МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ

ФЕДЕРАЦИИ

ФГБОУ «ДонНТУ»

Институт КНТ

Кафедра ПИ

Лабораторная работа №4

по курсу: «Профессиональная практика программной инженерии»

по теме: «Создание самодокументирующегося кода»

Выполнил:

ст. гр. ПИ-21в

Неснов А.А.

Проверил:

асс. каф. ПИ

Филипишин Д.А.

Донецк 2024

Цель работы: научиться добавлять в программный код специальным образом оформление докблок-комментарии, для последующей автоматической генерации API reference, а также познакомиться с форматом оформления документации DocBook.

Задание к лабораторной работе:

Внедрить в ранее разработанный программный код докблоки, соответствующие выбранному языку программирования. Разработать DocBook и проверить корректность созданного DocBook файла с помощью средств редактора или одного из онлайн валидаторов. Автоматически сформировать справочное руководство для программистов и выслать вместе с отчётом. В руководстве должны присутствовать все модули, описанные в лабораторной работе №1.

Задание по лабораторной работе

1. Во всех файлах с исходным кодом добавить докблоки ко всем классам, методам, свойствам и функциям.

2. Закоммитить изменения в репозиторий.

3. Установить один из XML-редакторов (по желанию).

4. Оформить краткое описание разработанного продукта (не менее пяти абзацев).

5. Рассмотреть с примерами кода не менее трех сценариев использования вашего приложения.

6. Проверить корректность созданного DocBook файла с помощью средств редактора или одного из онлайн валидаторов, например, mashuosoft.com/docbook/validator

Содержание отчёта

1. Титульный лист.

2. Тема выбранного проекта.

3. Несколько примеров докблоков из программного кода разрабатываемого проекта подходящих к выбранному языку программирования, а также XML.

4. Скриншот полученного справочного руководства.

5. Ссылка на аккаунт.

6. Ссылка на репозиторий.

7. Руководство высылать вместе с отчётом.

ССЫЛКА НА АККАУНТ:

<https://github.com/anesnov>

ССЫЛКА НА РЕПОЗИТОРИЙ:

<https://github.com/anesnov/PPII.git>

Вариант 4

Форум

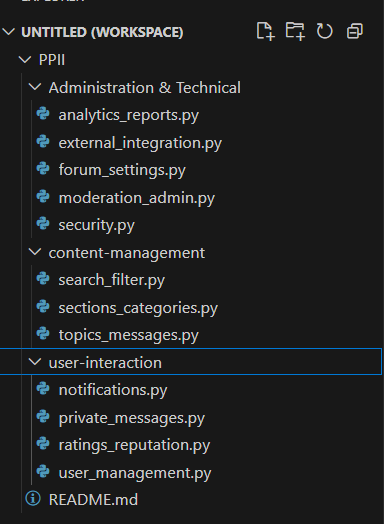


Рисунок 1- Структура проекта

Код с добавленными docstring'ами в формате Google Style, который используется для автоматической генерации документации:

class Security:

"""Класс для обеспечения безопасности форума: защита от спама, шифрование данных.

Attributes:

spam\_filters (list): Список правил для фильтрации спама.

encryption\_key (str): Ключ для шифрования данных (в демо-версии фиксированный).

"""

def \_\_init\_\_(self):

"""Инициализирует Security со стандартными настройками."""

self.spam\_filters = []

self.encryption\_key = "secure\_key"

def add\_spam\_filter(self, filter\_rule):

"""Добавляет правило для фильтрации спама.

Args:

filter\_rule (str): Правило или ключевое слово для фильтрации спама.

Returns:

str: Подтверждение добавления правила.

"""

self.spam\_filters.append(filter\_rule)

return "Spam filter added."

def encrypt\_data(self, data):

"""Шифрует данные (демонстрационная реализация).

Args:

data (str): Данные для шифрования.

Returns:

str: Зашифрованные данные.

