МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему:

«Інформаційна технологія керування людськими ресурсами під час технічного обслуговування серверу»

Завідувач випускаючої кафедри — **Довбиш А. С.**

Керівник роботи — **Берест О. Б.**

Студент групи ІН.м-02 — **Черняк І. Ю.**

СУМИ 2021

Сумський державний університет

(назва вузу)

Факультет *ЕлІТ* Кафедра *Комп'ютерних наук*

Спеціальність *«122 - Комп'ютерні науки»*

Затверджую:

зав.кафедрою _____

_____

"_____" _____ 20___р.

ЗАВДАННЯ

НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТОВІ

Черняку Івану Юрійовичу

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) *Інформаційна технологія керування людськими ресурсами під час технічного обслуговування серверу*

затверджую наказом по інституту від "____" _____ 20 ___ р. № _____

2. Термін здачі студентом закінченого проекту (роботи)_____

3. Вхідні данні до проекту (роботи) _____

_____

_____

_____

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)

1) Огляд методів уніфікації процесів та програмних рішень для керування процесами; 2) Постановка завдання й формування завдань дослідження; 3) Вибір апаратно-програмного інструментарію; 4) Розробка інформаційного й програмного забезпечення інтелектуальної системи; 5) Аналіз результатів розробки та роботи додатку.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) _____

_____

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання _____

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів дипломного проекту (роботи)	Термін виконання проекту (роботи)	Примітка
1.	Огляд методів уніфікації процесів та програмних рішень для керування процесами		
2.	Постановка задачі та формування завдань дослідження.		
3.	Проектування інформаційної системи		
4.	Програмна реалізація інформаційної системи		
5.	Оформлення пояснювальної записки до кваліфікаційної магістерської роботи		

Студент – дипломник _____
(підпис)

Керівник проекту _____
(підпис)

РЕФЕРАТ

Записка: 48 стор., 24 рис., 0 табл., 1 додаток, 24 джерела.

Об’єкт дослідження — автоматизований процес керування людськими ресурсами.

Мета роботи — дослідження та розробка рішення для автоматизації процесу керування людськими ресурсами в процесі деплойменту із використанням чат-боту.

Методи дослідження — метод розробки чат-боту з використанням API.

Результати — розроблено додаток для автоматизації керування процесом. При цьому використано програмне рішення Camunda для побудови та управління процесами. Додаток реалізовано у формі чат-боту, створеного за допомогою мови програмування Java та Telegram API.

WORKFLOW, АВТОМАТИЗАЦІЯ ПРОЦЕСІВ, КЕРУВАННЯ РЕСУРСАМИ, DEPLOYMENT, CAMUNDA, ЧАТ-БОТ, TELEGRAM

ЗМІСТ

ВСТУП

Із розвитком IT-компаній та їх клієнтів посилюються вимоги клієнтів до продуктів та послуг, що їм надаються. А сама підтримка клієнта стає все більш складною та напруженішою. Зокрема, однією із найважливіших складових пітримки є деплоймент – процес встановлення кінцевого програмного продукту на сервер клієнта.

У швидкому, якісному та надійному деплойменті зацікавлені як клієнти, так і співробітники компанії. Адже не завжди цей процес відбувається як заплановано, а затримки та проблеми під час деплойменту часто можуть приводити до штрафів та репутаційних втрат компанії.

Тому виникає закономірне питання: як пришвидшити та уніфікувати процес деплойменту?

Адже у великих компаніях та для великих клієнтів процес деплойменту може включати в себе сотні кроків та тривати від декількох годин до декількох днів, а в самому процесі можуть приймати участь працівники із багатьох команд, різних часових зон та робочих змін.

До того ж у більшості випадків такі процеси відбуваються у чатах різних месенджерів під керівництвом людини, що підвищує ризик людської помилки: помилки під час виконання самого кроку, несвоєчасно виконані або ж зовсім пропущені кроки.

Також підхід уповільнює аналіз вже виконаного деплойменту (ручне оновлення плану, пошук ключових повідомлень у чатах) та подальші оновлення плану.

Одним з головних питань, що постають тут є автоматизація та уніфікація процесу та задач деплойменту. Наприклад, видача задачі конкретній людині, обробка повідомлень чату й автоматичне підтримування актуального статусу.

Із вирішенням цієї проблеми можуть допомогти чат-боти для популярних месенджерів. Боти можуть реалізувати широкий функціонал: від простого надсилання повідомлення в чат до взаємодії із сервером та базою даних через API. До того ж використання чат боту допоможе уніфікувати процес деплойменту, полегшити процес керування ресурсами, збільшити прозорість процесу в цілому.

Отже, метою даної роботи можна вважати дослідження та розробку рішення для автоматизації процесу керування людськими ресурсами в процесі деплойменту із використанням чат-боту.

1 ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Огляд методів уніфікації процесів

Основою усього процесу деплойменту є структурований план, що містить у собі конкретні задачі – кроки із назвою, відповідальною командою, часом виконання, порядком задачі та залежностями на інші кроки. Задачі у такому процесі пов’язані між собою: результат однієї задачі являє собою вхідні дані для іншої або одна задача не може розпочатися поки не буде виконана інша. До того ж усі задачі мають одну кінцеву мету – завершений процес деплойменту.

Тобто по суті план являє собою потік роботи або ворфлоу (workflow), а його планування, виконання та координацію називають оркеструванням (orchestration) [1].

Зазвичай воркфлоу використовують для керування автоматизованими програмними процесами, де участь людини мінімальна та обмежується запуском та моніторингом процесу. Головну роль тут відіграє послідовність виконання задач.

Ще одним популярним напрямком використання воркфлоу є бізнес-процеси. Це вже автоматизовані бізнес-рішення, де більшу роль відіграють якісь дані та те у якій послідовності вони будуть оброблені.

Оркестрація у більшості випадків здійснюється за допомогою workflow engine (WE) – спеціального програмного забезпечення, що автоматично виконує кроки процесу.

Для досягнення мети даної робити потрібне рішення для організації й управління ручних кроків та людських ресурсів.

Розглянемо приклади популярних готових програмних рішень для побудови та управління процесами [2], проаналізуємо їхній функціонал та

основні можливості при роботі з вокфлоу та виділимо ті особливості, що можна використати при розробці власного рішення.

1.2 Camunda

Одним із популярних готових рішень для підтримки бізнес-процесів для середнього та великого бізнесів є Camunda [3]. Camunda дозволяє інтегрувати в процеси ручні задачі, взаємодію з API, мікросервісами, Інтернетом речей та штучним інтелектом.

Даний програмний продукт реалізує принципи специфікації Business Process Model and Notation (BPMN) [4], що надає можливість користувачам описати процеси у графічний та уніфікований спосіб.

Тобто робота тут починається саме з опису бізнес-процесу в графічній нотації. Його можна змоделювати в окремому додатку Camunda Modeler (рис. 1.1). Потім можна запускати процеси та керувати задачами через веб-інтерфейс (рис. 1.2, 1.3 та 1.4).

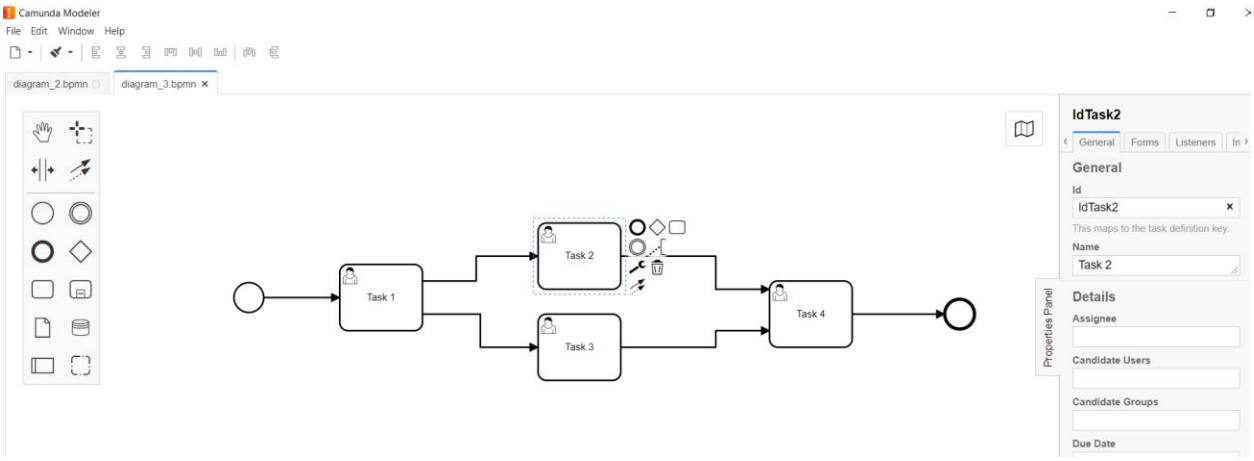

Рисунок 1.1 – Інтерфейс додатку Camunda Modeler

На рисунку 1.1 зображено графічне представлення процесу із його задачами та можливостями для редагування атрибутів задач.

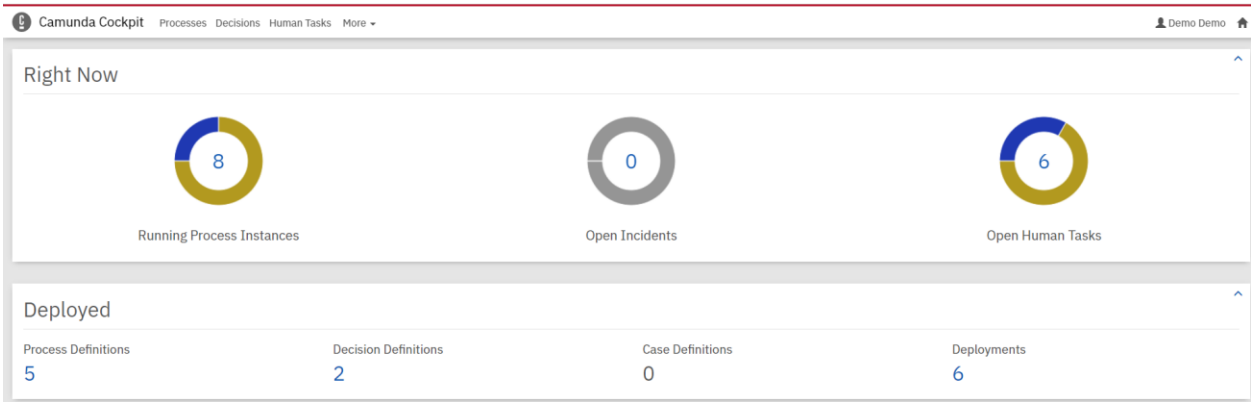

Рисунок 1.2 – Веб-інтерфейс платформи Camunda зі статистикою по процесам

У користувацькому інтерфейсі платформи Camunda можна переглянути статистику процесів. Наприклад, на рисунку 1.2 можно побачити кількість запущенних процесів, помилок виканання задач та відкритих ручних задач.

