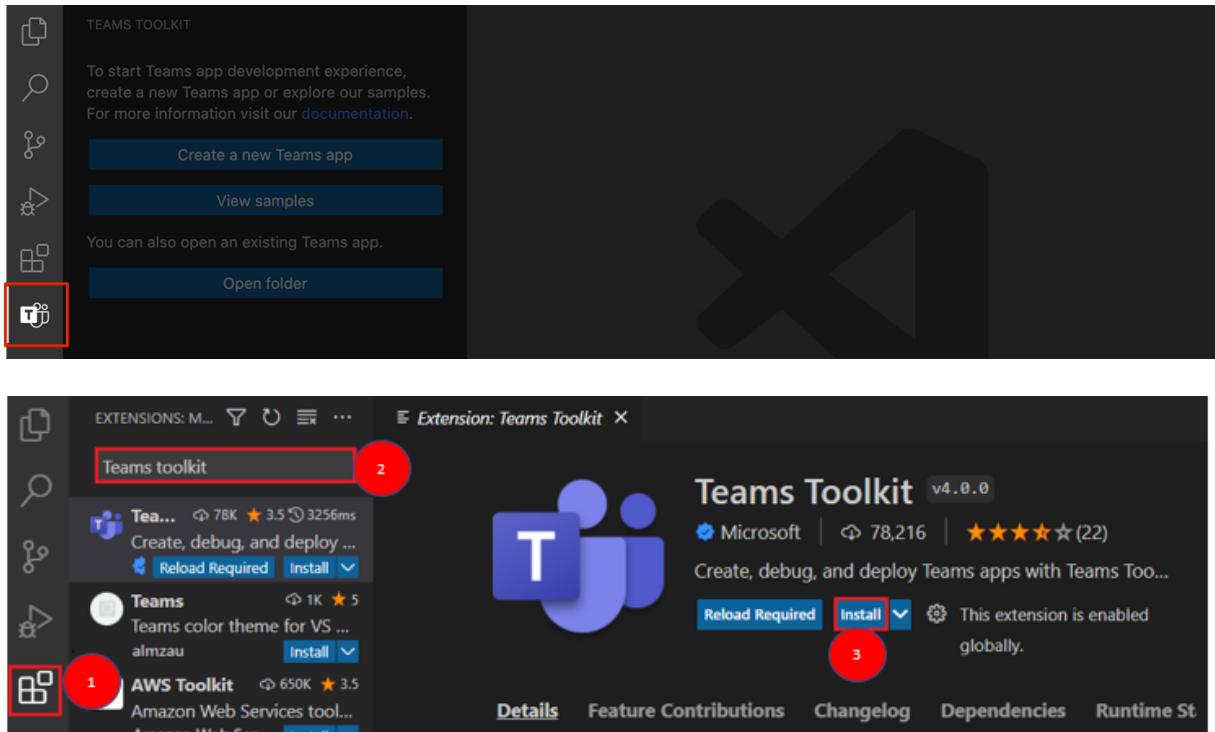


Abbildung 17: Teams Toolkit

3.1.4.3 Erstellen eines Teams-Entwicklermandanten

Nachdem man die App erstellt hat, muss die App für die Vorschau und das Testen in Teams hochgeladen werden, ohne sie zu veröffentlichen. Dafür wird über ein Teams-Entwicklerkonto verfügt werden. Ein Konto sollte über [Microsoft 365-Entwicklerprogramm](#) angelegt werden. Nachdem ein Konto angelegt wird, muss man sich mit dem Administratorkonto bei Microsoft-Teams einloggen.

3.1.4.4 Erstellen des Projektarbeitsbereichs

Falls auf Teams Erweiterung in Visual Studio Code geklickt wird, kommt ein Fenster wie in Abbildung 18 in Sicht. Danach wird neue Teams-App erstellen ausgewählt. Anschließend muss es sichergestellt werden, dass **Tab** als Funktion ausgewählt ist.

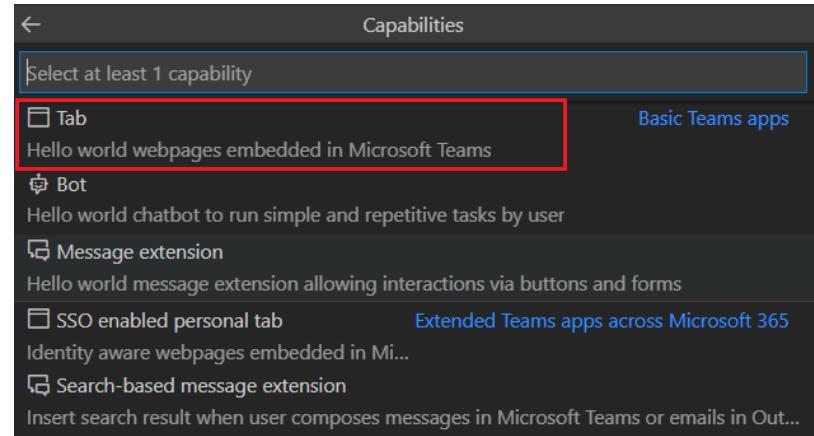
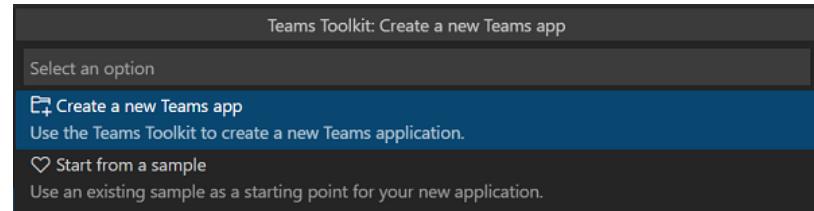
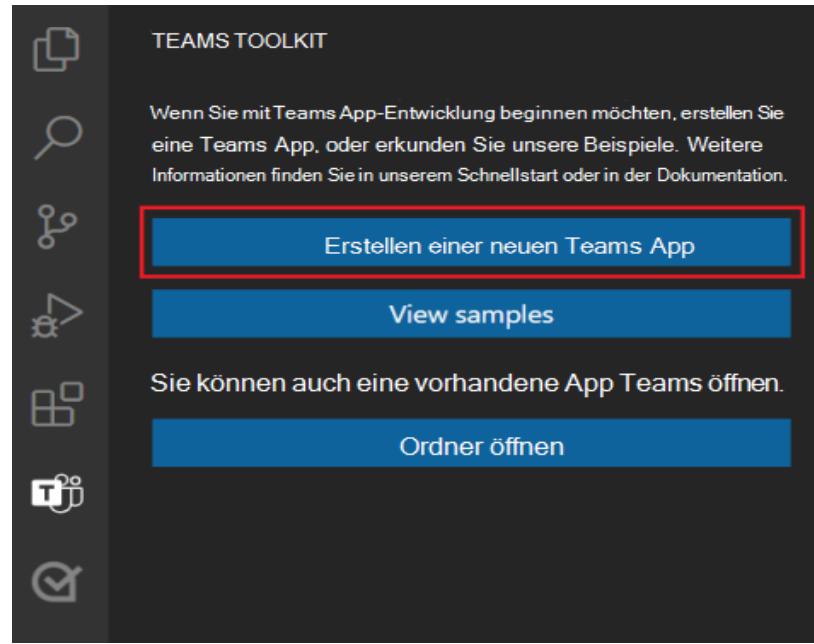


Abbildung 18: Erstellen einer neuen Team-App

Danach sollte die Programmiersprache ausgewählt werden. Javascript wird in diesem Projekt als Sprache ausgewählt. Nach der Bestimmung der Programmiersprache wird der Ordner ausgewählt und der Name des Projekts eingegeben. (siehe Abbildung 19).

