

# Код-ревью (из 262 ревью) от 01.12.2024 08:04

Файл: flask-realworld-example-app-master/conduit/database.py

Строка 19

- **Комментарий:**

Использование символа обратного слэша для переноса строки считается устаревшим. Рекомендуется использовать круглые скобки для переноса строки.

- **Текущий код:**

```
# https://speakerdeck.com/zzzeek/building-the-app
class SurrogatePK(object):
    """A mixin that adds a surrogate integer 'primary key' column named ``id`` \
        to any declarative-mapped class.
    """
```

---

Строка 32

- **Комментарий:**

Использование basestring несовместимо с Python 3. Необходимо использовать isinstance(record\_id, str) вместо isinstance(record\_id, basestring).

- **Текущий код:**

```
"""Get record by ID."""
if any(
    (isinstance(record_id, basestring) and record_id.isdigit(),
     isinstance(record_id, (int, float))),
```

---

Строка 34

- **Комментарий:**

Проверка на float здесь избыточна, так как primary key обычно целочисленный. Рекомендуется проверять только на int.

- **Текущий код:**

```
(isinstance(record_id, basestring) and record_id.isdigit(),
    isinstance(record_id, (int, float))),
):
    return cls.query.get(int(record_id))
```

---

Строка 37

- **Комментарий:**

Метод get возвращает None, если запись не найдена. Рекомендуется добавить обработку этого случая, если это поведение ожидаемо.

- **Текущий код:**

```
):
    return cls.query.get(int(record_id))
```

---

Строка 40

- **Комментарий:**

Функция reference\_col может быть улучшена с использованием f-строк для форматирования строки, что сделает код более читаемым и современным. Рекомендуется заменить '{0}.{1}'.format(tablename, pk\_name) на f'{tablename}.{pk\_name}'.

- **Текущий код:**

```
def reference_col(tablename, nullable=False, pk_name='id', **kwargs):
    """Column that adds primary key foreign key reference.
```

---

**Файл:** flask-realworld-example-app-master/conduit/extensions.py

Строка 23

- **Комментарий:**

Использование `cls(**kwargs)` может привести к ошибкам, если переданы неверные аргументы. Рассмотрите возможность валидации входных данных.

- **Текущий код:**

```
def create(cls, **kwargs):  
    """Create a new record and save it the database."""  
    instance = cls(**kwargs)  
    return instance.save()
```

---

Строка 31

- **Комментарий:**

Использование конструкции `commit and self.save()` or `self` является нечитаемым. Рекомендуется использовать явное условие `if commit: self.save()`

- **Текущий код:**

```
for attr, value in kwargs.items():  
    setattr(self, attr, value)  
    return commit and self.save() or self  
  
def save(self, commit=True):
```

---

Строка 44

- **Комментарий:**

Метод `delete` возвращает результат `db.session.commit()`, который всегда `None`. Рекомендуется возвращать `self` или `None` в зависимости от `commit`.

- **Текущий код:**

```
"""Remove the record from the database."""  
db.session.delete(self)  
return commit and db.session.commit()
```

---

## Файл: flask-realworld-example-app-master/conduit/utils.py

Строка 10

- **Комментарий:**

Функция не использует аргумент payload. Возможно, следует использовать payload['identity'] или аналогичный ключ для получения id пользователя.

- **Текущий код:**

```
from conduit.user.models import User # noqa

def jwt_identity(payload):
    return User.get_by_id(payload)
```

---

Строка 13

- **Комментарий:**

Метод get\_by\_id может вызвать исключение, если пользователь не найден. Рекомендуется добавить обработку исключений или использовать метод, который возвращает None при отсутствии пользователя.

- **Текущий код:**

```
def jwt_identity(payload):
    return User.get_by_id(payload)
```

---

Строка 16

- **Комментарий:**

Функция identity\_loader принимает аргумент user, но не проверяет его тип или наличие атрибута id. Рекомендуется добавить проверку, чтобы избежать возможных ошибок при обращении к атрибуту.

- **Текущий код:**

```
def identity_loader(user):  
    return user.id
```

---

## Файл: flask-realworld-example-app-master/conduit/settings.py

Строка 14

- **Комментарий:**

Использование 'secret-key' в качестве значения по умолчанию для SECRET\_KEY небезопасно. Рекомендуется использовать более сложный и уникальный ключ, например, сгенерированный с помощью os.urandom(24).

- **Текущий код:**

```
"""Base configuration."""  
  
SECRET_KEY = os.environ.get('CONDUIT_SECRET', 'secret-key') # TODO: Change me  
APP_DIR = os.path.abspath(os.path.dirname(__file__)) # This directory  
PROJECT_ROOT = os.path.abspath(os.path.join(APP_DIR, os.pardir))
```

---

Строка 24

- **Комментарий:**

Использование CORS\_ORIGIN\_WHITELIST устарело в последних версиях Flask-CORS. Рекомендуется использовать CORS\_ORIGINS.

- **Текущий код:**

```
JWT_AUTH_USERNAME_KEY = 'email'  
JWT_AUTH_HEADER_PREFIX = 'Token'  
CORS_ORIGIN_WHITELIST = [  
    'http://0.0.0.0:4100',  
    'http://localhost:4100',
```

---

Строка 35

- **Комментарий:**

JWT\_HEADER\_TYPE дублирует JWT\_AUTH\_HEADER\_PREFIX. Рекомендуется удалить одну из этих переменных для избежания путаницы.

- **Текущий код:**

```
'http://localhost:4000',  
]  
JWT_HEADER_TYPE = 'Token'
```

---

Строка 44

- **Комментарий:**

Использование os.environ.get без импорта модуля os. Рекомендуется добавить import os в начале файла.

- **Текущий код:**

```
ENV = 'prod'  
DEBUG = False  
SQLALCHEMY_DATABASE_URI = os.environ.get('DATABASE_URL',  
                                         'postgresql://localhost/example')
```

---

Строка 56

- **Комментарий:**

Использование os.path.join без импорта модуля os. Рекомендуется добавить import os в начале файла.

- **Текущий код:**

```
DB_NAME = 'dev.db'  
# Put the db file in project root  
DB_PATH = os.path.join(Config.PROJECT_ROOT, DB_NAME)  
SQLALCHEMY_DATABASE_URI = 'sqlite:///{}'.format(DB_PATH)  
CACHE_TYPE = 'simple' # Can be "memcached", "redis", etc.
```

---

Строка 60

- **Комментарий:**

Использование timedelta с большим числом может привести к переполнению. Рассмотрите использование более понятных значений, например, timedelta(days=100000), для указания срока действия токена.

- **Текущий код:**

```
SQLALCHEMY_DATABASE_URI = 'sqlite:///{}'.format(DB_PATH)
CACHE_TYPE = 'simple' # Can be "memcached", "redis", etc.
JWT_ACCESS_TOKEN_EXPIRES = timedelta(10 ** 6)
```

---

Строка 63

- **Комментарий:**

Класс TestConfig наследуется от Config, но это не указано в документации. Рассмотрите возможность добавления комментария о том, что этот класс расширяет базовую конфигурацию.

- **Текущий код:**

```
class TestConfig(Config):
    """Test configuration."""
```

---

Строка 70

- **Комментарий:**

Использование 'sqlite:/' для тестов может быть неэффективным при больших наборах тестов или сложных сценариях. Рассмотрите возможность использования в памяти базы данных или другого более быстрого решения для тестирования.

- **Текущий код:**

```
TESTING = True
DEBUG = True
```

```
SQLALCHEMY_DATABASE_URI = 'sqlite://'  
# For faster tests; needs at least 4 to avoid "ValueError: Invalid rounds"
```

---

Строка 72

- **Комментарий:**

Значение BCrypt\_LOG\_ROUNDS установлено на 4, что может быть слишком мало для реальных сценариев безопасности, но достаточно для тестов. Убедитесь, что это значение не используется в продакшене.

- **Текущий код:**

```
SQLALCHEMY_DATABASE_URI = 'sqlite://'  
# For faster tests; needs at least 4 to avoid "ValueError: Invalid rounds"  
BCRYPT_LOG_ROUNDS = 4
```

---

## Файл: flask-realworld-example-app-master/conduit/app.py

Строка 20

- **Комментарий:**

Использование `__name__.split('.')[0]` для получения имени приложения может привести к ошибкам в некоторых случаях. Рекомендуется использовать `__name__` напрямую.

- **Текущий код:**

```
:param config_object: The configuration object to use.  
"""  
app = Flask(__name__.split('.')[0])  
app.url_map.strict_slashes = False
```

---

Строка 22

- **Комментарий:**



Отключение строгого соответствия слешей может привести к проблемам с SEO и консистентностью URL. Убедитесь, что это необходимо для вашего приложения.

- **Текущий код:**

```
app = Flask(__name__.split('.')[0])
app.url_map.strict_slashes = False
app.config.from_object(config_object)
register_extensions(app)
```

---

Строка 33

- **Комментарий:**

Отсутствует отступ перед строкой документации функции. Добавьте отступ в 4 пробела.

- **Текущий код:**

```
def register_extensions(app):
    """Register Flask extensions."""
    bcrypt.init_app(app)
```

---

Строка 36

- **Комментарий:**

Использование глобальных переменных (bcrypt, cache, db, migrate, jwt) может привести к проблемам с поддержкой и тестированием. Рассмотрите возможность передачи этих расширений как аргументов функции или инициализации внутри функции.

- **Текущий код:**

```
"""Register Flask extensions."""
bcrypt.init_app(app)
cache.init_app(app)
db.init_app(app)
```

---

Строка 46

- **Комментарий:**

Использование ключа 'CORS\_ORIGIN\_WHITELIST' устарело. Рекомендуется использовать 'CORS\_ORIGINS'.

- **Текущий код:**

```
def register_blueprints(app):  
    """Register Flask blueprints."""  
    origins = app.config.get('CORS_ORIGIN_WHITELIST', '*')  
    cors.init_app(user.views.blueprint, origins=origins)
```

---

Строка 48

- **Комментарий:**

Повторяющийся код для инициализации CORS для различных блюпринтов. Рассмотрите возможность использования цикла или функции для уменьшения дублирования.

- **Текущий код:**

```
origins = app.config.get('CORS_ORIGIN_WHITELIST', '*')  
cors.init_app(user.views.blueprint, origins=origins)  
cors.init_app(profile.views.blueprint, origins=origins)  
cors.init_app(articles.views.blueprint, origins=origins)
```

---

Строка 53

- **Комментарий:**

Повторяющийся код для регистрации блюпринтов. Рассмотрите возможность использования цикла или функции для уменьшения дублирования.

- **Текущий код:**

```
cors.init_app(articles.views.blueprint, origins=origins)  
  
app.register_blueprint(user.views.blueprint)  
app.register_blueprint(profile.views.blueprint)  
app.register_blueprint(articles.views.blueprint)
```

---

Строка 59

- **Комментарий:**

Пустые строки не нужны. Удалите лишние пустые строки для улучшения читаемости кода.

- **Текущий код:**

```
def register_errorhandlers(app):  
    def errorhandler(error):
```

---

Строка 63

- **Комментарий:**

Метод `to_json()` не является стандартным для исключений в Python. Убедитесь, что класс `InvalidUsage` имеет этот метод, иначе замените на другой подход, например, `jsonify({'message': error.message})`.