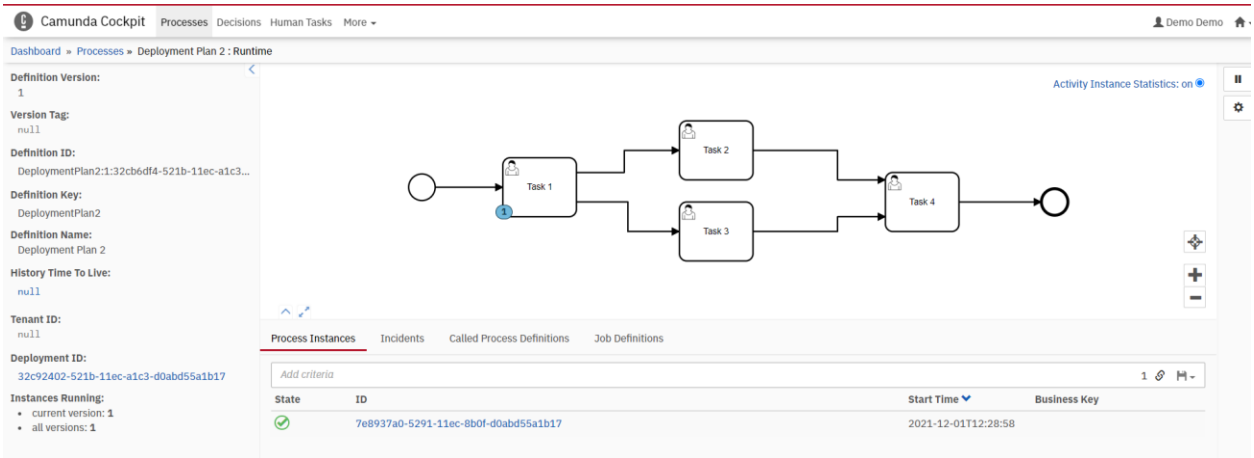

Рисунок 1.3 – Веб-інтерфейс платформи Camunda із конкретним процесом

Користувацький інтерфейс платформи Camunda надає можливість переглядати інформацію про поточний стан конкретного процесу. Наприклад, на рисунку 1.3 зображено поточний стан процесу, час його запуску та задачі, що виконуються в даний момент.

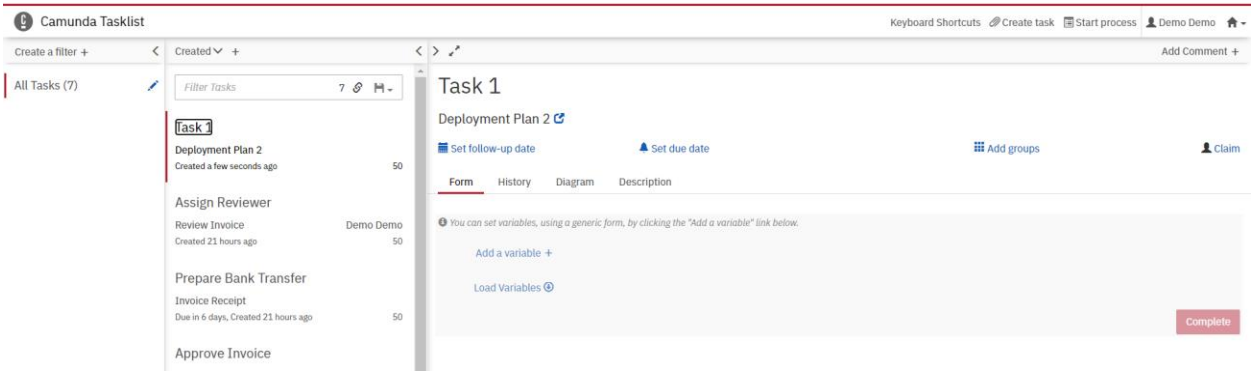

Рисунок 1.4 – Веб-інтерфейс платформи Camunda із конкретною задачею

Також за допомогою веб-інтерфейсу можна переглянути інформацію про конкретну задачу. На рисунку 1.4 зображено задачу «Task 1», до якого процесу вона відноситься («Deployment Plan 2») тощо.

Camunda реалізована на мові програмування Java, за замовчуванням використовує базу даних H2 Database (можна налаштувати на використання іншої СКБД).

Отже, до переваг Camunda можна віднести:

- створення воркфлоу за допомогою графічної нотації;
- обширна документація;
- наявність ручних та персоналізованих задач;
- open-source версія та платна версія з підтримкою від вендора.

До недоліків:

- високий поріг для старту використання системи.

1.3 Apache Airflow

Іншим прикладом системи для керування воркфлоу є Apache Airflow [5]. Це програмний продукт з відкритим вихідним кодом, реалізований на мові програмування Python.

Airflow оперує воркфлоу як орієнтованим ациклічним графом (Directed Acyclic Graph, DAG).

Основним застосуванням Airflow – є автоматизація й управління задачами, що більше пов’язані з виконанням коду та скриптів.

Конфігурація самого процесу відбувається за допомогою коду, де описується сам процес та його задачі (рис. 1.5). Потім за допомогою веб-інтерфейсу можно здійснювати керування процесом (рис. 1.6 та 1.7).

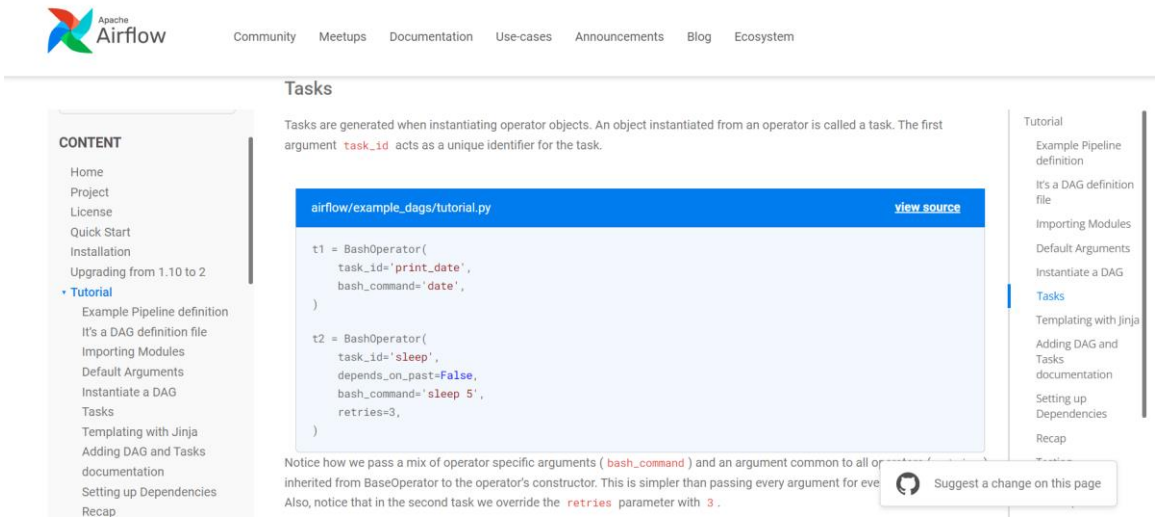

Рисунок 1.5 – Приклад конфігурації задачі в Apache Airflow

Задачі для Apache Airflow описуються за допомогою мови програмування Python. Наприклад, на рисунку 1.5 наведено приклади конфігурації задач «t1» та «t2». А сама задача «t1» з ідентифікатором «print date» виконає Bash-команду «date».

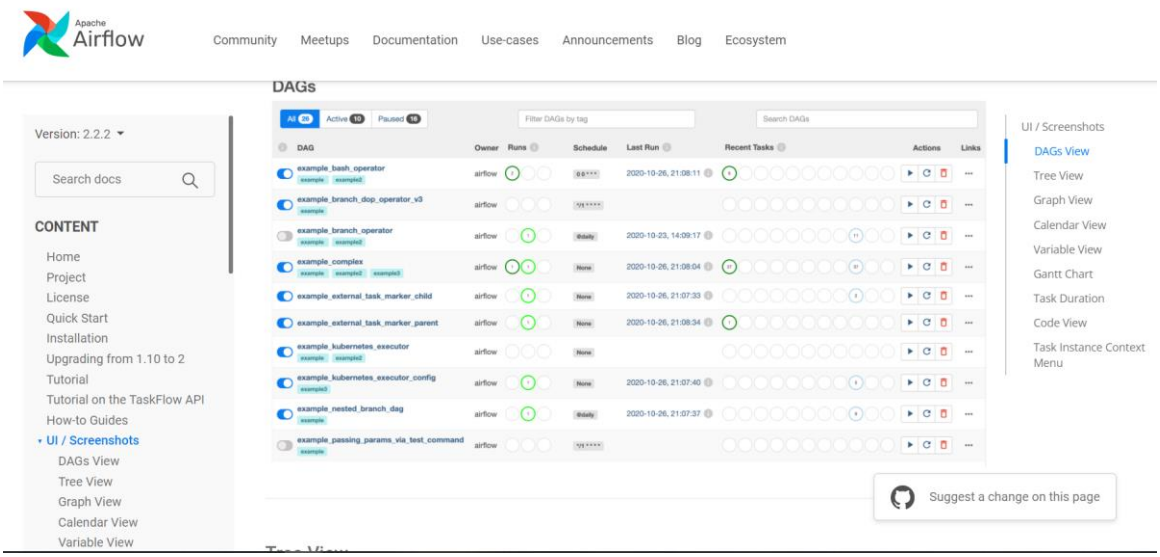

Рисунок 1.6 – Приклад веб-інтерфейсу Apache Airflow зі списком процесів

Після конфігурації задач процесу ми маємо змогу переглянути список усіх процесів в системі. На рисунку 1.6 наведено приклад користувацького інтерфесу зі списком процесів та їх атрибутів («Owner», «Runs», «Last Run» тощо) та дій (запустити, видалити).

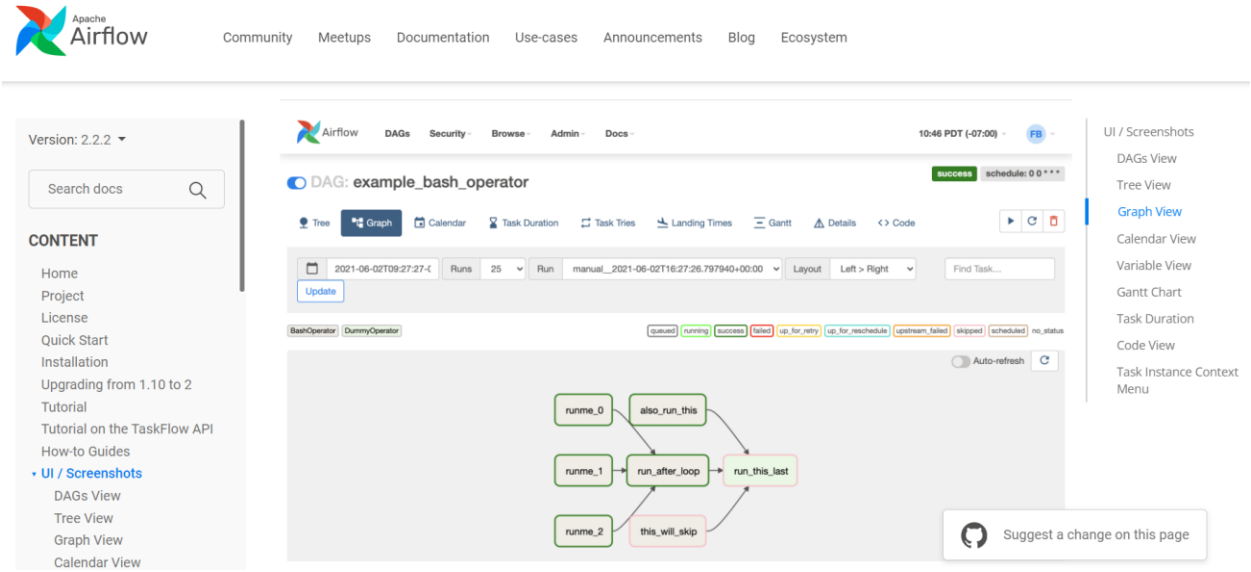

Рисунок 1.7 – Приклад веб-інтерфейсу Apache Airflow із конкретним процесом

Також Apache Airflow надає можливість переглядати стан конкретного процесу зі списком задач. На рисунку 1.7 можемо побачити процес «example_bach_operator» та список його задач («runme_0», «runme_1» тощо).

