Міністерство освіти і науки України
Національний авіаційний університет
Факультет кібербезпеки, комп'ютерної і програмної інженерії
Кафедра комп'ютеризованих систем управління

# Звіт
## з проектно-технологічної практики

студента 3 курсу СП-325 групи
напряму 123 «Комп'ютерна інженерія»
Клокуна Владислава Денисовича

База практики: ТОВ «Смарт Медіа Інвест»

Керівник практики:
від університету — старший викладач Кашкевич Іван Фуркатович
від бази практики — керівник проектів Чезганов Олександр Сергійович

Київ 2019

Підстава для проходження практики: робочий навчальний план № РС-4-6.05010202/12 підготовки фахівців за напрямом підготовки 6.050102 «Комп'ютерна інженерія», договір на проведення практики між університетом та ТОВ «Смарт Медіа Інвест», наказ ректора № 687/ст від 10.04.2019.

Термін проходження практики: з 03.06.2019 по 23.06.2019

Індивідуальні завдання: розробити програмний продукт — чат-бота для платформи Telegram.

ВІДМІТКА
про проходження практики

Прибув на базу практики
«_____» _____ 2019 р. (підпис) _____ М. П.

Вибув з бази практики
«_____» _____ 2019 р. (підпис) _____ М. П.

КАЛЕНДАРНИЙ ПЛАН-ГРАФІК ПРАКТИКИ

| № п/п | Об'єкт практики та види робіт | Термін виконання | |
|---|---|---|---|
| | | початок | закінчення |
| 1 | Знайомство з інфраструктурою і забезпеченням організації | 03.06.2019 | 05.06.2019 |
| 2 | Постановка і аналіз задачі. Вибір необхідних інструментів для розробки. Затвердження специфікації | 06.06.2019 | 07.06.2019 |
| 3 | Розробка програмного продукту | 10.06.2019 | 14.06.2019 |
| 4 | Розробка рішення для розгортання програми | 17.06.2019 | 18.06.2019 |
| 5 | Тестування і запуск розробленого програмного продукту | 19.06.2019 | 21.06.2019 |

Керівник практики від університету: _____

## Зміст

**1 Характеристика бази практики** **4**
1.1 Загальні відомості про організацію . . . . . 4
1.2 Сфера діяльності . . . . . 4

**2 Зміст та результати виконаних робіт** **5**
2.1 Знайомство з інфраструктурою і забезпеченням організації . . . 5
2.2 Постановка і аналіз задачі . . . . . 7
2.3 Вибір інструментів і затвердження специфікації . . . . . 8
2.3.1 Взаємодія з платформою Telegram . . . . . 8
2.3.2 Інструменти веб-додатку . . . . . 10
2.3.3 Результати . . . . . 10
2.4 Розробка програмного продукту . . . . . 11
2.4.1 Підготовка середовища розробки . . . . . 11
2.4.2 Аналіз і підготовка вхідних даних . . . . . 12
2.4.3 Розробка моделей для записів бази даних . . . . . 13
2.4.4 Розробка модулів керування моделями даних . . . . . 16
2.4.5 Розробка модуля чат-бота . . . . . 17
2.4.6 Результати . . . . . 19
2.5 Розробка рішення для розгортання продукту . . . . . 20
2.6 Тестування і запуск розробленого програмного продукту . . . . . 23

# 1. Характеристика бази практики

## *1.1. Загальні відомості про організацію*

Організація ТОВ «Смарт Медіа Інвест» зареєстрована за адресою 47201, Тернопільська область, Зборівський район, місто Зборів, вулиця Б. Хмельницького. За класифікатором видів економічної діяльності вона є рекламним агентством, займається створенням, запуском, супроводом, просуванням і підтримкою медіапроектів у соціальних мережах на кшталт «Вконтакте», «Instagram» і «Telegram».

## *1.2. Сфера діяльності*

Щоб підтримувати медіапроект, необхідно збирати, створювати, оформлювати і публікувати інформацію, на якій він спеціалізується, розважати користувачів проекту, а також тримати його у належному стані. Для досягнення цих цілей і покращення результатів використовують різні інструменти: для розваги користувачів — мультимедійні матеріали і інтерактивні заходи, для підтримки проекту — спеціальні програмні продукти.

За кожну зі складових відповідають певні структурні підрозділи, які тісно взаємодіють між собою. Пошуком і створенням інформації, розробкою ідей, креативу, а також інтерактивних заходів займаються редактори. Мультимедійні матеріали для оформлення розроблених ідей надають дизайнери. За створення і запуск інструментів, необхідних для організації інтерактивних розважальних заходів і підтримки медіапроектів, відповідають розробники. Крім цього, розробники відповідають за налаштування і підтримку апаратно-програмної інфраструктури, яка необхідна для правильної роботи вже запущених продуктів, у справному стані.

Інструменти, створенням і запуском яких займаються розробники, можна класифікувати так:

- розважальні інструменти:
  - чат-боти,
  - боти-коментатори,
- супровідні інструменти:
  - очищувачі,
  - спеціалізовані сценарії (живі обкладинки тощо),
  - інструменти збору статистики.