- **Текущий код:**

```
def errorhandler(error):  
    response = error.to_json()  
    response.status_code = error.status_code
```

---

Строка 65

- **Комментарий:**

Использование `error.status_code` предполагает, что у объекта `error` есть атрибут `status_code`. Убедитесь, что это так, иначе добавьте проверку или используйте стандартный код ошибки.

- **Текущий код:**

```
response = error.to_json()  
response.status_code = error.status_code
```

```
return response
```

---

Строка 69

- **Комментарий:**

Регистрация обработчика ошибок через `app.errorhandler(InvalidUsage)(errorhandler)` может быть менее понятной. Рассмотрите использование декоратора `@app.errorhandler(InvalidUsage)` над определением функции `errorhandler` для большей ясности.

- **Текущий код:**

```
return response

app.errorhandler(InvalidUsage)(errorhandler)
```

---

Строка 79

- **Комментарий:**

Используйте точные имена модулей для импортов внутри функции. Рассмотрите возможность импорта необходимых моделей непосредственно в начале функции или модуля.

- **Текущий код:**

```
return {
    'db': db,
    'User': user.models.User,
    'UserProfile': profile.models.UserProfile,
    'Article': articles.models.Article,
```

---

Строка 83

- **Комментарий:**

Типографическая ошибка в имени модели `Tags`. Вероятно, должно быть `Tag`.

- **Текущий код:**

```
'UserProfile': profile.models.UserProfile,  
    'Article': articles.models.Article,  
    'Tag': articles.models.Tags,  
    'Comment': articles.models.Comment,  
}
```

---

Строка 92

- **Комментарий:**

Отсутствует документирование параметров функции. Добавьте описание параметра app в docstring.

- **Текущий код:**

```
def register_commands(app):  
    """Register Click commands."""  
    app.cli.add_command(commands.test)
```

---

Строка 94

- **Комментарий:**

Используется неявный импорт commands. Рекомендуется явно импортировать необходимые команды.

- **Текущий код:**

```
"""Register Click commands."""  
app.cli.add_command(commands.test)  
app.cli.add_command(commands.lint)
```

---

Строка 96

- **Комментарий:**

Используется неявный импорт `commands`. Рекомендуется явно импортировать необходимые команды.

- **Текущий код:**

```
app.cli.add_command(commands.test)
app.cli.add_command(commands.lint)
app.cli.add_command(commands.clean)
```

---

Строка 98

- **Комментарий:**

Используется неявный импорт `commands`. Рекомендуется явно импортировать необходимые команды.

- **Текущий код:**

```
app.cli.add_command(commands.lint)
app.cli.add_command(commands.clean)
app.cli.add_command(commands.urls)
```

---

Строка 99

- **Комментарий:**

Используется неявный импорт `commands`. Рекомендуется явно импортировать необходимые команды.

- **Текущий код:**

```
app.cli.add_command(commands.clean)
app.cli.add_command(commands.urls)
```

---

**Файл: flask-realworld-example-app-master/conduit/exceptions.py**

Строка 5

- **Комментарий:**

Функция `template` не использует ни один из стандартных аргументов или параметров Flask для создания ответа. Рассмотрите использование `jsonify` для создания JSON-ответа.

- **Текущий код:**

```
from flask import jsonify

def template(data, code=500):
    return {'message': {'errors': {'body': data}}, 'status_code': code}
```

---

Строка 8

- **Комментарий:**

Структура словаря возвращаемого значения может быть улучшена для большей гибкости и понятности. Например, можно использовать более универсальную структуру, которая позволит легко добавлять дополнительные поля.

- **Текущий код:**

```
def template(data, code=500):
    return {'message': {'errors': {'body': data}}, 'status_code': code}
```

---

Строка 24

- **Комментарий:**

Вызов конструктора базового класса без передачи аргументов `message`.  
Рекомендуется использовать `super()` и передавать `message` в `Exception.__init__()`.

- **Текущий код:**

```
def __init__(self, message, status_code=None, payload=None):
    Exception.__init__(self)
    self.message = message
    if status_code is not None:
```

---

Строка 32

- **Комментарий:**

Переменная `rv` содержит только сообщение об ошибке, что может быть недостаточно для формирования полного JSON-ответа. Рекомендуется включать статус код и `payload` в JSON.

- **Текущий код:**

```
def to_json(self):  
    rv = self.message  
    return jsonify(rv)
```

---

Строка 34

- **Комментарий:**

Использование `jsonify` вне класса или модуля Flask может вызвать ошибку. Рекомендуется импортировать `jsonify` или передавать его как аргумент в метод `to_json`.

- **Текущий код:**

```
rv = self.message  
    return jsonify(rv)  
  
@classmethod
```

---

Строка 39

- **Комментарий:**

Использование глобальной переменной `USER_NOT_FOUND` без импорта или объявления. Рекомендуется передать необходимые данные напрямую или импортировать их.

- **Текущий код:**

```
@classmethod  
    def user_not_found(cls):
```



```
        return cls(**USER_NOT_FOUND)

    @classmethod
```

---

Строка 44

- **Комментарий:**

Использование глобальной переменной `USER_ALREADY_REGISTERED` без импорта или объявления. Рекомендуется передать необходимые данные напрямую или импортировать их.

- **Текущий код:**

```
    @classmethod
    def user_already_registered(cls):
        return cls(**USER_ALREADY_REGISTERED)

    @classmethod
```

---

Строка 49

- **Комментарий:**

Использование глобальной переменной `UNKNOWN_ERROR` без импорта или объявления. Рекомендуется передать необходимые данные напрямую или импортировать их.

- **Текущий код:**

```
    @classmethod
    def unknown_error(cls):
        return cls(**UNKNOWN_ERROR)

    @classmethod
```

---

Строка 54

- **Комментарий:**

Использование глобальной переменной ARTICLE\_NOT\_FOUND без импорта или объявления. Рекомендуется передать необходимые данные напрямую или импортировать их.

- **Текущий код:**

```
@classmethod
def article_not_found(cls):
    return cls(**ARTICLE_NOT_FOUND)

@classmethod
```

---

Строка 58

- **Комментарий:**

Использование глобальной переменной COMMENT\_NOT\_OWNED без импорта или объявления. Рекомендуется передать необходимые данные напрямую или импортировать их.

- **Текущий код:**

```
@classmethod
def comment_not_owned(cls):
    return cls(**COMMENT_NOT_OWNED)
```

---

## **Файл: flask-realworld-example-app-master/conduit/commands.py**

Строка 20

- **Комментарий:**

Функция test() не использует аргументы и возвращает код завершения, что может быть неудобно для использования в других местах кода. Рассмотрите возможность использования pytest как библиотеки и возврата результатов напрямую.

- **Текущий код:**

```
@click.command()
def test():
    """Run the tests."""
```

---

Строка 25

- **Комментарий:**

Использование `exit()` внутри функции может быть нежелательным, так как это прерывает выполнение всей программы. Рассмотрите возможность возврата кода завершения вместо использования `exit()`.

- **Текущий код:**

```
"""Run the tests."""
import pytest
rv = pytest.main([TEST_PATH, '--verbose'])
exit(rv)
```

---

Строка 37

- **Комментарий:**

Использование `glob` без импорта модуля. Рекомендуется добавить `import glob` в начале файла.

- **Текущий код:**

```
skip = ['requirements']
root_files = glob('*.py')
root_directories = [
    name for name in next(os.walk('.'))[1] if not name.startswith('.')]

```

---

Строка 39

- **Комментарий:**

Использование `next(os.walk('.'))[1]` для получения директорий может быть неэффективным. Рассмотрите использование `list comprehension` напрямую с `os.walk`.

- **Текущий код:**

```
root_directories = [
```

```
name for name in next(os.walk('.'))[1] if not name.startswith('.')]
files_and_directories = [
```

---

Строка 41

- **Комментарий:**

Конкатенация списков с помощью + может быть неэффективной для больших списков. Рассмотрите использование extend или list comprehension.

- **Текущий код:**

```
name for name in next(os.walk('.'))[1] if not name.startswith('.')]
files_and_directories = [
    arg for arg in root_files + root_directories if arg not in skip]
```

---

Строка 48

- **Комментарий:**

Использование format для строк можно заменить на f-строки для улучшения читаемости кода.

- **Текущий код:**

```
"""Execute a checking tool with its arguments."""
command_line = list(args) + files_and_directories
click.echo('{}: {}'.format(description, ' '.join(command_line)))
rv = call(command_line)
if rv != 0:
```

---

Строка 52

- **Комментарий:**

Использование exit в функции может быть нежелательным, так как это завершит выполнение всего приложения. Рассмотрите возможность выброса исключения или возврата кода ошибки.

- **Текущий код:**

```
rv = call(command_line)
    if rv != 0:
        exit(rv)

if fix_imports:
```

---

Строка 55

- **Комментарий:**

Логика выполнения инструментов должна быть отделена от определения функций. Рассмотрите возможность перемещения вызовов `execute_tool` ниже определения функции `lint`.

- **Текущий код:**

```
if fix_imports:
    execute_tool('Fixing import order', 'isort', '-rc')
    execute_tool('Checking code style', 'flake8')
```

---

Строка 67

- **Комментарий:**

Отсутствует импорт модуля `os`. Необходимо добавить `import os` в начале файла.

- **Текущий код:**

```
Borrowed from Flask-Script, converted to use Click.
"""
for dirpath, _, filenames in os.walk('.'):
    for filename in filenames:
        if filename.endswith('.pyc') or filename.endswith('.pyo'):
```

---

Строка 72

- **Комментарий:**

Используйте f-строки для форматирования строк. Замените 'Removing {}'.format(full\_pathname) на f'Removing {full\_pathname}'.

- **Текущий код:**

```
if filename.endswith('.pyc') or filename.endswith('.pyo'):
    full_pathname = os.path.join(dirpath, filename)
    click.echo('Removing {}'.format(full_pathname))
    os.remove(full_pathname)
```

---

Строка 74

- **Комментарий:**

Добавьте обработку исключений для os.remove, чтобы избежать ошибок при удалении файлов. Например, используйте try-except блок.

- **Текущий код:**

```
click.echo('Removing {}'.format(full_pathname))
os.remove(full_pathname)
```

---

Строка 83

- **Комментарий:**

Функция urls принимает параметр order, но не проверяет его на допустимые значения. Рекомендуется добавить проверку.

- **Текущий код:**

```
@click.option('--order', default='rule',
              help='Property on Rule to order by (default: rule)')
@click.with_appcontext
def urls(url, order):
    """Display all of the url matching routes for the project.
```

---

Строка 96

- **Комментарий:**

Использование 'localhost' в методе bind может быть непереносимым. Рассмотрите возможность использования None или передачи хоста как параметра.

- **Текущий код:**

```
try:
    rule, arguments = (
        current_app.url_map.bind('localhost')
        .match(url, return_rule=True))
    rows.append((rule.rule, rule.endpoint, arguments))
```

---

Строка 102

- **Комментарий:**

Использование форматирования строк через format устарело. Рекомендуется использовать f-строки: f'<{e}>'

- **Текущий код:**

```
column_length = 3
except (NotFound, MethodNotAllowed) as e:
    rows.append(('<{}>'.format(e), None, None))
    column_length = 1
else:
```

---

Строка 108

- **Комментарий:**

Использование getattr без проверки наличия атрибута может вызвать AttributeError. Рассмотрите возможность использования getattr с значением по умолчанию или словаря с допустимыми значениями для сортировки.

