Dokumentation für die Semesterarbeit in den Fächern ia4.netz und Softwarengineering bei Herrn Klever und Herrn Seifert SoSe 21 von Richard Trembecki 2013222

# 1. Anforderungsanalyse mit Komplexitätsschätzung

- 1.1 User Stories und Planning Poker
- 1.2 MoSCoW-Priorisierung

### 2.Planung

2.1 Framework, Extensions und Libraries

## 3.Implementierung

- 3.1 Beschreibung der Software für Benutzer
- 3.2 Beschreibung der Software für Entwickler

### 4. Inbetriebnahme

### 5.Fazit

# **Einleitung**

Im Folgenden wird die Entwicklung und Umsetzung der Webanwendung "JAM" dokumentiert. "JAM" ist ein Tool für Musiker, um Bandproben zu arrangieren und sich dabei keinen Kopf um Terminverhandlungen machen zu müssen.

# 1. Anwendungsanalyse mit Komplexitätsschätzung 1.1 User Stories und Planning Poker

Folgende User Stories erzählen eine kurze Geschichte aus Sicht des Anwenders, welche Funktionalität an die Anwendung beschreiben soll. Jede Story wird nach dem Prinzip des Planning Pokers von trivial, einfach, mittel, schwierig, sehr schwierig, bis extrem schwierig in der Komplexität der Umsetzung eingestuft.

Als Arbeitnehmerin mit wenig Zeit möchte ich meine Freizeit optimal nutzen, um so oft wie möglich mit den Mitgliedern meiner Band zu musizieren.	mittel
Als Mitglied einer Band mit einem bevorstehenden Auftritt möchte ich einen Terminplan mit potenziellen Treffen, um mit der kompletten so effektiv wie möglich mit allen zu proben.	mittel
Als Chorleiterin möchte ich die große Anzahl der zeitlichen Verfügbarkeiten, der einzelnen Mitglieder, nicht selbst erfassen, um Zeit, beim planen einer Probe mit allen, zu sparen.	schwierig

### 1.2 MoSCoW-Priorisierung

Um die Ziele und Anforderungen an das Projekt zu konkretisieren und zu priorisieren, bediene ich mich der MoSCoW Methode.

Hierbei werden die Ziele in vier Kategorien unterteilt und zwar nach Must-,Should-, Could- und Would-Zielen.

Must	Mitglieder der Band geben ihre Verfügbarkeiten an, diese werden abgeglichen und ein Liste mit Terminen wird errechnet, die für jeden der Band einsehbar ist.
Should	Jedes Mitglied hat Einsicht in allen angegeben Verfügbarkeiten und kann sie gegebenenfalls löschen und neue Hinzufügen.
Could	Die errechneten Termine werden in einem Kalender dargestellt
Would	Die Anwendung ist in mehreren Sprachen verfügbar. Einbindung eines Raumplaners für den Proberaum, falls dieser von mehrere Parteien genutzt wird.

# 2. Planung

### 2.1 Framework

Für die Umsetzung des Projekts wurde Flask als Webframework gewählt.

Flask bietet eine sehr gute Dokumentation mit ausführlichen Erklärungen und Beispielen und lässt sich gut erweitern. Mit Flask benutzt man **Jinja2** als Templating-Engine, so wird der Programm-Code vom HTML-Code getrennt und macht diesen somit übersichtlicher. Zudem wird die Bibliothek **Werkzeug** verwendet, mit deren Hilfe WSGI-Anwendungen erstellt werden und die Passwörter verschlüsselt wurden.

Außerdem hat Flask einen eingebauten Development-Server, der das testen der Anwendung vereinfacht.

## 2.2 Extensions, Libraries und Werkzeuge

Da die Anwendung User Accounts sichert und verwaltet, werden mit Hilfe der Extension **Flask-Login** Prozesse wie Authentifizierung und Sitzungen des Users abgewickelt und verwaltet.

