МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЧЕРНІГІВСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНОЛОГІЧНИЙ УНІВЕРСИТЕТ

КАФЕДРА ІНФОРМАЦІЙНИХ І КОМП'ЮТЕРНИХ СИСТЕМ

ЗАТВЕРДЖУЮ

Зав. кафедрою ІКС

д.т.н.

С.В. Зайцев

“\_\_\_\_” \_\_\_\_\_\_\_\_\_\_\_2018 р.

**Розробка інтерактивного фінансового додатку до програмного засобу Telegram**

**Кваліфікаційна робота бакалавра за напрямом 6.050102**

**“Комп’ютерна інженерія”**

Виконавець

студент гр. КІ-143                                                               Є.В.Пшеничний

Керівник

доцент, к.т.н.                                         Є.В. Риндич

Чернігів 2018

**ТЕХНІЧНЕ ЗАВДАННЯ**

на виконання кваліфікаційної роботи бакалавра

**Пшеничного Є.В. КІ 143**

Тема роботи: **Розробка інтерактивного фінансового додатку до програмного засобу Telegram**

**Передбачувані технічні результати роботи**

Розробити фінансовий додаток для програмного засобу Telegram.

Розроблюваний додаток повинен надавати користувачу інформацію:

* курс валют по введених місту та банку;
* курс валют по введених місту, банку та даті;
* міжбанковий курс валют.

У ході виконання кваліфікаційної роботи потрібно розробити і описати:

* архітектуру програмного забезпечення фінансового додатку;
* структуру бази даних, яка буде забезпечувати зберігання курсів валют;
* розробку програмного забезпечення фінансового додатку.

**Об'єм текстової і графічної документації**

Робота об'ємом 60-70 с. формату А4. 8-10 електронних слайдів, ілюстрацій і креслень.

**Передбачувана трудомісткість роботи –** 100 людино-годин**.**

**Планові терміни по етапах**

Передзахист з повним поданням чистових роздруківок текстів та ілюстративного матеріалу 14.06.2018.

**Плановий термін захисту роботи**

Робота планується до захисту на засіданні ДЕК 21.06.2018.

**Керівник роботи**

Керівник роботи – доцент, к.т.н. Риндич Є.В.

Керівник роботи

доцент, к.т.н. Є.В. Риндич

Дата видачі завдання

« » \_\_\_\_\_\_\_\_\_\_\_\_ 2018 р.

РЕФЕРАТ

Кваліфікаційна робота бакалавра, \_\_ с., \_\_ рис., \_\_ табл., \_додатки, \_\_ джерел.

Об'єктом розробки була програмна система ”Фінансовий додаток для програмного засобу Telegram”.

Метою роботи було створення програмної системи для надання користувачу курсів валют.

Основним методом проектування було структурне проектування із застосуванням UML-діаграм в інструменті IntelliJ IDEA та веб інструментах ресурсу draw.io.

У ході виконання кваліфікаційної роботи проведено аналіз існуючих систем і розроблені:

– архітектура програмного забезпечення;

– реалізація програмного забезпечення;

– розробка структури бази даних;

Користувач може отримати курси валют по введених місту, банку та/або даті, та отримати міжбанковий курс валют.

Виконання роботи проводилось у відповідності з поставленими вимогами.

Можливе подальше вдосконалення системи шляхом додавання нового функціоналу: отримання готівкового курсу валют, конвертація валют.

Робота має практичну цінність. Розрахунок економічної ефективності не проводився.

Java, REST, Json, Spring, Spring Boot, MongoDB

РЕФЕРАТ

Квалификационная работа бакалавра, \_\_ с., \_\_ рис., \_\_ табл., \_приложения, \_\_ источников.

Объектом разработки была программная система ”Финансовое приложение для программного средства Telegram ”.

Целью работы было создание программной системы для предоставления пользователю курсов валют.

Основным методом проектирования было структурное проектирование с использованием UML-диаграмм в инструменте Intellij Idea и веб инструментах ресурса draw.io.

В ходе выполнения квалификационной работы был проведен анализ существующих систем и разработаны:

– архитектура программного обеспечения;

– реализация программного обеспечения;

– структура базы данных;

Пользователь может получить курсы валют по введенным городу, банка и / или дате, и получить межбанковый курс валют.

Выполнение роботы проводилось в соответствии с поставленными требованиями.

Возможно дальнейшее усовершенствование системы путем добавления нового функционала: получение наличного курса валют, конвертация валют.

Работа имеет практическую ценность. Расчет экономической эффективности не проводился.

Java, REST, Json, Spring, Spring Boot, MongoDB

ABSTRACT

Qualifying work specialist, \_\_ p., \_\_ fig., \_ tab., \_annexes, \_\_ sources.

The object was to develop hardware-software system " Financial application for the Telegram".

The aim was to create software system to listen to provide the user with currency exchange rates.

The main method of design was structured design using UML-diagram tool Intellij Idea and web instruments at the resource draw.io.

In the course of the qualification of the analysis of existing systems and developed:

- The architecture of the software;

- Implementation of the software;

- Database structure;

They user can get exchange rates for the city, bank and/or date entered, and get the interbank exchange rate.

Implementation work was conducted in accordance with the requirements.

It is possible to further improve the system by adding new functionality: obtaining a cash exchange rate, currency conversion.

The work has a practical value. Calculation of cost-effectiveness was not conducted.

Java, REST, Json, Spring, Spring Boot, MongoDB

Перелік умовних позначень, символів, одиниць, скорочень і термінів

БД – База даних

ПЗ – Програмне забезпечення

СУБД – Система управління базами даних

REST – Representational State Transfer

API – Application Programming Interface

HTTP – HyperText Transfer Protocol

ВСТУП 9

1. ОГЛЯД ІСНУЮЧИХ РЕАЛІЗАЦІЙ ФІНАНСОВИХ ПРОГРАМНИХ ДОДАТКІВ 10

1.1. Базові поняття роботи інтернет-ботів 10

1.2. Характеристики розповсюджених реалізацій фінансових ботів 11

1.2.1. Exchange Rates 11

1.2.2. Valuta.Today 12

1.2.3. Сайт kurs.com.ua 13

1.2.4. Privatbank bot - курс валют 13

1.3. Порівняльний аналіз існуючих реалізацій 14

2. ПРОЕКТУВАННЯ ФІНАНСОВОГО ДОДАТКУ ДО ПРОГРАМНОГО ЗАСОБУ TELEGRAM 16

2.1. Вимоги до реалізації 16

2.1.1. Вимоги до програмної частини 16

2.1.2. Вимоги до інтерфейсу 16

2.2. Вибір мови програмування та середовища програмування 16

2.3. Вибір системи збирання 18

2.3.1. Ant 18

2.3.2. Maven 19

2.3.3. Gradle 19

2.3.4. Вибір системи збирання 20

2.4. Додаткові програмні засоби 20

2.4.1. Spring Boot 20

2.4.2. Google JSON Simple 20

2.4.3. Apace HTTP Components 21

2.5. Розробка архітектури 21

2.5.1. Пректування архітектури застосування 22

2.5.2. REST API 22

2.6. Проектування бази даних 23

2.6.1. Сховище ключ-значення 24

2.6.2. Розподілене сховище 25

2.6.3. Документо-орієнтовані сховища 26

2.6.4. Бази даних на основі графів 26

2.6.5. Вибір та проектування бази даних 27

2.7. Uml діаграми 29

2.8. Рзробка алгоритмів роботи 34

3. РЕАЛІЗАЦІЯ ФІНАНСОВОГО ДОДАТКУ ДО ПРОГРАМНОГО ЗАСОБУ TELEGRAM 37

3.1. Реалізація програмного забезпечення 37

3.1.1. Розробка бота 37

3.1.2. Розробка REST API 40

3.2. Реалізація інтерфейсу 40

3.3. Опис процесу автоматизації збірки та установки додатку 45

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ 47

ВИСНОВКИ 48

ДОДАТКИ 50

ДОДАТОК А 51

ВСТУП

Internet розвивається досить стрімко. Швидко зростає кількість видань, присвячених Мережі, що віщує широке її розповсюдження навіть у далекі від техніки областях. Internet перетворюється з великої іграшки для інтелектуалів на повноцінне джерело різноманітної корисної інформації для будь-якої категорії користувачів.

Мессенджер – це програма, мобільний додаток або веб-сервіс для миттєвого обміну повідомленнями.[1]

Найчастіше під месенджером розуміють програму, в яку ви пишете повідомлення і де ви їх читаєте. Однак, за кожною такою програмою варто мережу обміну повідомленнями, яка теж входить в поняття "мессенджер". Це може бути мережу всередині вашої компанії, а може бути глобальна мережа, наприклад Jabber.

Telegram – месенджер, програмне забезпечення для планшетів, смартфонів, ПК, функціонал якого дозволяє обмінюватись текстовими та мультимедійними повідомленням з іншими користувачами, а також виконувати безкоштовні голосові дзвінки.

Кожен з нас вже зараз може зробити свій внесок у розвиток Internet. Для цього досить створити свій бот і розмістити його в Мережі.

Об’єктом розробки слугує фінансовий бот який представляє собою 2 окремі програми: перша названа «ботом», яка являє собою інтерфейс та логіку для вибору операції. Друга названа «серверною», яка являє собою REST API і використовується ботом для отримання даних.

Обмін даними між ботом та сервером відбувається за допомогою технології «HTTP», який є загально прийнятим и доступним для запитів до API.

Бот дозволяє оперативно отримувати курси валют найбільших банків усіх обласних центрів України, отримати міжбакновий курс валют та курс валют по введеній даті. Бот дозволяє економити час користувача, щоб останній не витрачав багато часу на пошуки необхідної інформації вручну. Також є можливість створення аналогічних ботів для інших месенджерів або соцмереж, реалізувавши лише логіку для вибору операції та використавши уже готове REST API.

1. **ОГЛЯД ІСНУЮЧИХ РЕАЛІЗАЦІЙ ФІНАНСОВИХ ПРОГРАМНИХ ДОДАТКІВ**

У данному пункті виконано детальний аналіз предметної області, побудована концептуальна модель предметної області, а також проведений аналіз існуючих реалізацій фінансових ботів та їх порівняльна характеристика.

* 1. **Базові поняття інтернет ботів**

«Бот» (скорочення від «робот») - програма, що імітує діяльність людини. Чат-бот, відповідно, імітує співрозмовника в чаті. Перші експерименти по створенню подібних програм почалися ще півстоліття тому. Один з успішних прикладів надання взаємодії з машиною ілюзії людського спілкування - Еліза, написаний в 1966 році чат-бот, що імітує (або, скоріше, пародіює) діалог з психотерапевтом[2].