- **Текущий код:**

```
rules = sorted(
    current_app.url_map.iter_rules(),
    key=lambda rule: getattr(rule, order))
for rule in rules:
    rows.append((rule.rule, rule.endpoint, None))
```

---

Строка 117

- **Комментарий:**

Переменная `column_length` устанавливается несколько раз. Рекомендуется установить ее один раз перед использованием.

- **Текущий код:**

```
table_width = 0

if column_length >= 1:
    max_rule_length = max(len(r[0]) for r in rows)
```

---

Строка 119

- **Комментарий:**

Вычисление максимальной длины строки для каждого столбца повторяется. Рассмотрите возможность вынесения этой логики в отдельную функцию.

- **Текущий код:**

```
if column_length >= 1:
    max_rule_length = max(len(r[0]) for r in rows)
    max_rule_length = max_rule_length if max_rule_length > 4 else 4
```

---

Строка 121

- **Комментарий:**

Использование тернарного оператора для установки минимальной длины столбца может быть улучшено с помощью функции `max`: `max_rule_length = max(4, max(len(r[0]) for r in rows))`

- **Текущий код:**

```
max_rule_length = max(len(r[0]) for r in rows)
max_rule_length = max_rule_length if max_rule_length > 4 else 4
```



```
str_template += '{:' + str(max_rule_length) + '}'
```

---

Строка 123

- **Комментарий:**

Конкатенация строк через += в цикле может быть неэффективной. Рассмотрите возможность использования списка и метода join.

- **Текущий код:**

```
max_rule_length = max_rule_length if max_rule_length > 4 else 4
str_template += '{:' + str(max_rule_length) + '}'
table_width += max_rule_length
```

---

Строка 127

- **Комментарий:**

Вычисление максимальной длины строки для каждого столбца повторяется. Рассмотрите возможность вынесения этой логики в отдельную функцию.

- **Текущий код:**

```
table_width += max_rule_length

if column_length >= 2:
    max_endpoint_length = max(len(str(r[1])) for r in rows)
    max_endpoint_length = (
```

---

Строка 130

- **Комментарий:**

Использование тернарного оператора для установки минимальной длины столбца может быть улучшено с помощью функции max: max\_endpoint\_length = max(8, max(len(str(r[1])) for r in rows))

- **Текущий код:**

```
max_endpoint_length = max(len(str(r[1])) for r in rows)
max_endpoint_length = (
    max_endpoint_length if max_endpoint_length > 8 else 8)
str_template += '  {' + str(max_endpoint_length) + '}'
```

---

Строка 133

- **Комментарий:**

Конкатенация строк через += в цикле может быть неэффективной. Рассмотрите возможность использования списка и метода join.

- **Текущий код:**

```
max_endpoint_length if max_endpoint_length > 8 else 8)
str_template += '  {' + str(max_endpoint_length) + '}'
table_width += 2 + max_endpoint_length
```

---

Строка 137

- **Комментарий:**

Вычисление максимальной длины строки для каждого столбца повторяется. Рассмотрите возможность вынесения этой логики в отдельную функцию.

- **Текущий код:**

```
table_width += 2 + max_endpoint_length

if column_length >= 3:
    max_arguments_length = max(len(str(r[2])) for r in rows)
```

---

Строка 139

- **Комментарий:**

Использование тернарного оператора для установки минимальной длины столбца может быть улучшено с помощью функции max: `max_arguments_length = max(9, max(len(str(r[2])) for r in rows))`

- **Текущий код:**

```
if column_length >= 3:
    max_arguments_length = max(len(str(r[2])) for r in rows)
    max_arguments_length = (
        max_arguments_length if max_arguments_length > 9 else 9)
```

---

Строка 142

- **Комментарий:**

Конкатенация строк через += в цикле может быть неэффективной. Рассмотрите возможность использования списка и метода join.

- **Текущий код:**

```
max_arguments_length = (
    max_arguments_length if max_arguments_length > 9 else 9)
str_template += ' {' + str(max_arguments_length) + '}'
table_width += 2 + max_arguments_length
```

---

Строка 147

- **Комментарий:**

Использование форматирования строк через format устарело. Рекомендуется использовать f-строки:

```
click.echo(f'{str_template.format(*column_headers[:column_length])}')
```

- **Текущий код:**

```
table_width += 2 + max_arguments_length

click.echo(str_template.format(*column_headers[:column_length]))
click.echo('-' * table_width)
```

---

Строка 151

- **Комментарий:**

Использование форматирования строк через `format` устарело. Рекомендуется использовать f-строки: `click.echo(f'{str_template.format(*row[:column_length])}')`

- **Текущий код:**

```
click.echo('-' * table_width)

for row in rows:
    click.echo(str_template.format(*row[:column_length]))
```

---

## Файл: flask-realworld-example-app-master/tests/conftest.py

Строка 19

- **Комментарий:**

Функция `app()` не использует декоратор фикстуры `pytest`. Используйте `@pytest.fixture` для объявления фикстуры.

- **Текущий код:**

```
@pytest.yield_fixture(scope='function')
def app():
    """An application for the tests."""
```

---

Строка 28

- **Комментарий:**

Создание контекста запроса происходит вне блока `with`, что может привести к утечкам ресурсов. Переместите `ctx.push()` внутрь блока `with`.

- **Текущий код:**

```
_db.create_all()
```

```
ctx = _app.test_request_context()
ctx.push()
```

---

Строка 34

- **Комментарий:**

Удаление контекста запроса происходит после выхода из блока with, что избыточно. Удалите ctx.pop() или переместите его внутрь блока with.

- **Текущий код:**

```
yield _app

    ctx.pop()
```

---

Строка 39

- **Комментарий:**

Функция testapp не является тестом и не должна начинаться с префикса test. Переименуйте функцию для избежания путаницы с тестами pytest.

- **Текущий код:**

```
@pytest.fixture(scope='function')
def testapp(app):
    """A Webtest app."""
```

---

Строка 42

- **Комментарий:**

Отсутствует описание того, что делает функция testapp. Укажите в документации, что функция возвращает экземпляр TestApp для переданного приложения Flask.

- **Текущий код:**

```
def testapp(app):
```

```
"""A Webtest app."""  
return TestApp(app)
```

---

Строка 48

- **Комментарий:**

Функция db не использует переданный параметр app для создания базы данных. Рассмотрите возможность использования app для инициализации \_db.

- **Текущий код:**

```
@pytest.yield_fixture(scope='function')  
def db(app):  
    """A database for the tests."""
```

---

Строка 52

- **Комментарий:**

Использование глобальной переменной \_db может привести к проблемам с областью видимости и тестированием. Рассмотрите возможность передачи \_db как параметра или инициализации внутри функции.

- **Текущий код:**

```
def db(app):  
    """A database for the tests."""  
    _db.app = app  
    with app.app_context():  
        _db.create_all()
```

---

Строка 60

- **Комментарий:**

Заккрытие сессии базы данных происходит до удаления всех таблиц. Рассмотрите возможность закрытия сессии после drop\_all().

- **Текущий код:**

```
# Explicitly close DB connection
_db.session.close()
_db.drop_all()
```

---

Строка 68

- **Комментарий:**

Функция user возвращает экземпляр класса User, который не имеет смысла в данном контексте. Рекомендуется возвращать напрямую экземпляр UserFactory.

- **Текущий код:**

```
@pytest.fixture
def user(db):
    """A user for the tests."""
    class User():
        def get(self):
```

---

Строка 71

- **Комментарий:**

Метод get в данном классе избыточен, так как он не выполняет никакой логики и просто возвращает пользователя. Удалите класс и метод, возвращайте пользователя напрямую.

- **Текущий код:**

```
class User():
    def get(self):
        muser = UserFactory(password='myprecious')
```

---

Строка 73

- **Комментарий:**

Использование жестко закодированного пароля 'myprecious' не рекомендуется. Лучше использовать фикстуру или переменную окружения для хранения паролей.

- **Текущий код:**

```
def get(self):  
    muser = UserFactory(password='myprecious')  
    UserProfile(muser).save()
```

---

Строка 75

- **Комментарий:**

Создание профиля пользователя внутри фикстуры может быть избыточным, если профиль не используется в тестах. Убедитесь, что это необходимо.

- **Текущий код:**

```
muser = UserFactory(password='myprecious')  
    UserProfile(muser).save()  
    db.session.commit()
```

---

Строка 77

- **Комментарий:**

Вызов `db.session.commit()` внутри фикстуры может привести к непредсказуемому состоянию базы данных между тестами. Рассмотрите возможность использования транзакций или отката изменений после тестов.

- **Текущий код:**

```
    UserProfile(muser).save()  
    db.session.commit()  
    return muser  
return User()
```

---

**Файл:** flask-realworld-example-app-master/tests/test\_profile.py



Строка 12

- **Комментарий:**

Использование `url_for` без импорта может вызвать ошибку. Убедитесь, что `url_for` импортирован из `flask`.

- **Текущий код:**

```
def _register_user(testapp, **kwargs):  
    return testapp.post_json(url_for('user.register_user'), {  
        "user": {  
            "email": 'foo@bar.com',
```

---

Строка 15

- **Комментарий:**

Захардкоженные значения для `email`, `username` и `password` не рекомендуется. Рассмотрите возможность передачи этих параметров через `kwargs` или использования фикстур для тестирования.

- **Текущий код:**

```
"user": {  
    "email": 'foo@bar.com',  
    "username": 'foobar',  
    "password": 'myprecious'
```

---

Строка 19

- **Комментарий:**

Использование `**kwargs` без проверки может привести к непредвиденным ошибкам. Рассмотрите возможность явного указания допустимых ключевых аргументов.

- **Текущий код:**

```
"username": 'foobar',  
    "password": 'myprecious'  
}}, **kwargs)
```

---

Строка 27

- **Комментарий:**

Использование фикстуры `_register_user` без проверки результата регистрации может привести к непредсказуемым результатам. Рассмотрите возможность проверки успешности регистрации перед выполнением запроса.

- **Текущий код:**

```
def test_get_profile_not_loggedin(self, testapp):
    _register_user(testapp)
    resp = testapp.get(url_for('profiles.get_profile', username='foobar'))
    assert resp.json['profile']['email'] == 'foo@bar.com'
    assert not resp.json['profile']['following']
```

---

Строка 37

- **Комментарий:**

Использование фикстуры `user` и вызов метода `get()` может быть избыточным, если фикстура уже возвращает объект пользователя. Убедитесь, что это необходимо.

- **Текущий код:**

```
assert resp.json == USER_NOT_FOUND['message']

def test_follow_user(self, testapp, user):
    user = user.get()
    resp = _register_user(testapp)
```

---

Строка 40

- **Комментарий:**

Переменная `user` перезаписывается результатом вызова `user.get()`. Это может быть путаницей. Рассмотрите возможность использования другого имени для переменной, например, `registered_user`.

