ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інженерії програмного забезпечення

ПОЯСНЮВАЛЬНА ЗАПИСКА

до бакалаврської роботи

на ступінь вищої освіти бакалавр

на тему: «**РОЗРОБКА НАСТІЛЬНОЇ ГРИ «МОНОПОЛІЯ» МОВОЮ JAVA ТА БІБЛІОТЕКОЮ TELEGRAM API**»

Виконав: студент 5 курсу, групи ПП3 51

спеціальності

121 Інженерія програмного забезпечення

(шифр і назва спеціальності)

Кенгерлі Е.Ф.

(прізвище та ініціали)

Керівник Негоденко О.В.

(прізвище та ініціали)

Рецензент_____

(прізвище та ініціали)

Нормоконтроль _____

(прізвище та ініціали)

Київ – 2022

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра Інженерії програмного забезпечення Ступінь вищої освіти - «Бакалавр» Спеціальність - 121 «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри

Інженерії програмного забезпечення
О.В. Негоденко

“_____” _____2022 року

ЗАВДАННЯ

НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

Кенгерлі Ельмара Фаіговича

(прізвище, ім’я, по батькові)

1. Тема роботи: «**Розробка настільної гри «Монополія» мовою Java та бібліотекою Telegram API»**

Керівник роботи Негоденко О.В., к.т.н., доцент ,

(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від “16” лютого 2022 року №22.

2. Строк подання студентом роботи 03.06.2022.

3. Вихідні дані до роботи:

3.1. Існуючі месенджери з можливістю створення ботів;

3.2. Положення побудови ботів через Telegram API

3.3. Науково-технічна література;

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):

4.1. Огляд архітектури існуючих інструментів автоматизації тестування

4.2. Розробка програмного забезпечення для телеграм-бота

4.3. Програмна реалізація додатку
4.4. Приклади використання та тестування системи
4.5. Висновки

5. Перелік графічного матеріалу.

5.1. Титульний слайд
5.2. Мета, об’єкт та предмет дослідження
5.3. Актуальність роботи
5.4. Аналоги
5.5. Порівняння з аналогами
5.6. Технічне завдання
5.7. Програмні засоби реалізації
5.8. Інструменти використані для реалізації
5.9. Архітектура структури програми
5.10. Висновки

6.Дата видачі завдання: 11.04.2022

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1.	Підбір науково-технічної літератури	11.04-14.04	Виконано
2.	Проведення аналізу правил та технік гри “Монополія”	15.04-17.04	Виконано
3.	Визначення обмежень та особливостей програмних застосунків для створення чат-ботів	18.04-21.04	Виконано
4.	Розробка серверної частини програмного застосунку для чат-боту	22.04-25.04	Виконано
5.	Висновки, оформлення роботи	26.04-05.05	Виконано
6.	Розробка демонстраційних матеріалів	07.05-15.05	Виконано
7.	Попередній захист роботи	16.05-01.06	
8.	Подання роботи в деканат	03.06	

Студент _____ Кенгерлі Е.Ф.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Негоденко О.В.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Текстова частина бакалаврської роботи 59 с., 27 рис., 10 літературних джерел.

Об'єкт дослідження - процес розробки настільної гри з використанням бібліотек Telegram API.

Предмет дослідження – технології обробки ігрових запитів та відсилання повідомлень у Telegram чат за допомогою бота.

Мета роботи – спрощення можливості доступу до розважального застосунку без великих втрат на розробку гри та її дизайну за рахунок використання серверного застосунку, створеного на мові програмування Java та бібліотеки Telegram API.

Методи дослідження – методи проектування та розробки ігор, аналізу, збору інформації.

Для реалізації поставленої мети потрібно вирішити наступні завдання:

1. Проаналізувати технічні засоби, що використовуються для розробки та обрати необхідні для створення ігрового додатку.

3. Розробити вимоги до ігрового додатку на основі аналізу переваг та недоліків існуючих додатків

4. Спроектувати та розробити новий додаток на основі аналізу потреб користувачів.

4. Провести тестування гри.

Практичне значення отриманих результатів полягає у можливості розробки ігрових платформ та застосунків на базі звичайних месенджерів без великих витрат і ресурсів.

Галузь використання – ігрові платформи.

ЗМІСТ

ВСТУП

Швидкий розвиток та популярність месенджерів дали початок створенню цілої екосистеми застосунків навколо них. Вони давно вже не використовуються тільки як спосіб текстової комунікації, оскільки мають у собі безліч розважальних медіа та інших інтеграцій, дозволяючих використовувати месенджер як єдиний іструмент для виконання великої кількості завдань. Розробка саме ігрових застосунків для месенджерів тільки розпочинає знаходити свою велику аудиторію, тому не має великої кількості розробників у цій галузі, проте вже має декілька вдалих прикладів реалізацій певних ігор в обмежених умовах текстового застосунку.

Об'єкт дослідження - процес розробки настільної гри з використанням бібліотек Telegram API.

Предмет дослідження – технології обробки ігрових запитів та відсилання повідомлень у Telegram чат за допомогою бота.

Мета роботи – спрощення можливості доступу до розважального застосунку без великих втрат на розробку гри та її дизайну за рахунок використання серверного застосунку, створеного на мові програмування Java та бібліотеки Telegram API.

Методи дослідження – методи проектування та розробки ігор, аналізу, збору інформації.

Для реалізації поставленої мети потрібно вирішити наступні завдання:

1. Проаналізувати технічні засоби, що використовуються для розробки та обрати необхідні для створення ігрового додатку.

2. Розробити вимоги до ігрового додатку на основі аналізу переваг та недоліків існуючих додатків

3. Спроектувати та розробити новий додаток на основі аналізу потреб користувачів.

4. Провести тестування гри.

Аналіз особливостей та обмежень API месенджера та правил настільної гри “Монополія” дозволяє сформулювати вимоги до майбутнього програмного продукту. Серверна частина для Telegram бота повинна бути реалізована для операційної системи Windows та забезпечувати наступні функції:

1. збереження даних про нових гравців і результати існуючих у базі даних;
2. можливість створювати нові ігрові кімнати та розпочинати гру;
3. можливість для адміністратора гри виключати гравців з кімнати за допомогою певної команди;
4. обробка ходу гравця і оновлення на основі нього мапи гри, яка мусить відсилатися у повідомленні;
5. пропуск ходу гравця після бездіяльності протягом певного часу, виключення його з гри після другого пропуску поспіль;
6. можливість перевірити свою статистику та рейтинг серед всіх гравців.

Практичне значення отриманих результатів полягає у можливості розробки ігрових платформ та застосунків на базі звичайних месенджерів без великих витрат і ресурсів.

Галузь використання – ігрові платформи.

1 БОТИ ЯК НОВІТНЄ ДЖЕРЕЛО ПОШИРЕННЯ ПОВІДОМЛЕНЬ ТА ВЗАЄМОДІЇ З ЛЮДЬМИ

1.1 Особливості месенджеру як застосунку

Месенджером називається будь-який програмний застосунок, який забезпечує функцію приватних повідомлень між двома або більше людьми. З кожним днем з'являється велика кількість нових програм для спілкування та обміну повідомленнями, даний вид технологій швидко набуває популярність як спосіб пересилання текстових повідомлень, замінюючи стандартні телефонні СМС служби. Месенджери активно інтегрують у всі сфери нашого життя. Їх застосовують у налагодженні комунікації між клієнтом та бізнесом, створенні автозамовлень, у різних державних установах з ціллю повідомлення населення про різну інформацію, у створенні єдиного “хабу”, де через роботу різних розважальних ботів можна прослуховувати музику, дивитися відео та навіть грати у різні міні-ігри.

Сьогодні є безліч служб обміну миттєвими повідомленнями, багато з яких прив'язані до певних соціальних мереж. Майже всі сучасні месенджери пропонують користувачам власні пропрієтарні програмні клієнти у вигляді окремої програми або програми на основі браузера, які підтримують найпопулярніші операційні системи – Windows, Linux, macOS/iOS та Android. Також месенджери завдяки відкритому або обмежено відкритому API дозволяють стороннім розробникам створювати кастомні клієнти, інтегрувати їх у свої розробки або розробляти додаткові можливості.

Текстове спілкування було головною функцією обміну миттєвими повідомленнями протягом тривалого часу, але зараз це одна з багатьох функцій. Серед інших можливостей:

Доступність. Технології дозволяють користувачам бачити доступність своїх контактів. Багато програм показують, чи знаходяться контакти в режимі онлайн чи офлайн, і чи встановили для себе статус вільний чи зайнятий. Деякі клієнти

дозволяють користувачам встановлювати повідомлення про вихід і надавати детальну інформацію про свою доступність. Під час активного сеансу багато програм в режимі реального часу вказують, коли користувач друкує.

Зображення. Багато месенджерів дозволяють користувачам вставляти зображення та емодзі в повідомлення, та навіть редагувати їх.

Передача файлів. Надсилання та обмін файлами також є стандартною частиною багатьох месенджерів.

Перехід на інші режими зв'язку. Численні додатки для обміну миттєвими повідомленнями дозволяють користувачам миттєвих повідомлень перейти до інших способів спілкування, таких як груповий чат, голосові дзвінки та відеоконференції, у межах програми.

Ігри та розваги. Деякі месенджери включають ігри та інші розважальні додатки. Можна грати у невеликі ігри з друзями, розроблені мовою розмітки HTML5 або текстові ігри, реалізовані через чат-ботів.

Оплата. Відносно новою функцією є можливість проведення транзакцій, вона доступна на основних платформах обміну повідомленнями. Ця функціональність дозволяє людям використовувати одну програму як для комунікаційних, так і для фінансових завдань. Відсутність плати за послуги також робить додатки для обміну повідомленнями вигідними для фінансових застосувань. Такі месенджери як Telegram вже пропонують можливість прив’язування банківських карт багатьох українських банків, та функції створення «лотів» та швидкої оплати картою.

1.2 Можливості месенджеру Telegram

“Telegram” - це безкоштовна, кросплатформна, хмарна служба обміну миттєвими повідомленнями. Застосунок також забезпечує функцію проведення зашифрованих відеодзвінків, VoIP, обмін файлами перегляд відео та прослуховування музики тощо. Сервери Telegram розповсюджені по всьому світу з п’ятьма центрами обробки даних у різних регіонах. Доступні різні клієнтські програми для настільних і мобільних платформ, включаючи офіційні програми для

Android, iOS, Windows, macOS та Linux роблять його універсальною платформою. Також існують офіційна програма Telegram і численні неофіційні клієнти, які використовують протокол Telegram. Усі офіційні компоненти Telegram є відкритим кодом, що може переглядатись та застосовуватись будь-ким.

Telegram надає додаткові наскрізні зашифровані чати. Хмарні чати та групи шифруються між додатком і сервером, тому провайдери та інші сторонні компанії в мережі не можуть отримати доступ до даних. Користувачі можуть надсилати текстові та голосові повідомлення, здійснювати голосові та відеодзвінки, а також ділитися необмеженою кількістю зображень, документів (2 ГБ на файл), розташуванням користувачів, анімованими наклейками, контактами та аудіофайлами. Користувачі також можуть стежити за каналами, якими користувалися такі знаменитості, як Арнольд Шварценеггер та політики, зокрема президент України Володимир Зеленський.

Категорія	Назва	Платформ(-и)	Підтримується	Підтримка секретних чатів	Примітки
Мобільні	Telegram	Android 4.1 і вище	Так	Так	Є підтримка планшетів та смарт-годинників на Wear OS.
	Telegram Messenger	iOS 9.0 і вище	Так	Так	Запущений у серпні 2013 року для iPhone та iPod Touch, перезапущений у липні 2014 року з підтримкою iPad та Apple Watch.
	Telegram X	Android 4.1 і вище	Так	Так	Альтернативний Telegram-клієнт, написаний з нуля, швидший, з плавнішою анімацією інтерфейсу, колірними темами та ефективнішим енергоспоживанням. Написаний з використанням бібліотеки TDLib. Раніше був неофіційним і називався Challegram. Із січня 2018 року офіційний і називається Telegram X.
	Telegram X	iOS 8.0 і вище	Ні	Так	Альтернативний Telegram-клієнт, написаний з нуля, швидший, з плавнішою анімацією інтерфейсу, колірними темами та ефективнішим енергоспоживанням. Написаний мовою Swift.
	Telegram Messenger	Windows Phone 8.1 і вище	Так	Так	
Десктопні	Telegram Desktop	Windows 7 і вище, macOS 10.10 і вище, Linux	Так	Ні	Десктопний клієнт на основі Qt. Windows NT-клієнт — традиційний десктопний застосунок у трьох варіантах: з інсталяцією, portable, застосунок з Windows Store.
	Telegram	macOS 10.11 і вище	Так	Так	Нативний macOS-клієнт.
Вебові	Telegram Web	Веб-браузер	Так	Ні	На основі Telegram API.
	Telegram	Google Chrome і Chrome OS	Так	Ні	На основі Telegram Web, опублікований у Chrome Web Store.
	Telegram React	Веб-браузер		Ні	У розробці.

Рис. 1.1. Сумісність месенджеру Telegram з різними платформами

Telegram об’єднав найкращі характеристики таких популярних месенджерів, як WhatsApp та Snapchat. Як і WhatsApp, Telegram видає інформацію та статус

співрозмовника, якщо його аккаунт є відкритим, має можливість надсилати та передавати тексти, фотографії, відео, голосові повідомлення, інформацію про місцезнаходження, контактні номера та документи. Також є функція відправлення повідомлень у певний час з можливістю обмежити період, коли це повідомлення можна побачити

Telegram реалізовує всі основні функції існуючих нині месенджерів – відправлення повідомлень, створення групових та приватних чатів, надсилання файлів та стікерів, однак він має декілька специфічних відмінностей, що робить його гарним конкурентом серед інших месенджерів.

Найголовнішою перевагою месенджеру Telegram є її унікальна сфокусованість на захищеності даних, шифруванні, та розповсюдженні його API за ліцензією відкритого коду. Завдяки цьому існує численна кількість сторонніх клієнтів, створених на базі офіціального клієнта, із унікальними дизайнами та новими функціями. Також це позволяє використовувати багато пристроїв через один і той же обліковий запис та багато облікових записів на одному пристрої. Усі текстові функції, які включають у себе текстові, приватні та публічні групи та інформаційні канали надійно шифруються. Це означає повну анонімність у мережі та впевненість, що твої дані не будуть вкрадені.

Telegram виконує наскрізну шифровку між співрозмовниками в месенджері за допомогою додаткового функціоналу «Secret Chats». Він створює конфіденційну бесіду між двома людьми без можливості пересилання повідомлень, картинок, відео та файлів у інші чати. Вона також блокує телефону можливість створення скріншотів цього чату, і знищує повідомлення через деякий час.

На рисунку 2.1 зображена функція створення секретного чату.

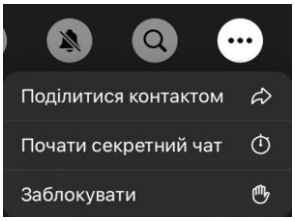


Рис. 1.2. Функція секретного чату у Telegram

Telegram також пропонує наскрізне шифрування для всіх аудіодзвінків та шифрування чатів між двома онлайн-користувачами за замовчуванням. Секретні чати мають режим самознищення.

Telegram також вважається одним з найшвидших та найстабільніших месенджерів на ринку. Повідомлення доставляються без затримки, і навіть в умовах поганого інтернету. Ви можете підключатися з більшості віддалених місць, координувати групи до 5000 членів, а також синхронізувати свої чати на всіх своїх пристроях.

Telegram надає необмежені можливості у використанні пам’яті його серверів. Це означає, що всі повідомлення, зображення, відео та документи будуть збережені у спеціальних хмарних сховищах. Це надає можливість користувачам мати доступ до даних з декількох пристроїв одночасно, не втрачаючи жодних даних, не турбуючись про їх резервне зберігання та відновлення. Це відмінна платформа для зберігання особистих відео та музики, оскільки він має дуже швидкі вбудовані програвачі для цих файлів. Єдиним обмеженням є максимально можливий розмір одного файла – це 2000МБ. Ніяких обмежень на кількість файлів, які можливо завантажити в чатах та каналах.

Важливою функцією безпеки є унікальні нікнейми користувачів. Замість надання людям свого номеру телефону, ви маєте змогу передати свій нікнейм, який не містить жодної персональної інформації.

Telegram є абсолютно безкоштовним додатком без реклами на будь-якій платформі. У найближчому майбутньому плануються свіжі оновлення стосовно розміщення реклами для приватних каналів, які включають рекламу на загальнодоступних каналах, преміум-функції для бізнес-команд та досвідчених користувачів. Це надасть додаткові можливості для творців та власників інформаційних каналів.

Найважливішою для даної роботи характеристикою цього месенджеру є можливість кастомізації чатів та створення так званих чат-ботів.

1.3 Сутність та можливості Telegram ботів

Чат-бот — це програмне забезпечення або комп’ютерна програма, яка імітує людську розмову за допомогою текстових або голосових взаємодій. Чат-боти мають різний рівень складності, вони можуть використовуватись для великого спектру функцій.

Додавання чат-бота до служби або відділу продажів вимагає низького рівня кодування або його відсутності. Багато постачальників послуг чат-ботів дозволяють розробникам створювати розмовні інтерфейси користувача для сторонніх бізнес-додатків.

Через стримкий розвиток сучасних компаній та інтернету з’являються наступні причини використання чат ботів:

• комунікація з цільовою аудиторією та клієнтами та швидка обробка повідомлень;

• розвантаження роботи найманих працівників від одноманітних задач, що заощаджує витрати та час компанії;

• невелика вартість інтегрування свого бізнесу та логіки продаж у чат боти

Telegram використовує звичайний тип бота, який відрізняється від звичайних користувачів лише наявністю в назві префікса «бот». Сам робот розділений на кілька областей:

• Чат-бот - це найпростіший чат, який імітує спілкування на визначену користувачем тему;

• Інформаційні боти - окремий вид бота, основною метою якого є інформування користувача про певні події (новини, публікації тощо);

• Ігрові боти – боти, через яких можна грати у html або текстові ігри

• Боти-помічники - боти, розроблені різними онлайн-сервісами на додаток до основної веб-версії. Насправді чіткого поділу немає, оскільки деякі боти включають кілька механізмів і успішно виконують багато завдань користувача. З їх допомогою ви можете перекладати, навчатися, тестувати, шукати інформацію, грати в ігри і навіть використовувати інші сервіси та взаємодіяти з речами, які

мають доступ до глобальної мережі. Функціонал ботів є безкоштовним у системі Telegram, та має обширний програмний функціонал для інтеграції.

Чат-боти Telegram мають величезну кількість різноманітних елементів інтерфейсу користувача для створити цілісного досвіду спілкування для користувачів. На додаток до перерахованих вище функцій, боти Telegram також підтримують команди та вбудовані запити. Можна з упевненістю сказати, що Telegram є потужнішим каналом обміну повідомленнями з точки зору функцій, які він надає своїм розробникам ботів.

Чат-ботів Telegram легко запустити. Ви можете оприлюднити свого чат-бота менш ніж за 4 кліки, починаючи з чату з гросмейстером усіх чат-ботів Telegram під назвою BotFather. Може знадобитися менше 5 хвилин, щоб ваш чат-бот вийшов і був готовий до ваших клієнтів.

Telegram постійно розвивається та випускає багато нових оновлень як для самого месенджера, так і для розширення можливостей ботів. Боти є зручним і способом для виконання широкого спектру завдань - від автозамовлень у інтернет-магазинах до звичайних інформаційних каналів. Зараз ліміт можливостей для ботів постійно звужується через появу великої кількості інтеграцій та нових розробок.