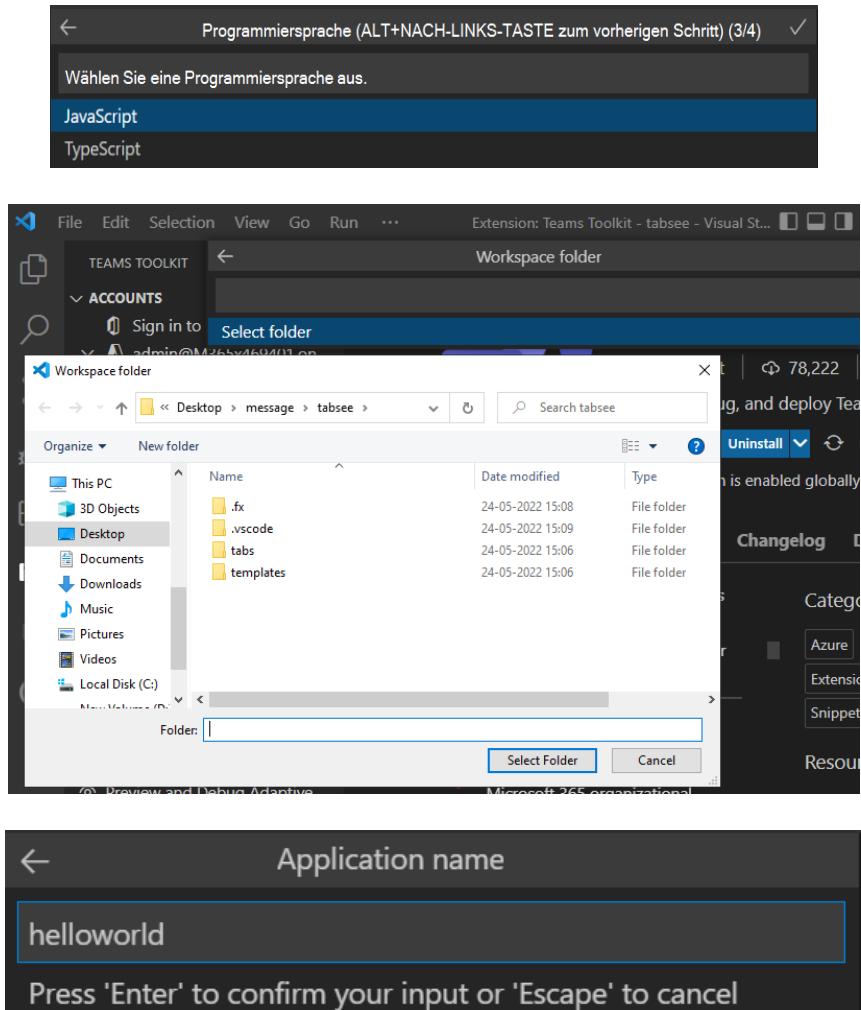


Abbildung 19: Erstellen einer neuen Team-App

3.1.4.5 Ausführung der App

Nachdem der Arbeitsbereich erstellt wird, sollte sich in Visual Studio Code durch Teams Toolkit Erweiterung bei Microsoft-365 Administratorkonto einloggt werden. (Siehe Abbildung 20)

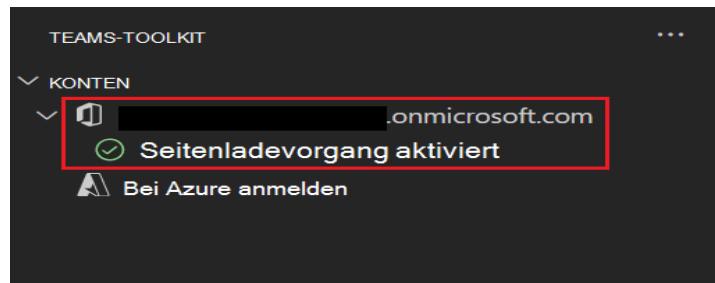


Abbildung 20: Einloggen des Administratorkontos

Um die App im Debugmodus auszuführen, wird danach in Visual Studio Code auf **F5** gedruckt. Danach wird die Template Microsoft Teams App erstellt und der Teams-Webclient wird in einem Browserfenster geöffnet. In diesem Fenster wird Hinzufügen ausgewählt, um die App in Teams hochzuladen. Anschließend wird die Template App in Microsoft Teams ausgeführt (Siehe Abbildung 21)

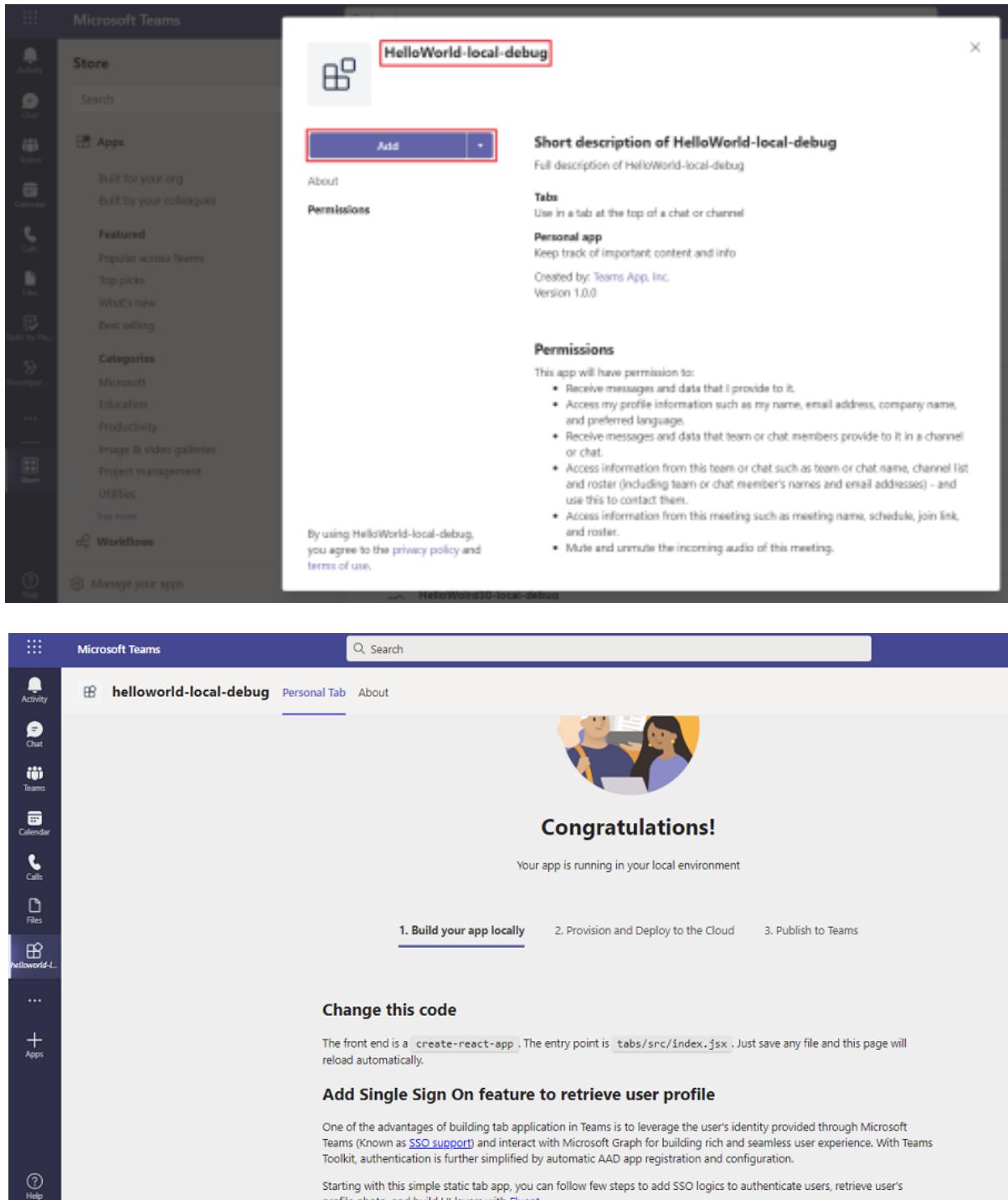


Abbildung 21: Ausführung der lokalen App

3.2 Integrieren der Softwarearchitektur Laboreinheit auf Teams App

Nachdem die Registerkarte also App aufgerufen wird, wird Tab Komponente danach automatisch aufgerufen. In Tab Komponente wird die Laboreinheit der Softwarearchitektur

