Міністерство освіти і науки України

Національний університет «Одеська політехніка»

Інститут комп’ютерних систем

Кафедра інформаційних систем

Грабовська Маргарита Володимирівна, студентка групи АІ-221

ДИСЦИПЛІНА

Об’єктно-орієнтоване програмування

КУРСОВА РОБОТА

Розробка телеграм-бота для керування нагадуваннями

Спеціальність:

122 Комп’ютерні науки

Освітня програма:

Комп’ютерні науки

Керівник:

Годовиченко Микола Анатолійович, кандидат технічних наук, доцент

Одеса – 2024

# ЗМІСТ

1. Огляд систем-аналогів та технологій їх розробки ................................................................. 5
   1. Особливості використання сучасних технологій для управління справами та підвищення особистої

продуктивності............................................................................................................................................. 5

* 1. Огляд додатків для ведення заміток .......................................................................................................... 6
     1. Додаток Remember The Milk ................................................................................................................. 6
     2. Додаток ColorNote .................................................................................................................................. 8
  2. Формування вимог до основних функцій телеграм-бота ........................................................................ 9
  3. Огляд інформаційних технологій для розробки телеграм-бота ........................................................... 10
     1. Фреймворк Spring Framework .............................................................................................................. 10
     2. Архітектурні компоненти Java ............................................................................................................ 11
     3. Висновки до першого розділу ............................................................................................................. 13

1. Проектування телеграм-бота для керування нагадуваннями ........................................ 14
   1. Мета та задачі телеграм боту ................................................................................................................... 14
   2. Визначення функціональних вимог до телеграм бота ........................................................................... 15
   3. Формування користувацьких історій для телеграм бота ....................................................................... 16
   4. Проектування користувацького інтерфейсу телеграм бота .................................................................. 17
   5. Висновки до другого розділу ................................................................................................................... 19
2. Програмна реалізація телеграм-бота для керування нагадуваннями …….………... 20
   1. Структура програмного проекту .......................................................................................................... 20
   2. Діаграма класів ....................................................................................................................................... 21
   3. Керування вихідним кодом веб додатку .............................................................................................. 21
   4. Функціональне тестування розробленого веб-ресурсу телеграм-бота ............................................. 22
   5. Інструкція користувача веб-ресурсу .................................................................................................... 24
   6. Вихідний код веб-ресурсу ..................................................................................................................... 28
   7. Висновки до третього розділу ............................................................................................................... 28

ВИСНОВКИ .................................................................................................................................................. 29

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ ........................................................................................ 30

АНОТАЦІЯ

Моя курсова робота спрямована на створення Java-телеграм-бота, використовуючи фреймворк Spring Boot та паттерн розробки State. Цей бот призначений для зручного створення та управління користувацькими нагадуваннями чи нотатками на мобільних пристроях. Основною метою моєї

роботи є створення простого, ефективного та інтуїтивно зрозумілого інструменту,

що дозволить користувачам легко створювати, редагувати, видаляти та

організовувати свої нагадування. У ході розробки додатку я використовую сучасні технології та практики розробки.

ABSTRACT

My coursework is aimed at creating a Java Telegram bot using the Spring Boot framework and the State design pattern. This bot is designed for convenient creation and management of user reminders or notes on mobile devices. The main goal of my work is to create a simple, efficient, and intuitively understandable tool that allows users to easily create, edit, delete, and organize their reminders. During the development of the application, I utilize modern technologies and development practices.

ВСТУП

Останнім часом месенджери стали невід'ємною частиною нашого повсякденного життя, зросла потреба в ефективних і зручних інструментах для організації наших справ, ідей та нагадувань. Такими засобами стали телеграм-боти, які дозволяють нам швидко отримувати інформацію, виконувати різноманітні завдання та спростити наше щоденне життя.

З урахуванням цієї потреби, я вирішила зосередитися на розробці телеграмбота для організації та управління задачами, нагадуваннями та замітками. Мета цього проекту - створення інтуїтивно зрозумілого інструменту, доступного на платформі Telegram, який надасть користувачам зручний і швидкий доступ до організації завдань незалежно від місця та часу.

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

У світлі вищевикладених обставин, тема розробки телеграм-бота для організації завдань залишається актуальною. Такий бот дозволив би користувачам створювати, переглядати та керувати задачами, зберігаючи їх історію та поточний

стан для подальшого перегляду та використання.

Метою курсової роботи є розробка телеграм бота для управління задачами.

Для досягнення цієї мети необхідно виконати наступні задачі:

— оглянути 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 сучасних технологій для управління справами та підвищення особистої продуктивності

Використання технологій грає важливу роль у поліпшенні особистої продуктивності та організації справ. Технології надають можливість доступу до інформації, організації завдань і спрощення щоденних обов'язків. Ось кілька ключових аспектів використання технологій у цьому контексті:

— Різноманітність функцій: Телеграм боти можуть виконувати різні завдання, включаючи надання інформації, пошук, відправлення сповіщень, операції з даними та підтримку користувачів. Ці можливості можуть бути розширені за допомогою додаткових модулів та інтеграцій.

— Простота використання: Телеграм боти мають простий та зрозумілий інтерфейс, що робить їх доступними для широкого кола користувачів без спеціальних технічних навичок.

— Інтерактивність та зручність: Телеграм боти забезпечують інтерактивну взаємодію з користувачем через меню, кнопки, шаблонні повідомлення і т.д. Це спрощує спілкування та робить користування ботом зручним.

— Автоматизація та нагадування: Телеграм боти можуть автоматично відправляти нагадування, сповіщення або повідомлення відповідно до попередньо налаштованого розкладу або у відповідь на певні події, що допомагає ефективно керувати часом та планувати задачі.

— Мобільний доступ: Телеграм боти можна використовувати на мобільних пристроях, що забезпечує постійний доступ до них незалежно від місця

перебування. Це дозволяє легко спілкуватися з ботом і керувати завданнями прямо зі смартфона або планшета.

Все враховуючи, телеграм боти стають зручним інструментом для виконання завдань, комунікації та отримання інформації через мобільні пристрої, що спрощує процес взаємодії з користувачем.

1.2 Огляд додатків для ведення заміток

Для створення телеграм бота потрібно визначити, які функції він має мати. Це можна зробити, проаналізувавши інші додатки для ведення заміток та вивчивши їх особливості, переваги та недоліки. Цей аналіз допоможе краще зрозуміти потреби користувачів у цій області. Після пошуку в Інтернеті були знайдені найбільш популярні додатки, які можуть бути використані як аналоги для даної роботи:

— додаток TickTick

— додаток Structured

Далі необхідно докладно проаналізувати зазначені додатки, щоб встановити, як вони працюють, і виявити як позитивні, так і негативні аспекти їх використання. Цей процес дозволить краще зрозуміти, які функції повинен мати телеграм-бот для створення заміток, і відповісти на потреби користувачів у цій сфері. Цей аналіз допоможе зрозуміти, які особливості і функції слід включити в телеграм бота для ефективного ведення заміток та відповіді на потреби користувачів у цій області.

1.2.1 Додаток TickTick

TickTick - це додаток, який вирізняється своєю гнучкістю та різноманітними можливостями. Він дозволяє створювати різні списки завдань, налаштовувати нагадування та пріоритети, а функція спільної роботи дозволяє спільно працювати над проектами з іншими користувачами. Інтеграція з календарем дозволяє зручно планувати час і завдання.

