КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет комп'ютерних наук та кібернетики

Кафедра Інтелектуальних прикладних систем

Кваліфікаційна робота
на здобуття ступеня бакалавра

за спеціальністю 121 Інженерія програмного забезпечення

на тему:

РОЗРОБКА ОНЛАЙН ПЛАТФОРМИ З ВИКОРИСТАННЯМ TELEGRAM БОТІВ ДЛЯ ДОПОМОГИ ШКОЛЯРАМ

Виконав студент 4-го курсу

Нікіта ПУПОВ _____
(Підпис)

Науковий керівник:

доцент

Олександр ГАЛКІН _____
(Підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри інтелектуальних програмних систем
25.05.2022 р., протокол № 10

Завідувач кафедри

Олександр ПРОВОТАР _____
(підпис)

Київ - 2022

РЕФЕРАТ

Обсяг роботи 31 сторінка, 31 ілюстрація, 3 джерела посилань.

TELEGRAM БОТ, ОНЛАЙН ПЛАТФОРМА, РОЗВ’ЯЗОК ШКІЛЬНИХ ЗАВДАНЬ, ВЕБ СЕРВІС, МІКРОСЕРВІСНА АРХІТЕКТУРА

Об’єктом роботи є процес створення платформи для вирішення шкільних завдань і надання доступу до неї користувачу за допомогою інтерфейса Telegram бота. Предметом роботи є програмний засіб для допомоги з розв’язками шкільних завдань.

Метою роботи є створення програмного засобу, що реалізує собою платформу для допомоги у вирішенні шкільних завдань з різних предметів.

Інструменти розроблення: середовище розробки Intellij Idea 2019.2.4, мова програмування Java з використанням фреймворку Spring. Платформа хмарних обчислень Amazon Web Services.

Результати роботи: виконано аналіз існуючих практик по створенню та архітектурі веб сервісів, вивчено інтерфейс Telegram по використанню ботів, створено повноцінну платформу для вирішення шкільних завдань, що дозволяє якісно розв’язувати шкільні завдання за невеликий час.

Програмний продукт може використовуватися школярами для отримання правильних розв’язків завдань та аналізу своїх помилок на їх базі.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

AWS – Amazon Web Services, платформа хмарних обчислень;

БД – база даних;

ТЗ – технічне завдання;

ПЗ – програмне забезпечення;

API - Application Programming Interface

JWT – Json Web Token

REST API - Representational State Transfer API, архітектурний стиль надання веб-інтерфейсу сервіса

ВСТУП

Оцінка сучасного стану об’єкта розробки. В етап всесвітньої пандемії все більш критичною стає проблема навчання. Багато закладів переходять на навчання онлайн. Значно знижується якість викладання шкільних предметів, зменшується індивідуальність викладання. Багато школярів не можуть зрозуміти, як правильно виконувати свої завдання та через деякі причини (сором, нерозуміння) не можуть запитати про це у викладача на онлайн занятті.

Актуальність роботи та підстави для її виконання. Багато школярів вже не можуть отримати приватну консультацію щодо вирішення завдань та зрозуміти, який саме розв’язок буде найбільш правильним. Існує потреба у швидкому поясненні та наведенні правильного розв’язку. Оскільки через карантин, її досить складно отримати особисто, виникає попит на онлайнсервіси, які змогли б це зробити.

Мета й завдання роботи. Метою кваліфікаційної роботи є створення платформи, що допоможе школярам вирішувати свої завдання швидко та зручно. Для досягнення цієї мети поставлено такі завдання:

- Дослідити існуючі платформи для виконання завдань
- Дослідити інструменти для побудови платформи та обрати найбільш зручний
- Розробити ТЗ для продукта
- Розробити платформу, виконавши основні вимоги ТЗ

Об'єкт, методи й засоби розроблення. Об’єктом роботи є процес створення платформи для вирішення шкільних завдань і надання доступу до неї користувачу за допомогою інтерфейса Telegram бота. Вивчення та аналіз архітектурних патернів, систем хмарних обчислень.

Під час розробки програмного продукту використовувалася еволюційна модель, заснована на таких принципах. Розробляється початкова версія продукту, яка передається кінцевим користувачам для оцінки, після

чого продукт дороблюється, враховуючи думку замовника. Аналогічно розробляються, передаються й оцінюються проміжні версії програмного продукту, поки не з’явиться повністю готовий продукт, який відповідає всім вимогам замовника. Процеси специфікації, розробки та атестації програмного продукту ведуться паралельно з постійним обміном інформації між ними.

Таким чином, створивши мінімальний функціонал нашої платформи (так звану MVP), ми одразу почали проводити тестування. Ми дозволили доступ до платформи невеликій кількості замовників і дали їм змогу безкоштовно отримувати допомогу по вирішенні своїх завдань. Так ми змогли виявити та виправити багато помилок. Ми постійно просимо користувачів залишати відгуки та використовуємо їх побажання при подальшій розробці продукту. Постійний діалог з нашими користувачами дозволив нам швидко виділити найбільш корисний для них функціонал і задовольнити їх потреби.

РОЗДІЛ 1 АНАЛІЗ ПРОЕКТУ ТА ІНСТРУМЕНТІВ

1.1 Технічне завдання

Планувалося розробити платформу для допомоги школярам у вирішенні різних завдань зі шкільного курсу. На цій платформі школяр міг би розмістити своє завдання, а якась людина вирішити її і таким чином заробити гроші. Сервіс буде отримувати комісію від вирішених завдань.

Система має підтримувати три види користувачів, що матимуть різний функціонал на платформі:

- Замовники (користувачі, що пропонують гроші за вирішення завдань)
- Виконавці (користувачі, що вирішують завдання замовників, отримуючи за це гроші)
- Адміністратори (можуть змінювати деякі параметри системи, редагувати доступ та ролі користувачів, тощо)

Таким чином, щоб платформа працювала, вона повинна реалізувати такі основні функції:

- Інтерфейс користувача для замовника
- Інтерфейс користувача для виконавця
- Можливість замовнику поповнювати баланс на платформі за допомогою онлайн оплати
- Можливість виконавцю перевести гроші з балансу платформи на свій рахунок
- Можливість працювати з матеріалами до задач (картинки, документи) та зберігати їх
- Технічна підтримка
- Реферальні посилання
- Можливість користувачам змінювати мову

- Автоматичне повернення коштів в разі невирішення завдання
- Функціонал для адміністратора

1.2 Вибір інструментів для побудови програми

Відповідно до ТЗ, нам потрібно було швидко і якісно розробити інтерфейс для користувача. Найбільш гнучким варіантом було б створювати веб-сайт за допомогою таких JavaScript бібліотек як React, Angular, тощо. Але це зайняло б досить багато часу та сил. Тому було вирішено використовувати Telegram. Адже це досить популярний месенджер, що надає API для створення ботів – програм, що взаємодіють з користувачами.

Також треба було вирішити проблему з цілодобовим доступом користувачів до сервісу, розміщенню БД, можливістю швидкої побудови складної інфраструктури. Для таких послуг було вирішено використати Amazon Web Services, що представляє собою популярну платформу хмарних обчислень, якою користуються тисячі компаній по всьому світу.

Наступним етапом був аналіз сервісів, що спеціалізуються на банківських переказах. Найбільш доцільною виявився «Liqpay». Він надає досить зручний та захищений API для інтеграції оплат та виплат з нашого сервісу.