Колись в популярному мессенджере ICQ існували десятки ботів, які виконували функції збірки анекдотів, перекладача, калькулятора, які надсилали на вимогу або розкладом прогноз погоди, курси валют, новини, кіноафішу і т.д. Для вирішення елементарних завдань користувачеві не потрібно було залишати месенджер і відкривати браузер - нерідко в офісах їх блокували або обмежували зовнішній веб-трафік.

Сьогодні чат-боти здатні і повсюдно заміняють собою служби підтримки всіляких сервісів.

Замовити квитки, викликати таксі, зробити платіж або переказ, побудувати маршрут, скачати фільм або музику теж допоможуть боти. Бот-юрист, написаний 19-річним британцем, допоміг автовласникам оскаржити штрафів на $ 3 млн.

Віртуальні помічники можуть функціонувати на основі певних правил. У цьому випадку вони реагують тільки на певні команди, і вже не здатні впоратися з ситуацією, коли клієнт робить «крок вліво або вправо».[3]

Чат-боти, що працюють з використанням машинного навчання, є самонавчального, вони «розумнішають» у міру того, як спілкуються з людьми.

Використання штучного інтелекту - дуже перспективний напрямок. За прогнозами, в 2018 році 75% компаній-розробників включать ІІ, або схожу функцію самонавчання хоча б в одне з розроблюваних додатків. А тому ті інтернет-магазини, які хочуть бути попереду, інтегрують чат-боти в свої системи.

* 1. **Характеристики розповсюджених реалізацій фінансових ботів**

Інтернет-бот — спеціальна програма, що виконує автоматично і/або за заданим розкладом які-небудь дії через ті ж інтерфейси, що й звичайний користувач. Найбільш розповсюдженими реалізаціями є боти, які надають курс валют незалежно від міста та банку, можуть конвертувати валюту по поточному курсу. Розглянемо декілька готових реалізацій фінансових ботів, знайдених в мережі Інтернет.

* + 1. **Exchange Rates**

Бот Exchange Rates дозволяє конвертувати суми з однієї валюти в іншу, отримати курс валют(підтримує їх більн ніж 150), має досить великий список мов, серед яких є і українська.

На рисунку 1.2 зображено інтерфейс Exchange Rates.

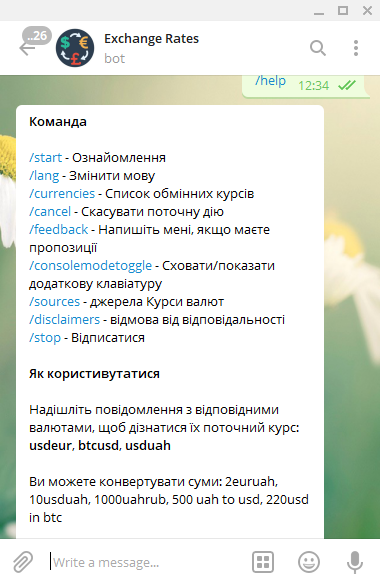


Рисунок 1.2 - Інтерфейс Exchange Rates

Exchange Rates - типовий приклад фінансового боту. Не можна сказати, що він має багатий функціонал, але має велику кількість валют, які можна конвертувати, та досить непогану інтернаціоналізацію(13 мов).

Конвертація валют відбувається за допомогою відправки простих

команд: «сума»«валюта, з якої конвертуємо»« валюта, в яку конвертуємо». Це не є зручно, адже можна легко допустити помилку.

Також є недоліком реклама, яка, хоч і не в великих кількостях, але присутня.

Серед переваг даного рішення можна виділити:

* Наявність великої кількості підтримуваних валют;
* Інтернаціоналізацю.

До недоліків слід віднести:

* Реклама;
* Незручний інтерфейс.
  + 1. **Valuta.Today**

Розглянемо інший приклад фінансового боту. Він є більш простим, ніж попередньо розглянутий. Основними відмінностями від попереднього є відсутність реклами, вибір країни, курс валют, якої хочемо бачити, менша кількість валют та лише одна мова інтерфейсу(російська).

Отримати дані можна ввівши команду «/cash {країна} [валюта]» - для отримання курсу валют готівкою або «/official {країна} [валюта]»» - для отримання офіційного курсу валют.

На рисунку 1.3 зображений інтерфейс Valuta.Today.

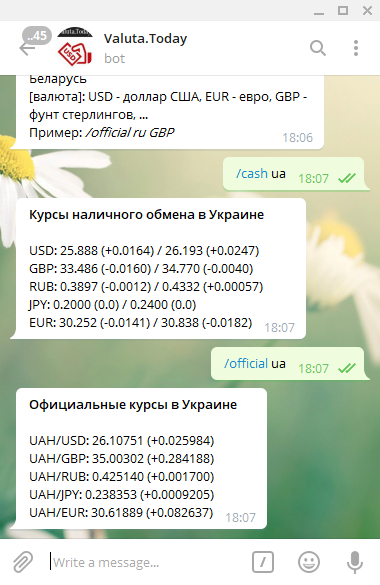
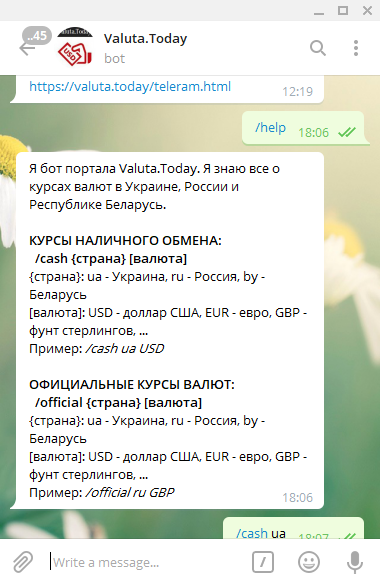


Рисунок 1.3 - Інтерфейс Valuta.Today

Серед переваг даного рішення можна виділити:

* Відсутність реклами;
* Можливість вибору країни для перегляду курсів валют;
* Простий інтерфейс.

До недоліків можна віднести:

* Відсутність можливості зміни мови інтерфейсу;
* Порівняно мала кількість валют.
  + 1. **Сайт kurs.com.ua**

Розлянемо приклад сайту, який надає більше інформації, ніж боти.

На рисунку 1.4 зображений інтерфейс сайту.

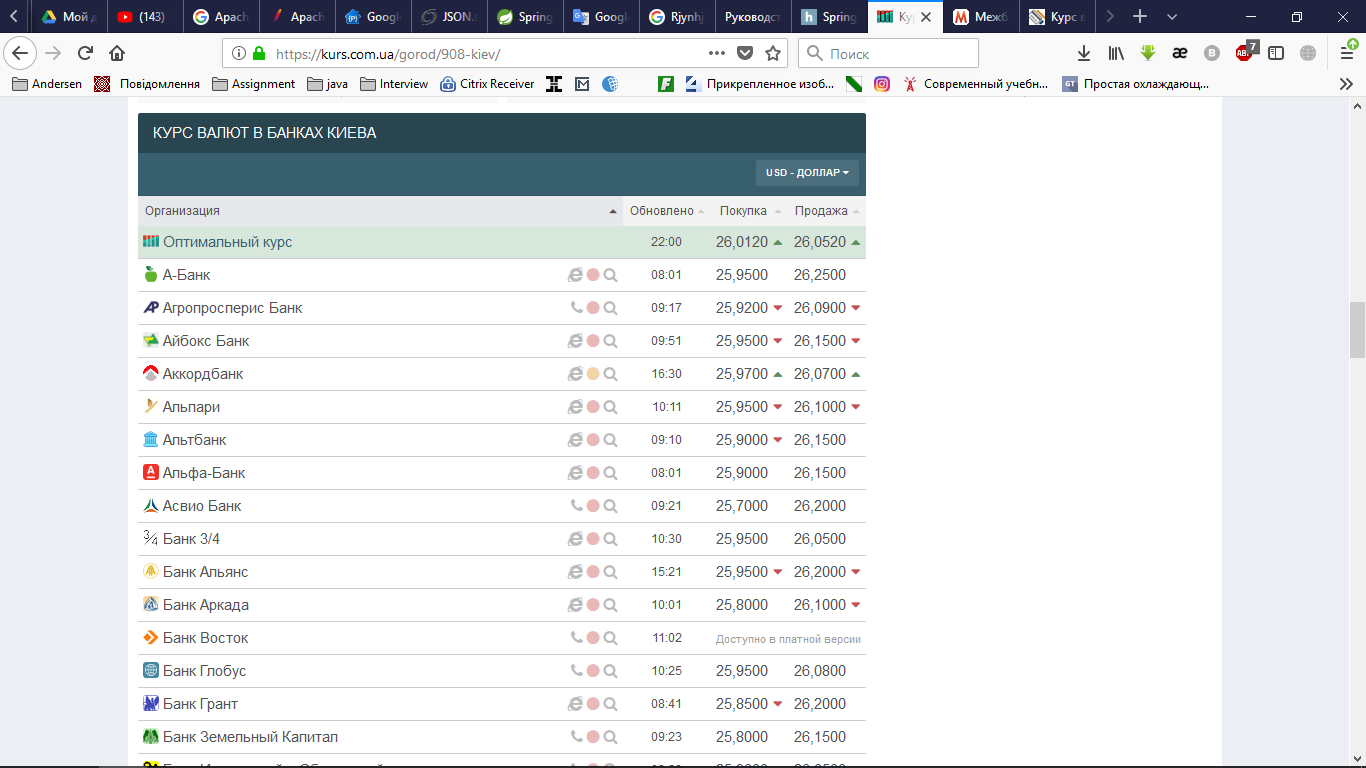


Рисунок 1.3 - Інтерфейс сайту kurs.com.ua

Серед переваг даного рішення можна виділити:

* Відсутність реклами;
* Можливість вибору міста та банку для перегляду курсів валют;
* Отримання історії курсів валют.

До недоліків можна віднести:

* Загромаджений інтерфейс;
* Тільки російська мова інтерфейсу.
  + 1. **PrivatBank Bot - Курс валют**

PrivatBank Bot - Курс валют – має тільки одну функціональну команду: «Показать курс».

Також пропонує підписатися на канал, в якому кожного дня публікуються курси валют.

Даний бот має невеликий перелік курсів валют та зручний інтерфейс.