Отже, до переваг Apache Airflow можна віднести:

- обширну документацію;
- open-source версію.

До недоліків:

- відносно високий поріг для старту використання системи;
- відсутність графічного інтерфейсу для моделювання процесу.

Беручи до уваги те, що дана робота розглядає процес, де задіяно людські ресурси та ручні задачі, то ми можемо використати програмне рішення Camunda для досягнення цілей роботи. Також перевагою цієї платформи є наявність графічного інтерфейсу для побудови процесу, що полегшує сприйняття та досвід користування для кінцевого користувача.

1.4 Постановка задачі

Отже, основною задачею даної роботи є побудова системи управління процесом та розробка Telegram-боту, що дозволяє користувачеві:

- видати задачу користувачеві;
- перевіріти статус задачі;
- завершити задачу;

Ще однією ціллю даної роботи є поглиблення знань щодо побудови та управляння процесами, зокрема використовуючи нотацію BPMN та програмне рішення Camunda.

Для досягнення мети можна виділити наступні задачі:

- виконати проектування інформаційної системи;
- провести аналіз та побудову моделі програмного продукту;

- реалізувати програмний продукт;
- провести тестування програмного продукту.

Результатом роботи має стати чат-бот Telegram, що відповідає поставленим вимогам та є зрозумілим у використанні.

2 ВИБІР МЕТОДІВ РІШЕННЯ ТА ПОСТАНОВКА ЗАДАЧІ

2.1 Вибір апаратно-програмного інструментарію

Для реалізації проекту було обрано програмну платформу Camunda. Вона написана з використанням мови програмування Java та СКДБ H2.

Тому і мову програмування для реалізації чат боту системи було обрано Java [6].

Це проста в засвоєнні мова програмування, що має знайомий та простий синтаксис, який базується на синтаксисі C++, та є зрозумілим багатьом розробникам [7].

Це об’єктно-орієнтована мова програмування. Вона дозволяє використовувати усі переваги даної парадигми такі як наслідування, інкапсуляція та поліморфізм.

Java приділяє також багато уваги продуктивності програми та звільняє розробників від самостійного контролю ресурсів пам’яті, що надає більше часу та можливостей для реалізації логіки програми.

Java має велику спільноту розробників, яка може надати підтримку в разі виникнення питань під час роботи.

Сама ж мова дозволяє ефективно та легко реалізувати логіку будь-якої складності, використовуючи велику кількість вбудованих інструментів самої мови (колекції, робота з датою та часом, інструменти доступу до бази даних тощо) та сторонніх фреймворків та бібліотек. Використання бібліотек дозволяє суттєво зменшити час розробки програмного продукту. Часто вони також мають відкритий програмний код та та підтримуються досвідченими Java-розробниками.

Це робить Java гарним вибором у якості мови програмування для реалізації додатку.

Для системи керування базою залишимо H2 Database [8].

Це система для управління реляційними базами даних з відкритим вихідним кодом. Для доступу до даних використовує SQL. Може зберігати дані як на диску, так і в пам’яті.

Це досить проста у використанні та підтримці база даних, що може бути запущена на сервері з досить обмеженими ресурсами.

У якості месенджера для чат-боту було обрано Telegram [9].

Telegram-бот - це особливий тип користувача, який є не людиною, а комп’ютерною програмою, що може надавати користувачам додаткові функції, такі як надсилання інформації, нагадування, відтворення мелодій тощо. Бот може опублікувати повідомлення в групі або каналі.

Telegram надає безкоштовне API для створення ботів для соціальних взаємодій, ігор та послуг електронної комерції.

До переваг чат-боту в Telegram можна віднести:

- безкоштовність платформи. Telegram є безкоштовним, незалежно від кількості повідомлень і мети використання (для професійного чи особистого користування). Telegram безкоштовний як для користувачів, так і для бізнесу. Створення телеграм-бота також безкоштовне;
- безпека. Враховуючи глобальні тенденції щодо випадків злому на кожній платформі соціальних мереж, Telegram вважається досить безпечним, головним чином, тому що повідомлення надсилаються через платформу в зашифрованому вигляді. Це є досить вагомою причиною використовувати його в бізнес-цілях, оскільки він захистить дані системи та користувачів;
- доступність. Месенджер Telegram охоплює всі основні платформи, такі як телефони під управлінням Android та iOS; десктоп-додатки для Mac, Linux та Windows. Крім того, він також має веб-версію. Це значно полегшує доступ користувачів до боту та вирішує питання крос-платформенності кінцевого програмного продукту.

2.2 Вибір середовища розробки

Для написання, тестування та відлагодження програмного коду додатку найкраще використати інтегроване середовище розробки (Integrated Development Environment, далі IDE).

IDE – це система, що підтримує усі стадії розробки програмного додатку: від написання коду до деплойменту готового додатку на сервері.

Переваги використання IDE:

- мінімальні витрати часу та зусиль для написання програмного коду;
- наявність інструментів управління кодом для автоматизації багатьох рутинних операцій;
- використання інструментів відлагодження для тестування коду під час розробки додатку.

У якості IDE для розробки програмних додатків за допомогою Java було обрано IntelliJ IDEA [10]. IntelliJ IDEA розроблюється та підтримується компанією JetBrains. Дане середовище доступне як у безкоштовній, так і в комерційній версіях. Версії різняться наявними інструментами, але основний функціонал доступний в обох версіях.

IntelliJ IDEA дає пропозиції щодо швидкого завершення коду, аналізу коду та надійних інструментів дебагу та рефакторингу.

Особливості:

- розумне завершення коду: програмісту подається список найбільш релевантних символів, що можна застосувати до поточного контексту. Він пропонує останні використані класи, змінні, методи, тощо. Таким чином це пришвидшує написання коду;
- IntelliJ IDEA пропонує ефективний та потужний рефакторинг: виокремлення констант, змінних, методів;

- IntelliJ IDEA має велику кількість вбудованих інструментів, таких як декомпілятор, системи контролю версій, консоль база даних SQL тощо;
- потужний компілятор, що може виявляти дублікати, неефективне написання коду тощо.

Intellij IDEA має безкоштовну учнівську ліцензію для студентів, що є великим плюсом даного продукту.

Для роботи над проектом було обрано Intellij IDEA через переваги, що пов’язані із аналізом коду та авто-доповненням.

3 ІНФОРМАЦІЙНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ КЕРУВАННЯ ЛЮДСЬКИМИ РЕСУРСАМИ ПІД ЧАС ТЕХНІЧНОГО ОБСЛУГОВУВАННЯ СЕРВЕРУ

3.1 Проектування інформаційної системи

На початку роботи над проектом розробники часто прагнуть якнайшвидше приступити до програмної реалізації та написання коду. Часто такий підхід призводить до складного програмного коду із непотрібними функціями або модулями, архітектури програмного продукту, що розуміють лише самі розробники, прострочення термінів роботи та до згортання проекту, у найгіршому випадку.

Саме тому етап проектування та планування стає важливою фазою процесу розробки.

На цьому етапі формують чіткі функціональні та нефункціональні вимоги до продукту, після чого відбувається планування архітектури та програмних модулів для реалізації [11]. Саме попереднє планування може виявляти слабкі місця програмного продукту, його потенційні проблеми продуктивності навіть до старту програмної реалізації. У подальшому це може зекономити людські та матеріальні ресурси при розробці та підтримці системи.

Саме ж планування, у залежності від поставленої задачі, може відбуватися як на звичайній дошці чи аркуші паперу, так і згідно з міжнародними стандартами документації.

Під час планування також важливо доступно та зрозуміло донести інформацію про функції системи до тих, хто працює зараз із нею, та тих, хто буде супроводжувати її далі.

Для цього чудово підійдуть графічні інструменти моделювання.

Розглянемо їх основні види.

Модель ERD

Для того, щоб спроектувати та зобразити основні сутності системи, буде використано Entity relation діаграму (ERD). Вона призначена для проектування даних, що потім будуть зберігатися в системі керування баз даних. Також дана діаграма дозволяє показати зв'язки та залежності між даними. Проектування ERD розпочинається одним із перших та дозволяє показати типи даних системи, їх обмеження та можливості для подальшого проектування.

ER-діаграма оперує наступними поняттями:

- сутності – об'єкти уявного або реального світу, дані про які будуть зберігатися та оброблятися в системі;
- атрибути – характеристики або властивості сутності;
- зв'язки – зображають як сутності взаємодіють одна з одною.

За допомогою ERD можна спроектувати нову базу даних або ж використати для опису вже існуючої.

Згідно зі статтями [12, 13] до переваг даної моделі можна віднести:

- незалежність від СКБД;
- візуальне представлення структури бази даних;
- легкість у використанні;
- високу сумісність із реляційними таблицями – готову діаграму можно легко конвертувати в таблицю.

Серед недоліків ERD можна виділити:

- відсутність єдиного стандарту нотації;
- складність у зображенні перетворень даних.

Модель UML

UML – це стандартизована мова моделювання, що складається з діаграм, розроблених для візуалізації, ідентифікації, конструювання та

документування програмних систем [14]. UML складається з найкращих практик та рішень, що є ефективними при вирішенні проблем моделювання складних і великих програмних систем.

Використання UML є важливою частиною розробки об'єктно-орієнтованого програмного рішення. Даний стандарт дозволяє командам розробників легше та краще розуміти один одного та ефективно вести комунікацію на одному рівні абстракції.

Серед переваг використання UML можна виділити [15]:

- поширеність. Чимало розробників вже знайомі з основними концептами, інструментами та діаграмами UML, що значно полегшує комунікацію в команді;
- відокремленість від мови програмування. UML надає можливості, що показують функціонал та архітектуру програми, не зважаючи на те яка мова програмування буде використана у подальшому для реалізації;
- обширні можливості для опису. UML має велику кількість діаграм, що описують систему з різних функціональних точок зору.

Перейдемо безпосередньо до реалізації зазначених вище методів пректування для нашої системи.