Як мову програмування було обрано Java. Разом з Spring Boot, вона дозволяє швидко та зручно створювати веб сервіси. Також, вже існують бібліотеки для інтеграції з Telegram API та «Liqpay».

1.3 Вибір типу архітектури

Перед початком створення нашої системи, треба було проаналізувати основні патерни побудови складної системи та обрати найбільш спроможний для масштабування патерн. Серед видів архітектур найбільш популярними є мікросервісна та монолітна архітектури.

Монолітна архітектура. Моноліт - давнє слово, яке посилається на величезний камінь. Незважаючи на те, що цей термін широко використовується сьогодні, сутність залишається незмінним для всіх варіантів вживання. В інженерії програмного забезпечення монолітний підхід відноситься до однієї неподільної одиниці. Концепція монолітного програмного забезпечення полягає в різних компонентах програми, об'єднаних в єдину програму на одній платформі. Як правило, монолітний додаток складається з бази даних, клієнтського інтерфейсу користувача та серверного додатку. Всі частини програмного забезпечення уніфіковані, а всі його функції керуються в одному місці.

Мікросервісна архітектура. Мікросервіс - це тип архітектури програмного забезпечення, який зосереджується на створенні ряду автономних компонентів, що складають додаток. На відміну від монолітних додатків, побудованих як єдина неподільна одиниця, програми побудовані на мікросервісній архітектурі складаються з декількох незалежних компонентів, які спілкуються через API

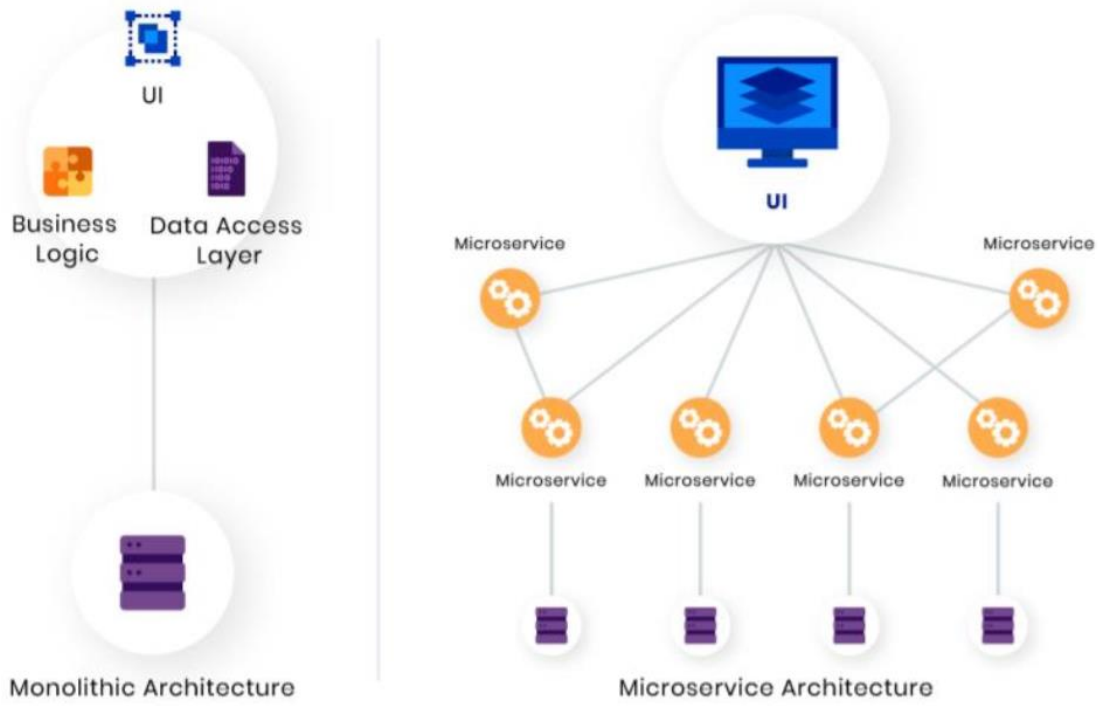

Рисунок 1.3 Монолітна і мікросервісна архітектури

Оскільки наш сервіс планує розвиватися в багатьох напрямках, нам не зручно буде працювати з монолітною архітектурою. Адже якщо усі функції нашої платформи будуть знаходитись в одній програмі, досить скоро нам буде важко в ній орієнтуватися та робити якісь зміни. Хотілося б розподілити функціонал платформи по різним програмам, щоб зменшити навантаження на кожну з програм та зробити платформу більш гнучкою.

Тому було обрано притримуватися мікросервісної архітектури під час реалізації платформи.

1.4 Види масштабування веб архітектури

Оскільки ми плануємо, що в майбутньому наша платформа буде обслуговувати велику кількість клієнтів, нам треба завчасно подбати про те, що програма зможе відповісти на всі запити клієнтів та не створювати помилок при збільшенні навантаження. Для того, щоб динамічно збільшувати потужність системи при збільшенні навантаження на неї, можна використовувати різні види масштабування.

Вертикальне масштабування. Цей вид масштабування передбачує збільшення продуктивності кожного компонента системи з метою підвищення загальної продуктивності. Масштабування у цьому контексті означає можливість змінити в існуючій обчислювальній системі компоненти, більш потужні та швидкі. Це найпростіша можливість масштабування, так як не потрібно жодних змін у підручних програмах, що працюють на таких системах. Тобто, якщо на нашому сервері не вистачає оперативної пам’яті, ми можемо додати до нього ще один SSD-накопичувач.

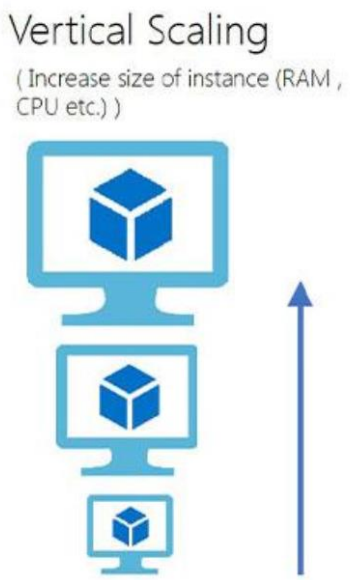

Рисунок 1.4.1 Вертикальне масштабування

Але постійно збільшуючи потужність нашого сервера, його подальше покращення стає все дорожчим.

