Міністерство освіти і науки України
Національний університет «Одеська політехніка»
Інститут комп’ютерних систем
Кафедра інформаційних систем

Сауляк Владислав Васильович,
студент групи AI-215

ДИСЦИПЛІНА
Об’єктно-орієнтоване програмування

КУРСОВА РОБОТА
Розробка телеграмм бота для ведення заміток

Спеціальність:
122 Комп’ютерні науки

Освітня програма:
Комп’ютерні науки

Керівник:
Годовиченко Микола Анатолійович,
кандидат технічних наук, доцент

Одеса – 2023

# ЗМІСТ

# АНОТАЦІЯ

Моя курсова робота спрямована на створення Java-телеграм-бота з використанням фреймворку Spring Boot та паттерну розробки State. Даний бот розроблений для зручного створення та керування користувацькими замітками на мобільних пристроях. Основною метою роботи є створення простого, ефективного та інтуїтивно зрозумілого інструменту, який дозволить користувачам легко створювати, редагувати, видаляти та організовувати свої замітки. В ході розробки додатку будуть використані сучасні технології та практики розробки.

# ABSTRACT

My course work is aimed at creating a Java Telegram bot using the Spring Boot framework and the State development pattern. This bot is designed for easy creation and management of user notes on mobile devices. The main goal of the work is to create a simple, efficient and intuitive tool that will allow users to easily create, edit, delete and organize their notes. The application will be developed using modern technologies and development practices.

# ВСТУП

У сучасному світі, де месенджери стали невід'ємною частиною нашого повсякденного життя, зростає потреба в ефективних та зручних інструментах для організації наших справ, ідей та нагадувань. Одним з найпоширеніших і водночас простих засобів є телеграм боти. Вони дозволяють нам швидко отримувати інформацію, виконувати різноманітні завдання та спростити наше щоденне життя.

З урахуванням цієї потреби, було вирішено зосередитися на розробці телеграм бота для організації та управління задачами, нагадуваннями та замітками. Такий бот дозволить користувачам зручно створювати, переглядати та управляти своїми задачами прямо з месенджера Telegram. Мета роботи полягає у створенні інтуїтивно зрозумілого інструменту, який буде доступний на платформі Telegram і забезпечуватиме зручний і швидкий доступ до організації завдань незалежно від місця та часу.

Розробка такого телеграм бота вимагає використання сучасних технологій, програмування та архітектурних принципів, що дозволяють створити стійкий, ефективний і надійний продукт. Бот повинен мати інтуїтивний інтерфейс, що дозволить користувачам швидко засвоїти його функціонал і легко використовувати його у повсякденному житті.

Враховуючи викладені обставини, актуальною є тема розробки телеграм бота для організації задач. Такий бот дозволяв би створювати, переглядати та управляти задачами з можливістю збереження їх історії та стану, щоб мати можливість переглядати їх при наступних запитах.

Метою курсової роботи є розробка телеграм бота для управління задачами.

Для досягнення цієї мети необхідно виконати наступні задачі:

– здійснити огляд інших ботів для створення користувацьких заміток;

– проаналізувати та обрати сучасні технології для телеграм боту;

– провести проєктування телеграм боту заміток;

– виконати програмну реалізацію спроєктованого телеграм бота.

# 1 ОГЛЯД СИСТЕМ-АНАЛОГІВ ТА ТЕХНОЛОГІЙ ЇХ РОЗРОБКИ

## 1.1 Особливості використання технологій для ведення справ та особистої продуктивності

Технологіїв відіграють значну роль у поліпшенні особистої продуктивності та веденні справ. Вони надають користувачам широкі можливості доступу до інформації, організації задач і спрощення повсякденних обов'язків. Ось декілька особливостей використання технологій у цьому контексті:

– доступність через мобільні пристрої: Телеграм боти можна використовувати на мобільних пристроях, що дає можливість мати постійний доступ до бота незалежно від місця перебування. Можна легко комунікувати з ботом, отримувати інформацію та керувати задачами прямо зі свого смартфона або планшета;

– інтерактивність та зручність взаємодії: Телеграм боти забезпечують інтерактивну взаємодію з користувачем. Вони можуть пропонувати меню, кнопки, шаблонні повідомлення тощо, що спрощує спілкування та надає зручність в користуванні ботом. Користувач може вибирати опції, вводити дані або отримувати інформацію шляхом простих дій;

– автоматизація завдань та оповіщень: Телеграм боти можуть виконувати різноманітні завдання автоматично або за запитом користувача. Вони можуть надсилати нагадування, сповіщення або повідомлення згідно з попередньо налаштованим розкладом або у відповідь на певні події. Це дозволяє ефективно керувати часом, планувати задачі та отримувати важливу інформацію вчасно;

– підтримка різних функціональностей: Телеграм боти можуть виконувати різні завдання залежно від їх налаштувань. Вони можуть надавати інформацію, виконувати пошук, відправляти сповіщення, проводити операції з даними, надавати підтримку користувачам та багато іншого. Можливості бота обмежуються його функціональністю, але можуть бути розширені шляхом додавання нових модулів та інтеграцій;

– користувачка адаптація: Телеграм боти зазвичай працюють у зручному та зрозумілому для користувача форматі. Вони можуть мати простий та інтуїтивно зрозумілий інтерфейс, який дозволяє легко орієнтуватися та взаємодіяти з ботом. Користувачам не потрібно мати спеціальні технічні навички для користування ботом, що робить його доступним для широкої аудиторії;

Загалом, телеграм боти є зручним інструментом для виконання завдань, комунікації та отримання інформації через мобільні пристрої. Вони надають широкий спектр можливостей та спрощують процес взаємодії з користувачем.

## 1.2 Огляд додатків для ведення заміток

Для розробки телеграм бота необхідно з'ясувати, які функції повинні бути доступні в ньому. Для цього потрібно дослідити та проаналізувати інші додатки, які також дозволяють вести замітки. Це дозволить встановити особливості роботи цих додатків та з'ясувати позитивні та негативні моменти їх використання. Також цей аналіз допоможе краще зрозуміти потреби користувачів в цій області. Після проведення пошуку в Інтернеті, були знайдені найбільш популярні та відомі додатки, які будуть використовуватися як аналоги в даній роботі:

– додаток Remember The Milk [3];

– додаток ColorNote [4];

Далі потрібно провести аналіз цих додатків з метою встановлення особливостей їх роботи та з'ясування позитивних та негативних моментів їх використання. Це допоможе краще зрозуміти, які функції повинні бути доступні мобільному додатку для створення заміток, та відповісти на потреби користувачів в цій області.

### 1.2.1 Додаток Remember The Milk

Додаток Remember The Milk є зручним додатком для ведення справ для зайнятих людей. Ви ніколи не забудете придбати молоко (або що-небудь ще) ще раз (рис. 1.2) [3].

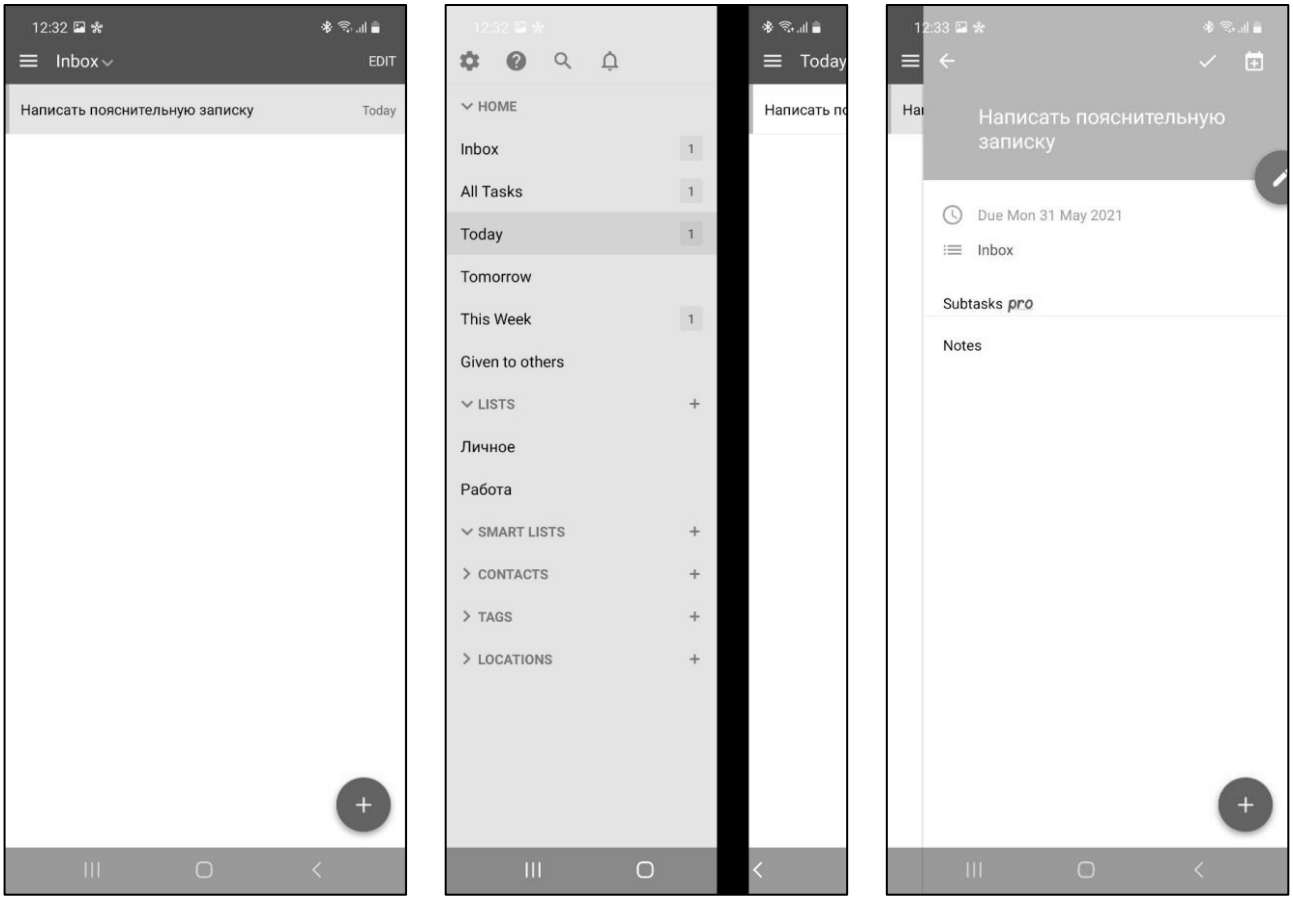



Рисунок 1.2 – Знімки екрану додатку Remember The Milk

Основний функціонал додатку:

– можливість створювати списки справ;

– можливість отримати нагадування через електрону пошту, мобільний пристрій чи популярні месенджери;

– можливість ділитися своїми списками з іншими людьми;

– синхронізація справ на всіх пристроях;

– можливість додавати пріоритети, терміни закінчення, кількість повторів справи, мітки тощо;

– пошук справ та нотаток за текстом, збереження справ та нотаток в розумних списках;

– можливість бачити справи, які фізично розташовані найближче та у найбільш ефективний спосіб.

Платна версія додатку має додатковий функціонал:

– підзадачі – можна розбивати задачі на менші частини;

– можливість ділитися задачами без обмежень;

– додатковий параметр кольору для задач;

– нагадування про необхідність виконання задачі через мобільний пристрій;

– синхронізація з Microsoft Outlook;

– безлімітне сховище для даних.

### 1.2.2 Додаток ColorNote

ColorNote – це простий блокнот. Він надасть можливість легкого і простого користування блокнотом при написанні заміток, нагадувань, email, повідомлень, переліків справ і покупок (рис 1.3) [4]. З ColorNote створення заміток простіше, ніж з будь-якими іншими блокнотами і органайзерами. Розглянемо можливості додатку:

– організація заміток за кольором;

– віджет стікерів;

– списки справ і покупок;

– організація розкладу в календарі;

– блокування замітки паролем;

– захищене резервне копіювання заміток на SD карту – перед вивантаженням замітки будуть зашифровані за стандартом AES, який використовується банками для захисту даних клієнтів;

– підтримка онлайн-синхронізації і резервного копіювання;

– синхронізація заміток між телефоном і планшетом;

– нагадування в статус-барі;

– нагадування про нотатки;

– можливість ділитися нотатками за допомогою SMS, електроної пошти чи твітеру;

– можливість кольорової організації нотаток.