eingebunden. Alle Dateien des Clients befinden sich in `/tabs` Ordner. Der Ablauf der App kann folgendermaßen dargestellt werden:

```
export default function App() {
  const { theme } = useTeams({} )[0];
  return (
    <Provider theme={theme || teamsTheme} styles={{ backgroundColor: "ghostWhite" }}>
      <Router>
        <Route exact path="/">
          <Redirect to="/tab" />
        </Route>
        <>
          <Route exact path="/tab" component={Tab} />
        </>
      </Router>
    </Provider>
  );
}
```

(App.jsx)



```
return (
  <>
  {
    this.state.loggedIn ? (
      <Login
        url={theUrl}
        authorized={this.authorized}
        companyID={this.setCompanyID}
        role={this.setRole}
        token={this.setToken}
      />
    ) : (
      <>
        {this.state.role == "adminSystem" && (
          <HomePageAdminSystem
            url={theUrl}
            role={this.state.role}
            token={this.state.token}
            authorized={this.authorized}
          />
        )}
        {this.state.role == "adminCompany" && (
          <HomePageAdminCompany
            url={theUrl}
            role={this.state.role}
          />
        )}
      </>
    )
  }
)
```

(Tab.jsx)



```

handleLoginSubmit = (event) => {
  var formdata = JSON.stringify({
    username : this.state.username,
    password: this.state.password
  });
  fetch(this.props.url + "/login", {
    headers: {
      "Content-Type": "application/json",
      Accept: "application/json",
    },
    method: "get",
    body: formdata,
  })
  Complexity is 7 It's time to do something...
  .then(response) => {
    if (response.status > 200 || response.statusText)
      (Event Handler in Login.jsx)

```

Abbildung 22: Der Ablauf des Aufrufs der App

3.3 Environment Variables

In `/tabs/.env` Datei können Konfigurationsangaben des Backendes definiert werden, somit in der App beispielsweise in dieser Projektarbeit keine feste [URL](#) des Backendes verwendet (Siehe Abbildung 23). Vor dem Build kann diese Backend Endpoint geändert werden:

```

HOSTNAME=http://localhost
PORTBACKEND=8811
REACT_APP_URL=${HOSTNAME}:${PORTBACKEND}

```

Konfigurationsdatei .env

```

const theUrl = process.env.REACT_APP_URL;

```

Aufruf der Konfiguration Variable in Tab.jsx

Abbildung 23: Definition und Aufruf der URL des Backendes

3.4 Starten Client und Backend

Um backend zu starten, soll in command lines folgende Kommandos durchgeführt werden:

- Gehen Sie zum root Ordner des Projekts
`cd [path_to_root_folder]`
- In root Ordner
`gradle quarkusDev`

Um Client zu starten, soll in command lines folgende Kommandos durchgeführt werden:

- Gehen Sie zum Client Ordner
`cd [path_to_root_folder]/tabs`
- Falls es der erste Start ist, in Client Ordner wird folgendes Kommando durchgeführt, um die erforderliche Libraries zu installieren. Dieser Schritt kann weiterhin vernachlässigt werden.
`npm install`
- Um die Client zu starten, in Client Ordner wird folgendes Kommando durchgeführt:
`npm run dev:teamsfx`
- Client ist nun erreichbar auch in local auf `https://localhost:53000`

3.5 Stellung der App zur Verfügung für Teams

Zunächst in Visual Studio Code anhand Teams Toolkit wird die App als .zip Datei abgespeichert. Es sollte zuerst auf Teams Toolkit geklickt werden. Danach unter **/Deployment** muss *Zip Teams metadata package* ausgewählt werden. Die .zip Datei wird anschließend automatisch unter */build/appPackage* erstellt. (Siehe Abbildung 24)

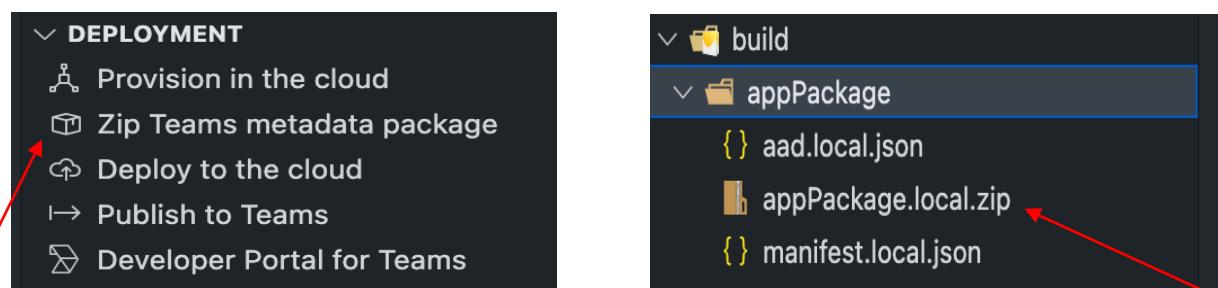


Abbildung 24: Erstellen .zip Datei der App als package

In Microsoft Teams wird zunächst auf Apps unten links geklickt und dann App hochladen wird ausgewählt. (Siehe Abbildung 25)