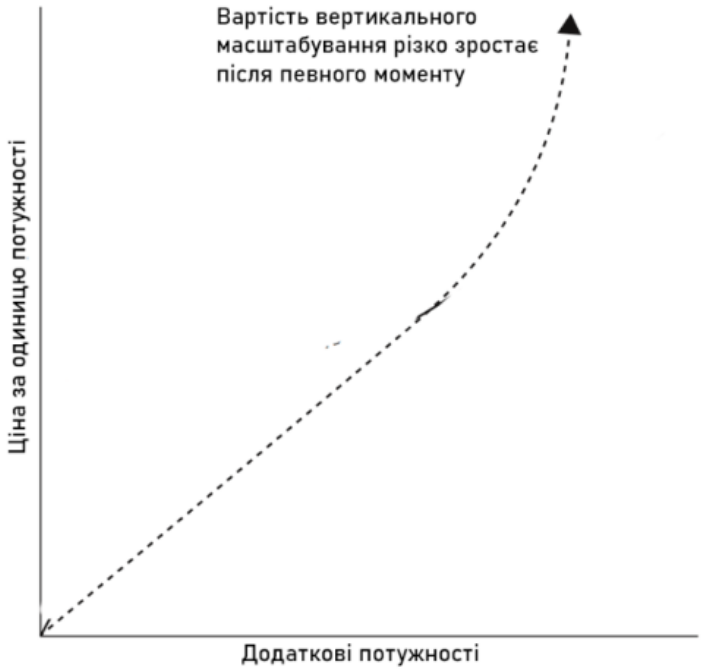

Рисунок 1.4.2 Залежність збільшення вартості від потужності при вертикальному масштабуванні

Горизонтальне масштабування. Цей вид масштабування передбачає розбиття системи на більш дрібні структурні компоненти та рознесення їх по окремим фізичним машинам (або їх групам), і збільшення кількості серверів, що паралельно виконують одну і ту ж функцію. Масштабованість в цьому контексті означає можливість додавати до системи нові вузли, сервери для збільшення загальної продуктивності. Цей спосіб масштабування може вимагати внесення змін до програми, щоб програми могли повною мірою користуватися дедалі більшій кількості ресурсів.

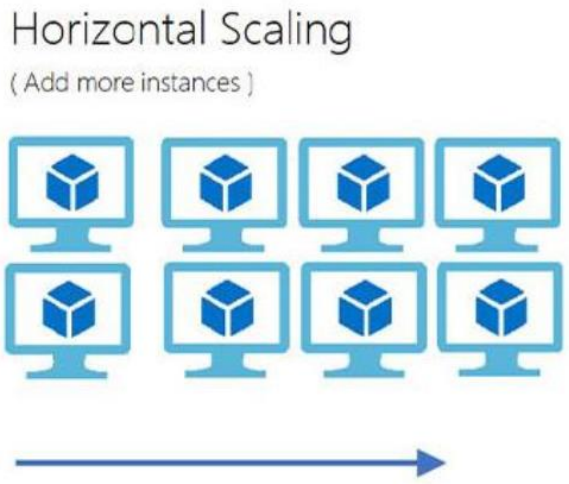

Рисунок 1.4.3 Горизонтальне масштабування

Таким чином, при збільшенні навантаження на сервери, будуть створюватися нові сервера, збільшуючи потужність системи.

Рисунок 1.4.4 Залежність вартості від потужності

Коли ми реалізуємо горизонтальне масштабування, нам треба намагатися порівну розподілити навантаження на кожний сервер. Як рішення, для цього можна використовувати балансування навантаження.

Балансування навантаження (англ. load balancing). У комп'ютерах балансування навантаження розподіляє навантаження між декількома обчислювальними ресурсами, такими як комп'ютери, комп'ютерні кластери, мережі. Мета балансування навантаження - оптимізація використання ресурсів, максимізація пропускної здатності, зменшення часу відгуку і запобігання перевантаження будь-якого одного ресурсу. Балансування навантаження передбачає зазвичай наявність спеціального програмного забезпечення або апаратних засобів, таких як багаторівневий комутатор або система доменних імен, як серверний процес.

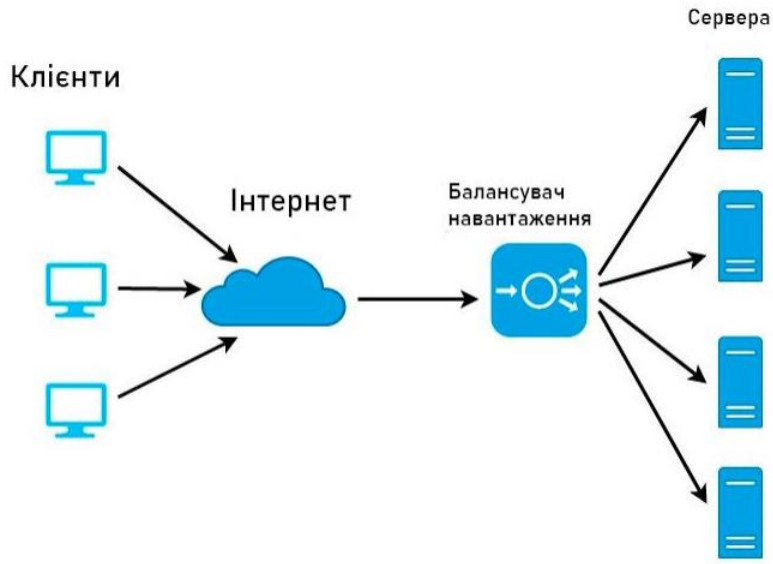

Рисунок 1.4.5 Приклад роботи окремого серверу, що відповідає за розподіл навантаження

1.5 Неперервна інтеграція та безперервна поставка

За останні роки процедура розроблення програмного забезпечення набуває все більшої складності. Це зумовлено зростанням комплексності функціональних та нефункціональних вимог до програмного продукту. Саме тому виникла необхідність у застосуванні нових та гнучких методологій розроблення ПЗ. Одним з таких підходів є DevOps.

DevOps (акронім від англ. Development і operations) – методологія розроблення програмного забезпечення, скерована на активну взаємодію та інтеграцію фахівців з розроблення та фахівців з інформаційнотехнологічного обслуговування. Базується на ідеї про тісну взаємозалежність розроблення та експлуатації програмного забезпечення для того, щоб допомагати організаціям швидше створювати і оновлювати програмні продукти і сервіси.

Завдання DevOps – зробити процес розроблення і постачання програмного забезпечення узгодженим з експлуатацією, часто ці завдання вирішують за підтримки автоматичних засобів.

Рисунок 1.5.1 Завдання, що вирішуються DevOps підходом

Одним з процесів, які мають бути реалізовані за підходом DevOps - неперервна інтеграція та безперервна поставка.

Неперервна інтеграція (англ. Continuous Integration). Це практика розроблення програмного забезпечення, яка полягає у виконанні частих автоматизованих складань проекту для якнайшвидшого виявлення та вирішення інтеграційних проблем. У звичайному проекті, де над різними частинами системи розробники працюють незалежно, стадія інтеграції є завершальною. Вона може непередбачувано затримати закінчення робіт. Перехід до неперервної (постійної) інтеграції дає змогу знизити трудомісткість інтеграції і зробити її передбачуванішою внаслідок найбільш раннього виявлення та усунення помилок і суперечностей.

Безперервна поставка (англ. Continuous Delivery). Являє собою програмний інженерний підхід, за яким команди виробляють програмне забезпечення в коротких циклах, гарантуючи, що програмне забезпечення може бути надійно випущене в будь-який час. Вона спрямована на створення, тестування і випуск програмного забезпечення швидше і частіше. За таким підходом можна знизити витрати, час і ризик доставки зміни, що надає більше додаткових оновлень для додатків у виробництві. Прямолінійний і повторюваний процес розгортання має важливе значення для безперервної доставки.

При розробці нашої платформи активно використовуються ці корисні методології.

1.6 Авторизація та автентифікація

Частим питанням, яке постає при створенні додатків, є запобігання та протидія використанню функціонала від імені іншої людини. В новинах ми часто чуємо про витік паролей та конфіденціальної інформації про користувачів різних сервісів. А відповідальність за це лежить як на злочинців, що дістають цю інформацію, так і на власників цих сервісів, що не змогли правильно її захистити. Таким чином, велику увагу приділяють забезпеченню безпеки доступу до сервісу і зберігання конфіденційної інформації про користувачів.