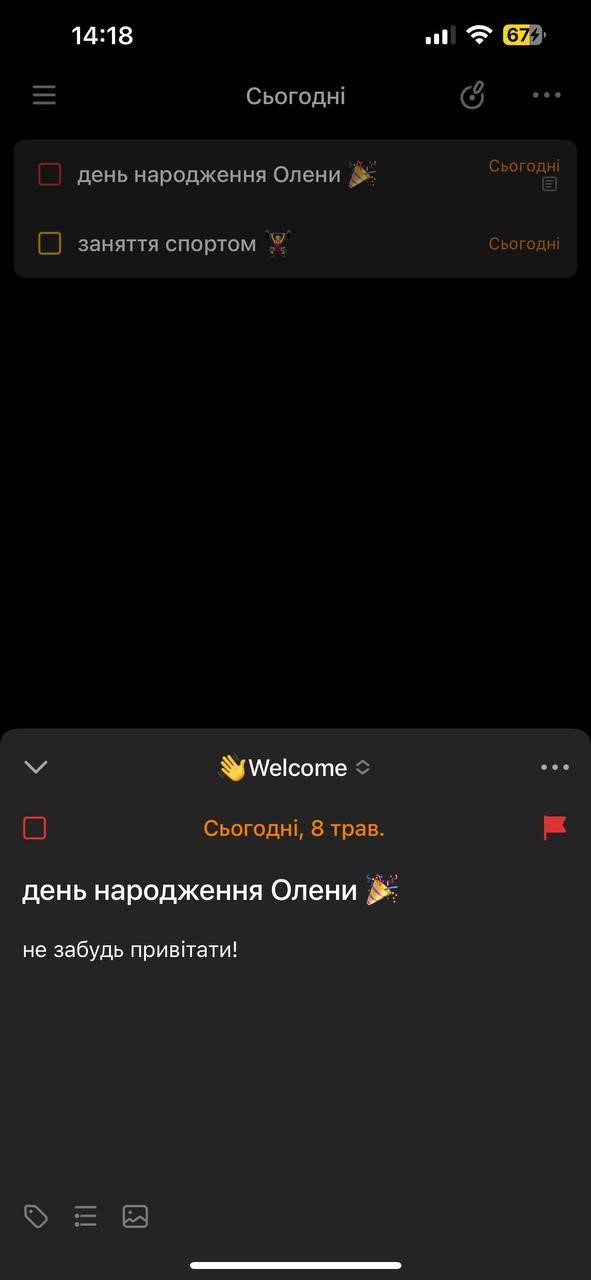
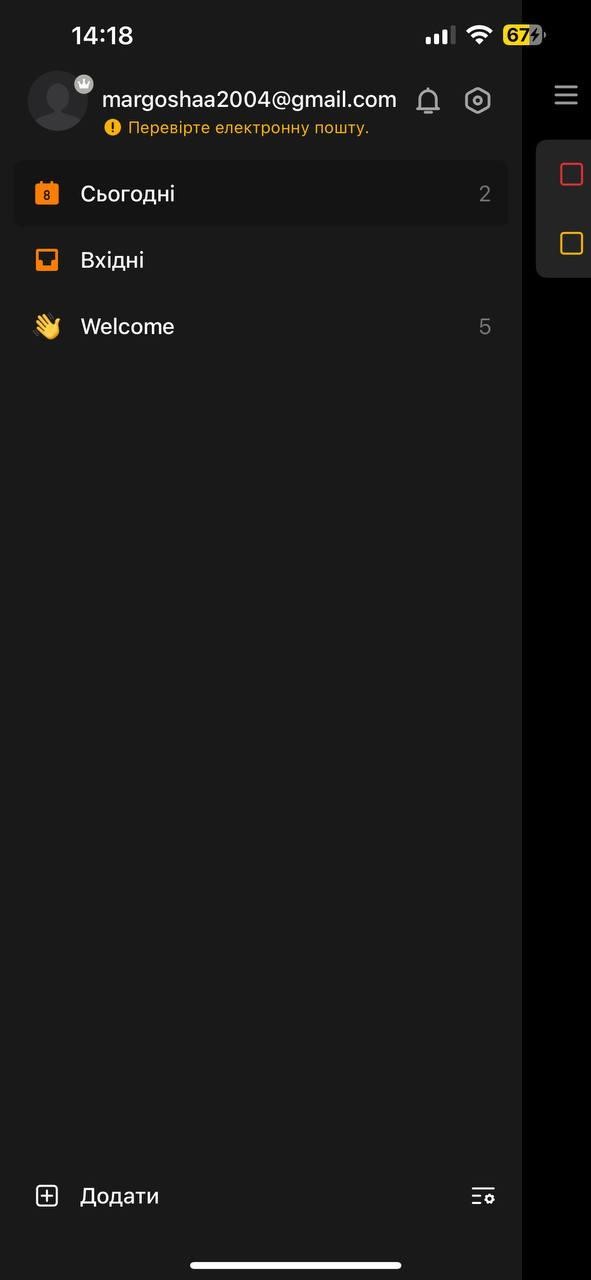
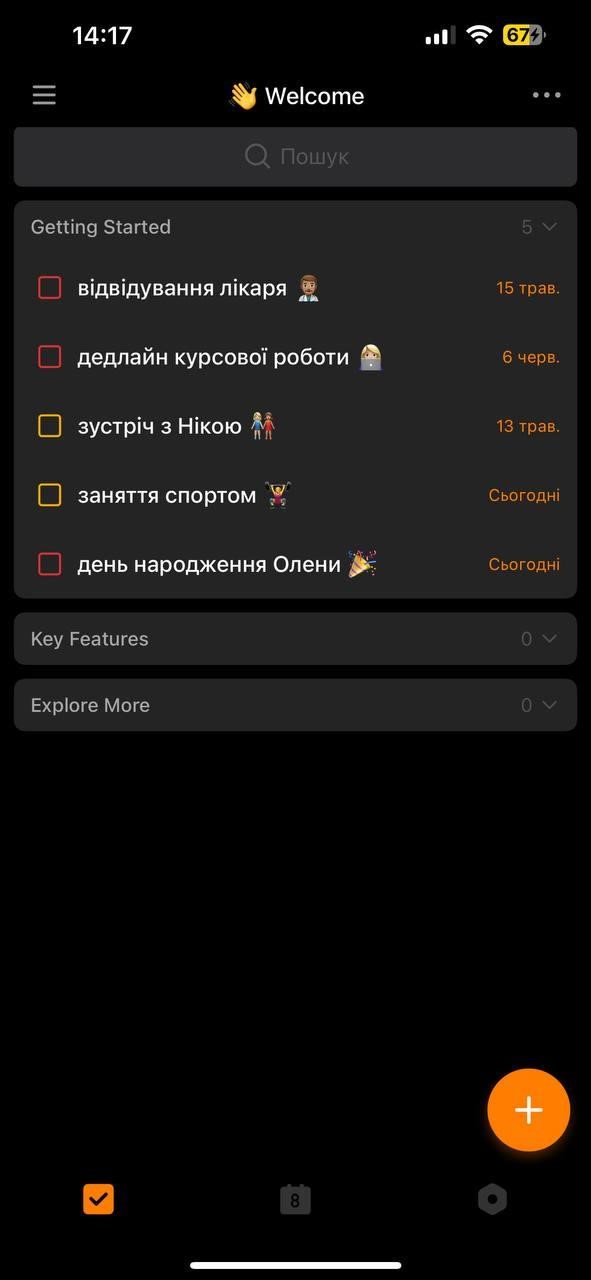


Рисунок 1.2 – Знімки екрану додатку TickTick

Основний функціонал додатку:

— Створення списків завдань: Користувач може створювати різні списки завдань для організації робочих, особистих або проектних завдань.

— Нагадування: Можливість налаштовувати нагадування для завдань з використанням різних параметрів, таких як час, дата та місце.

— Календар: Інтеграція з календарем, що дозволяє відображати завдання та події разом для зручного планування.

— Спільна робота: Можливість ділитися списками завдань з іншими користувачами та спільно працювати над проектами.

— Планування: Функції планування та управління пріоритетами для ефективного виконання завдань вчасно.

— Крос-платформеність: Доступ до додатку на різних пристроях, включаючи смартфони, планшети та комп'ютери.

Платна версія додатку має додатковий функціонал:

— Більше нагадувань: Дозволяє налаштовувати необмежену кількість нагадувань для завдань і подій.

— Шаблони завдань: Дозволяє створювати шаблони для часто використовуваних завдань або списків.

— Збереження задач в історії: Можливість перегляду історії завдань, включаючи виконані та видалені.