Щоб зрозуміти, що входить в обов'язки розробників, розглянемо кожен клас цих інструментів детальніше. *Розважальні інструменти* призначені для того, щоб розважити користувача, безпосередньою взаємодією з ним. Прикладом розважального інструменту є *чат-бот* — програмний продукт, який веде діа-

лог (чат) з користувачем відповідно до заданого сценарію або специфікації. Наприклад, за допомогою чат-бота можна реалізувати текстову пригоду, головним героєм якого буде кожен окремий користувач, організувати лотерею або віртуального помічника, який відповідатиме на питання користувача.

*Бот-коментатор* — це програмний продукт, який додає коментарі у спеціальних дошках для обговорення у соціальних мережах залежно від певної умови. Наприклад, якщо користувач залишає коментар до запису про товар, в якому питає про деталь, відому боту-коментатору, цей бот може залишити коментар з відповіддю на поставлене питання.

Наступною категорією є *супровідні інструменти*, тобто інструменти, призначені для автоматизованої підтримки представництв медіапроекту у певному стані. Першим представником супровідних інструментів є *очищувач* — програмний продукт який очищує ту чи іншу складову присутності медіапроекту у соціальній мережі: записи на дошці обговорень та її коментарі, зміст фото-, аудіо- і відеоальбомів тощо.

*Спеціалізований сценарій* (або *скрипт*) — це програмний продукт, який виконує різноманітні вузькоспеціалізовані дії в залежності від свого призначення. Поширеним прикладом спеціалізованого сценарію є «жива обкладинка» — програмний продукт, який змінює обкладинку представництва медіапроекту у соціальній мережі в залежності від певних параметрів: кількості записів, створених користувачами, проведених дій, переглядів тощо.

*Інструмент збору статистики* — це програмний продукт, який збирає статистику різних представництв медіапроекту у соціальних мережах, оброблює зібрані дані та експортує їх у потрібний формат. Результати роботи цих інструментів використовуються для аналізу, відстеження і планування стану та життєздатності проекту, а також для звітності.

Отже, як бачимо, компанія займається повним циклом управління медіапроектами. Для цього вона використовує набір різноманітних інструментів, до яких також входять програмні інструменти, за створення і запуск яких відповідає підрозділ розробників, в якому автор проходив практику.

## 2. Зміст та результати виконаних робіт

### *2.1. Знайомство з інфраструктурою і забезпеченням організації*

Організована у компанії інфраструктура для розробки програмних рішень складається з апаратного і програмного забезпечення, а також віддалених сервісів. Віддалені (або *хмарні*) сервіси використовуються для координації команди і збереження її напрацювань. Організація використовує такі сервіси:

— хостинг для репозитаріїв початкового коду GitHub;
— система організації документообігу Google Suite: Docs, Sheets і Slides;

— дошка завдань Trello.

Хостинг для репозитаріїв початкового коду GitHub потрібен, щоб зберігати і відстежувати версії початкового коду усіх програмних продуктів і інструментів, які створюються і використовуються в організації. Система організації документообігу Google Suite використовується для звітності та обміну документами між працівниками компанії. На дошці завдань Trello відстежують завдання, які зараз поставлені перед працівниками, і оцінюють їх завантаженість, щоб планувати наступні дії.

До апаратного забезпечення входить виділений сервер, розташований за межами компанії, на якому встановлене програмне забезпечення, необхідне для розробки продуктів, якими користується організація, зокрема:

— операційна система GNU/Linux Ubuntu LTS;
— веб-сервер Nginx;
— система керування базами даних MongoDB;
— інтерпретатор мови програмування Python;
— система управління контейнерами Docker.

Розглянемо вищезазначене програмне забезпечення детальніше. Операційна система GNU/Linux Ubuntu LTS дозволяє встановити всі засоби, необхідні для правильної роботи потрібних продуктів, наприклад, Docker Server Community Edition. На робочій машині встановлена серверна версія дистрибутиву, що дозволяє зменшити обсяг пам'яті, який займає система, бо в ній встановлюється лише набір мінімально необхідних компонентів. Позначка LTS (Long-Term Support) означає, що ця версія дистрибутиву буде підтримуватись тривалий час, тобто в разі виникнення проблем з системою можна розраховувати на необхідну підтримку протягом декількох років з моменту випуску даної версії.

На операційній системі встановлений веб-сервер Nginx, необхідний для запуску програмних продуктів, які надають послуги в мережі Інтернет, зокрема веб-додатків та веб-сайтів. На одному сервері водночас можуть працювати одразу декілька продуктів, тому веб-сервер виступає як зворотний проксі (англ. *reverse proxy*), тобто отримує запити від зовнішніх клієнтів і перенаправляє їх на інші, локальні веб-сервери.

Система керування базами даних MongoDB використовується для збереження і управління постійними відносно структурованими даними, які створюються під час роботи веб-додатків.

Основною мовою програмування для розробки продуктів в організації є Python, тому на сервері встановлений її інтерпретатор. З його допомогою розробники створюють модулі та пакети на мові програмування Python і запускають їх.

Щоб зручно розгортати продукти і запускати їх в ізольованому середовищі, яке можна точно відтворити на будь-якій сумісній платформі, використовують систему управління контейнерами Docker.

### *2.2. Постановка і аналіз задачі*

Для проходження проектно-технологічної практики була поставлена така задача: розробити чат-бота для платформи «Telegram», який надсилає казки своїм користувачам. Коли користувач запускає бота, він повинен привітати його і надіслати випадкову казку. Якщо користувач просить надіслати ще, бот надсилає наступну випадкову казку. Також, бот повинен регулярно розсилати казки своїм користувачам: щодня в певний час.