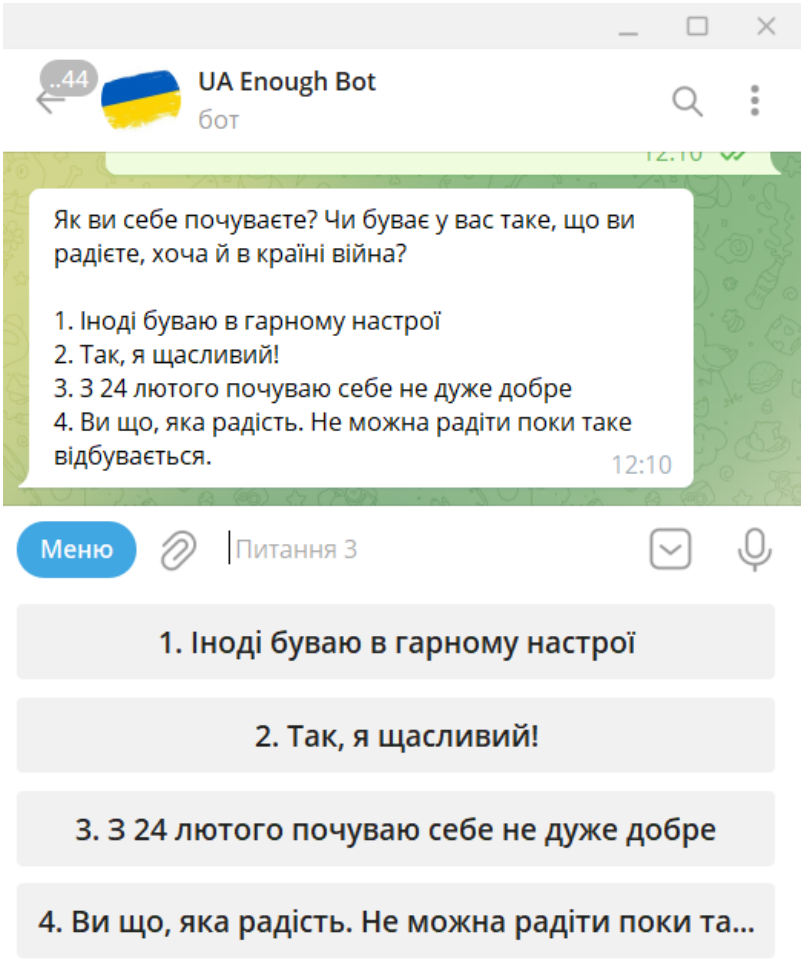


Рис. 1.3. Приклад Телеграм-бота

Боти допомогають у покращенні функціональності месенджеру. Як приклад, можна кожен день надсилати сповіщення рідним та друзям, створювати різні повідомлення, нагадувати про важливі івенти, отримувати інформацію про погоду, грати в ігри тощо

Телеграм-бот може реагувати на ваші повідомлення відповідним образом, та дуже гнучко налаштовуватись. Завдяки телеграм-ботам можно налагодити комунікацію між людьми та різними запрограмованими пристроями. Таким чином, ми можемо запустити та керувати навіть дроном або роботом пилосмоком, використовуючи тільки декілька кнопок.

Телеграм-боти надають можливість швидко надсилати новини, рекламні новинні статті та повідомлення автоматізовано у певні проміжки часу, а отже підходять і для задач, пов’язаних з часовими обмеженнями.

Для компаній різної величини чат-боті можуть стати в нагоді для створення підтримки користувачів, автоматичних відповідей найчастіші запитання, надання персональної інформації користувачів за певними даними без створення для цього окремого сайту.

Телеграм-боти можуть використовуватись для створення електронної інтерактивної газети та присилати оновлення контенту з інших сайтів. Через відповідні API ви можете слідкувати за діями Youtube каналів та новостних сайтів, стягувати музику у файлах з різних джерел, відео та навіть серії улюблених кінострічок.

Телеграм-бот може повністю замінити людину у чаті для надання відповідних інформаційних послуг.

За запитом або через автоматизацію телеграм-бот може надати вам наступну інформацію:

- текстові повідомлення;
- рисунки та зображення;
- відеофайли;
- файли будь-якого формату.

Однією з важливих функцій телеграм-бота є наявність спеціальних команд, за допомогою яких користувач може зручно запустити якусь дію або запитати інформацію, просто натиснувши на цю команду. Приклад такої команди - “/help”, може містити у собі текстову інформацію із усіма командами чат-боту, можливими для виконання даним ботом.

Telegram зберігає всі дані з чатів та ботів у вигляді зашифрованої інформації в хмарному сховищі, тому користувачі можуть не турбуватися про зовнішнє резервне копіювання даних. За своєю політикою Telegram відмовляє спеціальним службам безпеки у наданні будь-якої персональної інформації і є нейтральною у цьому плані, тому він вважається одним з найбезпечніших месенджерів.

Для створення нових телеграм-ботів потрібно пройти через легку реєстрацію за допомогою спеціально створеного бота BotFather. Його можна зайти у додатку Telegram у пошуку за спеціальним тегом – @BotFather. Він є центральним

інструментом для розробки всіх ботів, для реєстрації бота потрібно придумати спеціальний унікальний та зрозумілий тег, який буде ідентифікуватися після реєстрації за допомогою суфікса “bot”

Наприклад, "@ua_enough_bot"

1.4 Порівняння Telegram з іншими месенджерами за можливістю створення ботів

Серед засобів для вирішення поставленої нами задачі на ринку програмного забезпечення є три основні конкуренти - Telegram, WhatApp та Messenger від Facebook. Всі інші месенджери з підтримкою ботів, такі як Skype, Slack, Viber тощо ми розглядати не будемо через те, що їх використовуют загалом тільки у робочих цілях, і середній час, який людина витрачає у них не перевершує одної години в день.

Для того, щоб вирішити, який з трьох месенджерів можна використати для деплою чатботу, потрібно зробити дослідження всіх потрібних функцій та можливостей кожного з них.

WhatsApp Chatbot. WhatsApp пропонує прості, надійні та безпечні швидкі повідомлення та дзвінки, доступні на телефонах у всьому світі. Понад 1,5 мільярда користувачів у всьому світі щомісяця в більш ніж 180 країнах використовують WhatsApp, щоб підтримувати зв’язок з родиною та друзями, будь-де й у будь-який час. Зараз він відкритий для компаній будь-якого розміру з обліковими записами WhatsApp Business і WhatsApp Business API, WhatsApp є найпотужнішим у світі каналом обміну бізнес-повідомленнями.

Довіра, безпека та конфіденційність є ключовими основами екосистеми ботів WhatsApp. Однією з причин, чому так багато людей використовують WhatsApp, є відсутність реклами та спаму. WhatsApp вірить у те, що інтерфейс повинен бути мінімальним, оскільки їхнім остаточним конкурентом завжди були SMS повідомлення.

Рис. 1.4. Приклад чат боту у месенджері WhatsApp

У результаті через наявність такої ідеології, розробники чат-ботів не отримують багато функцій, з якими можна експериментувати, створюючи розмовний досвід користувачів зі своїми чат-ботами в WhatsApp. WhatsApp не підтримує розширені елементів інтерфейсу, такі як швидкі відповіді, кнопки та картки, які є критичними для навігації. Також великим мінусом є те, що чат-ботів WhatsApp не легко запустити. Для запуску чат-бота WhatsApp потрібна інтеграція з постачальником послуг API WhatsApp, наприклад, Twilio або UIB. Процес запуску чат-бота від першого його створення до надання клієнтам може зайняти приблизно 3-4 тижні.

Facebook Messenger Bot. Facebook Messenger має понад 1,3 мільярда активних користувачів щомісяця і є одним з найбагатших каналів обміну повідомленнями, доступних для компаній, щоб охопити клієнтів. Залучайте їх, використовуючи картки, швидкі відповіді та кнопки. Messenger підтримує величезний набір функцій. Facebook Messenger виявляється потужним суперником серед усіх каналів обміну повідомленнями для швидкого розгортання вашого чат-бота.

Facebook Messenger, мабуть, найпростіше запустити. Популярна функція входу через Facebook і надання дозволів допоможе вам налаштувати чат-бота і зробити його доступним для ваших клієнтів менш ніж за 1 хвилину.

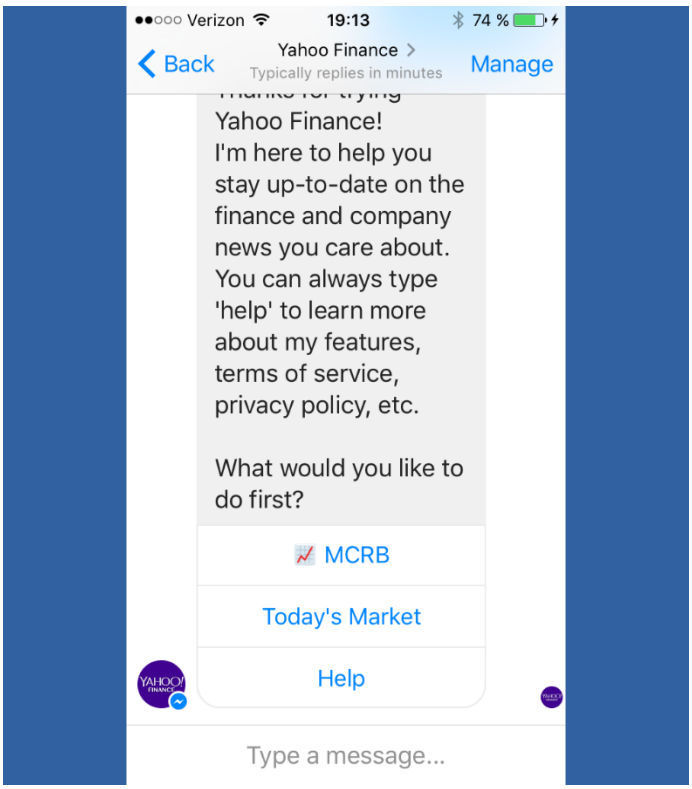


Рис. 1.5. Приклад чат боту у месенджері від Facebook

Зведені результати аналізу характеристик розглянутих додатків наведено у таблиці 3.1.

Таблиця 3.1 – Зведені результати аналізу характеристик додатків для контролю персональної економічної активності

	Telegram	**WhatsApp**	**Facebook**
Текстові повідомлення	+	+	+
Картинки та GIF	+	+	+
Відео	+	+	+
Файли	+	+	+
Відеотрансляції	+	+	+
Функція ‘Quick Replies’	+	-	+
Геолокація	+	-	-
Стікери	+	-	-
Кастомізовані кнопки	+	-	+
Наявність десктопної версії	+	-	+

Якщо підсумувати, то маємо:

- WhatsApp, який є потужним інструментов для розвернення комунікації свого бізнесу, оскільки має дуже велику аудиторію саме для бізнесу, проте обмежений через відсутність розширеного та інтуїтивного інструментарію для створення кастомізованих ботів, чому і не підходить у нашому випадку
- Facebook Messenger, що не містить цього недоліку, проте непопулярність серед українських користувачів та дуже повільний розвиток API викреслює його як підходящого дня нас
- Telegram, що стрімко розвивається, постійно оновлюється, має дуже великі можливості у створенні кастомізованих ботів та легкий у освоєнні, і найголовніше - дуже популярний серед українців та за кордоном серед молоді.

Telegram є найбільш релевантним месенджером для створення розважальних ботів за рахунок своєї широкої аудиторії та можливостей.

1.5 Взаємодія ботів з користувачем

Взаємодія між користувачем і ботом виглядає наступним чином:

Користувач бота віддає йому команду -> Бот передає команду на сервер -> Програма на сервері обробляє отриманий від бота запит -> Сервер віддає відповідь боту -> Бот виводить відповідь на вікні керування користувачеві. Цей цикл повторюється раз за разом, коли ви натискаєте на кнопки і взаємодієте з будь-яким телеграм-ботом.

Спілкування проходить з серверами за допомогою простого HTTPS-інтерфейсу, який є спрощеною версією API Телеграму. Інакше цей інтерфейс можна назвати програмним каталогом або бот-алгоритмом. Детальніше про те, як працює бот в телеграм можна дізнатися, ставши розробником програмного забезпечення. Нові бот-утиліти створюються за допомогою спеціальної утиліти @BotFather, який значно спрощує процес розробки.

Користувачі можуть взаємодіяти з програмами, відправляючи їм повідомлення, команди і запити. У їх числі можуть бути зазначені ключові слова, голосові повідомлення, геолокації. Для початку спілкування з ботом є два способи:

Відправляти команди, відкриваючи чат з ними або додаючи їх в групи. Це корисно для чат-ботів або новинних ботів.

Відправляти запити безпосередньо з поля введення, ввівши ім'я користувача @bot і запит. Це дозволяє відправляти контент з вбудованих ботів безпосередньо в будь-який чат, групу або канал.

На сьогоднішній день навряд чи знайдеться бот, який викличить труднощі навіть у недосвідчених користувачів програми. Вони стають простіше, доступніше і швидше.

1.6 Особливості розробки боту на мові програмування Java

Повідомлення, яке відправляє користувач передається ботом на спеціальне програмне забезпечення, яке обробляє його і дає відповідь боту, який в свою чергу надсилає відповідь користувачеві.

Для реєстрації нового чат-боту потрібно скористуватися послугами боту від розробників Telegram – BotFather. Знайти його можно прямо у месенджері за нікнеймом @BotFather.

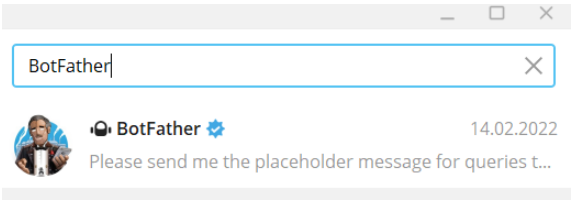


Рис. 1.6. Пошук телеграм-боту Botfather

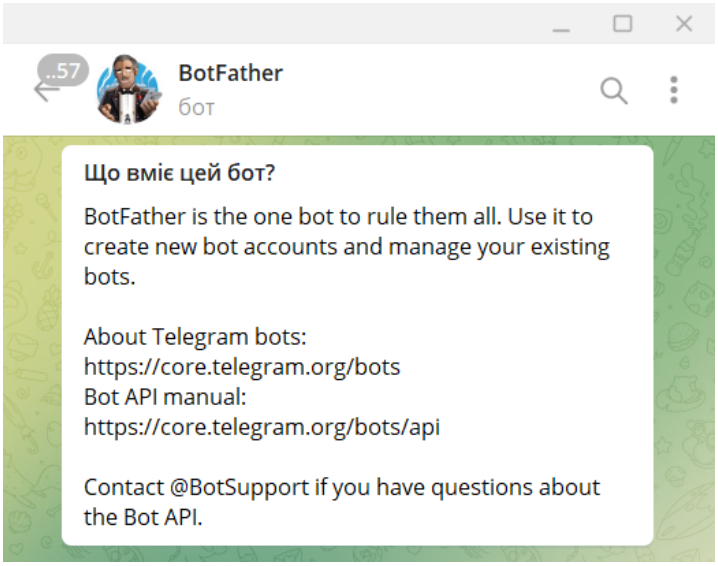


Рис. 1.7. Телеграм-бот BotFather

Почавши діалог з цим ботом необхідно буде надіслати йому команду «/start» - після цього ви отримаєте повідомлення з усіма перерахованими можливостями цього боту, зображене на рисунку 5.

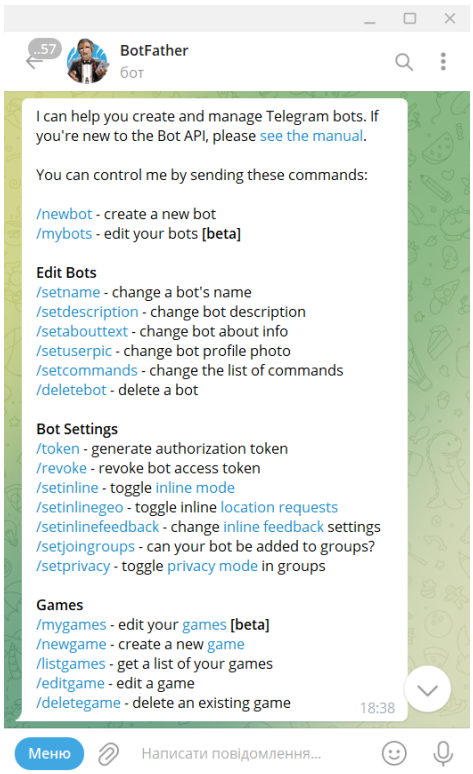


Рис. 1.8. Команди телеграм-бота BotFather

Далі необхідно надіслати команду /newбот .

Після цього з’явиться наступне повідомлення: «Alright, a new бот. How are we going to call it? Please choose a name for your бот.». «БотFather» попросить вас назвати свого бота – для цього потрібно надіслати йому назву.

Назвавши свого бота, ви отримаєте ще одне повідомлення: «Good. Now let's choose a username for your бот. It must end in `бот`. Like this, for example: TetrisБот or tetris_бот.». В ньому проситься вигадати нік-нейм для свого бота, який має обов’язково закінчуватися словом «бот». Зауважу, що цей нік-нейм повинен бути унікальним, тобто, якщо такий нік-нейм уже використовується, то прийдеться вигадати інший. Якщо нік-нейм не використовується, то бот успішно буде створено і ви отримаєте повідомлення, яке інформує вас про його створення та надасть вам усю необхідну для використання боту інформацію (рис. 6).

1.7 Огляд документації Telegram API

Telegram API – це відкрита бібліотека для створення власного телеграм-боту. Вона містить документацію для понад 22 мови програмування, розташовану у відкритому доступі. В ній є пошагові інструкції по створенню нового боту через BotFather та взаємодії з ним, всі доступні REST ендпоінти серверу Telegram, можливі помилки, часті питання та відповіді на них.

