

Dokumentation für die
Semesterarbeit in den Fächern
ia4.netz und **Softwareengineering**
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 eine Liste mit Terminen wird errechnet, die für jeden der Band einsehbar ist.
Should	Jedes Mitglied hat Einsicht in allen angegebenen 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 mehreren 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

127.0.0.1:5000/login?next=%2F

JAM Sign Up Login

Please login to access this page.

Login

Email Address
Enter email

Password (Confirm)
Confirm password

Login

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.

127.0.0.1:5000/sign-up

JAM Sign Up Login

Sign Up

Email Address
1@gmail.com

Name of your band
Trasher

Password

Confirm your password

Submit

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

127.0.0.1:5000/memberlist

JAM Plan Availability Memberlist Logout

Account created! You can now add names to the memberlist

Memberlist

Tom

Add Member

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**.

Memberlist

Enter the name of a new member.

Add Member

Tom	x
Max	x

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**

Availability

Member

Tom

When starts your availability

from to

tt.mm.jjjj

Submit

Tom	2021-07-08 10:40:00 - 2021-07-08 17:00:00	x
-----	---	---

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

Availability

Member

Tom

when starts your availability

from

to

tt.mm.jjjj

Submit

Tom

2021-07-08 10:40:00 - 2021-07-08 17:00:00

New availability added!

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

Rehearsal plan for "Example"

update plan

08.07.

update plan

08.07.2021 13:40 - 17:00

Klicken sie auf **Update Plan** um die Auswertung und den **errechneten Termin**, oder die **errechneten Termine** 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 angegebenen Verfügbarkeiten auszulesen zu sortieren und auszugeben

Das passiert unter **IA_NETZ_PROJEKT> website > views.py**

```
@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()
    strtlds = []
    nested_strtlds = []
    enddts = []
    nested_enddts = []
    days = []
    final_strtlds = []
    final_enddts = []
    result = []

    if request.method == 'POST':

        # get all start and end dates from db

        for x in range(1, len(av)+1, 1):
            y = Availability.query.filter_by(id=x).first()
            if y.user_id == current_user.id:
                #if y.startdate >= tday:
                strtlds.append(y.startdate)
                #latest startdate should be first index
                strtlds = sorted(strtlds, reverse=True)
                #earliest enddate should be first index
                #if y.enddate >= tday:
                enddts.append(y.enddate)
                enddts = sorted(enddts, reverse=False)

        #get days where availabilities are declared
        for b in range(len(strtlds)):
            days.append(strtlds[b].day)
            daycount= list(set(days))

        #creating a nested list with sub lists of dates with same day
        #fort start dates
        for c in range(len(daycount)):
            innerList = []
            for d in range (len(strtlds)):
                if strtlds[d].day == daycount[c] :#and strtlds[d].month == tday.month:
                    innerList.append(strtlds[d])
            nested_strtlds.append(innerList)
        #and end dates
        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)

        #checking if every member gave a availability for this day.
        #if not, the plan should not be calculated
        for c in range(len(daycount)):
            if(len(nested_strtlds[c]))== len(memb):
                final_strtlds.append(nested_strtlds[c][0])

            if(len(nested_enddts[c]))== len(memb):
                final_enddts.append(nested_enddts[c][0])

        for c in range(len(final_strtlds)):
            result.append(final_strtlds[c].strftime('%d.%m.%Y %H:%M') + ' - ,+ final_enddts[c].strftime('%H:%M'))

        if len(result)>1:
            if result[0] > result[1]:
                result.reverse()

    return render_template("plan.html", user=current_user, result = result)
```

Zuerst werden alle angegebenen Startdaten und Enddaten abgefragt und in jeweils zwei Listen abgespeichert und sortiert. Sodass in der Liste der Startdaten das späteste zuerst kommt und bei den Enddaten das früheste zuerst.

„daycount“ ist eine Liste mit Tagen an denen mindestens eine Verfügbarkeit angegeben wurde

Dann wird eine Liste mit Listen gefüllt, deshalb „nested“. Die Unter-Listen enthalten immer Verfügbarkeiten des selben Tages. Einmal für Startdaten und Enddaten der Mitglieder.

Sollte eine Unter-Liste nicht die gleiche Länge wie die Liste aller Mitglieder haben, heißt das, dass für diesen Tag nicht alle Mitglieder eine Verfügbarkeit angegeben haben. Diese Tage werden aus der finalen Liste ausgeschlossen

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:

```
#creating a nested list with sub lists of dates with same day
#fort start dates
for c in range(len(daycount)):
    innerList = []
    for d in range (len(strtdts)):
        if strtdts[d].day == daycount[c] :#and strtdts[d].month == tday.month:
            innerList.append(strtdts[d])
    nested_strtdts.append(innerList)
#and end dates
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)
```

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()
        if user:
            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:
            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/\(Stand: 2.7.21\).](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:
            flash('Your availability ends earlier than it starts.', category = 'error')
        else:
            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 = Flask(__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\(Stand:2.7.21\)](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\(Stand:2.7.21\)*](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.



Augsburg, 3.7.21

Ort, Datum

Unterschrift des/der Studierenden