МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ

ФЕДЕРАЦИИ

ФГБОУ ВО «ДонНТУ»

Факультет Интеллектуальных систем и программирования

Кафедра «Программная инженерия» им. Л.П. Фельдмана

Лабораторная работа №4

по курсу: «Профессиональная практика программной инженерии»

по теме: «Создание самодокументирующегося кода»

Выполнил:

ст. гр. ПИ-21б

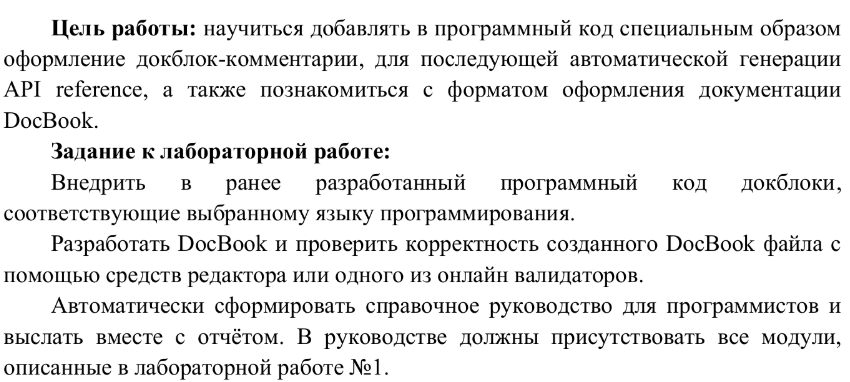
Евтушенко С.Н.

Проверил:

асс. каф. ПИ

Ищенко А.П.

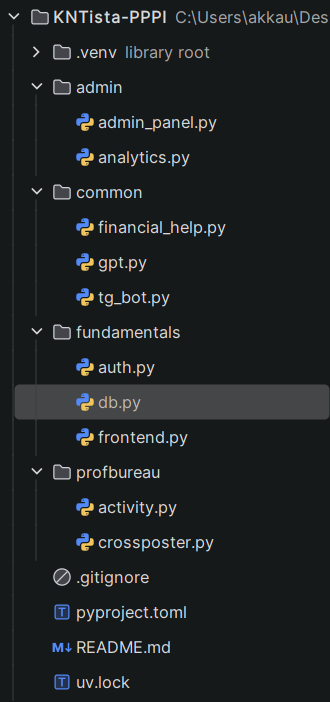
ДОНЕЦК – 2025



**Тема:** проект для обслуживания нужд профбюро Института компьютерных наук и технологий ДонНТУ.

**Репозиторий:** <https://github.com/Scorpi-ON/KNTista-PPPI>

**Структура проекта:**



**Листинг программы с докблоками:**

admin/admin\_panel.py

class AdminPanel:

"""

Класс AdminPanel представляет собой модуль для управления административной панелью.

Он предоставляет функциональность для работы с администраторами и отправки сообщений.

"""

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

"""

Инициализирует экземпляр AdminPanel.

При создании объекта выводит сообщение о том, что модуль административной панели был инициализирован.

"""

print("This is Admin Panel module")

@property

def admins\_count(self) -> int:

"""

Возвращает количество администраторов в системе.

:return: Количество администраторов. В текущей реализации всегда возвращает 0.

:rtype: int

"""

return 0

def send\_message(self, user\_ids: list[int]) -> None:

"""

Отправляет сообщение указанным пользователям.

:param user\_ids: Список идентификаторов пользователей, которым будет отправлено сообщение.

:type user\_ids: list[int]

:return: None

"""

print("This message will be sent to", user\_ids)

admin/analytics.py

from datetime import datetime

class Analytics:

"""

Класс Analytics представляет собой модуль для работы с аналитикой.

Он предоставляет методы для получения метрик и проверки времени последнего обращения к данным.

"""

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

"""

Инициализирует экземпляр Analytics.

При создании объекта выводит сообщение о том, что модуль аналитики был инициализирован.

"""

print("This is Analytics module")

def get\_metrics(self, metrics\_name: str) -> str:

"""

Возвращает метрики по указанному имени.

:param metrics\_name: Название метрики, которую необходимо получить.

:type metrics\_name: str

:return: Строка с информацией о запросе метрики.

:rtype: str

"""

return "Getting metrics " + metrics\_name

@staticmethod

def last\_checked() -> datetime:

"""

Возвращает текущее время как время последней проверки.

:return: Текущее время в формате datetime.

:rtype: datetime

"""

return datetime.now()

common/financial\_help.py

class FinancialHelp:

"""

Класс FinancialHelp представляет собой модуль для работы с финансовой помощью.

Он предоставляет функциональность для управления запросами на финансовую помощь,

отслеживания статуса заявок и обработки платежей.

"""

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

"""

Инициализирует экземпляр FinancialHelp.

При создании объекта выводит сообщение о том, что модуль финансовой помощи был инициализирован.

