Flask основы

pip install flask

Что такое flask вообще?

И почему именно он?

Можно деплоить

```python
from flask import Flask
app = Flask(__name__)

@app.route('/hello')
def hello():
    return 'Hello World'


@app.route('/projects/')
def projects():
    return 'The project page'


@app.route('/about')
def about():
    return 'The about page'

if __name__ == '__main__':
    app.run(host="127.0.0.1", port="5000")
```

Гоняем данные

json, переменные части

application.py

```
from flask import request, jsonify

posts = {
    1: {
        'title': 'Cool post'
    },
    2: {
        'title': 'Another cool post'
    }
}

@app.route('/posts/', methods=['GET', 'POST'])
def posts_list():
    if request.method == 'GET':
        return jsonify(posts)

    if request.method == 'POST':
        new_post = request.get_json()
        posts[3] = new_post
        return jsonify(posts)

@app.route('/posts/<int:post_id>', methods=['PUT', 'DELETE'])
def edit_post(id):
    if request.method == 'PUT':

        edited_post = request.get_json()
        post[post_id] = edited_post
        return jsonify(edited_post)

if request.method == 'DELETE':
    del posts[post_id]
    return jsonify(posts)
```

Ближе к делу

Но сначала пару слов про декораторы

```python
def logger(f):
    def wrapper(*args, **kwargs):
        print('func started')
        f(*args, **kwargs)
        print('func stopped')
    return wrapper

@logger
def add(a, b):
  print(a + b)

add(5, 2)
```

Защита роутов

```python
def auth_required(f):
    @wraps(f)
    def _verify(*args, **kwargs):
        auth_headers = request.headers.get('Authorization', '').split()

        invalid_msg = {
            'message': 'Пользователь не красавчик',
            'autheticated': False
        }

        if len(auth_headers) == 2:
            token = auth_headers[1]
            if token == 'Красавчик':
                return f(*args, **kwargs)

        return jsonify(invalid_msg), 401

    return _verify
```

Теперь ваше приложение под защитой (хоть какой-то)

```
@app.route('/posts/<int:post_id>', methods=['PUT', 'DELETE'])
@auth_required
def edit_post(id):
    if request.method == 'PUT':
        edited_post = request.get_json()
        post[post_id] = edited_post
        return jsonify(edited_post), 200

    if request.method == 'DELETE':
        del posts[post_id]
        return jsonify(posts), 200
```

База данных

models.py

```python
from flask_sqlalchemy import SQLAlchemy
from werkzeug.security import generate_password_hash, check_password_hash

db = SQLAlchemy()


class User(db.Model):
    __tablename__ = 'users'

    id = db.Column(db.Integer, primary_key=True)
    email = db.Column(db.String(120), unique=True, nullable=False)
    password = db.Column(db.String(255), nullable=False)
    posts = db.relationship('Post', backref='user', lazy=False)

    def __init__(self, email, password):
        self.email = email
        self.password = generate_password_hash(password, method='sha256')

    @classmethod
    def authenticate(cls, **kwargs):
        email = kwargs.get('email')
        password = kwargs.get('password')

        if not email or not password:
            return None

        user = cls.query.filter_by(email=email).first()
        if not user or not check_password_hash(user.password, password):
            return None

        return user

    def to_dict(self):
        return dict(id=self.id,
                    email=self.email,
                    posts=[post.to_dict() for post in self.posts]
                    )


class Post(db.Model):
    __tablename__ = 'posts'

    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(100), nullable=False)
    body = db.Column(db.String(300), default="")
    author = db.Column(db.Integer, db.ForeignKey('users.id'))
    comments = db.relationship('Comment', backref='post', lazy=False)

    def to_dict(self):
        return dict(
            id=self.id,
            title=self.title,
            body=self.body,
            author=self.author,
            comments=[comment.to_dict() for comment in self.comments]
        )


class Comment(db.Model):

    __tablename__ = 'comments'

    id = db.Column(db.Integer, primary_key=True)
    body = body = db.Column(db.String(300), unique=True)
    post_id = db.Column(db.Integer, db.ForeignKey('posts.id'))

    def to_dict(self):
        return dict(
            id=self.id,
            body=self.body,
            post_id=self.post_id
        )
```

Наводим красоту в application.py

```
from models import db, User, Post, Comment
def create_app():
    app = Flask(__name__)

    app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///flask_basics.db'
    app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

    db.init_app(app)

    return app
```

manage.py

```python
from flask_script import Manager
from flask_migrate import Migrate, MigrateCommand

from application import create_app
from models import db, User, Post, Comment

app = create_app()

migrate = Migrate(app, db)
manager = Manager(app)

manager.add_command('db', MigrateCommand)

@manager.shell
def shell_ctx():
    return dict(app=app,
                db=db,
                Post=Post,
                Comment=Comment,
                User=User)


if __name__ == "__main__":
    manager.run()
```