Реалізація ERD

Для ефективної реалізації ER-діаграми, варто дотримуватися наступних кроків:

1. визначити сутності системи;
2. зазначити з'язки між сутностями;
3. описати взаємозв'язки;
4. описати атрибути сутностей.

Тепер можемо перейти до завершення створення діаграми, результат зображено на рисунку 3.1.

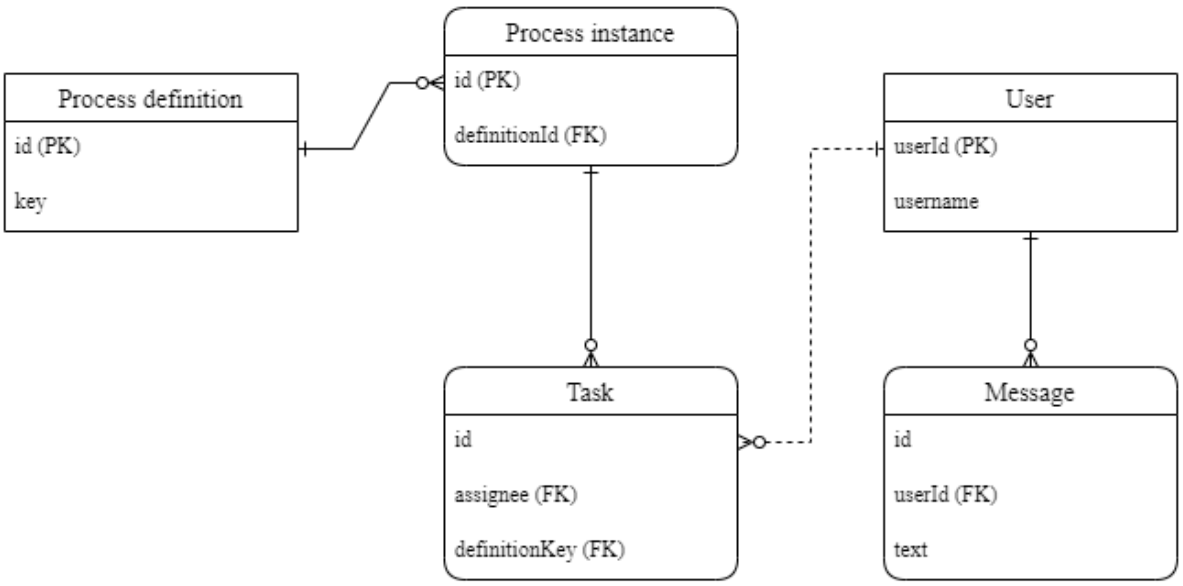

Рисунок 3.1 – ER-діаграма інформаційної системи

Система в основному використовує сутності, що оголошені на рівні сутностей платформи Camunda та її API [16] та рівні Telegram API [17].

Розглянемо найлоговніші з них та використані атрибути:

- Process Definition [18] – сутність Camunda, що описує план процесу. Для використання в даній роботі нам вистачить наступних атрибутів «id» та «key» – назва процесу в XML-схемі BPMN.
- Process Instance [19] – сутність Camunda, що описує створений конкретний об’єкт плану процесу. У системі використовується атрибут «id». Очевидно, що об’єкт плану процесу не може бути створений без самого плану процесу. Можемо виділити сильний зв’язок один до багатьох.
- User [20] [21] – сутність Telegram та сутність Camunda, представляє собою користувача. Для системи з точки зору Telegram важливим є атрибут «username». З перспективи Camunda для системи є важливим атрибут «userId».
- Task [22] – сутність Camunda, для опису задачі, що видана користувачеві. У цієї сутності важливі для системи атрибути – «id», «definitionKey» (назва

задачі в XML-схемі BPMN), «assignee» (id користувача, якому видана задача). Задача не може бути створена без конкретного об’єкту процесу – можемо виділити сильний зв’язок один до багатьох. Задача може бути спочатку створена та не бути видана жодному користувачеві – тут присутній слабкий зв’язок.

- Message [23] – сутність Telegram, що описує об’єкт повідомленння у чаті. Зрозуміло, що повідомлення не може бути надіслане без користувача – виділяємо сильний зв’язок один до багатьох. Для нас важливими є атрибути «text» та «MessageEntity» [24] - представляє спеціальну сутність у текстовому повідомленні. Наприклад, хештеги, імена користувачів, команди для бота.

Реалізація діаграми варіантів використання

Для того, щоб показати можливі сценарії взаємодії із системою, використаємо діаграму варіантів використання. Модель варіантів використання є однією з поведінкових діаграм UML та показує функціональні вимоги до системи у вигляді варіантів використання (прецедентів). Усі варіанти використання та актори системи позначені на діаграмі варіантів використання (рис. 3.2).

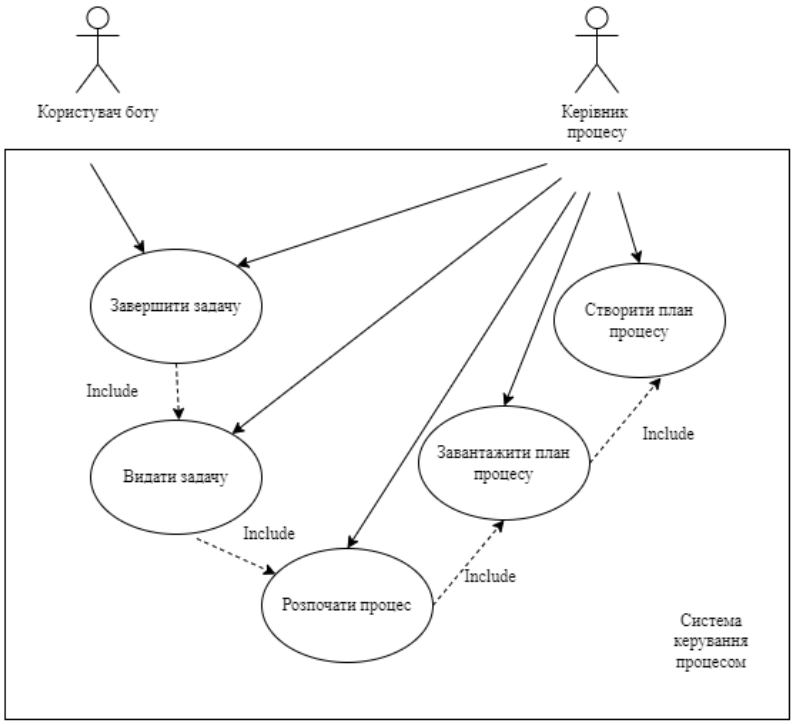

Рисунок 3.2 – Діаграма варіантів використання системи

У нашій системі присутні два типи акторів (користувачів):

- користувач боту – користувач, що отримує задачу та звітує про її виконання;
- керівник процесу – користувач, який адмініструє процес (створює його план та розподіляє залачі між іншими користувачами).

Згідно з вимогами, що були поставлені до системи та до боту зокрема, можемо виділити наступні варіанти використання:

- створити план процесу – створення плану системи за допомогою інструментів платформи Camunda;
- завантажити план процесу – завантажити попередньо створений план в систему Camunda для його подальшого використання;
- розпочати процес – запуск процесу за попередньо створеним та завантаженим планом;

- видати задачу – назначити конкретну задачу для конкретного користувача в розпочатому процесі;
- завершити задачу – завершити раніше видану задачу.

Реалізація діаграми класів

Діаграма класів – це одна із головних діаграм при проектуванні об'єктно-орієнтованої системи. Це структурна UML-діаграма. Вона показує класи програмного додатку, методи та атрибути класів та їх зв'язки між собою.

На діаграмі клас позначається за допомогою прямокутника з назвою класу, атрибутами класу та методами. Різні типи стрілочок позначають різні зв'язки між класами.

Розглянемо основні зв'язки, що використовуються при побудові діаграми класів:

- асоціація – позначає наявність деякого зв'язку між класами.
- агрегація – позначається в тому випадку, якщо один з класів є деякою сутністю, що включає в себе в якості складових частин інші сутності.
- наслідування – показує зв'язок між батьківськими та дочірніми сутностями.

Основним класом системи є клас DeploymentBot, що наслідує клас TelegramLongPollingBot – клас із бібліотеки Telegram API для роботи із ботом та сутностями Telegram.

Для уніфікації роботи із командами користувача в повідомленнях створимо клас Command, де будемо обробляти сутності MessageEntity із повідомлення.

Для формування діаграми класів використаємо вбудований інструментарій середевища розробки IntelliJ IDEA.

Діаграму класів разом із позначенням зв'язків наведено на рисунку 3.3.

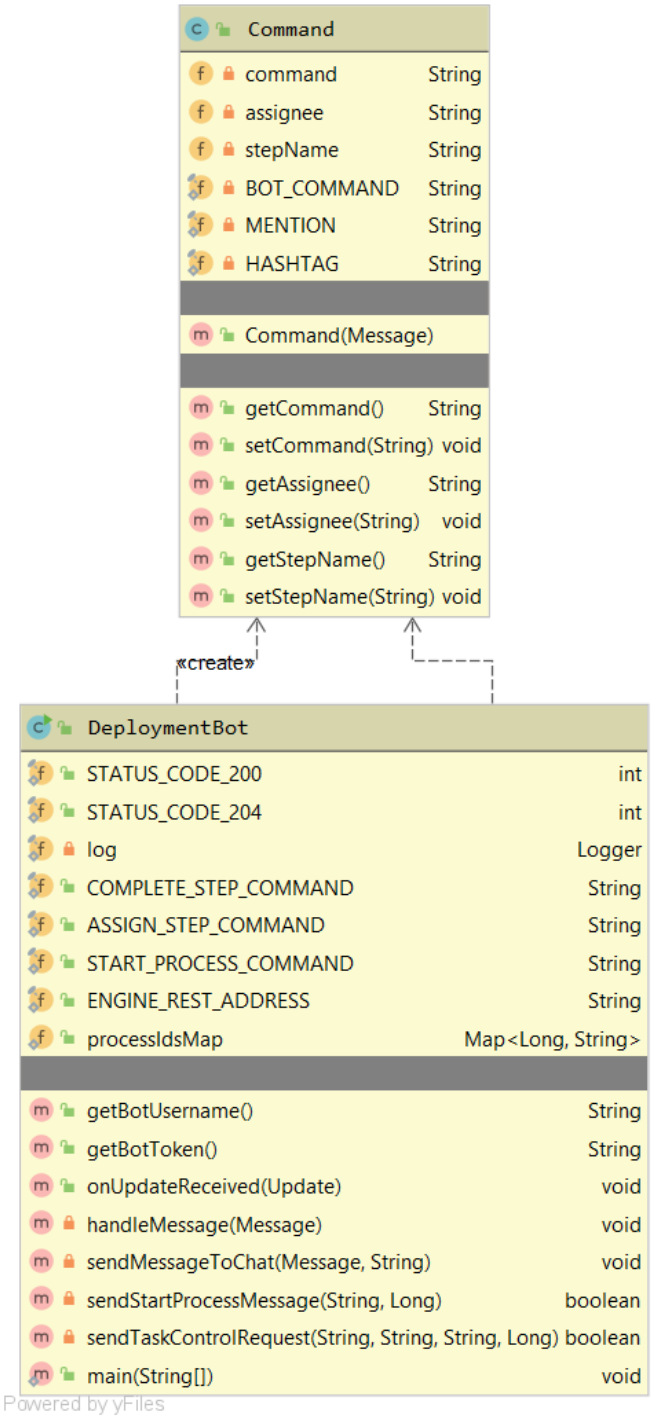

Рисунок 3.3 – Діаграма класів системи

На побудованій діграмі класів (рис. 3.3) зображено класи Command та DeploymentBot, їх атрибути (назва й тип) та методи зі списком параметрів та типом даних, що повертає метод.