На рисунку 1.5 зображений інтерфейс PrivatBank Bot - Курс валют.

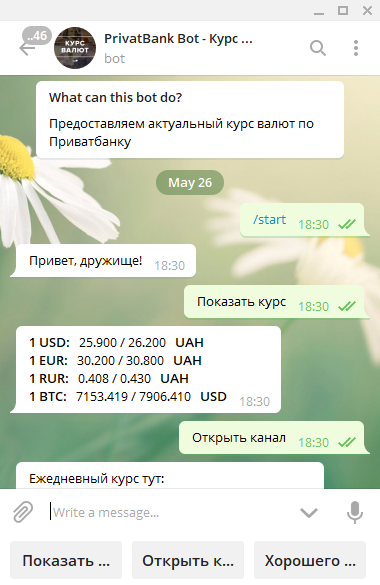


Рисунок 1.5 - Інтерфейс PrivatBank Bot - Курс валют

Серед переваг даного рішення можна виділити:

* Відсутність реклами;
* Зручний інтерфейс;

До недоліків слід віднести:

* Порівняно мала кількість валют;
  1. **Порівняльний аналіз існуючих реалізацій**

Для того щоб провести порівняльний аналіз, необхідно виділити базові вимоги до розроблюваного рішення - критерії за якими власне і буде проводитися порівняння існуючих реалізацій. Базові вимоги:

* Зручний інтерфейс;
* Вибір міста та банку для отримання курсу валют;
* Отримання міжбанкового курсу валют;
* Отримання курсу валют по обраній даті.

Наявність або відсутність цих вимог у кожному з розглянутих рішень проаналізовано у таблиці 1.1

Таблиця 1.1 - Порівняльний аналіз існуючих реалізацій.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Назва інтернет-радіо | Exchange Rates | Valuta.  Today | PrivatBank  Bot – Курс  валют | Сайт  kurs.com  .ua | Розроб-люваний  бот |
| Зручний інтерфейс | - | + | + | + | + |
| Вибір міста та банку | - | - | - | + | + |
| Отримання міжбанкового курсу валют | - | - | - | - | + |
| Отримання курсу валют по обраній даті | - | - | - | + | + |

На основі проведення порівняльного аналізу існуючих реалізацій фінансових ботів, у відповідності з визначеними вимогами, можна сказати наступне: сформульованим критеріям порівняння найбільше відповідає kurs.com.ua. Його недоліком є відсутність отримання міжбанкового курсу валют. При розробці свого бота планується дані недоліки усунути.

1. **ПРОЕКТУВАННЯ ФІНАНСОВОГО ДОДАТКУ ДО ПРОГРАМНОГО ЗАСОБУ TELEGRAM**
   1. **Вимоги до реалізації**

Виходячи з разультатів аналізу, проводеного на попередньому етапі, можна сформувати базовий перелік критеріїв і вимог, яким повинен відповідати розроблюваний фінансовий бот.

* + 1. **Вимоги до програмної частини**

Розглянемо основні вимоги до розроблюваного боту та його функцій:

* Бот повинен надавати курс валют по вибраному місту та банку, при цьому місто користувач вводить з клавіатури(введення не залежить від регістру), а банк обирає зі списку;
* Бот повинен при помилці введення міста користувачем надати йому вибір серед ;
* Бот повинен надавати міжбанковий курс валют;
* Бот повинен надавати курс валют по обраній даті
* Дані в БД повинні оновлюватись автоматично по таймеру(один раз в добу, о 9:00).
  + 1. **Вимоги до інтерфейсу**

Розроблюваний бот матиме програмний інтерфейс у вигляді клієнта Telegram, основними функціями якого будуть:

* Отримання між банкового курсу;
* Отримання курсу по місту і банку;
* Отримання курсу по місту, банку і даті;
* Відображення курсу валют.
  1. **Вибір мови програмування та середовища програмування**

На сьогоднішній день існує велика кількість мов програмування: C, C++, C#, Java і т.д. Кожна з яких має свої переваги і недоліки. Автором була обрана об’єктно-орієнтована мова Java. Розглянемо декілька середовищ для програмування на Java:

– IntelliJ IDEA;

– Eclipse IDE;

– NetBeans.

IntelliJ IDEA з точки зору можливостей і ціни поставляється в двох варіантах: безкоштовного Community edition, і платного Ultimate edition з розширеною функціональністю.

Community edition призначена для JVM- і Android-розробки. Безкоштовна версія підтримує Java, Kotlin, Groovy і Scala; Android; Maven, Gradle і SBT; працює з системами контролю версій Git, SVN, Mercurial і CVS.

Ultimate edition пристосована для веб-і enterprise-розробки. Ця версія IDE працює не тільки з Git, SVN, Mercurial і CVS, але також з Perforce, ClearCase і TFS; в ній ви зможете писати на JavaScript і TypeScript; природно, є підтримка Java EE, Spring, GWT, Vaadin, Play, Grails і ряду інших фреймворків. І, звичайно, не обійшлося без SQL і інструментів для роботи з базами даних.

Eclipse, довгі роки впевнено тримав пальму першості за популярністю СЕРЕД Java IDE. Це середовище повністю безкоштовна, з відкритим вихідним кодом, написаним переважно на Java. Проте, її модульна архітектура дозволяє використовувати Eclipse, і з іншими мовами.

Портативність Java Eclipse, допомагає бути кроссплатформенной середовищем: IDE ця працює на Linux, Mac OS X, Solaris і Windows.

Добре це чи погано, Java Standard Widget Toolkit (SWT), принаймні частково, відповідає за зовнішній вигляд Eclipse.

Своєю продуктивністю (або, як кажуть деякі доброзичливці, її відсутності) в Eclipse JVM зобов'язана. Eclipse, працює досить повільно, оскільки впирається корінням в досить старе «залізо» і стародавні версії віртуальної машини Java. Навіть сьогодні вона здається повільною, особливо якщо начепити на неї багато плагінів.

Частина витрат ресурсів Затемнення можна віднести на рахунок її вбудованого інкрементного компілятора, який запускається при старті файлу або оновленні коду. Корисна штука, саме вона ловить помилки при введенні тексту.

Незалежно від збірки, проект Eclipse, підтримує модель контенту, яка містить інформацію про ієрархію типів, посилань і оголошеннях Java-елементів[4].

У порівнянні з іншими IDE середовищами IDE NetBeans середовище забезпечує висококласну комплексну підтримку Java новітніх технологій і останніх удосконалень Java стандартів. Це перша безкоштовна середовище IDE, що підтримує JDK 8, JDK 7, Java EE 7, Включаючи відповідні удосконалення і HTML5 JavaFX 2.

Завдяки постійно удосконалюються редактору Java, різноманітним можливостям і широкого спектру інструментів, шаблонів і зразків, IDE NetBeans встановлює стандарт розробки за допомогою новітніх готових технологій[5].

Редактор підтримує безліч мов - від Java, C / C ++, XML і HTML до PHP, Groovy, Javadoc, JavaScript і JSP. Редактор є розширюваним, тому можна підключити підтримку для безлічі інших мов.

Оглянувши всі недоліки та переваги та огляд можливостей було обрано IntelliJ IDEA Ultimate edition(за навчальною ліцензією), тому що вона досить потужна, швидко працює, підтримує багато фреймворків.

* 1. **Вибір системи збирання**
     1. **Ant**

Apache Ant - фреймворк для автоматизації процесу складання програмного продукту. Разом з Apache Maven є найпопулярнішими збирачами проектів. Також, як і Maven, дуже активно використовується в великих проектах.[6]

Ant, на відміну від іншого збирача проектів Apache Maven, забезпечує імперативну, а не декларативну збірку проекту. Управління процесом складання відбувається за допомогою XML-сценарію, який також називають Build-файлом. В першу чергу цей файл містить визначення проекту, що складається з окремих цілей (Targets). Цілі можна порівняти з процедурами в мовах програмування і містять виклики команд-завдань (Tasks). Кожне завдання являє собою неподільну, атомарному команду, що виконує деякий елементарне дію.

Між цілями можуть бути визначені залежності - кожна мета виконується тільки після того, як виконані всі цілі, від яких вона залежить (якщо вони вже були виконані раніше, повторного виконання не проводиться).

Найпопулярнішими завданнями (Tasks) є:

1) Javac - компіляція Java-коду

2) Copy - копіювання файлів

3) Delete - видалення файлів і директорій

4) Move - переміщення файлів і директорій

5) Replace - заміщення фрагментів тексту в файлах

6) JUnit - автоматичний запуск юніт-тестів

7) Exec - виконання зовнішньої команди

Загалом, таких команд близько 150, але малоймовірно, що вони все вам знадобляться.

* + 1. **Maven**

Maven – це інструмент для збірки Java проекту: компіляції, створення jar, створення дистрибутива програми, генерації документації. Прості проекти можна зібрати в командному рядку. Якщо збирати великі проекти з командного рядка, то команда для збірки буде дуже довгою, тому її іноді записують в bat / sh скрипт. Але такі скрипти залежать від платформи. Для того щоб позбутися від цієї залежності і спростити написання скрипта використовують інструменти для збірки проекту.[7]

Основні переваги Maven:

* Незалежність від OS. Збірка проекту відбувається в будь-якій операційній системі. Файл проекту один і той же.  
  Управління залежностями. Рідко які проекти пишуться без використання сторонніх бібліотек (залежностей). Ці сторонні бібліотеки часто теж в свою чергу використовують бібліотеки різних версій. Мавен дозволяє управляти такими складними залежностями. Що дозволяє вирішувати конфлікти версій і в разі потреби легко переходити на нові версії бібліотек.
* Можлива збірка з командного рядка. Таке часто необхідно для автоматичного складання проекту на сервері (Continuous Integration).
* Гарна інтеграція з середовищами розробки. Основні середовища розробки на java легко відкривають проекти які збираються c допомогою maven. При цьому найчастіше проект налаштовувати не потрібно - він відразу готовий до подальшої розробки.
* Як наслідок - якщо з проектом працюють в різних середовищах розробки, то maven зручний спосіб зберігання налаштувань. Її конфігураційний файл середовища розробки і для збірки один і той же - менше дублювання даних і відповідно помилок.
* Декларативний опис проекту.
  + 1. **Gradle**

Gradle - система автоматичного складання, побудована на принципах Apache Ant і Apache Maven, але надає DSL на мові Groovy замість традиційної XML-подібної форми подання конфігурації проекту.[8]

На відміну від Apache Maven, заснованого на концепції життєвого циклу проекту, і Apache Ant, в якому порядок виконання завдань (targets) визначається відносинами залежності (depends-on), Gradle використовує спрямований ациклічний граф для визначення порядку виконання завдань.