Также инициализирует пустой список для хранения заявок на финансовую помощь.

"""

print("This is Financial Help module")

self.\_requests = [] # Список для хранения заявок на финансовую помощь

@property

def total\_requests(self) -> int:

"""

Возвращает общее количество заявок на финансовую помощь.

:return: Количество заявок.

:rtype: int

"""

return len(self.\_requests)

def add\_request(self, user\_id: int, amount: float, description: str) -> None:

"""

Добавляет новую заявку на финансовую помощь.

:param user\_id: Идентификатор пользователя, подающего заявку.

:type user\_id: int

:param amount: Сумма запрашиваемой финансовой помощи.

:type amount: float

:param description: Описание причины запроса финансовой помощи.

:type description: str

:return: None

"""

request = {

"user\_id": user\_id,

"amount": amount,

"description": description,

"status": "pending", # Статус заявки по умолчанию

"created\_at": datetime.now(),

}

self.\_requests.append(request)

print(f"New financial help request added by user {user\_id} for amount {amount}.")

def approve\_request(self, request\_id: int) -> None:

"""

Одобряет заявку на финансовую помощь по её идентификатору.

:param request\_id: Индекс заявки в списке заявок.

:type request\_id: int

:return: None

"""

if 0 <= request\_id < len(self.\_requests):

self.\_requests[request\_id]["status"] = "approved"

print(f"Request {request\_id} has been approved.")

else:

print(f"Request {request\_id} not found.")

def reject\_request(self, request\_id: int) -> None:

"""

Отклоняет заявку на финансовую помощь по её идентификатору.

:param request\_id: Индекс заявки в списке заявок.

:type request\_id: int

:return: None

"""

if 0 <= request\_id < len(self.\_requests):

self.\_requests[request\_id]["status"] = "rejected"

print(f"Request {request\_id} has been rejected.")

else:

print(f"Request {request\_id} not found.")

def get\_request\_status(self, request\_id: int) -> str:

"""

Возвращает статус заявки на финансовую помощь по её идентификатору.

:param request\_id: Индекс заявки в списке заявок.

:type request\_id: int

:return: Статус заявки.

:rtype: str

"""

if 0 <= request\_id < len(self.\_requests):

return self.\_requests[request\_id]["status"]

else:

return "Request not found."

def list\_requests(self) -> list:

"""

Возвращает список всех заявок на финансовую помощь.

:return: Список заявок.

:rtype: list

"""

return self.\_requests

common/gpt.py

class GPT:

"""

Класс GPT представляет собой модуль для работы с генеративными предобученными трансформерами (GPT).

Он предоставляет функциональность для генерации текста, настройки модели и управления контекстом.

"""

def \_\_init\_\_(self, model\_name: str = "gpt-4"):

"""

Инициализирует экземпляр GPT.

:param model\_name: Название модели GPT (по умолчанию "gpt-4").

:type model\_name: str

"""

print("This is GPT module")

self.model\_name = model\_name

self.\_context = [] # Список для хранения контекста диалога

self.\_temperature = 0.7 # Параметр температуры для генерации текста

self.\_max\_tokens = 100 # Максимальное количество токенов в ответе

@property

def context(self) -> list:

"""

Возвращает текущий контекст диалога.

:return: Список сообщений в контексте.

:rtype: list

"""

return self.\_context

@property

def temperature(self) -> float:

"""

Возвращает текущее значение параметра температуры.

:return: Значение температуры.

:rtype: float

"""

return self.\_temperature

@temperature.setter

def temperature(self, value: float) -> None:

"""

Устанавливает значение параметра температуры.

:param value: Новое значение температуры (должно быть между 0 и 1).

:type value: float

:raises ValueError: Если значение температуры выходит за допустимые пределы.

"""

if 0 <= value <= 1:

self.\_temperature = value

else:

raise ValueError("Temperature must be between 0 and 1.")

@property

def max\_tokens(self) -> int:

"""

Возвращает текущее значение максимального количества токенов.

:return: Максимальное количество токенов.

:rtype: int

"""

return self.\_max\_tokens

@max\_tokens.setter

def max\_tokens(self, value: int) -> None:

"""

Устанавливает значение максимального количества токенов.

:param value: Новое значение максимального количества токенов (должно быть положительным).

:type value: int

:raises ValueError: Если значение максимального количества токенов отрицательное.

"""

if value > 0:

self.\_max\_tokens = value

else:

raise ValueError("Max tokens must be a positive integer.")

def generate\_text(self, prompt: str) -> str:

"""

Генерирует текст на основе заданного промпта и текущего контекста.

:param prompt: Текст запроса для генерации.

:type prompt: str

:return: Сгенерированный текст.

:rtype: str

"""

# Имитация генерации текста

response = f"Generated response for prompt: '{prompt}' using {self.model\_name}."

self.\_context.append({"role": "user", "content": prompt})

self.\_context.append({"role": "assistant", "content": response})

return response

def clear\_context(self) -> None:

"""

Очищает текущий контекст диалога.