- **Текущий код:**

```
user = user.get()
    resp = _register_user(testapp)
    token = str(resp.json['user']['token'])
```

---

Строка 42

- **Комментарий:**

Приведение токена к строке через `str()` избыточно, так как он уже строка. Удалите преобразование.

- **Текущий код:**

```
resp = _register_user(testapp)
    token = str(resp.json['user']['token'])
    resp = testapp.post(url_for('profiles.follow_user', username=user.username), headers={
        'Authorization': 'Token {}'.format(token)
```

---

Строка 45

- **Комментарий:**

Использование метода `format()` для формирования строки авторизации устаревшо. Рассмотрите использование f-строк: `'Authorization': f'Token {token}'`

- **Текущий код:**

```
resp = testapp.post(url_for('profiles.follow_user', username=user.username), headers={
    'Authorization': 'Token {}'.format(token)
})
assert resp.json['profile']['following']
```

---

Строка 52

- **Комментарий:**

Тот же комментарий, что и для строки 32: использование фикстуры user и вызов метода get() может быть избыточным.

- **Текущий код:**

```
def test_unfollow_user(self, testapp, user):  
    user = user.get()  
    resp = _register_user(testapp)  
    token = str(resp.json['user']['token'])
```

---

Строка 54

- **Комментарий:**

Тот же комментарий, что и для строки 34: переменная user перезаписывается результатом вызова user.get(). Это может быть путаницей. Рассмотрите возможность использования другого имени для переменной, например, registered\_user.

- **Текущий код:**

```
resp = _register_user(testapp)  
token = str(resp.json['user']['token'])  
resp = testapp.delete(url_for('profiles.unfollow_user', username=user.username), headers={
```

---

Строка 56

- **Комментарий:**

Тот же комментарий, что и для строки 37: использование метода format() для формирования строки авторизации устаревшо. Рассмотрите использование f-строк: 'Authorization': f'Token {token}'

- **Текущий код:**

```
token = str(resp.json['user']['token'])  
resp = testapp.delete(url_for('profiles.unfollow_user', username=user.username), headers={  
    'Authorization': 'Token {}'.format(token)  
})
```

---

## Файл: flask-realworld-example-app-master/tests/factories.py

Строка 21

- **Комментарий:**

Использование глобальной переменной db может привести к проблемам с областью видимости и тестированием. Рассмотрите возможность передачи сессии через параметры или контекст.

- **Текущий код:**

```
abstract = True
sqlalchemy_session = db.session
```

---

Строка 29

- **Комментарий:**

Используйте f-строки для форматирования строк. Рекомендуется заменить 'user{0}'.format(n) на f'user{n}'.

- **Текущий код:**

```
"""User factory."""

username = Sequence(lambda n: 'user{0}'.format(n))
email = Sequence(lambda n: 'user{0}@example.com'.format(n))
```

---

Строка 31

- **Комментарий:**

Используйте f-строки для форматирования строк. Рекомендуется заменить 'user{0}@example.com'.format(n) на f'user{n}@example.com'.

- **Текущий код:**

```
username = Sequence(lambda n: 'user{0}'.format(n))
email = Sequence(lambda n: 'user{0}@example.com'.format(n))
password = PostGenerationMethodCall('set_password', 'example')
```

---

## Файл:

**flask-realworld-example-app-master/tests/test\_authentication.py**

Строка 12

- **Комментарий:**

Использование `url_for` без импорта может вызвать ошибку. Убедитесь, что `url_for` импортирован из `flask`.

- **Текущий код:**

```
def _register_user(testapp, **kwargs):  
    return testapp.post_json(url_for("user.register_user"), {  
        "user": {  
            "username": "mo",
```

---

Строка 15

- **Комментарий:**

Использование жестко закодированных значений для `username`, `email` и `password` может привести к проблемам при изменении тестовых данных. Рассмотрите возможность передачи этих значений через параметры функции или использование фикстур.

- **Текущий код:**

```
    "user": {  
        "username": "mo",  
        "email": "mo@mo.mo",  
        "password": "momo"
```

---

Строка 20

- **Комментарий:**

Передача дополнительных аргументов через `**kwargs` может быть непонятной для других разработчиков. Рассмотрите возможность явного указания ожидаемых параметров или добавления документации к функции.

- **Текущий код:**

```
"password": "momo"  
    }  
}, **kwargs)
```

---

Строка 25

- **Комментарий:**

Пустые строки между методами класса не нужны. Удалите лишние строки.

- **Текущий код:**

```
class TestAuthenticate:  
  
    def test_register_user(self, testapp):  
        resp = _register_user(testapp)
```

---

Строка 29

- **Комментарий:**

Захардкоженные значения в тестах негативно сказываются на поддерживаемости кода. Рассмотрите возможность использования фикстур или конфигурационных файлов для хранения таких данных.

- **Текущий код:**

```
def test_register_user(self, testapp):  
    resp = _register_user(testapp)  
    assert resp.json['user']['email'] == 'mo@mo.mo'  
    assert resp.json['user']['token'] != 'None'  
    assert resp.json['user']['token'] != ''
```

---

Строка 33

- **Комментарий:**

Проверка на 'None' и '' может быть объединена в одну проверку на пустое значение. Используйте `assert not resp.json['user']['token']` для проверки на пустое значение.

- **Текущий код:**

```
assert resp.json['user']['token'] != 'None'
    assert resp.json['user']['token'] != ''

def test_user_login(self, testapp):
    _register_user(testapp)
```

---

Строка 44

- **Комментарий:**

Проверка на 'None' и '' может быть объединена в одну проверку на пустое значение. Используйте `assert not resp.json['user']['token']` для проверки на пустое значение.

- **Текущий код:**

```
assert resp.json['user']['email'] == 'mo@mo.mo'
    assert resp.json['user']['token'] != 'None'
    assert resp.json['user']['token'] != ''
```

---

Строка 55

- **Комментарий:**

Проверка на равенство токенов может быть ненадежной, так как токены обычно уникальны. Рассмотрите возможность проверки на наличие токена в ответе, а не на его точное значение.

- **Текущий код:**

```
    })
    assert resp.json['user']['email'] == 'mo@mo.mo'
    assert resp.json['user']['token'] == token

def test_register_already_registered_user(self, testapp):
```



---

Строка 63

- **Комментарий:**

Захардкоженные значения в тестах негативно сказываются на поддерживаемости кода. Рассмотрите возможность использования фикстур или конфигурационных файлов для хранения таких данных.

- **Текущий код:**

```
assert resp.status_int == 422
    assert resp.json == USER_ALREADY_REGISTERED['message']

def test_update_user(self, testapp):
    resp = _register_user(testapp)
```

---

Строка 77

- **Комментарий:**

Захардкоженные значения в тестах негативно сказываются на поддерживаемости кода. Рассмотрите возможность использования фикстур или конфигурационных файлов для хранения таких данных.

- **Текущий код:**

```
'Authorization': 'Token {}'.format(token)
    })
    assert resp.json['user']['bio'] == 'I\'m a simple man'
    assert resp.json['user']['email'] == 'meh@mo.mo'
```

---

## Файл: flask-realworld-example-app-master/tests/test\_config.py

Строка 10

- **Комментарий:**

Пустая строка между определением функции и строкой документации не требуется. Удалите пустую строку.

- **Текущий код:**

```
def test_production_config():  
    """Production config."""  
    app = create_app(ProdConfig)
```

---

Строка 15

- **Комментарий:**

Отсутствует точка в конце строки документации. Добавьте точку.

- **Текущий код:**

```
app = create_app(ProdConfig)  
assert app.config['ENV'] == 'prod'  
assert not app.config['DEBUG']
```

---

Строка 17

- **Комментарий:**

Утверждение `assert not app.config['DEBUG']` может быть неочевидным для других разработчиков. Добавьте сообщение об ошибке для улучшения читаемости: `assert not app.config['DEBUG'], 'DEBUG should be False in production config'`.

- **Текущий код:**

```
assert not app.config['DEBUG']  
  
def test_dev_config():
```

---

Строка 21

- **Комментарий:**

Отсутствует пробел после двоеточия в строке документации. Добавьте пробел после двоеточия.

- **Текущий код:**

```
def test_dev_config():  
    """Development config."""  
    app = create_app(DevConfig)  
    assert app.config['ENV'] == 'dev'
```

---

Строка 24

- **Комментарий:**

Утверждение не завершено. Необходимо указать ожидаемое значение для `app.config['DEBUG']`. Например, `assert app.config['DEBUG'] is True`

- **Текущий код:**

```
app = create_app(DevConfig)  
assert app.config['ENV'] == 'dev'  
assert app.config['DEBUG']
```

---

## Файл: flask-realworld-example-app-master/tests/test\_models.py

Строка 27

- **Комментарий:**

Создание пользователя без сохранения в базе данных может привести к ошибкам. Убедитесь, что пользователь корректно сохраняется перед тестированием.

- **Текущий код:**

```
def test_get_by_id(self):  
    """Get user by ID."""  
    user = User('foo', 'foo@bar.com')  
    user.save()
```

---

Строка 38

- **Комментарий:**

Использование `bool()` для проверки наличия значения может быть неочевидным. Лучше использовать `assert user.created_at is not None`.

- **Текущий код:**

```
user = User(username='foo', email='foo@bar.com')
    user.save()
    assert bool(user.created_at)
    assert isinstance(user.created_at, dt.datetime)
```

---

Строка 46

- **Комментарий:**

Проверка на `None` может быть неэффективной, если поле `password` может быть пустым строковым значением. Лучше использовать `assert user.password == ""` или `assert user.password is None` в зависимости от ожидаемого поведения.

- **Текущий код:**

```
user = User(username='foo', email='foo@bar.com')
    user.save()
    assert user.password is None

def test_factory(self, db):
```

---

Строка 61

- **Комментарий:**

Создание пользователя через `User.create()` внутри теста может быть нежелательным, если этот метод не является фабрикой. Лучше использовать `User(username='foo', email='foo@bar.com', password='foobarbaz123')` и `user.save()`.

- **Текущий код:**

```
def test_check_password(self):
    """Check password."""
    user = User.create(username='foo', email='foo@bar.com',
                        password='foobarbaz123')
```

---

Строка 63

- **Комментарий:**

Длинная строка с параметрами метода `create()` может быть улучшена путем переноса параметров на новые строки для улучшения читаемости.

- **Текущий код:**

```
user = User.create(username='foo', email='foo@bar.com',  
                    password='foobarbaz123')  
assert user.check_password('foobarbaz123')  
assert not user.check_password('barfoobaz')
```

---

Строка 73

- **Комментарий:**

Создание пользователей и профилей в каждом тесте может быть вынесено в `setUp` метод для уменьшения дублирования кода.

- **Текущий код:**

```
class TestProfile:  
  
    def test_follow_user(self):  
        u1 = User('foo', 'foo@bar.com')  
        u1.save()
```

---

Строка 85

- **Комментарий:**

Добавьте проверку, что `p2` не следует за `p1` после выполнения `p1.follow(p2)`.

- **Текущий код:**

```
p1.follow(p2)  
assert p1.is_following(p2)
```

```
def test_unfollow_user(self):
    u1 = User('foo', 'foo@bar.com')
```

---

Строка 106

- **Комментарий:**

Метод follow вероятно не должен возвращать значение, которое можно использовать в assert. Рассмотрите возможность проверки состояния через is\_following.