Gradle був розроблений для розширюваних многопроектной збірок, і підтримує інкрементальні збірки, визначаючи, які компоненти дерева збірки не змінилися і які завдання, залежні від цих частин, не вимагають перезапуску.

Основні плагіни призначені для розробки і розгортання Java, Groovy і Scala додатків, але готуються плагіни і для інших мов програмування.

* + 1. **Вибір системи збирання**

Оглянувши всі недоліки та переваги та огляд можливостей було обрано систему збирання - maven, тому що вона має гарну інтеграцію з середовищем розробки, дуже розповсюджена, що дає дуже велику кількість користувачів, що в свою чергу дозволяє швидко знаходити вирішення проблем, також автор має великий досвід користування нею.

* 1. **Додаткові програмні засоби**
     1. **Spring Boot**

Spring Boot дозволяє вам лекго створювати повноцінні, виробничого класу Spring-додатки, про які можна сказати - "просто запусти". Ми включили Spring-платформу і сторонні бібліотеки, щоб ви могли запустити з мінімум зусиллями. Більшості Spring Boot додатків потрібно зовсім маленька Spring-конфігурація.[9]

Можливості:

* Створення повноцінних Spring додатків
* Вбудований Tomcat або Jetty (не потрібно установки WAR файлів)
* Забезпечує «початкові» POMs для спрощення вашої Maven конфігурації
* Автоматична конфігурірація Spring коли це можливо
* Забезпечує такими можливостями, як метрики, моніторинг станами і розширена конфігурація
* Абсолютно без генерації коду і без написання XML конфігурації.
  + 1. **Google JSON Simple**

JSON.simple - це простий набір інструментів Java для JSON. Ви можете використовувати JSON.simple для кодування або декодування тексту JSON.[10]

Особливості:

* Повна відповідність специфікації JSON (RFC4627)
* Забезпечує безліч функціональних можливостей, таких як кодування, декодування / розбір та вилучення з тексту JSON при одночасному збереженні легкої бібліотеки
* Гнучкий, простий і простий у використанні, повторно використовуючи інтерфейси Map і List
* Підтримує потокове виведення тексту JSON
* Зупинка SAX-подібного інтерфейсу для потокового введення тексту JSON
* Синтаксичний аналізатор
* Висока продуктивність (див. Тестування продуктивності)
* Немає залежності від зовнішніх бібліотек
* І вихідний код, і бінарний - сумісні з JDK1.2.
  + 1. **Apache HTTP Components**

Проект Apache HttpComponents ™ відповідає за створення та підтримку набору інструментів низькорівневих компонентів Java, орієнтованих на HTTP та пов'язані протоколи.[11]

Цей проект працює під управлінням Apache Software Foundation (http://www.apache.org), і є частиною великої спільноти розробників і користувачів.

Протокол Hyper-Text Transfer (HTTP), мабуть, є найважливішим протоколом, що використовується в Інтернеті сьогодні. Веб-сервіси, пристрої, що підтримують мережу, та зростання мережевих обчислень продовжують розширювати роль HTTP-протоколу за винятком користувальницьких веб-браузерів, одночасно збільшуючи кількість програм, які потребують підтримки HTTP.

Призначений для розширення, забезпечуючи надійну підтримку базового HTTP-протоколу, HttpComponents можуть представляти інтерес для всіх, хто створює HTTP клієнтські та серверні додатки, такі як веб-браузери, HTTP-проксі, бібліотеки транспортних мереж веб-служб або системи, що використовують HTTP-протокол для розподіленої комунікації.

* 1. **Розробка архітектури**

Архітектура проекту під собою розуміє серверну частину та бот.

Серверна частина являє собою прошарок, з однії сторони якого - БД, а з іншої REST API, яке в свою чергу використовується ботом.

* + 1. **Проектування архітектури застосування**

В даному пункті описана архітектура застосування фінансового боту, яка представлена ​​на рисунку 2.1.

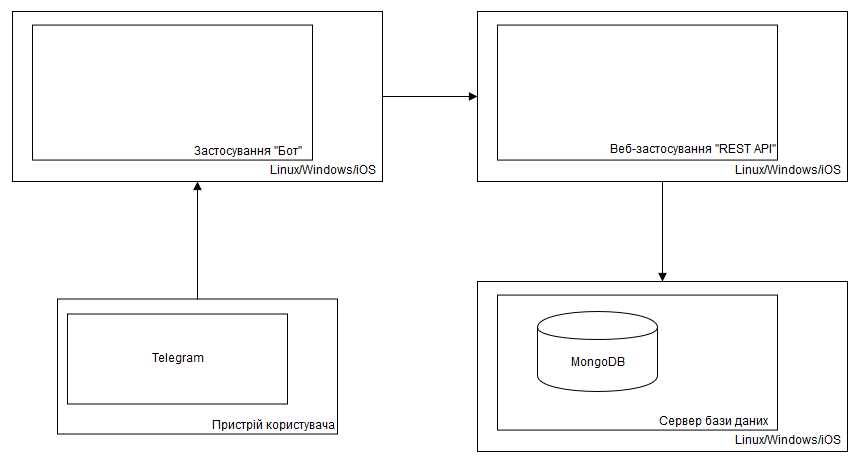


Рисунок 2.1 – Архітектура застосування

Основними функціональними елементами архітектури веб-застосування інтернет магазину спортивної продукції, є:

Користувачі - кінцеві користувачі, які використовують бот;

Сервер – сервер, на якому розгортається бот та сервер, на якому розгормається API;

СУБД MongoDB - система управління базами даних MongoDB, яка є сховищем даних.

* + 1. **REST API**

Архітектура серверної частини побудована на основі фреймворку Spring. Він інкаплулює обробку високорівневих об'єктів додатку, наприклад, такі об'єкти, як об'єкти запиту та відповіді, надаючи їх в розпорядження розробнику[12].

Таблиця 2.1 – Семантика запитів відповідно REST

|  |  |
| --- | --- |
| Ресурс \ Метод | GET |
| /cities | Повернути список міст |
| /banks/{city} | Повернути список банків з указаного міста |
| /currency/{city}/{bank} | Повернути курс валют по вказаному місту та банку |
| /currency/{city}/{bank}/{date} | Повернути курс валют по вказаному місту, банку та даті |
| /interbank | Повернути міжбанковий курс валют |

* 1. **Проектування бази даних**

Бази даних це логічні змодельовані сховища призначені для різних типів даних. Кожна база даних, за винятком безсхемних (NoSQL), використовує модель, яка описує структуру даних. СУБД - це спеціальні програми або набір бібліотек для роботи з різними базами даних.[13]

Протягом останнього десятиліття, вибір багатьох розробників і системних адміністраторів для їх додатків завжди падав на реляційні СУБД. Не дивлячись на те, що вони не такі вже й адаптивні, функціонал реляційних СУБД дозволяє створювати досить складні системи даних. Цього було цілком достатньо, поки не з'явилися NoSQL СУБД.

Сам термін NoSQL з'явився близько десяти років тому, як би це не здавалося смішним, але спочатку це було назвою черговий реляційної системи управління базами даних. Проте мала на увазі вона трохи інше - уникнути використання стандартів SQL. У наступні кілька років інші підхопили цю ідею і стали використовувати цей термін щодо нереляційних баз даних.

За задумом NoSQL бази даних і СУБД не мають на увазі внутрішніх зв'язків. Вони не ґрунтуються на одній моделі, а кожна база даних в залежності від цілей використовує різні моделі.

Існує досить багато різних моделей і функціональних систем для NoSQL баз даних:

* Сховище ключ-значення - Redis, MemcacheDB і т.д. (Зазвичай зберігають дані в пам'яті)
* Розподілене сховище (Column-oriented) - Cassandra, HBase і т.д (призначені для дуже великих обсягів даних).
* Документо-орієнтовані СУБД - MongoDB, Couchbase і т.д. (Призначені для зберігання ієрархічних структур даних - документів)
* БД на основі графів - OrientDB, Neo4J і т.д.

Щоб мати чітке уявлення ніж NoSQL відрізняється від реляційних систем, давайте складемо список з основними відмінностями.

Приклади використання NoSQL:

* Швидкість - NoSQL бази даних зазвичай швидше, а іноді набагато швидше, коли справа доходить до запису. Операції читання також можуть бути досить швидкими в залежності від того яку саме БД ви використовуєте
* Безсхемна розробка - Реляційні СУБД вимагають чітко описану структуру даних до початку роботи. NoSQL рішення пропонують більш гнучкі рішення.
* Автоматизована (або дуже проста) реплікація / Збільшення - NoSQL бази даних швидко розвиваються - розробники активно вирішують основні проблеми, одна з яких - реплікація і масштабування. На відміну від реляційних СУБД, NoSQL рішення легко масштабуються і працюють з кластерами.
* Великий вибір - коли справа доходить до вибору NoSQL сховища, існує великий вибір різних моделей. Як було сказано раніше, то наскільки правильно ви зробите свій вибір вплине на продуктивність вашої системи.
  + 1. **Сховище ключ-значення**

Почнемо наш розгляд з типу сховища ключ-значення, так як це найголовніше рішення з сімейства NoSQL. Цей тип БД працює з даними типу ключ-значення, наприклад як словник. Тут немає місця ні структурі, ні зв'язків. Після підключення до сервера (наприклад Redis) додаток може задати ключ і його значення, а в наслідку отримувати ці данниe за запитом.

Такі СУБД зазвичай використовуються для швидкого збереження базових даних, а іноді не таких вже й базових, якщо підрахувати витрати процесора і пам'яті. Вони, зазвичай, дуже швидкі, працездатні або легко масштабуються (добре використовувати такі БД для зберігання сесій, кеша, лічильників відвідувань або переглядів і т.д.).

Такі БД дуже продуктивні, прості у використанні і легко масштабуються

Популярні СУБД:

* Redis - БД в оперативній пам'яті з додатковою відмово стійкістю;
* Riak - Розподілене, реплікаційне сховище;
* Memcached / MemcacheDB - розподілена БД в оперативній пам'яті.

