Verteilte Systeme Praktikum

Christopher Epp 10013650
Erik Stotz 10013461

Anleitung zum Starten der Anwendung

Es muss Phyton installiert werden und ausgeführt.
Ebenfalls das Paket Management Tool pip.

Folgende Sachen müssen ausgeführt werden.

```
apt-get update
pip install --upgrade setuptools
python -m pip install --upgrade pip
pip install flask
pip install flask-restplus
python main.py
```

Die Swagger Dokumentation ist danach über folgenden Link aufrufbar.

```
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
Beispiel: http://0.0.0.0:5000//users
```

Begründung der gewählten Programmiersprache und der verwendeten Frameworks:

Wir haben die Sprache Phyton gewählt, da wir mit ihr mehr vertraut sind als mit Java.
Ebenfalls finden wir es bei Phyton übersichtlicher einen Fehler in dem Code zu finden.

Die Frameworks die verwendet wurden sind flask und flask-restplus.
Wir haben Flask gewählt, da es besonders für Einsteiger leicht zu bedienen ist und man es schnell verstehen und anwenden kann. Ebenfalls hat Flask eine sehr gute Bedienung mit Pyhton und der API die wir gemacht haben.
Flask-Restplus haben wir gewählt, weil es die Implementierung von der REST möglich macht.

Teststrategie:
Wir haben es mit Swagger getestet und einzeln ausgeführt.
Hierfür haben wir auch die wie oben beschriebene Seite mit http://0.0.0.0:5000/ genutzt um die Daten und eingaben zu testen.
UI Tests über den Web Browser, durch Klick auf „execute“.

Eventuelle Limitierungen:
Muss auf Linux ausgeführt werden.
Es wurde auf einem MAC ausgeführt unter einer Linux Umgebung und man muss bestimmt Sachen installieren (siehe oben).