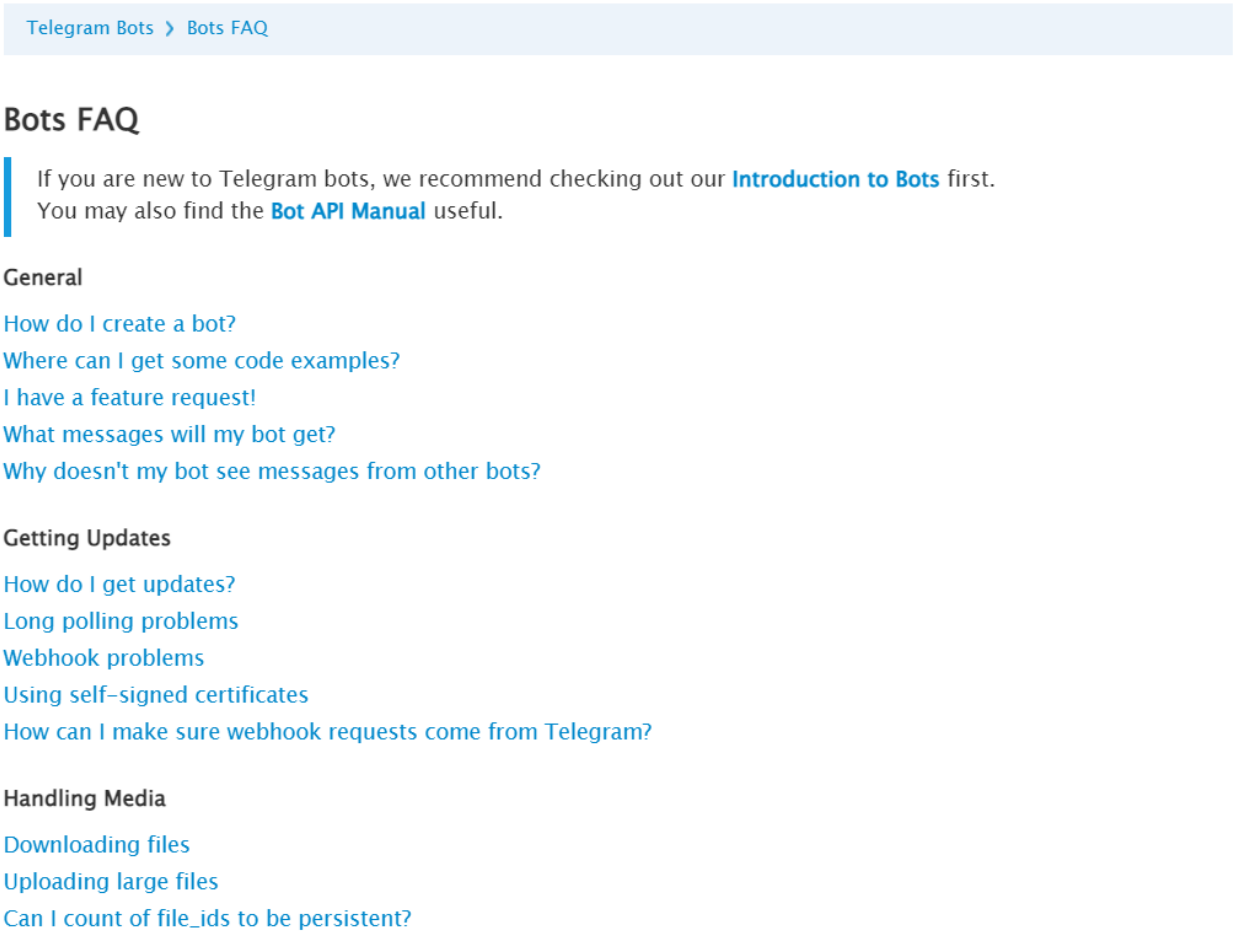


Рис. 1.9. Сайт з документацією Telegram API

Що стосується коду, то для кожної з мов програмування існують відкриті репозиторії на сайті GitHub. Зокрема, якщо розглядати бібліотеку для Java під назвою “Java Telegram Bot API”, то ми маємо дуже багато інформації для розгортання та розробки телеграм-боту саме на цій мові.

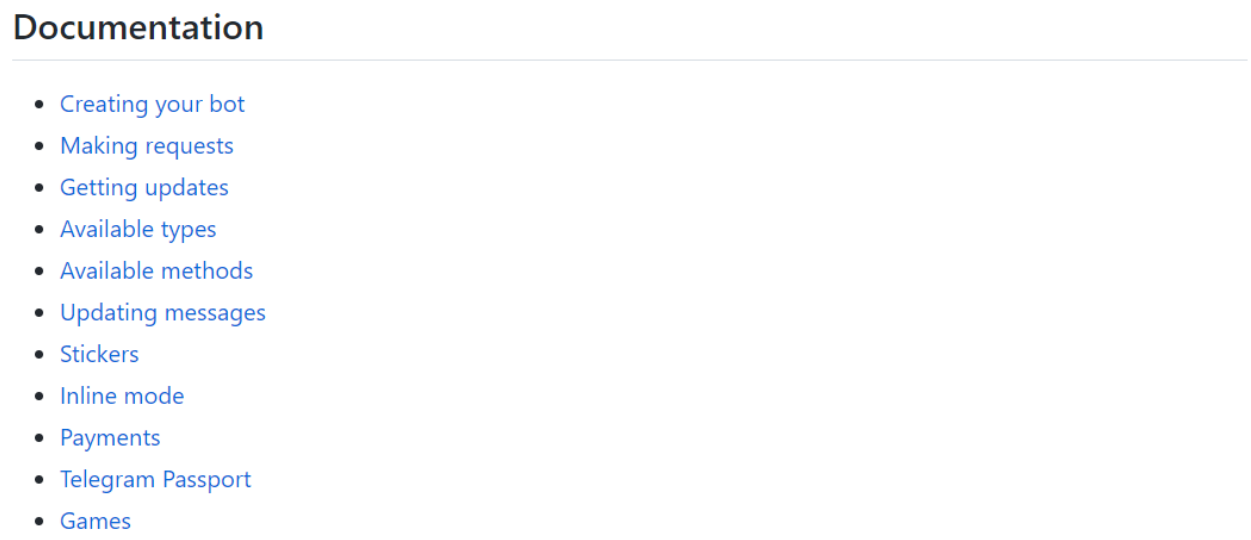


Рис. 1.10. Приклад документації телеграм-боту

1.8 Розробка серверного додатку за допомогою мови Java

Мова програмування Java була розроблена компанією Sun Microsystems і має об'єктно-орієнтовану парадигму програмування. Кінцевий код програми Java перетворюється компілятором javac в спеціальний байт-код для виконання під управлінням віртуальної Java машиною.

Віртуальна Java машина JVM (Java Virtual Machine) – це програма, яка обробляє байт-код і передає інструкції обладнанню як інтерпретатор. Одним з основних переваг даного способу виконання програм є повна незалежність від операційної системи і устаткування, що дозволяє виконувати Java додатки на будь-якому пристрої, для якого існує відповідна віртуальна машина.

Також до важливих особливостей технології Java слід віднести гнучку систему безпеки, в рамках якої виконання програми повністю контролюється 16 віртуальною машиною. Будь-які дії, які порушують встановлені програмою повноваження (наприклад, спроба несанкціонованого доступу до даних або з'єднання з іншим комп'ютером), викликають негайне переривання роботи програми.

До недоліків концепції використання віртуальної машини слід віднести зниження продуктивності, з яким борються різними способами:

- застосування технології трансляції байт-коду в машинний код безпосередньо під час роботи програми – JIT-технологія;
- широкого використання переносних орієнтованого коду (native коду) в стандартних бібліотеках, наприклад SWT;

Сьогодні Java використовується для розробки різних програм, таких як ігри, додатки в соціальних мережах, аудіо та відео додатки тощо. Вона користується високою популярністю і домінує в цій галузі з початку 2000-х до сьогоднішнього дня. З переваг цієї мови програмування - полегшений код через відсутність покажчиків та перевантаження операторів, наявність потужної системи управління пам’яттю, незалежність від платформ, підтримка багатопоточності, обмеженість та структурність коду. Серед недоліків - потреба значного обсягу пам’яті, повільніший час обробку серед інших мов, потреба наявності віртуальної машини для обробки Java коду на пристрої.

Java має величезну кількість бібліотек з відкритим кодом та документацією, також великою перевагою над іншими мовами є строга специфікація стосовно структурності коду, що зменшує кількість потенційних помилок у створенні гарного та чистого коду. Існує бібліотека Telegram Java API, за допомогою якої можна легко створити чат-бота в месенджері Telegram, використовуючи уже готові класи та функції. Також є можливість взаємодіяти з будь-яким REST API та інтегрувати сторонні сервіси.

Telegram має дуже велику бібліотеку для розробки ботів на мові програмування Java. Для того, щоб користатися всіма можливостями, потрібно завантажити бібліотеку до проекту напряму, або скористаючись засобами автоматичного складання проекту, такими як Maven або Gradle.

Gradle:

```
implementation 'com.github.pengrad:java-telegram-bot-api:6.0.1'
```

Maven:

```
<dependency>
  <groupId>com.github.pengrad</groupId>
  <artifactId>java-telegram-bot-api</artifactId>
  <version>6.0.1</version>
</dependency>
```

Рис. 1.11. Імплементація бібліотеки Telegram через Maven

Далі, для розгортання боту потрібно у проекті створити екземпляр класу TelegramBot та передати токен заздалегідь створеного боту. Обов'язково має бути реалізован для цього класу метод setUpdateListener, який викликається кожен раз, коли на сторону бота надходять якісь повідомлення. Після цього бот може бути запущений методом execute.

```
// Create your bot passing the token received from @BotFather
TelegramBot bot = new TelegramBot("BOT_TOKEN");

// Register for updates
bot.setUpdatesListener(updates -> {
    // ... process updates
    // return id of last processed update or confirm them all
    return UpdatesListener.CONFIRMED_UPDATES_ALL;
});

// Send messages
long chatId = update.message().chat().id();
SendResponse response = bot.execute(new SendMessage(chatId, "Hello!"));
```

Рис. 1.12. Приклад телеграм-боту

Отже, процес розгортання боту на Java є досить легким та займає не більше 5-10 хвилин часу.

Java 2D API надає можливості створення та обробки двовимірної графіки, тексту та зображень для програм Java за допомогою розширень до Abstract Windowing Toolkit (AWT). Цей комплексний пакет візуалізації підтримує геометричні фігури, текст і зображення в гнучкому, повнофункціональному фреймворку для розробки більш багатих інтерфейсів користувача, складних програм для малювання та редакторів зображень.

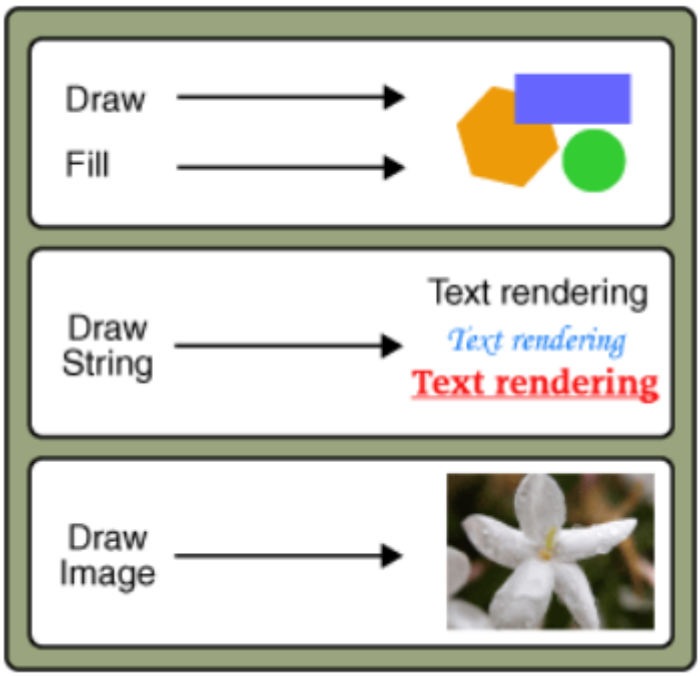


Рис. 1.13. Можливості бібліотеки Java2D

Однією з бібліотек інструментарію Java2D є Graphics2D. Вона забезпечує більш складний контроль над геометрією, перетвореннями координат, керуванням кольором та макетом тексту. Це основний клас для відтворення двовимірних фігур, тексту та зображень на платформі Java. За допомогою нього ми також можемо об’єднувати зображення, обрізати їх та робити певні графічні зміни.

База даних JSON, можливо, є найпопулярнішою категорією в сімействі баз даних NoSQL. Управління базою даних NoSQL відрізняється від традиційних реляційних баз даних, які намагаються зберігати дані за межами стовпців і рядків.

Натомість вони гнучко адаптуються до широкого спектру типів даних, змінюючи вимоги до програм і моделі даних. У часи, коли ліміти можливостей пристроїв зберігання пам’яті є дуже високими, бази даних JSON забезпечують чудовий масштаб і продуктивність.

Ця гнучкість зробила бази даних JSON головною структурою зберігання для систем NoSQL, які підтримують багатомодельну або мультимодальну обробку. Їх популярність пояснюється в основному простотою і гнучкістю структури документів бази даних JSON.

Вперше представлений у 2006 році, JSON розшифровується як «JavaScript Object Notation» і пропонує менш докладний формат даних, ніж популярний XML (eXtensible Markup Language). Бази даних JSON, такі як Couchbase або MongoDB, використовують простий синтаксис стандарту, забезпечуючи структури даних, доступні для читання як людьми, так і машинами.

Бази даних документів JSON зберігають свої дані у файлах, використовуючи конкретні позначення, призначені для усунення жорсткості схем реляційних баз даних. Вони можуть швидше задовольняти нові вимоги до структури даних, отримані після початкового проектування схеми бази даних і випуску програми.

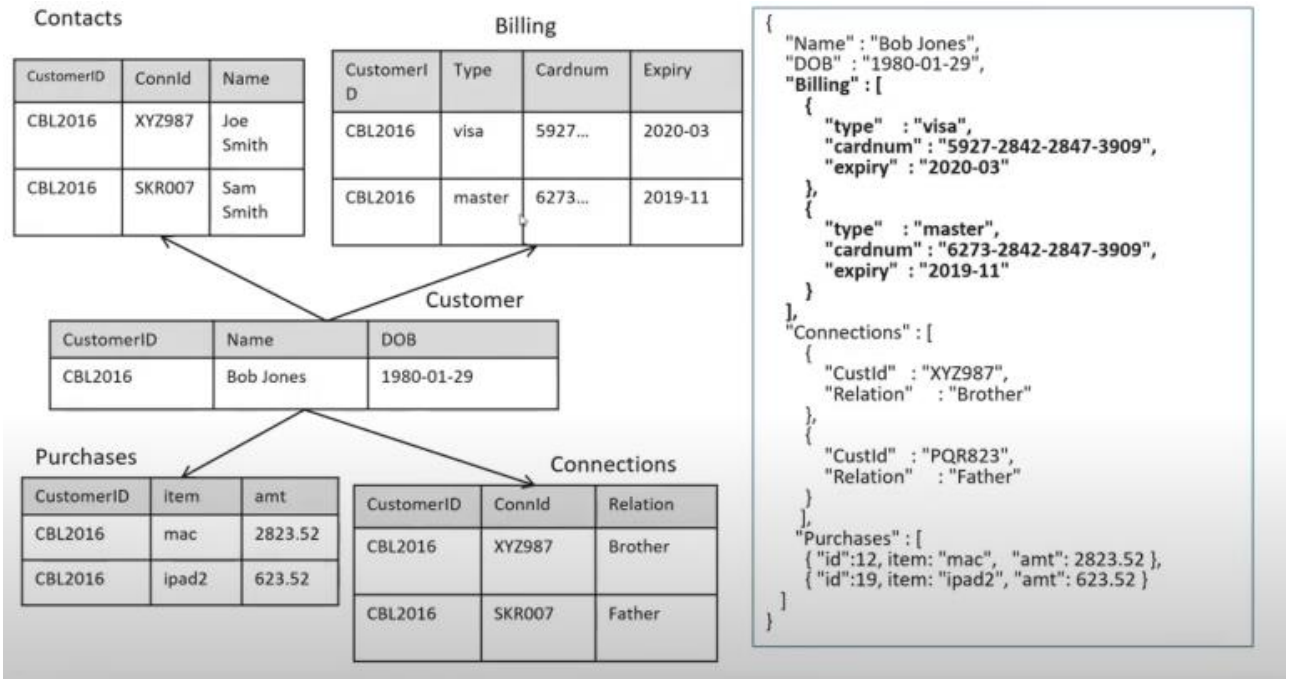


Рис. 1.14. Структура NoSQL бази даних

Рисунок – порівняння між звичайними базами даних та NoSQL

Дані в базі даних JSON легко читати та записувати як для людей, так і для машин завдяки своїй простоті.

Як і JavaScript, документи містять набори імен ключів, пов’язаних з атрибутами або об’єктами. Використання пробілу може зробити документи більш читабельними для людей.

Вирішуючи, яку базу даних використовувати, зазвичай виявляють один або кілька з наступних факторів, які спонукають до вибору бази даних NoSQL:

- Зберігання структурованих і напівструктурованих даних
- Величезні обсяги даних
- Вимоги масштабованості до архітектури програми
- Сучасні парадигми додатків, такі як мікросервіси та потокове передавання в реальному часі

Однією з найкращих NoSQL баз даних є MongoDB.

MongoDB — це документно-орієнтована база даних NoSQL, яка використовується для зберігання великих обсягів даних. Замість використання таблиць і рядків, як у традиційних реляційних базах даних, MongoDB використовує колекції та документи. Документи складаються з пар ключ-значення, які є основною одиницею даних у MongoDB. Колекції містять набори документів і функцій, що є еквівалентом таблиць реляційної бази даних. MongoDB — це база даних, яка з’явилася приблизно в середині 2000-х років.

Особливості MongoDB. Кожна база даних містить колекції, які в свою чергу містять документи. Кожен документ може відрізнятися різною кількістю полів. Розмір і зміст кожного документа можуть відрізнятися один від одного. Структура документа більше відповідає тому, як розробники будують свої класи та об’єкти на відповідних мовах програмування. Розробники часто кажуть, що їхні класи не є рядками і стовпцями, а мають чітку структуру з парами ключ-значення. Рядки (або документи, як вони називаються в MongoDB) не повинні мати заздалегідь визначену схему. Натомість поля можна створювати на льоту.

Модель даних, доступна в MongoDB, дозволяє вам легше представляти ієрархічні зв’язки, зберігати масиви та інші складніші структури.

Середовища MongoDB дуже масштабовані. Компанії по всьому світу працюють із понад 100 вузлами з близько мільйонами документів у базі даних.

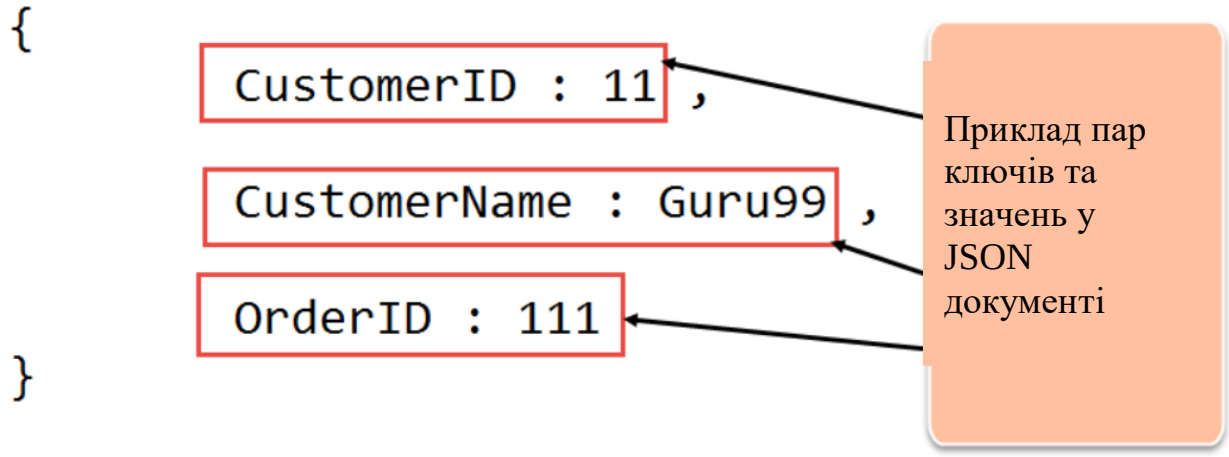


Рис. 1.15. Приклад об’єкту у JSON документі

2 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ТЕЛЕГРАМ-БОТА

2.1 Завдання додатку для Телеграм-боту

Аналіз особливостей та обмежень API месенджера та правил настільної гри “Монополія” дозволяє сформулювати вимоги до майбутнього програмного продукту. Серверна частина для Telegram бота повинна бути реалізована для операційної системи Windows та забезпечувати наступні функції:

- збереження даних про нових гравців і результати існуючих у базі даних
- можливість створювати нові ігрові кімнати та розпочинати гру
- можливість для адміністратора гри виключати гравців з кімнати за допомогою певної команди
- обробка ходу гравця і оновлення на основі нього мапи гри, яка мусить відсилатися у повідомленні
- пропуск ходу гравця після бездіяльності протягом певного часу, виключення його з гри після другого пропуску поспіль
- можливість перевірити свою статистику та рейтинг серед всіх гравців

Мета роботи – спрощення можливості доступу до розважального застосунку без великих втрат на розробку гри та її дизайну за рахунок використання серверного застосунку, створеного на мові програмування Java на базі найпопулярнішого месенджера Telegram та його бібліотеки Telegram API.