Приклади використання:

* Кешування - швидке і часте збереження даних для майбутнього використання;
* Черга - деякі БД типу ключ-значення підтримують списки, набори і черги;
* Розподіл інформації/завдань - використовується для реалізації патерну Pub/Sub;
* Живе оновлення інформації – додатки, які використовують стан.
  + 1. **Розподілене сховище**

По суті це наступний крок після СУБД типу ключ-значення. Незважаючи на досить складну для розуміння сутність, ці бази даних відмінно працюють просто створюючи колекції з одного або декількох пар ключ-значення, які в сумі відповідають запису.

На відміну від звичних таблиць в реляційних моделях, ці СУБД не вимагає попереднього опису структури даних. Кожен запис складається з одного або декількох стовпців містять дані, а кожен стовпець різних записів може зберігати різні типи даних.

В цілому, розподілене сховище ні що інше як двовимірний масив, де кожен ключ (запис) містить одну або кілька пар ключ-значення прив'язаних до нього. Така система дозволяє зберігати і використовувати великі обсяги неструктурованих даних. (Один запис з великим кількість додаткової інформації). Такі бази даних зазвичай використовуються, коли недостатньо простих пар ключ-значення, а зберігати необхідно великий обсяг записів з різною інформацією. Такі СУБД при відповідному використанні можуть бути дуже продуктивними.

Такі системи баз даних дуже ефективні і можуть бути використані для зберігання важливої ​​інформації великих обсягів. Може вони де щось не дуже гнучкі в плані даних, зате вони функціональні і продуктивні.

Популярні СУБД:

* Cassandra - структура даних заснована на BigTable і DynamoDB;
* HBase - сховище для Apache Hadoop засноване на принципах BigTable.

Приклади використання:

* Зберігання неструктурованих, що не руйнуються даних - якщо вам необхідно зберігати великі обсяги даних протягом довгого часу, то такі БД дуже добре впораються з завданням;
* Масштабування - за задумом такі бази даних легко масштабуються. Вони легко справляються з будь-яким обсягом даних.
  + 1. **Документо-орієнтовані сховища**

Такі NoSQL СУБД дуже швидко захопили свій ринок. Вони працюють так само як і попередні системи, але вони допускають набагато більшу вкладеність і складність структури даних. (Наприклад, документ вкладений в документ, вкладений в документ). Документи знімають обмеження вкладеності першого і другого рівнів типу ключ-значення в розподілених сховищах. В цілому, можна описати як завгодно складну структуру даних як документ і зберегти в такій БД.

Незважаючи на досить великий функціонал і здатність доступу до даних по одному ключу, такі СУБД мають ряд своїх проблем. Наприклад, при доступі до одного документу ви повністю отримуєте його в відповідь на запит, навіть якщо вам необхідно якесь одне поле, що не може не позначитися на продуктивності. Документ-орієнтовані сховища відмінно зберігають незв'язану інформацію великих обсягів, навіть якщо вона дуже відрізняється від суті до сутності.

Популярні СУБД:

* Couchbase - засноване на JSON, сумісний з Memcached;
* CouchDB - передове документо-орієнтоване сховище;
* MongoDB - дуже популярне і функціональне сховище.

Приклади використання:

* Вкладена інформація - документо-орієнтовані сховища відмінно працюють з глибоко вкладеної, складною інформацією;
* Підтримка JavaScript - одна з відмінних рис документо-орієнтованих сховищ це те, як вони працюють з іншими додатками: підтримка JSON.
  + 1. **Бази даних на основі графів**

БД на основі графів зберігають дані в зовсім іншому вигляді. Вони використовують деревовидні структури з вузлами і зв'язками з'єднують їх. Так само як і в математиці, деякі операції набагато зручніше виконувати з такими даними завдяки зв'язкам між ними і їх угруповання (наприклад, зв'язку між людьми).

Такі бази даних часто використовуються в додатках, де потрібно мати чітко встановлені зв'язку. Наприклад, коли ви входите в будь-якої соціальної мережі, зберігати зв'язку між вами і вашими друзями зберігати набагато простіше при використанні баз даних на основі графів.

Такі типи БД зберігають інформацію абсолютно особливі, зовсім не як всі інші СУБД.

Популярні СУБД:

* OrientDB - дуже швидке документо-орієнтоване сховище гібрид типу граф написане на Java. Включає в себе різні режими роботи;
* Neo4J - безсхемне, дуже потужне і популярне сховище написане на Java.

Приклади використання:

* Робота зі складно пов'язаної інформацією - як було сказано у вступі, сховища типу граф відмінно справляються зі складно пов'язаної інформацією. Наприклад зберігання зв'язків між двома сутностями і цілого ряду різнорівневих зв'язків між сутностями не пов'язаних з першими безпосередньо.
* Моделювання та підтримка класифікацій - такі БД досягли успіху всюди де є зв'язку. Моделювання даних і класифікація різної інформації зі зв'язків можна з легкістю уявити використовуючи ці БД.

Порівняння NoSQL СУБД і реляційних СУБД

Щоб мати чітке уявлення ніж NoSQL відрізняється від реляційних систем, давайте складемо список з основними відмінностями.

Приклади використання NoSQL:

* Швидкість - NoSQL бази даних зазвичай швидше, а іноді набагато швидше, коли справа доходить до запису. Операції читання також можуть бути досить швидкими в залежності від того яку саме БД ви використовуєте.
* Безсхемна розробка - Реляційні СУБД вимагають чітко описану структуру даних до початку роботи. NoSQL рішення пропонують більш гнучкі рішення.
* Автоматизована (або дуже проста) реплікація / Збільшення - NoSQL бази даних швидко розвиваються - розробники активно вирішують основні проблеми, одна з яких - реплікація і масштабування. На відміну від реляційних СУБД, NoSQL рішення легко масштабуються і працюють з кластерами.
* Великий вибір - коли справа доходить до вибору NoSQL сховища, існує великий вибір різних моделей. Як було сказано раніше, то наскільки правильно ви зробите свій вибір вплине на продуктивність вашої системи.
  + 1. **Вибір та проектування бази даних**

Для зберігання даних, необхідних для роботи приймача було прийнято рішення використати JSON. За рахунок своєї простоти та гнучкості даний формат зберігання даних може бути використаний майже де завгодно та як завгодно.

Для зберігання даних, необхідних для роботи боту, було прийнято рішення використовувати СУБД MongoDB.

MongoDB зберігає неструктуровані документи типу BSON(являє собою тип, схожий на JSON, але має підтримку додаткових типів даних), що буде зручно для передавання даних між сервером та ботом, то для зберігання даних прийнято рішення використовувати її.

На основі вимог до розроблюваного програмного забезпечення були створені базові об'єкти даних бази даних бота: "city", "bank", "currency".

Розглянемо детальніше створені об'єкти.

Цілісність зв'язків в MongoDB не підтримується. Для того щоб мати уявлення про відносини між сутностями, була розроблена схема БД, на якій відображені псевдо-зв'язки, рисунок 2.2.

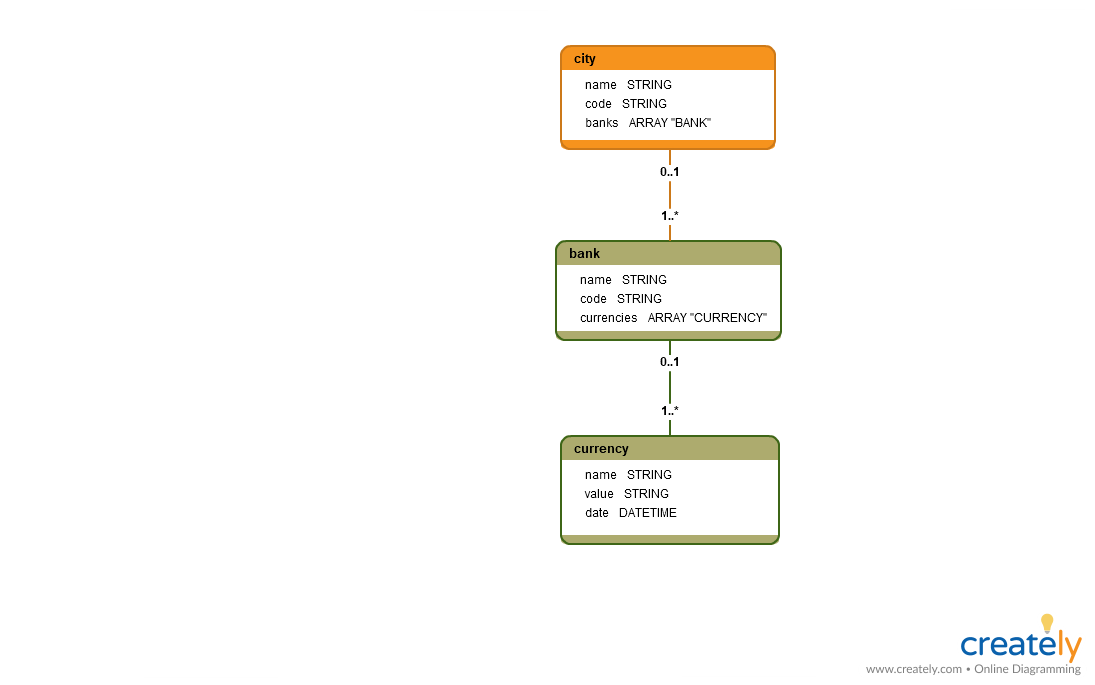


Рисунок 2.2- Схема бази даних фінансового боту з позначенням псевдо-зв’язків

Об'єкт "city" створений для зберігання інформації про місто, включає в себе поля:

* name – назва міста українською мовою;
* code – назва міста латиницею
* banks – список банків.

Поля name і code мають тип String - рядок символів, а banks - являє собою масив з об'єктів типу bank.

Об'єкт "bank" створений для зберігання інформації про банки, а саме їх назву та список валют.

Bank включає в себе поля:

* code – назва банку латиницею;
* name - назва банку;
* currencies – список валют.

Поля name і code мають тип String - рядок символів, а currencies - являє собою масив з об'єктів типу currency.

Об'єкт "currency" створений для зберігання інформації про валюту, а саме їх назву, курс та дату, коли курc був актуальним.

Сurrency включає в себе поля:

* name – назва валюти;
* value – курс валюти відносно гривні;
* date – дата, коли курс був актуальним.

Поля name і value мають тип String - рядок символів, а date – має тип DateTime – дата і час.

**2.7 Uml Діаграми**

Фінансой бот призначений для швидкого та комфортного отримання інформації про курси валют в банках та міжбанкового курсу валют. Основною і єдиною роллю користувача системи буде, власне, користувач. Згідно з вимогами пункту 2.1.2 розділу 2 користувач повинен мати змогу отримати курс валют по місту і банку, по даті, та отримати міжбанковий курс валют.