3.2 Опис та формування вхідних даних

Опис даних системи

Перед початком роботи системи необхідно створити план процесу за допомогою додатку Camunda Modeler (рис 3.4).

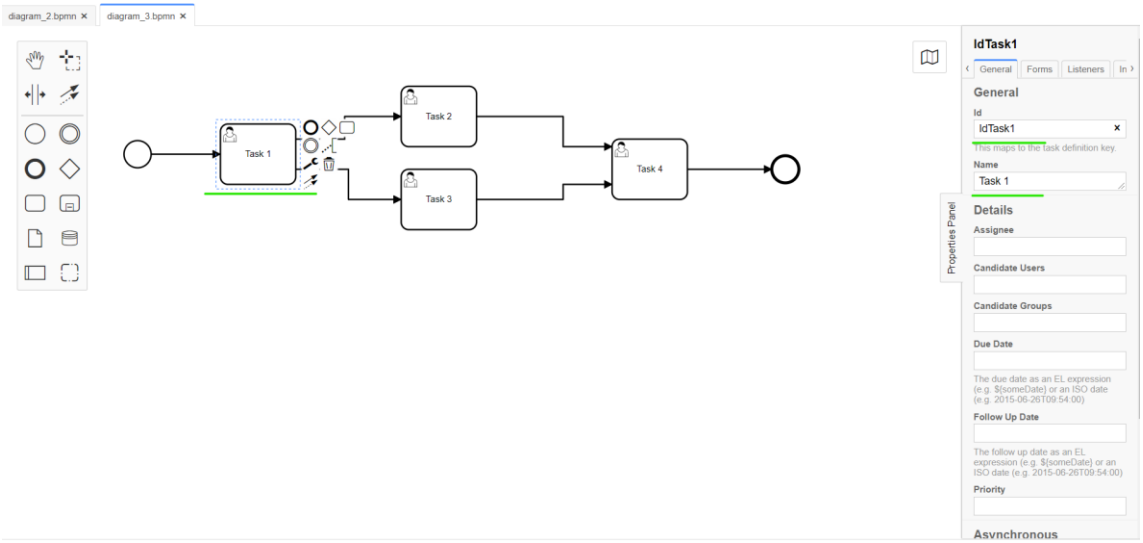

Рисунок 3.4 – Етап моделювання процесу

Тут за допомогою графічних інструментів потрібно описати діаграму процесу, де зазначити усі необхідні задачі із відповідними ідентифікаторами, назвами, залежностями та іншими необхідними атрибутами Після чого необхідно завантажити діаграму до системи Camunda, це можна зробити безпосередньо із Modeler'у.

Після чого керівнику процесу варто працювати із даними з діаграми (назви та ідентифікатори процесів та задач).

Опис даних користувача

Під час роботи із самим ботом користувач може вводити команди згідно із варіантами використання у наступному форматі:

- розпочати процес – /start_process #ProcessKey, де start_process – назва команди, ProcessKey – ідентифікатор плану процесу;

- видати задачу – /assign_step @userId #taskId, де assign_step – назва команди, userId – ідентифікатор користувача, якому видається задача, taskId – ідентифікатор задачі;
- завершити задачу – /complete_step @userId #taskId, де complete_step – назва команди, userId – ідентифікатор користувача, який завершує задачу, taskId – ідентифікатор задачі.

3.3 Програмна реалізація інформаційної системи

Архітектура додатку та організація коду

У даній роботі система буде складатися із трьох головних модулів, що схематично зображено на рисунку 3.5.

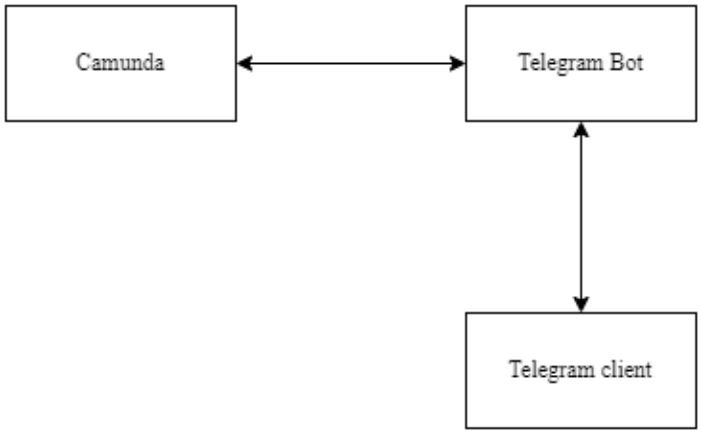

Рисунок 3.5 – Спрощена архітектура системи

Кожен із модулів реалізує певну частину функціоналу системи:

- Telegram client – це модуль для взаємодії із кінцевим користувачем, де він зможе виконувати команди керування процесом та отримувати інформацію про поточний стан.
- Camunda – це модуль, де зосереджена головна бізнес-логіка системи: процес та його задачі.
- Telegram-bot – це модуль для обробки запитів користувача та делегування команд до модуля Camunda.

Для логічної організації коду використаємо раніше побудовану діаграму класів та Java-підхід до збереження коду та класів в проекті за допомогою пакетів. Результат організації класів додатку показано на рисунку 3.6.

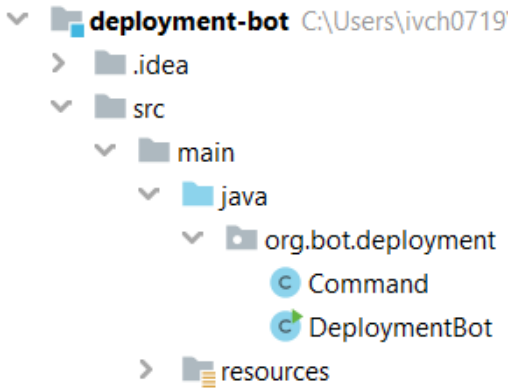

Рисунок 3.6 – Розміщення класів додатку в IDE Intellij IDEA

Як видно на рисунку 3.6, було створено пакет org.bot.deployment, у якому розміщенно два класи: Command та DeploymentBot.

Короткий опис програмної реалізації

Розглянемо загальний підхід до програмної реалізації функціоналу системи.

Розглянемо клас Command, що зберігає відомості про команду для бота від користувача. Даний клас містить поля (command, assignee, stepName) та методи доступу до цих полів, а також конструктор, у який ми передаємо об’єкт класу Message, а потім, у залежності від змісту повідомлення, ми заповнюємо поля класу. Розлянемо приклад заповнення поля command. Приклад коду для заповнення поля command наведено на рисунку 3.7.

```
public Command(Message message) {
    Optional<MessageEntity> commandEntity =
            message.getEntities().stream().filter(e -> BOT_COMMAND.equals(e.getType())).findFirst();
    if (commandEntity.isPresent()) {
        this.command = commandEntity.get().getText();
        Optional<MessageEntity> mentionEntity =
                message.getEntities().stream().filter(e -> MENTION.equals(e.getType())).findFirst();
```

Рисунок 3.7 – Приклад коду для заповнення поля command

Спочатку ми отримуємо з повідомлення спеціальні сутності MessageEntity, а потім фільтруємо їх за типом, значення потім заносимо до відповідної змінної класу. Решта полів заповнюються аналогічно.

У класі DeploymentBot ми вже оброблюємо команди користувача в методі handleMessage. Тут ми спочатку отримуємо об'єкт класу Command та в залежності від значень його полів виконуємо різну логіку програми.

Розглянемо, наприклад, обробку команди /start_process. Приклад методу для надсилання повідомлень до REST API наведено на рисунку 3.8.

```
private boolean sendStartProcessMessage(String processKey, Long chatId) {
    HttpPost postRequest = new HttpPost( uri: ENGINE_REST_ADDRESS
            + "/process-definition/key/" + processKey + "/start");
    postRequest.addHeader( name: "content-type", value: "application/json");
    String jsonRequestBody = "{}";
    int statusCode = 0;
    log.info("sendStartProcessMessage - send" + ENGINE_REST_ADDRESS + "/process-definition/key/" + processKey + "/start");
    try (CloseableHttpClient httpClient = HttpClients.createDefault();
         CloseableHttpResponse response = httpClient.execute(postRequest)) {
        postRequest.setEntity(new StringEntity(jsonRequestBody));
        statusCode = response.getStatusLine().getStatusCode();
        log.info("sendStartProcessMessage - statusCode = " + statusCode);
        if (statusCode == STATUS_CODE_200) {
            JSONObject jsonResponse = new JSONObject(EntityUtils.toString(response.getEntity()));
            processIdsMap.put(chatId, jsonResponse.get("id").toString());
            log.info("sendStartProcessMessage - PROCESS_ID = " + jsonResponse.get("id").toString() + " is started");
        }
    } catch (IOException e) {
        log.error("Error sendStartProcessMessage " + e.getMessage());
    }
    return statusCode == STATUS_CODE_200;
}
```

Рисунок 3.8 – Приклад коду для надсилання повідомлень до REST API

Якщо повідомлення містить цю команду, то ми виконуємо метод sendStartProcessMessage, формуємо HTTP-запит до Camunda Rest API. Якщо запит успішний (ми отримали у відповідь HTTP статус-код 200), то потім відправляємо у методі sendMessageToChat повідомлення користувачеві, що процес розпочато успішно. У іншому випадку повідомляємо, що система не змогла розпочати процес. Методи для обробки інших команд реалізовано у такий самий спосіб.

Повний програмний код наведено у Додатку А.