2.2 Реєстрація нового боту

Перш за все перед початком роботи над програмною частиною потрібно зареєструвати бота через спеціальний сервіс BotFather. Для цього зробимо наступні кроки:

- Знайдемо утиліту BotFather у клієнті Telegram за спеціальним нікнеймом @BotFather. Почнемо

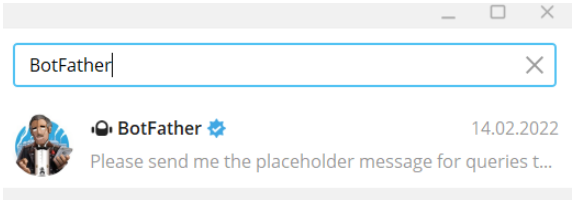


Рис. 2.1. Телеграм-бот BotFather

- Через команду /newbot ініціювати створення нового боту, передати ім’я – MonopolyGameOnlinebot у нашому випадку

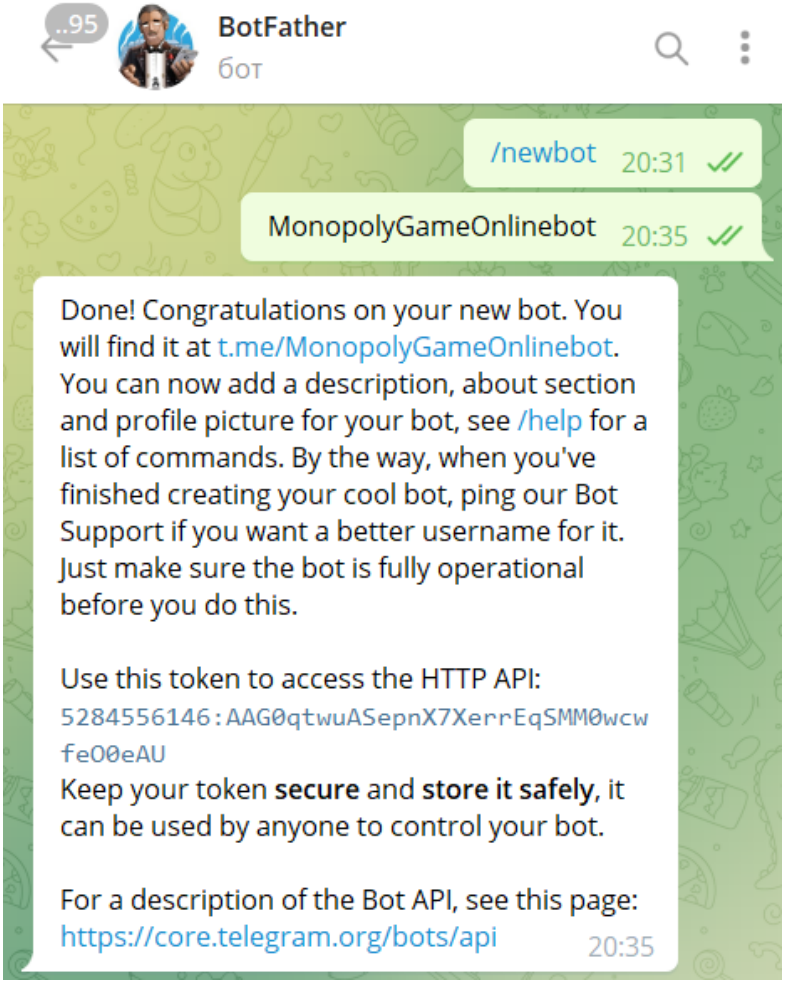


Рис. 2.2. Реєстрація нового боту

- Тепер збережемо згенерований HTTP API токен, який знадобиться нам у реалізації коду телеграм-бота

2.3 Моделювання об’єкту проектування

Діаграма UML - це діаграма, заснована на UML (Unified Modeling Language) з метою візуального представлення системи разом з її основними акторами, ролями, діями, артефактами або класами, щоб краще зрозуміти, змінити, підтримувати або документувати інформацію про систему.

UML — це абревіатура, що розшифровується як уніфікована мова моделювання. Простіше кажучи, UML — це сучасний підхід до програмного забезпечення моделювання та документування. Насправді, це один з найпопулярніших методів моделювання бізнес-процесів.

Він заснований на схематичних представленнях програмних компонентів. Використовуючи візуальні уявлення, ми можемо краще зрозуміти можливі недоліки або помилки в програмному забезпеченні або бізнес-процесах.

UML був створений в результаті хаосу навколо розробки програмного забезпечення та документації. У 1990-х роках існувало кілька різних способів представлення та документування програмних систем. Виникла потреба в більш уніфікованому способі візуального представлення цих систем, і в результаті в 1994-1996 роках UML був розроблений трьома інженерами-програмістами, які працювали в Rational Software. Пізніше він був прийнятий як стандарт у 1997 році і з тих пір залишається стандартом.

Діаграми варіантів використання описують функції високого рівня та область застосування системи. Ці діаграми також визначають взаємодії між системою та її акторами. Варіанти використання та дійові особи на діаграмах варіантів використання описують, що робить система і як учасники її використовують, але не те, як система працює всередині.

Діаграми варіантів використання ілюструють і визначають контекст і вимоги або всієї системи, або важливих частин системи. Ви можете змоделювати складну

систему за допомогою однієї діаграми варіантів використання або створити багато діаграм варіантів використання для моделювання компонентів системи.

На рисунку 3.1 ви можете побачити діаграму варіантів використання нашого серверного доданку через взаємодію з телеграм-ботом.

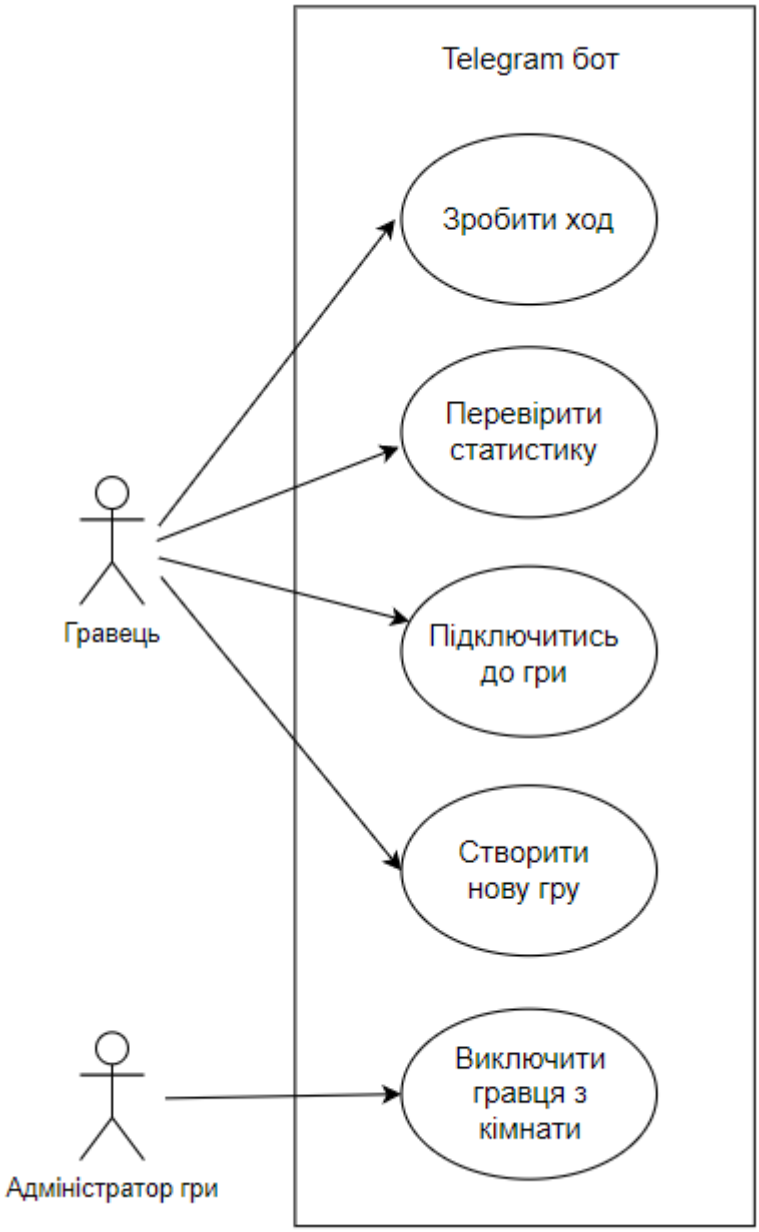


Рис. 2.3. Діаграма прецендентів

У розробці програмного забезпечення діаграма класів на уніфікованій мові моделювання (UML) — це тип діаграми статичної структури, яка описує структуру

системи, показуючи класи системи, їхні атрибути, операції (або методи) та зв’язки між об’єктами.

Діаграма класів є основним будівельним блоком об'єктно-орієнтованого моделювання. Використовується як для загального концептуального моделювання структури програми, так і для детального моделювання, переведення моделей у програмний код. Діаграми класів також можна використовувати для моделювання даних. Класи на діаграмі класів представляють як основні елементи, взаємодії в програмі, так і класи, які будуть запрограмовані.

Для подальшого опису поведінки систем ці діаграми класів можуть бути доповнені діаграмою станів або автоматом станів UML.

Діаграми класів є основним будівельним блоком будь-якого об'єктно-орієнтованого рішення. Він показує класи в системі, атрибути та операції кожного класу та відносини між кожним класом.

У більшості інструментів моделювання клас складається з трьох частин. Ім’я вгорі, атрибути посередині та операції чи методи внизу. У великій системі з багатьма пов’язаними класами класи групуються разом для створення діаграм класів. Різні відносини між класами показані різними типами стрілок.

Нижче наведено зображення діаграми класів. Перейдіть за посиланням нижче, щоб отримати додаткові приклади діаграм класів або миттєво розпочніть роботу з нашими шаблонами діаграм класів.

При розробці системи ряд класів ідентифікуються та групуються разом у діаграмі класів, яка допомагає визначити статичні відносини між ними. При детальному моделюванні класи концептуального проектування часто поділяють на підкласи.

2.4 Підготовка інструментів розробки

IntelliJ IDEA — це спеціальне середовище розробки Java, створена компанією JetBrains. Це одна з найкращих IDE Java. Intellij IDEa зосереджена на продуктивності розробників і має дуже ергономічний дизайн. Він забезпечує

підтримку мов на основі JVM, таких, як Groovy і Kotlin. Для виконання різних вимог користувача IntelliJ IDEA поставляється в 2 варіантах; Ultimate and Community Edition. IDE доступні для Windows, macOS та Linux.

Community Edition — це безкоштовна версія IntelliJ з відкритим вихідним кодом. Він призначений для любителів, розробників-аматорів і студентів. Відкритий код означає, що його вихідний код можна налаштувати відповідно до вимог.

IntelliJ Ultimate — це комерційний, готовий до виробництва аватар IntelliJ IDEA. Він має всі функції, доступні в Community Edition, а також широкий набір розширених функцій, таких як інструменти профілювання та підтримка інструментів SQL та баз даних, а також негайну підтримку клієнтів. Він призначений для використання у виробничих і професійних середовищах.

Щоб допомогти викладачам і учням, JetBrains пропонує спеціальний налаштований варіант Java IDE, який отримав назву IntelliJ IDEA Edu. Як і Community Edition, він є безкоштовним і відкритим. Expedia, Netflix, Samsung, Twitter і Volkswagen є одними з відомих клієнтів IntelliJ IDEA.

Також, сильною стороною Intellij IDEa є її інтерфейс. Він може повністю змінюватись під потреби розробника, від кольору коду, що набирається, до зміни положення, розміру та змісту всіх модулей середовища. Приклад інтерфейсу можно побачити на рисунку 4.1.

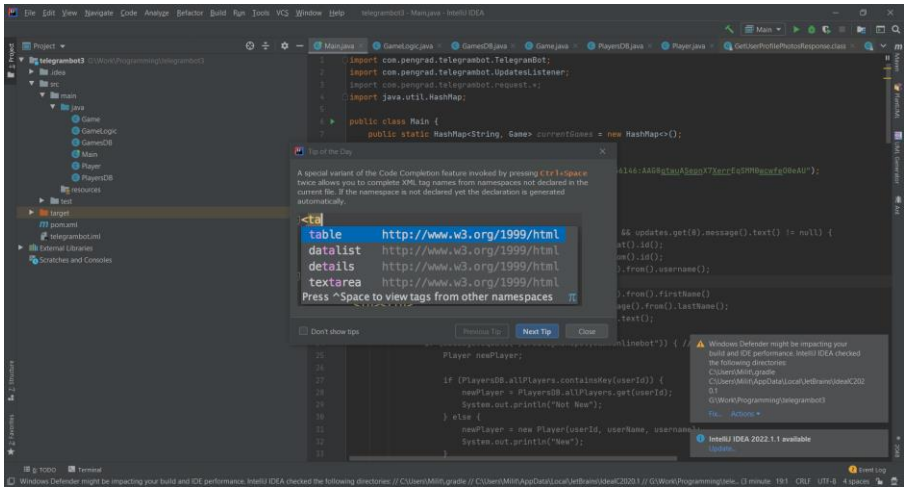


Рис. 2.4. Інтерфейс Intellij IDEa

Maven — це популярний інструмент з відкритим кодом, розроблений Apache Group для створення, публікації та розгортання кількох проектів одночасно для кращого керування проектами. Цей інструмент дозволяє розробникам створювати та документувати структуру життєвого циклу.

Maven написаний на Java і використовується для створення проектів, написаних на C#, Scala, Ruby тощо. Цей інструмент, заснований на об'єктній моделі проекту (POM), полегшив життя розробників Java під час розробки звітів, перевірок і автоматизації тестування. налаштування.

Під час розробки проекту ми стикаємося з багатьма проблемами, такими як:

1) Додавання набору Jars файлів в кожен проект: для розгортання розробки та деплою застосунків нам потрібно додати набір файлів jar в кожен проект. Він також повинен включати всі залежності jar.

2) Створення правильної структури проекту: ми повинні створити правильну структуру проекту в сервлеті, стійках тощо, інакше вона не буде виконана.

3) Створення та розгортання проекту: ми повинні створити та розгорнути проект, щоб він міг працювати.

Maven спрощує вищезгадані проблеми. Виконує в основному наступні завдання.

- Дозволяє легко створити проект із усіма потрібними бібліотеками, без попередньої підготовки у вигляді додавання необхідних jar файлів у проект та налаштування залежностей
- Забеспечує уніфікований процес збірки (проект maven може використовуватися всіма проектами maven)
- Надає інформацію про проект (документацію, джерела з перехресними посиланнями, список розсилки, список залежностей, звіти модульного тестування тощо).

Maven наповнений багатьма цінними та корисними функціями, що значною мірою пояснює його популярність. Ось деякі з більш примітних функцій Maven:

- Величезне, постійно зростаюче сховище бібліотек користувачів
- Можливість легко налаштовувати проекти, використовуючи передовий досвід

- Управління залежностями з автоматичним оновленням
- Зворотна сумісність із попередніми версіями
- Сильні звіти про помилки та цілісність
- Автоматичне встановлення батьківських версій
- Забезпечує послідовне використання в усіх проектах

У проекті за допомогою інтрументу Maven ми зтягнемо всі необхідні нам додаткові бібліотеки. Для цього при створенні проекту в Intellij Idea потрібно обрати Maven основним інструментом збірки та надати проекту ім’я, у нашому випадку – telegramMonopolyBot.

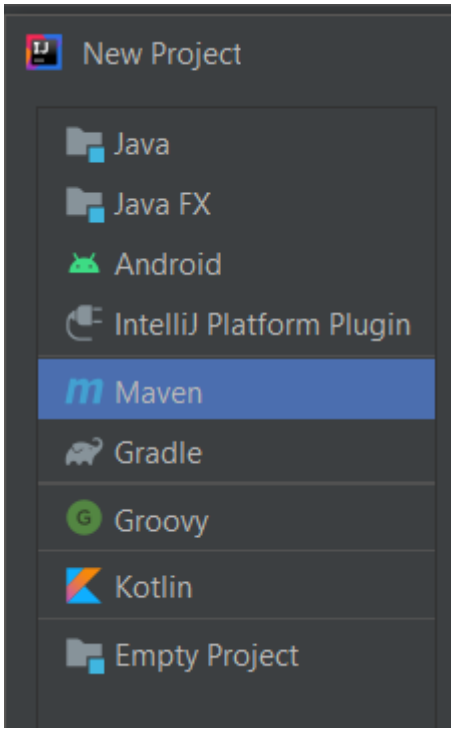


Рис. 2.5. Побудова нового проекту

Далі, для налаштування та підтягування всіх необхідних бібліотек, потрібно відкрити директорію проекту та перейти до файлу pom.xml.

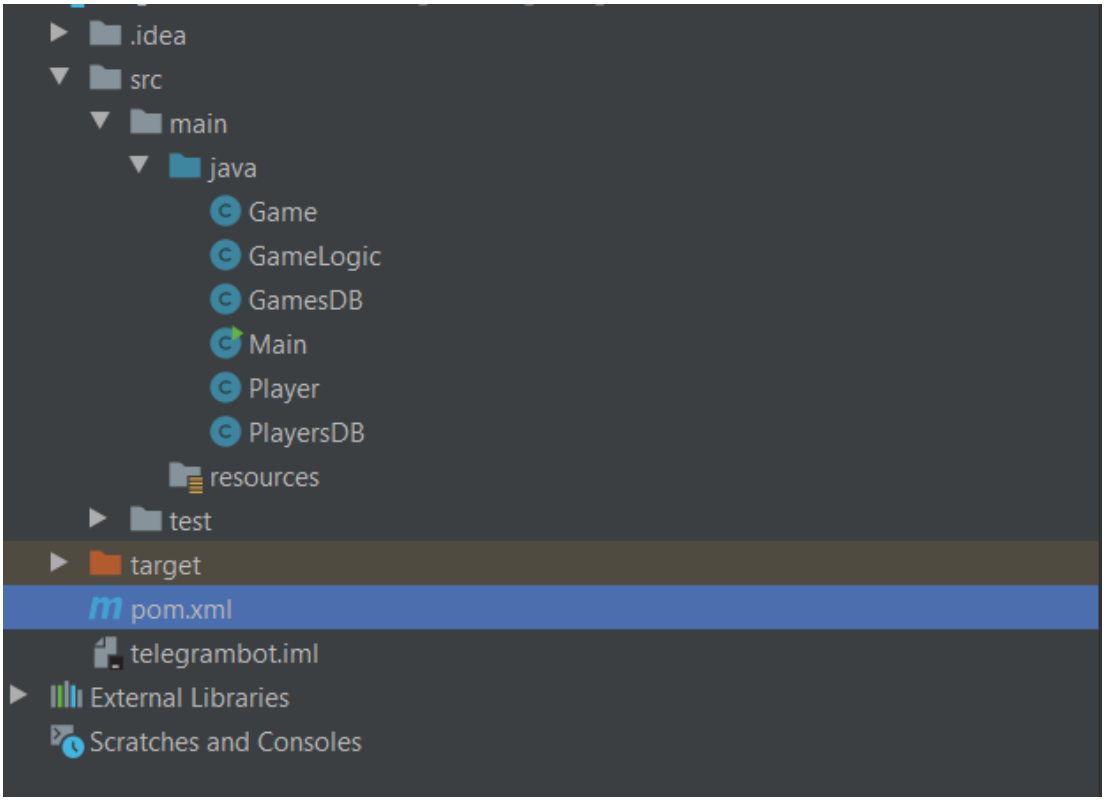


Рис. 2.6. Розташування файлу pom.xml

Для додавання нової залежності, нам потрібно додати до коду новий тег dependencies, та кожну бібліотеку обгортати у додатковий тег dependency. Для кожної залежності треба ввести наступну інформацію

- groupId, це назва директорії Maven, в якій знаходиться потрібна бібліотека
- artifactId, це назва бібліотеки
- version, це номер версії, яку нам потрібно витягнути

```
<dependencies>
    <dependency>
        <groupId>com.github.pengrad</groupId>
        <artifactId>java-telegram-bot-api</artifactId>
        <version>5.7.0</version>
    </dependency>
</dependencies>
```

Рис. 2.7. Dependency для Telegram API

Наступним кроком ми повинні додати залежність для бази даних MongoDB. Для цього додаємо dependency, який надано в офіційній документації.

```
<dependency>
    <groupId>org.mongodb</groupId>
    <artifactId>mongo-java-driver</artifactId>
    <version>3.4.1</version>
</dependency>
```

Рис. 2.8. Dependency для MongoDB

На даному етапі це всі залежні бібліотеки, потрібні нам для розробки серверного застосунку.

