Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА
ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

РОЗРОБКА ЧАТ-БОТУ ДЛЯ ВОЛОНТЕРІВ З ВИКОРИСТАННЯМ ТЕХНОЛОГІЙ ПРОГРАМУВАННЯ JAVA

DEVELOPMENT OF A CHATBOT FOR VOLUNTEERS USING JAVA PROGRAMMING TECHNOLOGIES

спеціальність 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

освітня програма «Інженерія програмного забезпечення»
(назва освітньої програми)

Виконав: здобувач вищої освіти
Групи ІПЗ-41
Хвищун Максим Олегович
_____
(підпис)

Керівник:
д.т.н., професор
Гордєєв Олександр Олександрович
_____
(підпис)

Кваліфікаційну роботу
допущено до захисту
«6» 06 2023р.
Гарант освітньої програми:
к.т.н., доцент
Ліщина Наталія Миколаївна
_____
(підпис)

Луцьк – 2023 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра інженерії програмного забезпечення
Ступінь вищої освіти: *бакалавр*
Галузь знань: *12 Інформаційні технології*
Спеціальність: *121 Інженерія програмного забезпечення*
Освітня програма: *«Інженерія програмного забезпечення»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____
«02» 02 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Хвищун Максим Олегович
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи: Розробка чат-боту для волонтерів з використанням технологій програмування Java

Керівник роботи: Гордєєв Олександр Олександрович
затверджені наказом закладу вищої освіти від «*23*» *грудня 2022* р. № *961-01-02*
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи *«8» червня 2023 р.*
3. Вихідні дані до роботи: *програмне забезпечення, вимоги до розробки програмного забезпечення, функціональні та не функціональні вимоги до програмного засобу, технології програмування клієнтської частини – Java, Telegram Boots API, розробка бази даних.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, що потрібно розробити):
Аналіз сучасного стану проблеми, існуючих методів і засобів її розв'язання, аналіз і вибір засобів розробки, опис функціонального наповнення об'єкта проєктування, розробка дизайну, розробка додатку.

5. Перелік графічного матеріалу: 6 рис.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз предметної області	*Гордєєв О.О.*		
Специфікація вимог до розробленої системи	*Гордєєв О.О.*		
Розробка об'єкта проєктування	*Гордєєв О.О.*		
Нормоконтроль	*Гордєєв О.О.*		
Гарант ОП	*Ліщина Н.М.*		
Показник запозичень тексту	17%		
Академічна доброчесність	*Ящук А.А.*		

7. Дата видачі завдання *«2» лютого 2023 р.*

КАЛЕНДАРНИЙ ПЛАН

№ З/п	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Приміт
1	*Огляд літературних джерел по темі кваліфікаційної роботи бакалавра*	*05.02.2023 р.*	Виконано
2	*Аналіз проблеми розробки та впровадження об'єкту проєктування*	*27.02.2023 р.*	Виконано
3	*Обґрунтування вибору шляхів, технологій (алгоритмів) і засобів вирішення поставленого завдання*	*10.03.2023 р.*	Виконано
4	*Розробка функціонально-структурної схеми роботи об'єкта проєктування та проєктування бази даних*	*17.04.2023 р.*	Виконано
5	*Практична реалізація об'єкта проєктування та розробка бази даних*	*18.05.2023 р.*	Виконано
6	*Тестування та налагодження об'єкта проєктування*	*01.06.2023 р.*	Виконано
7	*Здача чистового варіанту кваліфікаційної роботи бакалавра на кафедру*	*08.06.2023 р.*	Виконано

Здобувач вищої освіти _____ Кашуун М.О.
(підпис) (прізвище, ініціали)

Керівник кваліфікаційної роботи _____ Д. Гордєєв
(підпис) (прізвище, ініціали)

Міністерство освіти і науки України
Луцький національний технічний університет

Рецензія
на кваліфікаційну роботу

Здобувач вищої освіти:	Хвищун Максим Олегович
Спеціальність і група:	121 Інженерія програмного забезпечення, ІПЗ-41
Обсяг кваліфікаційної роботи бакалавра:	52 стор., 6 рис., 10 літературних джерел
Кількість сторінок записки:	52 сторінки

Тема: ***Розробка чат-боту для волонтерів з використанням технологій Java***

Коротка характеристика кваліфікаційної роботи та прийнятих рішень:
Кваліфікаційна робота бакалавра складається з трьох розділів, в яких проаналізовані проблематика та поставлені завдання за темою роботи, здійснено теоретичне дослідження та практичну реалізацію чат-боту для волонтерів з використанням технологій Java».

Самостійні розробки і пропозицій автора: *Кваліфікаційна робота бакалавра присвячена розробці програмного продукту, який буде використовуватись для комунікації військових та волонтерів. Користувачу представлений комфортний та функціональний інтерфейс для використання.*

Недоліки: *Істотних недоліків в роботі не виявлено.*

Загальний висновок: *У кваліфікаційній роботі бакалавра було спроектовано та створено чат-бот для волонтерів. Робота є результатом самостійного дослідження. Істотних недоліків не виявлено. Кваліфікаційна робота здобувача освіти Хвищуна Максима рекомендується до захисту екзаменаційній комісії та заслуговує позитивної оцінки.*

Рецензент: Яцишин Микола Миколайович, к.т.н., доцент
(прізвище, ім'я, по-батькові, посада)

Рецензент кваліфікаційної роботи _____
(підпис)

«08» червня 202__ р.

АНОТАЦІЇ

Хвищун М.О. Розробка чат-боту для волонтерів з використанням технологій програмування Java. Рукопис.

Кваліфікаційна робота бакалавра ОП «Інженерія програмного забезпечення» спеціальності Інженерія програмного забезпечення. Луцький національний технічний університет. Луцьк, 2023. 52 с.

Кваліфікаційна робота складається з вступу, трьох розділів, висновків, списку використаних джерел, додатків.

Метою роботи є розробка та реалізація телеграм чат-бота для комунікації волонтерів з військовими.

У першому розділі досліджено актуальний стан сфери волонтерства та комунікації між військовими та волонтерами.

У другому розділі визначено вимоги до розробки телеграм чат-боту. Проведено огляд і аналіз різних засобів, методів та алгоритмів, що можуть бути використані для розробки чат-боту.

У третьому розділі продемонстровано практичну реалізацію об’єкта проєктування, опис розробленого чат-боту та приклад роботи з чат-ботом.

Ключові слова: Java, Telegram, чат-бот, волонтерство.

ABSTRACT

Khvyshchun M. Development of a chatbot for volunteers using Java programming technologies. Manuscript.

Bachelor’s qualifying work of the OP "Software Engineering" specialty Software Engineering. Lutsk National Technical University. Lutsk, 2023. 52 p.

The qualification work consists of an introduction, three sections, conclusions, a list of used sources, and appendices.

The purpose of the work is the development and implementation of a telegram chatbot for communication between volunteers and the military.

The first chapter examines the current state of volunteering and communication between the military and volunteers.

The second section defines the requirements for the development of chatbot telegrams. An overview and analysis of various tools, methods and algorithms that can be used for the development of a chatbot has been carried out.

The third section demonstrates the practical implementation of the design object, a description of the developed chatbot and an example of working with the chatbot.

Keywords: Java, Telegram, chatbot, volunteering.

ЗМІСТ

ВСТУП

Волонтерська діяльність в наш час є особливо актуальною справою. Волонтер – це особа, яка працює на благо суспільства або держави без будь якого прибутку. Оскільки, в нашій країні війна і у волонтерів є багато завдань, то для комунікацій та для планування своєї діяльності існує необхідність в розробці чат-боту для підтримки їхньої роботи. Чат-боти використовують в різноманітних сферах діяльності, оскільки вони надають наступні переваги: по-перше, це доступність, оскільки вони завжди з волонтером в його смартфоні, планшеті або ноутбуці: по-друге, це практичність. Чат-боти існують на будь яку тему і потребність. По-третє, вони є безкоштовними.

Така інтерактивна підтримка допомагає волонтерам економити свій час та зусилля для виконання завдань різноманітної допомоги нашим військовим для майбутньої перемоги.

Метою даної роботи є розробка та реалізація телеграм чат-бота для комунікації волонтерів з військовими. Головна мета полягає у створенні зручного та ефективного інструменту, що дозволить волонтерам та військовим взаємодіяти, обмінюватися інформацією, ділитися ресурсами та координувати свої дії. Бот повинен забезпечувати швидку передачу повідомлень, надійність та конфіденційність даних, а також можливість отримання актуальної інформації щодо поточних потреб та вимог.

Постановка мети полягає у створенні функціональності, яка вирішить проблему неефективності комунікації, забезпечить зручну та швидку взаємодію між волонтерами та військовими, а також покращить організацію спільних дій та обмін необхідною інформацією.

Об’єктом дослідження є розроблений телеграм чат-бот, який дозволяє швидко та зручно здійснювати комунікацію між військовими та волонтерами. Дослідження включає розробку ПЗ, застосування сучасних технологій, мов програмування, API провайдерів для досягнення поставленого завдання.

Предметом дослідження є програмна реалізація телеграм чат-боту для волонтерів з використанням API та таких технологій як: бізнес-процеси. Основними сторонами дослідження є проєктування, розробка та тестування модуля.

Завданнями дослідження є:

– виконати аналіз предметної області використання програмного продукту та виділити основні вимоги до його функціональності;

– на основі майбутньої функціональності додатку здійснити вибір програмних технологій та засобів реалізації;

– дослідити можливості інтеграції «Telegram Bot API» ;

– розробити телеграм чат-бот, керуючись попередньо побудованими структурною та функціональною схемами, який буде забезпечувати комунікацію між військовими та волонтерами;

– забезпечити безпеку та конфіденційність даних;

– протестувати розроблений модуль для виявлення та виправлення можливих помилок та недоліків;

– виконати опис розробленого чат-боту, розробити інструкцію користувача з експлуатації. Підготувати звіт про результати роботи та представити його для захисту кваліфікаційної роботи бакалавра.