3.4 Аналіз роботи програми

Розглянемо результати виконаної роботи. Проглянемо реалізацію кожного з варіантів використання:

- створити план процесу (рис. 3.9);

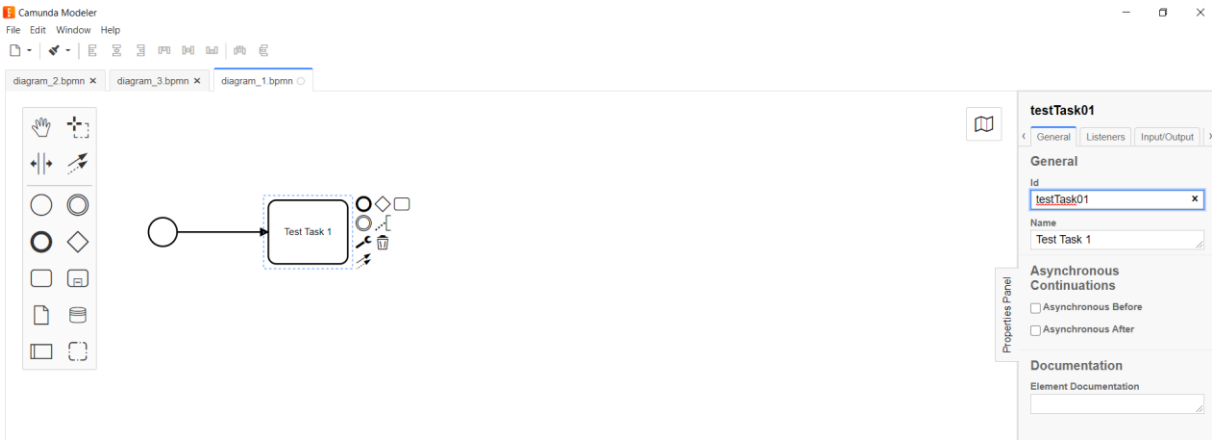

Рисунок 3.9 – Створення плану процесу в додатку Camunda Modeler

При створенні плану процесу користувач в додатку Camunda Modeler повинен зазначити усі задачі, їх ідентифікатори та назви. Після цього необхідно також зазначити усі зв’язки між задачами.

- завантажити план процесу (рис. 3.10, рис. 3.11);

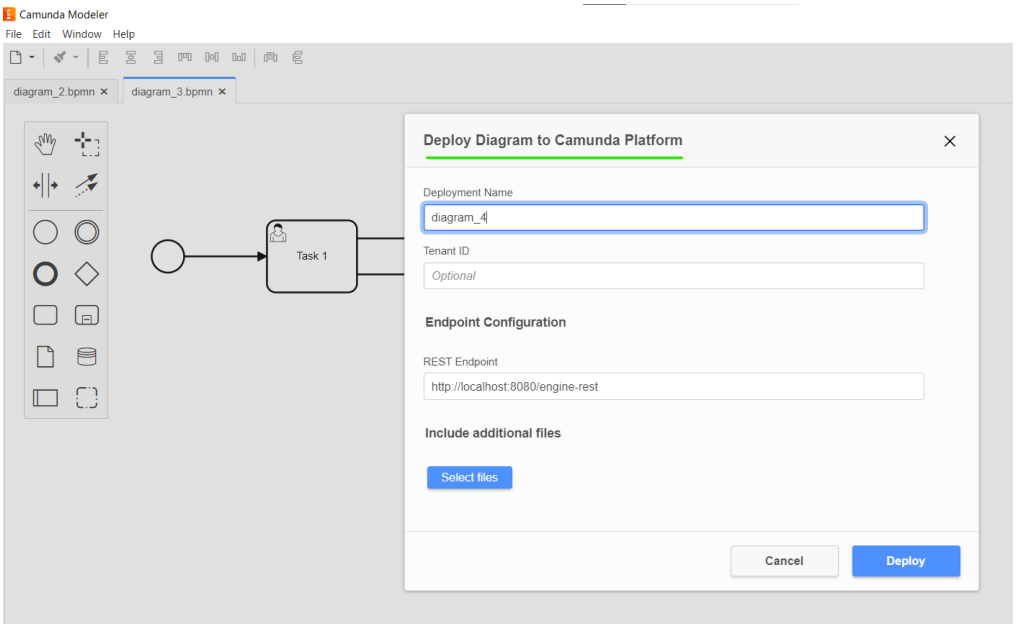

Рисунок 3.10 – Форма завантаження плану процесу

На рисунку 3.10 зображено форму для завантаження плану процесу. Для цього користувачеві необхідно натиснути кнопку «Deploy current diagram», заповнити поля «Deployment Name» та «REST Endpoint» та натиснути кнопку «Deploy».

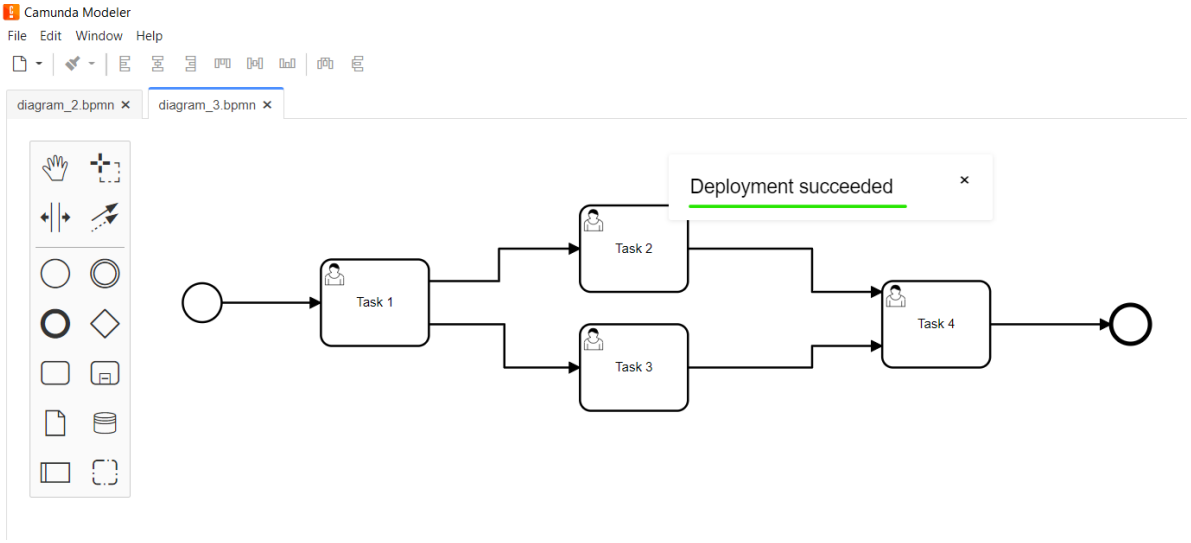

Рисунок 3.11 – Повідомлення про успішне завантаження плану процесу

Після успішного завантаження плану до системи користувач побачить повідомлення «Deployment succeeded».

- розпочати процес – (рис. 3.12, рис. 3.13);

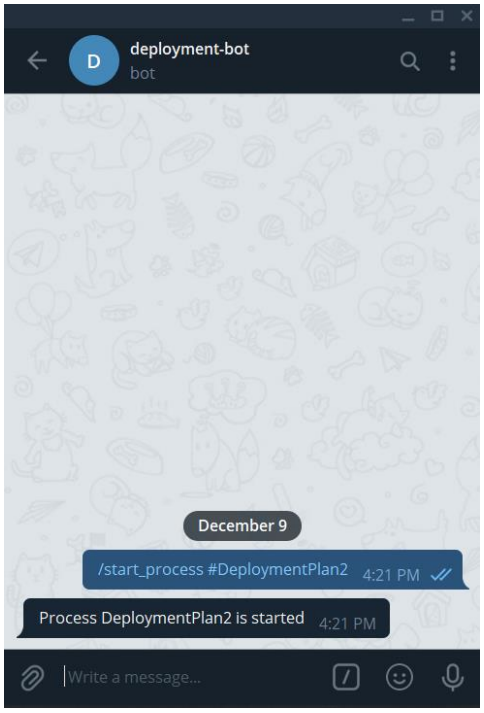

Рисунок 3.12– Команда для початку процесу та повідомлення-результат в чаті

Для початку процесу користувач повинен надіслати в чаті, де додано бота, повідомлення у форматі /start_process #ProcessKey. Після чого він повинен побачити повідомлення про те, що процес успішно розпочато.

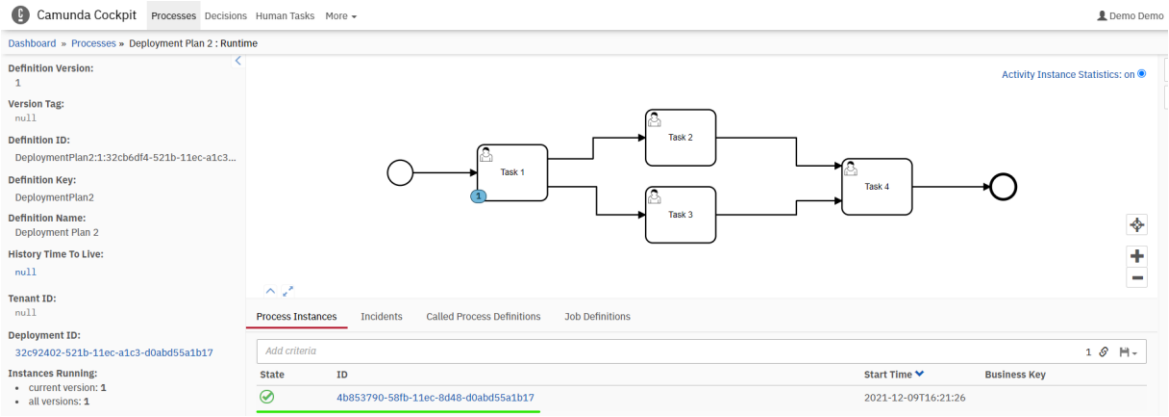

Рисунок 3.13 – Розпочатий процес та його ідентифікатор в системі Camunda

Користувач системи також має змогу переглядати поточний стан процесу за допомогою користувацького інтерфейсу поатформи Camunda. Наприклад, на рисунку 3.13 можна побачити запущенний процес з його

ідентифікатором та часом запуску. Також можна відслідковувати які задачі зараз виконуються (наприклад, задача «Task 1»).

- видати задачу – (рис. 3.14, рис. 3.15);

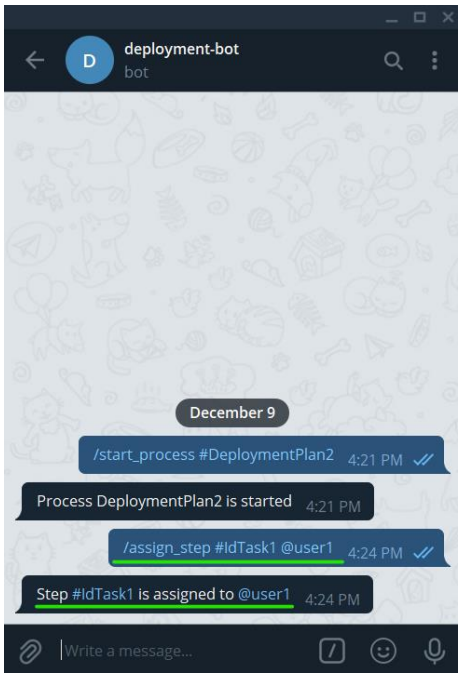

Рисунок 3.14 – Команда для видачі задачі та повідомлення-результат в чаті

Для того, щоб видати задачу користувачеві, потрібно надіслати до чату повідомлення у форматі /assign_step @userId #taskId. Після чого буде система надішле зворотнє повідомлення про те, що задачу дійсно видано зазначеному раніше користувачеві.