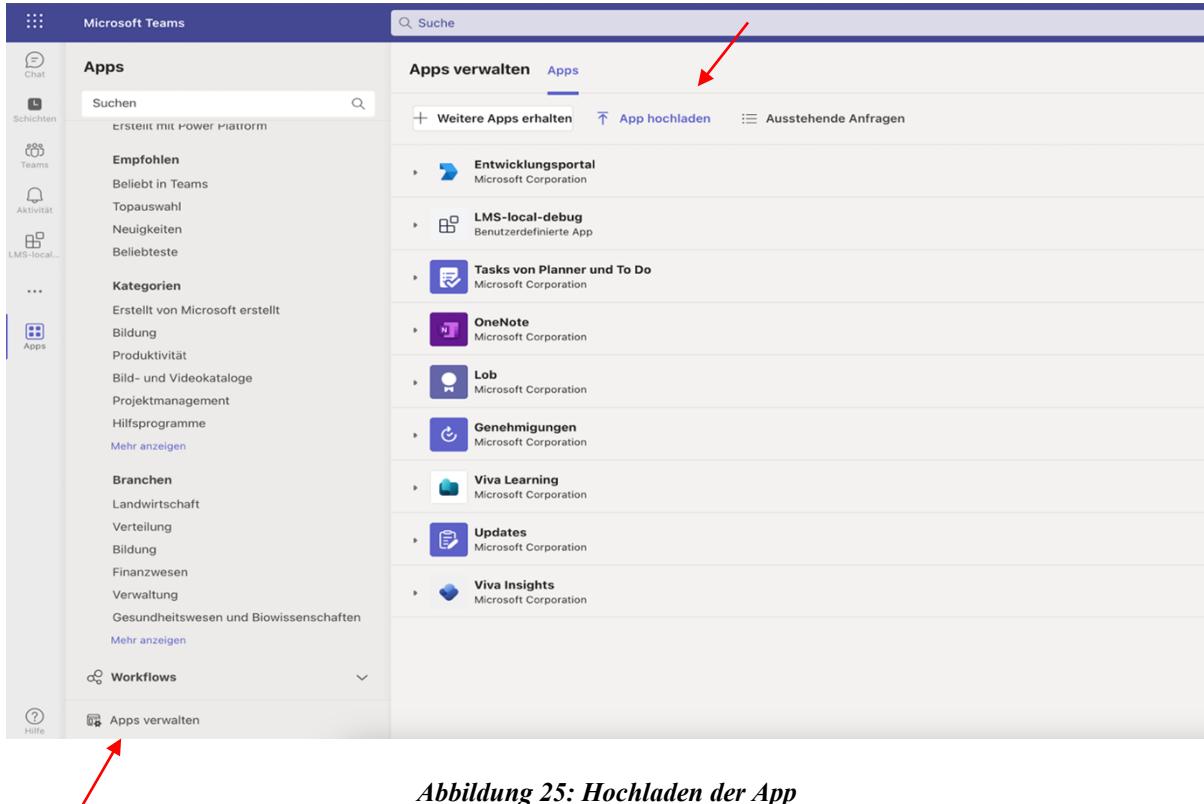


Abbildung 25: Hochladen der App

In erscheinen Fenster wird **Hochladen einer App in den App-Katalog Ihrer Organisation** ausgewählt und die .zip Datei aus lokalem Rechner hochgeladen. (Siehe Abbildung 26)

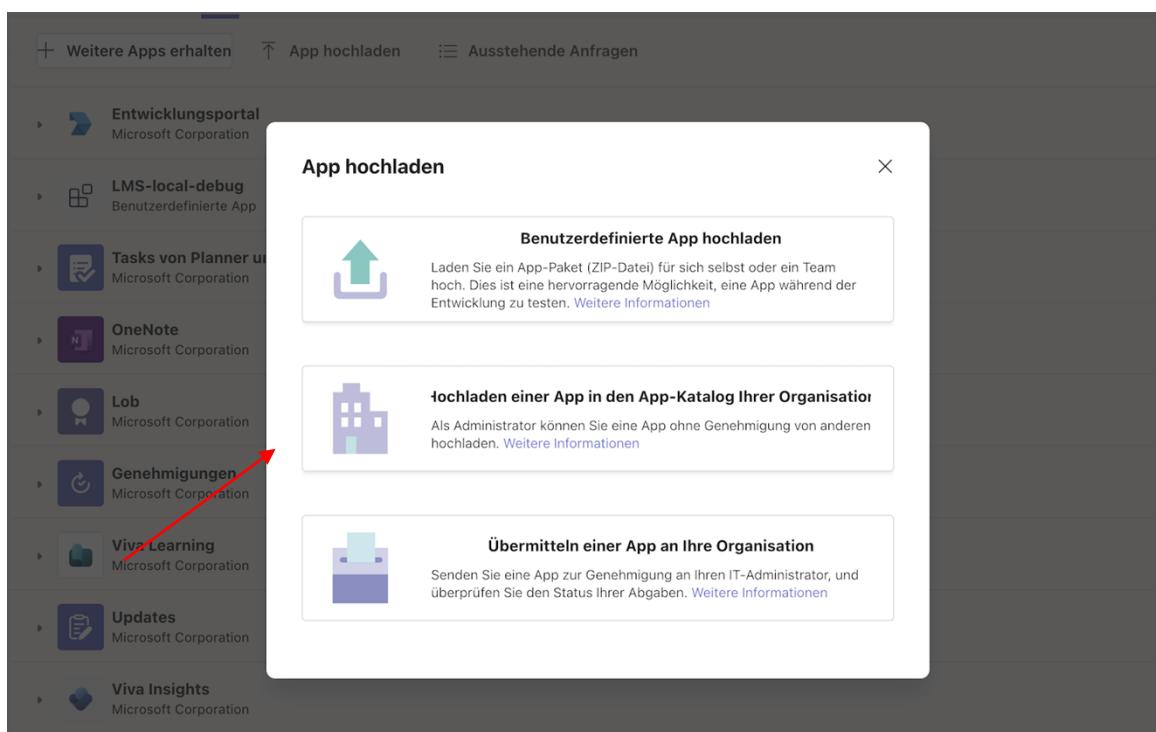


Abbildung 26: Hochladen der .zip Datei

In erscheinenden Fenster wird dann die App zu einem Team, falls sich das Team befindet, hinzugefügt (Siehe Abbildung 27) und anschließend wird auf Registerkarte einrichten geklickt.

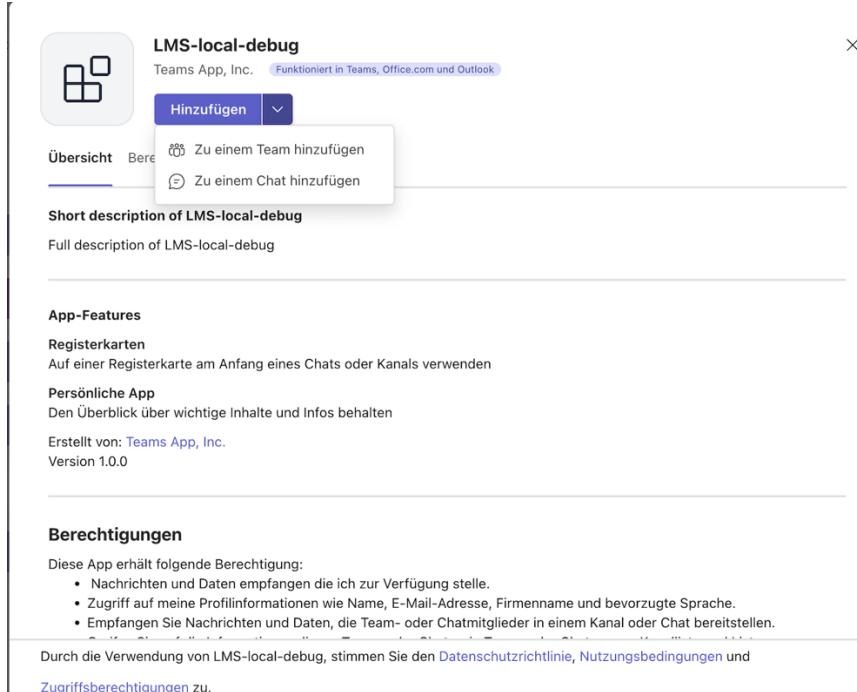


Abbildung 27: Hinzufügen der App zu einem Team

Dann wird es sich als ein Team Mitglied (ertugrulsevgili23) in Microsoft Teams eingeloggt, um die App manuell zu testen. Wenn auf das **Kanalinfo ausblenden Icon** oben rechts geklickt wird, kann die App unter Apps gesehen werden. (Abbildung 28) Anschließend wird auf die App geklickt und dann kommen Login Fenster und dann Homepage in Sicht (Abbildung 29 und 30)