РОЗДІЛ 1
АНАЛІЗ АКТУАЛЬНОСТІ ТЕМИ ТА ПОСТАНОВКА ЗАВДАНЬ ПРОЕКТУ

1.1 Аналіз сучасного стану проблеми

Створення чат-боту для комунікації між військовослужбовцями і волонтерами може бути важливим інструментом для покращення комунікації та координації допомоги українським військовим. Однак, на сьогоднішній день, ідеального чат-боту, який підійде для виконання цих потреб ще не існує в Україні.

Проте, українські військові та волонтери використовують інші засоби комунікації, такі як месенджери (наприклад, Viber, Telegram, Facebook Messenger) та соціальні мережі (наприклад, Facebook, Twitter), для координації допомоги. Зокрема, існують різні групи волонтерів у соціальних мережах, де обговорюється поточна ситуація на фронті, потреби військових та можливості допомоги.

Однак, недоліком цих засобів комунікації є те, що вони не забезпечують автоматизованої системи для координації та відстеження потреб та ресурсів, що є важливим для ефективної координації допомоги. Також, вони не забезпечують можливості автоматичної обробки та аналізу інформації для покращення процесів координації та використання ресурсів.

Отже, створення чат-боту для комунікації між військовими та волонтерами може бути корисним та ефективним інструментом для покращення координації допомоги. На даний момент в месенджері Telegram схожих ботів є дуже багато, оскільки в нашій країні війна, але більшість з них створена для допомоги біженцям, які залишили свої домівки через війну. Я хотів би виділити ті, які вважаю одні з найкращих.

Наприклад, SaveUA – даний бот має доволі великий функціонал. Ви зможете знайти допомогу в різних областях України: попутній транспорт, їжу, медикаменти, а також попросити різну допомогу у волонтерів. Також ви зможете запропонувати свою допомогу. Наприклад, прихисток у своєму домі для біженців

або допомогу для волонтерів, фінансову чи фізичну. Чат-бот дозволяє вам обмінятись контактами і зв’язатися в чаті або поговорити по телефоні.

Ви залишаєте заявку на отримання допомоги. Якщо хтось у вашому регіоні зашив заявку на надання допомоги, то ви отримаєте його контактні данні. Для надання допомоги все навпаки. Людина яка потребує допомоги отримує ваші данні.

Переваги:

– простий інтерфейс;

– висока популярність бота в деяких регіонах. Пошук може зайняти небагато часу.

Недолік – бот не має функціоналу для допомоги вирішення проблем, а лише допомагає підібрати співрозмовника (див. рис. 1.1).

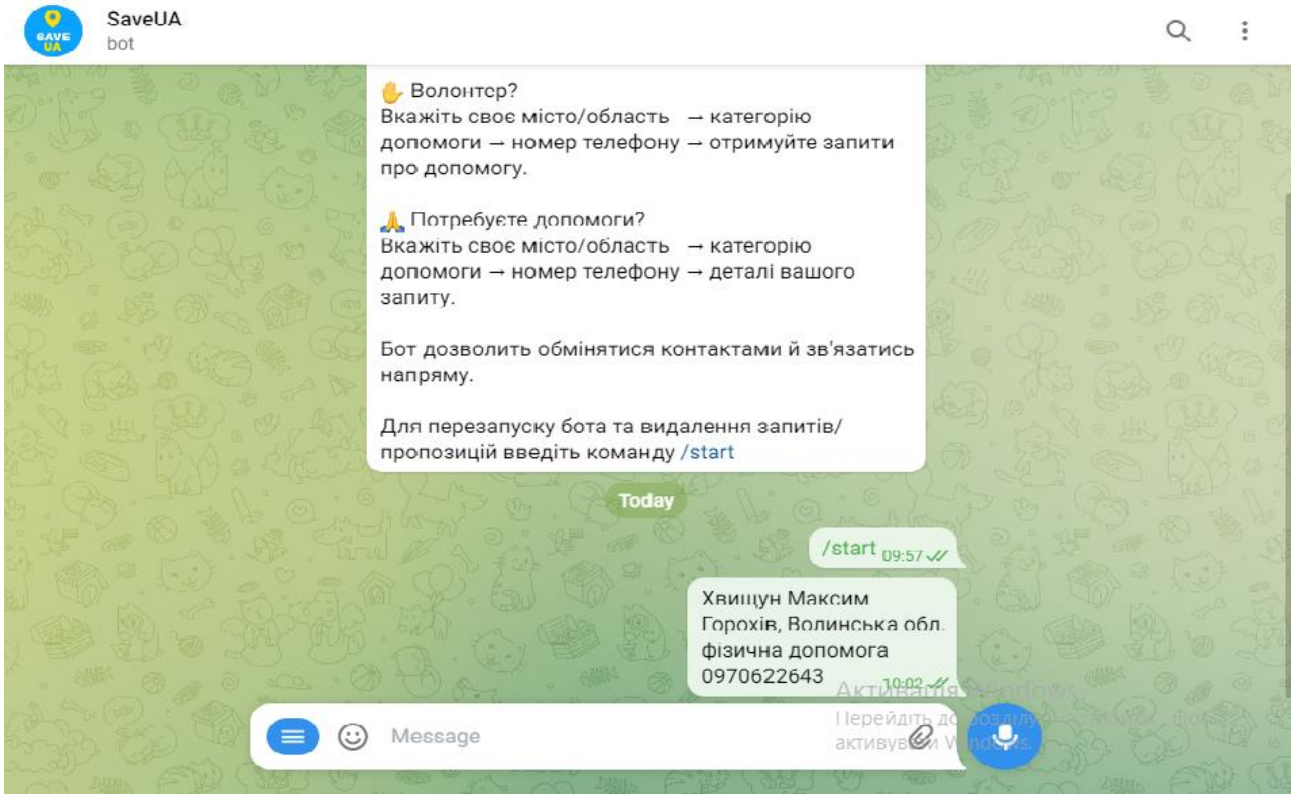

Рисунок 1.1 – Бот SaveUA

Наступний бот-аналог це Help UA | WUBC. В цьому боті ти можеш подати запит на отримання допомоги, запропонувати свою допомогу, а також допомогти фінансово для волонтерів.

Переваги:

– чат-бот має дуже зручний функціонал;

– є контакти самих волонтерів і ти можеш зв’язатись з ними на пряму;

– ця спілка є верифікованим партнером державного проекту СПІВДІЯ, що дає їхнім координаторам доступ до всіх гуманітарних складів України;

– фінансова допомога може проводитись в різних валютах;

– можна перевірити щоденні звіти волонтерів про їхню діяльність та витрачені кошти.

Недолік – всі заявки опрацьовують координатори-волонтери, що може зайняти багато часу.

Ще один схожий чат-бот – Турботник. Цей чат-бот допомагає українським переселенцям отримати тимчасову домівку та необхідні речі у ЦНАПах, що працюють як пункти турботи. Бот працює в багатьох областях України.

Переваги:

– працює в багатьох областях і населених пунктах країни, що дозволяє вибрати найзручніше місцезнаходження;

– простий інтерфейс;

– координатори працюють цілодобово, що дасть можливість в ночі знайти прихисток або потрібні речі в екстрених ситуаціях;

– не багато службових слів, що не засмічує переписку з ботом.

Недоліки – відносно не великий функціонал.

1.2 Постановка завдання на кваліфікаційну роботу бакалавра

На сьогоднішній день існує проблема недостатньої ефективності комунікації між волонтерами та військовими. У зв’язку з цим, необхідно створити зручний та ефективний механізм комунікації, який допоможе волонтерам та військовим взаємодіяти швидко та ефективно.

Чат-бот буде реалізований як бізнес-процес з інтуїтивно-зрозумілим інтерфейсом користувача. Він повинен мати наступні функції:

– реєстрація, збір інформації про волонтера чи військового;
– можливість попросити про допомогу;
– перегляд списку всіх потреб волонтерів та військових
– можливість вирішити будь-яку потребу;
– можливість запропонувати свою допомогу.

Для вирішення поставленого завдання було визначено наступні цілі:

– виконати аналіз предметної області використання програмного продукту та виділити основні вимоги до його функціональності;
– на основі майбутньої функціональності чат-боту здійснити вибір програмних технологій та засобів реалізації;
– дослідити можливості інтеграції «Telegram Bot API» ;
– розробити телеграм чат-бот, керуючись попередньо побудованими структурною та функціональною схемами, який буде забезпечувати комунікацію між військовими та волонтерами;
– забезпечити безпеку та конфіденційність даних;
– протестувати розроблений модуль для виявлення та виправлення можливих помилок та недоліків;
– виконати опис розробленого чат-боту, розробити інструкцію користувача з експлуатації. Підготувати звіт про результати роботи та представити його для захисту кваліфікаційної роботи бакалавра.

Висновок до розділу 1

В розділі «Аналіз сучасного стану проблеми» було проведено дослідження актуального стану сфери волонтерства та комунікації між військовими та волонтерами. В результаті аналізу було виявлено наявні проблеми та недоліки в існуючих засобах комунікації, таких як відсутність зручного та ефективного

інструменту для спілкування, ускладнений доступ до інформації та неефективне керування задачами.

Дане дослідження та розробка телеграм чат-боту для волонтерів є актуальними та значущими, оскільки допомагають поліпшити ефективність волонтерських проектів та полегшують спілкування між волонтерами та військовими. Застосування чат-боту дозволить знизити час та зусилля, необхідні для комунікації та організації волонтерських дій, а також покращить координацію та ефективність роботи волонтерських команд.

Отже, результатом кваліфікаційної роботи буде розроблений та реалізований телеграм чат-бот для волонтерів, який сприятиме покращенню комунікації та організації діяльності волонтерських проектів у сфері допомоги військовим.