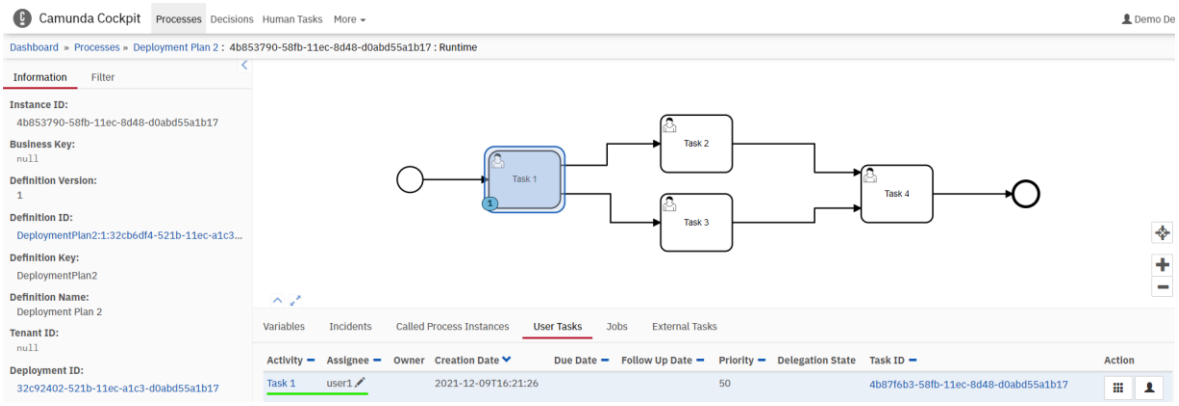

Рисунок 3.15 – Задача та її виконавець в системі Camunda

У користувацькому інтерфейсі також є можливість переглянути результат видачі задачі. Наприклад, на рисунку 3.15 видно, що задача «Task 1» видана користувачеві «user1» (поле «Assignee»).

- завершити задачу – (рис. 3.16, рис. 3.17).

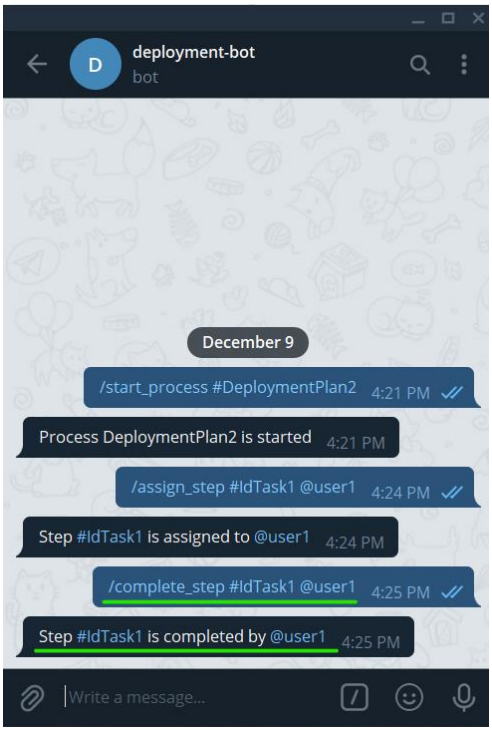

Рисунок 3.16 – Команда для завершення задачі та повідомлення-результат в чаті

Щоб завершити виконання задачі користувач повинен надіслати в чат повідомлення у форматі /complete_step @userId #taskId. Потім система надішле повідомлення у відповідь про те, що задача дійсно завершена.

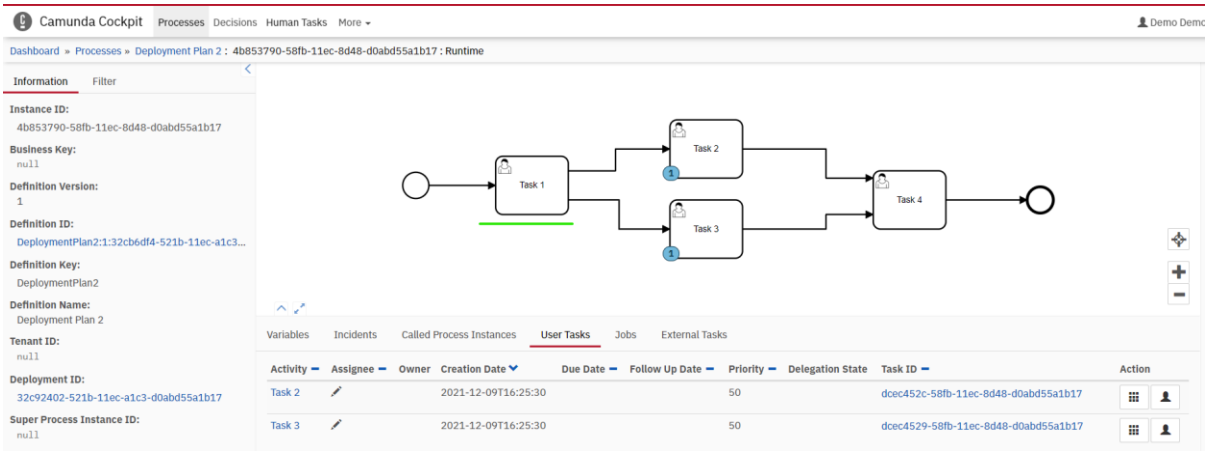

Рисунок 3.17 – Процес та його задачі після завершення однієї із задач в системі Camunda

У користувача також є можливість переглянути результат завершення задачі в користувацькому інтерфейсі. На рисунку 3.17, наприклад, видно, що після надсилання команди для завершення задачі «Task 1» тепер активними стали задачі «Task 2» та «Task 3».

ВИСНОВКИ

Під час роботи було розглянуто проблему керування людськими ресурсами під час процесу деплойменту та один із методів організації та уніфікації процесів – воркфлоу.

У ході інформаційного огляду було проаналізовано декілька популярних програмних рішень для побудови та оркестрації процесів: Camunda та Apache Airflow. У результаті чого було вирішено використати платформу Camunda у даній роботі.

Під час реалізації системи спочатку було проведено інформаційне проектування, у результаті якого було спроектовано:

- ER-діаграму;
- діаграму варіантів використання;
- діаграму класів.

Використовуючи побудовані діаграми, було розроблено Telegram-бот на мові програмування Java.

Під час роботи було поглиблено знання мови Java та методів роботи із HTTP REST запитами. Було детальніше розглянуто графічну нотацію BPMN для побудови бізнес-процесів у програмному рішенні Camunda.

Розроблена система має також перспективи для майбутнього розвитку та розробки нового функціоналу:

- автоматичне формування звіту процесу;
- відслідковування поточного статусу процесу;
- автоматичне надсилання нагадувань, якщо кроки довго не завершуються.

У результаті роботи було отримано систему з можливостями створення та завантаження процесу, чат-бот, що допомагає вести процес деплойменту, надаючи можливості надсилання команд початку процесу, видачі та завершення задач, та є зрозумілим у використанні.

СПИСОК ЛІТЕРАТУРИ

1. Workflow Orchestration: Introduction, Types, Tools and Use Cases [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://www.xenonstack.com/insights/workflow-orchestration (дата звернення 10.11.2021) – Назва з екрана.
2. awesome-workflow-engines - A curated list of awesome open source workflow engines [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://github.com/meirwah/awesome-workflow-engines (дата звернення 10.11.2021) – Назва з екрана.
3. Workflow and Decision Automation Platform | Camunda [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://camunda.com/ (дата звернення 10.11.2021) – Назва з екрана.
4. BPMN Specification - Business Process Model and Notation [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://www.bpmn.org/ (дата звернення 10.11.2021) – Назва з екрана.
5. Apache Airflow [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://airflow.apache.org/ (дата звернення 10.11.2021) – Назва з екрана.
6. Java Platform Standard Edition 8 Documentation [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://docs.oracle.com/javase/8/docs/ (дата звернення 12.11.2021) – Назва з екрана.
7. 15+ Top Reasons to Choose Java for Backend Development [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://www.yourteaminindia.com/blog/java-backend-development/ (дата звернення 12.11.2020) – Назва з екрана.

8. H2 Database Engine [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://www.h2database.com/html/main.html (дата звернення 12.11.2021) – Назва з екрана.
9. Telegram Messenger [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://telegram.org/ (дата звернення 12.11.2021) – Назва з екрана.
10. Top 10+ Best Java IDEs & Online Java Compilers [2020 Rankings] Bootstrap · The most popular HTML, CSS, and JS library in the world. [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://www.softwaretestinghelp.com/best-java-ide-and-online-compilers/ (дата звернення 12.11.2021) – Назва з екрана.
11. Why software design is important? [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://hackernoon.com/why-software-design-is-important-9ecbea883bbb (дата звернення 15.11.2021) – Назва з екрана.
12. Advantages And Disadvantages Of ER Model In DBMS [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://pctechnicalpro.blogspot.com/2017/04/advantages-disadvantages-er-model-dbms.html (дата звернення 15.11.2021) – Назва з екрана.
13. Key benefits of using entity relationship diagrams [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://pcdreams.com.sg/key-benefits-of-using-entity-relationship-diagrams/(дата звернення 15.11.2020) – Назва з екрана.
14. Introduction to unified modeling language [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://www.uml.org/what-is-uml.htm (дата звернення 15.11.2021) – Назва з екрана.

15. Why the Software Industry Has a Love-Hate Relationship with UML Diagrams [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://creately.com/blog/diagrams/advantages-and-disadvantages-of-uml/ (дата звернення 15.11.2021) – Назва з екрана.
16. REST API | camunda BPM docs [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://docs.camunda.org/manual/7.3/api-references/rest/ (дата звернення 29.11.2021) – Назва з екрана.
17. Telegram APIs [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://core.telegram.org/api (дата звернення 25.11.2021) – Назва з екрана.
18. Process Definition [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://docs.camunda.org/manual/7.3/api-references/rest/#process-definition (дата звернення 25.11.2021) – Назва з екрана.
19. Process Instance [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://docs.camunda.org/manual/7.3/api-references/rest/#process-instance (дата звернення 25.11.2021) – Назва з екрана.
20. User [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://core.telegram.org/bots/api#user (дата звернення 25.11.2021) – Назва з екрана.
21. User [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://docs.camunda.org/manual/7.3/api-references/rest/#user (дата звернення 25.11.2021) – Назва з екрана.

22. Task [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://docs.camunda.org/manual/7.3/api-references/rest/#task (дата звернення 25.11.2021) – Назва з екрана.

23. Message [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://core.telegram.org/bots/api#message (дата звернення 25.11.2021) – Назва з екрана.

24. MessageEntity [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://core.telegram.org/bots/api#messageentity (дата звернення 25.11.2021) – Назва з екрана.