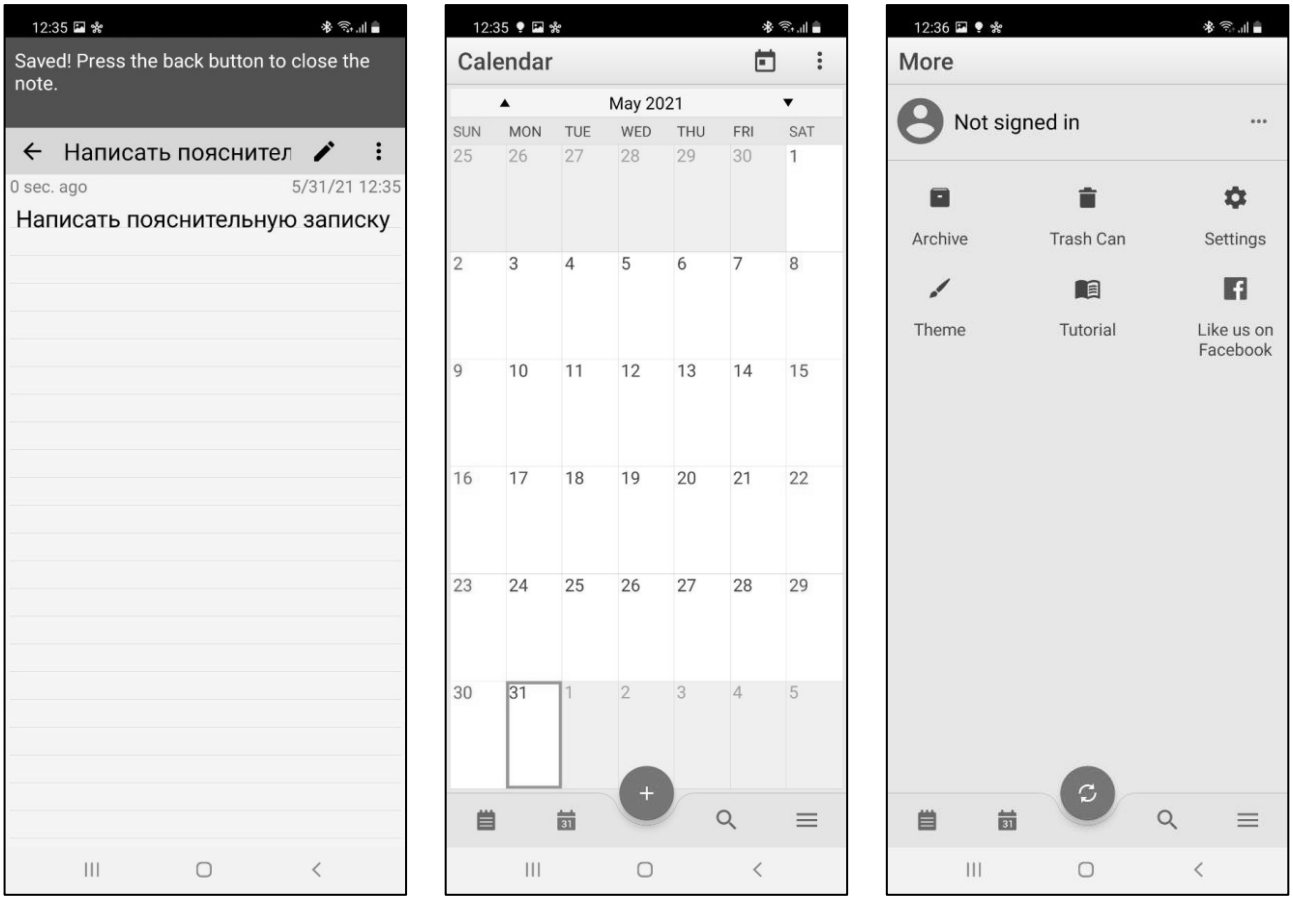



Рисунок 1.3 – Знімки екрану додатку ColorNote

## 1.3 Формування вимог до основних функцій телеграмм бота

Аналіз додатків для ведення справ та нотаток дозволив виділити їх переваги та недоліки та сформувати вимоги.

Основний функціонал телеграмм боту для ведення заміток може включати наступні можливості:

– створення заміток – користувач може створювати нові замітки, вводячи текстовий контент. Додаток повинен забезпечувати простий та зручний спосіб створення нових заміток;

– редагування та форматування – користувач може редагувати вміст заміток, включаючи видалення, редагування або переміщення тексту. Додаток може також

надавати можливість форматування тексту, таке як жирний, курсив, маркери списку, нумерація тощо, для зручності організації і вигляду заміток;

– категорії та мітки – додаток може дозволяти користувачу створювати категорії або мітки для заміток, що допомагає організувати їх за певними темами, проектами або контекстами. Це дозволяє швидко знаходити потрібну інформацію і забезпечує більшу структурованість;

– пошук та фільтрація – користувач може шукати замітки за ключовими словами або фільтрувати їх за певними критеріями, такими як дата створення, пріоритет, категорія тощо. Це допомагає знаходити необхідні замітки швидко і ефективно;

– безпека даних – з метою забезпечення конфіденційності та безпеки, додаток може пропонувати захист заміток за допомогою паролю, відбитка пальця або інших методів аутентифікації. Це гарантує, що тільки авторизовані користувачі можуть отримати доступ до приватних даних.

## 1.4 Огляд інформаційних технологій для розробки телеграмм бота

### 1.4.1 Фреймворк Spring Framework

Spring є одним з найпопулярніших фреймворків для розробки додатків на платформі Java. Spring Boot та Spring Data JPA є компонентами Spring, які значно полегшують розробку та управління додатками. Ось деякі переваги та недоліки використання Spring та його компонентів:

Розглянемо переваги фреймворку Spring та його компонентів:

– Легкість використання: Spring надає простоту в розробці завдяки своїм концепціям і шаблонам програмування. Він пропонує легкий спосіб організації коду та управління залежностями;

– Інверсія керування (IoC): Spring використовує принцип інверсії керування, що дозволяє розробникам зосередитися на реалізації бізнес-логіки, відділяючи його

від конфігурації та інфраструктури. Це полегшує тестування, розширення та підтримку додатків;

– Spring Boot: Spring Boot є компонентом, який надає конфігурацію за замовчуванням та автоматичну налаштування для додатків. Він полегшує створення самостійних, готових до використання додатків з меншими зусиллями щодо конфігурації;

– Spring Data JPA: Spring Data JPA надає простий спосіб роботи з базами даних за допомогою JPA (Java Persistence API). Він автоматично генерує SQL-запити з основних методів репозиторіїв, що значно спрощує доступ до даних та реалізацію персистентності;

Розглянемо недоліки фреймворку Spring та його компонентів:

– Навчання: Spring є великим та потужним фреймворком, що вимагає певного часу для вивчення та розуміння його концепцій і практик. Для новачків це може бути викликом, але з досвідом цей недолік зникає;

– Конфігурація: Налаштування Spring може бути складним завданням, особливо в складних або великих додатках. Правильне налаштування та управління залежностями може вимагати додаткових зусиль;

– Вагомість: За всю свою потужність та гнучкість, Spring може мати деякий вплив на продуктивність додатків. Деякі компоненти можуть використовувати більше ресурсів, ніж простіші альтернативи, і це потребує уважного масштабування;

Незважаючи на недоліки, Spring є популярним та широко використовуваним фреймворком для розробки додатків на платформі Java. Він пропонує широкий набір функціональності, хорошу підтримку спільноти та постійні оновлення, що забезпечують його розширення та покращення.

### 1.4.2 Архітектурні компоненти Java

Використання архітектурних компонентів в Java є важливим аспектом при розробці додатків, оскільки вони допомагають забезпечити структурованість,

модульність, перевикористовуваність та тестованість коду. Ось деякі ключові переваги використання архітектурних компонентів.

Масштабованість: Архітектурні компоненти, надають підхід, який сприяє масштабуваності. Вони допомагають розділити бізнес-логіку від інтерфейсу користувача і забезпечують зручний спосіб управління даними та станом додатка.

Розподілення обов'язків: Архітектурні компоненти, такі як Model-View-ViewModel (MVVM) або Model-View-Presenter (MVP), допомагають чітко розподілити обов'язки між компонентами додатка. Це полегшує розуміння та утримання кодової бази, а також сприяє зручній співпраці в командному середовищі.

Тестованість: Архітектурні компоненти підтримують тестування додатків, оскільки вони розділяють бізнес-логіку від інтерфейсу користувача. Це дозволяє розробникам легко писати автоматизовані тести для перевірки функціональності та надійності додатка.

Легка утримуваність: Архітектурні компоненти пропонують структуровану організацію коду, що спрощує утримання проекту. Це дозволяє розробникам швидко зорієнтуватися в коді, вносити зміни та вдосконалювати додаток з мінімальними зусиллями.

Перевикористання: Використання архітектурних компонентів сприяє перевикористанню коду. Вони дозволяють виділити загальну функціональність у відокремлені компоненти, які можна використовувати в різних частинах додатка або навіть в інших проектах.

Спільність розробки: Архітектурні компоненти забезпечують зручну інтеграцію та спільну роботу в командному середовищі. Це полегшує спільне управління кодовою базою та співпрацю між розробниками.

Загалом, використання архітектурних компонентів в Java рекомендується для покращення якості, швидкості розробки та легкості утримання додатків. Вони допомагають розділити обов'язки, спростити тестування та забезпечити модульність, що робить їх важливою складовою частиною процесу розробки Java-додатків.

## 1.5 Висновки до першого розділу

В першому розділі курсової роботи був проведений огляд розробки телеграм-бота за допомогою фреймворку Spring. Телеграм-боти є популярними рішеннями для комунікації та автоматизації завдань в месенджері Telegram. Фреймворк Spring надає зручні інструменти для розробки ботів та інтеграції з Telegram API.

Було розглянуто питання впливу технологій на задачу ведення справ та особистої продуктивності. Було визначено, що технології відіграють значну роль у поліпшенні особистої продуктивності та веденні справ. Вони надають користувачам широкі можливості доступу до інформації, організації задач і спрощення повсякденних обов'язків.

Далі був проведений огляд додатків-аналогів, який дозволив визначити основні переваги та недоліки існуючих мобільних додатків для ведення заміток, встановити їх основні функції та можливості. На базі проведеного огляду були визначені основні вимоги до власного телеграм боту, а також наданий детальний опис цих вимог.

Був проведений огляд інформаційних технологій для розробки телеграм бота для ведення заміток. У якості фреймворку для розробки було вирішено обрати фреймворк Spring.

# 2 ПРОЄКТУВАННЯ ТЕЛЕГРАМ БОТА ДЛЯ ВЕДЕННЯ ЗАМІТОК

## 2.1 Мета та задачі телеграм боту

Мета телеграм бота для ведення заміток полягає в наданні користувачам зручного та організованого інструменту для зберігання, організації та керування їх замітками. Головною метою додатку є полегшення процесу ведення особистих або професійних записів, щоб користувачі могли легко зберігати важливу інформацію та нагадування.

Задачі бота для ведення заміток можуть включати:

– створення нових заміток – користувачі повинні мати можливість швидко створювати нові записи заміток з використанням тексту

– організація заміток – бот повинен надати можливість створення категорій або міток для заміток, щоб користувачі могли легко групувати та організовувати свої записи;

– редагування та видалення заміток – користувачам необхідно мати можливість змінювати вміст заміток, редагувати або видаляти їх в разі потреби;

– сповіщення та нагадування – бот може надавати можливість встановлення нагадувань та сповіщень для заміток, щоб користувачі не пропустили важливі події або завдання;

Призначення телеграм бота для ведення заміток полягає в полегшенні організації та керування інформацією для користувачів. Він надає зручний спосіб зберігання і використання заміток, дозволяючи легко доступатися до них, редагувати, організовувати та забезпечувати їх безпеку. Мобільний додаток для ведення заміток сприяє покращенню особистої продуктивності, організації ідей, плануванню завдань та важливих подій, а також виконанню рутинних завдань швидко та ефективно.

## 2.2 Визначення функціональних вимог до телеграм бота

Визначення функціональних вимог є важливим етапом у процесі телеграм бота для ведення заміток. Функціональні вимоги визначають, які конкретні функції та можливості повинен мати додаток. Вони встановлюють чітку спрямованість розробки, допомагають уникнути неоднозначностей та непорозумінь.

Крім того, визначення функціональних вимог дозволяє зосередитися на потребах та вимогах користувачів. Вони допомагають врахувати, які функції та можливості будуть найбільш корисними для користувачів.

Також, функціональні вимоги слугують основою для комунікації між розробниками, дизайнерами та іншими учасниками проекту. Вони допомагають зрозуміти, що саме потрібно реалізувати та які очікувані результати.

Встановлення функціональних вимог дозволяє визначити обсяг роботи та потребу в ресурсах для реалізації додатку. Це допомагає планувати час, бюджет та ресурси проекту ефективно.

Єдиним актором мобільного додатку є актор «користувач»:

– користувач – користувач телеграм бота. Має доступ до всіх функцій та несе всю відповідальність за його роботу.

З метою визначення користувацьких історій та нефункціональних вимог до веб-ресурсу, було розроблено діаграму сценаріїв використання телеграм бота (рис. 2.1). Діаграма сценаріїв UML (Unified Modeling Language) - це графічний інструмент для опису функціональної взаємодії між користувачами та системою. Вона складається з акторів, сценаріїв та взаємодії між ними.

Ця діаграма містить основних акторів системи та описує сценарії їх взаємодії з системою. Вона допомагає проаналізувати залежності між акторами та можливими варіантами використання системи. Також ця діаграма швидко демонструє основні функції системи для розробників.

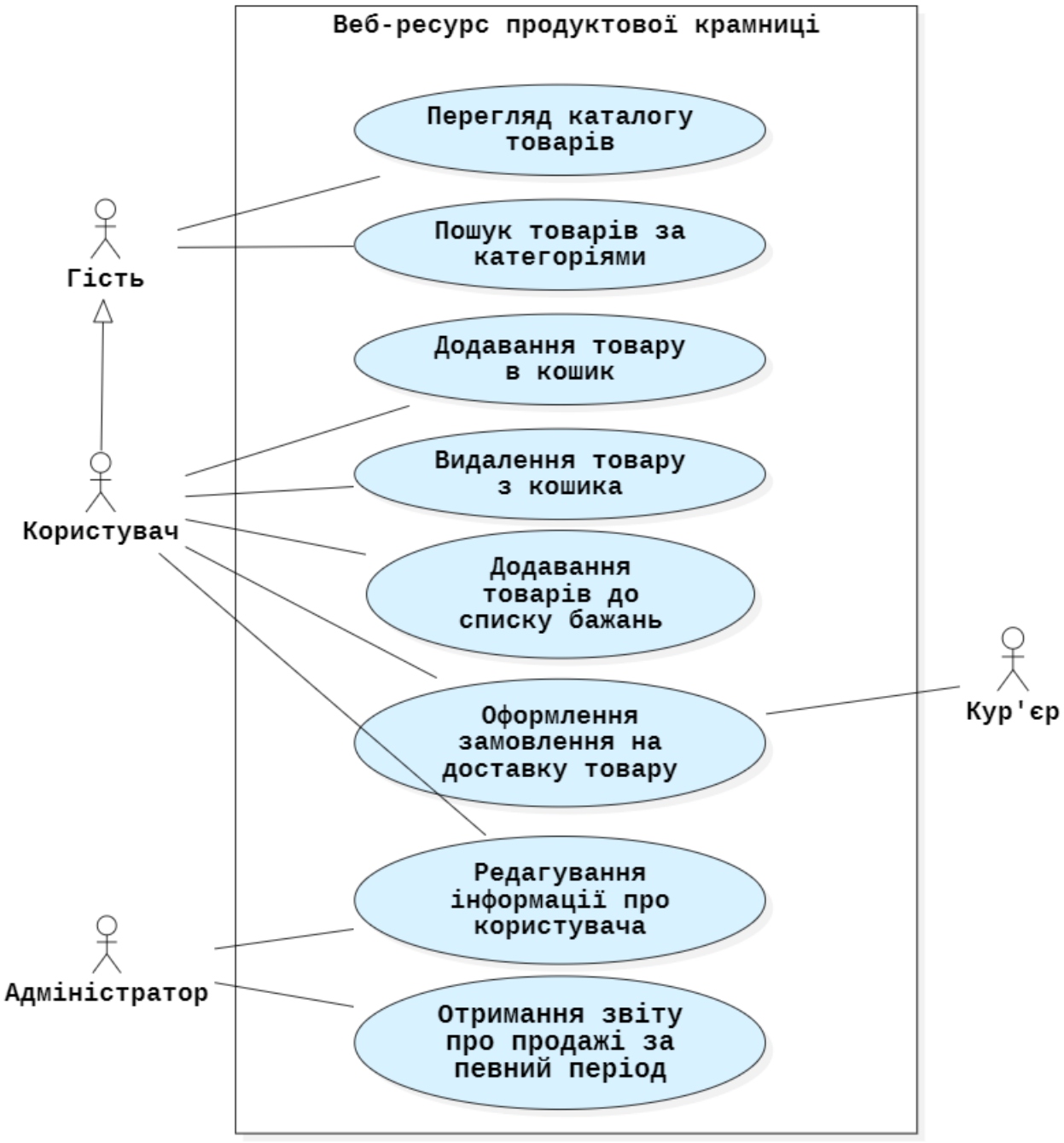



Рисунок 2.1 – Діаграма варіантів використання

## 2.3 Формування користувацьких історій для телеграм бота

Проектування діаграми прецедентів дозволяє визначити такі користувацькі історії телеграм боту для ведення заміток.