"""

encrypted = "".join(chr(ord(char) + 1) for char in data)

return encrypted

def decrypt\_data(self, encrypted\_data):

"""Дешифрует данные (демонстрационная реализация).

Args:

encrypted\_data (str): Данные для дешифрования.

Returns:

str: Расшифрованные данные.

"""

decrypted = "".join(chr(ord(char) - 1) for char in encrypted\_data)

return decrypted

def detect\_spam(self, message):

"""Проверяет сообщение на спам.

Args:

message (str): Сообщение для проверки.

Returns:

bool: True если найдено совпадение с правилами спама, иначе False.

"""

for rule in self.spam\_filters:

if rule in message:

return True

return False

class TopicsMessages:

"""Класс для управления темами и сообщениями на форуме.

Attributes:

topics (dict): Словарь тем форума в формате {topic\_id: topic\_data}.

"""

def \_\_init\_\_(self):

"""Инициализирует TopicsMessages с пустым списком тем."""

self.topics = {}

def create\_topic(self, topic\_id, title, author, content):

"""Создает новую тему.

Args:

topic\_id (str/int): Уникальный идентификатор темы.

title (str): Заголовок темы.

author (str): Автор темы.

content (str): Содержание темы.

Returns:

str: Сообщение о результате операции.

"""

if topic\_id in self.topics:

return "Topic already exists."

self.topics[topic\_id] = {"title": title, "author": author, "content": content, "comments": []}

return "Topic created successfully."

def add\_comment(self, topic\_id, comment, author):

"""Добавляет комментарий к теме.

Args:

topic\_id (str/int): Идентификатор темы.

comment (str): Текст комментария.

author (str): Автор комментария.

Returns:

str: Сообщение о результате операции.

"""

if topic\_id in self.topics:

self.topics[topic\_id]["comments"].append({"author": author, "comment": comment})

return "Comment added."

return "Topic not found."

def edit\_topic(self, topic\_id, new\_title=None, new\_content=None):

"""Редактирует тему.

Args:

topic\_id (str/int): Идентификатор темы.

new\_title (str, optional): Новый заголовок. Defaults to None.

new\_content (str, optional): Новое содержание. Defaults to None.

Returns:

str: Сообщение о результате операции.

"""

if topic\_id in self.topics:

if new\_title:

self.topics[topic\_id]["title"] = new\_title

if new\_content:

self.topics[topic\_id]["content"] = new\_content

return "Topic updated."

return "Topic not found."

def delete\_topic(self, topic\_id):

"""Удаляет тему.

Args:

topic\_id (str/int): Идентификатор темы.

Returns:

str: Сообщение о результате операции.

"""

if topic\_id in self.topics:

del self.topics[topic\_id]

return "Topic deleted."

return "Topic not found."

class UserManagement:

"""Класс для управления пользователями: регистрация, авторизация, роли.

Attributes:

users (dict): Словарь пользователей в формате {username: user\_data}.

"""

def \_\_init\_\_(self):

"""Инициализирует UserManagement с пустым списком пользователей."""

self.users = {}

def register\_user(self, username, password, email):

"""Регистрирует нового пользователя.

Args:

username (str): Логин пользователя.

password (str): Пароль пользователя.

email (str): Email пользователя.

Returns:

str: Сообщение о результате операции.

"""

if username in self.users:

return "User already exists."

self.users[username] = {"password": password, "email": email, "role": "user"}

return "User registered successfully."

def login\_user(self, username, password):

"""Авторизует пользователя.

Args:

username (str): Логин пользователя.

password (str): Пароль пользователя.

Returns:

str: Сообщение о результате операции.

"""

if username in self.users and self.users[username]["password"] == password:

return "Login successful."

return "Invalid credentials."

def edit\_profile(self, username, new\_email=None, new\_password=None):

"""Редактирует профиль пользователя.

Args:

username (str): Логин пользователя.

new\_email (str, optional): Новый email. Defaults to None.

new\_password (str, optional): Новый пароль. Defaults to None.