"""

self.\_context = []

print("Context has been cleared.")

def set\_model(self, model\_name: str) -> None:

"""

Устанавливает новую модель GPT.

:param model\_name: Название модели GPT.

:type model\_name: str

"""

self.model\_name = model\_name

print(f"Model has been set to {model\_name}.")

common/tg\_bot.py

class TGBot:

"""

Класс TGBot представляет собой модуль для работы с Telegram-ботом.

Он предоставляет функциональность для отправки сообщений, обработки команд и управления состоянием бота.

"""

def \_\_init\_\_(self, bot\_token: str):

"""

Инициализирует экземпляр TGBot.

:param bot\_token: Токен Telegram-бота для авторизации.

:type bot\_token: str

"""

print("This is TG Bot module")

self.bot\_token = bot\_token

self.\_commands = {} # Словарь для хранения команд и их обработчиков

self.\_users = {} # Словарь для хранения данных пользователей

self.\_is\_running = False # Флаг, указывающий, запущен ли бот

@property

def is\_running(self) -> bool:

"""

Возвращает текущее состояние бота (запущен или остановлен).

:return: True, если бот запущен, иначе False.

:rtype: bool

"""

return self.\_is\_running

def add\_command(self, command: str, handler: callable) -> None:

"""

Добавляет новую команду и её обработчик.

:param command: Название команды (например, "/start").

:type command: str

:param handler: Функция-обработчик команды.

:type handler: callable

"""

self.\_commands[command] = handler

print(f"Command '{command}' has been added.")

def remove\_command(self, command: str) -> None:

"""

Удаляет команду и её обработчик.

:param command: Название команды для удаления.

:type command: str

"""

if command in self.\_commands:

del self.\_commands[command]

print(f"Command '{command}' has been removed.")

else:

print(f"Command '{command}' not found.")

def start(self) -> None:

"""

Запускает Telegram-бота.

"""

if not self.\_is\_running:

self.\_is\_running = True

print("Bot has been started.")

else:

print("Bot is already running.")

def stop(self) -> None:

"""

Останавливает Telegram-бота.

"""

if self.\_is\_running:

self.\_is\_running = False

print("Bot has been stopped.")

else:

print("Bot is already stopped.")

def send\_message(self, chat\_id: int, text: str) -> None:

"""

Отправляет сообщение в указанный чат.

:param chat\_id: Идентификатор чата.

:type chat\_id: int

:param text: Текст сообщения.

:type text: str

"""

print(f"Message sent to chat {chat\_id}: {text}")

def handle\_update(self, update: dict) -> None:

"""

Обрабатывает входящее обновление от Telegram (например, сообщение или команду).

:param update: Словарь с данными обновления.

:type update: dict

"""

if "message" in update and "text" in update["message"]:

message\_text = update["message"]["text"]

chat\_id = update["message"]["chat"]["id"]

if message\_text in self.\_commands:

self.\_commands[message\_text](chat\_id)

else:

self.send\_message(chat\_id, "Unknown command. Type /help for a list of commands.")

else:

print("Unsupported update type.")

def set\_user\_data(self, user\_id: int, key: str, value: str) -> None:

"""

Устанавливает данные пользователя.

:param user\_id: Идентификатор пользователя.

:type user\_id: int

:param key: Ключ для хранения данных.

:type key: str

:param value: Значение данных.

:type value: str

"""

if user\_id not in self.\_users:

self.\_users[user\_id] = {}

self.\_users[user\_id][key] = value

print(f"Data for user {user\_id} has been updated.")

def get\_user\_data(self, user\_id: int, key: str) -> str | None:

"""

Возвращает данные пользователя по ключу.

:param user\_id: Идентификатор пользователя.

:type user\_id: int

:param key: Ключ для получения данных.

:type key: str

:return: Значение данных или None, если данные не найдены.

:rtype: str

"""

if user\_id in self.\_users and key in self.\_users[user\_id]:

return self.\_users[user\_id][key]

return None

fundamentals/auth.py

class Auth:

"""

Класс Auth представляет собой модуль для работы с аутентификацией и авторизацией пользователей.

Он предоставляет функциональность для регистрации, входа, выхода и управления пользовательскими сессиями.

"""

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

"""

Инициализирует экземпляр Auth.

При создании объекта выводит сообщение о том, что модуль аутентификации был инициализирован.

Также инициализирует пустой словарь для хранения зарегистрированных пользователей и текущих сессий.

"""

print("This is Auth module")

self.\_users = {} # Словарь для хранения зарегистрированных пользователей

self.\_sessions = {} # Словарь для хранения активных сессий

def register(self, username: str, password: str) -> bool:

"""

Регистрирует нового пользователя.

:param username: Имя пользователя.

:type username: str

:param password: Пароль пользователя.

:type password: str

:return: True, если регистрация прошла успешно, иначе False.