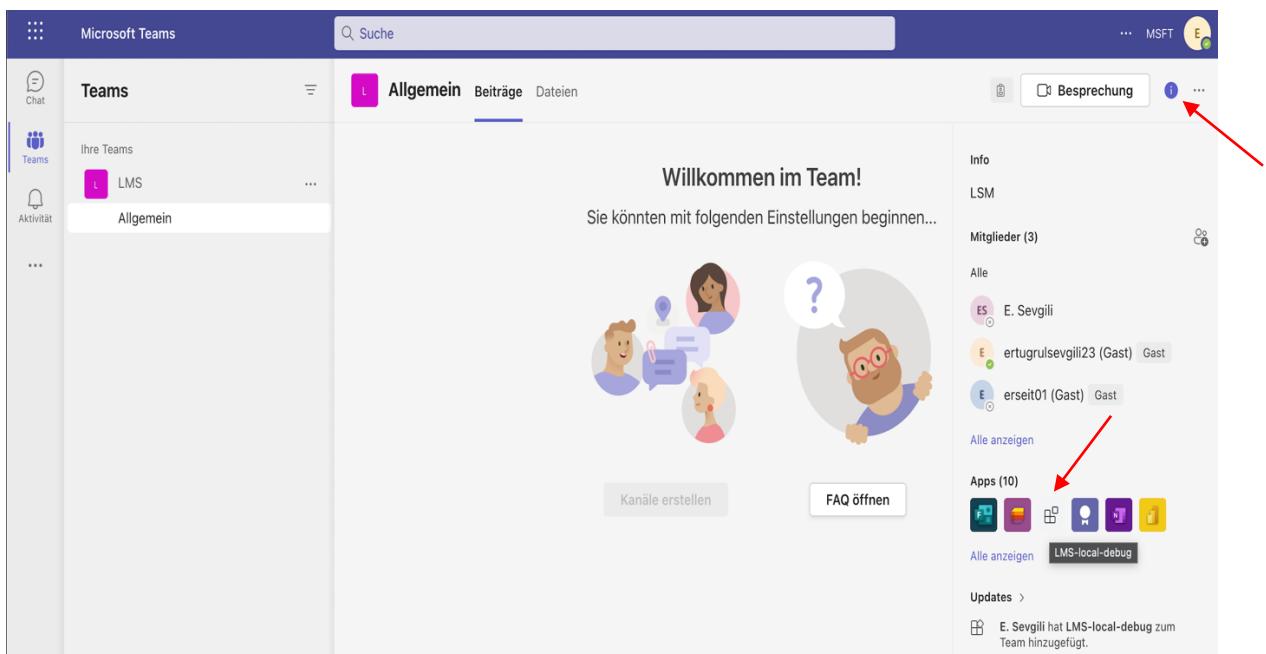


Abbildung 28: Homepage eines Teammitglieds

3.6 Ansichten der App

3.6.1 Login

Falls die App aufgerufen wird, sieht Benutzer das Login Fenster. Falls Username oder Password falsch ist, erhält Benutzer eine Fehlermeldung (Siehe Abbildung 29)

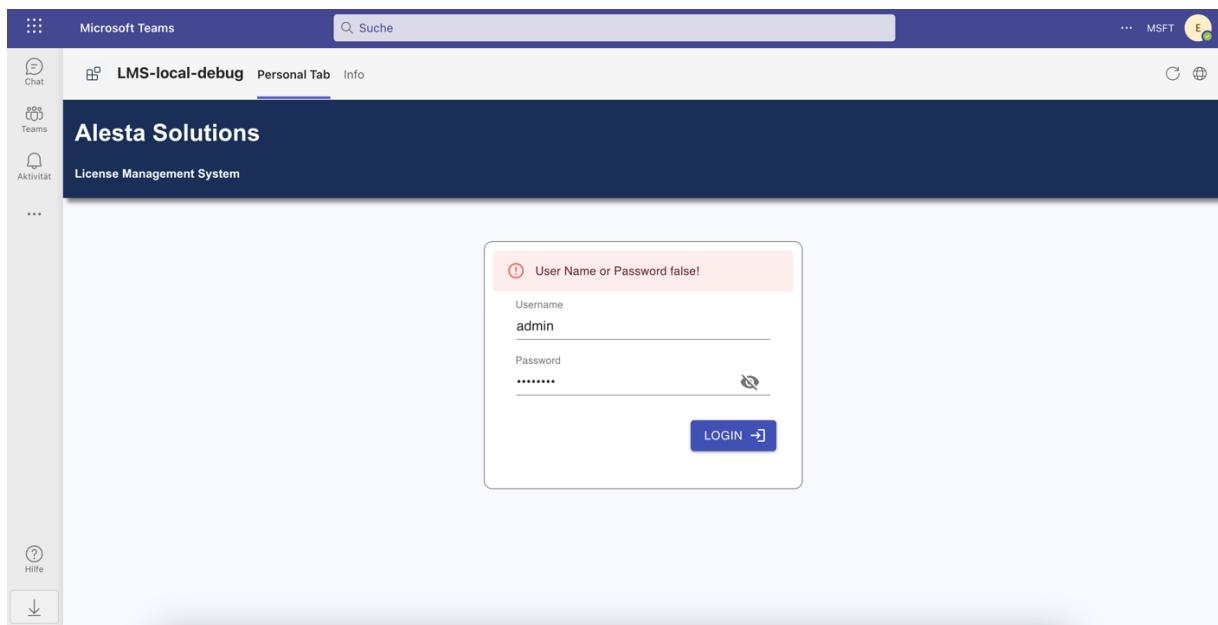


Abbildung 29: Login Page

3.6.2 Homepage

3.6.2.1 Admin System

Nach dem Einloggen sieht ein Benutzer, der "adminSystem" Rolle hat, alle Kunden in einer Liste. (Siehe Abbildung 30)

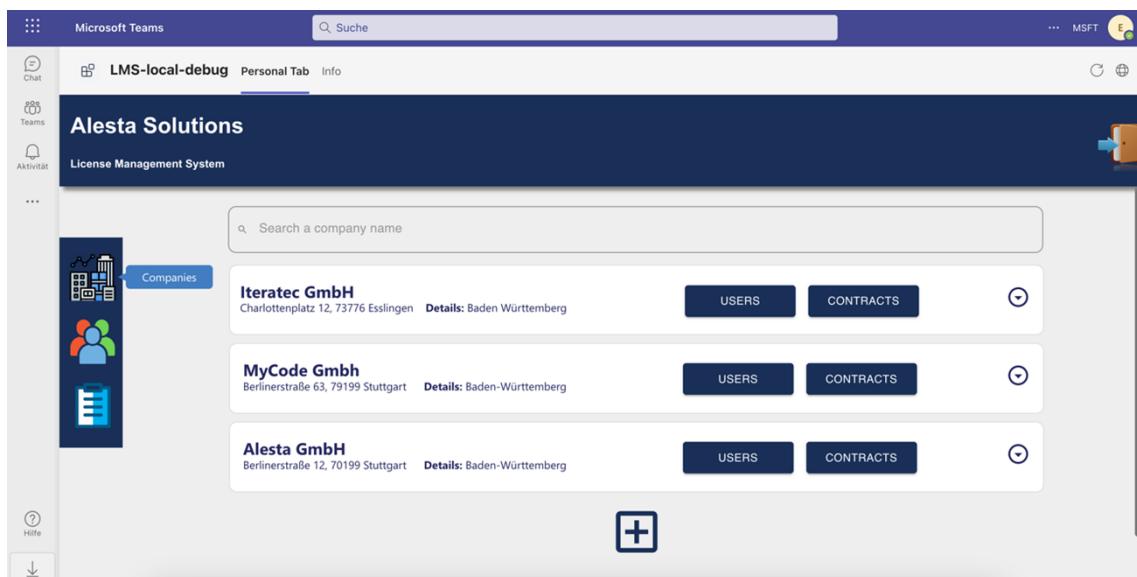


Abbildung 30: Homepage

Falls nach dem Namen des Unternehmens gesucht wird, werden entsprechende Treffe in Sicht geblieben. (Siehe Abbildung 31)

The screenshot shows a Microsoft Teams interface with the 'LMS-local-debug' tab selected. The main area displays the 'Alesta Solutions License Management System'. A search bar at the top contains the letters 'ite'. Below it, a card for 'Iteratec GmbH' is shown, including its address 'Charlottenplatz 12, 73776 Esslingen' and the note 'Details: Baden Würtemberg'. Two buttons, 'USERS' and 'CONTRACTS', are visible. On the left, there's a sidebar with icons for Chat, Teams, and Aktivität, along with a 'Hilfe' link.