Quellcode der Implementierung:

```
# import werkzeug
# werkzeug.cached_property = werkzeug.utils.cached_property
#
# from flask import Flask
# from flask_restplus import Api, Resource, fields
#
# app = Flask(__name__)
# api = Api(app)
#
# @api.route('/hello/')
# class HelloWorld(Resource):
#     def get(self):
#         return "Hello World"
#
# if __name__ == '__main__':
#     app.run()
from functools import partial

import pickle
def save(obj, filename):
    f=open("data/"+filename,'wb')
    pickle.dump(obj, f, 2)
    f.close()
def load(filename, dao):
    try:
        f=open("data/"+filename,'rb') #opening the file to read the data in the binary form
    except IOError:
        return dao
    with f:
        return pickle.load(f)



import werkzeug
werkzeug.cached_property = werkzeug.utils.cached_property

from flask import Flask

# import flask.scaffold
# flask.helpers._endpoint_from_view_func = flask.scaffold._endpoint_from_view_func
# import flask_restful
from flask_restplus import Api, Resource, fields
# import flask_restplus.

app = Flask(__name__)
#app.config['SERVER_NAME'] = 'restapi.usblion.de'
app.config['PREFERRED_URL_SCHEME'] = 'https'
api = Api(app, version='1.0', title='RestFul API',
    description='Christopher Epp - Flask - Swagger',
)
```

```
nsUsers = api.namespace('users', description='Stammdaten der Kunden')
nsProducts = api.namespace('products', description='Produktdaten')
nsQuestions = api.namespace('questions', description='Kontaktformular')
nsReviews = api.namespace('reviews', description='Rezensiondaten')
nsOrders = api.namespace('orders', description='Bestellungen')

user = api.model('User', {
    'id': fields.Integer(readOnly=True, description='The user unique identifier'),
    'firstname': fields.String(required=True, description='Vorname'),
    'lastname': fields.String(required=True, description='Nachname'),
    'email': fields.String(required=True, description='E-Mail Adresse'),
    'address': fields.String(required=True, description='Anschrift'),
    'password': fields.String(required=True, description='Passwort')
})

product = api.model('Product', {
    'id': fields.Integer(readOnly=True, description='The user unique identifier'),
    'name': fields.String(required=True, description='Titel'),
    'brand': fields.String(required=True, description='Marke'),
    'newprice': fields.String(required=True, description='Aktionspreis'),
    'oldprice': fields.String(required=True, description='Preis')
})

question = api.model('Question', {
    'id': fields.Integer(readOnly=True, description='The user unique identifier'),
    'firstname': fields.String(required=True, description='Vorname'),
    'lastname': fields.String(required=True, description='Nachname'),
    'email': fields.String(required=True, description='E-Mail Adresse'),
    'subject': fields.String(required=True, description='Betreff'),
    'customerid': fields.String(required=True, description='Kundennummer'),
    'description': fields.String(required=True, description='Beschreibung')
})

review = api.model('Review', {
    'id': fields.Integer(readOnly=True, description='The user unique identifier'),
    'publisher': fields.String(required=True, description='Publiziert von'),
    'starrating': fields.String(required=True, description='Bewertung'),
    'text': fields.String(required=True, description='Rezension')
})

order = api.model('Order', {
    'id': fields.Integer(readOnly=True, description='The user unique identifier'),
    'date': fields.String(required=True, description='Datum'),
    'payment': fields.String(required=True, description='Bezahlmethode'),
    'total': fields.String(required=True, description='Summe')
})

class UserDAO(object):
    def __init__(self):
        self.counter = 0
        self.users = []
```

```
        self.filename="UserDAO.bin"

    def get(self, id):
        for user in self.users:
            if user['id'] == id:
                return user
        api.abort(404, "User {} doesn't exist".format(id))

    def create(self, data):
        user = data
        user['id'] = self.counter = self.counter + 1
        self.users.append(user)
        save(self, self.filename)
        return user

    def update(self, id, data):
        user = self.get(id)
        user.update(data)
        save(self, self.filename)
        return user

    def delete(self, id):
        user = self.get(id)
        self.users.remove(user)
        save(self, self.filename)

    def updateAll(self, data):
        for user in data:
            self.update(user['id'], user)
        save(self, self.filename)
        return data

    def deleteAll(self):
        self.users = []
        save(self, self.filename)

class ProductDAO(object):
    def __init__(self):
        self.counter = 0
        self.products = []
        self.filename="ProductDAO.bin"

    def get(self, id):