— Розширені налаштування підкатегорій: Дозволяє створювати складніші системи категорій та підкатегорій для організації завдань.

— Більше кольорових тем: Додаткові опції для налаштування зовнішнього вигляду додатку.

1.2.2 Додаток Structured

Structured - це додаток, який є універсальним планувальником дня, який об'єднує календар, список завдань і відстеження звичок в одному місці. Візуальна лінія часу є основою додатка, де ваші зустрічі, події та завдання об'єднуються. Ви можете швидко створювати завдання, встановлювати терміни виконання, переглядати і переносити їх, а також налаштовувати день відповідно до своїх потреб.

Основний функціонал додатку Structured включає:

— Об'єднання календаря, списку завдань і відстеження звичок: Вам не потрібно використовувати кілька окремих додатків - Structured об'єднує ці функції в одному місці.

— Візуальна лінія часу: Основа додатка, де ви можете бачити всі ваші зустрічі, події та завдання в одному місці.

— Створення, налаштування та відстеження завдань: Швидко створюйте завдання, встановлюйте терміни виконання та відстежуйте їх виконання.

— Інтеграція з календарем і списками завдань: Зручна інтеграція з існуючими календарями та списками завдань.

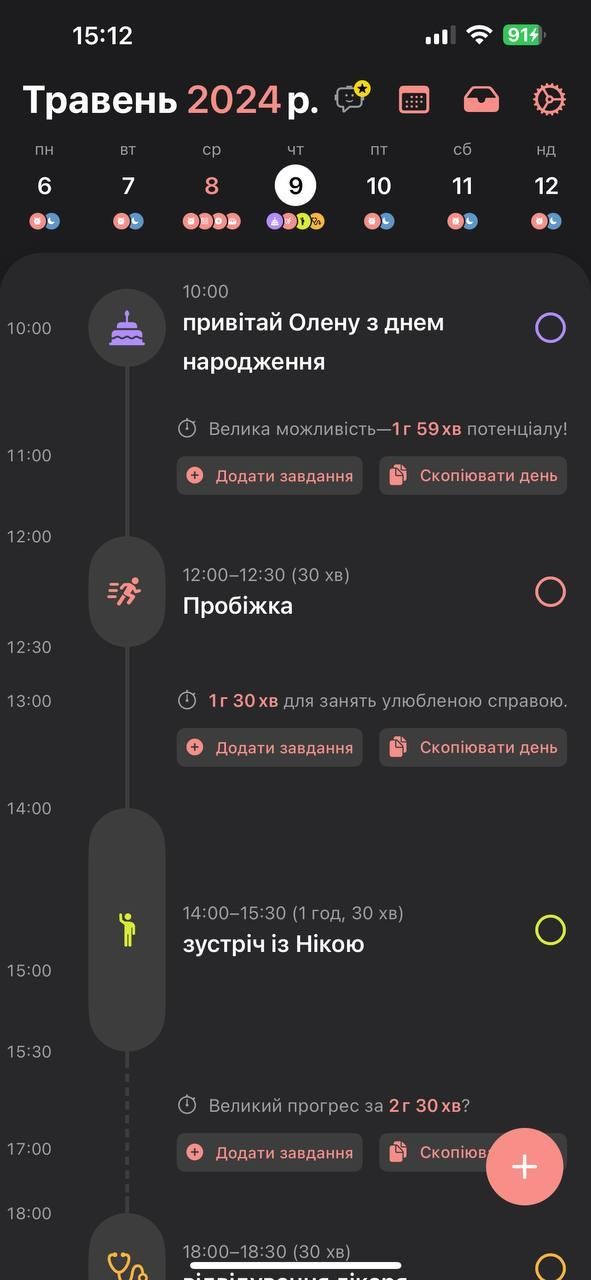
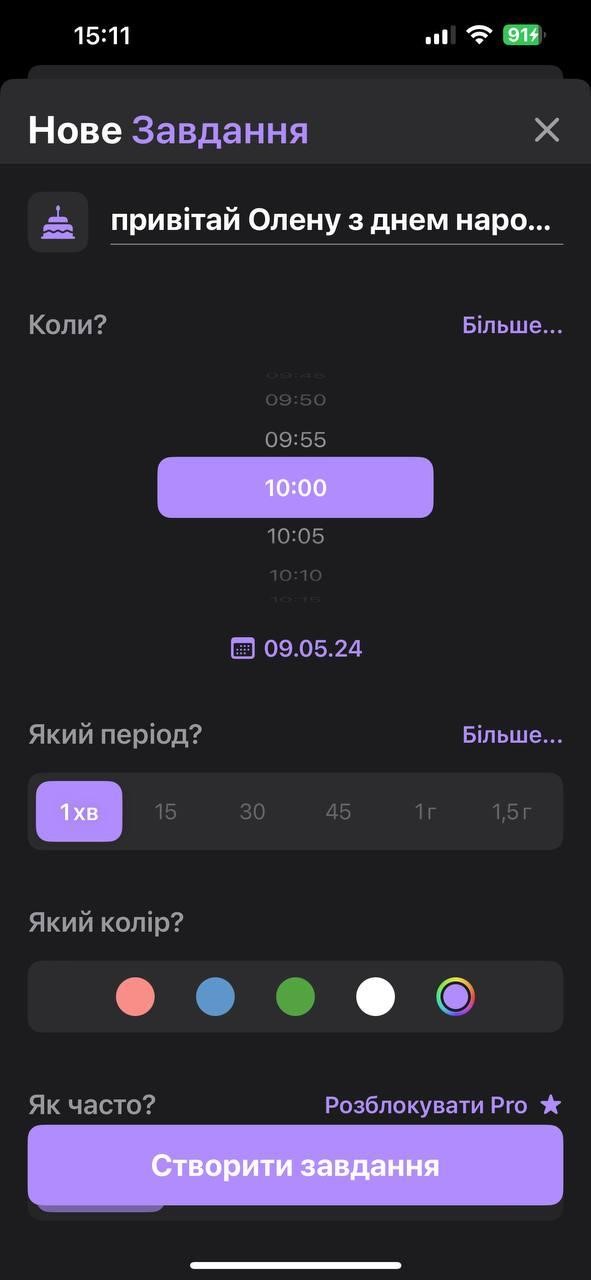
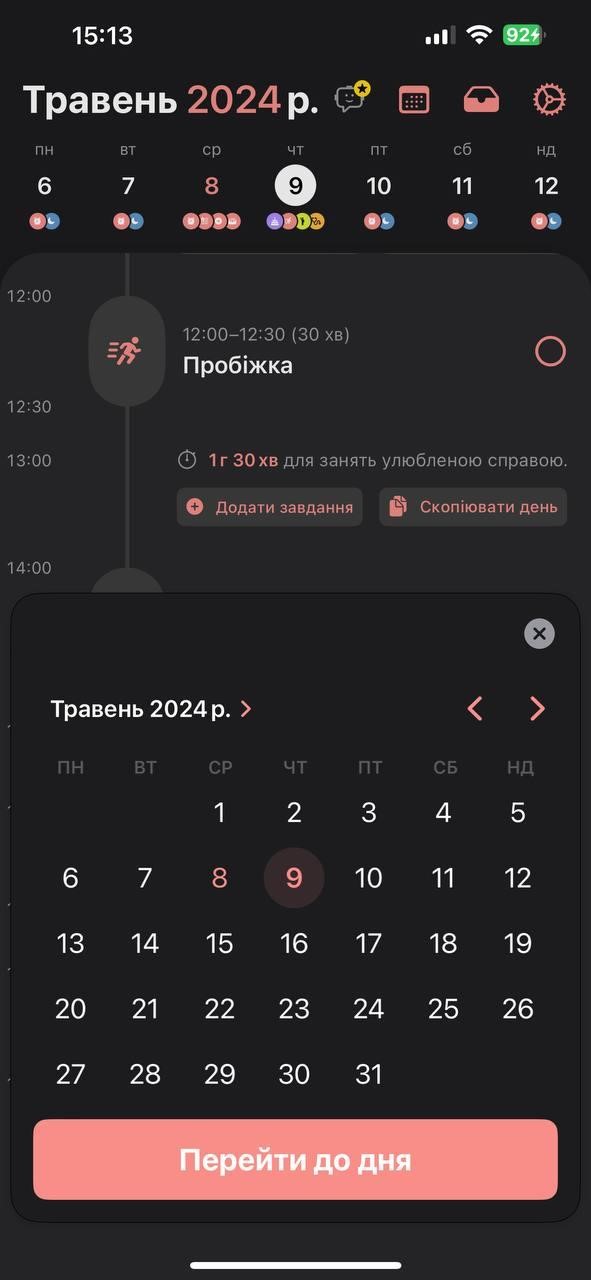


Рисунок 1.3 – Знімки екрану додатку Structured

1.3 Формування вимог до основних функцій телеграмм-бота

В результаті аналізу додатків для ведення справ та нотаток були визначені ключові функції для телеграм боту.