Abbildung 31: Suchfunktion

Falls auf "Users" Button geklickt wird, erhält man alle User des ausgewählten Unternehmens in einer Liste. (Siehe Abbildung 32)

The screenshot shows the same Microsoft Teams interface as in Abbildung 31, but now the 'USERS' button has been clicked. The main area displays a table titled 'Iteratec GmbH' showing two users: Ertugrul Sevgili and Zehra Sevgili. The table columns are 'User', 'Role', 'Email', 'Phone', and 'Mobile'. Each user row includes a small profile icon (green circle for ES, blue circle for ZS) and a dropdown arrow icon. The sidebar on the left remains the same.

User	Role	Email	Phone	Mobile
ES	Admin	ertugrul@gmail.com	+49123456789	+49987654321
ZS	User	zehra@gmail.com	+49125469775	+49526547895

Abbildung 32: Benutzer eines Unternehmens

In dieser Liste kann nach dem Namen eines Benutzers gesucht werden. Außerdem wenn auf das Drop Down Icon für einen Benutzer geklickt wird, kommen alle Verträge, die dem Benutzer zugewiesen werden, in Sicht. (Siehe Abbildung 33)

No	Validity Date	Status	Version
10	01.Jan.2021 - 31.Dec.2022	Expires Shortly	1.0.4
		Responsible: Zehra Sevgili IPv4: 128.172.198.192	Responsible (2): Zehra Sevgili IPv6: aa:bb:cc:dd:ee:ff
11	01.Jan.2023 - 31.Dec.2023	Aktiv	1.0.3
		Responsible: Zehra Sevgili IPv4: 192.178.128.192	Responsible (2): Ertugrul Sevgili IPv6: ff:ff:ff:dd:ee:ff

Abbildung 33: Suchfunktion

Falls auf "Contracts" Button geklickt wird, erhält man alle Verträge des ausgewählten Unternehmens in einer Liste. Außerdem wenn auf das Drop Down Icon für einen Vertrag geklickt wird, kommen alle Angaben z.B. Lizenzen, IP-Adresse, usw. in Sicht (Siehe Abbildung 34)

No	Validity Date	Status	Version
10	01.Jan.2021 - 31.Dec.2022	Expires Shortly	1.0.4
		Responsible: Zehra Sevgili IPv4: 128.172.198.192 IPv4 (Optional): 10.16.0.0 IPv6: aa:bb:cc:dd:ee:ff License Key: /DoC3gocLplcu5wpGuPC6e8uN1!FEugZGD4QwfngmqD9gj221Hadprg2RrYAh98fYJuDwfYYJgJCrk43VDHEvTvgFX6e8HrvuQPjih0WkFAkQzaiScnULC+ces1o3QhF4qXyVhKDGlhGxgpHTdp8jIRZNB8AxJ04cQgHLoNFTPjQCCF9wd8+n16lSn/xQausJ	
11	01.Jan.2023 - 31.Dec.2023	Aktiv	1.0.3

Abbildung 34: Verträge eines Unternehmens

3.6.2.2 Admin Company

Nach dem Einloggen sieht ein Benutzer, der "*adminCompany*" Rolle hat, alle Benutzer seines Unternehmens in einer Liste auf Homepage. (Siehe Abbildung 32)

3.6.3 Users Page

3.6.3.1 Admin System

Ein Benutzer, der "adminCompany" Rolle hat, kann alle Benutzer aller Unternehmen in einer Liste sehen, indem auf "Users" Button links klickt. (Siehe Abbildung 35)

The screenshot shows the Microsoft Teams interface with the 'LMS-local-debug' tab selected. The main content area displays two company profiles: 'Alesta Solutions' and 'Iteratec GmbH'. Under 'Alesta Solutions', there is a search bar and a table listing users. The first user listed is Stephan Scholl (Admin) with the ID SS. Under 'Iteratec GmbH', there is also a table listing users. The first user listed is Ertugrul Sevgili (Admin) with the ID ES.

User	Role	Email	Phone	Mobile
Stephan Scholl	Admin	stephan@gmail.com	+491213456789	+4912345678
Michael Ballack	User	michael@gmail.com	+49582698536	+498854546546

User	Role	Email	Phone	Mobile
Ertugrul Sevgili	Admin	ertugrul@gmail.com	+49123456789	+49987654321
Zehra Sevgili	User	zehra@gmail.com	+49125469775	+49526547895

Abbildung 35: Users Page

Falls nach dem Namen des Unternehmens gesucht wird, werden entsprechende Treffe in Sicht geblieben. Außerdem, wenn auf das Drop Down Icon für einen Benutzer geklickt wird, kommen 2 Buttons für Löschen und Bearbeiten in Sicht. (Siehe Abbildung 36)

The screenshot shows the Microsoft Teams interface with the 'LMS-local-debug' tab selected. The main content area displays the 'Iteratec GmbH' profile. A search bar at the top contains the letters 'ite'. Below the search bar, the user table for 'Iteratec GmbH' is shown. The first user listed is Ertugrul Sevgili (Admin) with the ID ES. The table includes columns for User, Role, Email, Phone, Mobile, and a circular icon with a drop-down arrow. The circular icon for Ertugrul Sevgili has a blue edit icon and a red delete icon.

User	Role	Email	Phone	Mobile
Ertugrul Sevgili	Admin	ertugrul@gmail.com	+49123456789	+49987654321
Zehra Sevgili	User	zehra@gmail.com	+49125469775	+49526547895
Bera Sevgili	User	bera@gmail.com	+49544654654	+49984151634

Abbildung 36: Suchfunktion

Ein administrativer Benutzer kann einen Benutzer aktualisieren oder aus dem System entfernen (Siehe Abbildung 37 - 38)