        for product in self.products:
            if product['id'] == id:
                return product
        api.abort(404, "Product {} doesn't exist".format(id))

    def create(self, data):
        product = data
        product['id'] = self.counter = self.counter + 1
        self.products.append(product)
```

```
        save(self, self.filename)
        return product

    def update(self, id, data):
        product = self.get(id)
        product.update(data)
        save(self, self.filename)
        return product

    def delete(self, id):
        product = self.get(id)
        self.products.remove(product)
        save(self, self.filename)

    def updateAll(self, data):
        for product in data:
            self.update(product['id'], product)
        save(self, self.filename)
        return data

    def deleteAll(self):
        self.products = []
        save(self, self.filename)

class QuestionDAO(object):
    def __init__(self):
        self.counter = 0
        self.questions = []
        self.filename="QuestionDAO.bin"

    def get(self, id):
        for question in self.questions:
            if question['id'] == id:
                return question
        api.abort(404, "Question {} doesn't exist".format(id))

    def create(self, data):
        question = data
        question['id'] = self.counter = self.counter + 1
        self.questions.append(question)
        save(self, self.filename)
        return question

    def update(self, id, data):
        question = self.get(id)
        question.update(data)
        save(self, self.filename)
        return question

    def delete(self, id):
        question = self.get(id)
        self.questions.remove(question)
```

```
        save(self, self.filename)

    def updateAll(self, data):
        for question in data:
            self.update(question['id'], question)
        save(self, self.filename)
        return data

    def deleteAll(self):
        self.questions = []
        save(self, self.filename)

class ReviewDAO(object):
    def __init__(self):
        self.counter = 0
        self.reviews = []
        self.filename="ReviewDAO.bin"

    def get(self, id):
        for review in self.reviews:
            if review['id'] == id:
                return review
        api.abort(404, "Review {} doesn't exist".format(id))

    def create(self, data):
        review = data
        review['id'] = self.counter = self.counter + 1
        self.reviews.append(review)
        save(self, self.filename)
        return review

    def update(self, id, data):
        review = self.get(id)
        review.update(data)
        save(self, self.filename)
        return review

    def delete(self, id):
        review = self.get(id)
        self.reviews.remove(review)
        save(self, self.filename)

    def updateAll(self, data):
        for review in data:
            self.update(review['id'], review)
        save(self, self.filename)
        return data

    def deleteAll(self):
        self.reviews = []
        save(self, self.filename)
```

```
class OrderDAO(object):
    def __init__(self):
        self.counter = 0
        self.orders = []
        self.filename="OrderDAO.bin"

    def get(self, id):
        for order in self.orders:
            if order['id'] == id:
                return order
        api.abort(404, "Order {} doesn't exist".format(id))

    def create(self, data):
        order = data
        order['id'] = self.counter = self.counter + 1
        self.orders.append(order)
        save(self, self.filename)
        return order

    def update(self, id, data):
        order = self.get(id)
        order.update(data)
        save(self, self.filename)
        return order

    def delete(self, id):
        order = self.get(id)
        self.orders.remove(order)
        save(self, self.filename)

    def updateAll(self, data):
        for order in data:
            self.update(order['id'], order)
        save(self, self.filename)
        return data

    def deleteAll(self):
        self.orders = []
        save(self, self.filename)


userDAO_prefab = UserDAO()
userDAO_prefab.create({'firstname': 'Max', 'lastname': 'Mustermann', 'email':
'maxmustermann@gmail.com', 'address': 'Duisburger Strasse 100', 'password': 'abc'})
userDAO_prefab.create({'firstname': 'Emil', 'lastname': 'Mustermann', 'email':
'maxmustermann@gmail.com', 'address': 'Duisburger Strasse 100', 'password': 'abc'})
userDAO = load('UserDAO.bin', userDAO_prefab)

productDAO_prefab = ProductDAO()
productDAO_prefab.create({'name': 'Tastatur', 'brand': 'BambooKeys', 'newprice': 98.88, 'oldprice':
119.99})
```

```
productDAO_prefab.create({'name': 'Maus', 'brand': 'Logitech Gaming', 'newprice': 127.79,
'oldprice': 149.99})
productDAO = load('ProductDAO.bin', productDAO_prefab)

questionDAO_prefab = QuestionDAO()