Щоб наочно зобразити високорівневу функціональність бота, була розроблена UML-діаграма варіантів використання чат-бота (рис. 1).

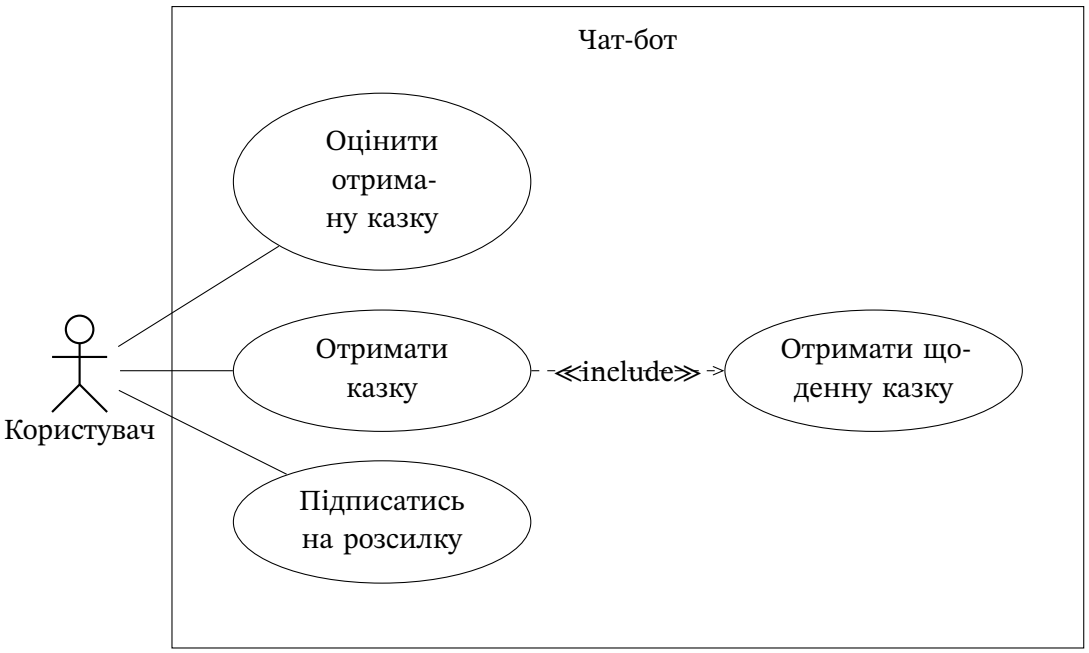


Рис. 1: Діаграма варіантів використання чат-бота

Розроблена діаграма варіантів використання допомогла затвердити вимоги до програмного продукту з працівниками, які розробляли ідею даного проекту, і знадобилась під час розробки архітектури програмного продукту.

Щоб проаналізувати задачу, розділимо її на підзадачі. В результаті декомпозиції отримаємо таку ієрархію:

1. Регулярно надсилати випадкову казку.
2. Обробити запит користувача:
   2.1. Визначити тип запиту.
   2.2. Додати користувача в список активних користувачів чат-бота.
   2.3. Дати відповідь на запит:
      2.3.1. Надіслати повідомлення-привітання.

2.3.2. Надіслати повідомлення з інструкцією, як користуватись чат-ботом.
2.3.3. Обрати випадкову казку.
2.3.4. Надіслати текст обраної казки.
2.3.5. Додати казку у список казок, що подобаються.

## *2.3. Вибір інструментів і затвердження специфікації*

Як відомо зі знайомства з інфраструктурою компанії, основною мовою програмування для створення продуктів є Python, тому для розробки чат-бота буде використана саме вона.

Розглянемо діаграму варіантів використання (рис. 1), з якої видно, що при взаємодії з системою (Чат-ботом) актор (Користувач) може виконувати такі дії:

— Підписатись на розсилку, тобто дати згоду на те, щоб бот працював з користувачем, який до нього звертається.
— Отримати казку, тобто надіслати запит чат-боту, щоб він надіслав користувачу казку.
— Отримати щоденну казку, яка є розширенням варіанту використання «Отримати казку». Якщо користувач підписався на розсилку, бот надсилає щоденну казку за заданим розкладом.
— Оцінити отриману казку, тобто мати змогу виразити, що казка сподобалась користувачу.

### 2.3.1. Взаємодія з платформою Telegram

Основною можливістю є взаємодія з чат-ботом. Він повинен працювати на платформі Telegram, тож необхідно використати її програмний інтерфейс (англ. *application programming interface, API*). Програмний інтерфейс Telegram, а тому і робота більшості чат-ботів побудована на основі двох дій: отримання і обробки оновлень та постановки запитів до серверів Telegram API.

Отримання оновлень дозволяє зрозуміти, які дії користувачі або інші треті сторони проводять з ботом. Наприклад, які повідомлення надсилають, які кнопки натискають та інше. Запити дозволяють визначити, які дії повинен провести сам чат-бот: яке повідомлення відправити у відповідь, яку інформацію отримати або оновити тощо.