- **Текущий код:**

```
p1 = UserProfile(u1)
    p1.save()
    assert not p1.follow(p1)

def test_unfollow_self(self):
```

---

Строка 113

- **Комментарий:**

Метод unfollow вероятно не должен возвращать значение, которое можно использовать в assert. Рассмотрите возможность проверки состояния через is\_following.

- **Текущий код:**

```
u1.save()
    p1 = UserProfile(u1)
    assert not p1.unfollow(p1)
```

---

Строка 122

- **Комментарий:**

Используйте фикстуры для создания пользователей и статей, чтобы избежать дублирования кода.

- **Текущий код:**

```
def test_create_article(self, user):  
    u1 = user.get()  
    article = Article(u1.profile, 'title', 'some body', description='some')  
    article.save()
```

---

Строка 124

- **Комментарий:**

Проверьте, что статья действительно сохраняется в базе данных, добавив проверку наличия статьи после сохранения.

- **Текущий код:**

```
article = Article(u1.profile, 'title', 'some body', description='some')  
article.save()  
assert article.author.user == u1
```

---

Строка 135

- **Комментарий:**

Метод favourite не возвращает ожидаемого значения. Возможно, стоит проверять изменение состояния статьи, а не возвращаемое значение метода.

- **Текущий код:**

```
article = Article(p1, 'title', 'some body', description='some')  
article.save()  
assert article.favourite(u1.profile)  
assert article.is_favourite(u1.profile)
```

---

Строка 137

- **Комментарий:**

Метод `is_favourite` не возвращает ожидаемого значения. Убедитесь, что метод возвращает булево значение.

- **Текущий код:**

```
assert article.favourite(u1.profile)
    assert article.is_favourite(u1.profile)

def test_unfavorite_an_article(self):
```

---

Строка 153

- **Комментарий:**

То же самое замечание, что и в строке 123, касается метода `favourite`.

- **Текущий код:**

```
article = Article(p1, 'title', 'some body', description='some')
article.save()
assert article.favourite(p1)
assert article.unfavourite(p1)
```

---

Строка 155

- **Комментарий:**

То же самое замечание, что и в строке 123, касается метода `unfavourite`.

- **Текущий код:**

```
assert article.favourite(p1)
    assert article.unfavourite(p1)
    assert not article.is_favourite(p1)
```

---

Строка 157

- **Комментарий:**



Проверьте, что статья действительно удалена из избранного, добавив проверку состояния статьи после unfavoriting.

- **Текущий код:**

```
assert article.unfavourite(p1)
    assert not article.is_favourite(p1)

def test_add_tag(self, user):
```

---

Строка 164

- **Комментарий:**

Используйте множественные теги для добавления сразу нескольких тегов, если такая функциональность поддерживается.

- **Текущий код:**

```
article = Article(user.profile, 'title', 'some body', description='some')
article.save()
t = Tags(tagname='python')
t1 = Tags(tagname='flask')
assert article.add_tag(t)
```

---

Строка 175

- **Комментарий:**

То же самое замечание, что и в строке 147, касается добавления тегов.

- **Текущий код:**

```
article = Article(user.profile, 'title', 'some body', description='some')
article.save()
t1 = Tags(tagname='flask')
assert article.add_tag(t1)
assert article.remove_tag(t1)
```

---

Строка 178

- **Комментарий:**

Проверьте, что тег действительно удален из статьи, добавив проверку состояния статьи после удаления тега.

- **Текущий код:**

```
assert article.add_tag(t1)
    assert article.remove_tag(t1)
    assert len(article.tagList) == 0
```

---

Строка 186

- **Комментарий:**

Пустые строки не нужны. Удалите лишние пустые строки для улучшения читаемости кода.

- **Текущий код:**

```
@pytest.mark.usefixtures('db')
class TestComment:

    def test_make_comment(self, user):
        user = user.get()
```

---

Строка 189

- **Комментарий:**

Метод `get()` вызывается для получения пользователя, но это может быть избыточным, если `user` уже является объектом пользователя. Убедитесь, что `user` передается корректно.

- **Текущий код:**

```
def test_make_comment(self, user):
    user = user.get()
    article = Article(user.profile, 'title', 'some body', description='some')
    article.save()
```

---

Строка 193

- **Комментарий:**

Текст комментария ('some body') дублируется. Рассмотрите возможность использования переменных для избежания дублирования кода.

- **Текущий код:**

```
article = Article(user.profile, 'title', 'some body', description='some')
article.save()
comment = Comment(article, user.profile, 'some body')
comment.save()
```

---

Строка 205

- **Комментарий:**

Текст комментария ('some body2') также дублируется. Используйте переменные для улучшения читаемости и поддерживаемости кода.

- **Текущий код:**

```
article.save()
comment = Comment(article, user.profile, 'some body')
comment1 = Comment(article, user.profile, 'some body2')
comment.save()
```

---

Строка 207

- **Комментарий:**

Комментарий comment сохраняется дважды. Удалите лишний вызов comment.save().

- **Текущий код:**

```
comment1 = Comment(article, user.profile, 'some body2')
comment.save()
comment1.save()
```

---

Строка 214

- **Комментарий:**

Проверка длины комментариев через `article.comments.all()` может быть неэффективной, так как `all()` загружает все комментарии в память. Рассмотрите использование `article.comments.count()` для более эффективной проверки количества комментариев.

- **Текущий код:**

```
assert comment.author == user.profile
assert comment1.article == article
assert comment1.author == user.profile
assert len(article.comments.all()) == 2
```

---

## Файл: `flask-realworld-example-app-master/tests/test_articles.py`

Строка 11

- **Комментарий:**

Удалите лишние пустые строки для улучшения читаемости кода.

- **Текущий код:**

```
class TestArticleViews:

    def test_get_articles_by_author(self, testapp, user):
        user = user.get()
```

---

Строка 14

- **Комментарий:**

Метод `get()` у объекта `user` вызывается несколько раз в разных тестах. Рассмотрите возможность использования фикстуры, которая будет возвращать уже полученного пользователя.

- **Текущий код:**

```
def test_get_articles_by_author(self, testapp, user):  
    user = user.get()  
    resp = testapp.post_json(url_for('user.login_user'), {'user': {
```

---

Строка 16

- **Комментарий:**

Используйте f-строки для форматирования строк вместо str.format() для улучшения читаемости кода.

- **Текущий код:**

```
user = user.get()  
    resp = testapp.post_json(url_for('user.login_user'), {'user': {  
        'email': user.email,  
        'password': 'myprecious'
```

---

Строка 26

- **Комментарий:**

Используйте f-строки для форматирования строк вместо str.format() для улучшения читаемости кода.

- **Текущий код:**

```
testapp.post_json(url_for('articles.make_article'), {  
    "article": {  
        "title": "How to train your dragon {}".format(_),  
        "description": "Ever wonder how?",  
        "body": "You have to believe",
```

---

Строка 58

- **Комментарий:**

Разделите длинную строку на несколько для улучшения читаемости кода.

- **Текущий код:**

```
    })

    resp = testapp.post(url_for('articles.favorite_an_article',
                                slug=resp1.json['article']['slug']),
                        headers={
```

---

Строка 78

- **Комментарий:**

Используйте f-строки для форматирования строк вместо str.format() для улучшения читаемости кода.

- **Текущий код:**

```
testapp.post_json(url_for('articles.make_article'), {
    "article": {
        "title": "How to train your dragon {}".format(_),
        "description": "Ever wonder how?",
        "body": "You have to believe",
```

---

Строка 135

- **Комментарий:**

Используйте f-строки для форматирования строк вместо datetime.now().isoformat() для улучшения читаемости кода.

- **Текущий код:**

```
resp = testapp.post_json(url_for('articles.make_comment_on_article', slug=slug), {
    "comment": {
        "createdAt": datetime.now().isoformat(),
        "body": "You have to believe",
    }
}
```

---

Строка 146

- **Комментарий:**

Метод `data` у объекта, возвращаемого `profile_schema.dump(user)`, устарел. Используйте `profile_schema.dump(user)['profile']` напрямую.

- **Текущий код:**

```
authorp = resp.json['comment']['author']
del authorp['following']
# assert profile_schema.dump(user).data['profile'] == authorp
assert profile_schema.dump(user)['profile'] == authorp
```

---

**Файл:**

**`flask-realworld-example-app-master/conduit/articles/models.py`**

Строка 27

- **Комментарий:**

Имя класса должно быть во множественном числе, так как оно представляет коллекцию тегов. Рекомендуется переименовать класс в `Tags`.

- **Текущий код:**

```
class Tags(Model):

    __tablename__ = 'tags'

    id = db.Column(db.Integer, primary_key=True)
```

---

Строка 33

- **Комментарий:**

Использование конструктора с вызовом базового класса через `db.Model.__init__` не является правильным. Используйте `super().__init__(tagname=tagname)` для вызова конструктора базового класса.

- **Текущий код:**

```
tagname = db.Column(db.String(100))

def __init__(self, tagname):
    # TODO @dataclass
    db.Model.__init__(self, tagname=tagname)
```

---

Строка 39

- **Комментарий:**

Метод `__repr__` должен возвращать строку, которая позволяет легко определить объект. Рекомендуется изменить возвращаемое значение на что-то вроде `f'Tag(tagname={self.tagname!r})'`.

- **Текущий код:**

```
def __repr__(self):
    return self.tagname
```

---

Строка 48

- **Комментарий:**

Используйте `db.Column` вместо `Column` для согласованности с другими полями модели.

- **Текущий код:**

```
id = db.Column(db.Integer, primary_key=True)
body = Column(db.Text)
createdAt = Column(db.DateTime, nullable=False, default=dt.datetime.utcnow)
updatedAt = Column(db.DateTime, nullable=False, default=dt.datetime.utcnow)
```

---

Строка 51

- **Комментарий:**

Поле `updatedAt` должно обновляться автоматически при каждом изменении записи. Рассмотрите использование `db.Column(db.DateTime, nullable=False,`



default=dt.datetime.utcnow, onupdate=dt.datetime.utcnow).

- **Текущий код:**

```
createdAt = Column(db.DateTime, nullable=False, default=dt.datetime.utcnow)
updatedAt = Column(db.DateTime, nullable=False, default=dt.datetime.utcnow)
author_id = reference_col('userprofile', nullable=False)
author = relationship('UserProfile', backref=db.backref('comments'))
```

---

Строка 57

- **Комментарий:**

Параметры article и author передаются в инициализатор, но используются как author и body. Проверьте правильность передачи аргументов.

- **Текущий код:**

```
article_id = reference_col('article', nullable=False)

def __init__(self, article, author, body, **kwargs):
    # TODO @dataclass
    db.Model.__init__(self, author=author, body=body, article=article, **kwargs)
```

---

Строка 60

- **Комментарий:**

Используйте super().\_\_init\_\_(...) вместо db.Model.\_\_init\_\_(...) для вызова конструктора базового класса.

- **Текущий код:**

```
# TODO @dataclass
db.Model.__init__(self, author=author, body=body, article=article, **kwargs)
```

---

Строка 69

- **Комментарий:**

Используйте `db.Column` вместо `Column` для согласованности с остальными полями модели.