questionDAO_prefab.create({'firstname': 'Max', 'lastname': 'Mustermann', 'email':
'musterm@mail.com', 'subject': 'Defektes Produkt', 'category': 'Retoure', 'customerid': 1,
'description': 'Mein Produkt ist leider defekt!'})
questionDAO = load('QuestionDAO.bin', questionDAO_prefab)

reviewDAO_prefab = ReviewDAO()
reviewDAO_prefab.create({'publisher': 'Max Mustermann', 'starrating': 2, 'text': 'Ging leider schnell
kaputt.'})
reviewDAO = load('ReviewDAO.bin', reviewDAO_prefab)

orderDAO_prefab = OrderDAO()
orderDAO_prefab.create({'date': '06.07.1990', 'payment': 'PayPal', 'total': 100.99})
orderDAO = load('OrderDAO.bin', orderDAO_prefab)




@nsUsers.route('/<int:id>')
@nsUsers.response(404, 'User not found')
@nsUsers.param('id', 'The user identifier')
class User(Resource):
    '''Show a single user item and lets you delete them'''
    @nsUsers.doc('get_user')
    @nsUsers.marshal_with(user)
    def get(self, id):
        '''Nutzer mit Id=<id> anzeigen'''
        return userDAO.get(id)

    @nsUsers.doc('delete_user')
    @nsUsers.response(204, 'User deleted')
    def delete(self, id):
        '''Nutzer mit Id=<id> loeschen'''
        userDAO.delete(id)
        return '', 204

    @nsUsers.doc('update_user')
    @nsUsers.expect(user)
    @nsUsers.marshal_with(user)
    def put(self, id):
        '''Nutzer mit Id=<id> aktualisieren'''
        return userDAO.update(id, api.payload)

@nsUsers.route('/')
class UserList(Resource):
    '''Shows a list of all users, and lets you POST to add new tasks'''
    @nsUsers.doc('list_users')
    @nsUsers.marshal_list_with(user)
```

```
    def get(self):
        '''Alle Nutzer auflisten'''
        return userDAO.users

    # @nsUsers.doc('create_user')
    # @nsUsers.expect(user)
    # @nsUsers.marshal_with(user, code=201)
    # def post(self):
    #     '''Nutzer anlegen'''
    #     return userDAO.create(api.payload), 201
    #     # todo 401

    @nsUsers.doc('update_user')
    @nsUsers.expect(user)
    @nsUsers.marshal_with(user, code=201)
    def put(self):
        '''Alle Nutzer aktualisieren'''
        return userDAO.updateAll(api.payload), 201

    @nsUsers.doc('delete_user')
    @nsUsers.expect(user)
    @nsUsers.marshal_with(user, code=201)
    def delete(self):
        '''Alle Nutzer loeschen'''
        return userDAO.deleteAll(), 201



@nsProducts.route('/<int:id>')
@nsProducts.response(404, 'Product not found')
@nsProducts.param('id', 'The product identifier')
class Product(Resource):
    '''Show a single product item and lets you delete them'''
    @nsProducts.doc('get_product')
    @nsProducts.marshal_with(product)
    def get(self, id):
        '''Produkt mit Id=<id> anzeigen'''
        return productDAO.get(id)

    @nsProducts.doc('delete_product')
    @nsProducts.response(204, 'Product deleted')
    def delete(self, id):
        '''Produkt mit Id=<id> loeschen'''
        productDAO.delete(id)
        return '', 204

    @nsProducts.doc('update_product')
    @nsProducts.expect(product)
    @nsProducts.marshal_with(product)
    def put(self, id):
        '''Produkt mit Id=<id> aktualisieren'''
        return productDAO.update(id, api.payload)
```

```
@nsProducts.route('/')
class ProductList(Resource):
    '''Shows a list of all products, and lets you POST to add new tasks'''
    @nsProducts.doc('list_products')
    @nsProducts.marshal_list_with(product)
    def get(self):
        '''Alle Produkte auflisten'''
        return productDAO.products

    # @nsProducts.doc('create_product')
    # @nsProducts.expect(product)
    # @nsProducts.marshal_with(product, code=201)
    # def post(self):
    #     '''Produkt anlegen'''
    #     return productDAO.create(api.payload), 201
    #     # todo 401

    @nsProducts.doc('update_product')
    @nsProducts.expect(product)
    @nsProducts.marshal_with(product, code=201)
    def put(self):
        '''Alle Produkte aktualisieren'''
        return productDAO.updateAll(api.payload), 201

    @nsProducts.doc('delete_product')
    @nsProducts.expect(product)
    @nsProducts.marshal_with(product, code=201)
    def delete(self):
        '''Alle Produkte loeschen'''
        return productDAO.deleteAll(), 201



@nsQuestions.route('/<int:id>')
@nsQuestions.response(404, 'Question not found')
@nsQuestions.param('id', 'The question identifier')
class Question(Resource):
    '''Show a single question item and lets you delete them'''
    @nsQuestions.doc('get_question')
    @nsQuestions.marshal_with(question)
    def get(self, id):
        '''Frage mit Id=<id> anzeigen'''
        return questionDAO.get(id)

    @nsQuestions.doc('delete_question')
    @nsQuestions.response(204, 'Question deleted')
    def delete(self, id):
        '''Frage mit Id=<id> loeschen'''
        questionDAO.delete(id)
        return '', 204
```

```
    @nsQuestions.doc('update_question')
    @nsQuestions.expect(question)