Щоб організувати взаємодію з платформою Telegram, скористаємось зовнішнім пакетом «Python Telegram Bot» (знаходиться за адресою `https://github.com/python-telegram-bot/python-telegram-bot`), який надає модулям, написаним на мові програмування Python зручний доступ до інтерфейсу програмування платформи Telegram. Цей модуль дозволяє отримувати оновлення за допомогою двох методів: довгого опитування (англ. *long polling*) і веб-

перехоплення (англ. *webhook*).

Метод *довгого опитування* — це pull-технологія (тобто спочатку клієнт робить запит, а потім сервер відповідає), при використанні якої клієнт опитує сервер з певним інтервалом. Однак, на відміну від традиційного опитування, клієнт очікує, що сервер може відповісти не одразу. Натомість, якщо сервер не має оновлень для клієнта, він не відсилає пусту відповідь, а чекає, поки з'явиться оновлення або закінчиться період опитування, і лише тоді надсилає відповідь і закінчує цикл «запит—відповідь».

Перевагами методу довгого опитування є:

1. Можливість працювати на будь-якому пристрої, здатному робити запити до веб-серверів.
2. Менша затримка на відповідь (тобто між моментами, коли інформація з'явилась і коли клієнт надіслав запит на її отримання) порівняно зі звичайним опитуванням.
3. Простота реалізації.

Його недоліком є більша потреба в обчислювальних і мережевих ресурсах, а отже менша продуктивність і швидкодія додатків, основаних на ньому.

Отже, техніку довгого опитування можна досить просто реалізувати на більшості пристроїв, підключених до мережі, і вона дозволяє досить швидко отримувати оновлення, однак, під великим навантаженням вона потребуватиме велику кількість обчислювальних і мережевих ресурсів.

Метод *веб-перехоплення* — це push-технологія (тобто сервер розпочинає обмін даними і надсилає повідомлення без попередього запиту), при використанні якої сервер надсилає запит за вказаною адресою, як тільки з'являється оновлення. Метод називається веб-перехопленням, тому що за адресою, куди надсилається запит, знаходиться веб-сервер, який його перехоплює, оброблює і, за потреби, дає відповідь.

Такий метод має серйозні переваги в області швидкодії, а саме:

1. Найменшу затримку на відповідь серед усіх методів.
2. Мінімальне навантаження на обчислювальні і мережеві ресурси.

Однак, висока продуктивність методу потребує компромісу і має суттєвий недолік: метод веб-перехоплення важче реалізувати порівняно з іншими, оскільки для цього потрібні такі складові:

— сервер зі статичною IP-адресою, який може отримувати запити з мережі Інтернет;
— веб-сервер з підтримкою протоколу TLS 1.0 і вище;
— відкриті порти 443, 80, 88 або 8443.
— SSL-сертифікат безпеки.

Отже, веб-перехоплення дозволяє отримувати оновлення одразу ж, без потреби так чи інакше опитувати сервер, і економить обчислювальні ресурси. Проте, йому необхідно багато сторонніх ресурсів, що ускладнює реалізацію,

тому використовувати метод веб-перехоплення рекомендують виключно тоді, коли в цьому є реальна потреба [1].

У завданні не передбачена вимога високої доступності та можливість витримати високе навантаження, крім того, апаратне забезпечення достатньо швидке, щоб витримати відносно велику кількість користувачів, тому для реалізації чат-бота був обраний метод довгого опитування.

### 2.3.2. Інструменти веб-додатку

Щоб кожен день розсилати казки користувачам, які звернулись до бота і підписались на розсилку, необхідно вести їх облік. Хорошою практикою при розробці веб-додатків є принцип відсутності стану: дії, які виконує програма — вебдодаток, не повинні залежати від її внутрішнього стану. Отже, облікові дані користувачів варто зберігати зовні, за логічними межами програми.

Так як облікові дані користувачів будуть часто змінюватись (будуть з'являтись нові користувачі, оцінки тощо), для збереження цих даних зручно використати базу даних. В організації вже використовують систему керування базами даних MongoDB, тому для вирішення поставленої задачі використаємо її.

Щоб отримати доступ до бази даних з Python-модуля, використаємо пакет PyMongo, який надає офіційний розробник системи керування базами даних MongoDB.

### 2.3.3. Результати

На основі дослідженої інформації та проведених порівнянь для розробки програмного продукту були обрані необхідні інструменти (табл. 1).

Табл. 1: Обрані інструменти

| Призначення | Обраний інструмент |
|---|---|
| Мова програмування | Python |
| Взаємодія з платформою Telegram | Пакет Python Telegram Bot з методом довгого опитування |
| Керування базами даних | Система керування базами даних MongoDB |
| Доступ до керування базами даних з програмного продукту | Модуль PyMongo |

Обравши інструменти, переходимо до затвердження вимог і специфікації, тобто за створеною діаграмою варіантів використання (рис. 1), декомпозицією задач (підрозділ 2.2) і переліком обраних інструментів (табл. 1) вносимо корективи та затверджуємо, які функції має виконувати кінцевий продукт.