Кожна розробка потребує попереднього планування всіх етапів для ефективного розподілу часу. Для цього існує багато методологій та безкоштовних інструментів для розбиття завдань на окремі підзавдання та розстановки приоритетів. Одним з таких інструментів, який використовувався у розробці бакалаврської роботи, є Trello.

Trello — це програма для спільного керування роботою, призначена для відстеження командних проектів, виділення завдань, що виконуються, показу, кому вони призначені, і детального опису прогресу на шляху до завершення.

По суті, Trello спирається на принципи проектних дощок Kanban для візуалізації робочих процесів, надаючи менеджерам і членам команди простий огляд проекту від початку до кінця. Ключовими компонентами Trello є дошки, списки та картки.

Дошки є відправною точкою і, як правило, зосереджені на масштабному проекті, як-от запуску нового веб-сайту, або на процесних завданнях, таких як адаптація співробітника. На кожній дошці може бути створено кілька списків, які вказують на прогрес проекту; Списки «зроблено», «виконується» та «зроблено» є типовими прикладами. Окремі картки в списках містять інформацію про конкретне

завдання, і їх можна переміщати зі списку до списку за потреби (наприклад, коли завдання виконано).

Кожна картка може містити широкий спектр інформації про завдання, включаючи текстовий опис, вкладені файли, засоби автоматизації, коментарі тощо.

Для роботи було створено нове середовище під назвою “Monopoly Development”, 4 основних стовбці для карток і один додатковий. Було вирішено розбити розробку на декілька ітерацій, кожна з них відповідала за версію 0.x.0, і вбирала у себе конкретну сукупність пов’язаних між собою підзавдань, які приводять до конкретного кінцевого релізу. Найперша версія, наприклад, мала у собі основний набір завдань для створення робочого прототипу гри, без витрат часу на графічний дизайн, додаткових функцій та оптимізації. Також, кожна версія розподілялася на певну кількість оновлень, які маркувалися підверсіями 0.x.n, де x – це версія релізу, а n – одне з оновлень.

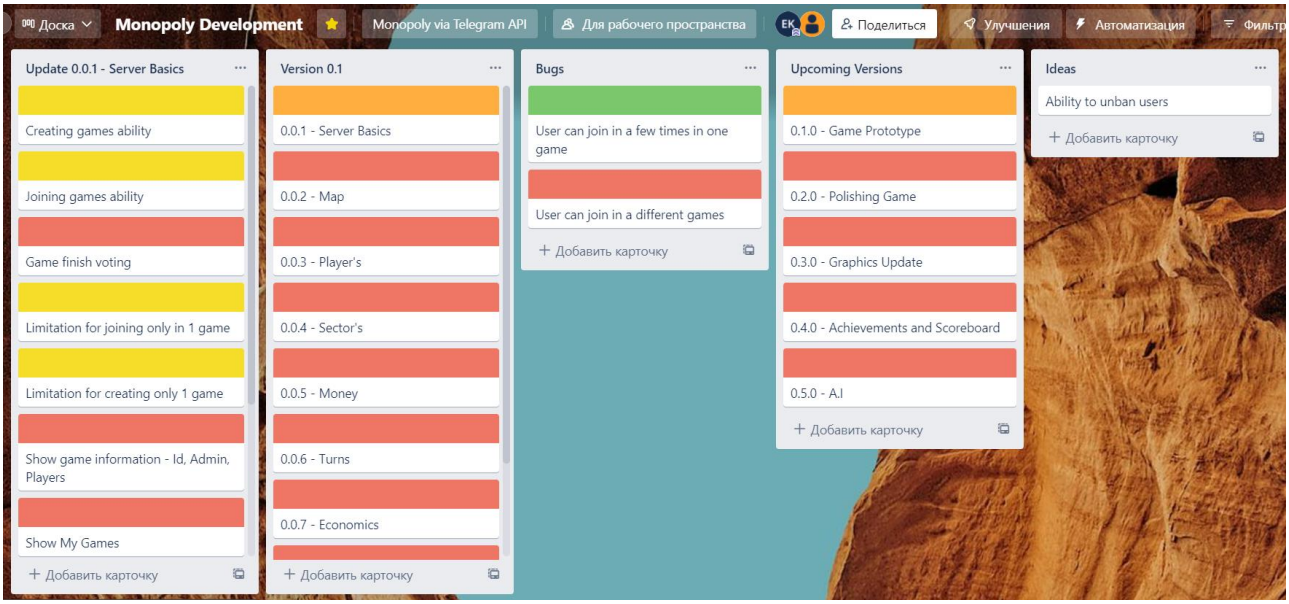


Рис. 2.9. Середовище проекта у Trello

Перший стовбець відповідає за всі завдання, які включає у себе конкретне оновлення. Ці завдання являють собою мінімальну одиницю у розробці та відповідні за конкретний механізм у грі. Як приклад – розробити можливість для адміністраторів виключати гравців із створеної партії.

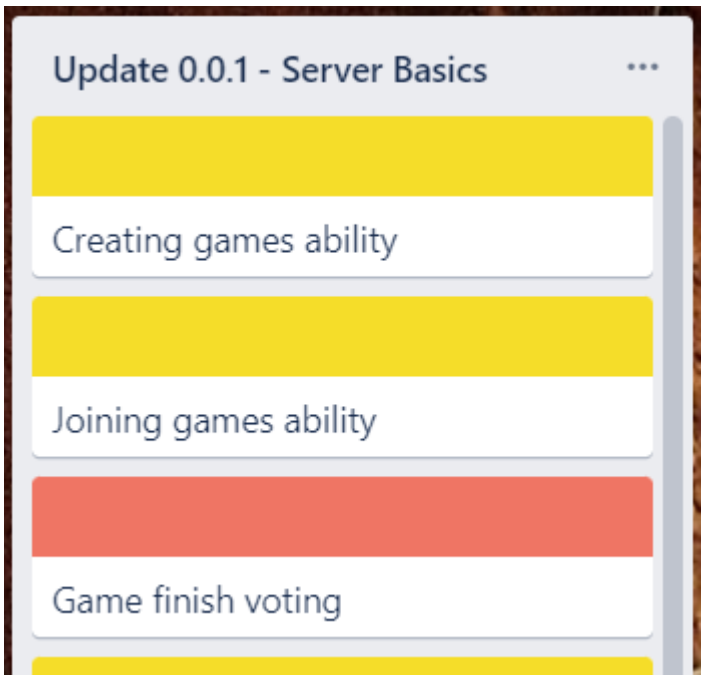


Рис. 2.10. Приклад першого стовбця

Кожне завдання помічається певним кольором:

- червоний, це завдання, що не було почате, аналог To Do
- помаранчевий, це завдання, що на даний момент часу знаходиться у прогресі, аналог In Progress
- жовтий, це завдання, що є закінченим і функціонуючим, проте потребує певне доопрацювання або оптимізацію
- зелений, це завдання, що вже зроблено, аналог Done

Наступний стовбець відображає конкретну версію розробки та включає в себе всі оновлення, що плануються для неї.

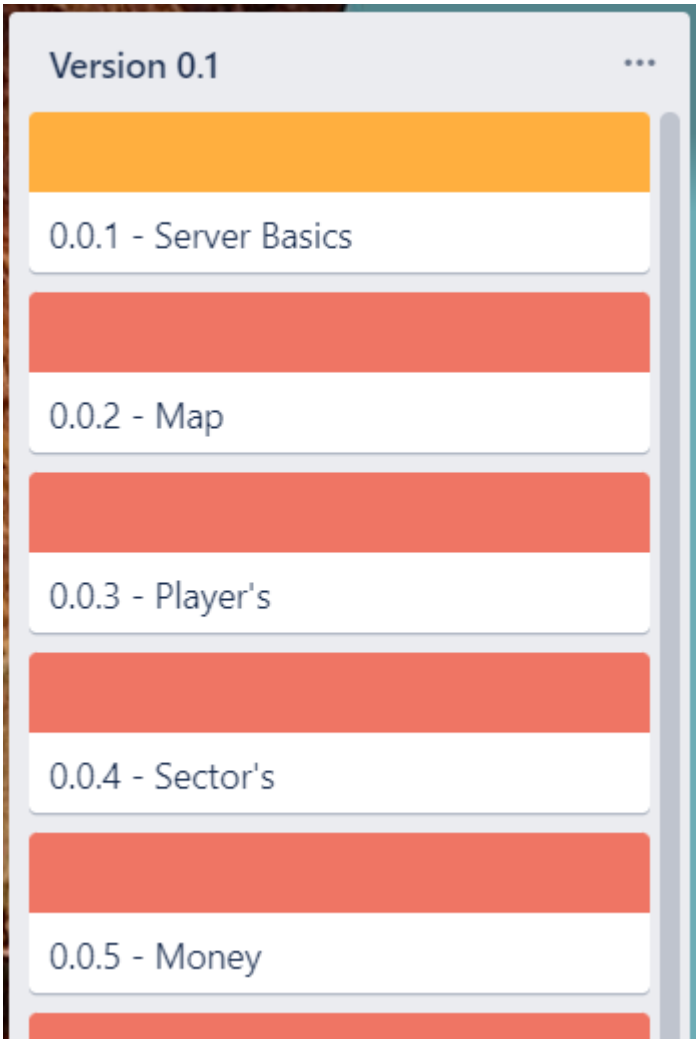


Рис. 2.11. Приклад другого стовбця

Кожне оновлення носить у собі певне ключове слово, що означає його сутність. Прикладом є оновлення 0.0.5, в якому має бути реалізована грошова система гри.

У третьому стовбці додаються всі помічені у розробці помилки, або баги, які потрібно виправити. Вони прив’язані до актуальної версії розробки, тобто повинні бути виправлені до наступної ітерації.

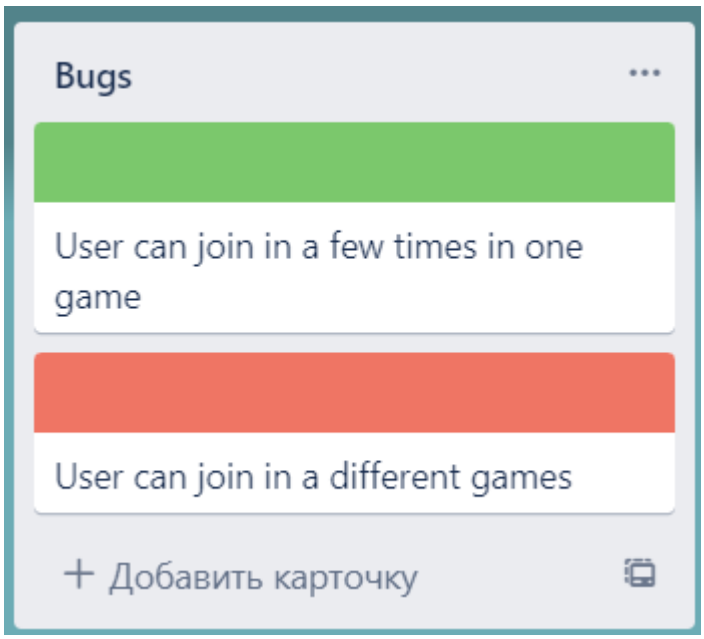


Рис. 2.12. Приклад третього стовбця

Четвертий стовбець вміщає всі версії розробки та відображають повний шлях оновлення гри від початку її створення до фінального релізу. В нього повинно увійти:

- 0.1.0, перший прототип гри із усіма робочими механіками гри, повноцінний реліз без додаткових функцій та з обмеженою графікою;
- 0.2.0, оптимізація та довершення коду до ідеального стану виходячи з правил чистого та масштабованого коду, виправлення всіх помилок;
- 0.3.0, оновлення графіки, створення нової карти та інших малюнків гри, більше графічного контенту;
- 0.4.0, рейтинг гравців, система досягнень та можливість «стримінгу» ходу гри в інших чатах;
- 0.5.0, версія, що не вийде в роботу, проте буде реалізована згодом, можливість додавання ботів різної складності при відсутності живих гравців;

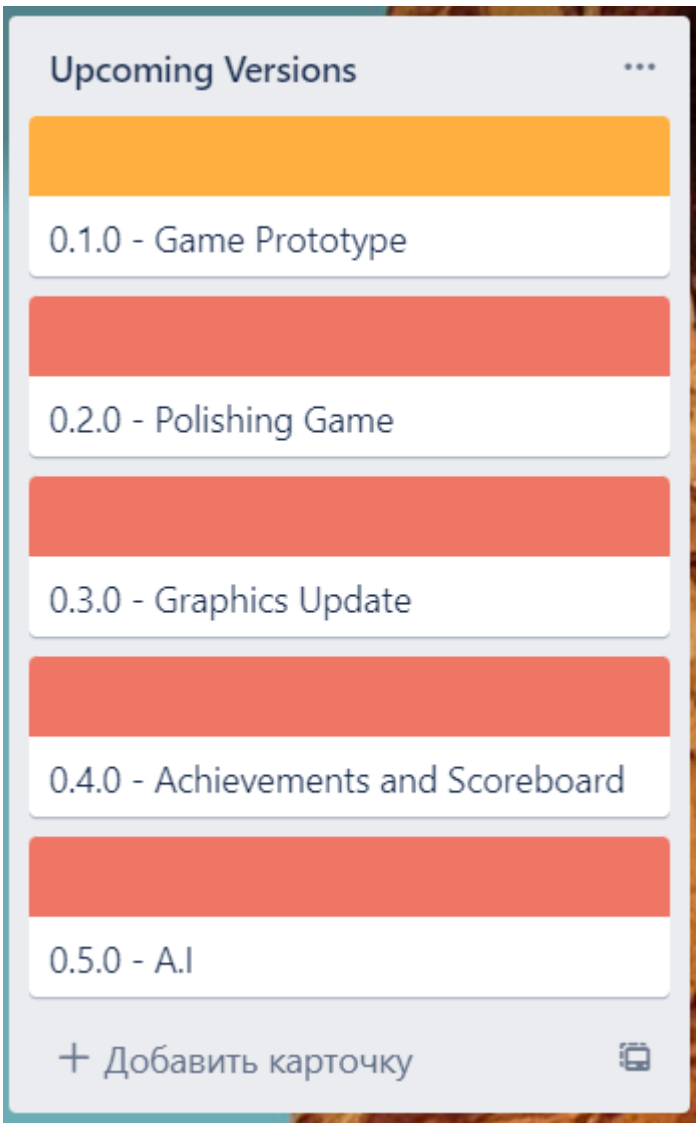


Рис. 2.13. Приклад четвертого стовбця

Останній стовбець є додатковим та містить у собі різні ідеї, можливі для реалізації у майбутніх оновленнях.

Таким чином, весь процес розробки є контрольованим і розбитим на маленькі частини, що є важливим у самоконтролі та розумінні прогресу.

3 ФУНКЦІОНАЛ ТА ДІАГРАМА КЛАСІВ РОЗРОБЛЕНОГО ПРОДУКТУ

3.1 Функції, реалізовані у додатку

3.1.1 Реалізовані можливості гри

Кінцева реалізація серверного додатку перевірена і виконує для користувача наступні функції:

- збереження даних про нових гравців і результати існуючих у базі даних
- можливість створювати нові ігрові кімнати та розпочинати гру
- можливість для адміністратора гри виключати гравців з кімнати за допомогою певної команди
- обробка ходу гравця і оновлення на основі нього мапи гри, яка мусить відсилатися у повідомленні
- пропуск ходу гравця після бездіяльності протягом певного часу, виключення його з гри після другого пропуску поспіль
- можливість перевірити свою статистику та рейтинг серед всіх гравців

Це основний набір можливостей, який містить у собі прототип гри. Гравець має змогу додати розробленого телеграм-бота до будь-якого чату та розпочати там нову партію, або ж приєднатися до існуючої. Важливою частиною кожної гри також є можливість перевірити свою статистикою та поділитися нею з друзями.

Вся основна логіка ходу гри оброблюється на серверній частині, вхідними даними вважається кожне повідомлення гравця, що починається на / та містить певну команду.

3.1.2. Доступні команди телеграм-боту

Список команд, що існують у реалізації:

/create – створити нову гру. Працює тількі у чатах, гравець не має можливості почати нову гру, якщо він вже долучений до існуючої.

/join “” – долучитись до створеної ігри. Замість лапок повинний бути спеціальний адрес гри, або Id, пред’явлений адміністратором.

/buy – купити ділянку, якщо на ній є певний бізнес. Працює тільки у грі під час ходу гравця, після того як він вже кинув кубик командою dice.

/properties – показує нумерований список всієї нерухомості гравця.

/sell “” – виставлення своєї нерухомості на аукціон. Після цього починаються торги, і всі гравці мають можливість зробити свою ставку. Замість лапок має бути номер вашої нерухомості, якщо вона є.

/auction “” – ставка на аукціоні, що проходить зараз у грі. Замість лапок повинна бути сума, більша за нинішню поставлену суму. Працює тільки під час аукціону, гравець, що заставляє свою нерухомість, робити ставку не може.

/dice – кидок кубика. Працює тільки під час ходу гравця. Є обов’язковою, служить для перевірки наявності гравця у грі та пропуску його ходу після 30 секунд простоювання.

/statistics – статистика гравця. Показує кількість партій, пройдених гравцем, кількість перемог та коефіцієнт перемог та програшів.

/players – перелічує всіх гравців у партії. Працює тільки для приєднаних користувачів та адміністратора.

/scoreboard – глобальний рейтинг. Показує перших трьох гравців за рейтингом, та місце користувача за кількістю перемог.

Деякі додаткові функції та механіки плануються у майбутній розробці проекту та найближчих оновленнях.