- **Текущий код:**

```
id = db.Column(db.Integer, primary_key=True)
slug = Column(db.Text, unique=True)
title = Column(db.String(100), nullable=False)
description = Column(db.Text, nullable=False)
```

---

Строка 75

- **Комментарий:**

Поле `createdAt` должно использовать серверное время. Рассмотрите использование `server_default`.

- **Текущий код:**

```
body = Column(db.Text)
createdAt = Column(db.DateTime, nullable=False, default=dt.datetime.utcnow)
updatedAt = Column(db.DateTime, nullable=False, default=dt.datetime.utcnow)
author_id = reference_col('userprofile', nullable=False)
```

---

Строка 77

- **Комментарий:**

Поле `updatedAt` должно автоматически обновляться. Рассмотрите использование `onupdate`.

- **Текущий код:**

```
updatedAt = Column(db.DateTime, nullable=False, default=dt.datetime.utcnow)
author_id = reference_col('userprofile', nullable=False)
author = relationship('UserProfile', backref=db.backref('articles'))
favoriters = relationship(
```

---

Строка 86

- **Комментарий:**

Имя класса Tags должно быть в единственном числе Tag, так как это отношение многие-ко-многим.

- **Текущий код:**

```
lazy='dynamic')

tagList = relationship(
    'Tags', secondary=tag_assoc, backref='articles')
```

---

Строка 109

- **Комментарий:**

Метод is\_favourite выполняет запрос каждый раз. Рассмотрите использование более эффективного способа проверки наличия.

- **Текущий код:**

```
return False

def is_favourite(self, profile):

    return bool(self.query.filter(favoriter_assoc.c.favoriter == profile.id).count())
```

---

Строка 126

- **Комментарий:**

Используйте snake\_case для свойства favoritesCount согласно PEP 8.

- **Текущий код:**

```
return False

@property
def favoritesCount(self):
    return len(self.favoriters.all())
```

---

Строка 132

- **Комментарий:**

Используйте snake\_case для свойства favorited согласно PEP 8.

- **Текущий код:**

```
@property
def favorited(self):
    if current_user:
        profile = current_user.profile
```

---

Строка 137

- **Комментарий:**

Метод favorited выполняет запрос каждый раз. Рассмотрите использование более эффективного способа проверки наличия.

- **Текущий код:**

```
profile = current_user.profile

    return self.query.join(Article.favoriters).filter(UserProfile.id == profile.id).count()
return False
```

---

## Файл:

**flask-realworld-example-app-master/conduit/articles/serializers.py**

Строка 12

- **Комментарий:**

Используйте более конкретное имя для поля tagname, например, name, чтобы избежать путаницы.

- **Текущий код:**

```
class TagSchema(Schema):  
    tagname = fields.Str()
```

---

Строка 24

- **Комментарий:**

Использование 'self' в Nested может привести к проблемам с сериализацией. Рассмотрите возможность использования lambda или явного определения схемы.

- **Текущий код:**

```
updatedAt = fields.DateTime(dump_only=True)  
author = fields.Nested(ProfileSchema)  
article = fields.Nested('self', exclude=('article',), default=True, load_only=True)  
tagList = fields.List(fields.Str())  
favoritesCount = fields.Int(dump_only=True)
```

---

Строка 35

- **Комментарий:**

Метод dump\_article изменяет входные данные, что может быть нежелательно. Рассмотрите возможность создания копии данных для модификации.

- **Текущий код:**

```
@post_dump  
def dump_article(self, data, **kwargs):  
    data['author'] = data['author']['profile']  
    return {'article': data}
```

---

Строка 40

- **Комментарий:**

Параметр strict в Meta устарел в marshmallow 3 и выше. Удалите его или используйте соответствующие параметры в конструкторе схемы.

- **Текущий код:**

```
return {'article': data}
```

```
class Meta:  
    strict = True
```

---

Строка 44

- **Комментарий:**

Название класса должно быть во множественном числе, так как он расширяет функциональность для нескольких статей. Рекомендуется переименовать класс в ArticlesSchemas.

- **Текущий код:**

```
strict = True
```

```
class ArticleSchemas(ArticleSchema):
```

---

Строка 50

- **Комментарий:**

Прямое обращение к ключам словаря может вызвать KeyError, если ключ отсутствует. Рекомендуется использовать метод get() для безопасного доступа к значениям.

- **Текущий код:**

```
@post_dump  
def dump_article(self, data, **kwargs):  
    data['author'] = data['author']['profile']  
    return data
```

---

Строка 55

- **Комментарий:**

Использование параметра `many` в декораторе `post_dump` не требуется, так как метод уже обрабатывает несколько объектов. Рекомендуется убрать `pass_many=True`.

- **Текущий код:**

```
@post_dump(pass_many=True)
def dump_articles(self, data, many, **kwargs):
    return {'articles': data, 'articlesCount': len(data)}
```

---

Строка 68

- **Комментарий:**

Использование `'self'` в `Nested` может привести к проблемам с инициализацией схемы. Рекомендуется использовать `lambda: CommentSchema(exclude=('comment',))`

- **Текущий код:**

```
# for the envelope
comment = fields.Nested('self', exclude=('comment',), default=True, load_only=True)

@pre_load
```

---

Строка 73

- **Комментарий:**

Отсутствие проверки наличия ключа `'comment'` в данных может вызвать `KeyError`. Рекомендуется добавить проверку: `return data.get('comment', data)`

- **Текущий код:**

```
@pre_load
def make_comment(self, data, **kwargs):
    return data['comment']

@post_dump
```

---

Строка 78

- **Комментарий:**

Прямое обращение к вложенным полям может быть ненадежным при изменении структуры данных. Рассмотрите использование метода `get` или проверку наличия ключа

- **Текущий код:**

```
@post_dump
def dump_comment(self, data, **kwargs):
    data['author'] = data['author']['profile']
    return {'comment': data}
```

---

Строка 83

- **Комментарий:**

Параметр `strict` в `Meta` устарел в `marshmallow 3` и выше. Рекомендуется удалить эту строку, так как поведение по умолчанию уже строгое

- **Текущий код:**

```
class Meta:
    strict = True
```

---

Строка 88

- **Комментарий:**

Пустые строки. Удалите лишние пустые строки для улучшения читаемости кода.

- **Текущий код:**

```
class CommentsSchema(CommentSchema):

    @post_dump
    def dump_comment(self, data, **kwargs):
```

---

Строка 92



- **Комментарий:**

Непроверенная структура данных. Убедитесь, что ключ 'profile' существует в словаре 'author'. Рассмотрите использование метода `get()` для избежания `KeyError`.

- **Текущий код:**

```
@post_dump
def dump_comment(self, data, **kwargs):
    data['author'] = data['author']['profile']
    return data
```

---

Строка 96

- **Комментарий:**

Непонятное название метода. Название метода 'make\_comment' не отражает его функционала. Рассмотрите переименование в 'wrap\_comments' или 'format\_comments\_response'.

- **Текущий код:**

```
return data

@post_dump(pass_many=True)
def make_comment(self, data, many, **kwargs):
```

---

Строка 98

- **Комментарий:**

Использование аргумента `many`. Метод `post_dump` с аргументом `pass_many=True` может быть вызван дважды для одиночного и множественного дампа. Рассмотрите возможность объединения логики в один метод с проверкой параметра `many`.

- **Текущий код:**

```
@post_dump(pass_many=True)
def make_comment(self, data, many, **kwargs):
    return {'comments': data}
```

---

## Файл:

**flask-realworld-example-app-master/conduit/articles/views.py**

Строка 34

- **Комментарий:**

Используйте более понятные имена переменных. res -> query

- **Текущий код:**

```
def get_articles(tag=None, author=None, favorited=None, limit=20, offset=0):  
  
    res = Article.query  
    if tag:  
        res = res.filter(Article.tagList.any(Tags.tagname == tag))
```

---

Строка 37

- **Комментарий:**

Используйте явное соединение с таблицей Tags для улучшения читаемости. res = res.join(Tags, Article.tagList.contains(Tags.tagname)).filter(Tags.tagname == tag)

- **Текущий код:**

```
    if tag:  
        res = res.filter(Article.tagList.any(Tags.tagname == tag))  
    if author:  
        res = res.join(Article.author).join(User).filter(User.username == author)
```

---

Строка 40

- **Комментарий:**

Избыточное соединение с User. Удалите второе join(User). res = res.join(Article.author).filter(User.username == author)

- **Текущий код:**

```
if author:
    res = res.join(Article.author).join(User).filter(User.username == author)
    if favorited:
        res = res.join(Article.favoriters).filter(User.username == favorited)
```

---

Строка 43

- **Комментарий:**

Избыточное соединение с User. Удалите второе join(User). res = res.join(Article.favoriters).filter(User.username == favorited)

- **Текущий код:**

```
if favorited:
    res = res.join(Article.favoriters).filter(User.username == favorited)
    return res.offset(offset).limit(limit).all()
```

---

Строка 45

- **Комментарий:**

Рассмотрите возможность использования paginate вместо offset и limit для улучшения работы с пагинацией. pagination = res.offset(offset).limit(limit).paginate(page=1, per\_page=limit, error\_out=False).items

- **Текущий код:**

```
res = res.join(Article.favoriters).filter(User.username == favorited)
return res.offset(offset).limit(limit).all()
```

---

Строка 56

- **Комментарий:**

Слишком длинная строка кода. Разбейте ее на несколько строк для улучшения читаемости.

- **Текущий код:**

```
def make_article(body, title, description, tagList=None):  
  
    article = Article(title=title, description=description, body=body,  
                      author=current_user.profile)  
    if tagList is not None:
```

---

Строка 60

- **Комментарий:**

Отсутствие обработки дублирующихся тегов в списке tagList. Рассмотрите возможность использования множества для исключения дубликатов.

- **Текущий код:**

```
        author=current_user.profile)  
        if tagList is not None:  
            for tag in tagList:
```

---

Строка 68

- **Комментарий:**

Неправильное использование конструктора Tags. Должно быть Tags(tagname=tag).

- **Текущий код:**

```
        mtag = Tags.query.filter_by(tagname=tag).first()  
        if not mtag:  
            mtag = Tags(tag)  
            mtag.save()  
        article.add_tag(mtag)
```

---

Строка 81

- **Комментарий:**

Удалите лишние пустые строки для улучшения читаемости кода.

- **Текущий код:**

```
@use_kwargs(article_schema)
@marshal_with(article_schema)
def update_article(slug, **kwargs):
```

---

Строка 85

- **Комментарий:**

Используйте f-строки для формирования сообщений об ошибках, если это необходимо в контексте приложения.

- **Текущий код:**

```
article = Article.query.filter_by(slug=slug, author_id=current_user.profile.id).first()
if not article:
```

---

Строка 87

- **Комментарий:**

Проверка наличия статьи может быть улучшена с использованием оператора if article is None: для большей ясности.

- **Текущий код:**

```
article = Article.query.filter_by(slug=slug, author_id=current_user.profile.id).first()
if not article:
    raise InvalidUsage.article_not_found()
article.update(updatedAt=dt.datetime.utcnow(), **kwargs)
```

---

Строка 90

- **Комментарий:**

Метод update не является стандартным для SQLAlchemy ORM. Возможно, стоит использовать setattr для установки значений или переопределить метод update в модели Article.