Також аби зловмисники не могли використовувати сервіс під чужим ім’ям, використовується авторизація та автентифікація.

Автентифікація – це підтвердження того, ким є користувач на вході. Це проходження перевірки автентичності.

Авторизація – це права на якісь дії, які користувач отримує після входу.

Як проходить процедура аутентифікації? Наприклад, користувач має доступ до електронної поштової скриньки і хоче прочитати свої листи, що прийшли на email. Зайшовши на сайт пошти, він може побачити тільки загальні сторінки, новини, тощо. Але свої листи він може прочитати лише після введення та відправки свого логіну та паролю. Адже на даному єтапі система не може ідентифікувати цього користувача. Що може перевіряти система при автентифікації:

- чи існує користувач з таким ім’ям
- чи збігається введений пароль з його обліковим записом
- тощо

Якщо користувач пройшов автентифікацію - він дійсно той, ким здається. Після цього починається процес авторизації. Авторизація допомагає системі визначити, що дозволено тим чи іншим користувачам:

- право налаштовувати рекламу на сайті
- право відправляти листи
- право обмежувати доступ іншим користувачам

Тобто, авторизація перевіряє наявність прав на конкретні дії.

В нашій системі має бути передбачений різний функціонал для замовників, виконавців та адміністраторів. Цей функціонал буде обмежено за допомогою системи ролей. Кожному користувачу буде виділено відповідні ролі. І потім при виконанні запитів від імені користувача, система буде перевіряти чи відповідають ролі користувача запиту, який він надсилає.

Але перед цим користувач має підтвердити хто він за допомогою автентифікації. Для цього в нашій системі будуть зберігатися логін та зашифрований пароль користувача. Таким чином, якщо ці дані потраплять у відкритий доступ, розшифрувати пароль користувача буде неможливо.

Але ми не можемо з кожним запитом передавати логін та пароль користувача, щоб виконати автентифікацію та авторизацію, адже це може поставити під небезпеку його облікові дані (в разі якщо запит буде йти через незахищені канали).

Тож для цього ми будемо використовувати JWT токени. Можемо уявити JWT токен як підпис користувача в запиті. Тобто замість того, щоб передавати свої облікові дані в кожному запиті, користувач буде надсилати JWT токен і таким чином підтверджувати свою особу. За своїм призначенням JWT токен дуже схожий на облікові дані, але він має термін придатності. Після деякого невеликого часу, JWT токен втрачає термін придатності і його треба оновлювати. Приклад JWT токена можна побачити на рисунку 1.6.1