## *2.4. Розробка програмного продукту*

### 2.4.1. Підготовка середовища розробки

Перш за все, створюємо файлову структуру проекту: створюємо директорії, яка буде містити проект, переходимо в неї і створюємо необхідні субдиректорії. Для цього у терміналі Bash виконуємо таку команду:

```
mkdir storyteller && cd storyteller && mkdir assets scripts
↪  storyteller
```

Тепер встановимо обрані інструменти і переконаємось, що продукт працюватиме саме з ними та їх зазначеними версіями. Для цього створюємо віртуальне середовище для програмного продукту:

```
python3 -m venv venv
```

Створивши віртуальне середовище, активуємо його за допомогою команди:

```
source venv/bin/activate
```

Тепер ми знаходимось у віртуальному середовищі, яке місттить власну версію інтерпретатора Python, його утиліт та бібліотек, потрібних для його роботи.

Встановимо обрані інструменти. Система управління базами даних уже встановлена і запущена на сервері, тому залишається встановити Python-пакети для розробки програмного продукту. Для цього знадобиться утиліта `pip`, яка дозволяє встановлювати пакети з ресурсу Python Package Index (PyPI). З її допомогою встановлюємо обрані пакети, запустивши таку команду:

```
pip install -U python-telegram-bot pymongo
```

Очевидно, що для запуску продукту необхідні сторонні бібліотеки, які ми встановили. Щоб зробити використання продукту зручнішим, прийнято наводити версії пакетів у спеціальному файлі `requirements.txt`. Отже, треба створити цей файл і записати в нього версії встановлених пакетів. Для цього виконуємо таку команду:

```
pip freeze > requirements.txt
```

Тепер середовище налаштоване і можна переходити безпосередньо до розробки програмного продукту.

### 2.4.2. Аналіз і підготовка вхідних даних

Вхідними даними для чат-бота є казки. Казки були надані у форматі текстового файлу з розширенням «.txt» і складаються з декількох частин: ідентифікатора (порядкового номера), назви (заголовка) і змісту. Так як необхідно вести облік, скільком користувачам сподобалась певна казка, також необхідно зберігати список людей, які оцінили її.

Ці вхідні дані необхідно додати у базу даних казок MongoDB, тому розробимо модуль, який перетворює казки у наданому форматі у формат, який підходить для запису в MongoDB — формат документів BSON (лістинг 2.1).

Лістинг 2.1: Файл `tale_db_txt_to_json.py`: модуль для підготовки вхідних даних

```
import argparse
import json


def main(args):
    with open(args.infile, encoding='utf8') as ifile:
        # Load entire database file into memory
        data = ifile.read()

    # Tales are separated by two newlines, so split the file by two newlines
    data = data.split('\n\n')

    # Build an array of tales
    tales = []
    for tale in data:
        # To separate tale header and contents, split by the first newline
        res = tale.split(sep='\n', maxsplit=1)
        # Header is in format '1. <Header>', so split by '. ' to get at most
        # 2 items: Tale ID and Tale Header
        tale_id, header = res[0].split(sep='. ', maxsplit=1)
        # Contents are the second element of the first split
        contents = res[1]
        tales.append(
            {
                'tale_id': tale_id,
                'liked_by': [],  # initialize like count to 0
                'header': header,
                'contents': contents
            }
        )
```

```
31
32      print(tales)
33      with open(args.outfile, 'w', encoding='utf-8') as outfile:
34          json.dump(
35              tales,
36              fp=outfile,
37              indent=2,
38              ensure_ascii=False
39          )
40
41
42  if __name__ == '__main__':
43      p = argparse.ArgumentParser()
44      p.add_argument(
45          '--infile',
46          help='Name of the tale database TXT format'
47      )
48
49      p.add_argument(
50          '--outfile',
51          help='Name of the tale database in JSON format'
52      )
53
54      args = p.parse_args()
55      main(args)
```

Розробивши модуль, перетворюємо наданий опис казок (файл `tale_db.txt`) у файл транзакцій для MongoDB, виконуючи таку команду:

```
python3 tale_db_txt_to_json.py --infile tale_db.txt --outfile
↪  tales.json
```

В результаті виконання цієї команди буде створений файл `tales.json`, в якому буде міститись транзакція для запису в базу даних MongoDB.

### 2.4.3. Розробка моделей для записів бази даних

Передбачено, що у базі даних зберігатимуться казки і дані про користувачів, які користуються чат-ботом. Щоб зберігати і зручно керувати цими записами, використаємо шаблон Model–View–Controller (MVC), який передбачає наявність таких компонентів: модель даних, представлення та модуль керування.

Перш за все, розробимо моделі даних для казок (лістинг 2.2) і користувачів (лістинг 2.3). Модель даних для казки передбачає доступ до її атрибутів: ідентифікатора, заголовку, тексту і списку людей, яким вона сподобалась. Крім того, вона передбачає набір методів, які дозволяють отримати або змінити дані про

них. Модель даних про користувачів передбачає доступ до єдиного атрибуту — ідентифікатора користувача, а також можливості додати нового користувача, а також отримати список усіх користувачів.