US1 Як користувач, я хочу створити нову замітку, щоб зафіксувати важливу інформацію.

Сценарій користувацької історії виглядає таким чином:

– користувач відкриває телеграм та заходить в чат до бота

– він натискає кнопку «Створити нову замітку»;

– користувач вводить текстове повідомлення або нотатку;

– нова замітка зберігається в додатку і з'являється у списку заміток;

US2 Як користувач, я хочу редагувати вміст замітки, щоб вносити зміни або доповнення до інформації.

Сценарій користувацької історії виглядає таким чином:

– користувач відкриває конкретну замітку;

– він натискає на кнопку «Редагувати» або вибирає опцію редагування;

– користувач вносить необхідні зміни або доповнення до тексту замітки;

US3 Як користувач, я хочу видаляти замітки, які більше не потрібні, для підтримки чистоти та організованості списку заміток.

Сценарій користувацької історії виглядає таким чином:

– користувач відкриває конкретну замітку або виділяє багато заміток у списку;

– він натискає кнопку «Видалити» або вибирає опцію видалення;

– замітки видаляються з додатку та зникають зі списку заміток.

## 2.4 Проектування користувацького інтерфейсу телеграм бота

На основі вимог до веб-ресурсу, які включають функціональні та нефункціональні вимоги, можна приступити до проектування користувацького інтерфейсу веб-додатку. Це вимагає розробки макетів телеграм бота, визначення сценаріїв взаємодії користувачів з цими вікнами, складання стислого опису кожного макету та визначення їх ролі в досягненні визначеної мети в цій кваліфікаційній роботі.

На рисунку 2.4 зображений макет головного вікна. Головне вікно містить два елементи. Елемент меню включає випадаючий список із командами, які користувач може викликати.

Елемент кнопок включає в собі функції відповідні назви кнопки.

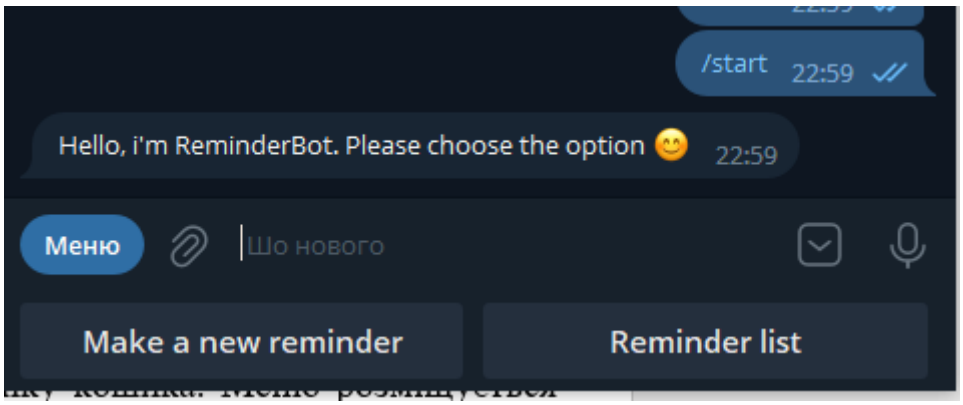



Рисунок 2.4 – Макет сторінки головного вікна

На рисунку 2.5 зображений макет вікна перегляду задач. Даний макет містить попередні елементи з головного меню. Також в макеті міститься додаткова секція із кнопками, що дозволяє користувачам обирати які саме задачі показувати.

Крім того, у вікні знаходиться посилання на розділ, де можна знайти інші товари того ж бренду або категорії. Також на сторінці знаходиться посилання на обрані товари користувача (список бажань або кошик)

Макет повинен мати привабливий дизайн і забезпечувати користувачам легку навігацію і доступ до усієї необхідної інформації про товар. Також, повинен бути доданий функціонал рекомендацій на основі попередніх покупок, щоб користувачі могли знайти товари, які їм можуть сподобатися.

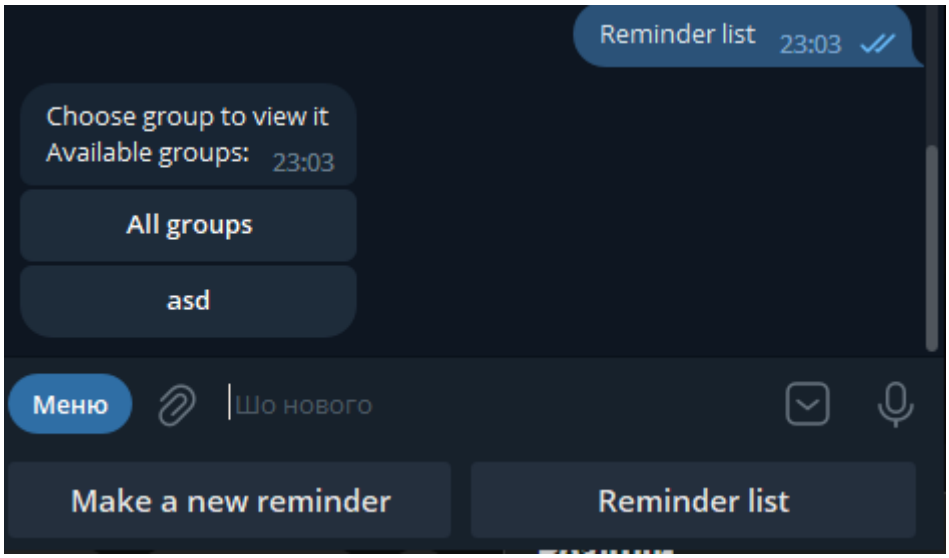



Рисунок 2.5 – Макет сторінки

## 2.5 Висновки до другого розділу

У другому розділі цієї курсової роботи було проведено проектування телеграм бота.

У процесі проектування була визначена мета, його потенційна аудиторія та основні можливості. На основі цих даних були встановлені основні вимоги, включаючи функціональні та нефункціональні. Також була розроблена діаграма сценаріїв використання веб-ресурсу. Крім того, була розроблена діаграма інформаційних потоків та визначені вхідні та вихідні інформаційні потоки системи.

З метою проектування інтерфейсу користувача були розроблені макети ключових вікон, а також діаграма станів системи. Крім того, була розроблена діаграма логічного уявлення системи, а також діаграма розгортання веб-ресурсу. Для побудови веб-ресурсу була визначена схема даних системи, а також ключові технології, які були використані в процесі проєктування та реалізації веб-ресурсу.

# 3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТЕЛЕГРАМ БОТА ДЛЯ ВЕДЕННЯ ЗАМІТОК

## 3.1 Структура програмного проєкту

Пакет програмного проекту може містити різні компоненти. Визначено ці компоненти та визначимо, які класи реалізують ці компоненти.

Першим компонентом є сервіси. Сервіси відповідають за надання певних послуг або функціональностей користувачам або іншим програмним системам. Вони можуть виконувати широкий спектр завдань залежно від своєї призначеності і специфікацій. В рамках проєкту були створені наступні сервіси:

– сервіс для парсингу дати (*DataParser.java && DataParserImpl.java*) – відповідає за парсинг дати.

– сервіс для генерації клавіатур (*KeyboardService.java && KeyboardServiceImpl.java*) – відповідає за генерацію клавіатур до відповідних State.

– сервіс для відправлення повідомлень (*MessageService.java && MessageServiceImpl.java*) – відповідає за відправлення повідомлення користувачу

– сервіс для нагадування (*RemindService.java && RemindServiceImpl.java*) – відповідає за нагадування про завдання для користувача;

Компонент сервісів відповідає за логіку бізнесу та операції, які пов'язані з даними. Сервіси можуть включати методи, які повертають дані з бази даних, змінюють їх та зберігають зміни.

Також, в проєкті містяться такі компоненти:

– репозиторії– відповідають за збереження та отримання даних з бази даних. Репозиторії можуть містити методи, які зберігають, оновлюють та видаляють дані з бази даних;

– моделі – представляють сутності, які використовуються в системі. Моделі можуть містити поля, які відображають дані з бази даних, та методи, які допомагають взаємодіяти з цими даними;

– конфігураційні файли – відповідають за конфігурування та налаштування Spring Framework. Конфігураційні файли можуть містити налаштування зв'язку з базою даних, налаштування безпеки та інші параметри, які необхідні для правильної роботи системи.

– оброблювачі(handlers) - відповідає за обробку запитів, що виникають при натисканні на кнопку або іншого інтерактивного елемента або за обробку текстових повідомлень в Telegram-боті. Він перехоплює ці запити і виконує певні дії, такі як обробка даних, що супроводжують запит, та відправка відповідей користувачу.

## 3.2 Діаграма класів

Діаграма класів є одним з видів UML-діаграм і представляє зовнішній вигляд класів, інтерфейсів та їх зв'язків у системі або додатку. Діаграма класів допомагає візуалізувати структуру системи та відношення між її компонентами.

Діаграма класів веб-ресурсу продуктової крамниці на базі фреймворку Spring наведена у додатку А.

## 3.3 Керування вихідним кодом веб додатку

Використання системи контролю версій є дуже важливим для будь-якого програмного проекту, в тому числі і для веб-ресурсу продуктової крамниці. Використання системи контролю версій дозволяє зберігати історію всіх змін, які були внесені до веб-ресурсу.

Це дозволяє в разі потреби повернутися до попередніх версій веб-ресурсу та відновити попередній стан, який був на момент попередньої версії.

Також, система контролю версій дозволяє контролювати версії веб-ресурсу, що значно полегшує роботу з веб-ресурсом та дозволяє підтримувати його в актуальному стані.

Основні метрики репозиторію веб-ресурсу наведені в таблиці 3.1.

Таблиця 3.1 – Метрики керування програмним кодом додатку

| Веб-ресурс | Кількість комітів | Кількість pull-реквестів | Кількість гілок у репозиторії |
|---|---|---|---|
| Серверна частина | 34 | 1 | 2 |

## 3.4 Функціональне тестування розробленого веб-ресурсу телеграмм бота

Функціональне тестування веб-ресурсу є дуже важливим процесом, оскільки воно дозволяє перевірити правильність функціонування різних функцій та можливостей цього ресурсу перед його введенням в експлуатацію.

Функціональне тестування дозволяє перевірити, що всі функції та можливості веб-ресурсу працюють правильно, як очікувалося, що забезпечує коректну роботу веб-сайту та надійність для його використання. .

Для проведення функціонального тестування необхідно розробити протокол тестування.

Протокол тестування – це документ, який містить опис кроків, процедур та результатів тестування програмного продукту, системи чи окремої їх частини. Протокол тестування має на меті систематично задокументувати процес тестування та отримані результати, щоб забезпечити максимальну об'єктивність та повторюваність результатів.

Для функціонального тестування веб-ресурсу був розроблений наступний протокол тестування.

TC1 Тест-кейс для додавання нового товару:

– відкрити телеграмм бот;

– обрати опцію «Make a new reminder»;

– ввести назву групи;

– ввести завдання;

– натиснути кнопку «Enter»;

– переконатись, що нове завдання з'явилось в списку.

ТС2 Тест-кейс для редагування існуючого товару:

– відкрити телеграмм бот;

– відкрити список завдань

– натиснути кнопку «Редагувати»;

– обрати завдання, яке потрібно змінити;

– написати оновлене завдання;

– натиснути кнопку «Enter»;

– переконатись, що зміни збереглися і відображаються правильно.

ТС3 Тест-кейс для видалення товару:

– відкрити телеграм бот;

– відкрити список завдань;

– натиснути кнопку «Видалити»;

– обрати завдання, яке потрібно видалити;

– переконатись, що товар більше не відображається в програмі;

ТС4 Тест-кейс для встановлення часового нагадування

– відкрити телеграм бот;

– відкрити список завдань;

– натиснути кнопку «Deadline»;

– обрати завдання, про яке потрібно нагадати

– написати дату та час в який потрібно нагадати

– дочекатись та переконатись, що нагадування відбулось

Результати функціонального тестування веб-ресурсу продуктової крамниці наведені в таблиці 3.2.

Як можна побачити з результатів тестування, всі функціональні тести пройдені успішно, що свідчить про те, що фактична поведінка веб-ресурсу співпадає з очікуваною поведінкою, визначеною у функціональних вимогах.