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9          // header (заголовок)
.eyJrZXkiOiJ2YWwiLCJpYXQiOjE0MjI2MDU0NDV9     // payload (корисне навантаження)
.eUiabuiKv-8PYk2AkGY4Fb5KMZeorYBLw261JPQD5lM   // signature (підпис)
```

Рисунок 1.6.1 Структура JWT токена

Перша частина токену – заголовок. Він закодовує звичайний json об’єкт, що зберігає метадані про цей токен.

Друга частина токену – корисне навантаження. Тут зберігається основна інформація про користувача: ім’я користувача, рівень його доступа, інше.

І остання частина токену – підпис, згенерований на основі заголовка та корисного навантаження. Саме він використовується для перевірки того, що jwt є валідним.

В нашому сервісі користувач буде отримувати JWT токен після логіну.

РОЗДІЛ 2 РЕАЛІЗАЦІЯ ПЛАТФОРМИ ДЛЯ ВИРІШЕННЯ ШКІЛЬНИХ ЗАВДАНЬ

2.1 Структура програми

Відповідно до ТЗ було виконано розбиття програми на такі самостійні частини, що були винесені в окремі програми(сервіси):

- «Core» сервіс
- «TelegramBot» сервіс
- «Documents» сервіс

«Core» сервіс. Це головний сервіс, що відповідає за всю основну критичну логіку нашої платформи. Він надає REST API користувачам, за допомогою якого інші сервіси можуть отримувати інформацію та виконувати якісь дії з платформою. До основних обов’язків цього сервісу входять:

- аутентифікація та авторизація користувача
- створення замовлень
- управління користувачами
- інтеграція з системами платежів
- підрахунок статистики по використанню платформи
- управління доступом користувачів при проходженні альфа-тестування
- реферальна система
- надсилати запити в «TelegramBot» сервіс, щоб той міг динамічно реагувати на зміну деякої інформації та відображати ці зміни в інтерфейсі користувача

«TelegramBot» сервіс. Це сервіс, що відповідає за комунікацію з Telegram API. Він створює головний інтерфейс користувача, яким користується як замовник, так і виконавець. Він надає REST API, який Telegram використовує для надсилання подій(повідомлень) від користувача.

Відповідно від команди користувача, цей сервіс може надсилати/видаляти повідомлення до користувача, використовуючи Telegram API бібліотеку. Цей сервіс отримує необхідні дані з «Core» сервісу, використовуючи його REST API.

«Documents» сервіс. Це сервіс, що займається обробкою матеріалів завдань(файлів, картинок, тощо). Він завантажує потрібні файли з Telegram та надсилає їх на збереження в AWS S3(хмарне сховище). Таким чином, в майбутньому ми зможемо проаналізувати ці матеріали завдань та на їх основі підвищити правильність вирішених завдань.