Основний функціонал телеграмм-боту для ведення заміток може включати наступне:

— Створення заміток: Користувач може створювати нові замітки, вводячи текст. Додаток повинен надати простий та зручний інтерфейс для цього процесу.

— Безпека даних: Додаток може пропонувати захист заміток за допомогою пароля чи інших методів аутентифікації для забезпечення конфіденційності та безпеки даних.

— Пошук та фільтрація: Користувач може шукати замітки за ключовими словами або фільтрувати їх за різними критеріями, такими як дата створення, категорія тощо.

— Редагування та форматування: Користувач може редагувати вміст заміток, включаючи видалення, редагування або переміщення тексту. Також може бути доступне форматування тексту, таке як жирний, курсив, списки тощо.

— Категорії та мітки: Користувач може створювати категорії або мітки для заміток, що допомагає їх організувати за темами чи проектами. Це полегшує пошук та забезпечує більшу структурованість.

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.

Переваги:

— Інверсія керування (IoC): Використання цього принципу дозволяє

розробникам зосередитися на бізнес-логіці, розділяючи її від деталей конфігурації

та управління об'єктами. Це робить код більш зрозумілим та легко тестовим.

— Spring Boot: Цей компонент спрощує розгортання додатків, надаючи конфігурацію за замовчуванням та автоматичну налаштування. Це дозволяє швидко створювати готові до використання додатки з меншими зусиллями щодо налаштування.

— Spring Data JPA: Використання JPA спрощує роботу з базами даних, а Spring Data JPA робить це ще більш простим. Він автоматично генерує SQL-запити з методів репозиторіїв, що дозволяє легко виконувати операції з базою даних.

— Легкість використання: Spring пропонує простий та інтуїтивний підхід до розробки завдяки своїм концепціям та шаблонам програмування. Це дозволяє розробникам швидко створювати та управляти додатками.

Недоліки:

— Конфігурація: Складні або великі додатки можуть потребувати складних налаштувань та управління залежностями. Хоча Spring надає багато інструментів

для управління конфігурацією, правильне налаштування може вимагати додаткових зусиль.

— Навчання: Спочатку вивчення Spring може зайняти час, оскільки він має досить великий обсяг функціоналу та концепцій. Для новачків це може бути викликом, але з досвідом цей недолік зникає.

— Вагомість: За всю свою потужність та гнучкість, деякі компоненти Spring можуть впливати на продуктивність додатків через більше використання ресурсів.

Це може бути проблемою у великих або високонавантажених додатках.

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

Архітектурні компоненти є ключовими елементами у розробці додатків на Java, оскільки вони сприяють структурованості, модульності, перевикористанню та тестуванню коду. Використання цих компонентів має численні переваги:

— Легке утримання: Архітектурні компоненти пропонують структуровану організацію коду, що полегшує утримання проекту та зміни.

— Тестуваність: Вони сприяють створенню тестів для перевірки функціональності та надійності додатка, оскільки розділяють бізнес-логіку від інтерфейсу.

— Масштабованість: Вони дозволяють ефективно масштабувати додатки, допомагаючи розділити бізнес-логіку від інтерфейсу користувача та керувати даними та станом додатка.

— Спільна розробка: Архітектурні компоненти сприяють зручній інтеграції та спільній роботі в командному середовищі.

— Розподіл обов'язків: Архітектурні компоненти, такі як MVVM або MVP, чітко розподіляють обов'язки між різними частинами додатка, полегшуючи розуміння та утримання коду.

— Перевикористання: Вони дозволяють виділити загальну функціональність у відокремлені компоненти, що можна використовувати у різних частинах додатка або навіть в інших проектах.

Використання цих компонентів в Java допомагає покращити якість, швидкість розробки та легкість утримання додатків, роблячи їх важливою частиною процесу розробки.

1.5 Висновки до першого розділу

У першому розділі курсової роботи було розглянуто процес розробки телеграм-бота, що є популярним рішенням для автоматизації різних завдань та комунікації у месенджері Telegram. Фреймворк Spring був обраний як інструмент для створення бота, оскільки він надає широкий набір інструментів для розробки та інтеграції з Telegram API.

Також у цьому розділі було проведено аналіз впливу технологій на ведення справ та особисту продуктивність. Виявлено, що використання сучасних технологій може значно полегшити організацію робочих процесів та підвищити ефективність у веденні особистих справ.

Крім того, був проведений огляд існуючих додатків-аналогів для ведення заміток, що дозволило визначити їхні переваги та недоліки. На основі цього огляду були сформульовані вимоги до власного телеграм-бота для ведення заміток, які дозволили визначити необхідний функціонал та можливості.

У цьому розділі також було проведено огляд інформаційних технологій, які використовуються для розробки телеграм-бота. Вибір фреймворку Spring був обгрунтований його зручністю та ефективністю для цієї конкретної задачі.

Загалом, перший розділ курсової роботи надав загальний огляд процесу розробки телеграм-бота, вказав на важливість використання сучасних технологій для підвищення ефективності роботи та визначив основні кроки для подальшої реалізації проекту.

2 ПРОЕКТУВАННЯ ТЕЛЕГРАМ-БОТА ДЛЯ КЕРУВАННЯ НАГАДУВАННЯМИ

2.1 Мета та задачі телеграм боту

Мета телеграм-бота для ведення заміток полягає в створенні зручного та організованого інструменту для користувачів, щоб вони могли легко зберігати, організовувати та керувати своїми записами. Основною метою додатку є спрощення процесу ведення особистих або професійних записів, щоб користувачі могли швидко і легко зберігати важливу інформацію та нагадування.

Задачі телеграм-бота для ведення заміток включають:

— Створення нових заміток: надання можливості користувачам швидко створювати нові записи заміток з використанням тексту.

— Редагування та видалення заміток: дозвіл на зміну вмісту заміток, редагування або видалення їх в разі потреби.

— Організація заміток: надання можливості створення категорій або міток для заміток, щоб користувачі могли легко групувати та організовувати свої записи.

— Сповіщення та нагадування: надання можливості встановлення нагадувань та сповіщень для заміток, щоб користувачі не пропустили важливі події або

завдання.

Такий телеграм-бот призначений для полегшення організації та керування інформацією для користувачів. Він забезпечує зручний спосіб зберігання і використання заміток, дозволяючи легко доступатися до них, редагувати, організовувати та забезпечувати їх безпеку. Такий мобільний додаток сприяє підвищенню 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 для телеграм-бота для ведення заміток є ключовим етапом у процесі його розробки. Ці вимоги визначають конкретні функції та можливості, якими повинен бути наділен додаток. Вони створюють чітку спрямованість розробки, допомагають уникнути непорозумінь та неоднозначностей.

При визначенні функціональних вимог увага приділяється потребам та вимогам користувачів. Це допомагає врахувати, які функції будуть найбільш корисними для користувачів та як їх реалізувати найефективніше.

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

Основним фактором телеграм-бота є користувач, який має доступ до всіх функцій та несе відповідальність за їх використання.