Таблиця 3.2 – Протокол функціонального тестування веб-ресурсу

| Номер тест-кейсу | Очікуваний результат | Фактичний результат | Результат тестування |
|---|---|---|---|
| TC1 | Успішне додання нового завдання | Нове завдання успішно додається до бази даних | Успішно |
| TC2 | Завдання успішно оновлюється | Завдання успішно оновлюється в базі даних | Успішно |
| TC3 | Успішне видалення завдання | Запис в базі даних успішно видаляється | Успішно |
| TC4 | Успішне встановлення часового нагадування | Нагадування працює | Успішно |

## 3.5 Інструкція користувача веб-ресурсу

Для забезпечення успішного користувацького досвіду використання розробленого веб-ресурсу необхідно скласти інструкцію користувача. Це допоможе користувачам легко зорієнтуватись у функціоналі веб-ресурсу телеграмм бота та ефективно використовувати його.

На рисунку 3.2 зображено початок роботи з ботом. Для початку роботи потрібно написати в чат “/start” або обрати цю команду із меню зліва.

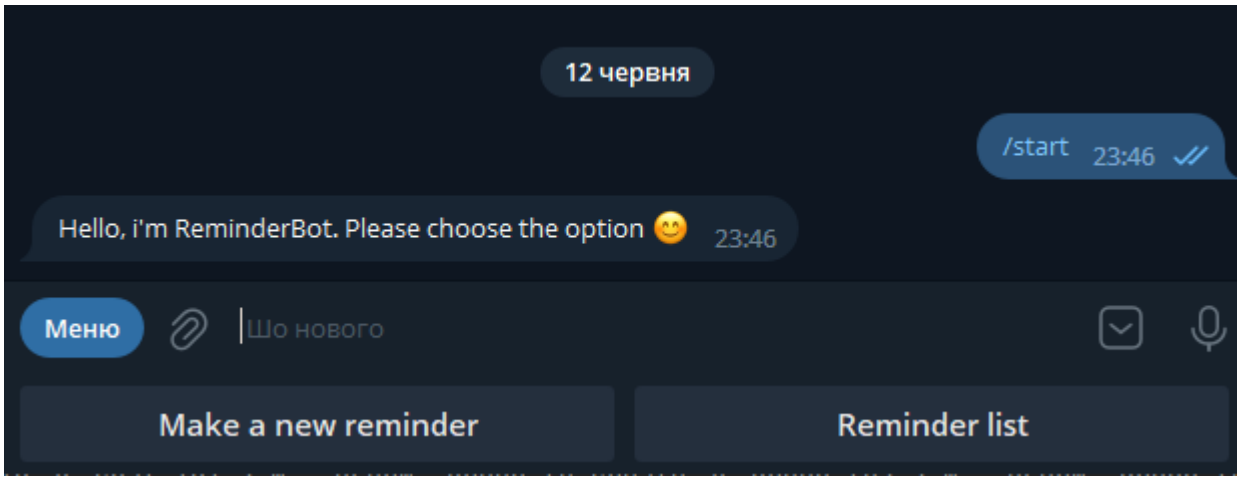



Рисунок 3.2 – Початок роботи с ботом

На рисунку 3.3 натиснуто кнопку “Make a new reminder”, потім потрібно написати назву групи для завдання, або натиснути на “GENERAL”(воно буде

скопійоване) і відправити боту. Потім бот попросить написати саме завдання, яке вам потрібно ввести.

.

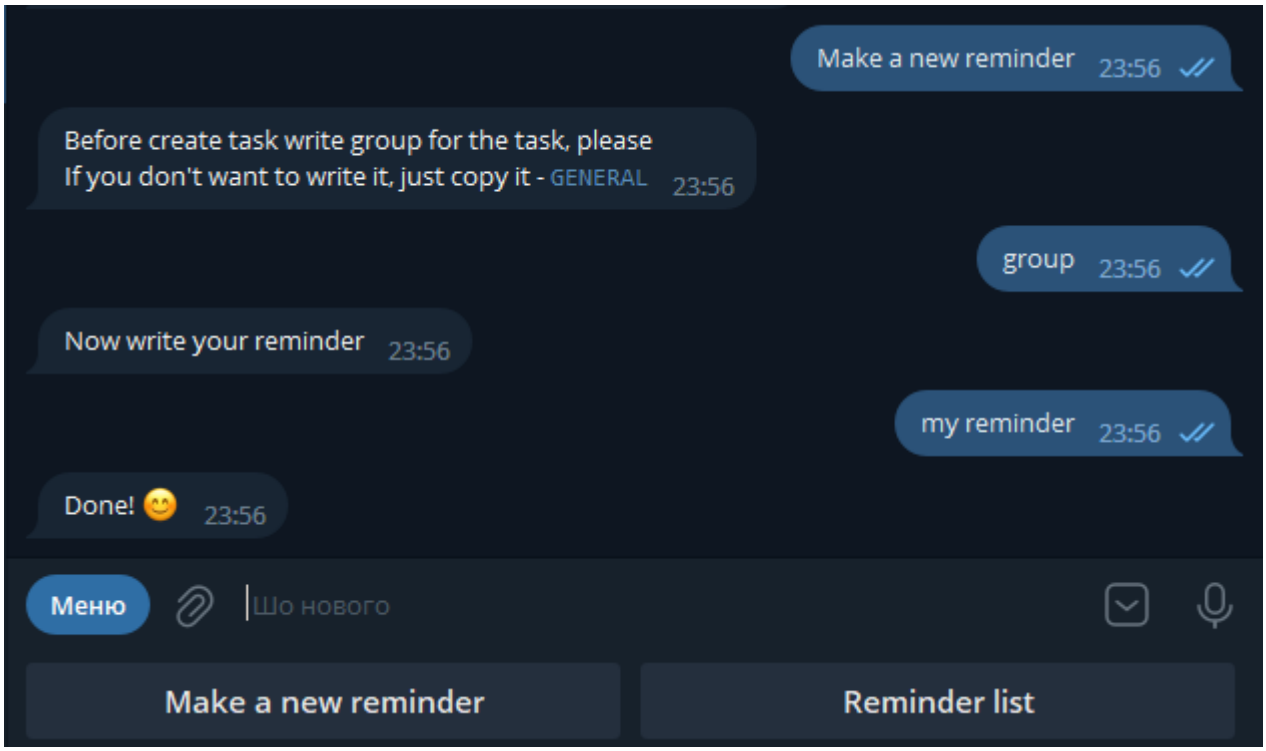



Рисунок 3.3 – Додання завдання

На рисунку 3.4 зображений перший етап, щоб переглянути завдання. Перший етап заключається в тому, щоб нажати кнопку “Reminder List” та обрати(нажати на кнопку) групу, яку хочете переглянути.

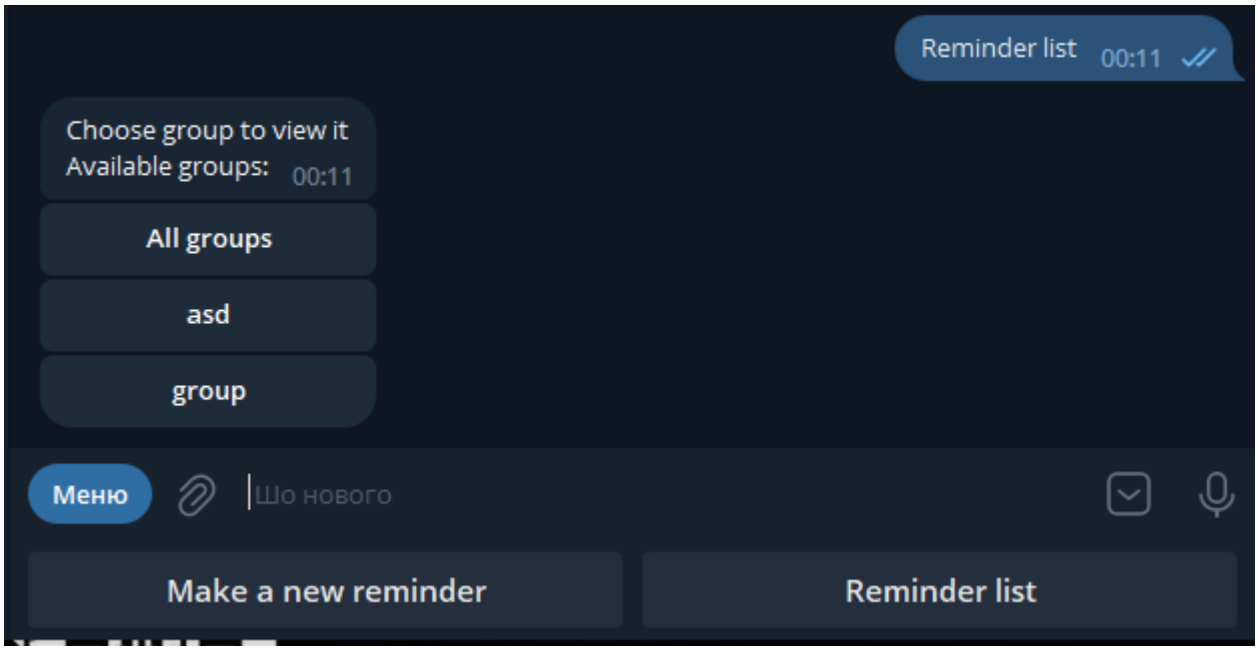



Рисунок 3.4 – Перегляд груп

На рисунку 3.5 з’являється список завдань після обрання групи та під всіма завданнями з’являються доступні дії з завданнями.

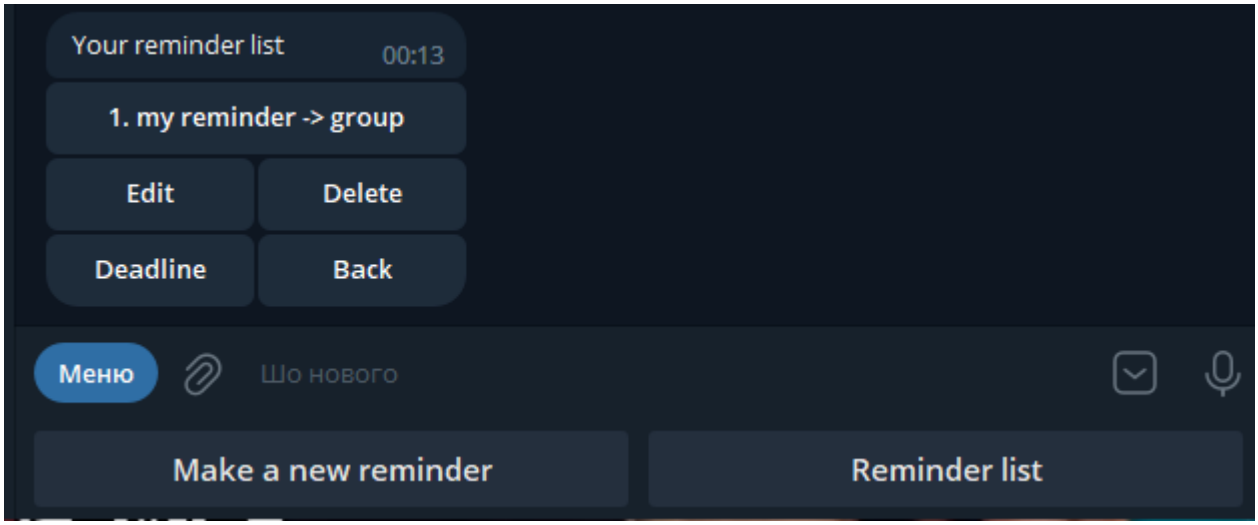



Рисунок 3.5 – Перегляд завдань

На рисунку 3.6 результат натискання кнопки “Edit”, потім потрібно обрати яке саме завдання ми хочемо оновити після цього пишемо оновлений зміст завдання.

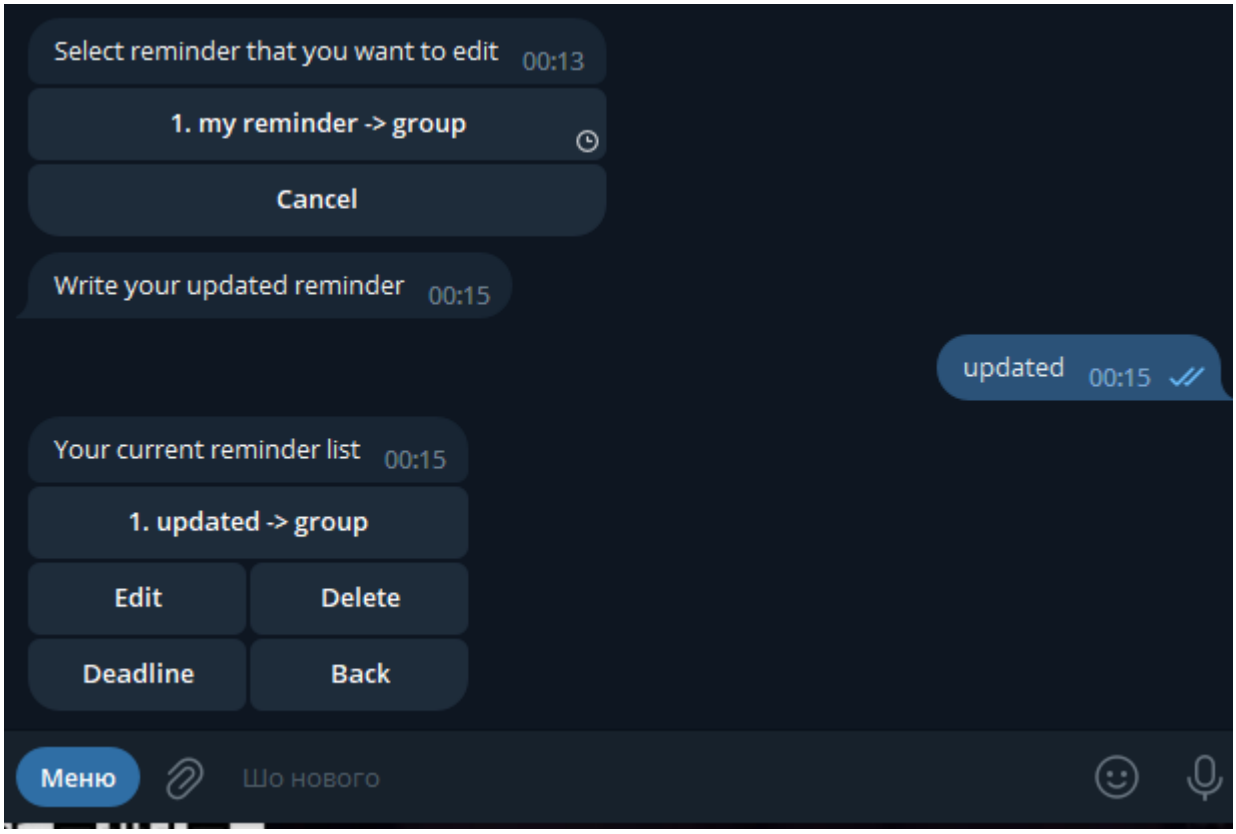



Рисунок 3.6 – Оновлення завдання

На рисунку 3.7 результат натискання кнопки “Deadline”, потім написати дату коли ми хочемо отримати нагадування і дочекатися цього моменту.