РОЗДІЛ 2
АНАЛІЗ, ВИЗНАЧЕННЯ ВИМОГ ДО РОЗРОБЛЮВАНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ. ВИБІР ЗАСОБІВ, МЕТОДІВ І АЛГОРИТМІВ ВИРІШЕННЯ ПОСТАВЛЕНОГО ЗАВДАННЯ

2.1 Аналіз, визначення вимог до розроблюваного програмного забезпечення та проектування програмного забезпечення

Проаналізуємо потенційні вимоги, які можуть висуватися до чат-боту для волонтерів.

Наведу перелік наступних потенційних вимог:

1) забезпечення швидкого та ефективного зв’язку між військовими та волонтерами;

2) забезпечення зручного та доступного інструменту для координації дій між військовими та волонтерами;

3) забезпечення широкого охоплення волонтерського руху та привертання нових волонтерів до участі у роботі з військовими;

4) забезпечення зручного та ефективного інструменту для збору та обробки інформації про волонтерів та їхніх здібностей;

5) забезпечення можливості швидкої та ефективної комунікації між військовими та волонтерами під час надзвичайних ситуацій та кризових ситуацій.

Ці вимоги транспонуються на рівень функціональності чат-боту, та плануються розробка такого функціоналу:

– інтерфейс користувача, який би був зручним та доступним для волонтерів та військових;

– алгоритми та механізми комунікації між волонтерами та військовими;

– функціонал для збору та обробки інформації про волонтерів та їхніх здібностей;

– автоматизація процесу реєстрації та аутентифікації користувачів;

– безпеки даних та захисту від несанкціонованого доступу до системи.

Далі можемо говорити, що множина функцій транспонуться на рівень сценарію для військових та волонтерів.

Сценарій використання для військових:

1) військовий реєструється в системі надаючи свій номер телефону для подальшого зв’язку з волонтерами;

2) військовий може додати в систему потребу (кошти на щось, ліки, спорядження);

3) військовий може передивитись перелік всіх потреб військових з інших підрозділів і перелік допомоги, яку добавили волонтери.

Сценарій використання для волонтерів:

1) волонтер реєструється в системі надаючи свій номер телефону для того, щоб військовий міг зв’язатися з ним;

2) волонтер може переглянути список всіх потреб, які додали військові і якщо має можливість допомогти, та передзвонити до військового за номером телефону, який він залишив;

3) будь-яка людина може зайти і закрити потребність (збір на щось, медицина, спорядження);

4) волонтер може залишати запит на щось.

Далі наведені додаткова необхідна функціональність у вигляді послідовності дій-прецедентів:

1) реєстрація нового користувача:

– користувач заходить в чат і вводить команду /start;

– чат-бот запитує в користувача номер телефону для подальшої реєстрації в системі;

– користувач вводить запитані данні;

– бот перевіряє чи правильно введенні дані і реєструє користувача в системі;

– чат-бот запитує роль користувача(військовий чи волонтер).

2) можливості волонтера:
   - може переглянути список всіх потреб;
   - може переглянути список своїх потреб;
   - може закрити якусь потребу;
   - може додати свою власну потребу.
3) можливості військового:
   - може переглянути список всіх потреб;
   - може переглянути список своїх потреб;
   - може закрити якусь потребу;
   - може додати свою власну потребу.

2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання

2.2.1 Засоби для створення телеграм чат-ботів

На даний момент, для розробки телеграм чат-ботів, є багато інструментів, сервісів та фреймворків, що дозволяють швидко та ефективно створювати ботів.

Розглянемо кілька з них та їхні можливості.

BotFather – це сервіс від Telegram, який дозволяє створити та налаштувати нового бота. Він забезпечує зручний інтерфейс для налаштування основних параметрів бота, таких як ім’я, опис, профільне зображення, а також для налаштування команд, які відповідає бот.

Переваги BotFather:

- безкоштовний інструмент від Telegram;
- простий та зрозумілий інтерфейс;
- швидко створює нового бота.

Недолік BotFather – обмежений функціонал. Не дозволяє створювати складні функції бота.

Dialogflow – це інструмент від Google, який дозволяє створювати природні мовні інтерфейси для різних платформ, включаючи Telegram. Він забезпечує

можливість створювати складні логічні взаємодії з користувачами та навчати бота розуміти природну мову.

Переваги Dialogflow:

– можливість створювати складні логічні взаємодії з користувачами;

– навчання бота розуміти природну мову.

Недолік Dialogflow – вартість використання.

Botpress – це фреймворк для розробки телеграм чат-ботів, що базується на Node.js та дозволяє швидко створювати складні боти з можливістю розширення та налаштування. Він забезпечує можливість створювати складні логічні взаємодії з користувачами та інтеграцію з іншими сервісами.

Переваги Botpress:

– легкість використання та широкий функціонал, що дозволяє створювати складні чат-боти зі складними логіками;

– можливість інтеграції з багатьма іншими платформами, які не обмежуються Telegram;

– наявність багатьох модулів та плагінів, які дозволяють додатково розширювати функціональність.

Недолік Botpress – не дуже добра документація для початківців, що може затримати процес розробки.

Для розробки даного чат-боту я обрав бібліотеку Telegram Bot API.

Telegram Bot API – це набір методів і функцій, які надаються Telegram для взаємодії з чат-ботами. Це програмний інтерфейс, який дозволяє розробникам створювати, налаштовувати та керувати функціями своїх чат-ботів.

Telegram Bot API надає різноманітні можливості для взаємодії з Telegram-платформою. Ці можливості включають:

– отримання та відправка повідомлень. Розробники можуть використовувати API для отримання вхідних повідомлень від користувачів та відправки відповідних відповідей;

– керування чатами. За допомогою API можна отримувати доступ до інформації про чати, створювати нові чати та керувати налаштуваннями чат-груп;

– робота з мультимедіа. API дозволяє отримувати та відправляти фотографії, відео, аудіо та інші мультимедійні файли;

– керування клавіатурою. Розробники можуть створювати спеціальні клавіатури для взаємодії з користувачами, використовуючи API Telegram Bot;

– отримання інформації про користувачів. API надає доступ до даних про користувачів, таких як ім’я, ідентифікатор, мова тощо.

API Telegram Bot можна використовувати для розробки різних функцій чат-бота, включаючи автоматичні відповіді, взаємодію зі сторонніми сервісами, розсилку повідомлень та інше. Для використання API необхідно отримати токен бота від Telegram і налаштувати з’єднання з платформою за допомогою мови програмування та бібліотеки, таких як Java і Telegram Bot API для Java.

2.2.2 Телеграм чат-бо з використанням мови програмування Java

Створення Telegram чат-боту з використанням мови програмування Java може бути виконано за допомогою використання бібліотеки Telegram Bots API, яка надає програмістам необхідний інтерфейс для звернення до Telegram API.

Спочатку потрібно створити бота на Telegram, отримати API-ключ і додати бота до потрібного чату або створити новий чат. Далі можна розпочати програмування бота на мові Java з використанням Telegram Bots API.

Основні переваги використання Java для створення Telegram чат-боту включають:

– одна з найпопулярніших мов програмування в світі, що дозволяє програмістам мати багато документації та джерел, які допоможуть у вирішенні будь-яких проблем, які виникають під час розробки бота;

– має велику кількість бібліотек, що дозволяє програмістам розширювати можливості свого бота та зменшувати час розробки;

– дозволяє програмістам розробляти програмне забезпечення для різних платформ (Windows, Mac, Linux, Android та ін.), що дозволяє розгорнути бота на різних платформах з легкістю.

До недоліків можна віднести:

– потреба в віртуальній машині. Для виконання програм на мові Java потрібна відповідна віртуальна машина, що може знизити швидкість виконання програм;

– споживання пам’яті. Програми на мові Java можуть споживати більше пам’яті порівняно з іншими мовами програмування.

Додаткові причини (переваги) розробки чат-боту з використанням мови програмування Java для створення Telegram чат-боту:

1) платформонезалежність. Це означає, що чат-бот написаний на мові програмування Java можна запустити на будь-якій операційній системі (Windows, Linux, MacOS);

2) вбудовані механізми оптимізації коду, які допомагають зменшити витрати пам’яті та збільшити швидкість виконання програм. Одним з таких механізмів є Just-In-Time (JIT) компілятор. Коли Java-програма запускається, вона спочатку компілюється в байт-код, який потім виконується в Java Virtual Machine (JVM). JIT компілятор працює в фоновому режимі та аналізує виконання байт-коду, щоб ідентифікувати часті шляхи виконання і перетворити їх на нативний машинний код. Це зменшує час виконання програми, оскільки нативний машинний код виконується швидше, ніж байт-код. Іншим механізмом є Garbage Collection (GC), який допомагає автоматично визначати та звільняти пам’ять, що не використовується програмою. GC допомагає зменшити витрати пам’яті та збільшити продуктивність програми, оскільки забезпечує ефективніше використання доступної пам’яті. Крім того, Java також має оптимізацію використання многопоточності, таку як синхронізацію потоків та пул потоків, що допомагає підвищити ефективність програми в умовах багатопоточності;

3) має власну добре розроблену інтегровану середу розробки (IDE). Дана середа розробки має велику кількість різноманітних інструментів, які допоможуть розробнику швидко і якісно створити код;

4) налічує величезну кількість стандартних бібліотек і фреймворків, які допоможуть розробникам створювати складні та багатофункціональні програми.

Для створення Telegram чат-боту на мові програмування Java будемо використовувати бібліотеку Telegram Bots API, яка дозволяє взаємодіяти з Telegram API та розробляти чат-ботів з різноманітним функціоналом.