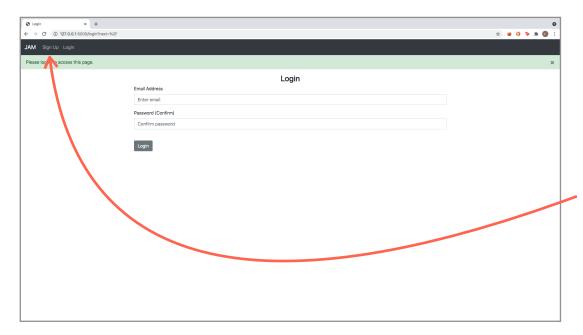
Die Extension **Flask-SQLAlchemy** vereinfacht die Verwendung der Library **SQLAlchemy** in Verbindung mit und die Ausführung allgemeiner Aufgaben.

Als Entwicklungsumgebung habe ich Visual Studio Code (VSCode) von Microsoft verwendet.

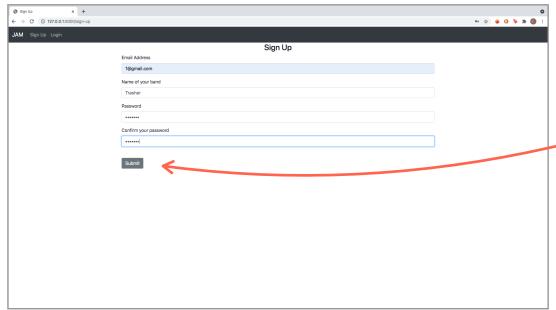
Zudem wurde ein Repository auf "github.com" angelegt und das Versionierungssystem Git verwendet. So kann der Code anderen Menschen zugänglich gemacht werden und vereinfacht eine Teamarbeit und ältere Versionen bleiben erhalten.

# 3. Implementierung

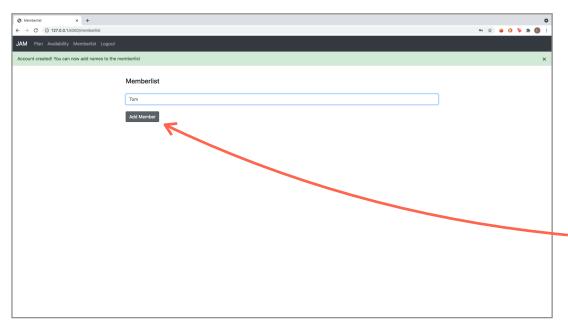
# 3.1 Beschreibung der Software für Benutzer



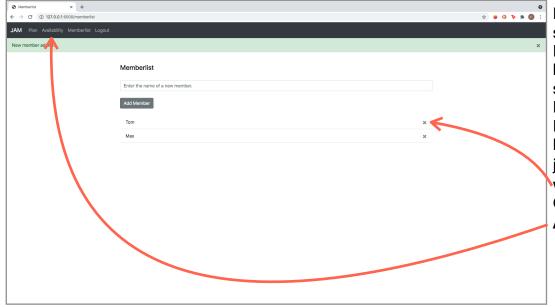
Nach öffnen der Website kommen sie auf **Login**. Sollten sie noch keine Login-Daten angegeben haben, dann klicken sie auf **Sign Up**, um sich erstmalig anzumelden.



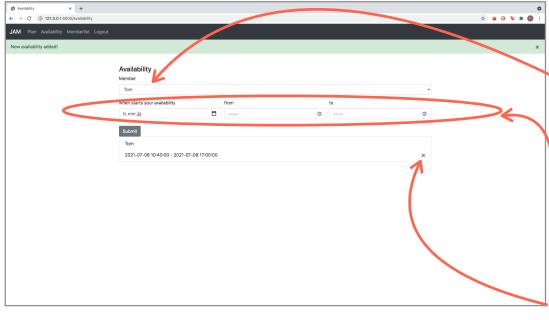
Geben sie die gefragten Daten an. Achten sie darauf, das Passwort muss 7 Stellen habe. Klicken sie auf Submit.