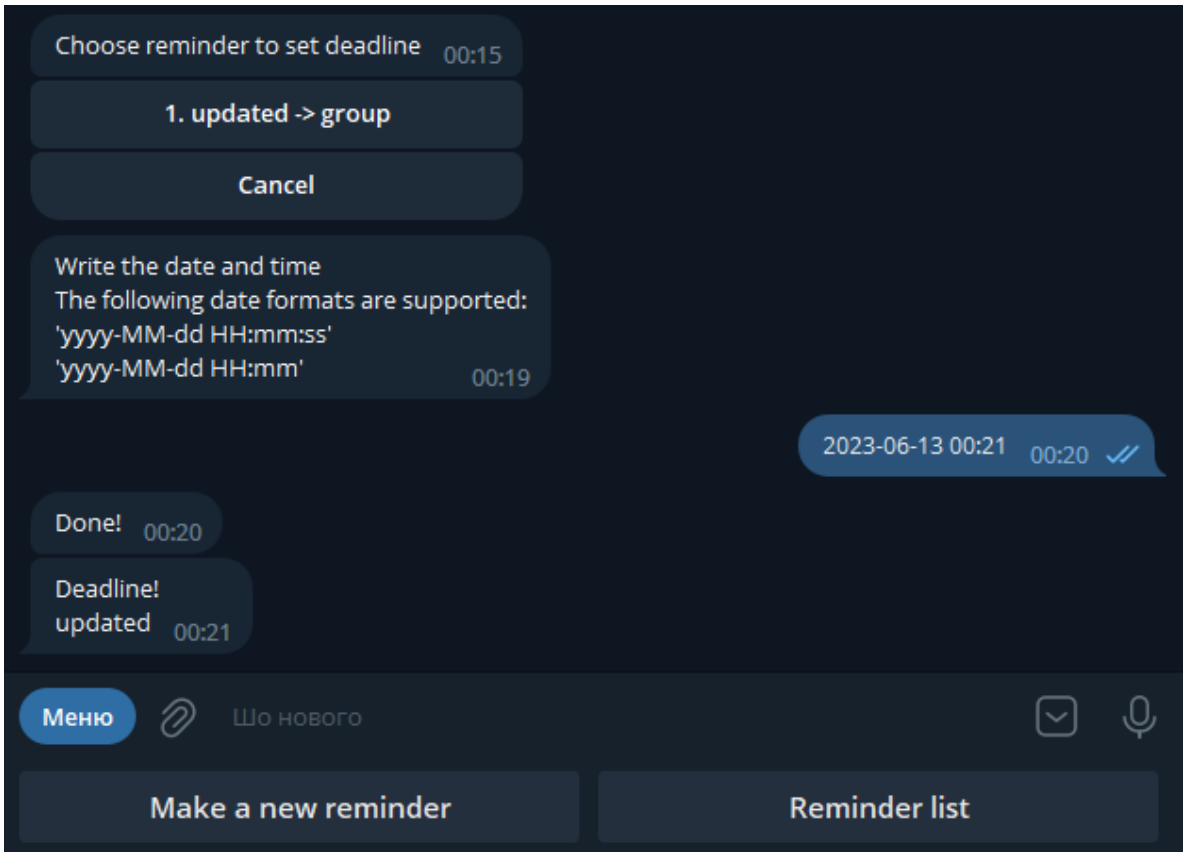



Рисунок 3.7 – Екранна форма вікна кошику товарів

На рисунку 3.8 дія кнопки “Back”, вона повертає до стартового стану телеграм бот.

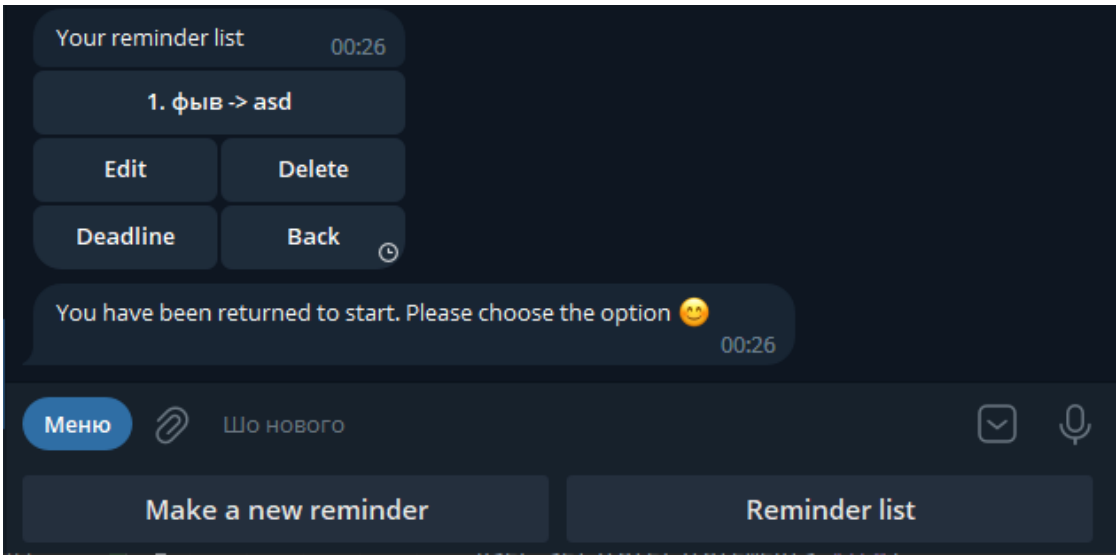



Рисунок 3.8 – Екранна форма оформлення замовлення

## 3.6 Вихідний код веб-ресурсу

Вихідний код програмних класів розробленого веб-ресурсу наведений в Додатку Б даної роботи.

## 3.7 Висновки до третього розділу

В даному розділі курсової роботи було проведено програмну реалізацію веб-ресурсу телеграмм бота на базі фреймворку Spring. Для того, щоб провести програмну реалізацію, був використаний проєкт веб-ресурсу, який був розроблений в другому розділі даної роботи.

Перш за все, була розглянута структура серверного програмного проєкту. Були визначені основні програмні компоненти, з яких складається програмний проєкт веб-ресурсу, а також наданий перелік основних класів, які відносяться до цих компонентів.

Далі була розроблена концептуальна діаграма класів, яка дозволяє визначити основні класи веб-ресурсу та визначити взаємовідносини між цими класами.

Окремо було розглянуто питання використання системи контроля версій для упорядкування розробки веб-ресурсу. Були визначені основні репозиторії проєкту та надані основні показники метрик цих репозиторіїв.

Далі було проведено функціональне тестування розробленого веб-ресурсу. Був розроблений протокол тестування у вигляді множини тест-кейсів та проведено тестування. Було визначено, що всі тест-кейси були пройдені, а отже фактична поведінка веб-ресурсу співпадає з очікуваною поведінкою, визначеною у функціональних вимогах.

Також була розроблена інструкція користувача, який визначає функціонал додатку та можливі дії користувача. Крім того, наданий вихідний код розробленого веб-ресурсу.

# ВИСНОВКИ

У даній курсовій роботі був розроблений телеграм бот для заміток на базі фреймворку Spring.

Такий веб-ресурс дозволяє користувачам переглядати задачі, оновлювати та видаляти їх, робити нагадування. Таким чином, мета, поставлена перед даною курсовою роботою досягнута в повному обсязі. Для досягнення мети були вирішені наступні задачі.

У першому розділі даної курсової роботи була детально розглянута предметна область створення веб-ресурсу. Були визначені основні завдання, які повинні бути вирішені в процесі розробки веб-ресурсу. Був проведений аналіз існуючих аналогів та визначені ключові вимоги до створення власного веб-ресурсу. Для розробки серверної частини – Java Spring. Також, було вирішено використовувати PostgreSQL в якості СУБД для даного веб-ресурсу.

У другому розділі роботи було проведено проектування веб-ресурсу. Була визначена мета, його потенційна аудиторія та основні можливості. Визначені основні вимоги до веб-ресурсу, включаючи функціональні та нефункціональні. Також була розроблена діаграма сценаріїв, діаграма інформаційних потоків, діаграма станів. Крім того, була розроблена діаграма логічного уявлення системи, а також діаграма розгортання веб-ресурсу. Також була визначена схема даних системи та наданий опис ключовим технологіям.

В третьому розділі роботи було проведено програмну реалізацію веб-ресурсу. Розглянута структура серверного програмного проєкту, визначені основні програмні компоненти, розроблена діаграма класів. Розглянуто питання використання системи контроля версій для упорядкування розробки веб-ресурсу. Проведено функціональне тестування, розроблена інструкція користувача у вигляді множини знімків екрану та пояснювального тексту та наданий вихідний код розробленого веб-ресурсу.

## ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. The importance of information technology in retail. URL: https://www.retailcouncil.org/community/technology/the-importance-of-information-technology-in-retail/ (дата звернення 01.03.2023).

2. The Importance of Information Technology in Retail. URL: https://www.aeologic.com/blog/the-importance-of-information-technology-in-retail/ (дата звернення 01.03.2023).

3. Fozzyshop. URL: https://fozzyshop.ua/odesa/ (дата звернення 01.03.2023).

4. Інтернет-магазин продуктів «Таврія В». URL: https://www.tavriav.ua/ (дата звернення 01.03.2023).

5. Онлайн-супермаркет «Сільно». URL: https://silpo.ua/ (дата звернення 01.03.2023).

6. Angular is a platform for building mobile and desktop web applications. URL: https://angular.io/ (дата звернення 01.03.2023).

7. Angular components overview. URL: https://angular.io/guide/component-overview (дата звернення 01.03.2023).

8. The Good and the Bad of Angular Development. URL: https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-angular-development/ (дата звернення 01.03.2023).

9. Node.js – an open-source, cross-platform JavaScript runtime environment. URL: https://nodejs.org/en (дата звернення 01.03.2023).

10. Django: The web framework for perfectionists with deadlines. URL: https://www.djangoproject.com/ (дата звернення 01.03.2023).

11.Ruby on Rails — A web-app framework. URL: https://rubyonrails.org/ (дата звернення 01.03.2023).

12. Laravel - The PHP Framework For Web Artisans. URL: https://laravel.com/ (дата звернення 01.03.2023).

13. Spring | Home. URL: https://spring.io/ (дата звернення 01.03.2023).

14. Express – fast, unopinionated, minimalist web framework for Node.js. URL: https://expressjs.com/ (дата звернення 01.03.2023).

15. Welcome to Flask. URL: https://flask.palletsprojects.com/ (дата звернення 01.03.2023).

16. ASP.NET – open-source web framework for .NET infrastructure. URL: https://dotnet.microsoft.com/en-us/apps/aspnet (дата звернення 01.03.2023).

17. Java spring framework – pros, cons, common mistakes. URL: https://bit.ly/3A1dS4w (дата звернення 01.03.2023).

18. Key Components and Internals of Spring Boot Framework. URL: https://bit.ly/3ojMdtb (дата звернення 01.03.2023).

19. MySQL Home. URL: https://www.mysql.com/ (дата звернення 01.03.2023).

20. PostgreSQL: The world's most advanced open source database. URL: https://www.postgresql.org/ (дата звернення 01.03.2023).

21. MongoDB Atlas Database - The Database For Modern Apps. URL: https://bit.ly/414ADjX (дата звернення 01.03.2023).

22. Industry-leading performance and security with SQL Server 2019. URL: https://www.microsoft.com/en-us/sql-server/sql-server-2019?rtc=1 (дата звернення 01.03.2023).

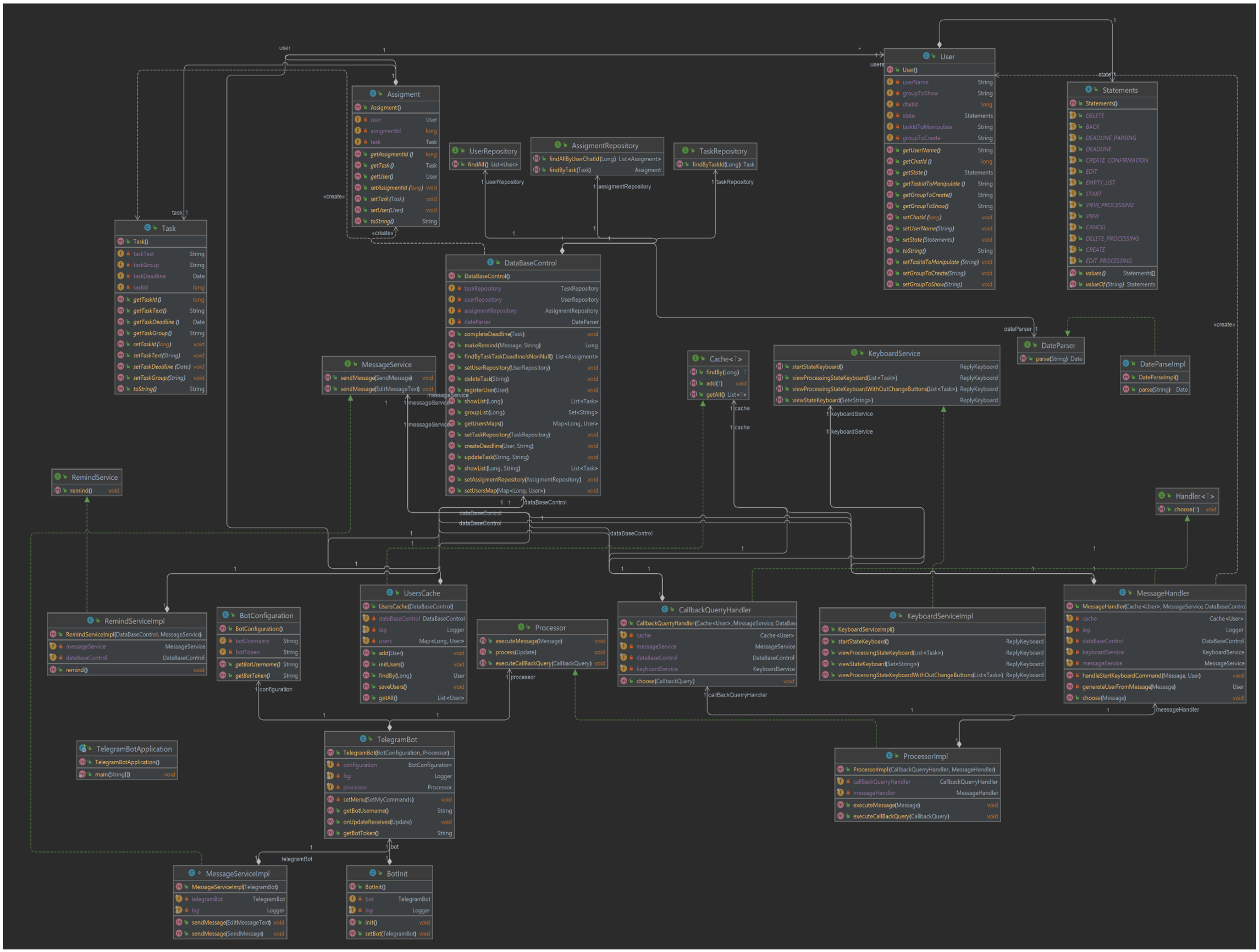
23. Database | Oracle. URL: https://www.oracle.com/database/ (дата звернення 01.03.2023).