3.1.3. Глобальна мапа гри

Кожна гра супроводжується графічним відображенням на глобальній мапі. Мапа створена на основі оригінальної гри «Монополія» з невеликими змінами

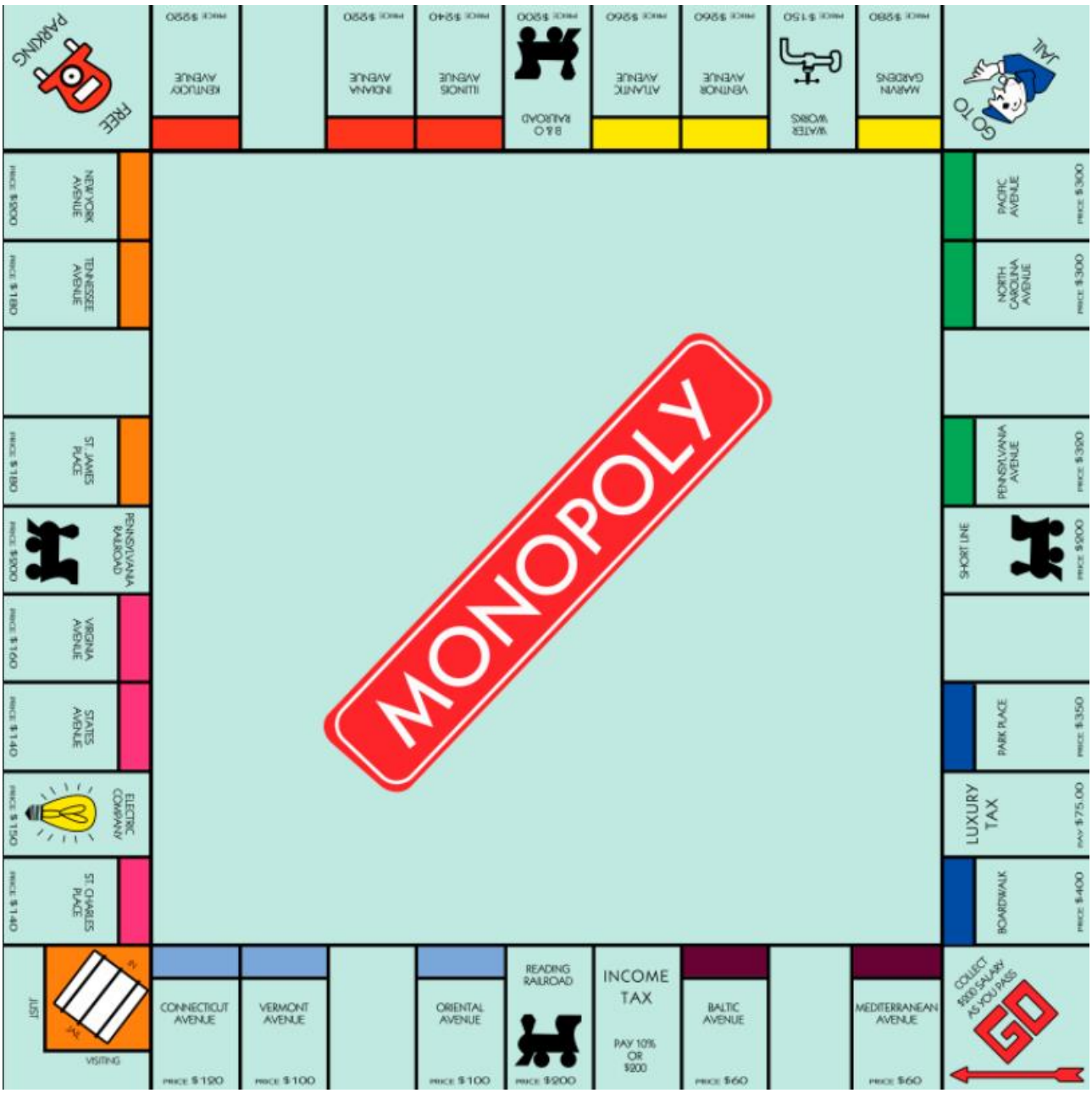


Рис. 3.1. Глобальна мапа гри

Основними змінами є відсутність карт «Шанс» та «Громадська скриня», це є тимчасово вирізаним функціоналом та буде імплементовано у майбутньому. Сама мапа також буде повністю перероблена у майбутньому графічному оновленні, і у прототипі є єдиним графічним елементом гри.

Після кожного кидку гравця сервер оновлює знаходження гравця на мапі та генерує фрагмент мапи, де зараз знаходиться гравець, та супроводжує його пояснювальним текстом з можливостями гравця на цій ділянці.

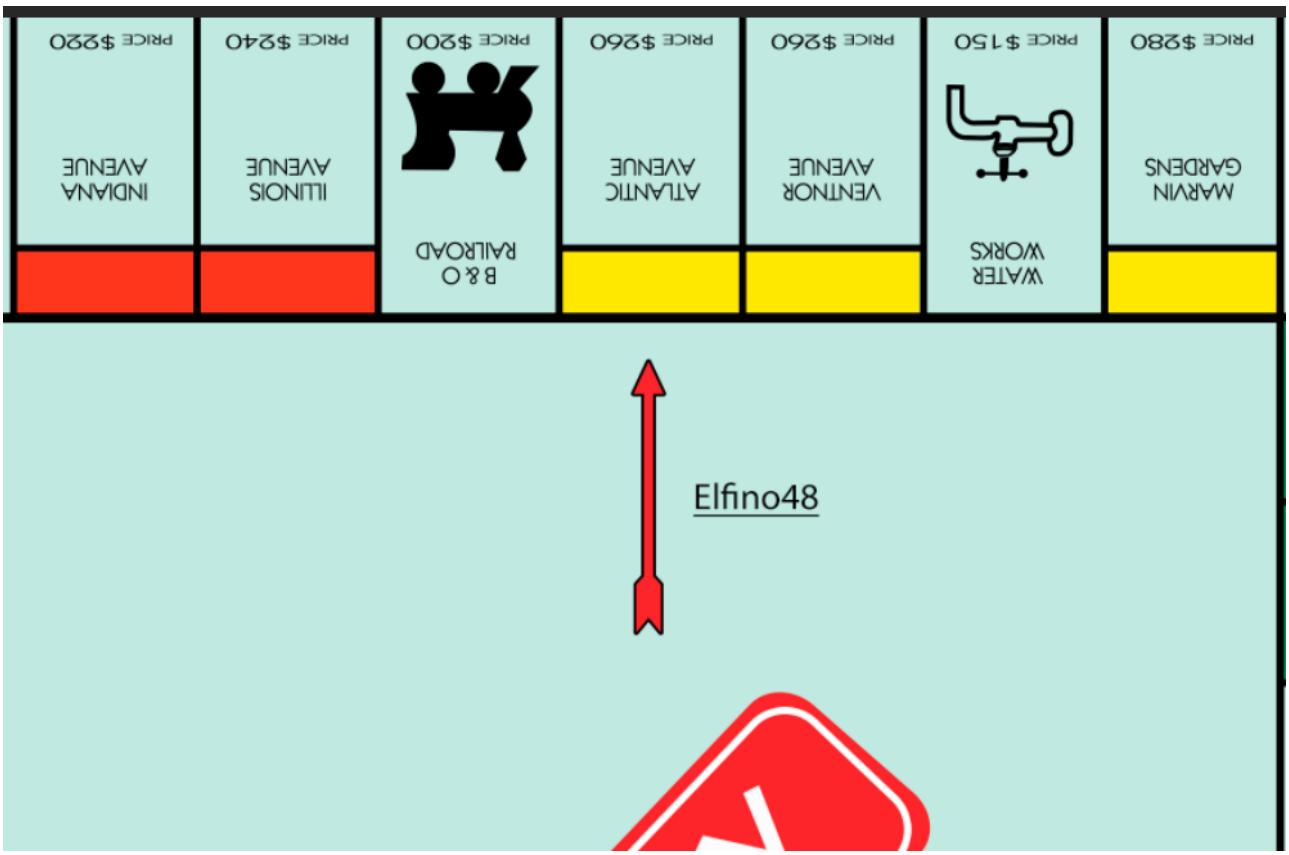


Рис. 3.2. Фрагмент мапи, що повертається із сервера після ходу гравця

На фрагменті показується місцезнаходження гравця на мапі – його нікнейм та стрілка, що вказує на нерухомість або іншу картку.

3.2 Структура та опис класів Java коду

У реалізації додатку було створено 7 класів, незалежних один від одного та відповідаючих за певну частину функціоналу.

Рис. 3.3. Структура класів додатку

Main – головний клас, з якого починається запуск боту та робота застосунку. Він відповідає за обробку всіх вхідних від користувачів команд, маніпулюванням класами, відповідними за обрахунок коду та створення нових гравців та партій.

```
5
6   public class Main {
7       public static void main(String[] args) {
8           TelegramBot bot = new TelegramBot( botToken: "5284556146:AAG0qtwuASepnX7XerrEqSMM0wcwfeO0eAU");
9
10          bot.setUpdatesListener(updates -> {
```

Рис. 3.4. Клас Main

У ньому першою чергою створюється екземпляр телеграм-боту і передається токен створеного раніше боту. Після цього, створюється updateListener метод, що реагує на всі повідомлення, що проходять через бота, та обробляє певні команди

GameLogic – клас, що відповідає за обробку ходу гравця та генерацію вихідного повідомлення.

MapGenerator – клас, що генерує глобальну карту відповідно до ходу гравця та даних певної партії.

Player – клас, що відповідає за всі дані та можливі маніпуляції з інформацією гравця. Екземпляр цього класу створюється при першому під'єднанні гравця до бота, після заповнення всієї потрібної інформації передається у клас PlayersDB і створює або оновлює рядок користувача. Також містить статистику та інформацію про партію, в якій у даний момент знаходиться гравець.

PlayersDB – клас, що відповідає за всі транзакції між кодом та базою даних гравців. Створює, оновлює та дістає рядки, пов'язані з користувачами. Всі транзакції проходять у базі даних MongoDB.

Game – клас, що відповідає за всі дані та можливі маніпуляції з інформацією про певну партію. Екземпляр цього класу створюється одразу при створенні нової партії гравцем, після заповнення всієї потрібної інформації передається у клас GamesDB і створює або оновлює рядок партії. Містить інформацію про під'єднаних гравців та все про даний етап гри. Оновлюється після кожного ходу гравця.

GamesDB – клас, що відповідає за всі транзакції між кодом та базою даних партій. Створює, оновлює та дістає рядки, пов'язані з партіями гри. Всі транзакції проходять у базі даних MongoDB.

3.3 Тестування механік гри

Для початку гри потрібно знайти у телеграмі бота за тегом користувача @MonopolyGameOnlinebot, та ввести команду /start.

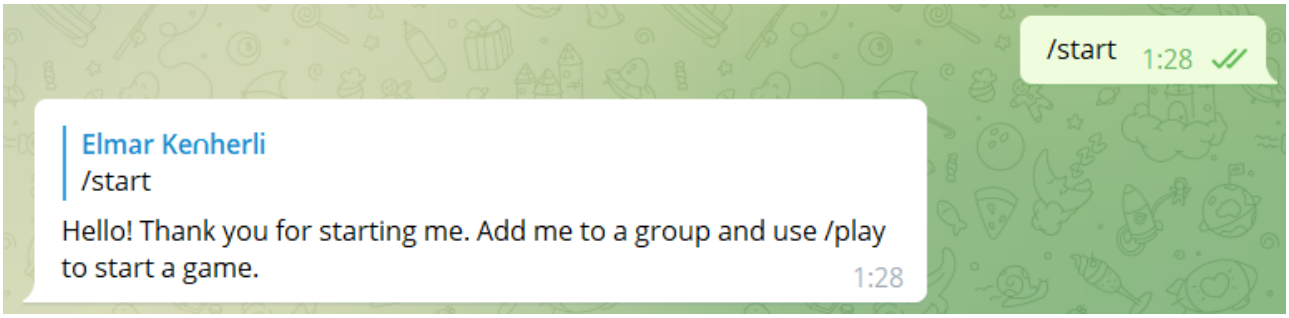


Рис. 3.5. Початок роботи з телеграм-ботом

Це створить відповідний рядок у базі даних про користувача. Далі для створення нової гри потрібно додати телеграм-бота у чат з іншими користувачами, та ввести команду /create.

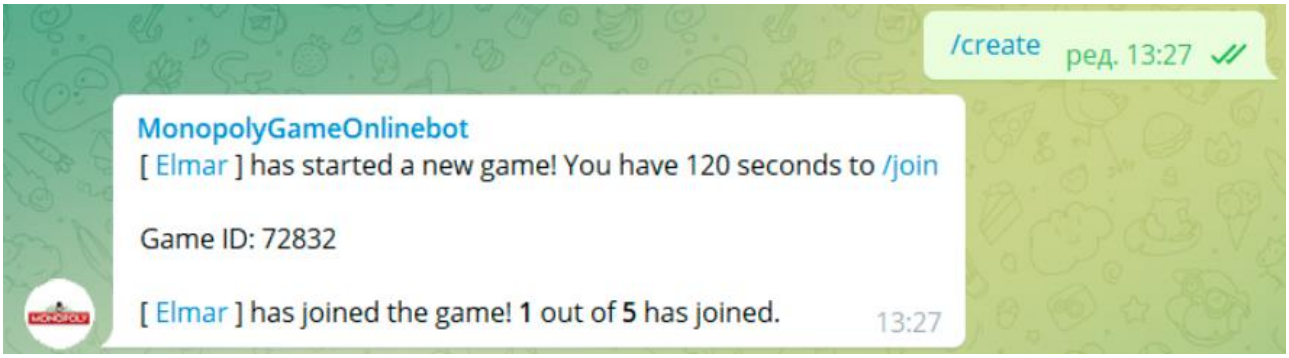


Рис. 3.6. Початок нової партії

Кожна партія може містити у собі до 5 користувачів. Для того, щоб під’єднатися до існуючої партії, потрібно ввести команду /join та певний Id гри, до якої ви хочете під’єднатися. Важливо знаходитися в тому чаті, в якому партія була ініційована. Також гравець не має можливості під’єднатися до гри, якщо він вже під’єднаний або є адміністратором іншої партії. Якщо гравець був виключений з гри, то він не буде мати можливості під’єднатися до неї знову. Також час, за який можна під’єднатись до гри, обмежений 120 секундами. Це обумовлено тим, що деякі партії так і не знаходять гравців, і повинні бути видаленими з бази даних.

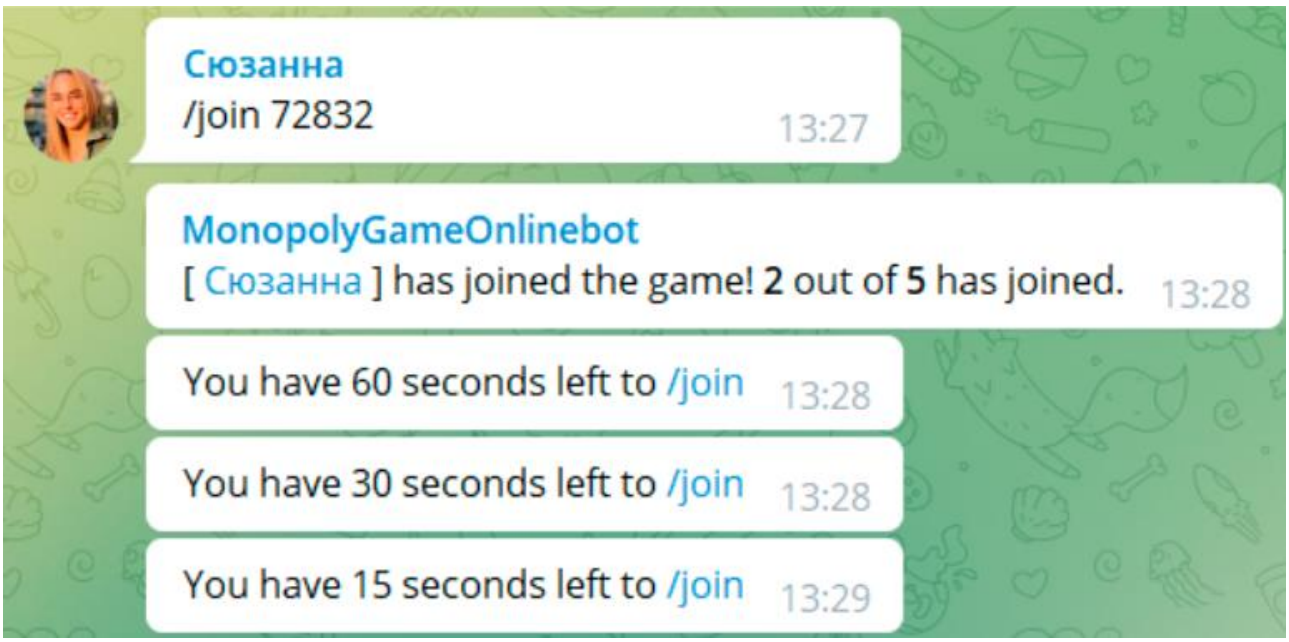


Рис. 3.7. Приєднання до гри та відлік до старту

Після кожного під’єднання гравців рахунок часу повертається до 120 секунд, після закінчення яких починається гра. Під час свого ходу гравець повинен викликати команду /dice для того, щоб кинути кубик.

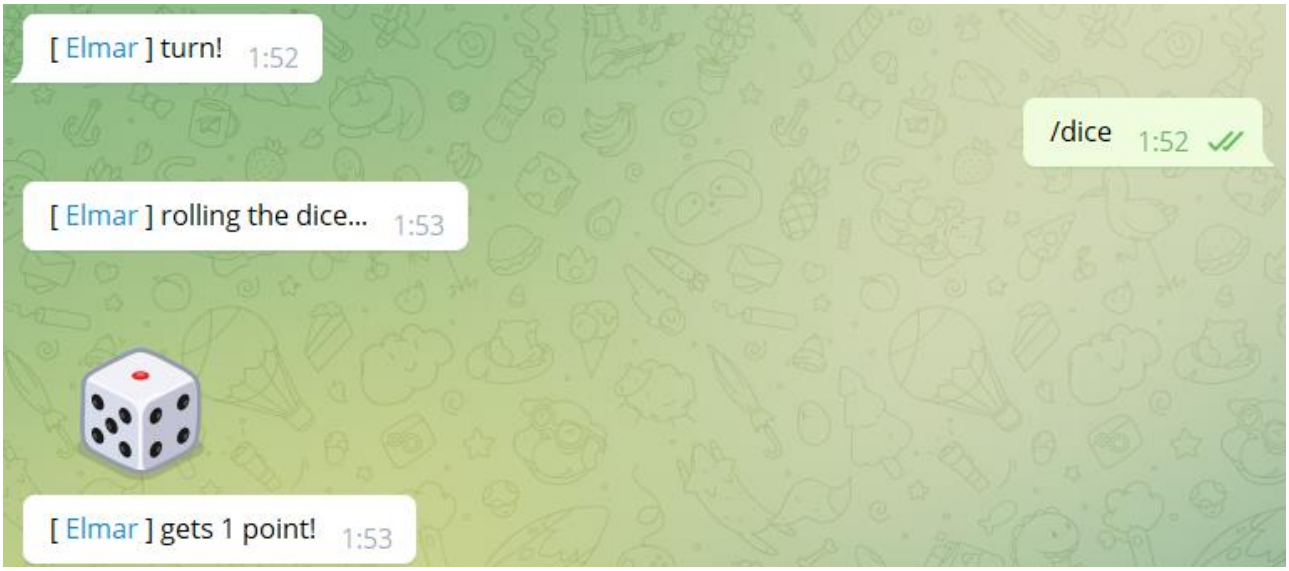


Рис. 3.8. Кидок кубика

Після оголошення кількості шагів, які випали на кубику, сервер опрацьовує положення гравця та додає певну кількість шагів, після цього оновлюються бази даних та генерується фрагмент мапи, який повертається до користувачів у вигляді рисунку та інформації про хід. Оголошується поле, на яке випав гравець, його вартість або інформація про те, що гравець знаходиться у в'язниці протягом трьох ходів. Якщо випала нерухомість, гравець має можливість її придбати командою /buy.