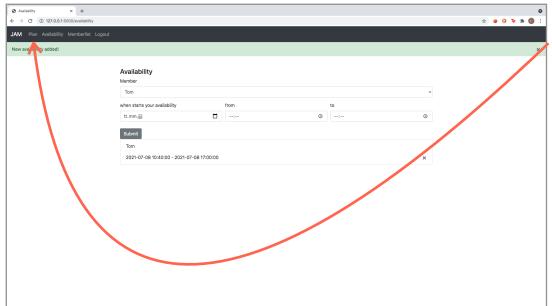
Sie haben nun einen Account für ihre Band erstellt und können die Namen der Mitglieder hinzufügen. Geben sie einfach den Namen ein und klicken sie auf Add Member.



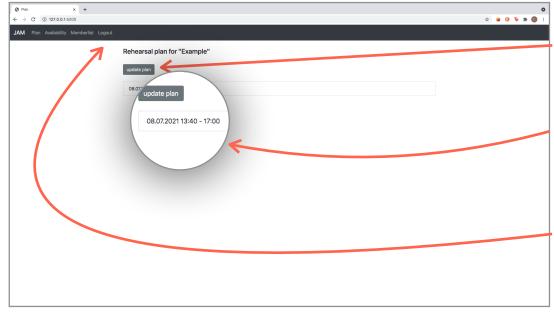
Hier wurden soeben zwei Mitglieder hinzugefügt. Sie sehen eine aktuelle Liste aller Mitglieder. Hier können sie das jeweilige Mitglied wieder löschen. Gehen sie dann auf Availability



Hier geben sie ihre Verfügbarkeit an. Wählen sie das Mitglied aus für das die Verfügbarkeit gilt. Dann **Datum**, Beginn und Ende als Uhrzeit angeben und auf Submit klicken. Hier sehen sie bereits angegebene Verfügbarkeiten, die **hier** wieder gelöscht werden können



Gehen sie dann auf **Plan** um zur Auswertung der Daten zu kommen.



Klicken sie auf
Update Plan um
die Auswertung
und den
errechneten
Termin, oder die
errechneten
Termine zu sehen
zu sehen.
Wenn sie fertig
sind klicken sie auf
Logout.

## 3. Beschreibung der Software für Entwickler

#### **Errechnen des Plans**

Der Kern der Anwendung ist es die angegeben Verfügbarkeiten auszulesen zu sortieren und auszugeben

