**Документация**

**К проекту**

Тема проекта: создание веб-приложения на библиотеке flask.

Проект выполнили и защитили: Сулимов Сергей и Нечай Даниил

2023/2024 учебный год

Оглавление

[Идея проекта 3](#_Toc164967447)

[Использованные библиотеки 4](#_Toc164967448)

[Классы 5](#_Toc164967449)

[Самописные библиотеки 10](#_Toc164967450)

[Итог 10](#_Toc164967451)

3

# Идея проекта

Идея проекта заключается в создании сервиса записок(заметок). На сервере храняться записки пользователя, так-же, если записка не приватная, другие пользователи смогут просматаривать её. Записки можно изменять и удалять. Пользователи могут взаимодействовать с записками с помощью api, на основе которого сделаны и сайт, и приложение.

4

# Использованные библиотеки

В нашем проекте используются:

Flask - это легковесный веб-фреймворк для языка Python, который предоставляет минимальный набор инструментов для создания веб-приложений.

Flask-login – это расширение Flask, которое позволяет легко реализовать аутентификацию пользователей в веб-приложении.

Flask-WTF – это расширение Flask, которое использует классы Python для представления веб-форм.

Flask-RESTful – расширение для flask, которое позволяет работать REST API быстро и с минимальной настройкой.

Email\_validator – позволяет легко реализовать подтверждение электронной почты пользователя.

Pyqt5 - набор расширений(биндингов) графического фреймворка Qt для языка программирования Python, выполненный в виде расширения Python.

Pyqtdarktheme – библиотека, позволяющая лего сделать темную тему и более красивое оформление для приложения на Pyqt5.

Requests – это модуль языка Python, который используют для упрощения работы с HTTP-запросами.

SQLAlchemy – это Python библиотека, которая позволяет работать с реляционными базами данных с помощью ORM.

SQLAlchemy-serializer – помогает оптимизировать код SQLAlchemy

WTForms – это неплохой инструмент для создания веб-форм в приложениях Flask на языке python.

5

# Классы

Сразу разберёмся с созданием, изменением и удалением записок. Классы находяться в в файле по пути api/api\_resource/note.py

1. Классы NoteResource(ресурс для записок с параметрами) и NoteListResource(для заметок без параметров) за всё вышеперечисленное, для идентификации пользователя используется электронная почта и пароль пользователя.

class NoteResource(Resource): # ресурс для заметки с параметрами  
 def get(self, note\_id): # отправляет заметку если она не приватная по айди  
 abort\_if\_note\_not\_found(note\_id)  
 session = db\_session.create\_session()  
 note = session.query(Note).filter(Note.private == 0, Note.id == note\_id).first()  
 if note:  
 \_dict = note.to\_dict()  
 \_dict.update({"code": OK})  
 session.close()  
 return jsonify(\_dict) # заметка успешно найдена и не приватна  
 session.close()  
 return jsonify(  
 {"message": f"note {note\_id} not found", "code": NOTFOUND}) # заметка или не найдена - или приватна  
  
 def delete(self, note\_id): # удаляет заметку  
 abort\_if\_note\_not\_found(note\_id)  
 session = db\_session.create\_session()  
 args = parser2.parse\_args()  
 user = session.query(User).filter(User.email == args["email"], User.password == args["password"]).first()  
 if not user:  
 session.close()  
 return jsonify({"message": "email or password - wrong", "code": WRONG\_PASSWORD\_EMAIL})  
 return delete\_note(args["id"], user, session)  
  
 def put(self, note\_id): # изменяет заметку  
 abort\_if\_note\_not\_found(note\_id)  
 session = db\_session.create\_session()  
 args = parser.parse\_args()  
 user = session.query(User).filter(User.email == args["email"], User.password == args["password"]).first()  
 if not user:  
 session.close()  
 return jsonify({"message": "email or password - wrong", "code": WRONG\_PASSWORD\_EMAIL})  
 message = change\_note(content=args["content"], private=args["private"], user\_notes=user.notes, note\_id=note\_id,  
 session=session)  
 return message  
  
  
class NoteListResource(Resource): # ресурс для заметок без параметров  
 def get(self): # отправляет все заметки которые не приватные  
 session = db\_session.create\_session()  
 notes = session.query(Note).filter(Note.private == 0).all()  
 return jsonify({"notes": [item.to\_dict() for item in notes], "code": OK})  
  
 def post(self): # создаёт заметку  
 args = parser.parse\_args()  
 session = db\_session.create\_session()  
 user = session.query(User).filter(User.email == args["email"], User.password == args["password"]).first()  
 if not user:  
 session.close()  
 return jsonify({"message": "email or password - wrong", "code": WRONG\_PASSWORD\_EMAIL})  
 \_id = create\_note(args["content"], args["private"], user.id, session)  
 return jsonify({"id": \_id, "code": OK})