Деякі з можливих алгоритмів та математичних моделей, які можна застосувати для розробки Telegram чат-боту на Java, включають:

1) алгоритми обробки природньої мови (Natural Language Processing, NLP), що дозволяють розпізнавати та розуміти текстові повідомлення, що надходять до бота. Застосування NLP може допомогти розробити бота з покращеним інтерфейсом та здатністю до автоматичної обробки запитів користувачів;

2) моделі машинного навчання (Machine Learning, ML), які можуть допомогти покращити функціонал бота та забезпечити кращу точність у відповіді на запити користувачів. Моделі машинного навчання можуть використовуватися для розпізнавання та класифікації текстових повідомлень, що дозволяє боту автоматично відповідати на запити користувачів;

3) алгоритми аналізу даних (Data Analysis), що дозволяють аналізувати та обробляти великі обсяги даних, які збираються від користувачів бота. Застосування алгоритмів аналізу даних може допомогти зрозуміти поведінку користувачів та покращити функціонал бота;

4) алгоритми оптимізації та пошуку (Optimization and Search algorithms), що можуть використовуватися для оптимізації деяких функцій бота та пошуку оптимального рішення для певних задач.

2.2.3 Основні компоненти системи

Будуть створені такі основні компоненти системи:

– користувацький інтерфейс. Цей компонент забезпечує взаємодію користувачів з системою. Він може бути реалізований у вигляді телеграм бота, який дозволяє вводити команди, відправляти повідомлення, отримувати інформацію тощо;

– реєстрація. Цей компонент відповідає за реєстрацію користувачів. Він забезпечує збір інформації про військового чи волонтера;

– обробка повідомлень. Цей компонент отримує вхідні повідомлення від користувачів і обробляє їх. Він може виконувати різноманітні дії, такі як збереження повідомлень у базі даних, розподіл їх до відповідних волонтерів або військових, генерація відповідей тощо;

– управління задачами. Цей компонент відповідає за управління поточними задачами або потребами, які виникають у волонтерів та військових. Він може допомагати у створенні нових задач, відстежуванні стану виконання задач тощо;

– звітність та аналітика. Цей компонент забезпечує збір та аналіз даних про взаємодію волонтерів та військових. Він може надавати звіти про кількість задач та їх завершеність;

– база даних. Цей компонент забезпечує збереження та управління даними, які використовуються в системі. Він може включати таблиці для зберігання інформації про користувачів, повідомлення, задачі, журнали тощо.

Кожен з цих компонентів має свою функціональність та взаємодіє з іншими компонентами для забезпечення повного функціонування системи телеграм чат-бота для комунікації волонтерів з військовими.

Висновок до розділу 2

В другому розділі було проведено аналіз потреб і вимог до розробки телеграм чат-боту для волонтерів. В результаті аналізу було визначено основні

функціональні та нефункціональні вимоги, такі як можливість обміну повідомленнями, доступ до інформації, керування задачами та інші.

На основі визначених вимог було проведено проектування програмного забезпечення. Вибір архітектурного рішення та компонентів системи був здійснений з урахуванням потреб користувачів та забезпечення оптимальної продуктивності та швидкості роботи. В результаті проектування була розроблена база даних та інтерфейс користувача.

Також було проведено огляд і аналіз різних засобів, методів та алгоритмів, що можуть бути використані для реалізації телеграм чат-боту. Було обрано мову програмування Java та Telegram Bots API для реалізації боту. Також було обрано базу даних Redis для зберігання і управління даними бота.

В результаті проведеного аналізу та проектування були визначені оптимальні засоби та методи для реалізації телеграм чат-боту для волонтерів, що відповідають поставленим вимогам та забезпечують ефективну та зручну роботу з програмним забезпеченням.

РОЗДІЛ 3
ПРАКТИЧНА РЕАЛІЗАЦІЯ ОБ'ЄКТА ПРОЕКТУВАННЯ. РОЗРОБКА БАЗИ ДАННИХ

3.1 Практична реалізація об'єкта проектування

Структура проекту телеграм чат-бота для комунікації волонтерів з військовими має наступну організацію:

– пакет «config». В цьому пакеті зберігаються конфігураційні класи, які встановлюють параметри та налаштування системи, наприклад, підключення до бази даних або зовнішніх сервісів;

– пакет «conversation» в структурі проекту телеграм чат-бота для комунікації волонтерів з військовими може містити класи та компоненти, що відповідають за керування бесідами з користувачами;

– пакет «model». В цьому пакеті знаходяться класи моделей, які відображають сутності системи, такі як користувачі, завдання, повідомлення тощо. Вони використовуються для збереження та передачі даних;

– пакет «repository». В цьому пакеті розташовуються класи репозиторіїв, які забезпечують взаємодію з базою даних. Вони виконують операції збереження, отримання, оновлення та видалення даних з бази даних;

– пакет «service». В цьому пакеті містяться класи сервісів, які реалізовують бізнес-логіку додатку. Вони виконують обробку вхідних повідомлень, взаємодію з базою даних, відправку повідомлень та інші операції.

У даному коді (див. лістинг 3.1) показана конфігурація додатку за допомогою Spring Framework у мові програмування Java.

Лістинг 3.1 – Демонстрація класу ApplicationConfig

```
@Configuration
@PropertySource("classpath:application.properties")
public class ApplicationConfig {
    @Bean
    public static PropertySourcesPlaceholderConfigurer
```

продовження лістингу 3.1

```
propertySourcesPlaceholderConfigurer() {
        return new PropertySourcesPlaceholderConfigurer();
    }
    @Bean
    TelegramBotsApi telegramBotsApi() throws TelegramApiException {
        return new TelegramBotsApi(DefaultBotSession.class);
    }
}
```

Кінець лістингу 3.1

Анотація @Configuration позначає клас як конфігураційний клас, що містить налаштування бінів.

Анотація @PropertySource(«classpath:application.properties») вказує на файл application.properties, який містить конфігураційні параметри додатку, що будуть завантажені.

Метод propertySourcesPlaceholderConfigurer() повертає PropertySourcesPlaceholderConfigurer, який дозволяє читати значення властивостей з файлу application.properties і використовувати їх у конфігурації.

Метод telegramBotsApi() повертає об’єкт TelegramBotsApi, який використовується для створення з’єднання з Telegram Bot API. Цей об’єкт ініціалізується з використанням DefaultBotSession.class, що означає, що використовується тип за замовчуванням для бот-сесії.

Клас ApplicationConfig може містити інші методи @Bean, які конфігурують інші біни або сервіси, що використовуються в додатку.

Цей код (див. лістинг 3.2) демонструє налаштування для додатку, який використовує Telegram Bot API. Він завантажує значення властивостей з файлу application.properties та створює об’єкт TelegramBotsApi, який може бути використаний для роботи з ботами у Telegram.

Лістинг 3.2 – Демонстрація файлу application.properties

```
logging.level.root=warn
logging.level.com.fastforward=info
telegram.bot.name=BotForVolunteers
```

продовження лістингу 3.2

```
spring.redis.host=localhost
spring.redis.port=6379
```

Кінець лістингу 3.2

У наведених значеннях властивостей з файлу application.properties містяться налаштування для логування, Telegram бота та Redis бази даних. Давайте розглянемо кожен з них:

– logging.level.root=warn встановлює рівень логування для кореневого логгера. У цьому випадку, будуть логуватися тільки повідомлення з рівнем WARN і вище;

– logging.level.com.fastforward=info встановлює рівень логування для логгера з пакетом com.fastforward. У цьому випадку, будуть логуватися повідомлення з рівнем INFO і вище для цього пакета;

– telegram.bot.name=BotForVolunteers вказує ім’я вашого Telegram бота;

– telegram.bot.token=5831737344:AAFQOJq0l1Mh_5S5gUl9vnRNRhrM2ppIbXY вказує токен вашого Telegram бота;

– spring.redis.host=localhost вказує хост Redis сервера, до якого буде встановлене з’єднання. У цьому випадку, Redis сервер розташований на локальній машині;

– spring.redis.port=6379 - це конфігураційна властивість, яка встановлює порт, на якому працює Redis сервер. В даному випадку, значення 6379 вказує на те, що Redis сервер використовує порт 6379 для з’єднання і обміну даними.

У наведеному фрагменті коду (див. лістинг 3.3) метод flows() повертає список ConversationStep.

Лістинг 3.3 – Демонстрація методу flows() в класі AddTaskConversation

```
public List<ConversationStep> flows() {
    return Lists.newArrayList(
        (context, input) -> new SuccessfulConversationResponse(
            context.getId(),
            Lists.newArrayList(new Reply(
```

```
                "Оберіть категорію",
```

продовження лістингу 3.3

```
                Arrays.stream(TaskCategory.values())
                        .map(TaskCategory::getDisplayName)
                        .collect(Collectors.toList()))
            )
        )
```

Кінець лістингу 3.3

Lists.newArrayList(...) створює новий список з переданими елементами. У цьому випадку, створюється список з одним елементом - лямбда-виразом.

Лямбда-вираз (context, input) -> new SuccessfulConversationResponse(...) виконує обробку контексту розмови та введеного користувачем повідомлення і створює відповідь на розмову.

new SuccessfulConversationResponse(...) створює об’єкт SuccessfulConversationResponse, який представляє успішну відповідь на розмову. Він містить ідентифікатор контексту розмови та список варіантів відповіді.

Lists.newArrayList(new Reply(...)) створює список з одним елементом – об’єктом Reply. Об’єкт Reply містить текст повідомлення, що буде відправлено користувачу, та список опцій відповіді.

«Оберіть категорію» − це текст повідомлення, яке буде відправлене користувачу.

Arrays.stream(TaskCategory.values()) створює потік (stream) з елементами перерахування TaskCategory.values(). TaskCategory є перерахуванням, яке містить категорії задач.

.map(TaskCategory::getDisplayName) виконує перетворення кожного елементу потоку на його відображення (display name) за допомогою методу getDisplayName() класу TaskCategory.

.map(r -> new Option(r, r)) створює новий об’єкт Option для кожного елементу потоку. Об’єкт Option містить значення та текст, які будуть відображені в якості варіантів відповіді для користувача.

.collect(Collectors.toList()) збирає всі об’єкти Option з потоку в список.

Таким чином, код створює відповідь на повідомлення, що містить текст «Оберіть категорію» та список варіантів відповіді, які виводяться на основі перерахування TaskCategory.

У наведеному фрагменті коду (див. лістинг 3.4) представлений лямбда-вираз, який виконує обробку контексту розмови та введеного користувачем повідомлення.