Для визначення користувацьких історій та нефункціональних вимог до телеграм-бота може бути створена діаграма сценаріїв використання, яка графічно показує взаємодію між користувачами та системою. Ця діаграма допомагає проаналізувати залежності та можливі варіанти використання системи.

2.3 Формування користувацьких історій для телеграм бота

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

Користувацька історія 1: Створення нової замітки

Я, як користувач, хочу створити нову замітку, щоб зафіксувати важливу інформацію. Тому я:

1. Відкриваю телеграм та заходжу в чат до бота.
2. Натискаю кнопку «Створити нову замітку».
3. Вводжу текстове повідомлення або нотатку.
4. Отримую підтвердження, що нова замітка збережена в додатку та з'явилася у списку заміток.

Користувацька історія 2: Редагування замітки

Я, як користувач, хочу редагувати вміст замітки, щоб вносити зміни або доповнення до інформації. Тому я:

1. Відкриваю конкретну замітку.
2. Натискаю на кнопку «Редагувати» або вибираю опцію редагування.
3. Вношу необхідні зміни або доповнення до тексту замітки.

Користувацька історія 3: Видалення замітки

Я, як користувач, хочу видалити замітки, які більше не потрібні, для підтримки чистоти та організованості списку заміток. Тому я:

1. Відкриваю конкретну замітку або виділяю багато заміток у списку.
2. Натискаю кнопку «Видалити» або вибираю опцію видалення.
3. Підтверджую видалення, і замітки видаляються з додатку та зникають зі списку заміток.

2.4 Проектування користувацького інтерфейсу телеграм бота

Для розробки користувацького інтерфейсу телеграм-бота, який відповідає вимогам до веб-ресурсу, потрібно спочатку розробити макети, що включають функціональні та нефункціональні вимоги. Це означає створення макетів для кожного елемента інтерфейсу, визначення сценаріїв взаємодії з користувачем, складання опису кожного макету та визначення їхньої ролі у досягненні мети.

На малюнку 2.4 показано макет головного вікна телеграм-бота. Головне вікно містить два основних елементи: меню та кнопки. Меню включає в себе випадаючий список команд, які користувач може викликати. Кнопковий елемент містить кнопки

з відповідними назвами функцій.

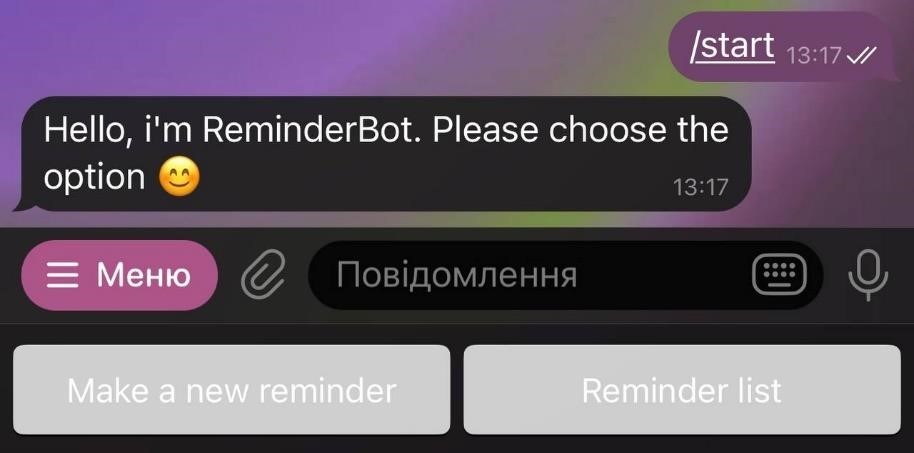


Рисунок 2.4 – Макет сторінки головного вікна

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.

Макет повинен мати привабливий дизайн і забезпечувати користувачам

легку навігацію і доступ до усієї необхідної інформації про товар.

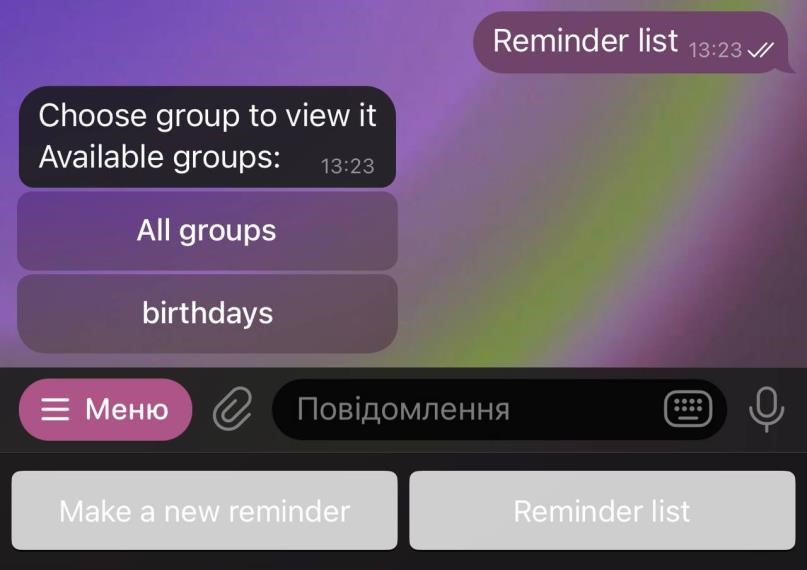


Рисунок 2.4 – Макет сторінки

2.5 Висновки до другого розділу

У другому розділі курсової роботи було зосереджено на проектуванні телеграм бота. Ця робота розпочалася з визначення цілей, цільової аудиторії та основних функціональних та нефункціональних вимог. Було використано ці дані для розробки діаграми сценаріїв використання, щоб краще зрозуміти, як користувачі будуть взаємодіяти з ботом.

Дана робота також включала розробку макетів ключових вікон, щоб уявити, як користувачі будуть використовувати даного бота.

В цілому, другий розділ дозволив нам детально розглянути процес проектування бота, визначити його ключові характеристики та підготувати підґрунтя для подальшої реалізації.

1. ПРОГРАМНА РЕАЛІЗАЦІЯ ТЕЛЕГРАМ БОТА ДЛЯ КЕРУВАННЯ НАГАДУВАННЯМИ

3.1 Структура програмного проекту

Цей проект має різноманітні компоненти, кожен з яких виконує свою функцію. Почнемо з компонента сервісів. Сервіси відповідають за надання певних функцій користувачам або іншим програмним системам. Вони можуть включати в себе широкий спектр операцій залежно від їхнього призначення та специфікацій. У рамках проекту були створені наступні сервіси:

— Сервіс для парсингу дати (DataParser.java && DataParserImpl.java) - він відповідає за перетворення дати у відповідний формат.

— Сервіс для генерації клавіатур (KeyboardService.java &&

KeyboardServiceImpl.java) - він відповідає за створення клавіатур для відповідних станів програми.

— Сервіс для відправлення повідомлень (MessageService.java && MessageServiceImpl.java) - він відповідає за передачу повідомлень користувачам.

— Сервіс для нагадувань (RemindService.java && RemindServiceImpl.java) - він відповідає за надсилання користувачам нагадувань про завдання.

Компонент сервісів включає в себе бізнес-логіку та операції, пов'язані з даними. Сервіси можуть містити методи, які отримують дані з бази даних, змінюють їх та зберігають зміни.

Крім того, у проекті є такі компоненти:

— Репозиторії - вони відповідають за зберігання та отримання даних з бази даних. Вони можуть включати методи для збереження, оновлення та видалення даних з бази даних.

— Моделі - вони представляють сутності, які використовуються в системі. Моделі можуть містити поля, що відображають дані з бази даних, а також методи, які полегшують взаємодію з цими даними.

— Конфігураційні файли - вони відповідають за налаштування та конфігурування Spring Framework. Такі файли можуть включати параметри зв'язку з базою даних, налаштування безпеки та інші параметри, необхідні для коректної роботи системи.

— Обробники (handlers) - вони відповідають за обробку запитів, які виникають при натисканні на кнопку або іншого інтерактивного елемента, або за обробку текстових повідомлень в Telegram-боті. Вони перехоплюють ці запити і виконують певні дії, такі як обробка даних, що супроводжують запит, та надсилання відповідей користувачу.