24. SQLite Home Page. URL: https://sqlite.org/index.html (дата звернення 01.03.2023).

25. Amazon RDS Features. URL: https://aws.amazon.com/rds/features/?nc1=h_ls (дата звернення 01.03.2023).

26. Firebase Home. URL: https://firebase.google.com/ (дата звернення 01.03.2023).

Додаток А


User
User()
userName String
groupToShow String
chatId long
state Statements
taskIdToManipulate String
groupToCreate String
getUserName() String
getChatId() long
getState() Statements
getTaskIdToManipulate() String
getGroupToCreate() String
getGroupToShow() String
setChatId(long) void
setUserName(String) void
setState(Statements) void
toString() String
setTaskIdToManipulate(String) void
setGroupToCreate(String) void
setGroupToShow(String) void
Statements
Statements()
DELETE
BACK
DEADLINE_PARSING
DEADLINE
CREATE_CONFIRMATION
EDIT
EMPTY_LIST
START
VIEW_PROCESSING
VIEW
CANCEL
DELETE_PROCESSING
CREATE
EDIT_PROCESSING
values() Statements[]
valueOf(String) Statements
Assigment
Assigment()
user User
assigmentId long
task Task
getAssigmentId() long
getTask() Task
getUser() User
setAssigmentId(long) void
setTask(Task) void
setUser(User) void
toString() String
Task
Task()
taskText String
taskGroup String
taskDeadline Date
taskId long
getTaskId() long
getTaskText() String
getTaskDeadline() Date
getTaskGroup() String
setTaskId(long) void
setTaskText(String) void
setTaskDeadline(Date) void
setTaskGroup(String) void
toString() String
UserRepository
findAll() List<User>
AssigmentRepository
findAllByUserChatId(Long) List<Assigment>
findByTask(Task) Assigment
TaskRepository
findByTaskId(Long) Task
DataBaseControl
DataBaseControl()
taskRepository TaskRepository
userRepository UserRepository
assigmentRepository AssigmentRepository
dateParser DateParser
completeDeadline(Task) void
makeRemind(Message, String) Long
findByTaskTaskDeadlineIsNonNull() List<Assigment>
setUserRepository(UserRepository) void
deleteTask(String) void
registerUser(User) void
showList(Long) List<Task>
groupList(Long) Set<String>
getUsersMaps() Map<Long, User>
setTaskRepository(TaskRepository) void
createDeadline(User, String) void
updateTask(String, String) void
showList(Long, String) List<Task>
setAssigmentRepository(AssigmentRepository) void
setUsersMap(Map<Long, User>) void
MessageService
sendMessage(SendMessage) void
sendMessage(EditMessageText) void
Cache<T>
findBy(Long) T
add() void
getAll() List<T>
KeyboardService
startStateKeyboard() ReplyKeyboard
viewProcessingStateKeyboard(List<Task>) ReplyKeyboard
viewProcessingStateKeyboardWithOutChangeButtons(List<Task>) ReplyKeyboard
viewStateKeyboard(Set<String>) ReplyKeyboard
DateParser
parse(String) Date
DateParseImpl
DateParseImpl()
parse(String) Date
RemindService
remind() void
Handler<T>
choose(T) void
RemindServiceImpl
RemindServiceImpl(DataBaseControl, MessageService)
messageService MessageService
dataBaseControl DataBaseControl
remind() void
BotConfiguration
BotConfiguration()
botUsername String
botToken String
getBotUsername() String
getBotToken() String
UsersCache
UsersCache(DataBaseControl)
dataBaseControl DataBaseControl
log Logger
users Map<Long, User>
add(User) void
initUsers() void
findBy(Long) User
saveUsers() void
getAll() List<User>
Processor
executeMessage(Message) void
process(Update) void
executeCallBackQuery(CallbackQuery) void
CallbackQuerryHandler
CallbackQuerryHandler(Cache<User>, MessageService, DataBas
cache Cache<User>
messageService MessageService
dataBaseControl DataBaseControl
keyboardService KeyboardService
choose(CallbackQuery) void
KeyboardServiceImpl
KeyboardServiceImpl()
startStateKeyboard() ReplyKeyboard
viewProcessingStateKeyboard(List<Task>) ReplyKeyboard
viewStateKeyboard(Set<String>) ReplyKeyboard
viewProcessingStateKeyboardWithOutChangeButtons(List<Task>) ReplyKeyboard
MessageHandler
MessageHandler(Cache<User>, MessageService, DataBaseContro
cache Cache<User>
log Logger
dataBaseControl DataBaseControl
keyboardService KeyboardService
messageService MessageService
handleStartKeyboardCommand(Message, User) void
generateUserFromMessage(Message) User
choose(Message) void
TelegramBotApplication
TelegramBotApplication()
main(String[]) void
TelegramBot
TelegramBot(BotConfiguration, Processor)
configuration BotConfiguration
log Logger
processor Processor
setMenu(SetMyCommands) void
getBotUsername() String
onUpdateReceived(Update) void
getBotToken() String
ProcessorImpl
ProcessorImpl(CallbackQuerryHandler, MessageHandler)
callBackQuerryHandler CallbackQuerryHandler
messageHandler MessageHandler
executeMessage(Message) void
executeCallBackQuery(CallbackQuery) void
MessageServiceImpl
MessageServiceImpl(TelegramBot)
telegramBot TelegramBot
log Logger
sendMessage(EditMessageText) void
sendMessage(SendMessage) void
BotInit
BotInit()
bot TelegramBot
log Logger
init() void
setBot(TelegramBot) void
user
users
state
task
«create»
userRepository
assigmentRepository
taskRepository
dateParser
messageService
dataBaseControl
cache
keyboardService
configuration
processor
callBackQuerryHandler
messageHandler
telegramBot
bot