2.2 Розгортання архітектури за допомогою AWS

Коли платформою починають користуватися користувачі, дуже важливо надавати їм можливість цілодобового доступу до платформи. Але розгортання усіх сервісів на власному комп’ютері не гарантує безперервність роботи цих сервісів (може вимкнутися світло, комп’ютер може бути випадково вимкненим).

Саме тому зараз через зростаючу складність веб додатків, набирають популярність платформи хмарних обчислень. Найбільш відомими є:

- Amazon Web services
- Google Cloud
- Microsoft Azure

Нами була обрана платформа «Amazon Web Services», або «AWS». Ця платформа надає дуже багато можливостей для побудови власної архітектури. Технологія дозволяє абонентам мати у своєму розпорядженні повноцінний віртуальний кластер комп'ютерів, який завжди доступний через Інтернет. оперативну пам'ять, жорсткий диск або SSD-накопичувач); операційну систему на вибір; мережу; і попередньо встановлені прикладні

програми, такі як веб-сервер, база даних, CRM і т. д. Кожна система AWS також віртуалізує консольний ввід/вивід (клавіатура, дисплей і миша), що дозволяє користувачам AWS підключитися до своєї системи AWS за допомогою браузера. Виходячи з того, що користувач потребує і оплачує, він може зарезервувати один віртуальний комп'ютер (VM), кластер віртуальних комп'ютерів (VM Cluster), фізичний (реальний) комп'ютер (Server), призначений для його виняткового використання, або навіть кластер фізичних комп'ютерів (Server Cluster). AWS працює в багатьох географічних регіонах, у тому числі в Канаді, Німеччині, Ірландії, Сінгапурі, Токіо, Сіднеї, Пекіні, Лондоні і т. д.

Для побудови та розгортання нашого сервісу ми використовували такі продукти AWS, як:

- Elastic Beanstalk
- SQS
- CodePipeline
- та інші

Elastic Beanstalk. Він надає сервіс PaaS(platform as a service) для розміщення хостингу програм. Еквівалентні сервіси: Google App Engine, Heroku та OpenShift для локального використання. За допомогою нього можна швидко розгорнути гнучку архітектуру, що буде використовувати балансувач навантаження та відповідно до цього навантаження створювати, або видаляти додаткові сервера. Дозволяє створювати різні середовища для програм (production, staging).

Наприклад, на Рис. 2.2.1 за допомогою Elastic Beanstalk створено 2 середовища: production, staging. Кожне середовище складається з групи автоматичного масштабування (Auto Scaling Group), що в залежності від навантаження, може запускати, чи зупиняти нові сервера (Instance). Навантаження між цими серверами розподіляється за допомогою розподілювача навантаження (Load Balancer).