Нижче, рисунку 2.3 наведена розроблена діаграма варіантів використання для даної ролі.



Рисунок 2.3 - Діаграма варіантів використання фінансового боту

Далі, згідно з вимогами пункту 2.1.2 розділу 2 та вищеописаної діаграми варіантів використання було створено діаграми послідовностей для кожної з функцій боту та для початку роботи самого боту.

Діаграма послідовностей є однією з різновиду діаграм взаємодії та призначена для моделювання взаємодії об'єктів системи в часі, а також обміну повідомленнями між ними.

У нашому випадку маємо одну діючу особу у вигляді користувача та три системи: власне бот, сервер та БД.

На рисунку 2.4 зображена діаграма послідовностей функції «Отримання курсу валют по місту та банку».

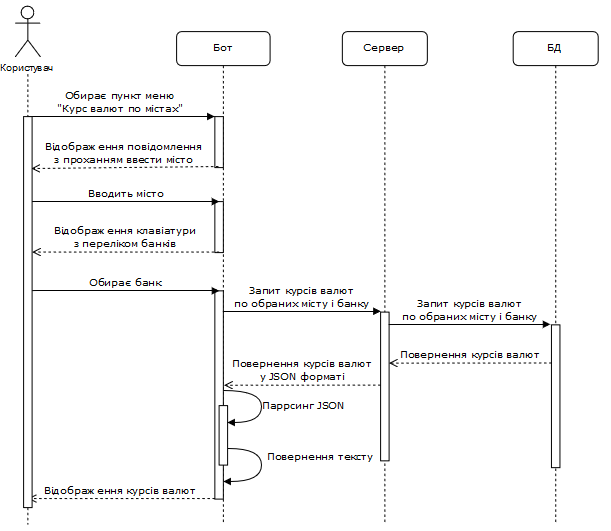


Рисунок 2.4 – Діаграма послідовностей функції «Отримання курсу валют по місту та банку»

На рисунку 2.5 зображена діаграма послідовностей функції «Отримання курсу валют по даті».

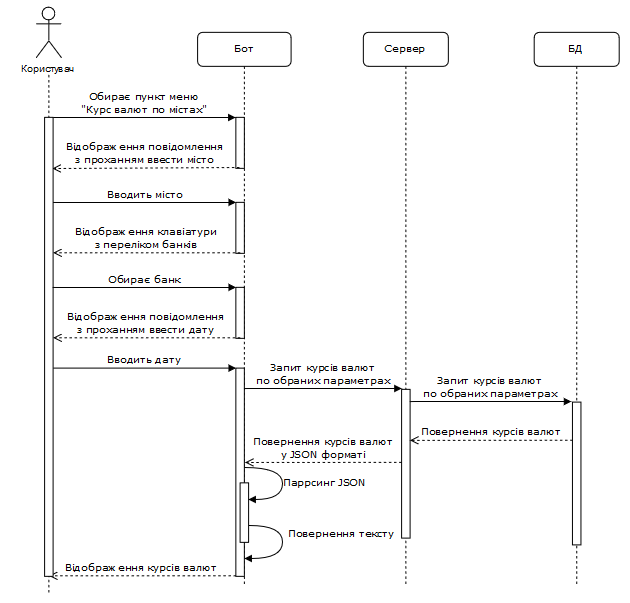


Рисунок 2.5 – Діаграма послідовностей функції «Отримання курсу валют по даті»

На рисунку 2.6 зображена діаграма послідовностей функції «Отримання міжбанкового курсу валют».

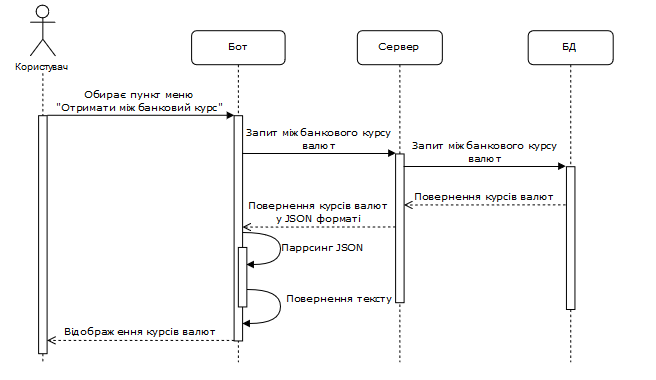


Рисунок 2.6 – Діаграма послідовностей функції «Отримання міжбанкового курсу валют»

На рисунку 2.7 зображена діаграма послідовностей функції «Початок роботи боту».

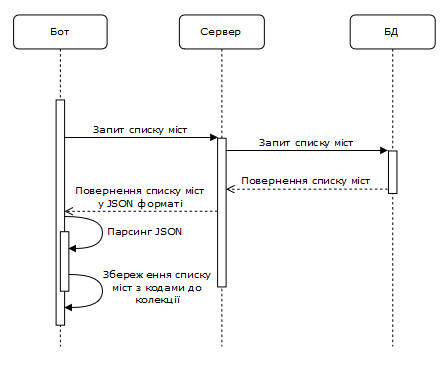


Рисунок 2.7 – Діаграма послідовностей функції «Початок роботи боту»

Для спрощення етапу розробки програмного забезпечення було створено діаграму класів (Static structure diagram).

Діаграма класів – це діаграма, яка демонструє класи системи, їх атрибути, методи та зв’язки між ними.

Створимо діаграму класів серверної частини додатку. Для реалізації цього шару нам необхідні наступні компоненти:

* Репозиторій для доступу до даних;
* Сервіс для отримання даних;
* Сервіс для виконання парсингу курсів валют за розкладом;
* Контроллер – для маппингу запитів.

Для доступу до даних використаємо шаблон проектування «репозиторій». Репозиторій - це абстрактний шар між будь-яким сховищем і вашим додатком або бізнес-логікою.

На рисунку 2.8 зображена діаграма класів для серверної частини додатку.

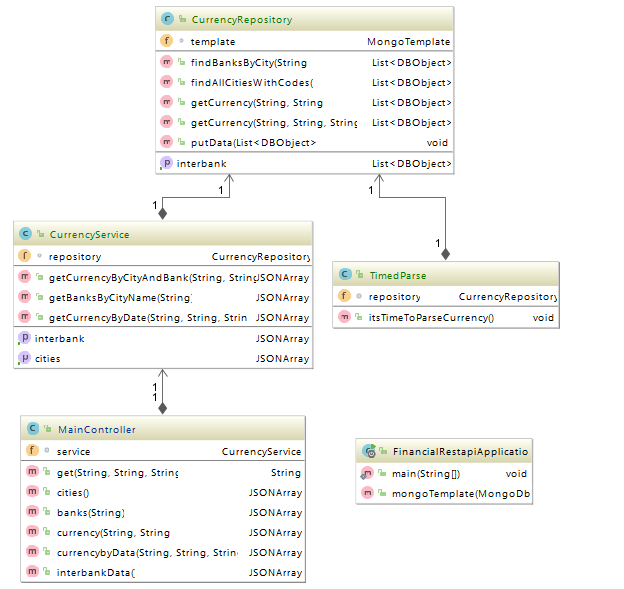


Рисунок 2.8 – Діаграма класів серверної частини додатку

З діаграмою класів для власне бота все доволі просто. Клас «Choice», який призначений для накопичування вибору користувача. Клас «Bot», який має поля для зберігання переліку доступних міст, «langTool» - для пошуку помилок в словах та для пропонування можливих варіантів, «choises» - для збереження виборів користувачів(після виведення курсу валют вибір користувача видаляється).

На рисунку 2.9 зображена діаграма класів бота.

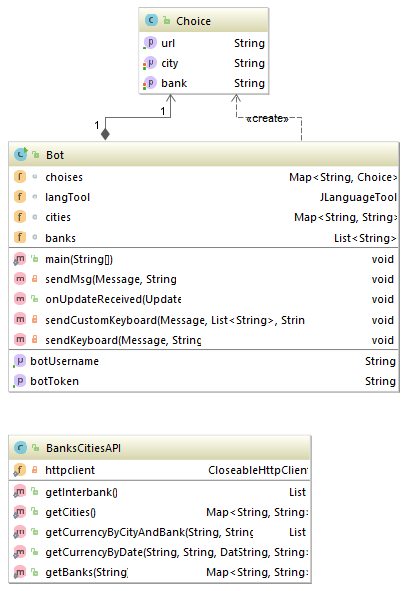


Рисунок 2.9 – Діаграма класів бота

**2.8 Розробка алгоритмів роботи**

Пістя виведення вимог до варіантів використання бота, описаних у діаграмах варіантів використання, послідовностей та классів пункту 2.7 розділу 2, приступимо до розробки алгоритмів роботи програмного забезпечення додатку.

Перше, що необхідно для роботи програмного забезпечення – отримати список доступних міст та зберегти їх у себе. Отже, необхідно їх отримати, виконавши запит.

Далі, програма готова і чекає на дії користувача. Слід виділити наступні варіанти використання:

* Отримати курс валют по місту та банку;
* Отримати курс валют по місту, банку та даті;
* Отримати міжбанковий курс валют.

Команда «Отримати курс валют по місту та банку» полягає у введенні користувачем міста та банку з подальшим запитом бота до REST API з введеною інформацією. Яке, в свою чергу, дані зчитає з БД, поверне їх боту, а бот відобразит їх користувачеві.

Команда отримання курсу валют по місту, банку та даті є майже ідентичною з попереднбою командою, з виключенням того, що потрібно ввести додатково ще й дату.

Команда «Отримати міжбанковий курс валют» полягає у поверненні коритувачеві міжбанкового курсу валют. Ніяких додаткових дій зі сторони користувача не потрібно.

Згідно з вищеописаними вимогами було створено схему алгоритму, приведену на рисунку 2.10.