Returns:

str: Сообщение о результате операции.

"""

if username in self.users:

if new\_email:

self.users[username]["email"] = new\_email

if new\_password:

self.users[username]["password"] = new\_password

return "Profile updated."

return "User not found."

def change\_role(self, username, new\_role):

"""Изменяет роль пользователя.

Args:

username (str): Логин пользователя.

new\_role (str): Новая роль (например, 'admin', 'moderator').

Returns:

str: Сообщение о результате операции.

"""

if username in self.users:

self.users[username]["role"] = new\_role

return "Role updated."

return "User not found."

**DokBook Документация**

**DOCX-версия документации, сгенерированная на сайте** [**https://pandoc.org/try/**](https://pandoc.org/try/)

Форум: документация системы

# Обзор проекта

Форум представляет собой многофункциональную платформу для обсуждений с полным набором инструментов для управления пользователями, контентом и безопасностью.

## Архитектура системы

* User Interaction - управление пользователями и их взаимодействием
* Content Management - работа с контентом форума
* Administration & Technical - администрирование и техническая инфраструктура

# Основные функциональные модули

## Модуль безопасности (Security)

security = Security()  
security.add\_spam\_filter("spam\_keyword")  
encrypted = security.encrypt\_data("secret")  
print(security.detect\_spam("This is spam\_keyword")) # True

## Управление темами (TopicsMessages)

forum = TopicsMessages()  
forum.create\_topic(1, "First Topic", "user1", "Hello world!")  
forum.add\_comment(1, "Nice topic!", "user2")  
print(forum.topics[1]["comments"]) # [{'author': 'user2', 'comment': 'Nice topic!'}]

## Управление пользователями (UserManagement)

users = UserManagement()  
users.register\_user("john", "pass123", "john@example.com")  
print(users.login\_user("john", "pass123")) # Login successful  
users.change\_role("john", "moderator")

# Типовые сценарии использования

## Сценарий 1: Регистрация пользователя и создание темы

# Инициализация модулей  
users = UserManagement()  
forum = TopicsMessages()  
  
# Регистрация нового пользователя  
users.register\_user("alice", "securepass", "alice@example.com")  
  
# Создание темы после успешной авторизации  
if users.login\_user("alice", "securepass") == "Login successful":  
 forum.create\_topic(101, "Python Tips", "alice", "Share your Python tips here!")  
 print("Topic created successfully")

## Сценарий 2: Модерация контента

# Инициализация модулей  
security = Security()  
forum = TopicsMessages()  
  
# Настройка спам-фильтров  
security.add\_spam\_filter("casino")  
security.add\_spam\_filter("viagra")  
  
# Проверка нового сообщения  
new\_message = "Visit our casino for viagra discounts!"  
if security.detect\_spam(new\_message):  
 print("Spam detected - message rejected")  
else:  
 forum.add\_comment(101, new\_message, "unknown\_user")

## Сценарий 3: Интеграция систем

# Комплексный пример работы системы  
users = UserManagement()  
forum = TopicsMessages()  
security = Security()  
  
# Регистрация администратора  
users.register\_user("admin", "adminpass", "admin@forum.com")  
users.change\_role("admin", "administrator")  
  
# Создание важного уведомления  
if users.login\_user("admin", "adminpass") == "Login successful":  
 encrypted\_notice = security.encrypt\_data("System maintenance at 3AM")  
 forum.create\_topic(  
 999,   
 "IMPORTANT",   
 "admin",   
 f"ENCRYPTED:{encrypted\_notice}"  
 )  
 print("Admin notification posted")

# Технические характеристики

* Гибкая система ролей (пользователь, модератор, администратор)
* Многоуровневая система безопасности
* Полнотекстовый поиск с поддержкой фильтров
* API для интеграции с внешними сервисами

# Перспективы развития

* Система плагинов
* Мобильное приложение
* Интеграция с социальными сетями
* Развитие системы аналитики