Лістинг 2.2: Файл `tales.py`: модель даних про казки

```
class Tale(object):
    DBNAME = 'storyteller'
    COLNAME = 'tales'

    def __init__(self, client):
        self.db = client[TaleModel.DBNAME]
        self.col = self.db[TaleModel.COLNAME]

    def get_random(self):
        """ Gets a random tale document from the database """
        rand_cursor = self.col.aggregate(
            # Query the current collection using a pipeline
            [{'$sample': {'size': 1}}]  # Random sample of 1
        )
        # Try to convert the cursor to a list
        res_list = list(rand_cursor)
        if not res_list:
            return None

        rand_tale = res_list[0]

        return rand_tale

    def get_by_id(self, tale_id):
        tale_query = {'tale_id': tale_id}
        tale = self.col.find_one(tale_query)

        return tale

    def like_id(self, tale_id, user_id):
        tale = self.get_by_id(tale_id)

        # To avoid duplicates, convert the list of likers to a set
        liked_by = set(tale['liked_by'])

        # Add a new liker
        liked_by.add(user_id)

        # MongoDB uses BSON and BSON does not accept sets, only lists, so
        # convert back to a list
        liked_by = list(liked_by)
```

```
42
43          # Push the update
44          self.col.update(
45              {'tale_id': tale_id},
46              {'$set': {'liked_by': liked_by}}
47          )
48
49      def remove_like_id(self, tale_id, user_id):
50          tale = self.get_by_id(tale_id)
51
52          # to avoid duplicates, convert the list of likers to a set
53          liked_by = set(tale['liked_by'])
54
55          # remove a like from user
56          try:
57              liked_by.remove(user_id)
58          except KeyError:
59              return
60
61          # MongoDB uses BSON and BSON does not accept sets, only lists, so
62          # convert back to a list
63          liked_by = list(liked_by)
64
65          # Push the update
66          self.col.update(
67              {'tale_id': tale_id},
68              {'$set': {'liked_by': liked_by}}
69          )
```

Лістинг 2.3: Файл `user.py`: модель даних про користувачів

```
1   class User(object):
2       DBNAME = 'storyteller'
3       COLNAME = 'users'
4
5       def __init__(self, client):
6           self.db = client[User.DBNAME]
7
8       def get_ids_iter(self):
9           collection = self.db[User.COLNAME]
10          for user in collection.find({}):
11              yield user['user_id']
12
13      def add_user(self, user_id):
14          collection = self.db[User.COLNAME]
15          try:
```

```
16              res = collection.insert_one({'user_id': user_id})
17          except pymongo.errors.DuplicateKeyError:
18              res = None
19          return res
```

Тепер, коли моделі даних розроблені, можна переходити до розробки модулей керування ними.

### 2.4.4. Розробка модулів керування моделями даних

Модулі керування дають можливість зручно керувати записами у базах даних, які представлені за допомогою моделей. Перевагою такого підходу є можливість змінювати лише модуль управління, якщо необхідно змінити запис або додати нову функцію для взаємодії з даними. При цьому, якщо модель даних вичерпно описує дані, її не потрібно змінювати, щоб додати нову функцію.

Отже, створюємо модулі керування для моделей даних про казки (лістинг 2.4) і про користувачів (лістинг 2.5).

Лістинг 2.4: Файл `tales.py`: модуль керування моделлю даних про користувачів

```
1  class TaleController(object):
2      def __init__(self, tale_model):
3          self.model = tale_model
4          self.summary = None
5
6      @property
7      def header(self):
8          return self.summary['header']
9
10     @property
11     def contents(self):
12         return self.summary['contents']
13
14     @property
15     def id(self):
16         return self.summary['tale_id']
17
18     @property
19     def liked_by(self):
20         return self.summary['liked_by']
21
22     @property
23     def likes(self):
```

```
24          return len(self.liked_by)
25
26      def refresh(self):
27          self.load_by_id(self.id)
28
29      def load_random(self):
30          self.summary = self.model.get_random()
31
32      def load_by_id(self, tale_id):
33          self.summary = self.model.get_by_id(tale_id)
34
35      def add_like(self, user_id):
36          self.model.like_id(self.id, user_id)
37          self.refresh()
38
39      def remove_like(self, user_id):
40          self.model.remove_like_id(self.id, user_id)
41          self.refresh()
```

Лістинг 2.5: Файл `user.py`: модуль керування моделлю даних про користувачів

```
1  class UserController(object):
2      def __init__(self, model):
3          self.model = model
4
5      def new_user(self, user_id):
6          self.model.add_user(user_id)
7
8      def get_all_user_ids(self):
9          for user_id in self.model.get_ids_iter():
10             yield user_id
```

Після створення модулів керування для описаних моделей даних їх можна зручно використовувати, щоб додавати і оновлювати записи у базі даних. Таким чином, залишається остання ланка у моделі MVC — представлення.

### 2.4.5. Розробка модуля чат-бота

Представленням для даних буде сам чат-бот: він буде отримувати інструкції від користувача і представлятиме йому дані. Бот побудований на принципі обробників — об'єктів, які перевіряють, чи потрібно їм працювати, і якщо так, то вони виконують свої функції. Обробники реалізовані за допомогою засобів пакету Python Telegram Bot. Цими засобами є об'єкти класів, які наслідують клас `telegram.ext.Handler`, а саме об'єкти таких класів:

1. `telegram.ext.CommandHandler`
2. `telegram.ext.MessageHandler`
3. `telegram.ext.CallbackQueryHandler`

ОБРОБКА КОМАНД Об'єкти класу `telegram.ext.CommandHandler` призначені для обробки команд формату `/<команда>`, які отримує чат-бот. Прототип об'єктів цього класу такий:

```
class CommandHandler(command, callback, filters=None,
    allow_edited=None, pass_args=False,
    pass_update_queue=False, pass_job_queue=False,
    pass_user_data=False, pass_chat_data=False)
```