- **Текущий код:**

```
raise InvalidUsage.article_not_found()
    article.update(updatedAt=dt.datetime.utcnow(), **kwargs)
    article.save()
```

---

Строка 92

- **Комментарий:**

Метод save также не является стандартным для SQLAlchemy ORM. Рассмотрите возможность использования db.session.commit() для сохранения изменений в базе данных.

- **Текущий код:**

```
article.update(updatedAt=dt.datetime.utcnow(), **kwargs)
    article.save()
    return article
```

---

Строка 100

- **Комментарий:**

Удаление пустых строк для улучшения читаемости кода.

- **Текущий код:**

```
@blueprint.route('/api/articles/<slug>', methods=('DELETE',))
@jwt_required
def delete_article(slug):
```

---

Строка 105

- **Комментарий:**

Проверка наличия статьи перед удалением. Использование метода first() может вернуть None, что приведет к ошибке при вызове article.delete(). Решение: добавить проверку на None.

- **Текущий код:**

```
article = Article.query.filter_by(slug=slug, author_id=current_user.profile.id).first()
article.delete()
```

---

Строка 107

- **Комментарий:**

Метод delete() не является стандартным для SQLAlchemy. Рекомендуется использовать session.delete(article) для удаления записи из базы данных.

- **Текущий код:**

```
article = Article.query.filter_by(slug=slug, author_id=current_user.profile.id).first()
article.delete()
return '', 200
```

---

Строка 109

- **Комментарий:**

Возвращение пустой строки в качестве ответа может быть неудобно для клиента. Рассмотрите возможность возврата JSON-ответа с сообщением об успешном удалении.

- **Текущий код:**

```
article.delete()
return '', 200
```

---

Строка 117

- **Комментарий:**

Пустые строки в начале функции. Удалите лишние пустые строки для улучшения читаемости кода.

- **Текущий код:**

```
@jwt_optional
@marshal_with(article_schema)
def get_article(slug):
```

---

Строка 124

- **Комментарий:**

Отсутствие документации для функции. Добавьте docstring для описания назначения функции и ее параметров.

- **Текущий код:**

```
article = Article.query.filter_by(slug=slug).first()
if not article:
    raise InvalidUsage.article_not_found()
```

---

Строка 127

- **Комментарий:**

Использование статического метода для создания исключения. Рассмотрите возможность использования конструктора класса или фабричного метода для создания исключения.

- **Текущий код:**

```
if not article:
    raise InvalidUsage.article_not_found()
    return article
```

---

Строка 136

- **Комментарий:**

Удалите лишние пустые строки для улучшения читаемости кода.



- **Текущий код:**

```
@jwt_required
@marshal_with(article_schema)
def favorite_an_article(slug):
```

---

Строка 144

- **Комментарий:**

Используйте метод `one_or_none()` вместо `first()` для более точного отображения намерений - либо один объект, либо ничего.

- **Текущий код:**

```
profile = current_user.profile
article = Article.query.filter_by(slug=slug).first()
if not article:
    raise InvalidUsage.article_not_found()
```

---

Строка 148

- **Комментарий:**

Метод `favourite(profile)` может быть неочевидным. Рассмотрите возможность переименования метода для большей ясности, например, `add_favorite(profile)`.

- **Текущий код:**

```
if not article:
    raise InvalidUsage.article_not_found()
article.favourite(profile)
article.save()
return article
```

---

Строка 151

- **Комментарий:**

Возвращайте сериализованный объект статьи, а не сам объект модели, чтобы клиент получил необходимые данные в нужном формате.

- **Текущий код:**

```
article.save()  
return article
```

---

Строка 160

- **Комментарий:**

Удалите лишние пустые строки для улучшения читаемости кода.

- **Текущий код:**

```
@marshal_with(article_schema)  
def unfavorite_an_article(slug):
```

---

Строка 166

- **Комментарий:**

Используйте метод `one_or_none()` вместо `first()` для более точного отображения намерений - если статья не найдена, возвращается `None`.

- **Текущий код:**

```
profile = current_user.profile  
article = Article.query.filter_by(slug=slug).first()  
if not article:  
    raise InvalidUsage.article_not_found()
```

---

Строка 170

- **Комментарий:**

Метод `unfavourite` следует переименовать в `unfavorite` для соответствия стандартам английского языка и стилю PEP 8.

- **Текущий код:**

```
if not article:
    raise InvalidUsage.article_not_found()
article.unfavourite(profile)
article.save()
return article
```

---

Строка 173

- **Комментарий:**

Возвращайте сериализованный объект статьи, а не сам объект модели, чтобы клиент получал данные в нужном формате.

- **Текущий код:**

```
article.save()
return article
```

---

Строка 181

- **Комментарий:**

Функция не использует переданные параметры limit и offset в запросе к базе данных. Убедитесь, что они корректно передаются в методы offset и limit.

- **Текущий код:**

```
@jwt_required
@use_kwargs({'limit': fields.Int(), 'offset': fields.Int()})
@marshal_with(articles_schema)
def articles_feed(limit=20, offset=0):
```

---

Строка 187

- **Комментарий:**

Использование обратного слэша для переноса строки считается устаревшим. Лучше использовать круглые скобки для переноса строк внутри выражений.

- **Текущий код:**

```
return Article.query.join(current_user.profile.follows). \
    order_by(Article.createdAt.desc()).offset(offset).limit(limit).all()
```

---

Строка 198

- **Комментарий:**

Функция `get_tags()` содержит лишние пустые строки. Удалите лишние пробелы для улучшения читаемости кода.

- **Текущий код:**

```
@blueprint.route('/api/tags', methods=('GET',))

def get_tags():
```

---

Строка 204

- **Комментарий:**

Использование `Tags.query.all()` может быть неэффективным при большом количестве тегов, так как загружает все записи в память. Рассмотрите возможность использования запроса с ограничением или агрегацией данных.

- **Текущий код:**

```
return jsonify({'tags': [tag.tagname for tag in Tags.query.all()]})
```

---

Строка 217

- **Комментарий:**

Удалите лишние пустые строки для улучшения читаемости кода.

- **Текущий код:**

```
@marshal_with(comments_schema)

def get_comments(slug):
```

---

Строка 223

- **Комментарий:**

Используйте метод `one_or_none()` вместо `first()` для более точного отображения намерений - если статья не найдена, это может быть ошибкой.

- **Текущий код:**

```
article = Article.query.filter_by(slug=slug).first()
if not article:
    raise InvalidUsage.article_not_found()
```

---

Строка 226

- **Комментарий:**

Рассмотрите возможность использования более специфического исключения или передачи дополнительной информации в `InvalidUsage.article_not_found()`.

- **Текущий код:**

```
if not article:
    raise InvalidUsage.article_not_found()
    return article.comments
```

---

Строка 236

- **Комментарий:**

Удалите лишние пустые строки для улучшения читаемости кода.

- **Текущий код:**

```
@use_kwargs(comment_schema)
@marshal_with(comment_schema)
def make_comment_on_article(slug, body, **kwargs):
```

---

Строка 245

- **Комментарий:**

Используйте явное указание аргументов при создании объекта Comment вместо `**kwargs` для повышения читаемости и предотвращения ошибок.

- **Текущий код:**

```
if not article:
    raise InvalidUsage.article_not_found()
comment = Comment(article, current_user.profile, body, **kwargs)
comment.save()
```

---

Строка 247

- **Комментарий:**

Рассмотрите возможность использования транзакций для сохранения комментария, чтобы избежать частичного сохранения данных в случае ошибок.

- **Текущий код:**

```
comment = Comment(article, current_user.profile, body, **kwargs)
comment.save()
return comment
```

---

Строка 255

- **Комментарий:**

Удалите лишние пустые строки для улучшения читаемости кода.

- **Текущий код:**

```
@blueprint.route('/api/articles/<slug>/comments/<cid>', methods=('DELETE',))  
@jwt_required  
def delete_comment_on_article(slug, cid):
```

---

Строка 265

- **Комментарий:**

Проверка наличия комментария у пользователя должна быть более явной. Рассмотрите возможность использования метода `exists()` для оптимизации запроса.

- **Текущий код:**

```
raise InvalidUsage.article_not_found()  
  
comment = article.comments.filter_by(id=cid, author=current_user.profile).first()  
comment.delete()
```

---

Строка 267

- **Комментарий:**

Возвращение пустой строки может быть непонятно для клиентов. Рассмотрите возможность возврата JSON-ответа с сообщением об успешном удалении.

- **Текущий код:**

```
comment = article.comments.filter_by(id=cid, author=current_user.profile).first()  
comment.delete()  
return '', 200
```

---

Строка 268

- **Комментарий:**

Метод `delete()` не является стандартным для объектов SQLAlchemy. Используйте `db.session.delete(comment)` для удаления комментария.

- **Текущий код:**

```
comment.delete()  
return '', 200
```

---

**Файл:**

**flask-realworld-example-app-master/conduit/user/models.py**

Строка 13

- **Комментарий:**

Удалите лишние пустые строки для улучшения читаемости кода.

- **Текущий код:**

```
class User(SurrogatePK, Model):  
  
    __tablename__ = 'users'
```

---

Строка 21

- **Комментарий:**

Поле `updated_at` должно обновляться автоматически при каждом изменении записи. Рассмотрите использование `server_default` и `onupdate`.

- **Текущий код:**

```
password = Column(db.Binary(128), nullable=True)  
created_at = Column(db.DateTime, nullable=False, default=dt.datetime.utcnow)  
updated_at = Column(db.DateTime, nullable=False, default=dt.datetime.utcnow)  
bio = Column(db.String(300), nullable=True)  
image = Column(db.String(120), nullable=True)
```

---

Строка 25



- **Комментарий:**

Использование атрибута token с типом str и значением по умолчанию может привести к проблемам с состоянием объекта. Рассмотрите возможность использования свойства или метода для генерации токена.

- **Текущий код:**

```
bio = Column(db.String(300), nullable=True)
image = Column(db.String(120), nullable=True)
token: str = ''

def __init__(self, username, email, password=None, **kwargs):
```

---

Строка 31

- **Комментарий:**

Используйте super() вместо прямого вызова конструктора базового класса для лучшей совместимости с множественным наследованием.

- **Текущий код:**

```
# TODO @dataclass
"""Create instance."""
db.Model.__init__(self, username=username, email=email, **kwargs)
```

---

Строка 40

- **Комментарий:**

Установка self.password в None не требуется, так как поле уже имеет значение по умолчанию None. Удалите эту строку.

- **Текущий код:**

```
self.set_password(password)
else:
    self.password = None

def set_password(self, password):
```

---

Строка 52

- **Комментарий:**

Используйте f-строки для форматирования строк, так как они более эффективны и читаемы.

- **Текущий код:**

```
def __repr__(self):  
    """Represent instance as a unique string."""  
    return '<User({username!r})>'.format(username=self.username)
```

---

## Файл:

**flask-realworld-example-app-master/conduit/user/serializers.py**

Строка 20

- **Комментарий:**

Использование 'self' в Nested вызове может привести к проблемам с рекурсией и читаемостью кода. Рассмотрите возможность использования явного импорта и ссылки на класс UserSchema.