Лістинг 3.4 – Демонстрація методу flows() в класі AddTaskConversation

```
(context, input) -> {
    if(TaskCategory.get(input.trim()) == null) {
        return new RetryConversationResponse(context.getId(),
Lists.newArrayList(new Reply(
            "Ви вказали неіснуючу категорію")
        ));
    }
    context.getContext().put("category", input);
    return new SuccessfulConversationResponse(
        context.getId(),
        Lists.newArrayList(new Reply(
            "Введіть опис")
        )
    );
}
```

Кінець лістингу 3.4

Давайте розберемо його по кроках:

– (context, input) -> { ... } – це лямбда-вираз, який приймає два параметри: context (контекст розмови) і input (введене користувачем повідомлення). Ви можете використовувати ці параметри для доступу до даних розмови та обробки введеного повідомлення;

– TaskCategory.get(input.trim()) – це виклик статичного методу get(...) класу TaskCategory, який приймає введене повідомлення і повертає відповідну категорію задачі з перерахування TaskCategory. Метод trim() викликається для видалення зайвих пробілів з початку і кінця рядка;

– if (TaskCategory.get(input.trim()) == null) { ... } – це умовний оператор if, який перевіряє, чи повертає метод TaskCategory.get(...) значення null. Якщо так, це означає, що введена користувачем категорія не існує;

– return new RetryConversationResponse(context.getId(), Lists.newArrayList(new Reply(«Ви вказали неіснуючу категорію»))); – це повернення об’єкту RetryConversationResponse, який представляє відповідь на розмову з повідомленням про помилку введеної категорії. У цьому випадку, повідомлення «Ви вказали неіснуючу категорію» передається в об’єкт Reply, який потім додається до списку варіантів відповіді;

– context.getContext().put(«category», input); – це додавання значення input в контекст розмови під ключем «category». Ви можете використовувати контекст розмови для зберігання додаткових даних та передачі їх між кроками розмови;

– return new SuccessfulConversationResponse(...); – це повернення об’єкту SuccessfulConversationResponse, який представляє успішну відповідь на розмову. У цьому випадку, створюється об’єкт Reply з повідомленням «Введіть опис» і додається до списку варіантів відповіді.

Загальною метою цього фрагменту коду є обробка введеного користувачем повідомлення, перевірка валідності категорії задачі та встановлення значення категорії в контексті розмови. Потім повертається відповідь з запитом на введення опису задачі.

У наведеному фрагменті коду (див. лістинг 3.5) представлений лямбда-вираз, який виконує обробку контексту розмови та введеного користувачем повідомлення.

Лістинг 3.5 – Демонстрація методу flows() в класі AddTaskConversation

```
(context, input) -> {
    context.getContext().put("description", input);
    String category = context.getContext().get("category").trim();
    String description = context.getContext().get("description");
    taskRepository.save(
        new Task(
```

продовження лістингу 3.3

```
                context.getId(),
                TaskCategory.get(category),
                description
            )
        );
        return new SuccessfulConversationResponse(
            context.getId(),
            Lists.newArrayList(new Reply(
                "Ви успішно додали потребу")
            ),
            true
        );
}
```

Кінець лістингу 3.5

Давайте розберемо його по кроках:

– (context, input) -> { ... } – це лямбда-вираз, який приймає два параметри: context (контекст розмови) і input (введене користувачем повідомлення). Ви можете використовувати ці параметри для доступу до даних розмови та обробки введеного повідомлення;

– context.getContext().put(«description», input); – це додавання значення input в контекст розмови під ключем «description». Ви використовуєте контекст розмови для зберігання даних, отриманих від користувача;

– String category = context.getContext().get(«category»).trim(); – це отримання значення категорії з контексту розмови під ключем "category" та виклик методу trim(), щоб видалити зайві пробіли з початку і кінця рядка. Значення категорії зберігається в змінній category;

– String description = context.getContext().get(«description»); – це отримання значення опису з контексту розмови під ключем "description". Значення опису зберігається в змінній description;

– taskRepository.save(...); – це збереження нового об’єкту Task в репозиторії (базі даних). Створюється новий об’єкт Task з використанням значень context.getId() (ідентифікатор розмови), TaskCategory.get(category) (категорія задачі) і description (опис задачі);

– return new SuccessfulConversationResponse(...); – це повернення об’єкту SuccessfulConversationResponse, який представляє успішну відповідь на розмову. У цьому випадку, повідомлення "Ви успішно додали потребу" передається в об’єкт Reply, який потім додається до списку варіантів відповіді. Параметр true вказує, що розмова завершена.

Цей фрагмент коду (див. лістинг 3.6) зберігає нову задачу в базі даних за допомогою репозиторію taskRepository і повертає успішну відповідь користувачу про додавання потреби.

Лістинг 3.6 – Демонстрація методу flows() в класі CloseTaskConversation

```
public List<ConversationStep> flows() {
    return Lists.newArrayList(
        (context, input) -> {
            Long taskId =
Long.parseLong(context.getContext().get("taskId"));
            Optional<Task> optTask =
taskRepository.findById(taskId);
            if (!optTask.isPresent()) {
                return new SuccessfulConversationResponse(
                    context.getId(),
                    Lists.newArrayList(new Reply(
                        "Неіснуюча потреба")
                    ),
                    true
                );
            }
```

Кінець лістингу 3.6

У наведеному фрагменті коду представлений лямбда-вираз, який виконує обробку контексту розмови та введеного користувачем повідомлення. Давайте розберемо його по кроках:

– (context, input) -> { ... } – це лямбда-вираз, який приймає два параметри: context (контекст розмови) і input (введене користувачем повідомлення). Ви можете використовувати ці параметри для доступу до даних розмови та обробки введеного повідомлення;

– Long taskId = Long.parseLong(context.getContext().get(«taskId»)); – це отримання значення ідентифікатора задачі з контексту розмови під ключем

«taskId» та перетворення його в тип Long. Значення ідентифікатора задачі зберігається в змінній taskId;

– Optional<Task> optTask = taskRepository.findById(taskId); − це отримання об’єкту Task з репозиторію (бази даних) за його ідентифікатором. Використовується метод findById(...) репозиторію taskRepository, який повертає обгортку Optional, що містить об’єкт Task або null, якщо задача не знайдена;

– if (!optTask.isPresent()) { ... } − це умовний оператор if, який перевіряє, чи присутній об’єкт Task в Optional. Якщо об’єкт не присутній (тобто задача не знайдена), виконується наступний блок коду;

– return new SuccessfulConversationResponse(...); − це повернення об’єкту SuccessfulConversationResponse, який представляє успішну відповідь на розмову. У цьому випадку, повідомлення "Неіснуюча потреба" передається в об’єкт Reply, який потім додається до списку варіантів відповіді. Параметр true вказує, що розмова завершена.

Цей фрагмент коду (див. лістинг 3.7) виконує перевірку наявності задачі за її ідентифікатором і повертає відповідь про неіснуючу потребу, якщо задача не знайдена.

Лістинг 3.7 – Демонстрація методу flows() в класі CloseTaskConversation

```
else {
    Task task = optTask.get();
    taskRepository.deleteById(taskId);
    return new SuccessfulConversationResponse(
        context.getId(),
        Lists.newArrayList(new Reply(
            String.format("Ви успішно закрили потребу: %s",
task.getDescription()))
        ),
        true
    );
}
```

Кінець лістингу 3.7

У наведеному фрагменті коду представлений лямбда-вираз, який виконує обробку контексту розмови та введеного користувачем повідомлення. Давайте розберемо його по кроках:

– (context, input) -> { ... } − це лямбда-вираз, який приймає два параметри: context (контекст розмови) і input (введене користувачем повідомлення). Ви можете використовувати ці параметри для доступу до даних розмови та обробки введеного повідомлення;

– Task task = optTask.get(); − це отримання об’єкту Task з обгортки Optional<Task>. Ви використовуєте метод get() обгортки Optional, щоб отримати об’єкт Task, якщо він присутній;

– taskRepository.deleteById(taskId); − це видалення задачі з бази даних за її ідентифікатором. Використовується метод deleteById(...) репозиторію taskRepository, щоб видалити задачу за заданим ідентифікатором;

– return new SuccessfulConversationResponse(...); − це повернення об’єкту SuccessfulConversationResponse, який представляє успішну відповідь на розмову. У цьому випадку, повідомлення про успішне закриття потреби формується за допомогою методу String.format(...), де %s підставляється опис задачі. Повідомлення додається до об’єкту Reply, який потім додається до списку варіантів відповіді. Параметр true вказує, що розмова завершена.

Цей фрагмент коду виконує перевірку наявності задачі за її ідентифікатором і повертає успішну відповідь про закриття потреби, якщо задача знайдена.

3.2 Тестування та налагодження інформаційно-комп’ютерної системи

Методи тестування і налагодження програми є важливою частиною процесу розробки телеграм-чат-боту. Ось деякі з методів, які можна використовувати:

1) юніт-тести. Використання цих тестів допомагає перевірити правильність роботи окремих функцій та модулів програми. Це можна зробити за допомогою фреймворків для тестування, таких як JUnit для Java.

JUnit є одним з найпопулярніших фреймворків для тестування в програмній мові Java. Він надає набір інструментів і прийомів для написання і виконання автоматизованих юніт-тестів.

Основні риси та особливості JUnit:

– анотації. JUnit використовує анотації для позначення методів, які слід виконати як тестові. Наприклад, анотація @Test вказує, що метод є тестовим;

– перевірка очікуваних результатів. JUnit надає набір методів-перевірок, таких як assertEquals, assertTrue, assertFalse і т. д., що допомагають перевіряти, чи отримані результати співпадають з очікуваними;

– властивості запуску. JUnit дозволяє налаштовувати спосіб запуску тестів, наприклад, виконувати їх послідовно або паралельно;

– спеціальні методи. JUnit має спеціальні методи, які можна використовувати для налаштування перед тестуванням (@Before) та після тестування (@After), що полегшує налаштування тестового середовища;

– контекст виконання. JUnit надає можливість створювати контекст виконання для тестів за допомогою аннотацій @BeforeClass та @AfterClass, які виконуються один раз перед виконанням всіх тестів у класі;

– параметризовані тести. JUnit дозволяє створювати параметризовані тести, що дозволяють виконати одну тестову логіку з різними наборами вхідних даних.