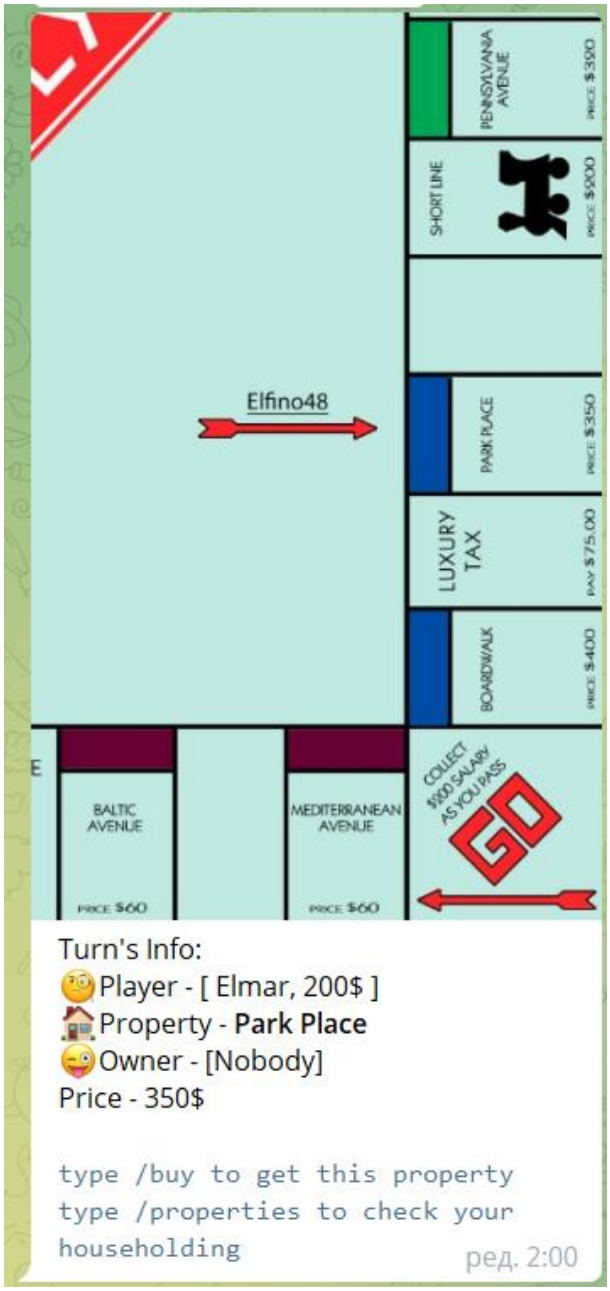


Рис. 3.9. Інформація про хід гравця, надана ботом

Гра закінчується тоді, коли всі гравці окрім одного втратять всі грошові збереження та нерухомість. Гравець має можливість перевірити свою ігрову статистику через команду /statistics.

Рис. 3.10. Перевірка статистики гравця

Таким чином, маємо повноцінну гру, реалізовану через телеграм-бота. Деякі можливості Монополії були тимчасово вирізані з гри, проте будуть добавлені у майбутньому.

ВИСНОВКИ

Робота присвячена розробці ігрового застосунку для проведення партій у «Монополію» в умовах Telegram месенджеру.

Проведено аналіз предметної галузі у розробці настільної гри “Монополія”. Ключові проблеми, які можна виділити, пов’язані з обмеженнями месенджеру як застосунку для відображення ігор, оскільки його можливості звужені до відображення тексту та картинок.

Розглянуто програмне забезпечення,. яке може бути використано для створення ігрових чат-ботів. Ключовими недоліками існуючих систем є відсутність цільової аудиторії, можливостей для кастомізації, невеликий функціонал та проблема конфідеційності.

Розроблено вимоги до ігрового додатку на основі аналізу переваг та недоліків існуючих додатків. Основними недоліками існуючих застосунків є відсутність певних графічних елементів та системи досягнень і статистики кожного гравця. Це значно погіршує ігровий досвід, і повинно бути реалізовано у даному ігровому додатку.

Розроблено ігровий застосунок та спеціальний телеграм-бот на основі аналізу потреб користувачів.

Проведено тестування гри серед звичайних користувачів Telegram месенджеру, опитування стосовно оригінальності застосунку та зацікавленностію

Робота пройшла апробацію на Науково-технічній конференції «Застосування програмного забезпечення в ІКТ», м. Київ, ДУТ, 20 квітня 2022 року.

ПЕРЕЛІК ПОСИЛАНЬ

1. Telegram Bot Platform [Електронний ресурс] – Режим доступу до ресурсу: https://telegram.org/blog/bot-revolution

2. Telegram APIs [Електронний ресурс] – Режим доступу до ресурсу: https://core.telegram.org/

3. WhatsApp, Viber and Telegram: which is the Best for Instant Messaging? [Електронний ресурс] – Режим доступу до ресурсу: https://api.semanticscholar.org/CorpusID:61685613

4. Python and Java: The Best of Both Worlds [Електронний ресурс] – Режим доступу до ресурсу: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.117.6454&rep=rep1&type=pdf

5. Comparative study of the Pros and Cons of Programming languages Java, Scala, C++, Haskell, VB .NET, AspectJ, Perl, Ruby, JS, PHP & Scheme - a Team 11 COMP6411-S10 Term Report [Електронний ресурс] – Режим доступу до ресурсу: https://api.semanticscholar.org/CorpusID:18989804

6. WhatsApp Cloud API [Електронний ресурс] – Режим доступу до ресурсу: https://developers.facebook.com/docs/whatsapp/cloud-api

7. Facebook API documentation [Електронний ресурс] – Режим доступу до ресурсу: https://developers.facebook.com/docs

8. A comparative study: MongoDB vs. MySQL [Електронний ресурс] – Режим доступу до ресурсу: https://ieeexplore.ieee.org/abstract/document/7158433

9. Applying Kanban principles to electronic resource acquisitions with Trello [Електронний ресурс] – Режим доступу до ресурсу: https://www.tandfonline.com/doi/abs/10.1080/1941126X.2016.1130464?journalCode=wacq20

10. The Rules of the Game: Experiencing Global Capitalism on a Monopoly Board [Електронний ресурс] – Режим доступу до ресурсу: https://www.tandfonline.com/doi/abs/10.1080/15512169.2015.1082475

11. Сайт Heroku [Електронний ресурс]. – Режим доступу: https://www.heroku.com/

12. Heroku – Вікіпедія [Електронний ресурс]. – Режим доступу: https://ru.wikipedia.org/wiki/Heroku

13. Огляд протоколу HTTP [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/ru/docs/Web/HTTP/Overview

14. Synbot [Електронний ресурс]. – Режим доступу: https://ru.telegramstore.com/catalog/bots/synbot_/

15. REST API [Електронний ресурс]. – Режим доступу: https://habr.com/ru/post/351890/

16. JSON [Електронний ресурс]. – Режим доступу: https://www.json.org/json-ru.html

17. Тестування телеграм-бота [Електронний ресурс]. – Режим доступу: https://habr.com/ru/post/322816/

18. What is Artificial Intelligence (AI) [Електронний ресурс]. – Режим доступу: https://www.techopedia.com/definition/190/artificial-intelligence-ai

19. Muldowney O. Chatbots. An Introduction And Easy Guide To Making Your Own / O. Muldowney. – Dublin: Curses & Magic, 2017. – 69 p.

20. Sheth B. The Bot Lifecycle. What to know before you make your chatbot [Електронний ресурс]. – Режим доступу: https://chatbotsmagazine.com/the-bot-lifecycle1ff357430db7