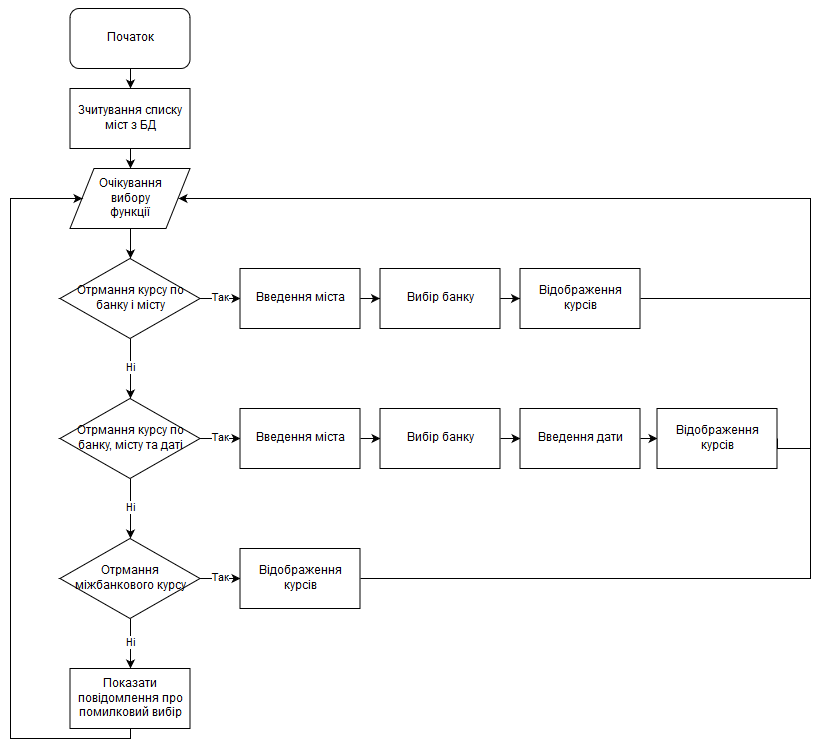


Рисунок 2.10 – Схема алгоритму роботи програмного забезпечення фінансового бота

**3 РЕАЛІЗАЦІЯ ФІНАНСОВОГО ДОДАТКУ ДО ПРОГРАМНОГО ЗАСОБУ TELEGRAM**

Виходячи з разультатів проектування, проведеного на попередньому етапі роботи, можна почати реалізацію апаратного та програмного забезпечення, з якого буде складатися спроектований інтернет-радіоприймач.

**3.1 Реалізація програмного забезпечення**

Згідно з вимогами до програмної частини, описаними у пункті 2.1.1 розділу 2, а також згідно з UML діаграмами та алгоритмами роботи, описаними у пунктах 2.5 та 2.6 розділу 2 відповідно, буде розроблене програмне забезпечення проекту. Розробка програмного забезпечення полягає у реалізації програмного модулю, який об’єднуватиме функціонал інтернет-радіоприймача та можливості апаратної частини платформи (такі як можливості підключення до мережі інтернет, декодування та виведення звуку, робота з інтерфейсом GPIO, тощо).

**3.1.1 Розробка бота**

Бот являє собою один клас який повинен унаслідуватись від класу «org.telegram.telegrambots.bots.TelegramLongPollingBot».

Для роботи потрібно перевизначити як мінімум три методи:

* onUpdateReceived – виконується кожного разу, коли користувач відправляє повідомлення боту;
* getBotUsername – повинен повертати ім’я боту, яке було обране при реєстрації боту;
* getBotToken – повинен повертати секретний токен боту, отриманий раніше, на етапі реєстрації боту.

Лістинг класу, з перевизначеними методами:

**public class** Bot **extends** TelegramLongPollingBot {  
 @Override  
 **public void** onUpdateReceived(Update update) {  
 //тут відбувається опрацювання повідомлення користувача  
 }  
  
 @Override  
 **public** String getBotUsername() {  
 **return null**;  
 }  
  
 @Override  
 **public** String getBotToken() {  
 **return null**;  
 }  
}

Для зберігання поточного вибору користувача використовується сутність Choice. Вона використовується при отриманні курсу по обраних місту та банку.

Лістинг класу Choice:

**public class** Choice {  
  
 **public** Choice(){  
  
 }  
 **private** String **city**;  
 **private** String **bank**;  
  
 **public** String getCity() {  
 **return city**;  
 }  
  
 **public void** setCity(String city) {  
 **this**.**city** = city;  
 }  
  
 **public** String getBank() {  
 **return bank**;  
 }  
  
 **public void** setBank(String bank) {  
 **this**.**bank** = bank;  
 }  
  
}

Для зберігання вибору при отриманні курсу по місту, банку та даті використовується сутність ChoiceWithDate. Він є наслідником класу Choice.

Лістинг класу ChoiceWithDate:

**public class** ChoiceWithDate **extends** Choice {  
 **public** ChoiceWithDate(){  
 **super**();  
 }

**private** Date **date**;  
  
 **public** Date getDate() {  
 **return date**;  
 }  
  
 **public void** setDate(Date date) {  
 **this**.**date** = date;  
 }  
  
}

Для виконання HTTP запитів до API було створено утилітний клас BanksCitiesAPI, в якому створено об’єкт – HTTP клієнт та методи для отримання інформації.

Лістинг частини класу ChoiceWithDate(HTTP клієнт та метод для отримання всіх доступних міст):

**private static** CloseableHttpClient *httpclient* = HttpClients.*createDefault*();  
  
**public static** Map<String, String> getCities(){  
 HttpGet httpGet = **new** HttpGet(**"http://localhost:8081/cities"**);

CloseableHttpResponse response1 = **null**;

**try** {  
 response1 = *httpclient*.execute(httpGet);  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
  
 StringBuilder stringBuilder = **new** StringBuilder();  
 BufferedReader rd = **null**;

**try** {  
 System.***out***.println(response1.getStatusLine());

HttpEntity entity1 = response1.getEntity();

rd = **new** BufferedReader (**new** InputStreamReader(entity1.getContent()));  
 String line = **""**;

**while** ((line = rd.readLine()) != **null**) {  
 stringBuilder.append(line);  
 System.***out***.println(line);  
 }

EntityUtils.*consume*(entity1);  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 } **finally** {  
 **try** {  
 response1.close();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }

JSONParser parser = **new** JSONParser();  
 System.***out***.println(rd.lines().toString());  
 Map<String, String> map = **new** HashMap<>();

**try** {  
 JSONArray jsonObject = (JSONArray) parser.parse(stringBuilder.toString());

jsonObject.forEach(obj -> {  
 map.put(((JSONObject)obj).get(**"code"**).toString(), ((JSONObject)obj).get(**"name"**).toString());  
 });

} **catch** (ParseException e) {  
 e.printStackTrace();  
 }  
 **return** map;  
}

**3.1.2 Розробка REST API**

Цей програмний модуль складається з трьох шарів: контролер, сервіси та репозиторій.

Контролер – відповідає за обробку запитів. Кожен його метод відповідає на запит по конкретному URL.

Лістинг частини класу MainController (метод, який відповідає на запит «/cities»):

@RequestMapping(**"/cities"**)  
**public** JSONArray cities(){  
 **return service**.getCities();  
}

Сервіси – виконують якусь бізнес-логіку. В нашому випадку вони займаються конвертуванням об’єктів з типу DBObject в JSON.

Лістинг частини класу CurrencyService (метод, який повертає JSON зі всіма містами з їх кодами):

**public** JSONArray getCities(){  
 List<DBObject> allCitiesWithCodes = **repository**.findAllCitiesWithCodes();

JSONArray jsonArray = **new** JSONArray();

allCitiesWithCodes.forEach(item ->{  
 jsonArray.add(item.toMap());  
 });  
 **return** jsonArray;  
}

Репозиторій – шар для доступу до даних(в нашому випадку БД).

Лістинг частини класу CurrencyRepository (метод, який дістане з БД всі міста і коди):

**public** List<DBObject> findAllCitiesWithCodes(){  
 BasicDBObject query = **new** BasicDBObject();  
 BasicDBObject fields = **new** BasicDBObject();

fields.append(**"code"**, **"1"**);  
 fields.append(**"name"**, **"1"**);

List<DBObject> cities = **template**.getCollection(**"cities"**).find(query, fields).toArray();

**return** cities;  
}

**3.2 Реалізація інтерфейсу**

Інтерфейс користувача повністю представлений додатком Telegram. Автор використовував офіційну бібліотеку «telegrambots» для формування елементів інтерфейсу бота.

Користувач взаємодіє з додатком методом введення даних к клавіатури, або вибору варіанту на віртуальній клавіатурі додатку.

На рисунку 3.1 зображено стартовий інтерфейс фінансового боту. Для початку користування потрібно обрати потрібну дію і натиснути відповідну кнопку.

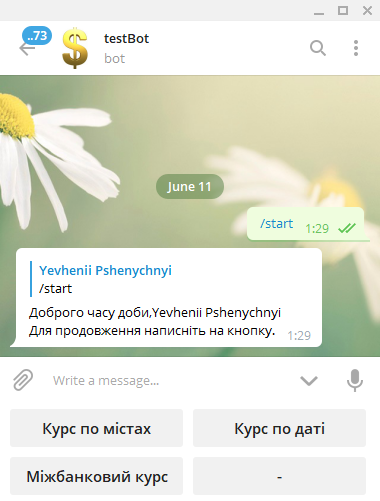


Рисунок 3.1 – стартовий інтерфейс боту

На рисунку 3.2 зображено повідомлення користувачу з проханням ввести назву міста.

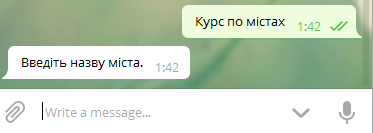


Рисунок 3.2 – Прохання ввести назву міста

На рисунку 3.3 зображено випадок, коли місто введено з помилкою. Бот старається знайти міста, схожі на введене, пропонуючи обрати схожі зі списку, або ввести ще раз.

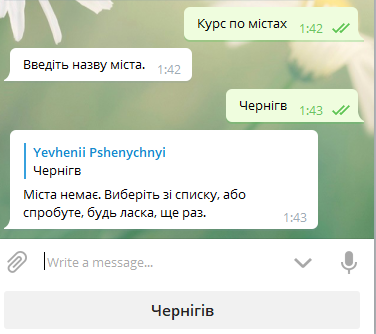


Рисунок 3.3 – Випадок помилки введення міста

На рисунку 3.4 зображено меню вибору банку після успішного введення міста

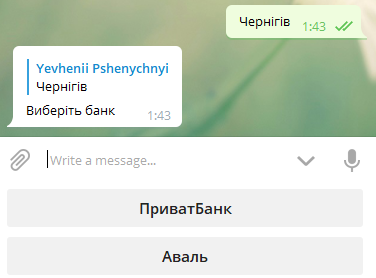


Рисунок 3.4 – Меню вибору банку

На рисунку 3.5 зображено результат виконання операції «Отримання курсу валют по місту і банку».