Das passiert unter IA\_NETZ\_PROJEKT> website > views.py

return render\_template("plan.html", user=current\_user, result = result)

```
@views.route('/', methods=['GET', 'POST'])
@login_required
def home():
   av = Availability.query.all()
   memb = Member.query.filter_by(user_id=current_user.id).all()
    tday = datetime.now()
   strtdts = []
   nested_strtdts = []
   enddts = []
   nested enddts = []
   days = []
    final_strtdts = []
   final_enddts = []
   result = []
   if request.method == 'POST':
                                                                   Zuerst werden alle angegeben
        # get all start and end dates from db
                                                                   Startdaten und Enddaten abgefragt
       for x in range (1, len(av) + 1, 1):
                                                                   und in jeweils zwei Listen
           y = Availability.query.filter_by(id=x).first()
           if y.user_id == current_user.id:
    #if y.startdate >= tday:
                                                                   abgespeichert und sortiert. Sodass in
               strtdts.append(y.startdate)
                                                                   der Liste der Startdaten das späteste
               #latest startdate should be first index
                                                                   zuerst kommt und bei den Enddaten
               strtdts = sorted(strtdts, reverse=True)
               #earliest enddate should be first index
                                                                   das früheste zuerst.
               #if v.enddate >= tday:
               enddts.append(y.enddate)
               enddts = sorted(enddts, reverse=False)
                                                                         "daycount" ist eine Liste mit Tagen an
       #get days where avalibilities are declared
       for b in range(len(strtdts)):
                                                                         denen mindestens eine Verfügbarkeit
           days.append(strtdts[b].day)
           daycount= list(set(days))
                                                                         angegeben wurde
       #creating a nested list with sub lists of dates with same day
                                                                                    Dann wird eine Liste mit Listen
        #fort start dates
       for c in range(len(davcount)):
                                                                                    gefüllt, deshalb "nested". Die
           innerList = []
                                                                                    Unter-Listen enthalten immer
           for d in range (len(strtdts)):
              if strtdts[d].day == daycount[c] :#and strtdts[d].month == tday.month: Verfügbarkeiten des selben
                   innerList.append(strtdts[d])
           nested_strtdts.append(innerList)
                                                                                    Tages. Einmal für Startdaten
       #and end dates
                                                                                    und Enddaten der Mitglieder.
       for c in range(len(daycount)):
           innerList = []
           for d in range (len(enddts)):
               if enddts[d].day == daycount[c]: #and enddts[d].month == tday.month:
                   innerList.append(enddts[d])
           nested_enddts.append(innerList)
                                                                       Sollte eine Unter-Liste nicht die gleiche
       #checking if every member gave a availability for this day.
           #if not, the plan should not be calculated
                                                                       Länge wie die Liste aller Mitglieder haben,
        for c in range(len(daycount)):
           if(len(nested_strtdts[c])) == len(memb):
                                                                      heißt das, dass für diesen Tag nicht alle
               final_strtdts.append(nested_strtdts[c][0])
                                                                      Mitglieder eine Verfügbarkeit angegeben
           if(len(nested enddts[c])) == len(memb):
                                                                      haben. Diese Tage werden aus der finalen
               final_enddts.append(nested enddts[c][0])
                                                                      Liste ausgeschlossen
       for c in range(len(final strtdts)):
           result.append(final\_strtdts[c].strftime('%d.%m.%Y %H:%M') + ' - ,+ final\_enddts[c].strftime('%H:%M'))
       if len(result)>1:
                                                                         Aus den beiden Unter-Listen wurden nun
           if result[0] > result[1]:
               result.reverse()
```

Aus den beiden Unter-Listen wurden nun immer die ersten Werte entnommen. Diese definieren nun den Start und das Ende der finalen Treffen der Band und können an das Frontend als String übergeben werden.

Die Schwierigkeit hierbei war, sich ein System zu überlegen, welches die Werte als Liste in kleinere Listen aufspaltet und dann wiederum in eine übergeordnete Liste, nach folgendem Prinzip ablegt.

```
Liste = [[Datum, Datum], [Datum, Datum, Datum], [Datum, Datum]]
```

Das Kriterium für die Unter-Listen war, dass sie den gleichen Wert für den Tag hatten. Hier nochmal der Abschnitt:

#### Verwaltung der User

siehe IA\_NETZ\_PROJEKT> website > auth.py

```
from flask import Blueprint, render template, request, flash, redirect, url for
from .models import User
from werkzeug.security import generate password hash, check password hash
from . import db
from flask login import login user, login required, logout user, current user
auth = Blueprint('auth', __name__)
@auth.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
       email = request.form.get('email')
        password = request.form.get('password')
        user = User.query.filter by(email=email).first()
        if user:
            if check password hash (user.password, password):
                flash('Logged in successfully!', category='success')
                login_user(user, remember=True)
                return redirect(url for('views.home'))
            else:
                flash('Incorrect password, try again.', category='error')
        else:
            flash('Email does not exist. Go to sign up', category='error')
    return render template("login.html", user=current user)
@auth.route('/logout')
@login required
def logout():
   logout_user()
    return redirect(url_for('auth.login'))
```

```
@auth.route('/sign-up', methods=['GET', 'POST'])
def sign_up():
    if request.method == 'POST':
        email = request.form.get('email')
        bandname = request.form.get('bandname')
        password1 = request.form.get('password1')
        password2 = request.form.get('password2')
        user = User.query.filter by(email=email).first()
            flash('Email already exists.', category='error')
        elif len(email) < 4:
            flash('Email must be greater than 3 characters.', category='error')
        elif len(bandname) < 2:
            flash('First name must be greater than 1 character.', category='error')
        elif password1 != password2:
            flash('Passwords don\'t match.', category='error')
        elif len(password1) < 7:</pre>
            flash('Password must be at least 7 characters.', category='error')
        else:
            new user = User(email=email, bandname=bandname,
password=generate password hash(
                password1, method='sha256'))
            db.session.add(new user)
            db.session.commit()
            login user(new user, remember=True)
            flash('Account created! You can now add names to the memberlist',
category='success')
            return redirect(url for('views.memberlist'))
    return render template("sign up.html", user=current user)
```