    @nsQuestions.marshal_with(question)
    def put(self, id):
        '''Frage mit Id=<id> aktualisieren'''
        return questionDAO.update(id, api.payload)

@nsQuestions.route('/')
class QuestionList(Resource):
    '''Shows a list of all questions, and lets you POST to add new tasks'''
    @nsQuestions.doc('list_questions')
    @nsQuestions.marshal_list_with(question)
    def get(self):
        '''Alle Fragen auflisten'''
        return questionDAO.questions

    # @nsQuestions.doc('create_question')
    # @nsQuestions.expect(question)
    # @nsQuestions.marshal_with(question, code=201)
    # def post(self):
    #     '''Frage anlegen'''
    #     return questionDAO.create(api.payload), 201
    #     # todo 401

    @nsQuestions.doc('update_question')
    @nsQuestions.expect(question)
    @nsQuestions.marshal_with(question, code=201)
    def put(self):
        '''Alle Fragen aktualisieren'''
        return questionDAO.updateAll(api.payload), 201

    @nsQuestions.doc('delete_question')
    @nsQuestions.expect(question)
    @nsQuestions.marshal_with(question, code=201)
    def delete(self):
        '''Alle Fragen loeschen'''
        return questionDAO.deleteAll(), 201




@nsReviews.route('/<int:id>')
@nsReviews.response(404, 'Review not found')
@nsReviews.param('id', 'The review identifier')
class Review(Resource):
    '''Show a single review item and lets you delete them'''
    @nsReviews.doc('get_review')
    @nsReviews.marshal_with(review)
    def get(self, id):
        '''Bewertung mit Id=<id> anzeigen'''
        return reviewDAO.get(id)
```

```
    @nsReviews.doc('delete_review')
    @nsReviews.response(204, 'Review deleted')
    def delete(self, id):
        '''Bewertung mit Id=<id> loeschen'''
        reviewDAO.delete(id)
        return '', 204

    @nsReviews.doc('update_review')
    @nsReviews.expect(review)
    @nsReviews.marshal_with(review)
    def put(self, id):
        '''Bewertung mit Id=<id> aktualisieren'''
        return reviewDAO.update(id, api.payload)

@nsReviews.route('/')
class ReviewList(Resource):
    '''Shows a list of all reviews, and lets you POST to add new tasks'''
    @nsReviews.doc('list_reviews')
    @nsReviews.marshal_list_with(review)
    def get(self):
        '''Alle Bewertungen auflisten'''
        return reviewDAO.reviews

    # @nsReviews.doc('create_review')
    # @nsReviews.expect(review)
    # @nsReviews.marshal_with(review, code=201)
    # def post(self):
    #     '''Bewertung anlegen'''
    #     return reviewDAO.create(api.payload), 201
    #     # todo 401

    @nsReviews.doc('update_review')
    @nsReviews.expect(review)
    @nsReviews.marshal_with(review, code=201)
    def put(self):
        '''Alle Bewertungen aktualisieren'''
        return reviewDAO.updateAll(api.payload), 201

    @nsReviews.doc('delete_review')
    @nsReviews.expect(review)
    @nsReviews.marshal_with(review, code=201)
    def delete(self):
        '''Alle Bewertungen loeschen'''
        return reviewDAO.deleteAll(), 201



@nsOrders.route('/<int:id>')
@nsOrders.response(404, 'Order not found')
@nsOrders.param('id', 'The order identifier')
class Order(Resource):
    '''Show a single order item and lets you delete them'''
```

```
    @nsOrders.doc('get_order')
    @nsOrders.marshal_with(order)
    def get(self, id):
        '''Bestellung mit Id=<id> anzeigen'''
        return orderDAO.get(id)

    @nsOrders.doc('delete_order')
    @nsOrders.response(204, 'Order deleted')
    def delete(self, id):
        '''Bestellung mit Id=<id> loeschen'''
        orderDAO.delete(id)
        return '', 204

    @nsOrders.doc('update_order')
    @nsOrders.expect(order)
    @nsOrders.marshal_with(order)
    def put(self, id):
        '''Bestellung mit Id=<id> aktualisieren'''
        return orderDAO.update(id, api.payload)

@nsOrders.route('/')
class OrderList(Resource):
    '''Shows a list of all orders, and lets you POST to add new tasks'''
    @nsOrders.doc('list_orders')
    @nsOrders.marshal_list_with(order)
    def get(self):
        '''Alle Bestellungen auflisten'''
        return orderDAO.orders

    # @nsOrders.doc('create_order')
    # @nsOrders.expect(order)
    # @nsOrders.marshal_with(order, code=201)
    # def post(self):
    #     '''Bestellung anlegen'''
    #     return orderDAO.create(api.payload), 201
    #     # todo 401

    @nsOrders.doc('update_order')
    @nsOrders.expect(order)
    @nsOrders.marshal_with(order, code=201)
    def put(self):
        '''Alle Bestellungen aktualisieren'''
        return orderDAO.updateAll(api.payload), 201

    @nsOrders.doc('delete_order')
    @nsOrders.expect(order)
    @nsOrders.marshal_with(order, code=201)
    def delete(self):
        '''Alle Bestellungen loeschen'''
        return orderDAO.deleteAll(), 201
```

```
if __name__ == '__main__':
    # app.run(debug=True)
    app.run (host = "0.0.0.0", port = 5000)
```