Додаток Б

```
package ua.onpu;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.scheduling.annotation.EnableScheduling;

@SpringBootApplication
@EnableScheduling
public class TelegramBotApplication {

    public static void main(String[] args) {
        SpringApplication.run(TelegramBotApplication.class, args);
    }

}
```

```
package ua.onpu;

import lombok.extern.log4j.Log4j;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import org.telegram.telegrambots.bots.TelegramLongPollingBot;
import org.telegram.telegrambots.meta.api.methods.commands.SetMyCommands;
import org.telegram.telegrambots.meta.api.objects.Update;
import org.telegram.telegrambots.meta.api.objects.commands.BotCommand;
import org.telegram.telegrambots.meta.api.objects.commands.scope.BotCommandScopeDefault;
```

```
import org.telegram.telegrambots.meta.exceptions.TelegramApiException;
import ua.onpu.config.BotConfiguration;
import ua.onpu.process.Processor;

import java.util.ArrayList;
import java.util.List;

@Component
@Log4j
public class TelegramBot extends TelegramLongPollingBot {

    private final BotConfiguration configuration;
    private final Processor processor;

    @Autowired
    public TelegramBot(BotConfiguration configuration, Processor processor) {
        this.configuration = configuration;
        this.processor = processor;

        List<BotCommand> botCommands = new ArrayList<>();
        botCommands.add(new BotCommand("/start", "start bot"));
        setMenu(new SetMyCommands(botCommands, new BotCommandScopeDefault(), null));
    }

    @Override
    public String getBotUsername() {
        return configuration.getBotUsername();
    }
```

```
    @Override
    public String getBotToken() {
        return configuration.getBotToken();
    }


    @Override
    public void onUpdateReceived(Update update) {
        processor.process(update);
    }

    private void setMenu(SetMyCommands commands) {
        try {
            execute(commands);
        } catch (TelegramApiException e) {
            log.error("Error: " + e.getMessage());
        }
    }

}
```

```
package ua.onpu.service;

public interface RemindService {

    void remind();

}
```

```
package ua.onpu.service;

public interface RemindService {

    void remind();

}
```

```
package ua.onpu.service;


import org.telegram.telegrambots.meta.api.objects.replykeyboard.ReplyKeyboard;
import ua.onpu.entity.Task;
import java.util.List;
import java.util.Set;

public interface KeyboardService {

    ReplyKeyboard startStateKeyboard();
    ReplyKeyboard viewProcessingStateKeyboard(List<Task> list);
    ReplyKeyboard viewProcessingStateKeyboardWithOutChangeButtons(List<Task> list);
    ReplyKeyboard viewStateKeyboard(Set<String> group);


}
```

```
package ua.onpu.service;

import java.text.ParseException;
```

```
import java.util.Date;

public interface DateParser {

    Date parse(String date) throws ParseException;

}
```

```
package ua.onpu.service.impl;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Service;
import org.telegram.telegrambots.meta.api.methods.send.SendMessage;
import ua.onpu.entity.Assigment;
import ua.onpu.entity.DataBaseControl;
import ua.onpu.service.MessageService;
import ua.onpu.service.RemindService;

import java.util.Date;
import java.util.List;

@Service
public class RemindServiceImpl implements RemindService {

    private final DataBaseControl dataBaseControl;
    private final MessageService messageService;

    @Autowired
```

```
    public RemindServiceImpl(DataBaseControl dataBaseControl, MessageService messageService) {
        this.dataBaseControl = dataBaseControl;
        this.messageService = messageService;
    }

    @Override
    @Scheduled(fixedRate = 60_000)
    public void remind() {
        List<Assigment> assigmentList = dataBaseControl.findByTaskTaskDeadlineIsNonNull();
        Date date = new Date();

        for (Assigment a : assigmentList) {
            if (a.getTask().getTaskDeadline().before(date)) {
                messageService.sendMessage(SendMessage.builder()
                        .chatId(a.getUser().getChatId())
                        .text("Deadline!\n" + a.getTask().getTaskText())
                        .build());

                dataBaseControl.completeDeadline(a.getTask());
            }
        }

    }
}
```

```
package ua.onpu.service.impl;

import lombok.extern.log4j.Log4j;
import org.springframework.beans.factory.annotation.Autowired;
```

```
import org.springframework.context.annotation.Lazy;
import org.springframework.stereotype.Service;
import org.telegram.telegrambots.meta.api.methods.send.SendMessage;
import org.telegram.telegrambots.meta.api.methods.updatingmessages.EditMessageText;
import org.telegram.telegrambots.meta.exceptions.TelegramApiException;
import ua.onpu.TelegramBot;
import ua.onpu.service.MessageService;

@Log4j
@Service
class MessageServiceImpl implements MessageService {

    private final TelegramBot telegramBot;

    @Autowired
    public MessageServiceImpl(@Lazy TelegramBot telegramBot) {
        this.telegramBot = telegramBot;
    }

    @Override
    public void sendMessage(SendMessage sendMessage) {
        try {
            telegramBot.execute(sendMessage);
        } catch (TelegramApiException e) {
            log.error(e.getMessage());
        }
    }
```

```
    @Override
    public void sendMessage(EditMessageText sendMessage) {
        try {
            telegramBot.execute(sendMessage);
        } catch (TelegramApiException e) {
            log.error(e.getMessage());
        }
    }

}
```

```
package ua.onpu.service.impl;

import org.springframework.stereotype.Service;
import org.telegram.telegrambots.meta.api.objects.replykeyboard.InlineKeyboardMarkup;
import org.telegram.telegrambots.meta.api.objects.replykeyboard.ReplyKeyboard;
import org.telegram.telegrambots.meta.api.objects.replykeyboard.ReplyKeyboardMarkup;
import org.telegram.telegrambots.meta.api.objects.replykeyboard.buttons.InlineKeyboardButton;
import org.telegram.telegrambots.meta.api.objects.replykeyboard.buttons.KeyboardRow;
import ua.onpu.entity.Task;
import ua.onpu.service.KeyboardService;

import java.util.ArrayList;
import java.util.List;
import java.util.Set;

@Service
public class KeyboardServiceImpl implements KeyboardService {
```

```
public KeyboardServiceImpl() {
}

@Override
public ReplyKeyboard startStateKeyboard() {
    ReplyKeyboardMarkup replyKeyboardMarkup = new ReplyKeyboardMarkup();

    List<KeyboardRow> keyboardRows = new ArrayList<>();

    KeyboardRow row = new KeyboardRow();

    row.add("Make a new reminder");
    row.add("Reminder list");

    keyboardRows.add(row);

    replyKeyboardMarkup.setKeyboard(keyboardRows);
    replyKeyboardMarkup.setResizeKeyboard(true);

    return replyKeyboardMarkup;
}

@Override
public ReplyKeyboard viewProcessingStateKeyboard(List<Task> list) {
    InlineKeyboardMarkup inlineKeyboardMarkup = new InlineKeyboardMarkup();

    List<List<InlineKeyboardButton>> rowButton = new ArrayList<>();
```

```
if (list.isEmpty()) {
    InlineKeyboardButton inlineKeyboardButton = new InlineKeyboardButton();

    List<InlineKeyboardButton> buttonList = new ArrayList<>();

    inlineKeyboardButton.setText("Add at least one reminder");
    inlineKeyboardButton.setCallbackData("EMPTY_LIST");
    buttonList.add(inlineKeyboardButton);
    rowButton.add(buttonList);

    inlineKeyboardMarkup.setKeyboard(rowButton);
    return inlineKeyboardMarkup;
}

int i = 1;
for (Task t : list) {
    InlineKeyboardButton inlineKeyboardButton = new InlineKeyboardButton();
    List<InlineKeyboardButton> buttonList = new ArrayList<>();

    inlineKeyboardButton.setText(i++ + ". " + t.getTaskText() + " -> " + t.getTaskGroup());
    inlineKeyboardButton.setCallbackData(Long.toString(t.getTaskId()));

    buttonList.add(inlineKeyboardButton);
    rowButton.add(buttonList);
}

List<InlineKeyboardButton> buttonList = new ArrayList<>();
InlineKeyboardButton button = new InlineKeyboardButton();
```

```
button.setText("Edit");
button.setCallbackData("EDIT");

buttonList.add(button);

button = new InlineKeyboardButton();
button.setText("Delete");
button.setCallbackData("DELETE");

buttonList.add(button);
rowButton.add(buttonList);

buttonList = new ArrayList<>();
button = new InlineKeyboardButton();

button.setText("Deadline");
button.setCallbackData("DEADLINE");

buttonList.add(button);

button = new InlineKeyboardButton();
button.setText("Back");
button.setCallbackData("BACK");

buttonList.add(button);
rowButton.add(buttonList);
```

```
    inlineKeyboardMarkup.setKeyboard(rowButton);
    return inlineKeyboardMarkup;
}

@Override
public ReplyKeyboard viewProcessingStateKeyboardWithOutChangeButtons(List<Task> list) {
    InlineKeyboardMarkup inlineKeyboardMarkup = new InlineKeyboardMarkup();

    List<List<InlineKeyboardButton>> rowButton = new ArrayList<>();

    if (list.isEmpty()) {
        InlineKeyboardButton inlineKeyboardButton = new InlineKeyboardButton();

        List<InlineKeyboardButton> buttonList = new ArrayList<>();

        inlineKeyboardButton.setText("Add at least one reminder");
        inlineKeyboardButton.setCallbackData("EMPTY_LIST");
        buttonList.add(inlineKeyboardButton);
        rowButton.add(buttonList);

        inlineKeyboardMarkup.setKeyboard(rowButton);
        return inlineKeyboardMarkup;
    }

    int i = 1;
    for (Task t : list) {
        InlineKeyboardButton inlineKeyboardButton = new InlineKeyboardButton();
        List<InlineKeyboardButton> buttonList = new ArrayList<>();
```

```
            inlineKeyboardButton.setText(i++ + ". " + t.getTaskText() + " -> " + t.getTaskGroup());
            inlineKeyboardButton.setCallbackData(Long.toString(t.getTaskId()));

            buttonList.add(inlineKeyboardButton);
            rowButton.add(buttonList);
        }

        InlineKeyboardButton button = new InlineKeyboardButton();
        List<InlineKeyboardButton> buttonList = new ArrayList<>();
        button.setText("Cancel");
        button.setCallbackData("CANCEL");

        buttonList.add(button);
        rowButton.add(buttonList);

        inlineKeyboardMarkup.setKeyboard(rowButton);
        return inlineKeyboardMarkup;
    }

    @Override
    public ReplyKeyboard viewStateKeyboard(Set<String> group) {
        InlineKeyboardMarkup inlineKeyboardMarkup = new InlineKeyboardMarkup();
        List<List<InlineKeyboardButton>> rowsInLine = new ArrayList<>();

        if (group.isEmpty()) {
            InlineKeyboardButton inlineKeyboardButton = new InlineKeyboardButton();
            List<InlineKeyboardButton> buttonList = new ArrayList<>();
```

```
    inlineKeyboardButton.setText("Add at least one reminder");
    inlineKeyboardButton.setCallbackData("EMPTY_LIST");

    buttonList.add(inlineKeyboardButton);
    rowsInLine.add(buttonList);

    inlineKeyboardMarkup.setKeyboard(rowsInLine);
    return inlineKeyboardMarkup;
}

InlineKeyboardButton inlineKeyboardButton = new InlineKeyboardButton();
List<InlineKeyboardButton> buttonList = new ArrayList<>();

inlineKeyboardButton.setText("All groups");
inlineKeyboardButton.setCallbackData("ALL");

buttonList.add(inlineKeyboardButton);
rowsInLine.add(buttonList);

for (String g : group) {
    inlineKeyboardButton = new InlineKeyboardButton();
    buttonList = new ArrayList<>();

    inlineKeyboardButton.setText(g);
    inlineKeyboardButton.setCallbackData(g);

    buttonList.add(inlineKeyboardButton);
```

```
            rowsInLine.add(buttonList);
        }

        inlineKeyboardMarkup.setKeyboard(rowsInLine);
        return inlineKeyboardMarkup;
    }

}
```

```
package ua.onpu.service.impl;

import org.apache.commons.lang3.time.DateUtils;
import org.springframework.stereotype.Service;
import ua.onpu.service.DateParser;

import java.text.ParseException;
import java.util.Date;

@Service
public class DateParseImpl implements DateParser {
    @Override
    public Date parse(String date) throws ParseException {
        return DateUtils.parseDate(date,
                "yyyy-MM-dd HH:mm:ss", "yyyy-MM-dd HH:mm");
    }
}
```

```
package ua.onpu.repository;
```

```
import org.springframework.data.repository.CrudRepository;
import ua.onpu.entity.Assigment;
import ua.onpu.entity.Task;

import java.util.List;

public interface AssigmentRepository extends CrudRepository<Assigment, Long> {
    List<Assigment> findAllByUserChatId(Long chatId);

    Assigment findByTask(Task task);
}
```

```
package ua.onpu.repository;

import org.springframework.data.repository.CrudRepository;
import ua.onpu.entity.Task;

public interface TaskRepository extends CrudRepository<Task, Long> {
    Task findByTaskId(Long taskId);
}
```

```
package ua.onpu.repository;

import org.springframework.data.repository.CrudRepository;
import ua.onpu.entity.User;

import java.util.List;

public interface UserRepository extends CrudRepository<User, Long> {
```

```
    List<User> findAll();

}
```

```
package ua.onpu.process;

import org.telegram.telegrambots.meta.api.objects.CallbackQuery;
import org.telegram.telegrambots.meta.api.objects.Message;
import org.telegram.telegrambots.meta.api.objects.Update;

public interface Processor {

    void executeMessage(Message message);

    void executeCallBackQuery(CallbackQuery callbackQuery);

    default void process(Update update) {
        if (update.hasMessage()) {
            executeMessage(update.getMessage());
        } else if (update.hasCallbackQuery()) {
            executeCallBackQuery(update.getCallbackQuery());
        }
    }
}
```

```
package ua.onpu.process;
```

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import org.telegram.telegrambots.meta.api.objects.CallbackQuery;
import org.telegram.telegrambots.meta.api.objects.Message;
import ua.onpu.handler.CallbackQuerryHandler;
import ua.onpu.handler.MessageHandler;

@Component
public class ProcessorImpl implements Processor {

    private final CallbackQuerryHandler callBackQuerryHandler;
    private final MessageHandler messageHandler;

    @Autowired
    public ProcessorImpl(CallbackQuerryHandler callBackQuerryHandler, MessageHandler messageHandler) {
        this.callBackQuerryHandler = callBackQuerryHandler;
        this.messageHandler = messageHandler;
    }

    @Override
    public void executeMessage(Message message) {
        messageHandler.choose(message);
    }

    @Override
    public void executeCallBackQuery(CallbackQuery callbackQuery) {
        callBackQuerryHandler.choose(callbackQuery);
```

```
    }
}
```

```
package ua.onpu.handler;

public interface Handler<T> {

    void choose(T t);

}
```

```
package ua.onpu.handler;

import com.vdurmont.emoji.EmojiParser;
import lombok.extern.log4j.Log4j;
import org.springframework.stereotype.Component;
import org.telegram.telegrambots.meta.api.methods.ParseMode;
import org.telegram.telegrambots.meta.api.methods.send.SendMessage;
import org.telegram.telegrambots.meta.api.objects.Message;
import org.telegram.telegrambots.meta.api.objects.replykeyboard.ReplyKeyboardRemove;
import ua.onpu.cache.Cache;
import ua.onpu.domain.Statements;
import ua.onpu.service.KeyboardService;
import ua.onpu.service.MessageService;
import ua.onpu.entity.DataBaseControl;
import ua.onpu.entity.User;

import java.text.ParseException;
```

```
@Component
@Log4j
public class MessageHandler implements Handler<Message> {

    private final Cache<User> cache;
    private final MessageService messageService;
    private final DataBaseControl dataBaseControl;
    private final KeyboardService keyboardService;


    public MessageHandler(Cache<User> cache, MessageService messageService, DataBaseControl dataBaseControl,
KeyboardService keyboardService) {
        this.cache = cache;
        this.messageService = messageService;
        this.dataBaseControl = dataBaseControl;
        this.keyboardService = keyboardService;

    }

    @Override
    public void choose(Message message) {
        User user = cache.findBy(message.getChatId());

        if (user != null) {

            if (message.getText().equals("/start")) {
                user.setState(Statements.START);
```

```
    messageService.sendMessage(SendMessage.builder()
            .chatId(user.getChatId())
            .text(EmojiParser.parseToUnicode("Hello, i'm ReminderBot. Please choose the option :blush:"))
            .replyMarkup(keyboardService.startStateKeyboard())
            .build());
}

switch (user.getState()) {
    case START:
    case VIEW:
    case VIEW_PROCESSING:
        handleStartKeyboardCommand(message, user);
        break;
    case CREATE:
        messageService.sendMessage(SendMessage.builder()
                .chatId(user.getChatId())
                .text("Now write your reminder")
                .replyMarkup(new ReplyKeyboardRemove(true))
                .build());

        user.setGroupToCreate(message.getText());
        user.setState(Statements.CREATE_CONFIRMATION);
        break;
    case CREATE_CONFIRMATION:
        dataBaseControl.makeRemind(message, user.getGroupToCreate());

        messageService.sendMessage(SendMessage.builder()
```

```
                .text(EmojiParser.parseToUnicode("Done! :blush:"))
                .chatId(user.getChatId())
                .replyMarkup(keyboardService.startStateKeyboard())
                .build());

            user.setState(Statements.START);
            break;
        case EDIT_PROCESSING:
            dataBaseControl.updateTask(user.getTaskIdToManipulate(), message.getText());

            messageService.sendMessage(SendMessage.builder()
                .chatId(user.getChatId())
                .text("Your current reminder list")
                .replyMarkup(keyboardService.viewProcessingStateKeyboard(dataBaseControl.showList(user.getChatId(),
user.getGroupToShow())))
                .build());

            user.setState(Statements.VIEW);
            break;
        case DEADLINE_PARSING:
            try {

                dataBaseControl.createDeadline(user, message.getText());

                messageService.sendMessage(SendMessage.builder()
                    .chatId(user.getChatId())
                    .text("Done!")
                    .replyMarkup(keyboardService.startStateKeyboard())
```

```
                        .build());

                handleStartKeyboardCommand(message, user);
                user.setState(Statements.START);
            } catch (ParseException e) {
                user.setState(Statements.VIEW);
                messageService.sendMessage(SendMessage.builder()
                        .chatId(user.getChatId())
                        .text("Cant parse the date, please try again")
                        .replyMarkup(keyboardService.viewProcessingStateKeyboard(dataBaseControl.showList(user.getChatId(),
user.getGroupToShow())))
                        .build());


            }
            break;
        }
    } else {
        user = generateUserFromMessage(message);
        cache.add(user);

        messageService.sendMessage(SendMessage.builder()
                .chatId(user.getChatId())
                .text(EmojiParser.parseToUnicode("Hello, i'm ReminderBot. Please choose the option :blush:"))
                .replyMarkup(keyboardService.startStateKeyboard())
                .build());
    }
}
```

```
private void handleStartKeyboardCommand(Message message, User user) {
   switch (message.getText()) {
      case "Reminder list":
         messageService.sendMessage(SendMessage.builder()
               .chatId(user.getChatId())
               .text("Choose group to view it\nAvailable groups:\n")
               .parseMode(ParseMode.MARKDOWN)
               .replyMarkup(keyboardService.viewStateKeyboard(dataBaseControl.groupList(user.getChatId())))
               .build());

         user.setState(Statements.VIEW_PROCESSING);
         break;
      case "Make a new reminder":
         user.setState(Statements.CREATE);

         messageService.sendMessage(SendMessage.builder()
               .text(EmojiParser.parseToUnicode("Before create task write group for the task, please" +
                     "\nIf you don't want to write it, just copy it - `GENERAL`"))
               .chatId(user.getChatId())
               .replyMarkup(new ReplyKeyboardRemove(true))
               .parseMode(ParseMode.MARKDOWN)
               .build());
         break;
   }
}

private User generateUserFromMessage(Message message) {
```

```
        User user = new User();
        user.setUserName(message.getFrom().getUserName());
        user.setChatId(message.getChatId());
        user.setState(Statements.START);

        return user;
    }

}
```

```
package ua.onpu.handler;

import com.vdurmont.emoji.EmojiParser;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import org.telegram.telegrambots.meta.api.methods.send.SendMessage;
import org.telegram.telegrambots.meta.api.methods.updatingmessages.EditMessageText;
import org.telegram.telegrambots.meta.api.objects.CallbackQuery;
import org.telegram.telegrambots.meta.api.objects.replykeyboard.InlineKeyboardMarkup;
import org.telegram.telegrambots.meta.api.objects.replykeyboard.ReplyKeyboardRemove;
import ua.onpu.cache.Cache;
import ua.onpu.domain.Statements;
import ua.onpu.service.KeyboardService;
import ua.onpu.service.MessageService;
import ua.onpu.entity.DataBaseControl;
import ua.onpu.entity.User;