В том же файле находится класс аналогичный двум предыдущим, NoteResourceToken, но для индентификации пользователя используется токен – он каждые 24 часа генерируется на сайте индивидульно для кажого пользователя. этот класс используется в навыке для алисы и сделан для повышения уровня безопасности. Функции взаимодействия с токеном тоже вынесены в отдельный файл по пути api/api\_resource/token.py

def check\_token(auth\_token):  
 now = datetime.datetime.now()  
 for key, value in cache.items():  
 if value[1] == auth\_token:  
 if (value[0] - now).days >= 1:  
 cache[key] = (now, generate\_auth\_token(DEFAULT\_COUNT))  
 return False  
 else:  
 return key  
 return False  
  
  
def generate\_auth\_token(count):  
 token = ''.join(random.choices(string.ascii\_letters, k=count))  
 while token in [value[0] for value in cache.values()]:  
 token = ''.join(random.choices(string.ascii\_letters, k=count))  
 return token  
  
  
def get\_token(\_id):  
 if not check\_token(cache.get(\_id, [1, 2])[1]):  
 cache[\_id] = (datetime.datetime.now(), generate\_auth\_token(DEFAULT\_COUNT))  
 return cache[\_id][1]  
 return cache[\_id][1]

7

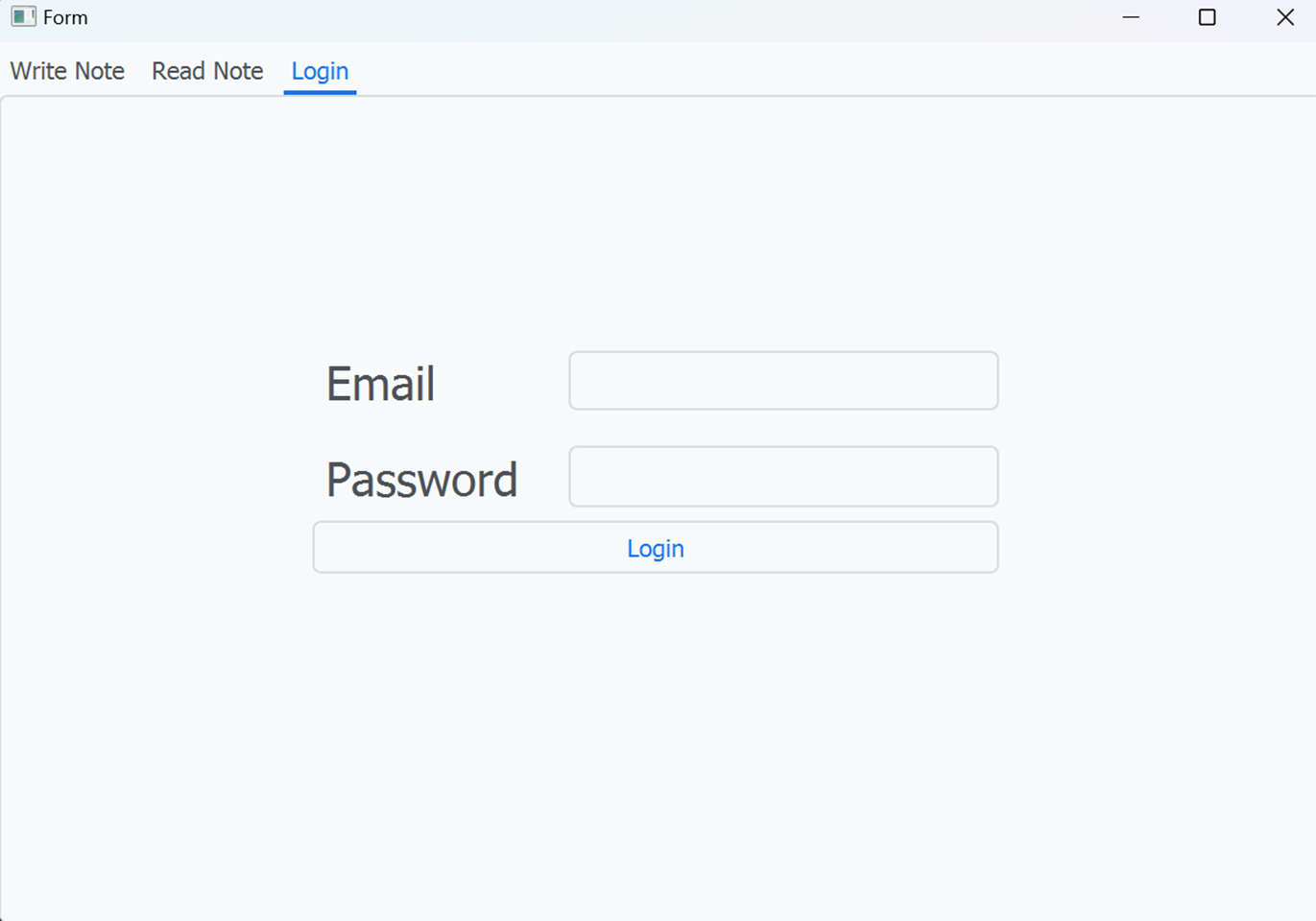
Функции создания, удаления и изменения записки(взаимодействие с базой данных) вынесены в отдельный файл по пути