3.2 Діаграма класів

Діаграма класів - це тип UML-діаграми, яка показує зв'язки між класами, інтерфейсами та їх структуру у системі або додатку. Вона надає візуальне представлення компонентів системи та їх взаємозв'язків. Наприклад, діаграма класів продуктової крамниці, що базується на фреймворку Spring, допомагає зрозуміти, як класи взаємодіють один з одним та як вони організовані у системі.

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 критично важливим для будь-якого програмного проекту, зокрема для веб-ресурсу продуктової крамниці.

Ця система дозволяє зберігати історію всіх змін, що внесені до веб-ресурсу, дозволяючи з легкістю повертатися до попередніх версій та відновлювати їх стан.

Також вона допомагає контролювати версії веб-ресурсу, спрощуючи процес роботи з ним та забезпечуючи його актуальність.

Основні метрики репозиторію веб-ресурсу наведені в таблиці 3.1.

Таблиця 3.1 – Метрики керування програмним кодом додатку

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

3.4 Функціональне тестування розробленого веб-ресурсу телеграмм бота

Функціональне тестування веб-ресурсів - важливий процес, який перевіряє правильність роботи різних функцій та можливостей ресурсу перед його введенням в експлуатацію. Це забезпечує коректну та надійну роботу веб-сайту. Для цього потрібно розробити протокол тестування - документ, що містить опис кроків, процедур та результатів тестування, з метою систематичного задокументування процесу та отриманих результатів. Це забезпечує об'єктивність та повторюваність результатів тестування.

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

TC1 Тест-кейс для додавання нового нагадування:

— відкрити телеграмм бот;

— обрати опцію «Make a new reminder»;

— ввести назву групи;

— ввести нагадування;

— натиснути кнопку «Enter»;

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

TC2 Тест-кейс для редагування існуючого нагадування:

— 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.

TC3 Тест-кейс для видалення нагадування:

— 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.

Результати тестування показують, що всі функціональні тести успішно пройдені. Це означає, що реальна робота веб-ресурсу відповідає очікуваній поведінці, яка визначена у функціональних вимогах.

Таблиця 3.2 – Протокол функціонального тестування веб-ресурсу

|  |  |  |  |
| --- | --- | --- | --- |
| Номер тесткейсу | 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

Щоб забезпечити комфортний користувацький досвід при використанні створеного веб-ресурсу, важливо підготувати інструкцію користувача. Це дозволить користувачам легко зорієнтуватись у функціоналі веб-ресурсу телеграмбота, та ефективно використовувати його.

На рисунку 3.1 зображена перша сторінка цього бота, на якій ви побачите ласкаве вітання та короткий огляд його можливостей. Бот розповість, що він створений для того, щоб допомогти вам залишатися організованим і пам'ятати всі ваші важливі справи та події. Щоб розпочати користування, вам потрібно буде натиснути кнопку "Почати".



Рисунок 3.1 – Початок роботи с ботом

На початковому етапі взаємодії з ботом, зображеному на рисунку 3.2, потрібно відправити повідомлення "/start" в чат або вибрати цю команду із меню, що розташоване зліва.

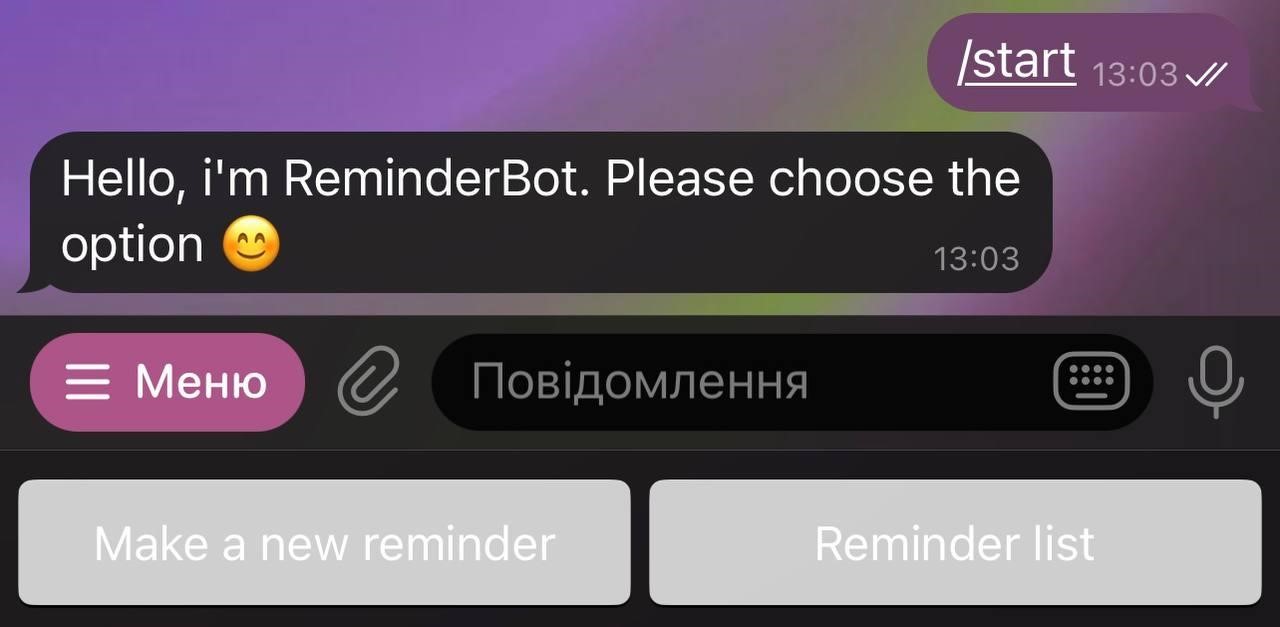


Рисунок 3.2 – Початок роботи с ботом

На рисунку 3.3 ми бачимо відображення після натискання кнопки "Make a new reminder". Відкривається розділ для створення нового нагадування. Тут ви вітаєтеся повідомленням, що вам доступна можливість обрати конкретну групу для розміщення вашого нового нагадування, що є дуже зручним і організованим спосібом керувати своїми завданнями та планами. Коли користувач може розподілити свої нагадування на різні папки, він зможе швидше знаходити та відстежувати свої завдання відповідно до їхньої тематики, пріоритету або інших критеріїв. Наприклад, папки можуть бути розділеними на робочі завдання, особисті справи, плани на вихідні, нагадування про важливі події тощо. Це дозволяє керувати завданнями більш ефективно, організовано та структуровано, що полегшує планування та виконання різних обов'язків у повсякденному житті.

Якщо ви не хочете обирати якусь конкретну папку, то можете просто залишити нагадування в загальному списку, обозначивши "General". Після вибору контексту для нагадування ви переходите до створення самого нагадування.

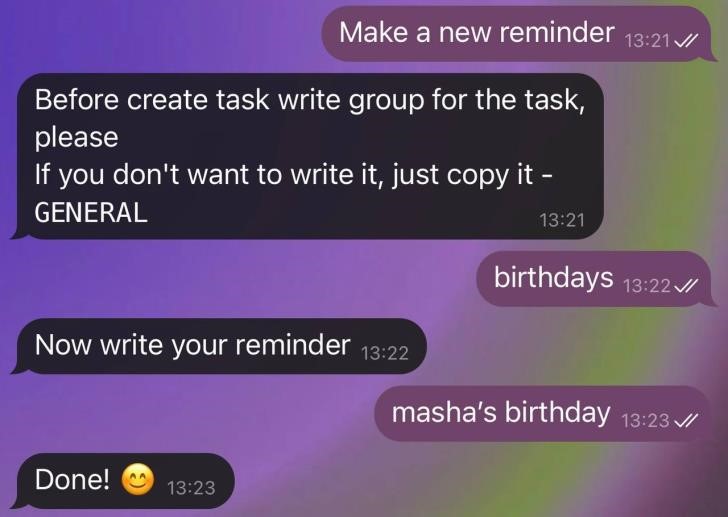


Рисунок 3.3 – Додання нагадування

Переглядаючи рисунок 3.4, ми бачимо перший крок, який потрібно виконати для перегляду завдань. Щоб розпочати, потрібно натиснути кнопку "Reminder List" і обрати потрібну групу, натиснувши на відповідну кнопку.