ДОДАТОК А. ЛІСТИНГ ПРОГРАМНОГО КОДУ

Файл pom.xml

```
<?xml version="1.0" encoding="UTF-
8"?>
<project
xmlns="http://maven.apache.org/POM/
4.0.0"

xmlns:xsi="http://www.w3.org/2001/X
MLSchema-instance"

xsi:schemaLocation="http://maven.ap
ache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-
4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>org.bot.deployment</groupI
d>
    <artifactId>deployment-
bot</artifactId>
    <version>1.0-SNAPSHOT</version>
    <properties>

<maven.compiler.source>1.8</maven.c
ompiler.source>

<maven.compiler.target>1.8</maven.c
ompiler.target>

<telegram.version>5.4.0</telegram.v
ersion>
    </properties>
    <dependencies>
        <!-- Telegram API -->
        <dependency>

<groupId>org.telegram</groupId>

<artifactId>telegrambots</artifactI
d>

<version>${telegram.version}</versi
on>
        </dependency>
        <dependency>

<groupId>org.telegram</groupId>

<artifactId>telegrambotsextensions<
/artifactId>

<version>${telegram.version}</versi
on>
        </dependency>
        <!-- Apache HttpComponents
-->
        <dependency>

<groupId>org.apache.httpcomponents<
/groupId>

<artifactId>httpclient</artifactId>

<version>4.5.10</version>
        </dependency>
        <!-- JSON-java -->
        <dependency>

<groupId>org.json</groupId>

<artifactId>json</artifactId>

<version>20180130</version>
        </dependency>
        <!-- SLF4J -->
        <dependency>

<groupId>org.slf4j</groupId>
            <artifactId>slf4j-
log4j12</artifactId>

<version>1.7.21</version>
        </dependency>
    </dependencies>
</project>
```

Клас DeploymentBot

```
package org.bot.deployment;

import
org.apache.http.HttpResponse;
import
org.apache.http.client.methods.Clos
eableHttpResponse;
import
org.apache.http.client.methods.Http
Get;
import
org.apache.http.client.methods.Http
Post;
import
```

```
org.apache.http.entity.StringEntity
;
import
org.apache.http.impl.client.Closeab
leHttpClient;
import
org.apache.http.impl.client.HttpCli
ents;
import
org.apache.http.util.EntityUtils;
import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import
org.telegram.telegrambots.bots.Tele
gramLongPollingBot;
import
org.telegram.telegrambots.meta.Tele
gramBotsApi;
import
org.telegram.telegrambots.meta.api.
methods.send.SendMessage;
import
org.telegram.telegrambots.meta.api.
objects.Message;
import
org.telegram.telegrambots.meta.api.
objects.Update;
import
org.telegram.telegrambots.meta.exce
ptions.TelegramApiException;
import
org.telegram.telegrambots.updatesre
ceivers.DefaultBotSession;

import java.io.IOException;
import java.util.HashMap;
import java.util.Map;

public class DeploymentBot extends
TelegramLongPollingBot {

    public static final int
STATUS_CODE_200 = 200;
    public static final int
STATUS_CODE_204 = 204;
    private static final Logger log
=
LoggerFactory.getLogger(DeploymentB
ot.class);

    public static final String
COMPLETE_STEP_COMMAND =
"/complete_step";
    public static final String
ASSIGN_STEP_COMMAND =
"/assign_step";
    public static final String
START_PROCESS_COMMAND =
"/start_process";

    public static final String
ENGINE_REST_ADDRESS =
"http://localhost:8080/engine-
rest";

    public static Map<Long, String>
processIdsMap = new HashMap<>();

    @Override
    public String getBotUsername()
{
        return
"@telegdeployment_bot";
    }

    @Override
    public String getBotToken() {
        return
"2129937346:AAHaOT6urHIPaKOkeB9Lflm
ipUjNVqJGqG4";
    }

    @Override
    public void
onUpdateReceived(Update update) {
        if (update.hasMessage()) {
            try {

handleMessage(update.getMessage());
            } catch
(TelegramApiException e) {
                log.error("Error
while handle message " +
e.getMessage());
            }
        }
    }

    private void
handleMessage(Message message)
throws TelegramApiException {
        if (message.hasText() &&
message.hasEntities()) {
            Command messageCommand
= new Command(message);
            if
(START_PROCESS_COMMAND.equals(messa
geCommand.getCommand())) {
                String
startProcessKey =
messageCommand.getStepName().replac
```

```
e("#", "");
                String
messageToChat = "";
                if
(sendStartProcessMessage(startProce
ssKey, message.getChatId())) {
                    messageToChat =
"Process " + startProcessKey + " is
started";
                } else {
                    messageToChat =
"Process " + startProcessKey + "
cannot be started";
                }

sendMessageToChat(message,
messageToChat);
            }

            if
(ASSIGN_STEP_COMMAND.equals(message
Command.getCommand())) {
                String
messageToChat = "";
                if
(sendTaskControlRequest(messageComm
and.getStepName().replace("#", ""),

messageCommand.getAssignee().replac
e("@", ""),

"/assignee",

message.getChatId())) {
                    messageToChat =
"Step " +
messageCommand.getStepName() + " is
assigned to " +
messageCommand.getAssignee();
                } else {
                    messageToChat =
"Step " +
messageCommand.getStepName() + "
cannot be assigned to " +
messageCommand.getAssignee();
                }

sendMessageToChat(message,
messageToChat);
            }

            if
(COMPLETE_STEP_COMMAND.equals(messa
geCommand.getCommand())) {
                String
messageToChat = "";
                if
(sendTaskControlRequest(messageComm
and.getStepName().replace("#", ""),

messageCommand.getAssignee().replac
e("@", ""),

"/complete",

message.getChatId())) {
                    messageToChat =
"Step " +
messageCommand.getStepName() + " is
completed by " +
messageCommand.getAssignee();
                } else {
                    messageToChat =
"Step " +
messageCommand.getStepName() + "
cannot be completed by " +
messageCommand.getAssignee();
                }

sendMessageToChat(message,
messageToChat);
            }
        }
    }

    private void
sendMessageToChat(Message message,
String text) throws
TelegramApiException {

execute(SendMessage.builder()

.chatId(message.getChatId().toStrin
g())
                .text(text)
                .build());
    }

    private boolean
sendStartProcessMessage(String
processKey, Long chatId) {
        HttpPost postRequest = new
HttpPost(ENGINE_REST_ADDRESS
                + "/process-
definition/key/" + processKey +
"/start");

postRequest.addHeader("content-
type", "application/json");
        String jsonRequestBody =
"{}";
        int statusCode = 0;

log.info("sendStartProcessMessage -
```

```
send" + ENGINE_REST_ADDRESS +
"/process-definition/key/" +
processKey + "/start");
        try (CloseableHttpClient
httpClient =
HttpClients.createDefault();
             CloseableHttpResponse
response =
httpClient.execute(postRequest)) {

postRequest.setEntity(new
StringEntity(jsonRequestBody));
            statusCode =
response.getStatusLine().getStatusC
ode();

log.info("sendStartProcessMessage -
statusCode = " + statusCode);
            if (statusCode ==
STATUS_CODE_200) {
                JSONObject
jsonResponse = new
JSONObject(EntityUtils.toString(res
ponse.getEntity()));

processIdsMap.put(chatId,
jsonResponse.get("id").toString());

log.info("sendStartProcessMessage -
PROCESS_ID = " +
jsonResponse.get("id").toString() +
" is started");
            }
        } catch (IOException e) {
            log.error("Error
sendStartProcessMessage " +
e.getMessage());
        }
        return statusCode ==
STATUS_CODE_200;
    }

    private boolean
sendTaskControlRequest(String
definitionKey, String userId,
String apiCommand, Long chatId) {
        HttpGet getRequest = new
HttpGet(ENGINE_REST_ADDRESS +
"/task/?taskDefinitionKey="
                + definitionKey +
"&processInstanceId=" +
processIdsMap.get(chatId));
        boolean isRequestSuccess =
false;
        try (CloseableHttpClient
httpClient =
HttpClients.createDefault()) {

log.info("sendTaskControlRequest -
send /task/?taskDefinitionKey=" +
definitionKey +
"&processInstanceId=" +
processIdsMap.get(chatId));
            HttpResponse
getResponse =
httpClient.execute(getRequest);
            int getStatusCode =
getResponse.getStatusLine().getStat
usCode();
            isRequestSuccess =
getStatusCode == STATUS_CODE_200;
            if (isRequestSuccess) {
                JSONArray
jsonResponse = new
JSONArray(EntityUtils.toString(getR
esponse.getEntity()));
                boolean
isEmptyResponse =
jsonResponse.length() == 0;
                isRequestSuccess =
!isEmptyResponse;
                if
(isRequestSuccess) {
                    String taskId =
jsonResponse.getJSONObject(0).get("
id").toString();

log.info("sendTaskControlRequest -
taskId=" + taskId);

log.info("sendTaskControlRequest -
send /task/" + taskId +
"/assignee");
                    HttpPost
postRequest = new
HttpPost(ENGINE_REST_ADDRESS +
"/task/" + taskId + apiCommand);

postRequest.addHeader("content-
type", "application/json");
                    String
jsonRequestBody = "{\"userId\": \""
+ userId + "\"}";

postRequest.setEntity(new
StringEntity(jsonRequestBody));
                    HttpResponse
postResponse =
httpClient.execute(postRequest);

isRequestSuccess =
postResponse.getStatusLine().getSta
tusCode() == STATUS_CODE_204;
                }
```

```
            }
        } catch (JSONException e) {
            log.error("Error
sendTaskControlRequest while
parsing json response" +
e.getMessage());
        } catch (IOException e) {
            log.error("Error
sendTaskControlRequest " +
e.getMessage());
        }
        return isRequestSuccess;
    }

    public static void
main(String[] args) {
        DeploymentBot bot = new
DeploymentBot();
        TelegramBotsApi
telegramBotsApi = null;
        try {
            telegramBotsApi = new
TelegramBotsApi(DefaultBotSession.c
lass);

telegramBotsApi.registerBot(bot);
        } catch
(TelegramApiException e) {
            log.error("Error while
starting the bot " +
e.getMessage());
        }
    }
}
```

Клас Command

```
package org.bot.deployment;

import
org.telegram.telegrambots.meta.api.
objects.Message;
import
org.telegram.telegrambots.meta.api.
objects.MessageEntity;

import java.util.Optional;

public class Command {

    private String command;

    private String assignee;

    private String stepName;

    private static final String
BOT_COMMAND = "bot_command";
    private static final String
MENTION = "mention";
    private static final String
HASHTAG = "hashtag";

    public Command(Message message)
{
        Optional<MessageEntity>
commandEntity =

message.getEntities().stream().filt
er(e ->
BOT_COMMAND.equals(e.getType())).fi
ndFirst();
        if
(commandEntity.isPresent()) {
            this.command =
commandEntity.get().getText();
            Optional<MessageEntity>
mentionEntity =

message.getEntities().stream().filt
er(e ->
MENTION.equals(e.getType())).findFi
rst();
            if
(mentionEntity.isPresent()) {
                this.assignee =
mentionEntity.get().getText();

            }
            Optional<MessageEntity>
hashtagEntity =

message.getEntities().stream().filt
er(e ->
HASHTAG.equals(e.getType())).findFi
rst();
            if
(hashtagEntity.isPresent()) {
                this.stepName =
hashtagEntity.get().getText();
            }
        }
    }

    public String getCommand() {
        return command;
    }

    public void setCommand(String
command) {
        this.command = command;
```

```
    }

    public String getAssignee() {
        return assignee;
    }

    public void setAssignee(String
assignee) {
        this.assignee = assignee;
    }

    public String getStepName() {
        return stepName;
    }

    public void setStepName(String
stepName) {
        this.stepName = stepName;
    }
}
```