def create\_note(content, private, id\_user, session):  
 note = Note(content=content, private=private, user\_id=id\_user)  
 session.add(note)  
 session.commit()  
 \_id = note.id  
 session.close()  
 return \_id  
  
  
def change\_note(content, private, user\_notes, note\_id, session):  
 \_id = note\_id  
 for note in user\_notes:  
 if note.id == note\_id:  
 note.content = content  
 note.private = private  
 session.commit()  
 session.close()  
 return jsonify({"message": f"note {\_id} change", "code": OK})  
 session.close()  
 return jsonify({"message": f"note {note\_id} not found", "code": NOTFOUND})  
  
  
def delete\_note(note\_id, user, session):  
 for note in user.notes:  
 if note.id == note\_id:  
 session.delete(note)  
 session.commit()  
 session.close()  
 return jsonify({"message": f"note {note\_id} deleted", "code": OK})  
 session.close()  
 return jsonify({"message": f"note {note\_id} not found", "code": NOTFOUND})

1. Классы UserResource и UserNoParamResource и UserNameResource отвечают за создание пользователя и получение данных о нём.

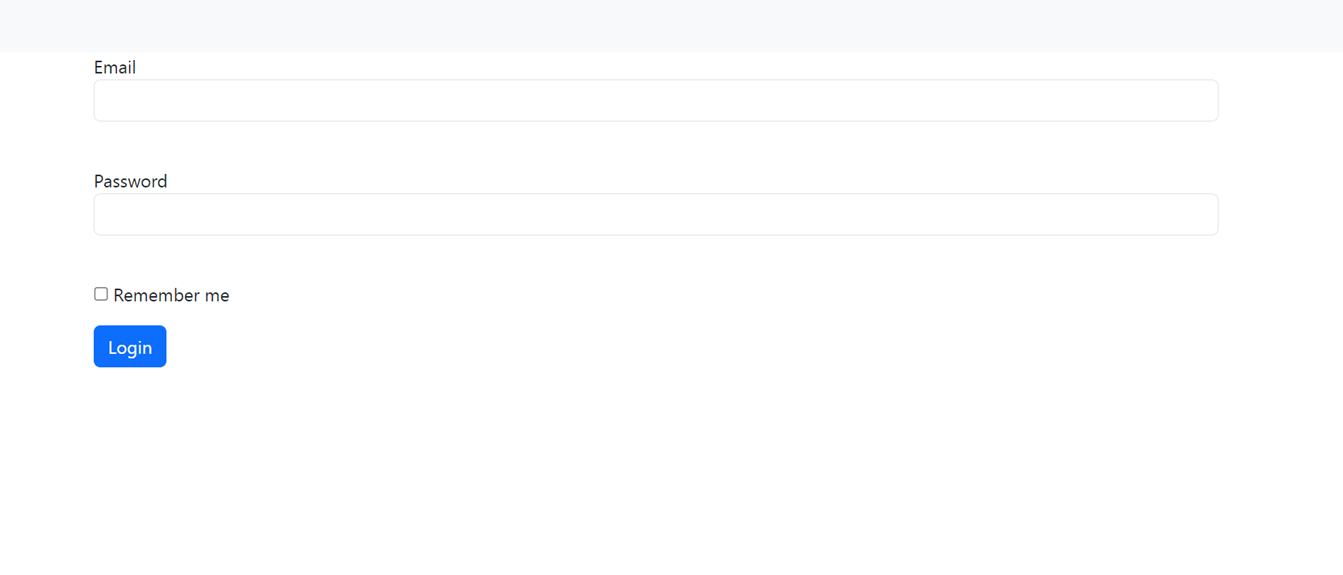
class UserResource(Resource): # ресурс для юзера с параметрами  
 def get(self, email\_user): # возвращает все не приватные заметки пользователя  
 abort\_if\_user\_not\_found(email\_user)  
 session = db\_session.create\_session()  
 user = session.query(User).filter(User.email == email\_user).first()  
 notes = []  
 for note in user.notes:  
 if not note.private:  
 notes.append(note)  
 session.close()  
 return jsonify({"notes": [note.to\_dict() for note in notes], "code": OK})  
  