JUnit є потужним інструментом для написання автоматизованих юніт-тестів, який допомагає забезпечити якість коду, знижує кількість помилок і полегшує розробку програм в мові Java.

2) інтеграційні тести перевіряють взаємодію між різними компонентами програми. Вони допомагають виявити можливі проблеми, які можуть виникнути при інтеграції окремих модулів.

Основні особливості та підходи до інтеграційного тестування:

– тестування взаємодії. Інтеграційні тести перевіряють, як різні компоненти системи взаємодіють один з одним. Це може включати передачу даних, виклики API, обмін повідомленнями тощо;

– розширення обсягу тестування. Інтеграційні тести доповнюють юніт-тести, які перевіряють окремі компоненти незалежно від інших частин системи. Інтеграційні тести дозволяють виявити помилки, які можуть виникнути лише при взаємодії між компонентами;

– мокування та стабилизація. Інтеграційні тести можуть використовувати моки або стаби, які замінюють реальні залежності системи, щоб зосередитися на тестуванні самої інтеграції. Це дозволяє контролювати залежності та забезпечує стабільність тестів;

– тестові середовища. Для проведення інтеграційних тестів можуть використовуватись окремі тестові середовища, які наближають реальне виробниче середовище. Це дозволяє перевірити взаємодію компонентів в аналогічних умовах, які вони будуть мати у реальному середовищі;

3) функціональні тести є одним з основних типів тестування програмного забезпечення, які перевіряють, чи виконує програма свої функції відповідно до вимог і очікувань користувачів. Основна мета функціональних тестів – переконатися, що програма працює правильно і надає очікувану функціональність.

Особливості функціональних тестів:

– сценарії використання. Функціональні тести базуються на сценаріях використання програми, що описують послідовність дій користувача. Кожен тест складається з набору вхідних даних та очікуваного результату для конкретного сценарію;

– тестування різних функцій. Функціональні тести перевіряють окремі функції програми, включаючи основні операції, обробку даних, взаємодію з користувачем та інші функціональні можливості. Кожен тест фокусується на конкретній функції або функціональності;

– валідація вхідних даних. Функціональні тести перевіряють, як програма обробляє різні типи вхідних даних, включаючи правильність обробки коректних даних та обробку помилкових ситуацій;

– перевірка результатів. Функціональні тести порівнюють фактичні результати виконання програми з очікуваними результатами. Це може включати порівняння вихідних значень, повідомлень про помилки, станів системи та інших вимірів функціональності;

– автоматизація тестів. Функціональні тести можуть бути автоматизовані за допомогою спеціальних фреймворків, таких як Selenium, Cucumber, Robot Framework.

4) тестування реального використання (End-to-End тести). Цей тип тестування включає симуляцію реального використання програми з точки зору кінцевого користувача. Він перевіряє, чи відповідає програма всім очікуванням та вимогам користувачів.

Особливості тестування реального використання:

– виконання реальних сценаріїв. У цьому типі тестування використовуються реальні сценарії використання програмного забезпечення або системи, які базуються на реальних вимогах та очікуваннях користувачів. Це включає реалістичні дані введення, реальні дії користувача та реальні умови використання;

– реплікація реальних умов. У тестуванні реального використання стараються реплікувати реальні умови, з якими зіштовхнеться програмне забезпечення або система в реальному світі. Це може включати тестування в реальних середовищах, з реальним обсягом даних, з врахуванням реальних обмежень, швидкості мережі тощо;

– тестування продуктивності та навантаження. Під час тестування реального використання здійснюються перевірки продуктивності та витривалості системи за реальними умовами навантаження. Це дозволяє оцінити ефективність, швидкість та стійкість системи під час реального використання;

– збір фідбеку користувачів. Під час тестування реального використання активно залучаються реальні користувачі програмного забезпечення або системи. Вони надають фідбек, спостереження та рекомендації, що допомагають виявити проблеми.

5) ручне тестування. Крім автоматизованих тестів, варто також проводити ручне тестування, де розробники або тестувальники перевіряють різні аспекти програми вручну. Це може включати перевірку коректності відображення, взаємодії з користувачем та інші аспекти, які важко автоматизувати.

Особливості ручного тестування:

– тест-дизайн. Ручне тестування вимагає планування та створення тест-кейсів та тест-сценаріїв. Тестувальник розробляє тести, виходячи з вимог, специфікацій та знань про доменну область, щоб виявити помилки або неполадки в програмному забезпеченні;

– виконання тестів. Тестувальник вручну виконує тест-кейси та тест-сценарії, дотримуючись певних критеріїв тестування. Він вводить вхідні дані, виконує дії та перевіряє, чи відповідають отримані результати очікуванням;

– виявлення та документування помилок. Ручне тестування дозволяє тестувальникам виявляти помилки, неполадки та недоліки в програмному забезпеченні. Вони фіксують ці помилки в спеціальних звітах або дефектних тикетах, де вказують деталі про проблему, кроки для відтворення та інші важливі відомості;

– експлораторське тестування. Ручне тестування також може включати експлораторський підхід, коли тестувальник вільно досліджує програмне забезпечення, виконуючи різні дії, намагаючись виявити помилки та непередбачені сценарії;

– валідація користувацького досвіду. Ручне тестування дозволяє перевірити користувацький досвід.

3.3 Розробка бази даних

Redis (Remote Dictionary Server) – це відкрите програмне забезпечення, яке забезпечує швидку та ефективну зберігання та обробку даних. Redis використовує модель ключ-значення, де кожен ключ має відповідне значення.

Основні характеристики Redis:

– швидкодія. Redis зберігає дані у пам’яті, що дозволяє досягати високої продуктивності. Він використовує оптимізовані структури даних та алгоритми для швидкого доступу до даних;

– типи даних. Redis підтримує різні типи даних, включаючи рядки, хеші, списки, набори та сортовані набори. Це надає гнучкість при зберіганні та обробці різних видів даних;

– кешування. Redis часто використовується як кеш-сховище для покращення продуктивності додатків. Він може зберігати часто використовувані дані у пам’яті, що дозволяє швидко отримувати доступ до них;

– підтримка операцій. Redis надає різноманітні операції для роботи з даними, такі як збереження, отримання, оновлення та видалення значень за ключем. Він також підтримує операції над множинами, сортуванням даних, публікацію та підписку на повідомлення;

– висока доступність. Redis може бути налаштований для забезпечення високої доступності та надійності. За допомогою механізмів реплікації та кластеризації, ви можете створити резервні копії даних та забезпечити безперервну роботу системи;

– розширення та інтеграція. Redis має багато функціональних можливостей та підтримує різні бібліотеки та мови програмування. Ви можете легко інтегрувати Redis у свої додатки та використовувати його API для взаємодії з базою даних.

Метод redisTemplate визначає RedisTemplate, який використовується для взаємодії з Redis. Він отримує JedisConnectionFactory (див. лістинг 3.8) як параметр

і налаштовує його як з'єднання для RedisTemplate. RedisTemplate параметризований типом ключа та значення. В наведеному прикладі типи ключа та значення встановлені як byte[].

Лістинг 3.8 – Демонстрація методу redisConnectionFactory в Java

```
@Bean
public JedisConnectionFactory
redisConnectionFactory(@Value("${spring.redis.host}") String
redisHost,

@Value("${spring.redis.port}") int redisPort) {
    return new JedisConnectionFactory(new
RedisStandaloneConfiguration(redisHost, redisPort));
}
@Bean
public RedisTemplate<?, ?> redisTemplate(JedisConnectionFactory
connectionFactory) {
    RedisTemplate<byte[], byte[]> template = new RedisTemplate<>();
    template.setConnectionFactory(connectionFactory);
    return template;
}
```

Кінець лістингу 3.8

За допомогою цього RedisTemplate ви можете виконувати операції з Redis, такі як збереження, отримання, оновлення або видалення даних.

Загальна ідея цього класу полягає в тому, що ви можете використовувати його у своєму Spring-проекті для налаштування з'єднання з Redis і отримання RedisTemplate, який дозволить вам взаємодіяти з Redis базою даних.

Загалом, Redis є потужним інструментом для швидкої та ефективної зберігання та обробки даних. Його простота використання та багатофункціональність роблять його популярним вибором для багатьох розробників та проектів.

Для реалізації бази даних за допомогою Redis нам потрібно виконати наступні дії.

Метод redisConnectionFactory визначає JedisConnectionFactory, який використовується для створення з'єднання з Redis сервером. Він приймає значення

хоста та порту Redis сервера, які можна налаштувати за допомогою анотації @Value.

Базу даних Redis ми будемо запускати через Docker. Docker – це відкрите програмне забезпечення, яке надає контейнеризацію програм та сервісів. Контейнеризація дозволяє упаковувати програми та їх залежності в логічні відокремлені контейнери, які можуть бути запущені та працювати на будь-якій системі, що підтримує Docker, незалежно від її конфігурації.

Це надає нам декілька переваг:

– легкість використання. Docker надає простий спосіб запуску та управління контейнерами. Завдяки контейнеризації, ви можете швидко запускати Redis і мати його налаштованим уніфікованим способом незалежно від операційної системи;

– портативність. Docker контейнери забезпечують велику портативність, оскільки вони включають всі необхідні залежності та конфігурацію. Це означає, що ви можете запакувати Redis із всіма його налаштуваннями в контейнер, а потім перенести його на будь-яку систему, яка підтримує Docker, безперервності роботи;

– ізольованість. Docker контейнери забезпечують ізольоване середовище виконання для Redis. Це означає, що Redis буде працювати в окремому контейнері, не впливаючи на інші складові вашої програмної системи. Це сприяє забезпеченню безпеки, стабільності та надійності системи;

– масштабованість. Docker дозволяє легко масштабувати Redis, запускаючи більше контейнерів з Redis серверами та використовуючи функціонал Docker Swarm або Kubernetes для управління цими контейнерами. Це дозволяє розподіляти навантаження і забезпечувати високу доступність;

– тестування та розробка. Docker дозволяє швидко піднімати тестове середовище з Redis для розробки та валідації вашого додатку. Ви можете використовувати готовий Redis образ з Docker Hub або створити власний образ, додавши до нього ваші дані тестової бази даних.