Управлять БД через интерпретатор

```
python manage.py shell
```

Соединим модули

```python
from flask import request, jsonify
from flask import Flask
from models import db, User, Post, Comment
from functools import wraps


def create_app():
    app = Flask(__name__)

    app.config['DEBUG'] = True
    app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///flask_basics.db'
    app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

    db.init_app(app)

    return app


app = create_app()


def auth_required(f):
    @wraps(f)
    def _verify(*args, **kwargs):
        auth_headers = request.headers.get('Authorization', '').split()

        invalid_msg = {
            'message': 'Пользователь не красавчик',
            'authenticated': False
        }

        if len(auth_headers) == 2:
            token = auth_headers[1]
            if token == 'Красавчик':
                return f(*args, **kwargs)

        return jsonify(invalid_msg), 401

    return _verify


@app.route('/posts/', methods=['GET'])
@auth_required
def get_posts():
    posts = [post.to_dict() for post in Post.query.all()]
    return jsonify(posts), 200


@app.route('/posts/', methods=['POST'])
@auth_required
def add_post():
    post_data = request.get_json()

    new_post = Post(title=post_data['title'])

    new_post.body = post_data.get('body')
    new_post.author = post_data.get('author')
    new_post.comments = []

    db.session.add(new_post)
    db.session.commit()

    return jsonify(new_post.to_dict()), 201


@app.route('/posts/<int:post_id>', methods=['PUT'])
@auth_required
def edit_post(post_id):
    post_data = request.get_json()
    post_to_edit = Post.query.get(post_id)

    post_to_edit.title = post_data['title']
    post_to_edit.body = post_data.get('body')
    post_to_edit.author = post_data.get('author')

    return jsonify({"message": f"Successfully updated post: '{post_to_edit.title}' "}), 200


@app.route('/posts/<int:post_id>', methods=['DELETE'])
@auth_required
def delete_post(post_id):
    post_to_delete = Post.query.get(post_id)

    db.session.delete(post_to_delete)
    db.session.commit()

    return jsonify({"message": f"Successfully deleted post: '{post_to_delete.title}' "}), 200


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

application.py

```python
from flask import request, jsonify
from flask import Flask
from models import db, User, Post, Comment
from functools import wraps


def create_app():
    app = Flask(__name__)

    app.config['DEBUG'] = True
    app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///flask_basics.db'
    app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

    db.init_app(app)

    return app


app = create_app()


def auth_required(f):
    @wraps(f)
    def _verify(*args, **kwargs):
        auth_headers = request.headers.get('Authorization', '').split()

        invalid_msg = {
            'message': 'Пользователь не красавчик',
            'authenticated': False
        }

        if len(auth_headers) == 2:
            token = auth_headers[1]
            if token == 'Красавчик':
                return f(*args, **kwargs)

        return jsonify(invalid_msg), 401

    return _verify


@app.route('/posts/', methods=['GET'])
@auth_required
def get_posts():
    posts = [post.to_dict() for post in Post.query.all()]
    return jsonify(posts), 200


@app.route('/posts/', methods=['POST'])
@auth_required
def add_post():
    post_data = request.get_json()

    new_post = Post(title=post_data['title'])

    new_post.body = post_data.get('body')
    new_post.author = post_data.get('author')
    new_post.comments = []

    db.session.add(new_post)
    db.session.commit()

    return jsonify(new_post.to_dict()), 201


@app.route('/posts/<int:post_id>', methods=['PUT'])
@auth_required
def edit_post(post_id):
    post_data = request.get_json()
    post_to_edit = Post.query.get(post_id)

    post_to_edit.title = post_data['title']
    post_to_edit.body = post_data.get('body')

    post_to_edit.author = post_data.get('author')

    return jsonify({"message": f"Successfully updated post: '{post_to_edit.title}' "}), 200


@app.route('/posts/<int:post_id>', methods=['DELETE'])
@auth_required
def delete_post(post_id):
    post_to_delete = Post.query.get(post_id)

    db.session.delete(post_to_delete)
    db.session.commit()

    return jsonify({"message": f"Successfully deleted post: '{post_to_delete.title}' "}), 200


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

models.py

```python
from flask_sqlalchemy import SQLAlchemy
from werkzeug.security import generate_password_hash, check_password_hash

db = SQLAlchemy()


class User(db.Model):
    __tablename__ = 'users'

    id = db.Column(db.Integer, primary_key=True)
    email = db.Column(db.String(120), unique=True, nullable=False)
    password = db.Column(db.String(255), nullable=False)
    posts = db.relationship('Post', backref='user', lazy=False)

    def __init__(self, email, password):
        self.email = email
        self.password = generate_password_hash(password, method='sha256')

    @classmethod
    def authenticate(cls, **kwargs):
        email = kwargs.get('email')
        password = kwargs.get('password')

        if not email or not password:
            return None

        user = cls.query.filter_by(email=email).first()
        if not user or not check_password_hash(user.password, password):
            return None

        return user

    def to_dict(self):
        return dict(id=self.id,
                    email=self.email,
                    posts=[post.to_dict() for post in self.posts]
                    )


class Post(db.Model):
    __tablename__ = 'posts'

    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(100), nullable=False)
    body = db.Column(db.String(300), default="")
    author = db.Column(db.Integer, db.ForeignKey('users.id'))
    comments = db.relationship('Comment', backref='post', lazy=False)

    def to_dict(self):
        return dict(
            id=self.id,
            title=self.title,
            body=self.body,
            author=self.author,
            comments=[comment.to_dict() for comment in self.comments]
        )


class Comment(db.Model):
    __tablename__ = 'comments'

    id = db.Column(db.Integer, primary_key=True)
    body = body = db.Column(db.String(300), unique=True)
    post_id = db.Column(db.Integer, db.ForeignKey('posts.id'))

    def to_dict(self):
        return dict(
            id=self.id,
            body=self.body,
            post_id=self.post_id
        )
```

manage.py

```python
from flask_script import Manager
from flask_migrate import Migrate, MigrateCommand

from application import create_app
from models import db, User, Post, Comment

app = create_app()

migrate = Migrate(app, db)
manager = Manager(app)

manager.add_command('db', MigrateCommand)

@manager.shell
def shell_ctx():
    return dict(app=app,
                db=db,
                Post=Post,
                Comment=Comment,
                User=User)


if __name__ == "__main__":
    manager.run()
```

3 скрипта, 187 строк кода

das Finale

https://github.com/kuderr/workshops