Основними параметрами прототипу є `command`, `callback`. Параметр `command` повинен містити текст команди, яку необхідну обробити. Наприклад, щоб обробити команду `/start`, необхідно ініціалізувати об'єкт так:

```
telegram.ext.CommandHandler(command='start', callback)
```

Параметр `callback` повинен містити об'єкт, який можна і треба викликати для обробки команди. Наприклад, щоб при обробці команди `/start` викликати функцію `h_start`, необхідно ініціалізувати об'єкт так:

```
def h_start(update, context):
    pass
telegram.ext.CommandHandler(command='start', callback=h_start)
```

ОБРОБКА ПОВІДОМЛЕНЬ Об'єкти класу `telegram.ext.MessageHandler` призначені для обробки текстових повідомлень, які отримує чат-бот. Прототип об'єктів цього класу такий:

```
class MessageHandler(filters, callback,
    pass_update_queue=False, pass_job_queue=False,
    pass_user_data=False, pass_chat_data=False,
    message_updates=None, channel_post_updates=None,
    edited_updates=None)
```

Основними параметрами прототипу є `filters`, `callback`. Параметр `filters` повинен містити фільтри (об'єкти класу `telegram.ext.BaseFiler`), що описують повідомлення, яке треба обробити. Наприклад, щоб обробити повідомлення, яке містить текст «Привіт», можна ініціалізувати об'єкт так:

```
MessageHandler(
    Filters.regex('^(Привіт)'),
    callback
)
```

Як і в інших обробниках, параметр `callback` повинен містити об'єкт, який можна і треба викликати для обробки повідомлення.

ОБРОБКА ЗАПИТІВ Об'єкти класу `telegram.ext.CallbackQueryHandler` призначені для обробки запитів від натискання кнопок, які отримує чат-бот. Прототип об'єктів цього класу такий:

```
class CallbackQueryHandler(callback, pass_update_queue=False,
↪  pass_job_queue=False, pattern=None, pass_groups=False,
↪  pass_groupdict=False, pass_user_data=False,
↪  pass_chat_data=False)
```

Основним параметром цього об'єкта є параметр `callback`, який, як і в інших обробниках, повинен містити об'єкт, який можна і треба викликати для обробки запиту.

ЩОДЕННА РОЗСИЛКА Також, важливою функцією є періодична розсилка повідомлень. Для цього використовується черга завдань, представлена об'єктом класу `telegram.ext.JobQueue`. У цей об'єкт можна додавати завдання, які виконуються один раз або з певною періодичністю (наприклад, щодня). Припустимо, що ініціалізований об'єкт `jq` класу `telegram.ext.JobQueue`. Тоді додати щоденне завдання до нього можна так:

```
import datetime
jq.run_daily(
    callback,
    time=datetime.time(12, 00)
)
```

Отже, вся функціональність чат-бота будується на основі вищезазначених об'єктів і спеціально написаних обробників. Робота бота відбувається за допомогою об'єкта `bot.Storyteller`, який в свою чергу працює з платформою Telegram за допомогою об'єкта класу `telegram.ext.Dispatcher` — диспетчера, який обмінюється даними з серверами Telegram API.

### 2.4.6. Результати

Під час розробки програмного продукту був створений чат-бот, який відповідає поставленим вимогам:

- надсилає користувачам, які до нього звернулись, випадкові казки з бази даних;
- періодично розсилає казки користувачам;
- підтримує оцінку казок.

У процесі створення чат-бота додавались нові файли, тому в результаті сформувалась нова файлова структура проекту (рис. 2).

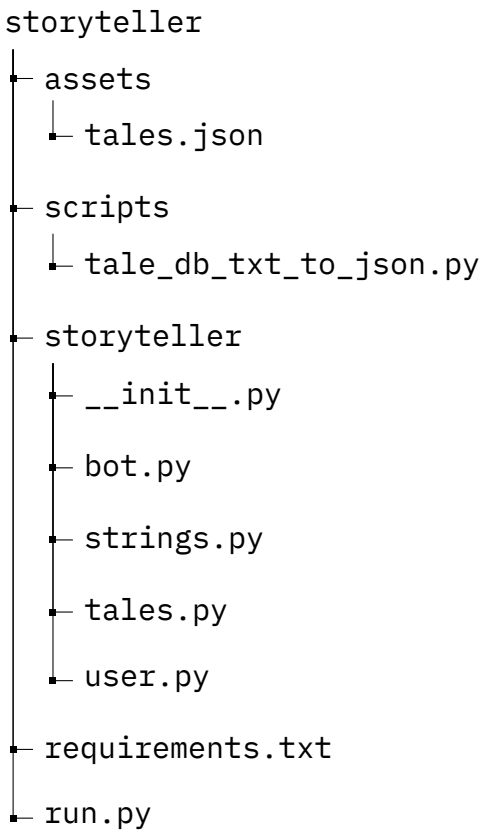


Рис. 2: Файлова структура проекту

Створивши необхідний продукт, можна переходити до розробки рішення для його розгортання і запуску.

### *2.5. Розробка рішення для розгортання продукту*

Розгортання продукту — це послідовність дій, яка передбачає його налаштування і запуск для кінцевих користувачів. Основними задачами при розгортанні продукту є можливість швидкого налаштування і запуску на максимальній кількості обчислювальних машин різних конфігурацій.