3.4 Розробка документації для програмного продукту

Великою перевагою телеграм чат-бота є те, що в ньому простий інтерфейс, який буде зрозумілий для всіх. Нижче наведено інструкцію використання розробленого програмного продукту для користувача.

Насамперед потрібно запустити бота натиснувши кнопку «/start», і ввести своє ім’я (рис. 3.1).

Рисунок 3.1 – Початок роботи з ботом

Наступним кроком, потрібно вибрати свою роль і вказати номер телефону для подальшого зв’язку (рис. 3.2).

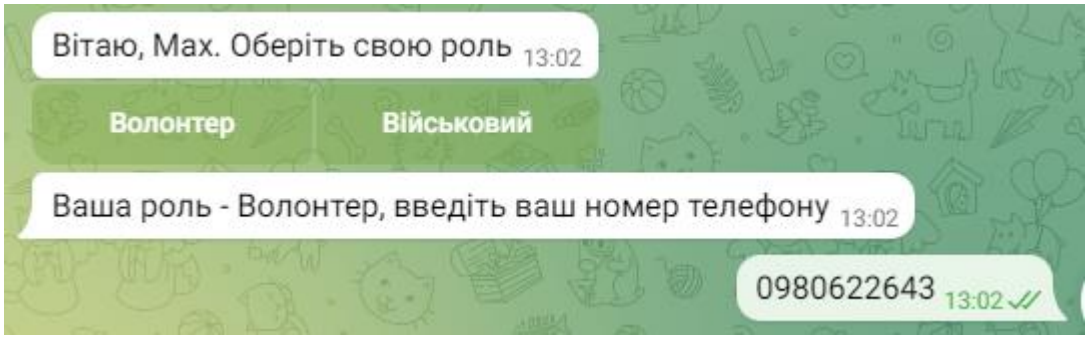

Рисунок 3.2 – Вибір ролі

Після цього ви зможете додати потребу, переглянути список всіх потреб та переглянути список всіх потреб. Додати потребу ви зможете в таких категоріях: збір, медицина, спорядження. А також ввести опис що саме вам потрібно тощо (рис. 3.3).

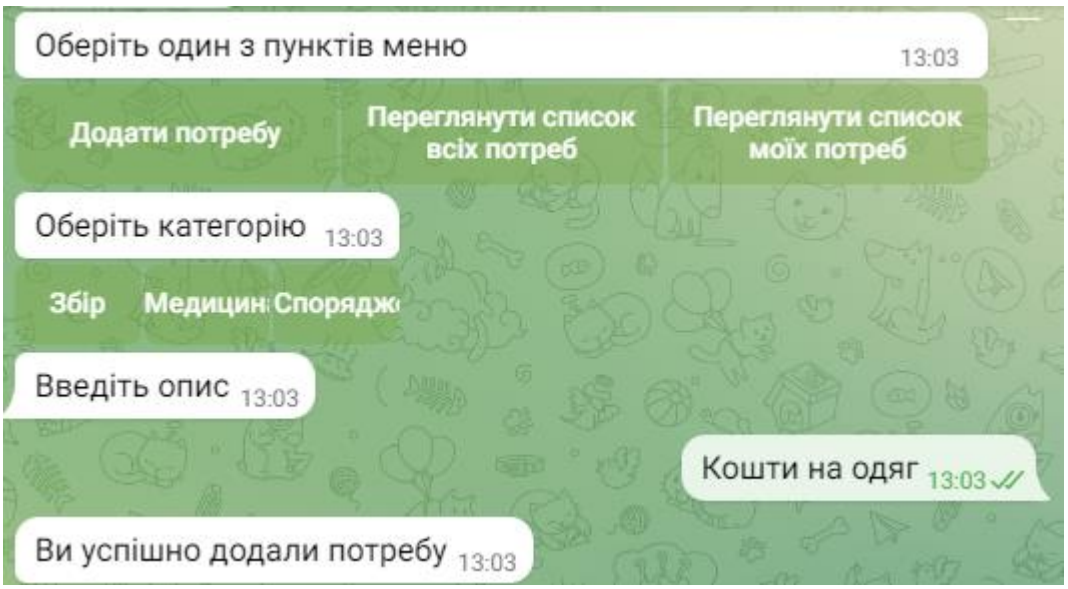

Рисунок 3.3 – Додавання потреби

Переглянувши список всіх потреб ви можете допомогти (рис. 3.4).

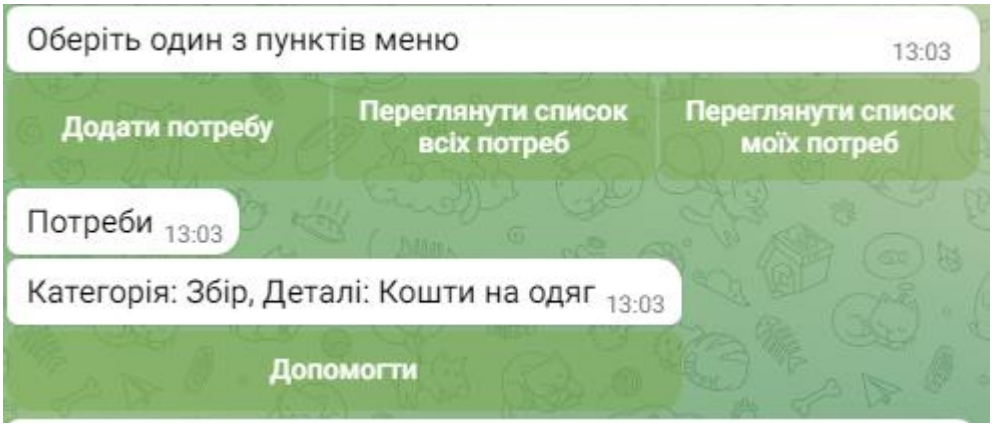

Рисунок 3.4 – Допомога

Також ви можете переглянути список власних потреб і закрити його, якщо вам допомогли (рис 3.5).

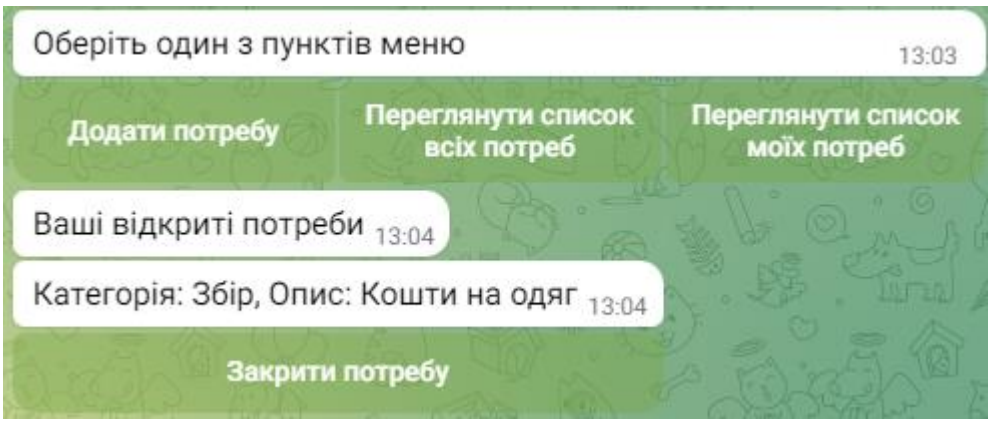

Рисунок 3.5 – Закриття своїх потреб

Висновок до розділу 3

У розділі «Практична реалізація об’єкта проектування» було описано процес реалізації телеграм чат-боту для комунікації волонтерів з військовими. Згідно з розробленою архітектурою системи, було розроблено та реалізовано необхідні компоненти, включаючи логіку обробки повідомлень, управління задачами, збереження даних та інтерфейс користувача.

Була проведена перевірка функціональності та коректності роботи телеграм чат-боту. Було розроблено та виконано різні види тестів, включаючи юніт-тести, інтеграційні тести та функціональні тести, щоб переконатися, що система працює належним чином і задовольняє вимогам, поставленим до неї.

Під час розробки бази даних були використані засоби Redis, які дозволили створити ефективну та швидку систему для збереження та управління даними. Були визначені необхідні схеми та структури даних, а також були реалізовані методи доступу до даних через Redis.

ВИСНОВКИ

Розроблений телеграм чат-боту для комунікації між військовими та волонтерами є потужним інструментом, який може значно полегшити працю волонтерів. А також допоможе військовим швидко закривати свої потреби або просити про допомогу. Цей чат-бот надає можливість швидко закривати потреби військовим та волонтерам тому що, в нього може підключитись безліч людей, які бажають допомогти, або військові, яким потрібна допомога.

Переваги такого чат-боту очевидні. Він знижує затрати часу та зусиль на пошук речей, які потрібні для військових, забезпечує швидку комунікацію між військовими та волонтерами і дозволяє військовим швидко добавляти потреби в будь-який час. Зв’язок з телеграм дозволяє забезпечити простий інтерфейс, а також швидку роботу механізму.

Під час розробки було виконано поставлені завдання

В ході написання пояснювальної записки було доведено актуальність розробки.

Виконано аналіз предметної області використання програмного продукту. Проаналізовано такі аналогічні чат-боти як SaveUA та Help UA | WUBC. Виділено основні вимоги до функціональності чат-боту.

На основі майбутньої функціональності додатку здійснено вибір програмних технологій та засобів реалізації, а саме обрана мова Java для розробки боту та середовище розробки − IntelliJ IDEA.