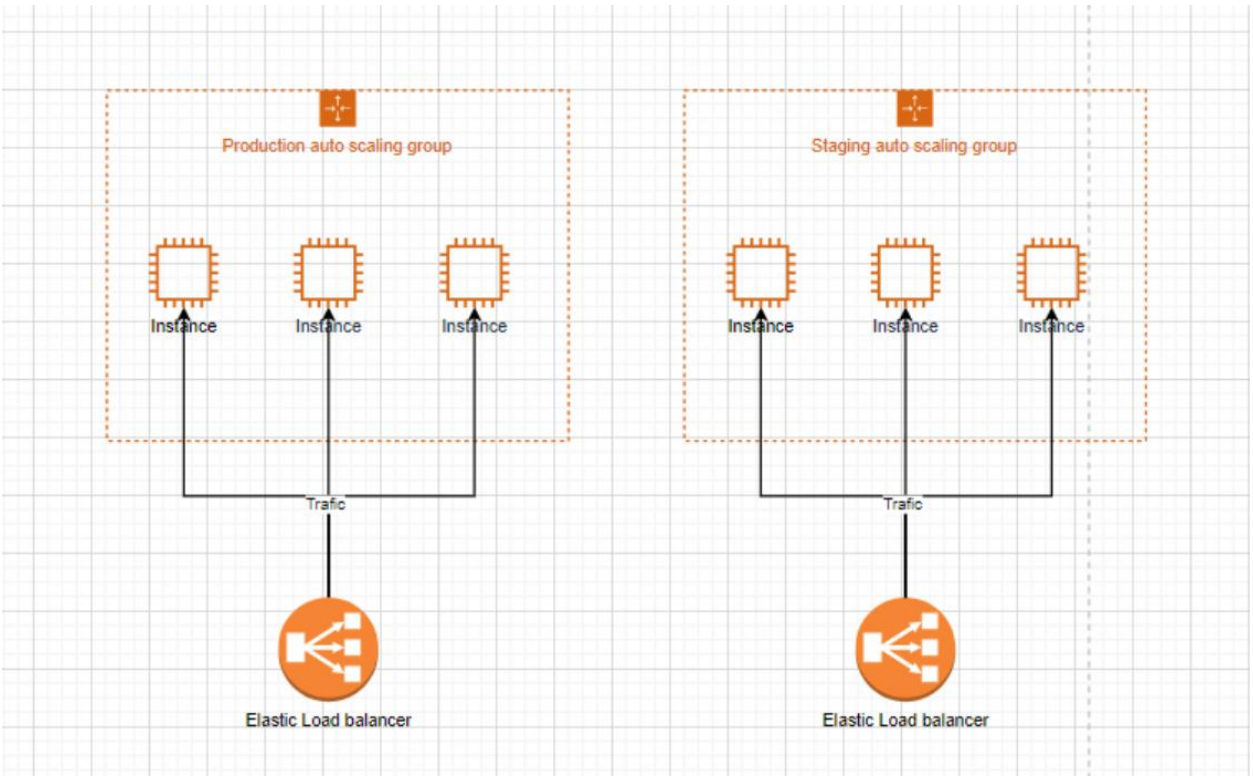

Рисунок 2.2.1 Приклад архітектури, яку створює Elastic Beanstalk

Amazon SQS(Simple Queue Service). Розподілена служба черги повідомлень. Підтримує програмне відправлення повідомлень через програми веб-сервісів як спосіб спілкування через Інтернет. SQS призначений для забезпечення високо масштабованої черги повідомлень. Розробники можуть легко переміщати дані, розподілені між компонентами програми, які виконують різні завдання, не втрачаючи при цьому повідомлення. При цьому досягається висока масштабованість та надійність передачі. Amazon SQS гарантує доставку принаймні один раз. Повідомлення зберігаються на декількох серверах для надмірності та забезпечення їх доступності. Якщо під час доставки повідомлення, сервер недоступний, то повідомлення не може бути вилучено із черги цього сервера і може бути повторно відправлено.

На Рис. 2.2.2 зображено абстрактний приклад архітектури з застовуванням Amazon SQS. Тут є декілька виробників (Producer), що відправляють повідомлення (Message) в SQS чергу. Також є декілька споживачів (Consumer), що періодично роблять запит на отримання

повідомлень з SQS черги. Коли споживач отримує та обробляє повідомлення, він видаляє це повідомлення з SQS черги.

Таким чином, комунікація між виробником та споживачем є асинхронною. Також існує гарантія, що повідомлення, які відправляє виробник, будуть доставлені.

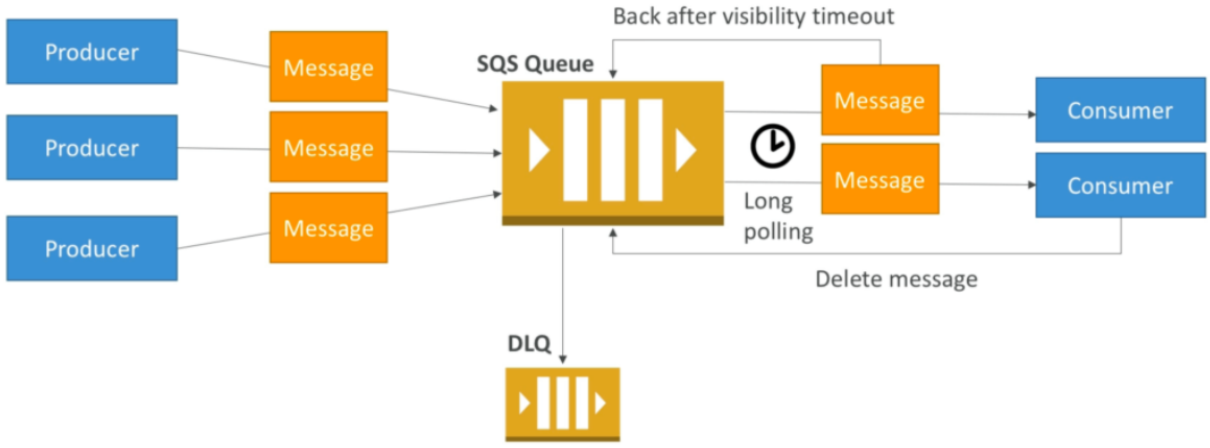

Рисунок 2.2.2 Приклад архітектури з застосуванням SQS

CodePipeline. Це сервіс безперервної доставки, який дозволяє моделювати, візуалізувати й автоматизувати дії, необхідні для випуску програмного забезпечення. Процес розгортання відбувається за концепцією pipeline, тобто труби. Це означає, що кожна збірка розроблюваного програмного продукту повинна пройти кожен етап тестування, перш ніж потрапити у публічне використання. Такий підхід дає змогу якісно тестувати програмний продукт, що збільшує його надійність та стабільність.

Як приклад, на рис. 2.2.3 зображено процес безперервної доставки за допомогою CodePipeline. Спочатку програміст відправляє свої зміни у репозиторій (в нашому випадку Github). Потім за допомогою веб-хуків, Github передає CodeBuild інформацію про те, що нова версія коду з’явилася у репозиторії. CodeBuild завантажує код з репозиторія, виконує тести, компілює та будує програму. Далі побудовану програму він передає до CodeDeploy, що розгортає нову версію веб-сервісу у Elastic Beanstalk.