Щоб вирішити ці задачі і розгорнути розроблений чат-бот, скористаємось платформою контейнеризації Docker, яка входить в інфраструктуру компанії. Ця технологія дозволяє використати віртуалізацію на рівні операційної системи, щоб запускати програмне забезпечення в ізольованому середовищі.

Такі ізольовані середовища запуску створюються з об'єктів платформи Docker, які називаються образами. *Образи* — це файли, які містять усі необхідні складо-

ві для запуску бажаного програмного забезпечення. Щоб створити образ, перш за все необхідно створити файл, який описує процес його створення. Такий файл називається `Dockerfile`. Розробимо відповідний файл для створеного чат-бота (лістинг 2.6).

Лістинг 2.6: `Dockerfile` для створення образу чат-бота

```
# Use official Python distro
FROM python:3.6-alpine

# Add a priviledged user who is not root inside the container
# This creates the /home/storyteller directory inside the
# container
RUN adduser -D storyteller

# Install build dependencies for Python telegram bot
RUN apk add build-base python-dev libffi-dev openssl-dev

# Set the containter's working directory to /storyteller
WORKDIR /home/storyteller

# Copy requirements.txt file to container
COPY requirements.txt requirements.txt
# Set up a venv inside the container
RUN python -m venv venv

# Install app requirements
RUN venv/bin/pip install -r requirements.txt

# Copy application contents to container
COPY assets assets
COPY scripts scripts
COPY storyteller storyteller
COPY run.py run.py
COPY run_storyteller.sh run_storyteller.sh

# Make `run_storyteller.sh` script executable
RUN chmod +x run_storyteller.sh

# Recursively change owner of files in current directory to user
# "storyteller"
RUN chown -R storyteller:storyteller ./
USER storyteller

ENTRYPOINT ["./run_storyteller.sh"]
```

Розробивши `Dockerfile` для створеного чат-бота, створюємо його образ.

Для цього за допомогою інтерфейсу командного рядка платформи Docker запускаємо створення образу «storyteller» за файлом у поточній директорії, виконавши таку команду:

```
docker build -t storyteller .
```

Після завершення виконання команди буде створений образ «storyteller:latest». Зі створеного образу можна створити ізольоване середовище, в якому буде запущений чат-бот. Таке середовище називається *контейнером*.

Однак, чат-бот — не єдина складова цілого програмного продукту. Для його правильної роботи також необхідна база даних, тому одного лише контейнера чат-бота буде недостатньо. Отже, необхідно контейнеризувати систему керування базами даних і об'єднати їх у багатосервісний додаток.

Для цього використаємо програмний продукт Docker Compose, який входить до складу платформи Docker. Щоб створити бажаний багатосервісний додаток, необхідно описати його у спеціальному файлі `docker-compose.yml`. У цьому файлі описують сервіси, які використовує багатосервісний додаток, а також параметри їх запуску.

Щоб зробити розгортання програмного продукту зручнішим, додамо конфігураційний файл, який описуватиме специфіку роботи розробленого чат-бота за допомогою змінних середовища (лістинг 2.7).

Лістинг 2.7: Файл `release.env`, який містить конфігурацію для чат-бота

```
STORYTELLER_TOKEN=YOUR_TELEGRAM_BOT_TOKEN
STORYTELLER_MONGO_URI=mongodb://mongo:27017/
STORYTELLER_MONGO_DBNAME=storyteller
STORYTELLER_TALES_PATH=./assets/tales.json
STORYTELLER_MAILING_HH=12
STORYTELLER_MAILING_MM=00
```

Після створення файлу, який описує конфігураційні змінні середовища, створюємо необхідний файл (лістинг 2.8) з описом багатосервісного додатку.

Лістинг 2.8: Файл `docker-compose.yml`, який описує необхідний багатосервісний додаток

```
version: "3"
services:
  # The bot itself
  storyteller:
    build: .
    depends_on:
```

```
7         - mongo
8       # MessageQueue prevents the bot from stopping gracefully, so time out
9       # after 30 seconds
10      stop_grace_period: 30s
11      env_file:
12        - release.env
13      entrypoint: ./run_storyteller.sh
14
15    mongo:
16      image: "mongo:4"
17      restart: always
18      environment:
19        # Set data directory inside the container for MongoDB
20        DATA_DIR: "/data/db"
21      volumes:
22        - "mongodata:/data/db"
23
24  volumes:
25    mongodata:
```

Створивши образ чат-бота і описавши багатосервісний додаток, його можна зручно розгортати за допомогою програми Docker Compose. Для цього треба перейти у директорію з файлом `docker-compose.yml` і виконати таку команду:

```
sudo docker-compose up -d
```

В результаті виконання цієї команди у фоновому режимі буде запущений багатосервісний додаток, який ми описали, тобто всі складові, необхідні для правильної роботи чат-бота.

### *2.6. Тестування і запуск розробленого програмного продукту*

## Література

1. *Höke J.*, *Kasim R.*, *Pathak A.* Python Telegram Bot Wiki : Webhooks. — 31.12.2017. — URL: `https://github.com/python-telegram-bot/python-telegram-bot/wiki/Webhooks` (дата зверн. 15.06.2019).