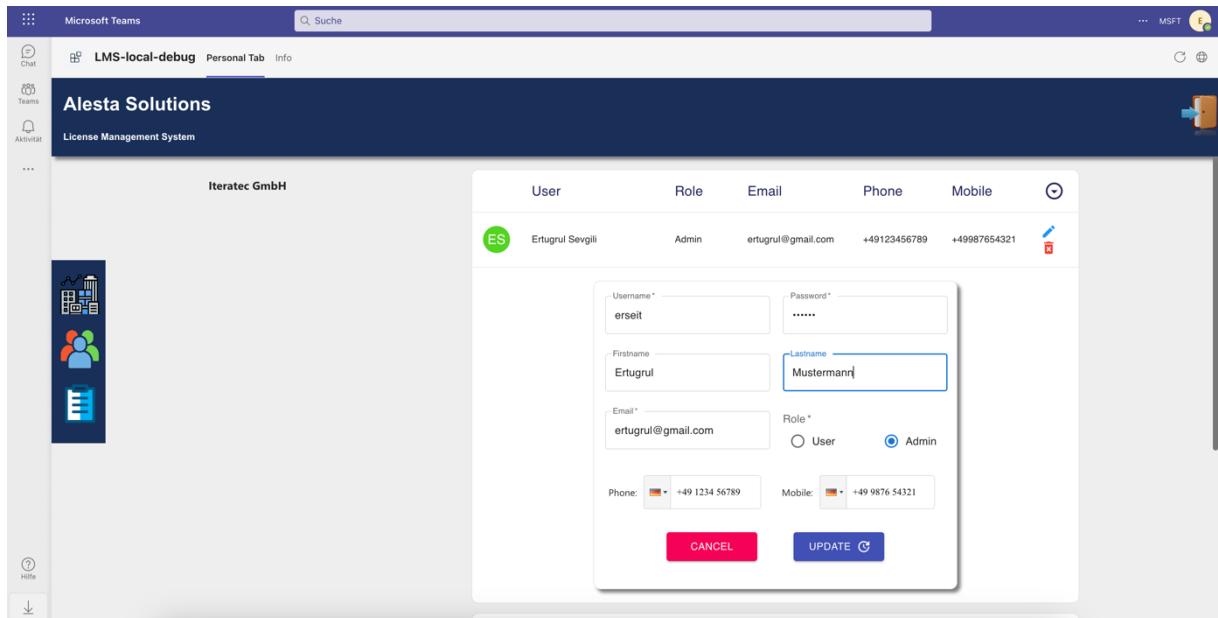


Abbildung 37: Aktualisieren eines Benutzers

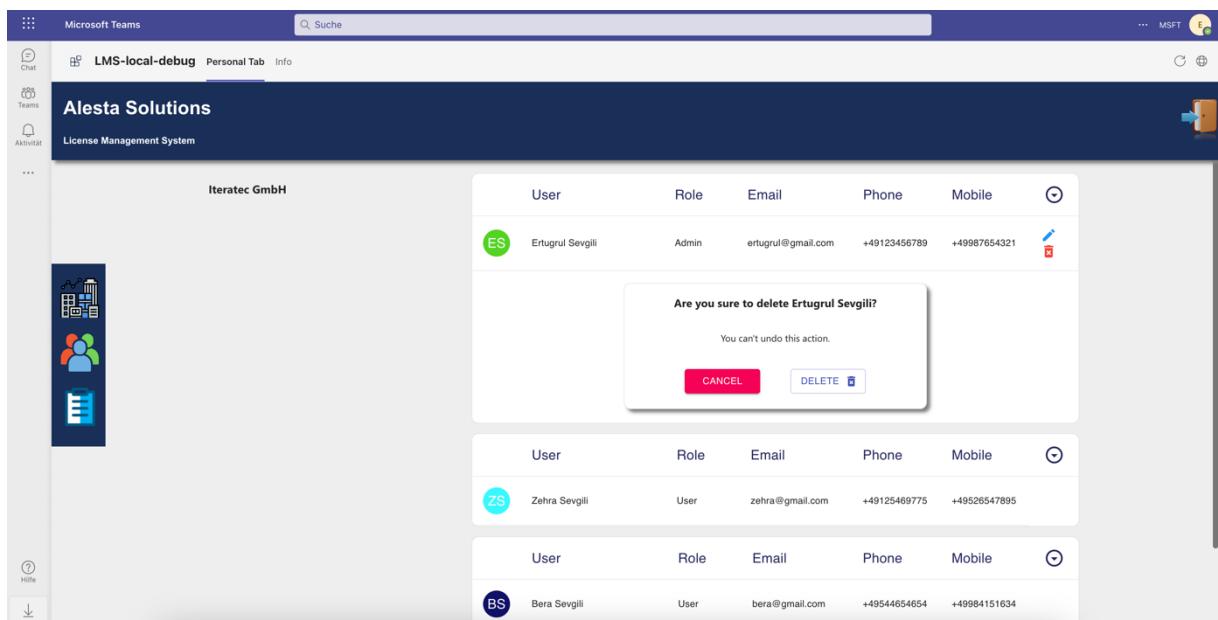


Abbildung 38: Löschen eines Benutzers

3.6.3.2 Admin Company

Ein administrativer Benutzer kann alle Benutzer seines Unternehmens in einer Liste sehen (Siehe Abbildung 32), bearbeiten (Siehe Abbildung 37) oder löschen (Siehe Abbildung 38).

3.6.4 Contracts Page

3.6.4.1 Admin System

Ein Benutzer, der "adminCompany" Rolle hat, kann alle Verträge aller Unternehmen in einer Liste sehen, indem auf "Contracts" Button links klickt. (Siehe Abbildung 39)

The screenshot shows the Microsoft Teams interface with the LMS-local-debug tab selected. On the left, there's a sidebar with icons for Chat, Teams, Aktivität, and Help. The main area is titled 'Alesta Solutions' and 'License Management System'. A search bar at the top says 'Search a company name'. Below it, there are two tables of contracts:

No	Validity Date	Status	Version
33	01.Jan.2021 - 31.Dec.2023	Aktiv	1.0.1

No	Validity Date	Status	Version
10	01.Jan.2021 - 31.Dec.2022	Expires Shortly	1.0.4

No	Validity Date	Status	Version
11	01.Jan.2023 - 31.Dec.2023	Aktiv	1.0.3

Abbildung 39: Contracts Page

Falls nach dem Namen des Unternehmens gesucht wird, werden entsprechende Treffe in Sicht geblieben. (Siehe Abbildung 40)

The screenshot shows the Microsoft Teams interface with the LMS-local-debug tab selected. The search bar at the top contains the text 'iter'. The main area is titled 'Iteratec GmbH' and 'License Management System'. The search results show one contract entry:

No	Validity Date	Status	Version
10	01.Jan.2021 - 31.Dec.2022	Expires Shortly	1.0.4

Details for this contract are expanded:

Responsible: Bera Sevgili
IPV4: 128.172.198.192
IPV4 (Optional): 10.16.0.0
IPV6: aa:bb:cc:dd:ee:ff
License Key:
X0zLMxSH980aQMeKAphjOlcPacF5tv940wfHmig7OpqXk8dykvjUU4T24Ry1HmY6whkgM2qagJ1zNzLk15+ZXDIKW
UkHsympQCPVmEqAFKvEGrV4zpoypUNMSxxaTADj4x9a0fIMRD6T29CgJ2/eGaFsznY9wB6zxw48G8cBYuClZAUst1
gxmpOzGYTTIUPdSc6jlOrfUnX

UPDATE KEY

No	Validity Date	Status	Version
11	01.Jan.2023 - 31.Dec.2023	Aktiv	1.0.3

Abbildung 40: Suchfunktion

Ein administrativer Benutzer kann einen Vertrag aktualisieren, einen neuen Key anfordern oder einen Vertrag aus dem System entfernen. (Siehe Abbildung 37 - 38)