- **Текущий код:**

```
updatedAt = fields.DateTime(attribute='updated_at')  
# ugly hack.  
user = fields.Nested('self', exclude=('user',), default=True, load_only=True)  
  
@pre_load
```

---

Строка 28

- **Комментарий:**

Логика в методе make\_user может быть улучшена. Условие data.get('email', True) всегда будет истинным, если ключ 'email' присутствует, даже если его значение -

пустая строка. Рекомендуется использовать `data.get('email') is not None` and `data.get('email') != ''` для проверки.

- **Текущий код:**

```
# some of the frontends send this like an empty string and some send
# null
if not data.get('email', True):
    del data['email']
if not data.get('image', True):
```

---

Строка 31

- **Комментарий:**

Аналогично предыдущему комментарию, условие `data.get('image', True)` может быть улучшено на `data.get('image') is not None` and `data.get('image') != ''` для корректной проверки наличия и значения ключа.

- **Текущий код:**

```
del data['email']
if not data.get('image', True):
    del data['image']
return data
```

---

Строка 41

- **Комментарий:**

Атрибут `strict` в классе `Meta` устарел в `Marshmallow 3` и выше. Рекомендуется удалить его или использовать соответствующие параметры в конструкторе схемы.

- **Текущий код:**

```
class Meta:
    strict = True
```

---

## Файл: flask-realworld-example-app-master/conduit/user/views.py

Строка 27

- **Комментарий:**

Избыточный вызов метода save(). Объект UserProfile не нуждается в двойном сохранении.

- **Текущий код:**

```
def register_user(username, password, email, **kwargs):  
    try:  
        userprofile = UserProfile(User(username, email, password=password, **kwargs).save()).save()
```

---

Строка 29

- **Комментарий:**

Создание объекта User и UserProfile в одной строке ухудшает читаемость кода. Рассмотрите возможность разделения на две строки.

- **Текущий код:**

```
    try:  
        userprofile = UserProfile(User(username, email, password=password, **kwargs).save()).save()  
        userprofile.user.token = create_access_token(identity=userprofile.user)  
    except IntegrityError:
```

---

Строка 33

- **Комментарий:**

Пустая строка не нужна. Удалите ее для улучшения читаемости кода.

- **Текущий код:**

```
        userprofile.user.token = create_access_token(identity=userprofile.user)  
        except IntegrityError:  
  
        db.session.rollback()
```

```
raise InvalidUsage.user_already_registered()
```

---

Строка 44

- **Комментарий:**

Функция не возвращает значение в случае, если пользователь не найден или пароль неверен. Рекомендуется явно указывать возвращаемое значение.

- **Текущий код:**

```
@jwt_optional
@use_kwargs(user_schema)
@marshal_with(user_schema)
def login_user(email, password, **kwargs):
```

---

Строка 48

- **Комментарий:**

Использование метода `first()` может быть неэффективным для больших таблиц, так как он загружает все строки в память перед возвратом первой. Рассмотрите использование `one_or_none()` для более эффективного запроса.

- **Текущий код:**

```
def login_user(email, password, **kwargs):

    user = User.query.filter_by(email=email).first()
    if user is not None and user.check_password(password):
        user.token = create_access_token(identity=user, fresh=True)
```

---

Строка 54

- **Комментарий:**

Исключение `InvalidUsage.user_not_found()` вызывается напрямую, что может быть неправильным, если `InvalidUsage` является классом исключений. Рассмотрите вызов через конструктор, например, `InvalidUsage('User not found')`.

- **Текущий код:**

```
return user
else:
    raise InvalidUsage.user_not_found()
```

---

Строка 63

- **Комментарий:**

Использование `current_user` напрямую может быть небезопасным и неэффективным. Рассмотрите возможность использования `user`, который уже был присвоен.

- **Текущий код:**

```
@marshal_with(user_schema)
def get_user():
    user = current_user
    # Not sure about this
    user.token = request.headers.environ['HTTP_AUTHORIZATION'].split('Token ')[1]
```

---

Строка 66

- **Комментарий:**

Доступ к `environ` через `request.headers` не является стандартным и может вызвать ошибку `KeyError`. Используйте `request.headers.get('Authorization', '').split('Token ', 1)` для безопасного доступа к заголовку.

- **Текущий код:**

```
# Not sure about this
user.token = request.headers.get('Authorization', '').split('Token ', 1)
return current_user
```

---

Строка 68

- **Комментарий:**

Возвращается `current_user` вместо `user`, что может быть неожиданным поведением. Рассмотрите возможность возврата `user`.

- **Текущий код:**

```
user.token = request.headers.environ['HTTP_AUTHORIZATION'].split('Token ')[1]
return current_user
```

---

Строка 77

- **Комментарий:**

Пустые строки не нужны. Удалите лишние пустые строки.

- **Текущий код:**

```
@use_kwargs(user_schema)
@marshal_with(user_schema)
def update_user(**kwargs):

    user = current_user
```

---

Строка 82

- **Комментарий:**

Использование `pop` для извлечения пароля может быть нежелательным, если пароль не должен быть удален из `kwargs`. Рассмотрите возможность использования `get`.

- **Текущий код:**

```
user = current_user
# take in consideration the password
password = kwargs.pop('password', None)
if password:
    user.set_password(password)
```

---

Строка 87

- **Комментарий:**

Обновление поля `updated_at` с `created_at` не имеет смысла. Используйте текущее время для обновления поля `updated_at`.

- **Текущий код:**

```
user.set_password(password)
    if 'updated_at' in kwargs:
        kwargs['updated_at'] = user.created_at.replace(tzinfo=None)
    user.update(**kwargs)
```

---

Строка 89

- **Комментарий:**

Метод `update` должен учитывать, какие поля действительно нужно обновить. Рассмотрите возможность фильтрации `kwargs` перед передачей в метод `update`.

- **Текущий код:**

```
kwargs['updated_at'] = user.created_at.replace(tzinfo=None)
    user.update(**kwargs)
    return user
```

---

## Файл:

**flask-realworld-example-app-master/conduit/profile/models.py**

Строка 22

- **Комментарий:**

Переопределение `id` не обязательно, если используется `SurrogatePK`. Рассмотрите возможность удаления этой строки.

- **Текущий код:**

```
# id is needed for primary join, it does work with SurrogatePK class
    id = db.Column(db.Integer, primary_key=True)

    user_id = reference_col('users', nullable=False)
```



---

Строка 34

- **Комментарий:**

Использование `db.Model.__init__` не соответствует принципам наследования. Используйте `super().__init__(user=user, **kwargs)`.

- **Текущий код:**

```
lazy='dynamic')

def __init__(self, user, **kwargs):
    # TODO @dataclass
    db.Model.__init__(self, user=user, **kwargs)
```

---

Строка 56

- **Комментарий:**

Использование `current_user` в модели может нарушить принцип разделения ответственности. Лучше переместить эту логику в сервисный слой или контроллер.

- **Текущий код:**

```
@property
def following(self):
    if current_user:
        return current_user.profile.is_following(self)
```

---

Строка 75

- **Комментарий:**

Свойство `email` не относится к профилю пользователя, а к пользователю. Рассмотрите возможность удаления этого свойства из `UserProfile`.

- **Текущий код:**

```
@property
```

```
def email(self):  
    return self.user.email
```

---

## Файл:

**flask-realworld-example-app-master/conduit/profile/serializers.py**

Строка 16

- **Комментарий:**

Использование 'self' в качестве строки для вложенного поля может привести к проблемам с импортом и читаемостью кода. Рассмотрите возможность использования lambda или явного импорта класса.

- **Текущий код:**

```
following = fields.Boolean()  
    # ugly hack.  
    profile = fields.Nested('self', exclude=('profile',), default=True, load_only=True)  
  
@pre_load
```

---

Строка 20

- **Комментарий:**

Метод make\_user предполагает, что ключ 'profile' всегда присутствует в данных, что может вызвать KeyError. Добавьте проверку наличия ключа.

- **Текущий код:**

```
@pre_load  
def make_user(self, data, **kwargs):  
    return data['profile']
```

---

Строка 25

- **Комментарий:**

Метод `dump_user` всегда оборачивает данные в словарь с ключом 'profile', что может быть избыточным в некоторых случаях. Убедитесь, что это поведение ожидаемо и необходимо.

- **Текущий код:**

```
@post_dump
def dump_user(self, data, **kwargs):
    return {'profile': data}
```

---

**Файл:**

**`flask-realworld-example-app-master/conduit/profile/views.py`**

Строка 21

- **Комментарий:**

Удалите лишние пустые строки для улучшения читаемости кода.

- **Текущий код:**

```
@blueprint.route('/api/profiles/<username>', methods=('GET',))
@jwt_optional
@marshal_with(profile_schema)
def get_profile(username):
```

---

Строка 30

- **Комментарий:**

Используйте метод `get()` вместо `filter_by().first()` для более эффективного поиска по первичному ключу или уникальному полю.

- **Текущий код:**

```
user = User.query.filter_by(username=username).first()
if not user:
    raise InvalidUsage.user_not_found()
```

---

Строка 33

- **Комментарий:**

Исключение `InvalidUsage.user_not_found()` может быть неинформативным. Рассмотрите возможность создания более специфического исключения или передачи дополнительной информации.

- **Текущий код:**

```
if not user:
    raise InvalidUsage.user_not_found()
return user.profile
```

---

Строка 42

- **Комментарий:**

Удалите лишние пустые строки для улучшения читаемости кода.

- **Текущий код:**

```
@jwt_required
@marshal_with(profile_schema)
def follow_user(username):
```

---

Строка 48

- **Комментарий:**

Используйте конкретные исключения вместо общего `InvalidUsage` для повышения точности обработки ошибок.

- **Текущий код:**

```
user = User.query.filter_by(username=username).first()
if not user:
    raise InvalidUsage.user_not_found()
```

---

Строка 52

- **Комментарий:**

Проверьте, что пользователь не пытается подписаться на самого себя, чтобы избежать нелогичных ситуаций.

- **Текущий код:**

```
if not user:
    raise InvalidUsage.user_not_found()
current_user.profile.follow(user.profile)
current_user.profile.save()
```

---

Строка 54

- **Комментарий:**

Рассмотрите возможность объединения сохранения профилей в один вызов, если это возможно, для оптимизации операций с базой данных.

- **Текущий код:**

```
current_user.profile.follow(user.profile)
current_user.profile.save()
return user.profile
```

---

Строка 63

- **Комментарий:**

Удалите лишние пустые строки для улучшения читаемости кода.

- **Текущий код:**

```
@jwt_required
@marshal_with(profile_schema)
def unfollow_user(username):
```

---

Строка 71

- **Комментарий:**

Проверка наличия пользователя должна быть выполнена раньше, чтобы избежать потенциальных ошибок. Убедитесь, что `current_user` также существует.

- **Текущий код:**

```
user = User.query.filter_by(username=username).first()
if not user:
    raise InvalidUsage.user_not_found()
```

---

Строка 76

- **Комментарий:**

Возвращать объект профиля другого пользователя может быть не безопасно. Рассмотрите возможность возврата более ограниченной информации или статуса операции.

- **Текущий код:**

```
raise InvalidUsage.user_not_found()
current_user.profile.unfollow(user.profile)
current_user.profile.save()
return user.profile
```

---