:rtype: bool

"""

if username in self.\_users:

print(f"User '{username}' already exists.")

return False

self.\_users[username] = {"password": password}

print(f"User '{username}' has been registered.")

return True

def login(self, username: str, password: str) -> bool:

"""

Выполняет вход пользователя в систему.

:param username: Имя пользователя.

:type username: str

:param password: Пароль пользователя.

:type password: str

:return: True, если вход выполнен успешно, иначе False.

:rtype: bool

"""

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

self.\_sessions[username] = True

print(f"User '{username}' has logged in.")

return True

print(f"Invalid credentials for user '{username}'.")

return False

def logout(self, username: str) -> bool:

"""

Выполняет выход пользователя из системы.

:param username: Имя пользователя.

:type username: str

:return: True, если выход выполнен успешно, иначе False.

:rtype: bool

"""

if username in self.\_sessions:

del self.\_sessions[username]

print(f"User '{username}' has logged out.")

return True

print(f"User '{username}' is not logged in.")

return False

def is\_authenticated(self, username: str) -> bool:

"""

Проверяет, аутентифицирован ли пользователь.

:param username: Имя пользователя.

:type username: str

:return: True, если пользователь аутентифицирован, иначе False.

:rtype: bool

"""

return username in self.\_sessions

def change\_password(self, username: str, old\_password: str, new\_password: str) -> bool:

"""

Изменяет пароль пользователя.

:param username: Имя пользователя.

:type username: str

:param old\_password: Текущий пароль пользователя.

:type old\_password: str

:param new\_password: Новый пароль пользователя.

:type new\_password: str

:return: True, если пароль изменен успешно, иначе False.

:rtype: bool

"""

if username in self.\_users and self.\_users[username]["password"] == old\_password:

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

print(f"Password for user '{username}' has been changed.")

return True

print(f"Failed to change password for user '{username}'.")

return False

def delete\_user(self, username: str) -> bool:

"""

Удаляет пользователя из системы.

:param username: Имя пользователя.

:type username: str

:return: True, если пользователь удален успешно, иначе False.

:rtype: bool

"""

if username in self.\_users:

del self.\_users[username]

if username in self.\_sessions:

del self.\_sessions[username]

print(f"User '{username}' has been deleted.")

return True

print(f"User '{username}' not found.")

return False

fundamentals/db.py

class DB:

"""

Класс DB представляет собой модуль для работы с базой данных.

Он предоставляет функциональность для подключения к базе данных, выполнения запросов и управления данными.

"""

def \_\_init\_\_(self, db\_name: str):

"""

Инициализирует экземпляр DB.

:param db\_name: Название базы данных.

:type db\_name: str

"""

print("This is DB module")

self.db\_name = db\_name

self.\_connection = None # Подключение к базе данных

self.\_tables = {} # Словарь для хранения таблиц и их данных

def connect(self) -> bool:

"""

Подключается к базе данных.

:return: True, если подключение успешно, иначе False.

:rtype: bool

"""

if not self.\_connection:

self.\_connection = True # Имитация успешного подключения

print(f"Connected to database '{self.db\_name}'.")

return True

print(f"Already connected to database '{self.db\_name}'.")

return False

def disconnect(self) -> bool:

"""

Отключается от базы данных.

:return: True, если отключение успешно, иначе False.

:rtype: bool

"""

if self.\_connection:

self.\_connection = None

print(f"Disconnected from database '{self.db\_name}'.")

return True

print(f"No active connection to database '{self.db\_name}'.")

return False

def create\_table(self, table\_name: str, columns: list) -> bool:

"""

Создает новую таблицу в базе данных.

:param table\_name: Название таблицы.

:type table\_name: str

:param columns: Список колонок таблицы.

:type columns: list

:return: True, если таблица создана успешно, иначе False.

:rtype: bool

"""

if table\_name in self.\_tables:

print(f"Table '{table\_name}' already exists.")

return False

self.\_tables[table\_name] = {"columns": columns, "data": []}

print(f"Table '{table\_name}' has been created with columns: {columns}.")

return True

def insert(self, table\_name: str, values: dict) -> bool:

"""

Вставляет данные в указанную таблицу.

:param table\_name: Название таблицы.

:type table\_name: str

:param values: Словарь с данными для вставки (ключи - названия колонок).

:type values: dict

:return: True, если данные вставлены успешно, иначе False.

:rtype: bool

"""

if table\_name not in self.\_tables:

print(f"Table '{table\_name}' does not exist.")

return False

self.\_tables[table\_name]["data"].append(values)

print(f"Data inserted into table '{table\_name}': {values}.")

return True

def select(self, table\_name: str, condition: callable = None) -> list:

"""

Выполняет выборку данных из указанной таблицы.

:param table\_name: Название таблицы.

:type table\_name: str

:param condition: Функция-условие для фильтрации данных (опционально).

:type condition: callable

:return: Список данных, удовлетворяющих условию.