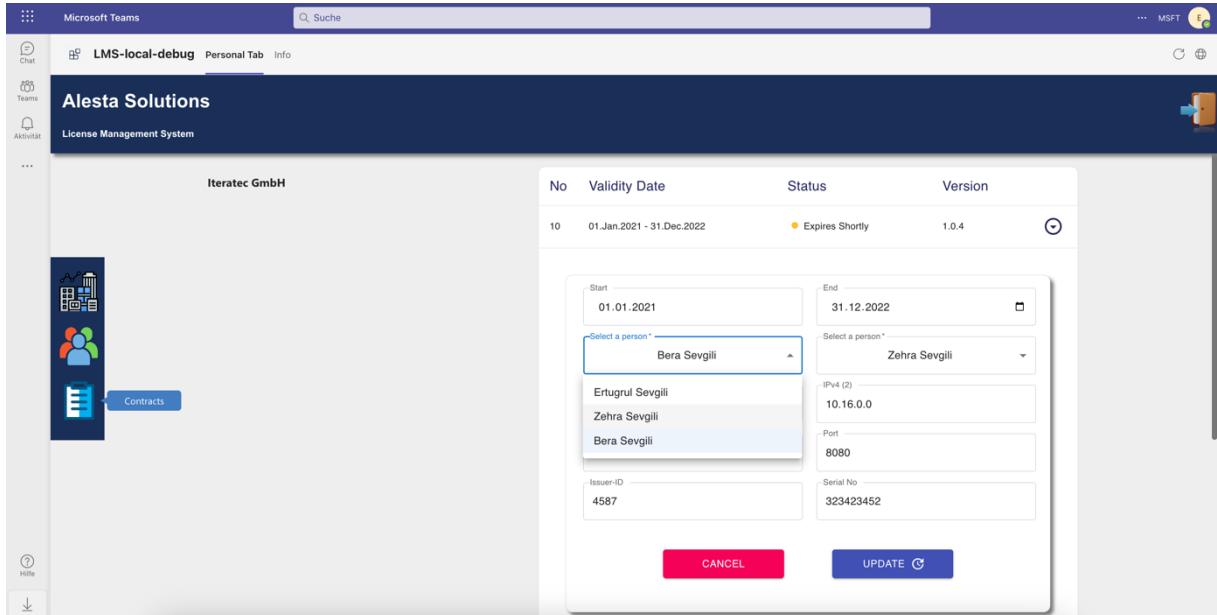


Abbildung 41: Aktualisieren eines Vertrags

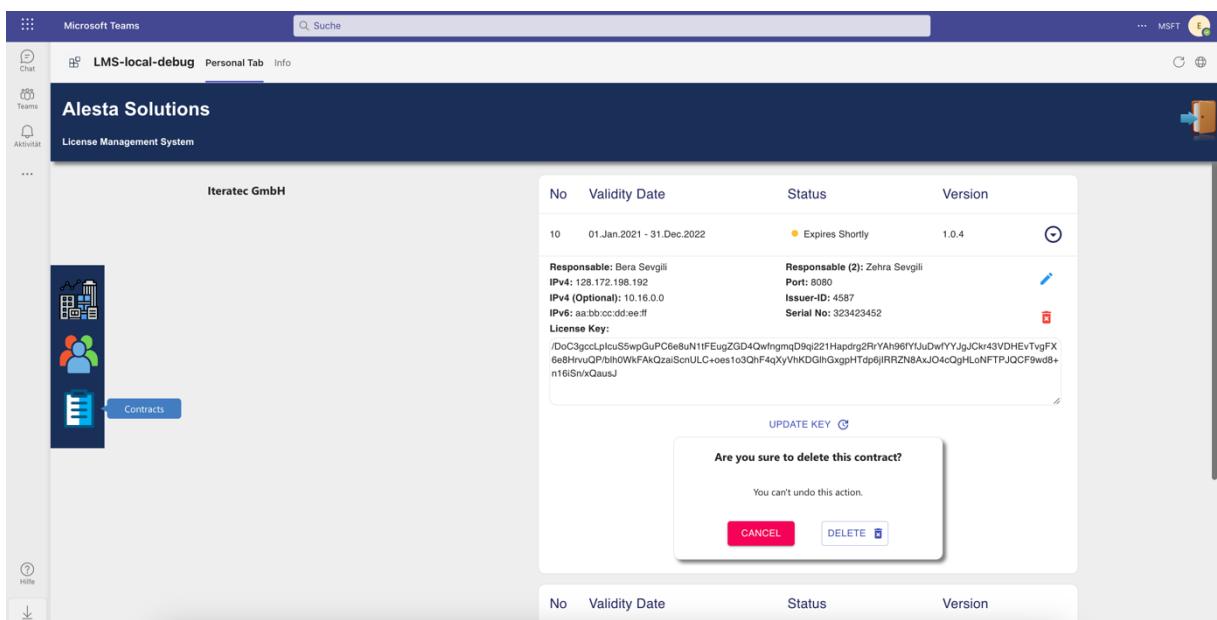


Abbildung 42: Löschen eines Vertrags

3.6.4.2 Admin Company

Ein administrativer Benutzer kann alle Verträge seines Unternehmens in einer Liste sehen (Siehe Abbildung 34), bearbeiten (Siehe Abbildung 41) oder löschen (Siehe Abbildung 42).

Literaturverzeichnis

1. Luber Stefan und Litzel Nico (2017), Was ist RDBMS?, in: bigdata-insider.de, 18.10.2017, <https://www.bigdata-insider.de/was-ist-rdbms-a-654230>, letzter Zugriff: 18.11.2022.
2. Friedrich, Jörg (2021), Software Architectures – Part 9: Relational Database Management System (Version 2.3) [Vorlesungsfolien], Hochschule Esslingen Esslingen a.N.,
3. IBM Cloud Education (2020), Dreischichtige Architektur, in: ibm.com, 28.10.2020, <https://www.ibm.com/de-de/cloud/learn/three-tier-architecture>, letzter Zugriff: 18.11.2022.
4. Friedrich, Jörg (2021), Software Architectures – Part 8: Project Starter (Version 2.1) [Vorlesungsfolien], Hochschule Esslingen Esslingen a.N.,
5. Hrsg: Wikipedia, Objekt Relationale Abbildung, in: https://de.wikipedia.org/w/index.php?title=Objektrelationale_Abbildung&oldid=223310082, letzter Zugriff: 14.11.2022
6. Friedrich, Jörg (2021), Software Architectures – Part 10: Java Persistent Architecture (Version 2.3) [Vorlesungsfolien], Hochschule Esslingen Esslingen a.N.,
7. Srocke Dirk und Karlstetter Florian (2017), Was ist eine REST API?, in: cloudcomputing-insider.de, 09.06.2017, <https://www.cloudcomputing-insider.de/was-ist-eine-rest-api-a-611116>, letzter Zugriff: 25.11.2022
8. Hrsg: Wikipedia, Objekt Representational State Transfer, in: https://de.wikipedia.org/w/index.php?title=Representational_State_Transfer&oldid=226788259, letzter Zugriff: 25.11.2022
9. Friedrich, Jörg (2021), Software Architectures – Part 12: Arcchitectural Pattern: SOA with JAX-WS and JAX-RS (Version 3.0) [Vorlesungsfolien], Hochschule Esslingen Esslingen a.N.,
10. Kranz Jan-Dirk (2020), Was ist JUnit? Was sind JUnit Tests?, in:it-talents.de, 03.02.2020, <https://it-talents.de/it-wissen/junit>, letzter Zugriff: 12.12.2022
11. Friedrich, Jörg (2021), Software Architectures – Part 11: Unit Testing with JUnit (Version 2.1) [Vorlesungsfolien], Hochschule Esslingen Esslingen a.N.,
12. Microsoft Teams (2022), Erstellen Ihrer ersten Registerkarten-App mit JavaScript, in: learn-microsoft.com, 17.10.2022, <https://learn.microsoft.com/de-de/microsoftteams/platform/sbs-gs-javascript?tabs=vscode%2Cvscode%2Cvscode>, letzter Zugriff: 27.11.2022