Додаток А

```
import java.util.*;
public class Game {
    private String id;
    private String ownerId;
    private ArrayList<String> playersIds;
    private String currentTurnPlayerId;

    public Game(String ownerId) {
        this.generateGameId();
        this.ownerId = ownerId;
        this.playersIds = new ArrayList<>();
    }

    private void generateGameId() {
        Random rand = new Random();
        int upperbound = 100000;
        this.id = String.valueOf(rand.nextInt(upperbound));
    }

    public void addPlayer(String id) {
        this.playersIds.add(id);
    }

    public String getPlayersList() {
        return "";
    }

    public String getGameId() {
```

```
        return String.valueOf(id);
    }
}

public class GameLogic {
        public GameLogic(Message msg, int wait, GroupInfo groupInfo) throws
ConcurrentGameException {

            long gid = msg.chat().id();
            if (games.containsKey(gid)) {
                throw new ConcurrentGameException(this);
            }
            this.gid = gid;
            this.turnWait = groupInfo.waitTime;
            this.groupInfo = groupInfo;
            this.lang = DealBot.lang(gid);
            this.translation = Translation.get(this.lang);
            if (maintMode) {
                // disallow starting games
                this.execute(new SendMessage(gid,
this.translation.MAINT_MODE_NOTICE()).parseMode(ParseMode.HTML), new
EmptyCallback<>());
                return;
            }

            try (Connection conn = Main.getConnection()) {
                PreparedStatement stmt = conn.prepareStatement("INSERT INTO games
(gid) values (?)", Statement.RETURN_GENERATED_KEYS);
                stmt.setLong(1, gid);
                stmt.executeUpdate();
```

```
            try (ResultSet generatedKeys = stmt.getGeneratedKeys()) {
                if (generatedKeys.next()) {
                    id = generatedKeys.getInt(1);
                }
                else {
                    throw new SQLException("Creating game failed, no ID obtained.");
                }
            }
            stmt.close();
        } catch (SQLException e) {
            e.printStackTrace();
            this.execute(new SendMessage(msg.chat().id(), this.translation.ERROR() +
e.getMessage()).replyToMessageId(msg.messageId()));
        }
        // notify queued users
        try (Connection conn = Main.getConnection()) {
            PreparedStatement stmt = conn.prepareStatement("SELECT tgid FROM
next_game where gid=?");
            stmt.setLong(1, gid);
            ResultSet rs = stmt.executeQuery();
            String startMsg = this.translation.GAME_STARTING_IN(msg.chat().title());
            while (rs.next()) {
                SendMessage send = new SendMessage(rs.getInt(1), startMsg);
                this.execute(send);
            }
            stmt.close();
        } catch (SQLException e) {
            e.printStackTrace();
            this.execute(new SendMessage(msg.chat().id(), this.translation.ERROR() +
e.getMessage()).replyToMessageId(msg.messageId()));
```

```
            }
            this.execute(new SendMessage(gid,
String.format(this.translation.GAME_START_ANNOUNCEMENT(), msg.from().id(),
msg.from().firstName(), wait, id)).parseMode(ParseMode.HTML), new
EmptyCallback<>());
            games.put(gid, this);
            addPlayer(msg);
            startTime = System.currentTimeMillis() + wait * 1000L;
            int i;
            i = 0;
            while (wait > remindSeconds[i]) {
                ++i;
            }

            schedule(this::remind, (wait - remindSeconds[--i]) * 1000);
            this.logf("Game created in %s [%d]", msg.chat().title(), msg.chat().id());
        }

        public static Game byUser(long tgid) {
            return uidGames.get(tgid);
        }

        public static RunInfo runInfo() {
            int total = games.size();
            int players = uidGames.size();
            int running = 0;
            for (Game game : games.values()) {
                if (game.started) {
                    ++running;
                }
```

```
        }
        return new RunInfo(running, total, players, games);
    }

    public void addPlayer(Message msg) {
        User from = msg.from();
        Game g = uidGames.get(from.id());
        if (g != null && g.ended) {
            uidGames.remove(from.id());
        }
        if (!started && !players.contains(from) && !uidGames.containsKey(from.id())
&& playerCount() < 5) {
            SendMessage send = new SendMessage(msg.from().id(),
String.format(Translation.get(DealBot.lang(msg.from().id())).JOIN_SUCCESS(),
msg.chat().title().replace("*", "\\*"), this.id)).parseMode(ParseMode.Markdown);
            if (msg.chat().username() != null) {
                send.replyMarkup(new InlineKeyboardMarkup(new
InlineKeyboardButton(this.translation.BACK_TO() +
msg.chat().title()).url("https://t.me/" + msg.chat().username())));
            }
            this.execute(send, new Callback<SendMessage, SendResponse>() {
                @Override
                public void onResponse(SendMessage request, SendResponse response) {
                    if (response.isOk()) {
                        if (playerCount() >= 5 || players.contains(msg.from())) {
                            return;
                        }
                        players.add(msg.from());
                        uidGames.put(msg.from().id(), Game.this);
                        int count = playerCount();
```

```
                    Game.this.execute(new SendMessage(msg.chat().id(),
String.format(Game.this.translation.JOINED_ANNOUNCEMENT(), msg.from().id(),
msg.from().firstName(), count)).parseMode(ParseMode.HTML), new
EmptyCallback<>());
                    if (count == 5) {
                        start();
                    }
                } else {
                    Game.this.execute(new SendMessage(msg.chat().id(),
Game.this.translation.START_ME_FIRST())
                            .replyToMessageId(msg.messageId())
                            .replyMarkup(new InlineKeyboardMarkup(new
InlineKeyboardButton(Game.this.translation.START_ME()).url("https://t.me/" +
Main.getConfig("bot.username")))));
                }
            }

            @Override
            public void onFailure(SendMessage request, IOException e) {
                e.printStackTrace();
            }
        });
    }
}

public boolean removePlayer(long tgid) {
    if (!started && players.removeIf(user -> user.id() == tgid)) {
        return uidGames.remove(tgid) != null;
    }
    return false;
```

```
}

public List<User> getPlayers() {
    return players;
}

public List<GamePlayer> getGamePlayers() {
    return gamePlayers;
}

public int playerCount() {
    return players.size();
}

public boolean started() {
    return started;
}

public int nextNonce() {
    return ++nextNonce;
}

public void extend() {
    if (started) {
        return;
    }
    cancelFuture();
    startTime += 30000;
    startTime = Math.min(startTime, System.currentTimeMillis() + 180000);
    long seconds = (startTime - System.currentTimeMillis() + 999) / 1000;
```

```
            this.execute(new SendMessage(gid,
String.format(this.translation.EXTENDED_ANNOUNCEMENT(), seconds)));
            int i;
            i = 0;
            while (i < 6 && remindSeconds[i] < seconds) {
                ++i;
            }
//          this.log("scheduled remind task");
            schedule(this::remind, (seconds - remindSeconds[--i]) * 1000);
        }

        /**
         * Starts the game
         */
        private void start() {
            SendMessage send = new SendMessage(gid,
translation.GAME_STARTING_ANNOUNCEMENT());
            started = true;
            this.execute(send);
            mainDeck.add(new PropertyCard(1, translation.PROPERTY_NAME(0), 0,
translation));
            mainDeck.add(new PropertyCard(1, translation.PROPERTY_NAME(1), 0,
translation));

            mainDeck.add(new PropertyCard(1, translation.PROPERTY_NAME(2), 1,
translation));
            mainDeck.add(new PropertyCard(1, translation.PROPERTY_NAME(3), 1,
translation));
            mainDeck.add(new PropertyCard(1, translation.PROPERTY_NAME(4), 1,
translation));
```

```
        mainDeck.add(new PropertyCard(2, translation.PROPERTY_NAME(5), 2,
translation));
        mainDeck.add(new PropertyCard(2, translation.PROPERTY_NAME(6), 2,
translation));
        mainDeck.add(new PropertyCard(2, translation.PROPERTY_NAME(7), 2,
translation));

        mainDeck.add(new PropertyCard(2, translation.PROPERTY_NAME(8), 3,
translation));
        mainDeck.add(new PropertyCard(2, translation.PROPERTY_NAME(9), 3,
translation));
        mainDeck.add(new PropertyCard(2, translation.PROPERTY_NAME(10), 3,
translation));

        mainDeck.add(new PropertyCard(3, translation.PROPERTY_NAME(11), 4,
translation));
        mainDeck.add(new PropertyCard(3, translation.PROPERTY_NAME(12), 4,
translation));
        mainDeck.add(new PropertyCard(3, translation.PROPERTY_NAME(13), 4,
translation));

        mainDeck.add(new PropertyCard(3, translation.PROPERTY_NAME(14), 5,
translation));
        mainDeck.add(new PropertyCard(3, translation.PROPERTY_NAME(15), 5,
translation));
        mainDeck.add(new PropertyCard(3, translation.PROPERTY_NAME(16), 5,
translation));
```

```
        mainDeck.add(new PropertyCard(4, translation.PROPERTY_NAME(17), 6,
translation));
        mainDeck.add(new PropertyCard(4, translation.PROPERTY_NAME(18), 6,
translation));
        mainDeck.add(new PropertyCard(4, translation.PROPERTY_NAME(19), 6,
translation));

        mainDeck.add(new PropertyCard(4, translation.PROPERTY_NAME(20), 7,
translation));
        mainDeck.add(new PropertyCard(4, translation.PROPERTY_NAME(21), 7,
translation));

        mainDeck.add(new PropertyCard(2, translation.PROPERTY_NAME(22), 8,
translation));
        mainDeck.add(new PropertyCard(2, translation.PROPERTY_NAME(23), 8,
translation));
        mainDeck.add(new PropertyCard(2, translation.PROPERTY_NAME(24), 8,
translation));
        mainDeck.add(new PropertyCard(2, translation.PROPERTY_NAME(25), 8,
translation));

        mainDeck.add(new PropertyCard(2, translation.PROPERTY_NAME(26), 9,
translation));
        mainDeck.add(new PropertyCard(2, translation.PROPERTY_NAME(27), 9,
translation));

        // wildcard properties
        mainDeck.add(new WildcardPropertyCard(3, translation.WILD_CARD(4, 5),
new int[]{4, 5}, translation));
```

```
        mainDeck.add(new WildcardPropertyCard(3, translation.WILD_CARD(4, 5),
new int[]{4, 5}, translation));
        mainDeck.add(new WildcardPropertyCard(4, translation.WILD_CARD(6, 7),
new int[]{6, 7}, translation));
        mainDeck.add(new WildcardPropertyCard(1, translation.WILD_CARD(0, 1),
new int[]{0, 1}, translation));
        mainDeck.add(new WildcardPropertyCard(2, translation.WILD_CARD(2, 3),
new int[]{2, 3}, translation));
        mainDeck.add(new WildcardPropertyCard(2, translation.WILD_CARD(2, 3),
new int[]{2, 3}, translation));
        mainDeck.add(new WildcardPropertyCard(4, translation.WILD_CARD(6, 8),
new int[]{6, 8}, translation));
        mainDeck.add(new WildcardPropertyCard(4, translation.WILD_CARD(1, 8),
new int[]{1, 8}, translation));
        mainDeck.add(new WildcardPropertyCard(2, translation.WILD_CARD(8, 9),
new int[]{8, 9}, translation));
        mainDeck.add(new WildcardPropertyCard(0, translation.WILD_CARD(), new
int[]{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}, translation));
        mainDeck.add(new WildcardPropertyCard(0, translation.WILD_CARD(), new
int[]{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}, translation));

        // action mainDeck
        for (int i = 0; i < 3; i++) {
            mainDeck.add(new ItsMyBirthdayActionCard(translation));
        }
        for (int i = 0; i < 3; i++) {
            mainDeck.add(new DebtCollectorCard(translation));
        }
        for (int i = 0; i < 10; i++) {
            mainDeck.add(new GoPassActionCard(translation));
```

```
}
for (int i = 0; i < 2; i++) {
    mainDeck.add(new DoubleRentActionCard(translation));
}
for (int i = 0; i < 3; i++) {
    mainDeck.add(new JustSayNoCard(translation));
}
for (int i = 0; i < 2; i++) {
    mainDeck.add(new DealBreakerActionCard(translation));
}
for (int i = 0; i < 3; i++) {
    mainDeck.add(new SlyDealActionCard(translation));
}
for (int i = 0; i < 3; i++) {
    mainDeck.add(new HouseActionCard(translation));
}
for (int i = 0; i < 2; i++) {
    mainDeck.add(new HotelActionCard(translation));
}
for (int i = 0; i < 3; i++) {
    mainDeck.add(new ForcedDealActionCard(translation));
}

// rent mainDeck
mainDeck.add(new RentActionCard(new int[]{0, 1}, translation));
mainDeck.add(new RentActionCard(new int[]{0, 1}, translation));
mainDeck.add(new RentActionCard(new int[]{2, 3}, translation));
mainDeck.add(new RentActionCard(new int[]{2, 3}, translation));
mainDeck.add(new RentActionCard(new int[]{4, 5}, translation));
mainDeck.add(new RentActionCard(new int[]{4, 5}, translation));
```

```
            mainDeck.add(new RentActionCard(new int[]{6, 7}, translation));
            mainDeck.add(new RentActionCard(new int[]{6, 7}, translation));
            mainDeck.add(new RentActionCard(new int[]{8, 9}, translation));
            mainDeck.add(new RentActionCard(new int[]{8, 9}, translation));
            for (int i = 0; i < 3; i++) {
                mainDeck.add(new RentActionCard(new int[]{0, 1, 2, 3, 4, 5, 6, 7, 8, 9},
translation));
            }
}

public class GamesDB {
    private static HashMap<String, Game> allGames = new HashMap<>();

    public static void addGame(String gameId, Game newGame) {
        allGames.put(gameId, newGame);
    }

    public static Game getGame(String gameId) {
        return allGames.get(gameId);
    }

    public static void addNewGameToDB(Game game) {
        try (Connection conn = Main.getConnection()) {
            PreparedStatement stmt = conn.prepareStatement("INSERT INTO games (gid)
values (?)", Statement.RETURN_GENERATED_KEYS);
            stmt.setLong(1, gid);
            stmt.executeUpdate();
            try (ResultSet generatedKeys = stmt.getGeneratedKeys()) {
                if (generatedKeys.next()) {
                    id = generatedKeys.getInt(1);
```

```
            }
            else {
                throw new SQLException("Creating game failed, no ID obtained.");
            }
        }
        stmt.close();
    } catch (SQLException e) {
        e.printStackTrace();
        this.execute(new SendMessage(msg.chat().id(), this.translation.ERROR() +
e.getMessage()).replyToMessageId(msg.messageId()));
    }
    // notify queued users
    try (Connection conn = Main.getConnection()) {
        PreparedStatement stmt = conn.prepareStatement("SELECT tgid FROM
next_game where gid=?");
        stmt.setLong(1, gid);
        ResultSet rs = stmt.executeQuery();
        String startMsg = this.translation.GAME_STARTING_IN(msg.chat().title());
        while (rs.next()) {
            SendMessage send = new SendMessage(rs.getInt(1), startMsg);
            this.execute(send);
        }
        stmt.close();
    } catch (SQLException e) {
        e.printStackTrace();
        this.execute(new SendMessage(msg.chat().id(), this.translation.ERROR() +
e.getMessage()).replyToMessageId(msg.messageId()));
    }
    this.execute(new SendMessage(gid,
String.format(this.translation.GAME_START_ANNOUNCEMENT(), msg.from().id(),
```

```
msg.from().firstName(), wait, id)).parseMode(ParseMode.HTML), new
EmptyCallback<>());
        games.put(gid, this);
        addPlayer(msg);
        startTime = System.currentTimeMillis() + wait * 1000L;
        int i;
        i = 0;
        while (wait > remindSeconds[i]) {
            ++i;
        }

        schedule(this::remind, (wait - remindSeconds[--i]) * 1000);
        this.logf("Game created in %s [%d]", msg.chat().title(), msg.chat().id());
    }

    public static Game getGameFromDB(String id) {
        try (Connection conn = Main.getConnection()) {
            PreparedStatement stmt = conn.prepareStatement("SELECT games (gid) values
(?)", Statement.RETURN_GENERATED_KEYS);
            stmt.setLong(1, gid);
            stmt.executeUpdate();
            try (ResultSet generatedKeys = stmt.getGeneratedKeys()) {
                if (generatedKeys.next()) {
                    id = generatedKeys.getInt(1);
                }
                else {
                    throw new SQLException("Getting Game From DB");
                }
            }
            stmt.close();
```

```
        } catch (SQLException e) {
            e.printStackTrace();
            this.execute(new SendMessage(msg.chat().id(), this.translation.ERROR() +
e.getMessage()).replyToMessageId(msg.messageId()));
        }
    }
}

public class Main {
    public static void main(String[] args) {
        TelegramBot bot = new
TelegramBot("5284556146:AAG0qtwuASepnX7XerrEqSMM0wcwfeO0eAU");

        bot.setUpdatesListener(updates -> {
            if (updates.get(0).message().chat() != null && updates.get(0).message().text() !=
null) {
                long chatId = updates.get(0).message().chat().id();
                long userId = updates.get(0).message().from().id();
                String username = updates.get(0).message().from().username();

                String userName = updates.get(0).message().from().firstName()
                        + updates.get(0).message().from().lastName();
                String message = updates.get(0).message().text();

                if (message.equals("/create@MonopolyGameOnlinebot")) { // CREATE
                    Player newPlayer;

                    if (PlayersDB.allPlayers.containsKey(userId)) {
                        newPlayer = PlayersDB.allPlayers.get(userId);
                        System.out.println("Not New");
```

```
            } else {
                newPlayer = new Player(userId, userName, username);
                System.out.println("New");
            }
            System.out.println(userId);
            System.out.println(newPlayer.checkIfGameOwner());
            if (newPlayer.checkIfGameOwner()) {
                SendMessage message1 = new SendMessage(chatId, "Вы уже создали
одну игру");
                bot.execute(message1);
            } else {
                Game newGame = new Game();
                newGame.addPlayer(newPlayer);
                newPlayer.setCreatedGameId(newGame.getGameId());
                newPlayer.connectToGame(newGame.getGameId());
                currentGames.put(newGame.getGameId(), newGame);

                PlayersDB.addPlayer(userId, newPlayer);
                System.out.println("current game added - " + newGame.getGameId());
                SendMessage message1 = new SendMessage(chatId, "Создана новая
игра\nЧтобы присоединиться, введите /join@MonopolyGameOnlinebot_"
                        + newGame.getGameId() + "\n"
                        + "Чтобы посмотреть всех игроков, введите
/players@MonopolyGameOnlinebot_"
                        + newGame.getGameId() + "\n"
                        + newGame.getPlayersList());

                bot.execute(message1);
            }
        } else if (message.contains("/join@MonopolyGameOnlinebot")) { // JOIN
```

```
                boolean isNew = false;
                Player newPlayer;
                if (PlayersDB.allPlayers.containsKey(userId)) {
                    newPlayer = PlayersDB.allPlayers.get(userId);
                } else {
                    newPlayer = new Player(userId, userName, username);
                    isNew = true;
                }

                String gameId = message.replace("/join@MonopolyGameOnlinebot_", "");
                Game gameToJoin = currentGames.get(gameId);
                System.out.println("current game added - " + gameId);
                System.out.println(newPlayer.getJoinedGameId() + "  " +
gameToJoin.getGameId());
                if (newPlayer.checkIfJoinedToGame() &&
!newPlayer.getJoinedGameId().equals(gameToJoin.getGameId())) {
                    SendMessage message1 = new SendMessage(chatId, "Вы уже
присоединены к другой игре");
                    bot.execute(message1);
                } else {
                    if (gameToJoin != null && gameToJoin.isANewPlayer(newPlayer)) {
                        gameToJoin.addPlayer(newPlayer);
                        SendMessage message1 = new SendMessage(chatId,
gameToJoin.getPlayersList());

                        bot.execute(message1);
                    } else if (gameToJoin != null &&
!gameToJoin.isANewPlayer(newPlayer)) {
                        SendMessage message1 = new SendMessage(chatId, "Ну ты ж уже
подключился, ну ебобо");
```

```
                    bot.execute(message1);
                } else {
                    SendMessage message1 = new SendMessage(chatId, "Такой игры
нет, долбаеб!");
                    bot.execute(message1);
                }
            }
        } else if (message.contains("/players@MonopolyGameOnlinebot")) { //
PLAYERS
            String gameId = message.replace("/players@MonopolyGameOnlinebot_",
"");
            Game gameToJoin = currentGames.get(gameId);
            if (gameToJoin != null) {
                SendMessage message1 = new SendMessage(chatId,
gameToJoin.getPlayersList());
                bot.execute(message1);
            } else {
                SendMessage message1 = new SendMessage(chatId, "Такой игры нет,
долбаеб!");
                bot.execute(message1);
            }
        }

        SendMessage req = new SendMessage(chatId, "Для создания новой игры
введите /start")
                .disableNotification(true).entities(new
MessageEntity(MessageEntity.Type.bot_command, 0, 0));
        SendDice dice = new SendDice(chatId).emoji("\uD83C\uDFB2");

        bot.execute(req);
```

```
            SendResponse responses = bot.execute(dice);
            GetChatMember chatMember = new GetChatMember(chatId,
updates.get(0).message().from().id());
            GetChatMemberResponse resp = bot.execute(chatMember);

            SendMessage message = new SendMessage(chatId,
updates.get(0).message().from().firstName() + ", ваш результат - " +
String.valueOf(responses.message().dice().value()))
                    .entities(new MessageEntity(MessageEntity.Type.text_mention, 0,
updates.get(0).message().from().firstName().length()).user(resp.chatMember().user()));
            bot.execute(message);

            GetUserProfilePhotos request = new GetUserProfilePhotos(chatId);
            GetUserProfilePhotosResponse getFileResponse = bot.execute(request);
            System.out.println(getFileResponse.photos());
            String id = getFileResponse.photos().photos()[0][0].fileId();

            GetFile requestAboba = new GetFile(id);
            GetFileResponse getFileResponseAboba = bot.execute(requestAboba);
        }
        return UpdatesListener.CONFIRMED_UPDATES_ALL;
    });

// Send messages

    }
}

public class MapGenerator {
    public static File generateMap() {
```

```
        File file = getFileResponseAboba.file();

        String fullPath = bot.getFullFilePath(file);
        System.out.println(fullPath);

        BufferedImage joined = new BufferedImage(1200, 1200,
BufferedImage.TYPE_INT_RGB);
        BufferedImage image1 = null;
        try {
            image1 = ImageIO.read(new java.io.File("E:\\aboba.png"));
        } catch (IOException e) {
            e.printStackTrace();
        }
        BufferedImage image2 = null;
        try {
            image2 = ImageIO.read(new URL(fullPath));
        } catch (IOException e) {
            e.printStackTrace();
        }

        Graphics2D graph = joined.createGraphics();
        graph.drawImage(image1, 0, 0, null);
        graph.drawImage(image2, 0, 0, null);

        java.io.File joinedFile = new java.io.File("E:\\joined.png");
        try {
            ImageIO.write(joined, "png", joinedFile);
        } catch (IOException e) {
            e.printStackTrace();
        }
```

```
    }
}

public class Player {
    private String id;
    private String username;
    private String fullName;
    private String currentGameId;

    public Player(Long id, String username, String fullName) {
        this.id = String.valueOf(id);
        this.username = username;
        this.fullName = fullName;
    }

    public Boolean isConnectedToGame() {
        return (currentGameId != null);
    }

    public void disconnectFromGame() {
        this.connectToGame(null);
    }

    public void connectToGame(String gameId) {
        this.currentGameId = gameId;
    }

    public String getFullName() {
        return this.fullName;
    }
```

```
    public String getUsername() {
        return this.username;
    }

    public String getCurrentGameId() {
        return this.currentGameId;
    }
}

public class PlayersDB {
    private static HashMap<String, Player> allPlayers = new HashMap<>();

    public static void addPlayer(String userId, Player newPlayer) {
        allPlayers.put(userId, newPlayer);
    }

    public static Player getPlayer(String userId) {
        return allPlayers.get(userId);
    }

    public static void addNewPlayerToDB(Game game) {
        try (Connection conn = Main.getConnection()) {
            PreparedStatement stmt = conn.prepareStatement("INSERT INTO players (gid)
values (?)", Statement.RETURN_GENERATED_KEYS);
            stmt.setLong(1, gid);
            stmt.executeUpdate();
            try (ResultSet generatedKeys = stmt.getGeneratedKeys()) {
                if (generatedKeys.next()) {
                    id = generatedKeys.getInt(1);
```

```
            }
            else {
                throw new SQLException("Creating player failed, no ID obtained.");
            }
        }
        stmt.close();
    } catch (SQLException e) {
        e.printStackTrace();
        this.execute(new SendMessage(msg.chat().id(), this.translation.ERROR() +
e.getMessage()).replyToMessageId(msg.messageId()));
    }
    // notify queued users
    try (Connection conn = Main.getConnection()) {
        PreparedStatement stmt = conn.prepareStatement("SELECT tgid FROM players
where gid=?");
        stmt.setLong(1, gid);
        ResultSet rs = stmt.executeQuery();
        String startMsg = this.translation.GAME_STARTING_IN(msg.chat().title());
        while (rs.next()) {
            SendMessage send = new SendMessage(rs.getInt(1), startMsg);
            this.execute(send);
        }
        stmt.close();
    } catch (SQLException e) {
        e.printStackTrace();
        this.execute(new SendMessage(msg.chat().id(), this.translation.ERROR() +
e.getMessage()).replyToMessageId(msg.messageId()));
    }
    this.execute(new SendMessage(gid,
String.format(this.translation.GAME_START_ANNOUNCEMENT(), msg.from().id(),
```

```
msg.from().firstName(), wait, id)).parseMode(ParseMode.HTML), new
EmptyCallback<>());
        games.put(gid, this);
        addPlayer(msg);
        startTime = System.currentTimeMillis() + wait * 1000L;
        int i;
        i = 0;
        while (wait > remindSeconds[i]) {
            ++i;
        }

        schedule(this::remind, (wait - remindSeconds[--i]) * 1000);
        this.logf("Game created in %s [%d]", msg.chat().title(), msg.chat().id());
    }

    public static Game getPlayerFromDB(String id) {
        try (Connection conn = Main.getConnection()) {
            PreparedStatement stmt = conn.prepareStatement("SELECT players (gid) values
(?)", Statement.RETURN_GENERATED_KEYS);
            stmt.setLong(1, gid);
            stmt.executeUpdate();
            try (ResultSet generatedKeys = stmt.getGeneratedKeys()) {
                if (generatedKeys.next()) {
                    id = generatedKeys.getInt(1);
                }
                else {
                    throw new SQLException("Getting Player From DB");
                }
            }
            stmt.close();
```

```
        } catch (SQLException e) {
            e.printStackTrace();
            this.execute(new SendMessage(msg.chat().id(), this.translation.ERROR() +
e.getMessage()).replyToMessageId(msg.messageId()));
        }
}
```

Додаток Б

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО- НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробканастільноїгри «Монополія» мовоюJava та бібліотекою Telegram API

ВИКОНАВ СТУДЕНТ 5 КУРСУ
ГРУПИ ПІЗ-51
КЕНГЕРЛІ ЕЛЬМАР ФАІГОВИЧ
КЕРІВНИК РОБОТИ
НЕГОДЕНКО ОЛЕНА ВАСИЛІВНА

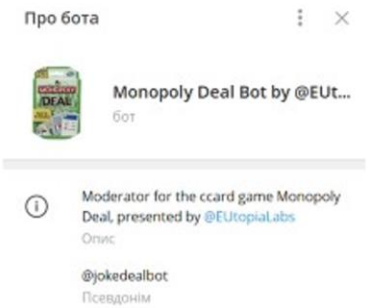
Київ – 2022

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

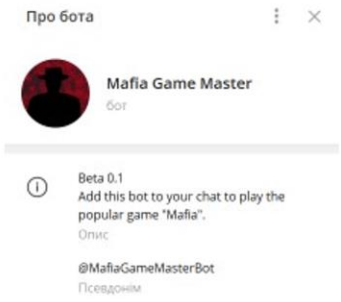
- **Мета роботи –** спрощення ігрового процесу настільної гри «Монополія» за допомогою телеграм-боту на мові програмування Java.
- **Об'єкт дослідження –** процес ігрового досвіду за допомогою застосування телеграм-ботів
- **Предмет дослідження –** телеграм бот на мові програмування Java

АНАЛОГИ

Телеграм-бот Monopoly Deal



Телеграм-бот Mafia Game Master



3

ПОРІВНЯННЯ З АНАЛОГАМИ

	MonopolyGameBot	Monopoly Deal	Mafia Game Master
Графічний контент	+	-	-
Загальна статистика	+	-	-
Підтримка декількох мов	+	-	+
Система досягнень	+	-	-

4

ТЕХНІЧНІ ЗАВДАННЯ

- *Розробити серверний застосунок для можливості грати в «Монополію» через телеграм-бота*
- *Розробити графічні елементи, систему статистики та досягнень для поліпшення ігрового досвіду*
- *Реалізувати можливість створення кімнат для гри та деякі команди адміністратора*

5

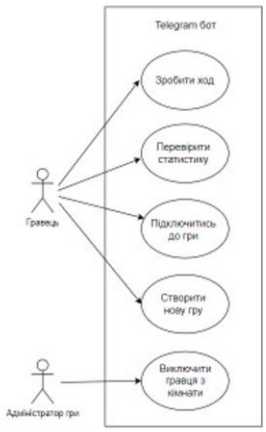
ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

6

ДІАГРАМА КЛАСІВ

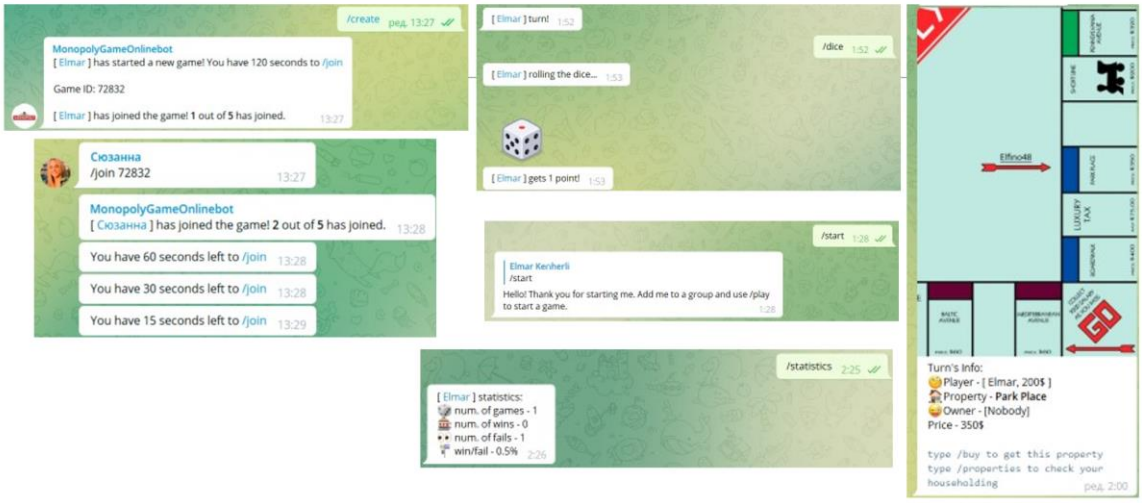
7

ДІАГРАМА ПРЕЦЕНДЕНТІВ



8

ЕКРАННІ ФОРМИ



9

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

Кенгерлі Е.Ф, Розробка настільної гри «Монополія» мовою Java та бібліотекою TelegramAPI / НАУКОВО -ТЕХНІЧНА КОНФЕРЕНЦІЯ «ЗАСТОСУВАННЯ ПРОГРАМНОГО ЗАБЕСПЕЧЕННЯ В ІНФОКОМУНІКАЦІЙНИХ ТЕХНОЛОГІЯХ» Збірник тез 20.04.2022, ДУТ, м. Київ – К.: ДУТ, 2022.

Кенгерлі Е.Ф, Розробка настільної гри «Монополія» мовою Java та бібліотекою TelegramAPI / Міжнародна студентська конференція «Наука сьогодення: від досліджень до стратегічних рішень» 17.06.2022, м . Івано-Франківськ

10

ВИСНОВКИ

1. Проаналізовано доступні реалізації ігрових застосунків для месенджерів
2. Зареєстровано новий бот та реалізовано серверний додаток для взаємодії з ботом
3. Реалізовано графічні елементи та система статистики для гри
4. Виконано тестування продукту користувача

Перспективи подальших досліджень:
В залежності від наявності майбутньої аудиторії перспективним є розвиток додаткових функції застосунку, таких як додавання штучного інтелекту замість реальних гравців

11

ДЯКУЮ ЗА УВАГУ!