:rtype: list

"""

if table\_name not in self.\_tables:

print(f"Table '{table\_name}' does not exist.")

return []

if condition:

return [row for row in self.\_tables[table\_name]["data"] if condition(row)]

return self.\_tables[table\_name]["data"]

def update(self, table\_name: str, condition: callable, new\_values: dict) -> bool:

"""

Обновляет данные в указанной таблице.

:param table\_name: Название таблицы.

:type table\_name: str

:param condition: Функция-условие для поиска данных, которые нужно обновить.

:type condition: callable

:param new\_values: Словарь с новыми значениями.

:type new\_values: dict

:return: True, если данные обновлены успешно, иначе False.

:rtype: bool

"""

if table\_name not in self.\_tables:

print(f"Table '{table\_name}' does not exist.")

return False

updated = False

for row in self.\_tables[table\_name]["data"]:

if condition(row):

row.update(new\_values)

updated = True

if updated:

print(f"Data in table '{table\_name}' has been updated.")

else:

print(f"No data matching the condition in table '{table\_name}'.")

return updated

def delete(self, table\_name: str, condition: callable) -> bool:

"""

Удаляет данные из указанной таблицы.

:param table\_name: Название таблицы.

:type table\_name: str

:param condition: Функция-условие для поиска данных, которые нужно удалить.

:type condition: callable

:return: True, если данные удалены успешно, иначе False.

:rtype: bool

"""

if table\_name not in self.\_tables:

print(f"Table '{table\_name}' does not exist.")

return False

initial\_count = len(self.\_tables[table\_name]["data"])

self.\_tables[table\_name]["data"] = [

row for row in self.\_tables[table\_name]["data"] if not condition(row)

]

if len(self.\_tables[table\_name]["data"]) < initial\_count:

print(f"Data deleted from table '{table\_name}'.")

return True

print(f"No data matching the condition in table '{table\_name}'.")

return False

def drop\_table(self, table\_name: str) -> bool:

"""

Удаляет таблицу из базы данных.

:param table\_name: Название таблицы.

:type table\_name: str

:return: True, если таблица удалена успешно, иначе False.

:rtype: bool

"""

if table\_name not in self.\_tables:

print(f"Table '{table\_name}' does not exist.")

return False

del self.\_tables[table\_name]

print(f"Table '{table\_name}' has been dropped.")

return True

fundamentals/frontend.py

class Frontend:

"""

Класс Frontend представляет собой модуль для работы с фронтенд-частью приложения.

Он предоставляет функциональность для управления интерфейсом пользователя, обработки событий и отображения данных.

"""

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

"""

Инициализирует экземпляр Frontend.

При создании объекта выводит сообщение о том, что модуль фронтенда был инициализирован.

Также инициализирует пустой словарь для хранения компонентов интерфейса.

"""

print("This is Frontend module")

self.\_components = {} # Словарь для хранения компонентов интерфейса

def add\_component(self, component\_id: str, component\_type: str) -> None:

"""

Добавляет новый компонент в интерфейс.

:param component\_id: Уникальный идентификатор компонента.

:type component\_id: str

:param component\_type: Тип компонента (например, "button", "input", "text").

:type component\_type: str

"""

self.\_components[component\_id] = {"type": component\_type, "state": "inactive"}

print(f"Component '{component\_id}' of type '{component\_type}' has been added.")

def remove\_component(self, component\_id: str) -> None:

"""

Удаляет компонент из интерфейса.

:param component\_id: Уникальный идентификатор компонента.

:type component\_id: str

"""

if component\_id in self.\_components:

del self.\_components[component\_id]

print(f"Component '{component\_id}' has been removed.")

else:

print(f"Component '{component\_id}' not found.")

def update\_component\_state(self, component\_id: str, state: str) -> None:

"""

Обновляет состояние компонента.

:param component\_id: Уникальный идентификатор компонента.

:type component\_id: str

:param state: Новое состояние компонента (например, "active", "inactive", "disabled").

:type state: str

"""

if component\_id in self.\_components:

self.\_components[component\_id]["state"] = state

print(f"State of component '{component\_id}' has been updated to '{state}'.")

else:

print(f"Component '{component\_id}' not found.")

def render(self) -> None:

"""

Отрисовывает интерфейс пользователя на основе текущих компонентов и их состояний.

"""

print("Rendering frontend components:")

for component\_id, component\_data in self.\_components.items():

print(f" - {component\_id} ({component\_data['type']}): {component\_data['state']}")

def handle\_event(self, component\_id: str, event\_type: str) -> None:

"""

Обрабатывает событие, связанное с компонентом.

:param component\_id: Уникальный идентификатор компонента.

:type component\_id: str

:param event\_type: Тип события (например, "click", "input", "hover").