vgl.:,creating a Basic website", URL: https://www.techwithtim.net/tutorials/flask/(Stand: 2.7.21). vgl.:https://github.com/techwithtim/Flask-Web-App-Tutorial/blob/main/website/auth.py(Stand:2.7.21)

# **Löschung von Mitgliedern und Verfügbarkeiten** siehe *IA\_NETZ\_PROJEKT> website > views.py*

```
@views.route('/memberlist', methods=['GET', 'POST'])
@login required
def memberlist():
    if request.method == 'POST':
        member = request.form.get('member')
        if len(member) < 1:
            flash('The name of the new member is too short!',
                  category='error')
        else:
            new member = Member(name=member, user id=current user.id)
            db.session.add(new member)
            db.session.commit()
            flash(' New member added!', category='success')
    return render_template("memberlist.html", user=current_user)
@views.route('/delete-member', methods=['POST'])
def delete member():
   member = json.loads(request.data)
   memberId = member['memberId']
   member = Member.query.get(memberId)
    if member:
        if member.user_id == current_user.id:
            db.session.delete(member)
            db.session.commit()
    return jsonify({})
```

```
@views.route('/availability', methods=['GET', 'POST'])
@login_required
def availability():
    if request.method == 'POST':
        form date = request.form.get('date')
        form starttime = request.form.get('starttime')
        form endtime = request.form.get('endtime')
        member = request.form.get('memberchoice')
        startdate = datetime.strptime(form_date +" "+ form_starttime,"%Y-%m-%d %H:%M") enddate = datetime.strptime(form_date +" "+ form_endtime,"%Y-%m-%d %H:%M")
        present = datetime.now()
        if not form date :
             flash('Values are missing.', category= 'error')
        elif startdate < present:
            flash('The date is in the past.', category = 'error')
        elif enddate <= startdate:</pre>
             flash('Your availability ends earlier than it starts.', category = 'error')
            new availability = Availability(startdate = startdate, enddate = enddate,
member = member, user id=current user.id)
            db.session.add(new availability)
            db.session.commit()
             flash(' New availability added!', category='success')
    return render template("availability.html", user=current user)
@views.route('/delete-availability', methods=['POST'])
def delete availability():
    availability = json.loads(request.data)
    availabilityId = availability['availabilityId']
    availability = Availability.query.get(availabilityId)
    if availability:
        if availability.user id == current user.id:
            db.session.delete(availability)
            db.session.commit()
    return jsonify({})
```

vgl.:,,creating a Basic website", URL: https://www.techwithtim.net/tutorials/flask/(Stand: 2.7.21). vgl.:https://github.com/techwithtim/Flask-Web-App-Tutorial/blob/main/website/views.py(Stand:2.7.21)

#### siehe IA\_NETZ\_PROJEKT> static> Index.js

```
function deleteMember(memberId) {
  fetch("/delete-member", {
    method: "POST",
    body: JSON.stringify({ memberId: memberId }),
  }).then(( res) => {
    window.location.href = "/memberlist";
  });
}
function deleteAvailability(availabilityId) {
  fetch("/delete-availability", {
   method: "POST",
   body: JSON.stringify({ availabilityId: availabilityId }),
  }).then(( res) => {
    window.location.href = "/availability";
 });
}
```

vgl.:"creating a Basic website", URL: https://www.techwithtim.net/tutorials/flask/(Stand: 2.7.21).vgl.:https://github.com/ techwithtim/Flask-Web-App-Tutorial/blob/main/static/index.js(Stand:2.7.21)