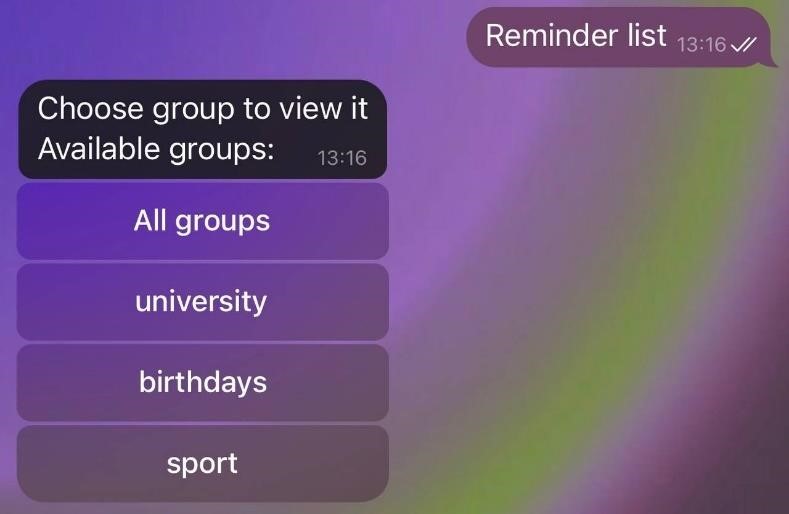


Рисунок 3.4 – Перегляд груп

На рисунку 3.5 відображається список завдань після вибору групи. Під кожним завданням відображаються доступні опції дій з цим завданням.

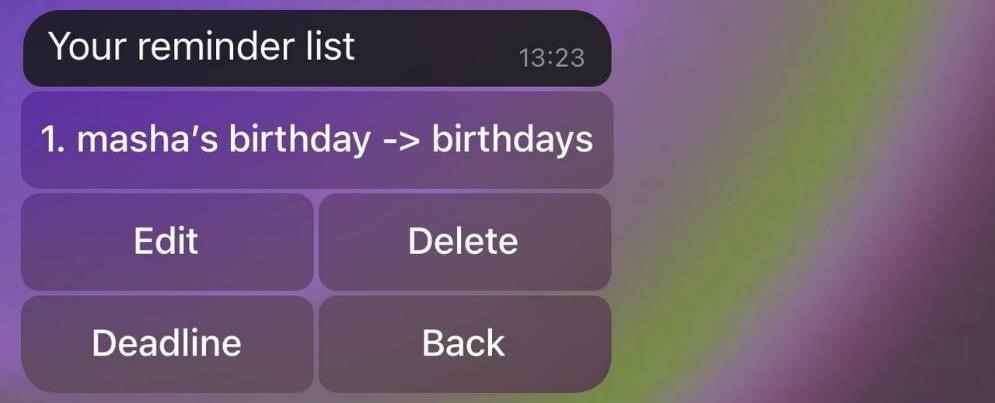


Рисунок 3.5 – Перегляд нагадувань

На рисунку 3.6 показано результат натискання кнопки "Edit". Після цього потрібно вибрати конкретне завдання, яке ми хочемо оновити, та ввести оновлену інформацію про завдання.

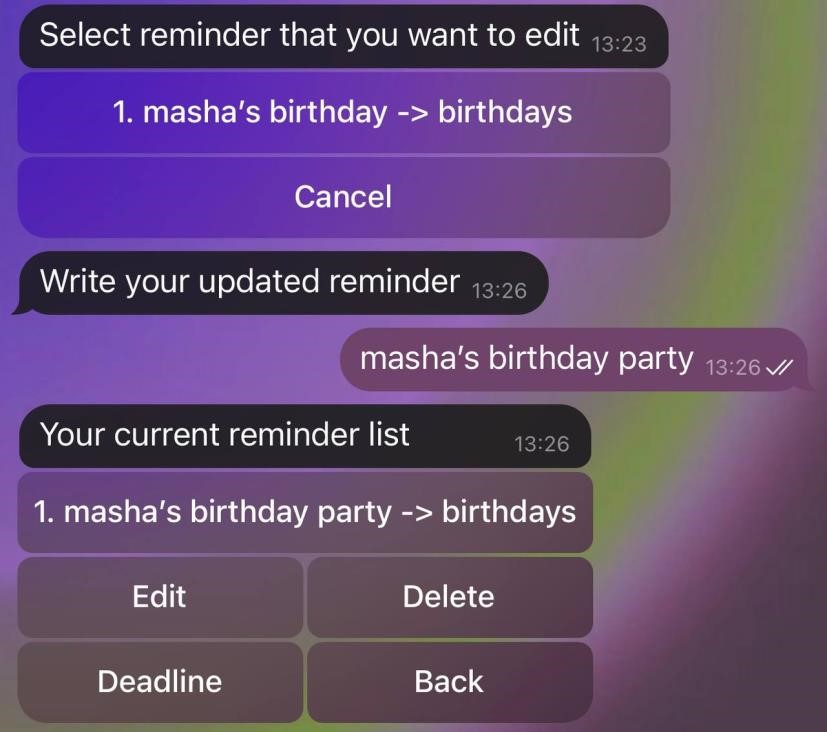


Рисунок 3.6 – Оновлення нагадування

На рисунку 3.7 показано результат натискання кнопки "Deadline". Потім потрібно вказати дату, коли ми хочемо отримати нагадування, і зачекати на цей момент.

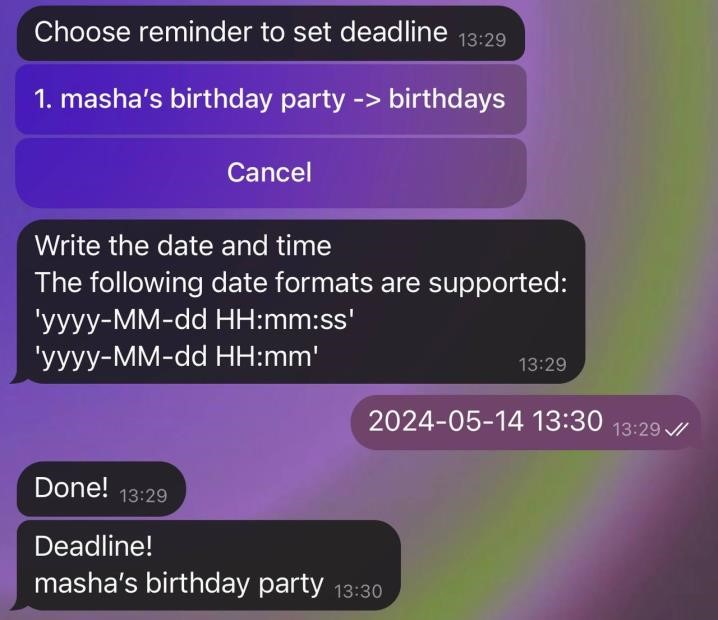


Рисунок 3.7 – Встановлення дедлайну нагадування

На рисунку 3.8 показана функціональність кнопки "Back", яка відправляє користувача до початкового стану телеграм-бота.



Рисунок 3.8 – Початковий стан телеграм-бота

На рисунку 3.9 показано дію кнопки "Delete". Вона відповідає за видалення обраного елемента або запису.