:type event\_type: str

"""

if component\_id in self.\_components:

print(f"Handling '{event\_type}' event for component '{component\_id}'.")

else:

print(f"Component '{component\_id}' not found.")

profbureau/activity.py

from datetime import datetime

class Activity:

"""

Класс Activity представляет собой модуль для отслеживания активности на факультете и составления отчетов.

Он предоставляет функциональность для внесения данных о мероприятиях, участии студентов и генерации отчетов в формате DOCX.

"""

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

"""

Инициализирует экземпляр Activity.

При создании объекта выводит сообщение о том, что модуль активности был инициализирован.

Также инициализирует пустые структуры для хранения данных о мероприятиях и участии студентов.

"""

print("This is Activity module")

self.\_events = [] # Список для хранения данных о мероприятиях

self.\_students = {} # Словарь для хранения данных об участии студентов

def add\_event(self, event\_name: str, date: str, participants: list[int]) -> None:

"""

Добавляет новое мероприятие в систему.

:param event\_name: Название мероприятия.

:type event\_name: str

:param date: Дата проведения мероприятия в формате "YYYY-MM-DD".

:type date: str

:param participants: Список идентификаторов студентов, участвующих в мероприятии.

:type participants: list[int]

"""

event = {

"name": event\_name,

"date": date,

"participants": participants

}

self.\_events.append(event)

for student\_id in participants:

if student\_id not in self.\_students:

self.\_students[student\_id] = []

self.\_students[student\_id].append(event\_name)

print(f"Event '{event\_name}' on {date} has been added with {len(participants)} participants.")

def get\_student\_activity(self, student\_id: int) -> list:

"""

Возвращает список мероприятий, в которых участвовал студент.

:param student\_id: Идентификатор студента.

:type student\_id: int

:return: Список мероприятий.

:rtype: list

"""

return self.\_students.get(student\_id, [])

def generate\_report(self, month: int, year: int) -> str:

"""

Генерирует отчет о мероприятиях за указанный месяц и год в формате DOCX.

:param month: Месяц, за который требуется отчет (1-12).

:type month: int

:param year: Год, за который требуется отчет.

:type year: int

:return: Путь к сгенерированному файлу отчета.

:rtype: str

"""

report\_events = [

event for event in self.\_events

if datetime.strptime(event["date"], "%Y-%m-%d").month == month

and datetime.strptime(event["date"], "%Y-%m-%d").year == year

]

report\_filename = f"activity\_report\_{year}\_{month}.docx"

# Имитация создания DOCX-файла

print(f"Report for {year}-{month} generated with {len(report\_events)} events. Saved as '{report\_filename}'.")

return report\_filename

def list\_events(self) -> list:

"""

Возвращает список всех мероприятий.

:return: Список мероприятий.

:rtype: list

"""

return self.\_events

def list\_students(self) -> list:

"""

Возвращает список всех студентов, участвовавших в мероприятиях.

:return: Список идентификаторов студентов.

:rtype: list

"""

return list(self.\_students.keys())

profbureau/crossposter.py

from datetime import datetime

class Crossposter:

"""

Класс Crossposter представляет собой модуль для публикации постов во всех соцсетях ИКНТ (ВК и Telegram).

Он предоставляет функциональность для одновременной публикации постов, управления атрибутами (хештеги, оформление)

и контроля прав доступа для публикации.

"""

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

"""

Инициализирует экземпляр Crossposter.

При создании объекта выводит сообщение о том, что модуль кросс-постинга был инициализирован.

Также инициализирует пустые структуры для хранения постов, черновиков и прав доступа.

"""

print("This is Crossposter module")

self.\_posts = [] # Список для хранения опубликованных постов

self.\_drafts = [] # Список для хранения черновиков

self.\_users = {} # Словарь для хранения прав доступа пользователей

def add\_user(self, user\_id: int, can\_publish: bool) -> None:

"""

Добавляет пользователя в систему с указанием прав на публикацию.

:param user\_id: Идентификатор пользователя.

:type user\_id: int

:param can\_publish: Имеет ли пользователь право на публикацию.

:type can\_publish: bool

"""

self.\_users[user\_id] = {"can\_publish": can\_publish}

print(f"User {user\_id} added with publish rights: {can\_publish}.")

def create\_post(self, user\_id: int, content: str, hashtags: list[str], images: list[str]) -> None:

"""

Создает пост. Если пользователь имеет право на публикацию, пост публикуется сразу.

В противном случае пост сохраняется как черновик для одобрения.

:param user\_id: Идентификатор пользователя, создающего пост.

:type user\_id: int

:param content: Текст поста.

:type content: str

:param hashtags: Список хештегов для поста.

:type hashtags: list[str]

:param images: Список путей к изображениям для поста.

:type images: list[str]

"""

post = {

"user\_id": user\_id,

"content": content,

"hashtags": hashtags,

"images": images,

"timestamp": datetime.now()

}

if user\_id in self.\_users and self.\_users[user\_id]["can\_publish"]:

self.\_publish\_post(post)

else:

self.\_drafts.append(post)

print(f"Post by user {user\_id} saved as draft for approval.")

def \_publish\_post(self, post: dict) -> None:

"""

Публикует пост во всех соцсетях (ВК и Telegram).