Досліджено різні API для розробки чат-боту, такі як: BotFather, Dialogflow, Botpress та Telegram Bot API. Для розробки обрано Telegram Bot API. Забезпечено безпеку та конфіденційність даних засобами API.

За допомогою JUnit протестовано розроблений модуль для виявлення та виправлення можливих помилок та недоліків.

Виконано опис розробленого чат-боту, розроблено інструкцію користувача з експлуатації. Підготований звіт про результати роботи.

В результаті виконаних завдань було успішно розроблено телеграм чат-бот для комунікації між військовими та волонтерами. Завдяки цьому чат-боту військові зможуть швидко отримати потрібні їм речі, а волонтери зможуть ефективно і швидко їм допомогти, що сприятиме пришвидшенню нашої перемоги.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Docker: Accelerated, Containerized Application Development. Docker. URL: https://www.docker.com/ (date of access: 18.05.2023).

2. Gradle Build Tool. Gradle. URL: https://gradle.org/ (date of access: 10.05.2023).

3. mr-pickles. Розбираємся з Redis. Habr. URL: https://habr.com/en/companies/wunderfund/articles/685894/ (дата звернення: 16.05.2023).

4. ph_piter. Lombok. Повне керівництво. Habr. URL: https://habr.com/en/companies/piter/articles/676394/ (дата звернення: 01.06.2023).

5. Save UA. Каталог компаній to4ka.fun. URL: https://to4ka.fun/listing/save-ua/ (date of access: 18.04.2023).

6. Telegram Bot API. Telegram APIs. URL: https://core.telegram.org/bots/api (date of access: 19.05.2023).

7. Telegram FAQ. Telegram. URL: https://telegram.org/faq (date of access: 20.05.2023).

8. topjava. Введення в Spring Boot: створення простого REST API на Java. Habr. URL: https://habr.com/en/articles/435144/ (дата звернення: 12.05.2023).

9. Допомога волонтерів: список, телеграм-бот, допомоги в Україні - ВзаємоДія. Допомога волонтерів: список, телеграм-бот, допомоги в Україні - ВзаємоДія. URL: https://viyna.net/dopomoga-bezdpritulnim-tvarinam-u-rivnomu (дата звернення: 14.04.2023).

10. Рильков О. Spring Data JPA. Habr. URL: https://habr.com/en/articles/435114/ (date of access: 14.05.2023).

ДОДАТКИ

Додаток А

Лістинг основних елементів

```
@Configuration
@PropertySource("classpath:application.properties")
public class ApplicationConfig {
    @Bean
    public static PropertySourcesPlaceholderConfigurer
propertySourcesPlaceholderConfigurer() {
        return new PropertySourcesPlaceholderConfigurer();
    }

    @Bean
    TelegramBotsApi telegramBotsApi() throws TelegramApiException {
        return new TelegramBotsApi(DefaultBotSession.class);
    }
}

@Configuration
@EnableRedisRepositories
public class RedisConfiguration {

    @Bean
    public JedisConnectionFactory
redisConnectionFactory(@Value("${spring.redis.host}") String
redisHost,

@Value("${spring.redis.port}") int redisPort) {
        return new JedisConnectionFactory(new
RedisStandaloneConfiguration(redisHost, redisPort));
    }

    @Bean
    public RedisTemplate<?, ?> redisTemplate(JedisConnectionFactory
connectionFactory) {
        RedisTemplate<byte[], byte[]> template = new
RedisTemplate<>();
        template.setConnectionFactory(connectionFactory);
        return template;
    }
}


@Slf4j
@Service
@RequiredArgsConstructor
public class AddTaskConversation implements Conversation {
    private final TaskRepository taskRepository;

    @Override
    public List<ConversationStep> flows() {
```

```
        return Lists.newArrayList(
            (context, input) -> new SuccessfulConversationResponse(
                context.getId(),
                Lists.newArrayList(new Reply(
                    "Оберіть категорію",
                    Arrays.stream(TaskCategory.values())
                          .map(TaskCategory::getDisplayName)
                          .map(r -> new Option(r, r))
                          .collect(Collectors.toList()))
                )
            ),
            (context, input) -> {
                if(TaskCategory.get(input.trim()) == null) {
                    return new
RetryConversationResponse(context.getId(), Lists.newArrayList(new
Reply(
                        "Ви вказали неіснуючу категорію")
                    ));
                }
                context.getContext().put("category", input);
                return new SuccessfulConversationResponse(
                    context.getId(),
                    Lists.newArrayList(new Reply(
                        "Введіть опис")
                    )
                );
            },
            (context, input) -> {
                context.getContext().put("description", input);
                String category =
context.getContext().get("category").trim();
                String description =
context.getContext().get("description");
                taskRepository.save(
                    new Task(
                        context.getId(),
                        TaskCategory.get(category),
                        description
                    )
                );
                return new SuccessfulConversationResponse(
                    context.getId(),
                    Lists.newArrayList(new Reply(
                        "Ви успішно додали потребу")
                    ),
                    true
                );
            }
        );
    }
}
```

```
@Slf4j
@Service
@RequiredArgsConstructor
public class CloseTaskConversation implements Conversation {
    private final TaskRepository taskRepository;

    @Override
    public List<ConversationStep> flows() {
        return Lists.newArrayList(
            (context, input) -> {
                Long taskId =
Long.parseLong(context.getContext().get("taskId"));
                Optional<Task> optTask =
taskRepository.findById(taskId);
                if (!optTask.isPresent()) {
                    return new SuccessfulConversationResponse(
                        context.getId(),
                        Lists.newArrayList(new Reply(
                            "Неіснуюча потреба")
                        ),
                        true
                    );
                } else {
                    Task task = optTask.get();
                    taskRepository.deleteById(taskId);
                    return new SuccessfulConversationResponse(
                        context.getId(),
                        Lists.newArrayList(new Reply(
                            String.format("Ви успішно закрили
потребу: %s", task.getDescription()))
                        ),
                        true
                    );
                }
            }
        );
    }
}
```

```
     @Override
         public        List<ConversationStep>        flows()        {
             return                                  Lists.newArrayList(
                 (context,            input)            ->            {
                     Long                    taskId                  =
     Long.parseLong(context.getContext().get("taskId"));
                     Optional<String>          creatorIdOpt          =
     taskRepository.findById(taskId).map(Task::getCreator);
                     Optional<User>            creatorOpt            =
     creatorIdOpt.flatMap(userRepository::findById);
                     if(!creatorOpt.isPresent())                     {
                         return  new  SuccessfulConversationResponse(
```

```
                                context.getId(),
                                Lists.newArrayList(new              Reply(
                                    "Неіснуюча                    потреба")
                                ),
                                true
                            );
                        }                       else                        {
                            User     creator     =     creatorOpt.get();
                            return  new  SuccessfulConversationResponse(
                                context.getId(),
                                Lists.newArrayList(
                                    new                                 Reply(
                                        String.format("Дякуємо,
    контактна          особа          %s,          тел.          %s",

    creator.getName(),                              creator.getPhone())
                                    )
                                ),
                                true);
                        }
                    }
                );
            }
        }

@Slf4j
@Service
@RequiredArgsConstructor
public class RegistrationConversation implements Conversation {
    private final UserRepository userRepository;

    @Override
    public List<ConversationStep> flows() {
        return Lists.newArrayList(
            (context, input) -> new SuccessfulConversationResponse(
                context.getId(),
                Lists.newArrayList(new Reply("Вітаю, введіть своє
ім'я"))
            ),
            (context, input) -> {
                context.getContext().put("name", input);
                return new SuccessfulConversationResponse(
                    context.getId(),
                    Lists.newArrayList(new Reply(
                        String.format("Вітаю, %s. Оберіть свою
роль", input),
                        Arrays.stream(Role.values())

.filter(Role::showRole).map(Role::getDisplayName)
                            .map(r -> new Option(r, r))
                              .collect(Collectors.toList()))
                    )
```

```
                );
            },
            (context, input) -> {
                context.getContext().put("role", input);
                return new SuccessfulConversationResponse(
                    context.getId(),
                    Lists.newArrayList(new Reply(
                        String.format("Ваша роль - %s, введіть ваш
номер телефону", input))
                    )
                );
            },
            (context, input) -> {
                context.getContext().put("phone", input);
                if
(!input.matches("(\\+\\d{1,2}\\s)?\\(?(\\d{3})\\)?[\\s.-
]?(\\d{3})[\\s.-]?(\\d{4})")) {
                    return new RetryConversationResponse(
                        context.getId(),
                        Collections.singletonList(
                            new Reply("Введіть справжній номер
телефону")
                        )
                    );
                }
                userRepository.save(
                    new User(
                        context.getId(),

Role.get(context.getContext().get("role").trim()),
                        context.getContext().get("name"),
                        context.getContext().get("phone")
                    )
                );
                return new SuccessfulConversationResponse(
                    context.getId(),
                    Lists.newArrayList(new Reply(
                        "Ви успішно зареєстровані в системі")
                    ),
                    true
                );
            }
        );
    }
}

public class ShowAllTasksConversation implements Conversation {
    private final TaskRepository taskRepository;

    @Override
    public List<ConversationStep> flows() {
        return Lists.newArrayList(
```

```
            (context, input) ->
                new SuccessfulConversationResponse(
                    context.getId(),
                    Stream.concat(
                        Stream.of(new Reply("Потреби")),
                        StreamSupport

.stream(taskRepository.findAll().spliterator(), false)
                                .map(t ->
                                        new Reply(
                                            String.format("Категорія:
%s, Деталі: %s", t.getCategory().getDisplayName(),
t.getDescription()),
                                            Collections.singletonList(
                                                new Option(
                                                    "Допомогти",

String.format("helpWithTaskConversation?taskId=%s", t.getId())
                                                )
                                            )
                                        )
                                )
                    ).collect(Collectors.toList()),
                    true
                )
        );
    }
}
```