  
class UserNoParamResource(Resource):  
 def post(self): # создаёт пользователя  
 db\_session.global\_init("db.db")  
 args = parser.parse\_args()  
 session = db\_session.create\_session()  
 user = session.query(User).filter(User.email == args["email"]).first()  
 if user:  
 session.close()  
 return jsonify({"message": "email is already used", "code": NAME\_TAKEN})  
  
 if args["notes"] is None:  
 args["notes"] = []  
  
 user = User(name=args["username"], password=args["password"],  
 notes=[Note(content=note['content'], private=note["private"]) for note in args["notes"]],  
 email=args["email"])  
 session.add(user)  
 session.commit()  
 \_id = user.id  
 session.close()  
  
 token = cache[\_id] = (datetime.datetime.now(), generate\_auth\_token(DEFAULT\_COUNT))  
  
 return jsonify({"id": \_id, "code": OK, "message": "user created", "auth-token": token[1]})  
  
 def get(self): # возвращает всю информацию о пользователе  
 db\_session.global\_init("db.db")  
 args = parser.parse\_args()  
 if args["notes"] is not None:  
 return jsonify({"message": "bad request", "code": BAD\_REQUEST})  
  
 session = db\_session.create\_session()  
 user = session.query(User).filter(User.email == args["email"], User.password == args["password"]).first()  
 if user is None:  
 session.close()  
 return jsonify({"message": "email or password - wrong", "code": WRONG\_PASSWORD\_EMAIL})  
  
 res = {"notes": [note.to\_dict() for note in user.notes], "code": OK, "user\_id": user.id,  
 "username": user.name, "auth-token": get\_token(user.id), "email": user.email}  
 session.close()  
 return res  
  
  
class UserNameResource(Resource):  
 def get(self, user\_id): # возвращает email пользователя  
 db\_session.global\_init("db.db")  
 session = db\_session.create\_session()  
 user = session.query(User).filter(User.id == user\_id).first()  
 if not user:  
 session.close()  
 return jsonify({"message": "user not found", "code": NOTFOUND})  
 username = user.name  
 email = user.email  
 session.close()  
 return jsonify({"username": username, "email": email, "code": OK, "id": user\_id})

1. Папка db\_session, расположенная в папке api содержит в себе классы и функции для взаимодейтсвия с базой данных.
2. В папке desktop храниться приложение аналогтчное сайту написанное на pyqt5. С помощью него можно взаимодействовать с веб приложением.



1. В папке web-site находяться файлы для работы сайта, css-стили, html страницы. На сайте можно зарегестрироваться, авторизироваться, посмотреть токен для работы с алисой, посмотреть чужие не приватные записки, свои записки,

10

создать, изменить, удалить свои записки.

# Самописные библиотеки

Также в проекте были задействованы самописные библиотеки. Их всего 3: catflask, api, alice\_module.

Catflask самый маленький модуль, он используя сторонний api вместо ошибок выдаёт смешные изображение котов.

Api – модуль для взаимодействия с api, он немного варьируется от приложение к приложение.

Alice\_module – модуль для удобной работы с json ответами и запросами от яндекс Алисы.

# Итог

Мы считаем нам удалось реализовать все поставленные задачи.

Проект можно развивать дальше следующими способами:

1. Добавление новых микросервисов или функций к старым(изменение пароля)
2. Создание новых приложений
3. Исправлений багов