:param post: Данные поста для публикации.

:type post: dict

"""

self.\_posts.append(post)

print(f"Post published on VK and Telegram: {post['content']}")

print(f"Hashtags: {', '.join(post['hashtags'])}")

print(f"Images: {', '.join(post['images'])}")

def approve\_draft(self, draft\_id: int, approver\_id: int) -> None:

"""

Одобряет черновик для публикации, если пользователь имеет право на одобрение.

:param draft\_id: Индекс черновика в списке черновиков.

:type draft\_id: int

:param approver\_id: Идентификатор пользователя, одобряющего черновик.

:type approver\_id: int

"""

if 0 <= draft\_id < len(self.\_drafts):

if approver\_id in self.\_users and self.\_users[approver\_id]["can\_publish"]:

post = self.\_drafts.pop(draft\_id)

self.\_publish\_post(post)

else:

print(f"User {approver\_id} does not have permission to approve drafts.")

else:

print(f"Draft {draft\_id} not found.")

def list\_posts(self) -> list:

"""

Возвращает список всех опубликованных постов.

:return: Список опубликованных постов.

:rtype: list

"""

return self.\_posts

def list\_drafts(self) -> list:

"""

Возвращает список всех черновиков.

:return: Список черновиков.

:rtype: list

"""

return self.\_drafts

def list\_users(self) -> list:

"""

Возвращает список всех пользователей и их прав на публикацию.

:return: Список пользователей и их прав.

:rtype: list

"""

return [{"user\_id": uid, "can\_publish": data["can\_publish"]} for uid, data in self.\_users.items()]

**Документация DocBook:**

<?xml version="1.0" encoding="UTF-8"?>

<book xmlns="http://docbook.org/ns/docbook" version="5.0">

<title>Описание и сценарии использования проекта KNTista</title>

<chapter>

<title>Краткое описание проекта KNTista</title>

<para>

Проект KNTista разработан для обслуживания нужд профбюро Института компьютерных наук и технологий ДонНТУ. Он представляет собой комплексный инструмент, который упрощает управление внутренними процессами, взаимодействие с студентами и администрирование мероприятий. Основная цель проекта — автоматизация рутинных задач, таких как управление аккаунтами, публикация постов в соцсетях, сбор аналитики и обработка заявок на материальную помощь.

</para>

<para>

Проект состоит из нескольких модулей, каждый из которых отвечает за определённый функционал. Например, модуль авторизации обеспечивает безопасный доступ к системе, а модуль админ-панели позволяет управлять пользователями и аналитикой. Модуль активности помогает отслеживать мероприятия и формировать отчёты, а кросспостер упрощает публикацию постов в социальных сетях.

</para>

<para>

Одной из ключевых особенностей проекта является его модульность. Каждый модуль независим и может быть легко расширен или изменён. Это делает проект гибким и адаптивным под нужды профбюро. Например, модуль GPT-ассистента интегрирован для предоставления быстрых ответов на вопросы студентов, что снижает нагрузку на членов профбюро.

</para>

<para>

Проект также поддерживает несколько способов взаимодействия: через веб-интерфейс и Telegram-бота. Это позволяет пользователям выбирать наиболее удобный для них способ работы с системой. Веб-интерфейс адаптивен и поддерживает работу на различных устройствах, а Telegram-бот предоставляет быстрый доступ к ключевым функциям.

</para>

<para>

В целом, KNTista — это мощный инструмент, который помогает профбюро ИКНТ эффективно управлять своими процессами, улучшать взаимодействие со студентами и автоматизировать рутинные задачи. Проект разработан с учётом современных технологий и требований к безопасности, что делает его надёжным и удобным в использовании.

</para>

</chapter>

<chapter>

<title>Сценарии использования</title>

<section>

<title>Сценарий 1: Управление административной панелью</title>

<para>

В этом сценарии администратор использует модуль <classname>AdminPanel</classname> для управления администраторами и отправки сообщений.

</para>

<programlisting language="python">

# Инициализация модуля AdminPanel

admin\_panel = AdminPanel()

# Получение количества администраторов

print(f"Количество администраторов: {admin\_panel.admins\_count}")

# Отправка сообщения пользователям

admin\_panel.send\_message([1, 2, 3])

</programlisting>

<para>

В результате выполнения кода будет выведено количество администраторов и отправлено сообщение пользователям с идентификаторами 1, 2 и 3.

</para>

</section>

<section>

<title>Сценарий 2: Отслеживание активности студентов</title>

<para>

В этом сценарии используется модуль <classname>Activity</classname> для добавления мероприятий и генерации отчетов.