#### siehe IA\_NETZ\_PROJEKT> website> \_\_init\_\_.py

```
from flask import Flask
from flask sqlalchemy import SQLAlchemy
from os import path
from flask login import LoginManager
db = SQLAlchemy()
DB_NAME = "database.db"
def create_app():
    app = \overline{F}lask(name)
    app.config['SECRET KEY'] = 'rt'
    app.config['SQLALCHEMY_DATABASE_URI'] = f'sqlite:///{DB_NAME}'
    db.init app(app)
    from .views import views
    from .auth import auth
    app.register_blueprint(views, url_prefix='/')
    app.register_blueprint(auth, url_prefix='/')
    from .models import User, Availability, Member
    create database(app)
    login manager = LoginManager()
    login_manager.login_view = 'auth.login'
    login manager.init app(app)
    @login_manager.user_loader
    def load user(id):
        return User.query.get(int(id))
    return app
def create database(app):
    if not path.exists('website/' + DB NAME):
        db.create_all(app=app)
        print('Created Database!')
```

vgl.:,,creating a Basic website", URL: https://www.techwithtim.net/tutorials/flask/(Stand: 2.7.21).vgl.:https://github.com/ techwithtim/Flask-Web-App-Tutorial/blob/main/website/\_init\_\_.py(Stand:2.7.21)

### siehe IA\_NETZ\_PROJEKT> website> main.py

```
from website import create_app
app = create_app()

if __name__ == '__main__':
    app.run(debug=True)
```

"creating a Basic website", URL: https://www.techwithtim.net/tutorials/flask/(Stand: 2.7.21).vgl.:https://github.com/techwithtim/Flask-Web-App-Tutorial/blob/main/website/main.py(Stand:2.7.21)

### 4. Inbetriebnahme

#### Setup

Zuerst sollten sie sich vergewissern das Python installiert ist, dann können sie fortfahren und alle Anforderungen installieren.

Über pip install -r requirements.txt greifen sie auf eine .txt Datei zu, die folgende Information enthält und dann installiert.

flask
Flask-SQLAlchemy
flask-login

#### Starten der Anwendung über den internen Development-Server

Starten sie main.py.

python main.py

sie erreichen über http://127.0.0.1:5000 nun die Anwendung und können die sie testen.

### 5. Fazit

Was dieses Projekt besonders interessant machte, war das Erkennen eines Problems, dieses zu Analysieren und durch Projektmethoden zu lösen.

Das Anwenden der mir neuen Programmiersprache Python, stellte sich dabei zunächst als Herausforderung dar. Durch ausführliche Recherche und zur Hand nehmen der Python-Dokumentation konnte ich anfängliche Unwissenheit beseitigen. Für die Umsetzung von zukünftigen Projekten dieser Art, werde wieder auf die genaue Analyse der Nutzeranforderungen und Projektmethode setzen.

### Quellen:

- https://www.techwithtim.net/tutorials/flask/a-basic-website/
   https://github.com/techwithtim/Flask-Web-App-Tutorial
   "Flask documentation" URL:https://flask.palletsprojects.com/en/2.0.x/

# Erklärung zur Abschlussarbeit

Hiermit versichere ich, die eingereichte Abschlussarbeit selbständig verfasst und keine andere als die von mir angegebenen Quellen und Hilfsmittel benutzt zu haben. Wörtlich oder inhaltlich verwendete Quellen wurden entsprechend den anerkannten Regeln wissenschaftlichen Arbeitens zitiert. Ich erkläre weiterhin, dass die vorliegende Arbeit noch nicht anderweitig als Abschlussarbeit eingereicht wurde.

Das Merkblatt zum Täuschungsverbot im Prüfungsverfahren der Hochschule Augsburg habe ich gelesen und zur Kenntnis genommen. Ich versichere, dass die von mir abgegebene Arbeit keinerlei Plagiate, Texte oder Bilder umfasst, die durch von mir beauftragte Dritte erstellt wurden.

	2. Tiembech
Augsburg, 3.7.21	
Ort, Datum	Unterschrift des/der Studierenden