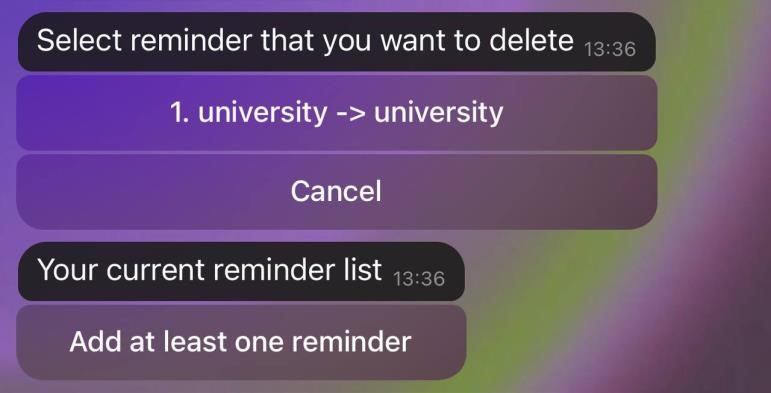


Рисунок 3.9 – Видалення нагадування

3.6 Вихідний код веб-ресурсу

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

3.7 Висновки до третього розділу

У цьому розділі курсової роботи було виконано програмну реалізацію вебресурсу для телеграм-бота на основі фреймворку Spring. Для того, щоб провести програмну реалізацію, був використаний проєкт веб-ресурсу, який був розроблений в другому розділі даної роботи.

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 здійснено функціональне тестування розробленого веб-ресурсу. Був розроблений протокол тестування, що складався з набору тест-кейсів, та проведено саме тестування. В результаті було визначено, що всі тест-кейси пройшли успішно, що свідчить про те, що фактична робота веб-ресурсу відповідає очікуваній поведінці, визначеній у функціональних вимогах.

Крім того, була підготовлена інструкція користувача, що описує функціонал додатку та можливі дії, які може виконати користувач. Також був наданий вихідний код розробленого веб-ресурсу.

ВИСНОВКИ

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

Цей веб-ресурс дозволяє користувачам переглядати, оновлювати та видаляти задачі, а також створювати нагадування. Отже, основна мета цієї курсової роботи була повністю досягнута. Для досягнення цієї мети було вирішено наступні завдання.

У першому розділі даної курсової роботи було детально досліджено область, пов'язану з створенням веб-ресурсу. Були визначені основні завдання, які необхідно вирішити під час розробки цього веб-ресурсу. Також був проведений аналіз існуючих аналогів та визначені ключові вимоги до створення власного вебресурсу. Щодо розробки серверної частини, було вирішено використовувати Java Spring, а також обрано PostgreSQL в якості системи управління базами даних для цього 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. 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 проекту, 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-ofinformationtechnology-in-retail/

1. The Importance of Information Technology in Retail.

URL:

https://www.aeologic.com/blog/the-importance-of-information-technology-in-retail/

1. Angular is a platform for building mobile and desktop web applications. URL:

https://angular.io/

1. Angular components overview. URL:

https://angular.io/guide/componentoverview

1. The Good and the Bad of Angular Development.

URL:

https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-

angulardevelopment/

1. Node.js – an open-source, cross-platform JavaScript runtime environment. URL: https://nodejs.org/en
2. Django: The web framework for perfectionists with deadlines. URL:

https://www.djangoproject.com/

8. Ruby on Rails — A web-app framework. URL: https://rubyonrails.org/

9. Laravel - The PHP Framework For Web Artisans. URL: https://laravel.com/

10. Spring | Home. URL: https://spring.io/

11. Express – fast, unopinionated, minimalist web framework for Node.js.

URL: https://expressjs.com/

1. Welcome to Flask. URL: https://flask.palletsprojects.com/
2. ASP.NET – open-source web framework for .NET infrastructure.

URL: https://dotnet.microsoft.com/en-us/apps/aspnet

1. Java spring framework – pros, cons, common mistakes. URL:

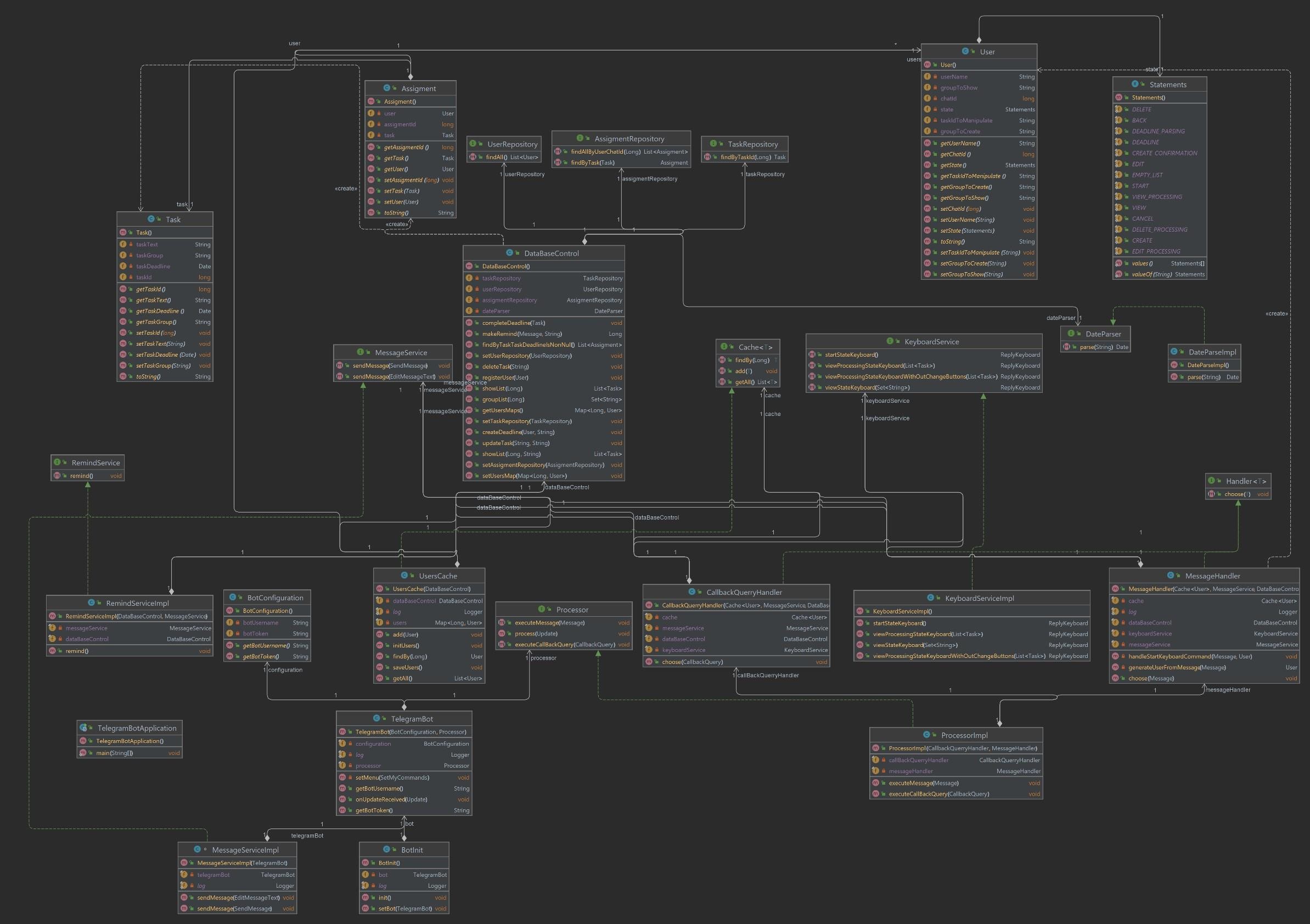
https://bit.ly/3A1dS4w

1. Key Components and Internals of Spring Boot Framework. URL: https://bit.ly/3ojMdtb
2. MySQL Home. URL: https://www.mysql.com/
3. PostgreSQL: The world's most advanced open source database. URL: https://www.postgresql.org/
4. MongoDB Atlas Database - The Database For Modern Apps. URL: https://bit.ly/414ADjX
5. Industry-leading performance and security with SQL Server 2019. URL:

https://www.microsoft.com/en-us/sql-server/sql-server-2019?rtc=1

1. Database | Oracle. URL: https://www.oracle.com/database/

Додаток А



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.r epository.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(); }