@Component
public class CallbackQuerryHandler implements Handler<CallbackQuery> {
```

```
    private final Cache<User> cache;
    private final MessageService messageService;
    private final DataBaseControl dataBaseControl;
    private final KeyboardService keyboardService;


    @Autowired
    public CallbackQuerryHandler(Cache<User> cache, MessageService messageService, DataBaseControl dataBaseControl,
KeyboardService keyboardService) {
        this.cache = cache;
        this.messageService = messageService;
        this.dataBaseControl = dataBaseControl;
        this.keyboardService = keyboardService;
    }

    @Override
    public void choose(CallbackQuery callbackQuery) {
        User user = cache.findBy(callbackQuery.getMessage().getChatId());

        if (callbackQuery.getData().matches("^\\d+$")) {
            user.setTaskIdToManipulate(callbackQuery.getData());
        } else if (dataBaseControl.groupList(user.getChatId()).contains(callbackQuery.getData()) ||
                callbackQuery.getData().equals("ALL")) {
            user.setGroupToShow(callbackQuery.getData());
        } else {
            user.setState(Statements.valueOf(callbackQuery.getData()));
        }
```

```
        switch (user.getState()) {
            case VIEW_PROCESSING:
                messageService.sendMessage(SendMessage.builder()
                        .chatId(user.getChatId())
                        .text("Your reminder list")
                        .replyMarkup(keyboardService.viewProcessingStateKeyboard(dataBaseControl.showList(user.getChatId(),
user.getGroupToShow())))
                        .build());

                user.setState(Statements.VIEW);
                break;
            case EDIT:
                messageService.sendMessage(EditMessageText.builder()
                        .chatId(user.getChatId())
                        .text("Select reminder that you want to edit")
                        .messageId(callbackQuery.getMessage().getMessageId())
                        .replyMarkup((InlineKeyboardMarkup)
keyboardService.viewProcessingStateKeyboardWithOutChangeButtons(dataBaseControl.showList(user.getChatId(),
user.getGroupToShow())))
                        .build());

                user.setState(Statements.EDIT_PROCESSING);
                break;
            case EDIT_PROCESSING:
                messageService.sendMessage(SendMessage.builder()
                        .chatId(user.getChatId())
                        .text("Write your updated reminder")
```

```
                .replyMarkup(new ReplyKeyboardRemove(true))
                .build());

            break;
        case DELETE:
            messageService.sendMessage(EditMessageText.builder()
                .chatId(user.getChatId())
                .text("Select reminder that you want to delete")
                .messageId(callbackQuery.getMessage().getMessageId())
                .replyMarkup((InlineKeyboardMarkup)
keyboardService.viewProcessingStateKeyboardWithOutChangeButtons(dataBaseControl.showList(user.getChatId(),
user.getGroupToShow())))
                .build());

            user.setState(Statements.DELETE_PROCESSING);
            break;
        case DELETE_PROCESSING:
            dataBaseControl.deleteTask(user.getTaskIdToManipulate());

            messageService.sendMessage(SendMessage.builder()
                .chatId(user.getChatId())
                .text("Your current reminder list")
                .replyMarkup(keyboardService.viewProcessingStateKeyboard(dataBaseControl.showList(user.getChatId(),
user.getGroupToShow())))
                .build());

            user.setState(Statements.VIEW);
            break;
```

```
        case EMPTY_LIST:
            user.setState(Statements.START);

            messageService.sendMessage(SendMessage.builder()
                    .chatId(user.getChatId())
                    .text(EmojiParser.parseToUnicode("Your reminder list is empty, add something"))
                    .replyMarkup(keyboardService.startStateKeyboard())
                    .build());

            break;
        case DEADLINE:
            messageService.sendMessage(EditMessageText.builder()
                    .chatId(user.getChatId())
                    .text(EmojiParser.parseToUnicode("Choose reminder to set deadline"))
                    .messageId(callbackQuery.getMessage().getMessageId())
                    .replyMarkup((InlineKeyboardMarkup)
keyboardService.viewProcessingStateKeyboardWithOutChangeButtons(dataBaseControl.showList(user.getChatId(),
user.getGroupToShow())))
                    .build());

            user.setState(Statements.DEADLINE_PARSING);
            break;
        case DEADLINE_PARSING:
            messageService.sendMessage(SendMessage.builder()
                    .chatId(user.getChatId())
                    .text("Write the date and time\n" +
                            "The following date formats are supported: \n'yyyy-MM-dd HH:mm:ss'\n'yyyy-MM-dd HH:mm'")
                    .replyMarkup(new ReplyKeyboardRemove(true))
```

```
                .build());
            break;
        case BACK:
            user.setState(Statements.START);

            messageService.sendMessage(SendMessage.builder()
                .chatId(user.getChatId())
                .text(EmojiParser.parseToUnicode("You have been returned to start. Please choose the option :blush:"))
                .replyMarkup(keyboardService.startStateKeyboard())
                .build());
            break;
        case CANCEL:
            messageService.sendMessage(EditMessageText.builder()
                .chatId(user.getChatId())
                .text("You have been returned ")
                .messageId(callbackQuery.getMessage().getMessageId())
                .replyMarkup((InlineKeyboardMarkup)
keyboardService.viewProcessingStateKeyboardWithOutChangeButtons(dataBaseControl.showList(user.getChatId(),
user.getGroupToShow())))
                .build());

            user.setState(Statements.VIEW);
        }

    }

}
```

```
package ua.onpu.entity;


import lombok.Getter;
import lombok.Setter;
import lombok.ToString;

import javax.persistence.*;

@Entity(name = "assigment")
@Getter
@Setter
@ToString
public class Assigment {
    @Id
    @GeneratedValue
    private long assigmentId;
    @ManyToOne
    @JoinColumn(name = "task_id", nullable = false)
    private Task task;
    @ManyToOne
    @JoinColumn(name = "user_id", nullable = false)
    private User user;


}
```

```
package ua.onpu.entity;
```

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.dao.DataAccessException;
import org.springframework.stereotype.Component;
import org.telegram.telegrambots.meta.api.objects.Message;
import ua.onpu.repository.AssigmentRepository;
import ua.onpu.repository.TaskRepository;
import ua.onpu.repository.UserRepository;
import ua.onpu.service.DateParser;

import java.text.ParseException;
import java.util.*;
import java.util.stream.Collectors;

@Component
public class DataBaseControl {

    private UserRepository userRepository;

    private TaskRepository taskRepository;

    private AssigmentRepository assigmentRepository;
    @Autowired
    private DateParser dateParser;

    @Autowired
    public void setUserRepository(UserRepository userRepository) {
        this.userRepository = userRepository;
    }
```

```
@Autowired
public void setTaskRepository(TaskRepository taskRepository) {
   this.taskRepository = taskRepository;
}

@Autowired
public void setAssigmentRepository(AssigmentRepository assigmentRepository) {
   this.assigmentRepository = assigmentRepository;
}

public Map<Long, User> getUsersMaps() {
   Map<Long, User> userMap = new HashMap<>();

   List<User> users = new ArrayList<>(userRepository.findAll());

   for (User user : users) {
      userMap.put(user.getChatId(), user);
   }

   return userMap;
}

public void setUsersMap(Map<Long, User> usersMap) {
   usersMap.forEach((aLong, user) -> userRepository.save(user));
}

public Long makeRemind(Message message, String group) throws DataAccessException {
```

```
    Task task = new Task();
    task.setTaskText(message.getText());
    task.setTaskGroup(group);
    taskRepository.save(task);

    Assigment assigment = new Assigment();

    userRepository.findById(message.getChatId())
            .ifPresent(u -> assigment.setUser(u));

    assigment.setTask(task);
    assigmentRepository.save(assigment);

    return task.getTaskId();
}

public List<Task> showList(Long chatId) throws DataAccessException {
    List<Assigment> assignments = assigmentRepository.findAllByUserChatId(chatId);
    List<Task> taskList = new ArrayList<>();
    for (Assigment a : assignments) {
        taskList.add(a.getTask());
    }

    return taskList;
}

public List<Task> showList(Long chatId, String group) throws DataAccessException {
    if (group.equals("ALL")) {
```

```
        return showList(chatId);
    }

    List<Assigment> assignments = assigmentRepository.findAllByUserChatId(chatId);
    List<Task> taskList = new ArrayList<>();

    for (Assigment a : assignments) {
        Task task = a.getTask();
        if (task.getTaskGroup() != null && task.getTaskGroup().equals(group)) {
            taskList.add(a.getTask());
        }
    }

    return taskList;
}

public void registerUser(User user) throws DataAccessException {
    if (userRepository.findById(user.getChatId()).isEmpty()) {
        userRepository.save(user);
    }
}

public void updateTask(String taskId, String text) throws DataAccessException {
    Task task = taskRepository.findByTaskId(Long.parseLong(taskId));
    task.setTaskText(text);

    taskRepository.save(task);
}
```

```
public void deleteTask(String taskId) throws DataAccessException {
    Task task = taskRepository.findByTaskId(Long.parseLong(taskId));
    Assigment assigment = assigmentRepository.findByTask(task);
    assigment.setUser(null);
    assigment.setTask(null);

    assigmentRepository.delete(assigment);
    taskRepository.delete(task);
}

public Set<String> groupList(Long chatId) {
    return assigmentRepository.findAllByUserChatId(chatId)
            .stream()
            .map(Assigment::getTask)
            .map(Task::getTaskGroup)
            .collect(Collectors.toSet());
}

public void createDeadline(User user, String dateInString) throws ParseException {
    Task task = taskRepository.findByTaskId(Long.valueOf(user.getTaskIdToManipulate()));

    Date date = dateParser.parse(dateInString);
    task.setTaskDeadline(date);
    taskRepository.save(task);
}

public List<Assigment> findByTaskTaskDeadlineIsNonNull() {
```

```
        List<Assigment> list = (List<Assigment>) assigmentRepository.findAll();

        return list.stream()
                .filter(a -> a.getTask().getTaskDeadline() != null)
                .collect(Collectors.toList());
    }

    public void completeDeadline(Task task) {
        Assigment a = assigmentRepository.findByTask(task);

        assigmentRepository.delete(a);
        taskRepository.delete(task);
    }
}
```

```
package ua.onpu.entity;

import lombok.Getter;
import lombok.Setter;
import lombok.ToString;
import javax.persistence.*;
import java.util.Date;

@Entity(name = "task")
@Getter
@Setter
@ToString
public class Task {
    @Id
```

```
    @GeneratedValue
    private long taskId;
    @Column(nullable = false)
    private String taskText;
    private Date taskDeadline;
    private String taskGroup;

}
```

```
package ua.onpu.entity;

import lombok.Getter;
import lombok.Setter;
import lombok.ToString;
import ua.onpu.domain.Statements;
import javax.persistence.Entity;
import javax.persistence.Id;

@Entity(name = "usersData")
@Getter
@Setter
@ToString
public class User {
    @Id
    private long chatId;
    private String userName;
    private Statements state;
    private String taskIdToManipulate;
    private String groupToCreate;
```

```
    private String groupToShow;
}
```

```
package ua.onpu.domain;

public enum Statements {
    START,
    CREATE,
    CREATE_CONFIRMATION,
    VIEW,
    VIEW_PROCESSING,
    EDIT,
    EDIT_PROCESSING,
    DELETE_PROCESSING,
    DELETE,
    EMPTY_LIST,
    BACK,
    DEADLINE,
    DEADLINE_PARSING,
    CANCEL

}
```

```
package ua.onpu.config;

import lombok.Getter;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.PropertySource;
```

```
@Configuration
@Getter
@PropertySource("classpath:application.properties")
public class BotConfiguration {

    @Value("${bot.username}")
    private String botUsername;
    @Value("${bot.token}")
    private String botToken;

}
```

```
package ua.onpu.config;

import lombok.extern.log4j.Log4j;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.event.ContextRefreshedEvent;
import org.springframework.context.event.EventListener;
import org.springframework.stereotype.Component;
import org.telegram.telegrambots.meta.TelegramBotsApi;
import org.telegram.telegrambots.meta.exceptions.TelegramApiException;
import org.telegram.telegrambots.updatesreceivers.DefaultBotSession;
import ua.onpu.TelegramBot;

@Component
@Log4j
public class BotInit {
```

```
    private TelegramBot bot;

    @EventListener({ContextRefreshedEvent.class})
    public void init() throws TelegramApiException {
        TelegramBotsApi telegramBotsApi = new TelegramBotsApi(DefaultBotSession.class);

        try {
            telegramBotsApi.registerBot(bot);
        } catch (TelegramApiException e) {
            log.error("Error: " + e.getMessage());
        }
    }

    @Autowired
    public void setBot(TelegramBot bot) {
        this.bot = bot;
    }
}
```

```
package ua.onpu.cache;

import lombok.extern.log4j.Log4j;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.dao.DataAccessException;
import org.springframework.stereotype.Component;
import ua.onpu.entity.DataBaseControl;
import ua.onpu.entity.User;

import javax.annotation.PostConstruct;
```

```
import javax.annotation.PreDestroy;
import java.util.*;

@Component
@Log4j
public class UsersCache implements Cache<User> {

    private final Map<Long, User> users;

    private final DataBaseControl dataBaseControl;

    @Autowired
    public UsersCache(DataBaseControl dataBaseControl) {
        this.users = new HashMap<>();
        this.dataBaseControl = dataBaseControl;
    }

    @PostConstruct
    public void initUsers() {
        users.putAll(dataBaseControl.getUsersMaps());
    }

    @PreDestroy
    public void saveUsers() {

        users.forEach(((aLong, user) -> {
            user.setGroupToShow(null);
            user.setGroupToCreate(null);
```

```
        }));

        dataBaseControl.setUsersMap(users);
    }

    @Override
    public void add(User user) {
        users.put(user.getChatId(), user);

        try {
            dataBaseControl.registerUser(user);
        } catch (DataAccessException e) {
            log.error(e.getMessage());
        }
    }

    @Override
    public User findBy(Long id) {
        return users.get(id);
    }

    @Override
    public List<User> getAll() {
        return new ArrayList<>(users.values());
    }
}
```

```
package ua.onpu.cache;
```

```
import java.util.List;

public interface Cache<T> {
    void add(T t);

    T findBy(Long id);

    List<T> getAll();
}
```