</para>

<programlisting language="python">

# Инициализация модуля Activity

activity = Activity()

# Добавление мероприятия

activity.add\_event("Хакатон", "2023-10-15", [101, 102, 103])

# Генерация отчета за октябрь 2023 года

report\_filename = activity.generate\_report(10, 2023)

print(f"Отчет сохранен в файле: {report\_filename}")

</programlisting>

<para>

В результате выполнения кода будет добавлено мероприятие "Хакатон" и сгенерирован отчет за октябрь 2023 года.

</para>

</section>

<section>

<title>Сценарий 3: Публикация постов в соцсетях</title>

<para>

В этом сценарии используется модуль <classname>Crossposter</classname> для публикации постов в ВК и Telegram.

</para>

<programlisting language="python">

# Инициализация модуля Crossposter

crossposter = Crossposter()

# Добавление пользователя с правом публикации

crossposter.add\_user(1, True)

# Создание и публикация поста

crossposter.create\_post(1, "Новое мероприятие!", ["#ИКНТ", "#Хакатон"], ["image1.jpg", "image2.jpg"])

# Одобрение черновика (если необходимо)

crossposter.approve\_draft(0, 1)

</programlisting>

<para>

В результате выполнения кода будет создан и опубликован пост с текстом "Новое мероприятие!" и изображениями.

</para>

</section>

</chapter>

</book>

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

Описание и сценарии использования проекта KNTista

1. Краткое описание проекта KNTista

Проект KNTista разработан для обслуживания нужд профбюро Института компьютерных наук и технологий ДонНТУ. Он представляет собой комплексный инструмент, который упрощает управление внутренними процессами, взаимодействие с студентами и администрирование мероприятий. Основная цель проекта — автоматизация рутинных задач, таких как управление аккаунтами, публикация постов в соцсетях, сбор аналитики и обработка заявок на материальную помощь.

Проект состоит из нескольких модулей, каждый из которых отвечает за определённый функционал. Например, модуль авторизации обеспечивает безопасный доступ к системе, а модуль админ-панели позволяет управлять пользователями и аналитикой. Модуль активности помогает отслеживать мероприятия и формировать отчёты, а кросспостер упрощает публикацию постов в социальных сетях.

Одной из ключевых особенностей проекта является его модульность. Каждый модуль независим и может быть легко расширен или изменён. Это делает проект гибким и адаптивным под нужды профбюро. Например, модуль GPT-ассистента интегрирован для предоставления быстрых ответов на вопросы студентов, что снижает нагрузку на членов профбюро.

Проект также поддерживает несколько способов взаимодействия: через веб-интерфейс и Telegram-бота. Это позволяет пользователям выбирать наиболее удобный для них способ работы с системой. Веб-интерфейс адаптивен и поддерживает работу на различных устройствах, а Telegram-бот предоставляет быстрый доступ к ключевым функциям.

В целом, KNTista — это мощный инструмент, который помогает профбюро ИКНТ эффективно управлять своими процессами, улучшать взаимодействие со студентами и автоматизировать рутинные задачи. Проект разработан с учётом современных технологий и требований к безопасности, что делает его надёжным и удобным в использовании.

1. Сценарии использования

Сценарий 1: Управление административной панелью

В этом сценарии администратор использует модуль AdminPanel для управления администраторами и отправки сообщений.

# Инициализация модуля AdminPanel  
admin\_panel = AdminPanel()  
  
# Получение количества администраторов  
print(f"Количество администраторов: {admin\_panel.admins\_count}")  
  
# Отправка сообщения пользователям  
admin\_panel.send\_message([1, 2, 3])

В результате выполнения кода будет выведено количество администраторов и отправлено сообщение пользователям с идентификаторами 1, 2 и 3.

Сценарий 2: Отслеживание активности студентов

В этом сценарии используется модуль Activity для добавления мероприятий и генерации отчетов.

# Инициализация модуля Activity  
activity = Activity()  
  
# Добавление мероприятия  
activity.add\_event("Хакатон", "2023-10-15", [101, 102, 103])  
  
# Генерация отчета за октябрь 2023 года  
report\_filename = activity.generate\_report(10, 2023)  
print(f"Отчет сохранен в файле: {report\_filename}")

В результате выполнения кода будет добавлено мероприятие "Хакатон" и сгенерирован отчет за октябрь 2023 года.

Сценарий 3: Публикация постов в соцсетях

В этом сценарии используется модуль Crossposter для публикации постов в ВК и Telegram.

# Инициализация модуля Crossposter  
crossposter = Crossposter()  
  
# Добавление пользователя с правом публикации  
crossposter.add\_user(1, True)  
  
# Создание и публикация поста  
crossposter.create\_post(1, "Новое мероприятие!", ["#ИКНТ", "#Хакатон"], ["image1.jpg", "image2.jpg"])  
  
# Одобрение черновика (если необходимо)  
crossposter.approve\_draft(0, 1)

В результате выполнения кода будет создан и опубликован пост с текстом "Новое мероприятие!" и изображениями.