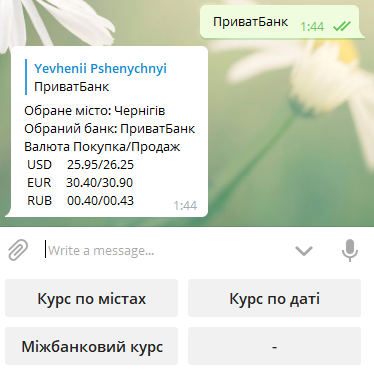


Рисунок 3.5 – Результат виконання операції «Отримання курсу валют по місту і банку»

На рисунку 3.6 зображено результат виконання операції «Отримання міжбанкового курсу валют»

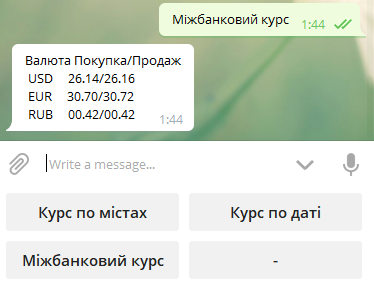


Рисунок 3.6 – Результат виконання операції «Отримання міжбанкового курсу валют»

На рисунку 3.7 зображено прохання ввести дату.

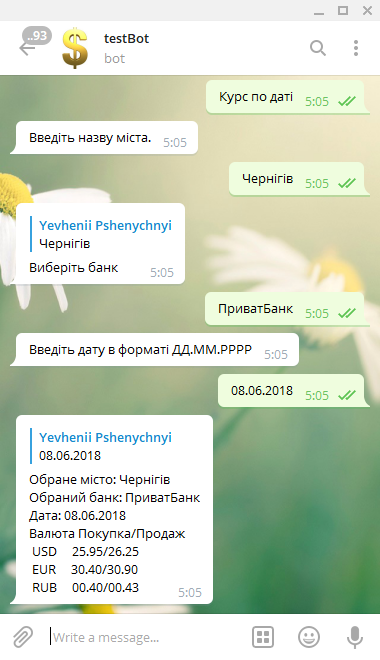


Рисунок 3.7 – Прохання ввести дату

На рисунку 3.8 зображено випадок неправильно введеної дати

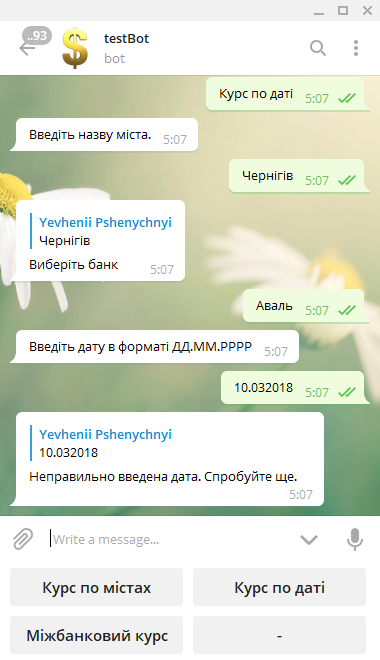


Рисунок 3.8 – Випадок неправильно введеної дати

**3.3 Опис процесу автоматизації збірки та установки додатку**

При розробці бота, що REST API – використовувалась система збірки Maven.

Для того, щоб зібрати проект потрібно виконати команду в консолі «mvn clean package». Це все. При цьому буде очищена директорія збірки, виконані тести, і зібраний виконуваний .jar файл.

Додаток не потребує установки. Для нормального запуску неохідно спочатку запустити REST API, потім слід запускати бот.

Для початку користування ботом користувач повинен у пошуку в своєму додатку Telegram ввести «@for\_course\_project\_bot». На рисунку 3.9 зображено результат ції дії.

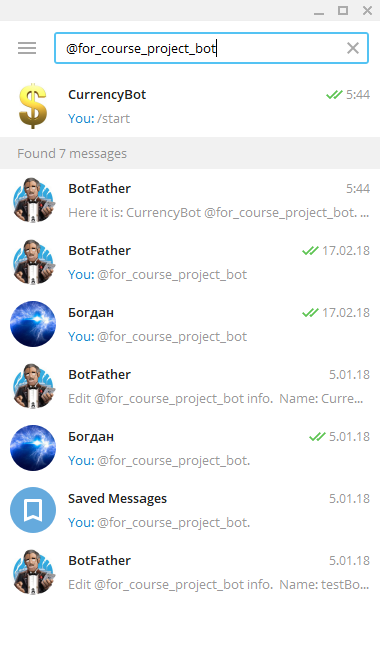


Рисунок 3.9 – Пошук бота в Telegram

Далі потрібно натиснути кнопку «Start» і користуватися ботом. Початковий інтерфейс зображено на рисунку 3.10.

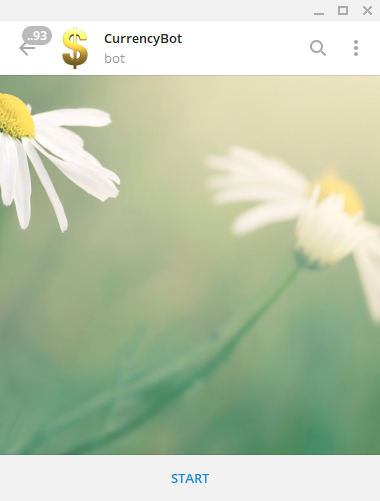


Рисунок 3.10 – Початковий інтерфейс

**ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ**

1. Обзор мессенджеров [Електронний ресурс] – Режим доступа: http://www.voipoffice.ru/tags/messendzhery/
2. Боты [Електронний ресурс] – Режим доступа: https://te-st.ru/2016/06/08/chatbots/
3. 4 преимущества, которые чат-бот может дать интернет-магазину [Електронний ресурс] – Режим доступа: https://www.cs-cart.ru/blog/4-preimushhestva-kotoryie-chat-bot-mozhet-dat-internet-magazinu/
4. Eclipse, NetBeans или IntelliJ IDEA? Выбираем IDE для Java-разработки [Електронний ресурс] – Режим доступа: https://javarush.ru/groups/posts/1642-eclipse-netbeans-ili-intellij-idea-vihbiraem-ide-dlja-java-razrabotki
5. IDE NetBeans - Обзор [Електронний ресурс] – Режим доступа: https://netbeans.org/features/index\_ru.html
6. Java Ant [Електронний ресурс] – Режим доступа: http://alfalavista.ru/idxfldr/2013-06-18-22-25-47/304-java-ant-java-ant.html
7. Руководство по Maven - Что такое Maven [Електронний ресурс] – Режим доступа: http://www.apache-maven.ru/
8. Gradle [Електронний ресурс] – Режим доступа: https://ru.wikipedia.org/wiki/Gradle
9. Spring Boot | Spring по-русски! [Електронний ресурс] – Режим доступа: http://spring-projects.ru/projects/spring-boot/
10. Google Code Archive - Long-term storage for Google Code Project Hosting. [Електронний ресурс] – Режим доступа: https://code.google.com/archive/p/json-simple/
11. Apache HttpComponents – Apache HttpComponents [Електронний ресурс] – Режим доступа: https://hc.apache.org
12. Spring [Електронний ресурс] – Режим доступа: https://spring.io/
13. ﻿Сравнение NoSQL систем управления базами данных [Електронний ресурс] – Режим доступа: http://devacademy.ru/posts/nosql/

**ВИСНОВКИ**

В ході виконання кваліфікаційної роботи бакалавра були вивчені основні питання, пов'язані з фінансовими інтернет-ботами та фінансовими сайтами. В роботі проведено аналіз існуючих фінансових дотатків та сайтів.

Згідно з вимогами для інтерактивного додатку для програмного засобу Telegram була використана трирівнева клієнт-серверна архітектура: в ролі клієнта виступає власне додаток Telegram(веб-застосування, мобільний або десктопний додаток), а в ролі сервера виступає веб-застосування фінансове REST API, яке і взаємодіє з СУБД.

Інтерактивний додаток для програмного засобу Telegram було реалізовано на основі фреймфорку Spring, з використанням мови програмування Java, і Spring Boot(для швидкого старту додатку, який надає нам embedded Tomcat) в середовищі розробки IntelliJ IDEA. Для зберігання і обробки даних інтерактивного додатку для програмного засобу Telegram була обрана СУБД MongoDB.

Були реалізовані всі вимоги до архітектури проекту, а також вимоги до інтерфейсу системи. Всі вимоги до реалізації інтерфейсу додатку також були дотримані. Додаток має простий інтерфейс.

Після виділення вимог та розробки архітектури було зроблено UML діаграми: діаграма варіантів використання та діаграми послідовностей. Також були розроблені діаграми класів, що полегшило розробку.

Подальший розвиток системи може змінюватися за рахунок додавання нового функціоналу, а саме: конвертація валют, надання готівкового курсу валют на чорному ринку, також за рахунок розробки нових елементів інтерфейсу, таких як календар(для більш зручного вибору дати), а також оптимізацію коду для підвищення ефективності роботи додатку.

|  |  |  |
| --- | --- | --- |
| Позначення | Назва | Примітки |
|  |  |  |
|  | **Документація** |  |
| ЧНТУ 143221.001 | Програмний проект |  |
|  | Розробка інтерактивного |  |
|  | фінансового додатку |  |
|  | до програмного засобу |  |
|  | Telegram |  |
|  | Текст програми |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  | Комплекси |  |
|  |  |  |
|  | ОС Windows 10 |  |
|  | Telegram |  |
|  | JetBrains IntelliJ IDEA |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Додатки

Дотаток А

Додаток А

Міністерство освіти і науки УкраЇни

ЧернігівсЬкий національний технологічний університет

Кафедра інформаційних і комп'ютерних систем

**Розробка інтерактивного фінансового додатку до програмного засобу Telegram**

**Програмний проект**

**ЧНТУ 143221**.0**01**

Виконавець

студент гр. КІ-143 Є.В. Пшеничний

Чернігів 2018

Міністерство освіти і науки УкраЇни

ЧернігівсЬкий національний технологічний університет

Кафедра інформаційних і комп'ютерних систем

**Розробка інтерактивного фінансового додатку до програмного засобу Telegram**

**Текст програми**

**ЧНТУ 143202**.0**02**

**Аркушів \_\_**

Виконавець

студент гр. КІ-143 Є.В. Пшеничний

Чернігів 2018