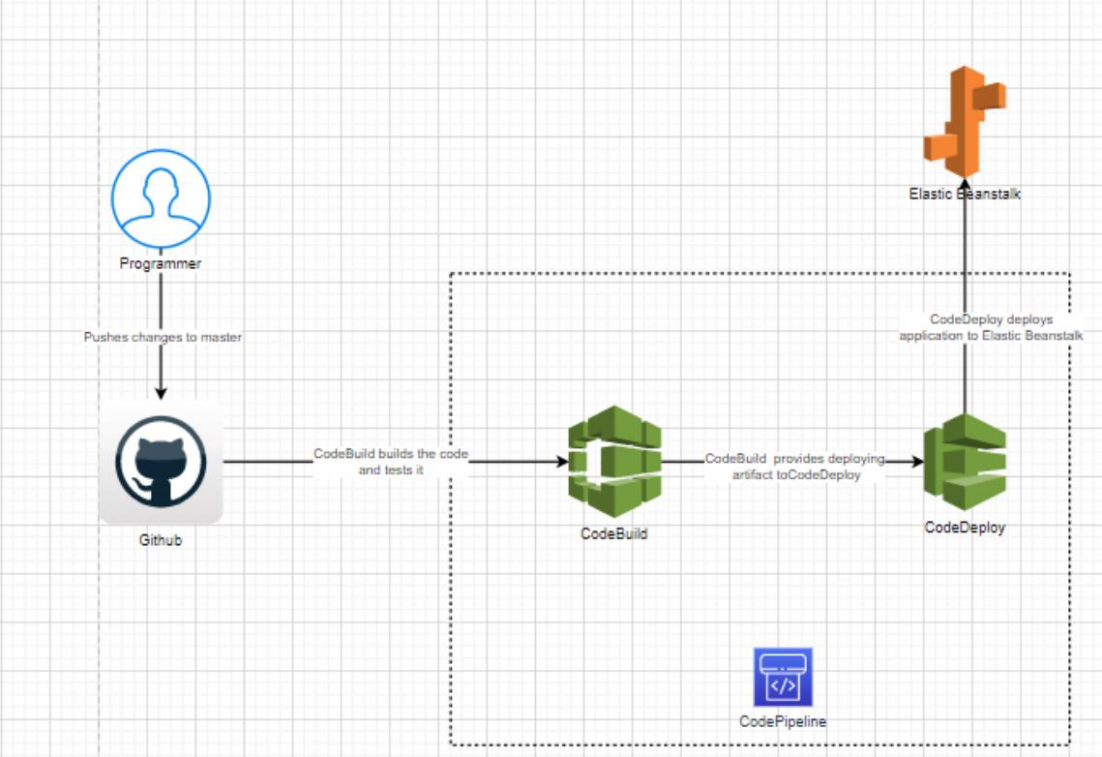

Рисунок 2.2.3 Приклад безперервної доставки з CodePipeline

Таким чином, поєднавши ці сервіси AWS ми змогли створити прозору та гнучку архітектуру для нашої платформи.

2.3 Опис взаємодії користувача з платформою

Взаємодія замовника з платформою. Для того, щоб почати роботу з платформою, замовнику просто необхідно зайти до бота за допомогою Telegram та натиснути на «Start».

Перед нами з’являється основне меню замовника.

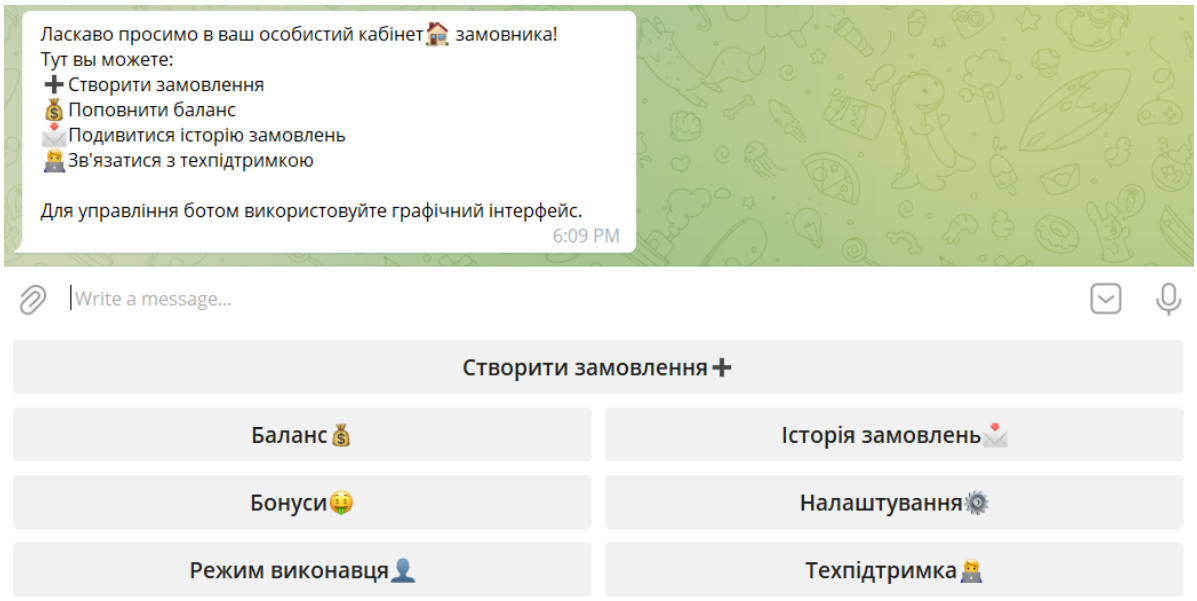

Рисунок 2.3.1 Головна сторінка замовника

На цій сторінці ми можемо створити замовлення, подивитися наш поточний баланс, історію наших замовлень, отримати реферальне посилання, налаштувати мову, звернутися до техпідтримки та перейти в режим виконавця.

Нехай ми зараз у дев’ятому класі та нам потрібно допомогти розв’язати завдання з фізики. Для цього ми натискаємо на “Створити замовлення”.

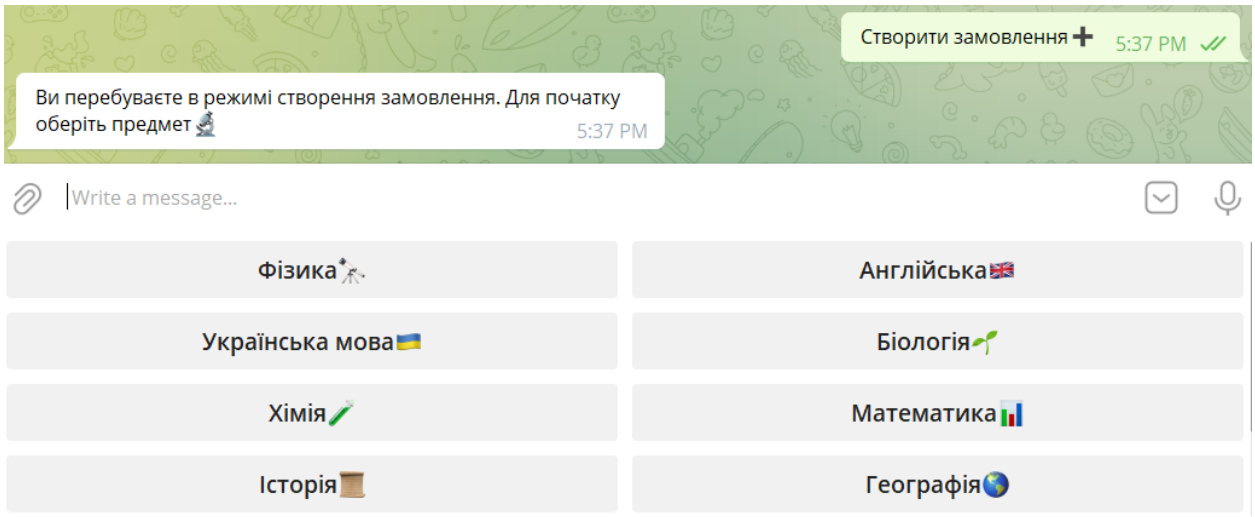

Рисунок 2.3.2 Вибір предмета

Тут обираємо предмет (Фізика).

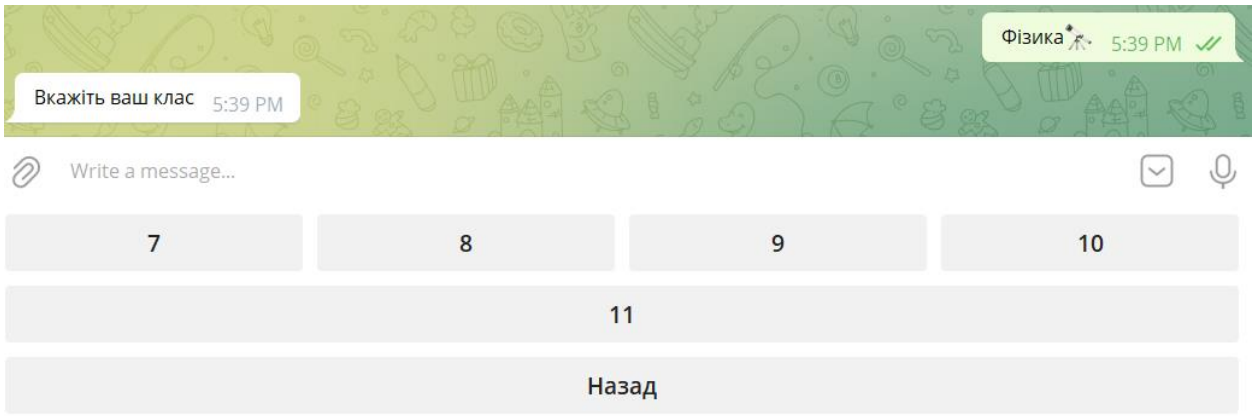

Рисунок 2.3.2 Вибір предмету

Далі обираємо клас(9)

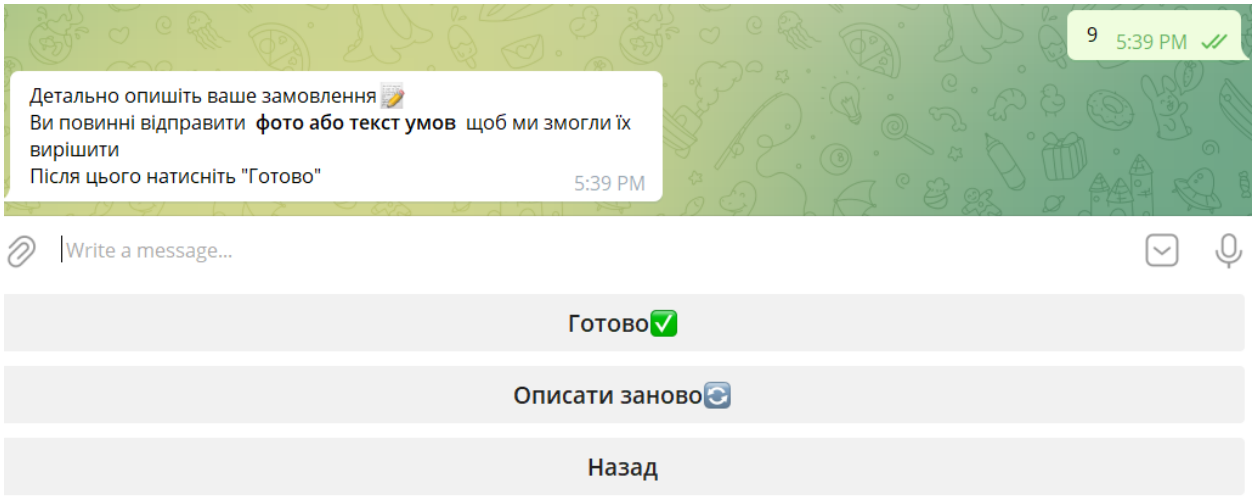

Рисунок 2.3.3 Вибір класу

Як опис замовлення, відправляємо картинку з умовою завдання та потім натискаємо на “Готово”.

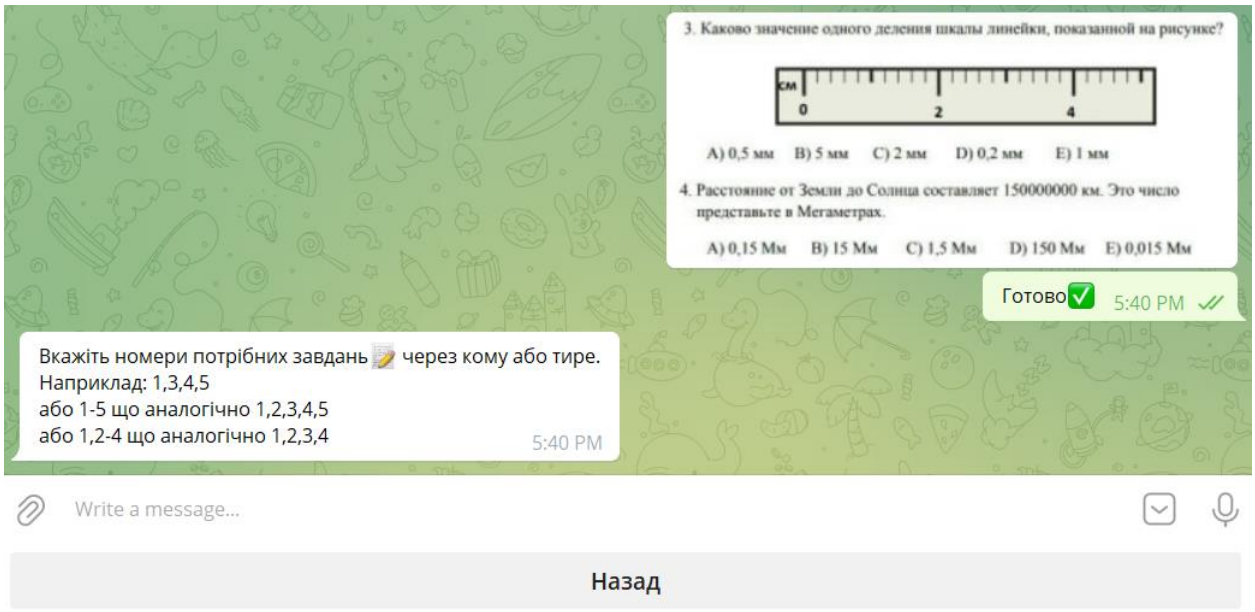

Рисунок 2.3.4 Вибір номерів завдань

Далі вказуємо номери завдань, що необхідно вирішити (в нашому випадку на картинці завдання з номерами 3 та 4)

Рисунок 2.3.5 Вибір типу завдання для третього завдання

Далі вказуємо тип завдання. Від типу залежить мінімальна вартість. В нашому випадку третє завдання тестове, тому обираємо “Тест”.

Рисунок 2.3.6 Вибір типу завдання для четвертого завдання

В нашому випадку четверте завдання тестове, тому також обираємо “Тест”.

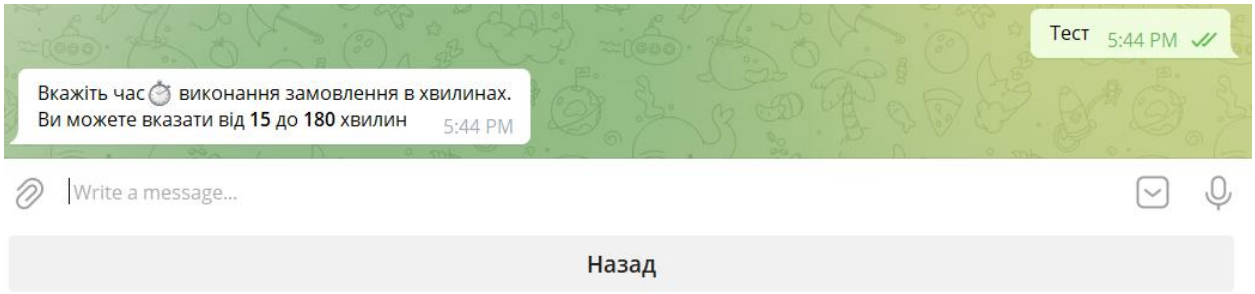

Рисунок 2.3.7 Вибір дедлайну замовлення

Тут замовник обирає граничний термін виконання замовлення. В нашому випадку замовлення досить легке, тому вказуємо 20 хвилин.

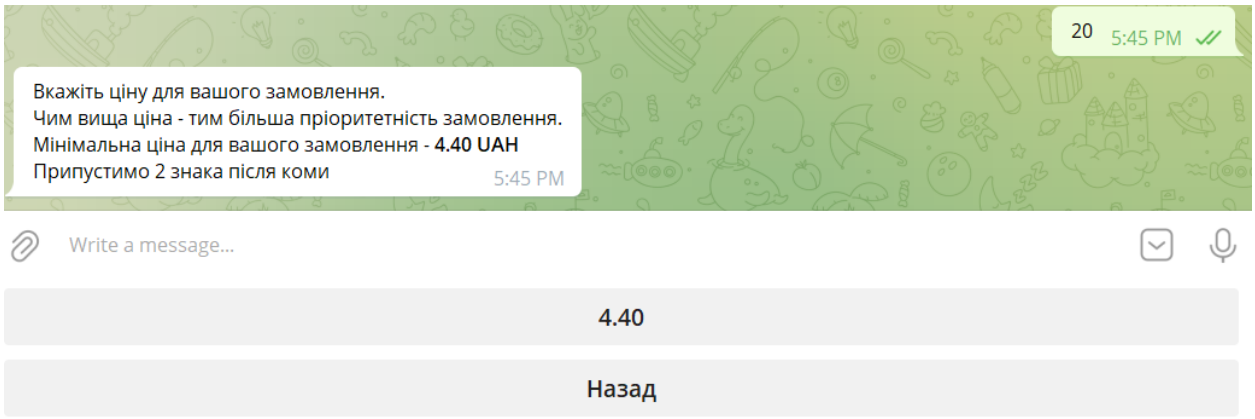

Рисунок 2.3.8 Вказати вартість замовлення

Тут замовник вказує вартість замовлення. Він може вказати будь-яку вартість вище мінімальної. Чим вище вартість, тим з більшою вірогідністю виконавці виконають його. Ми обираємо мінімальну вартість 4.40 грн.

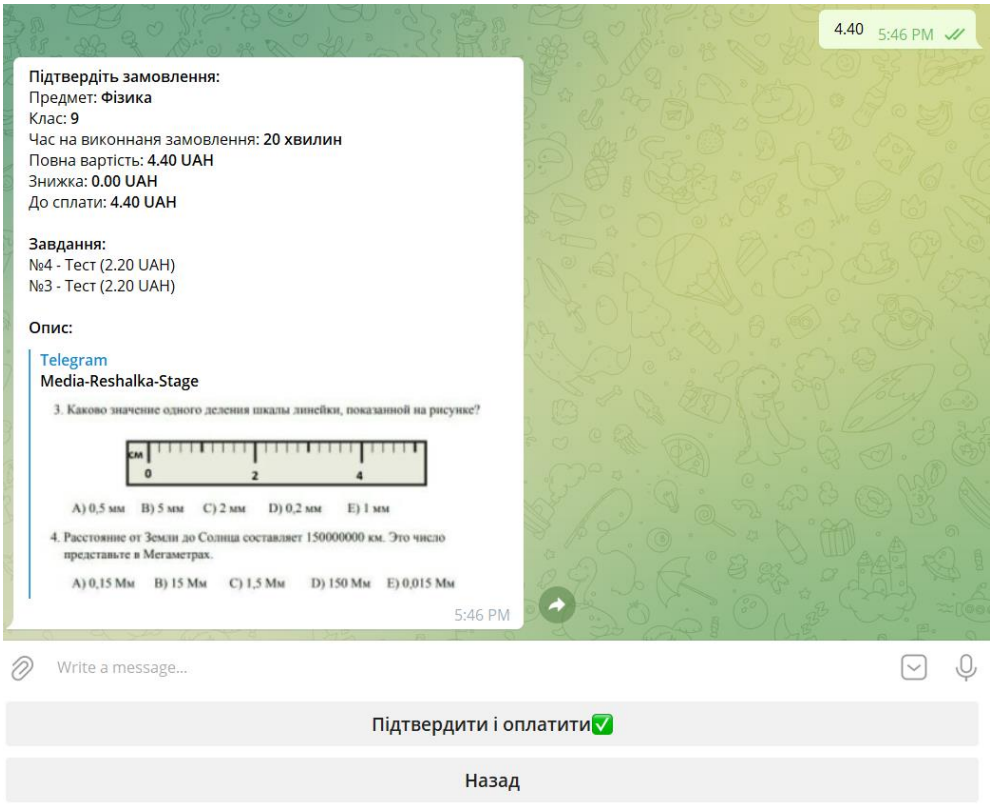

Рисунок 2.3.9 Підтвердження замовлення

Перевіряємо, чи все правильно та підтверджуємо замовлення.

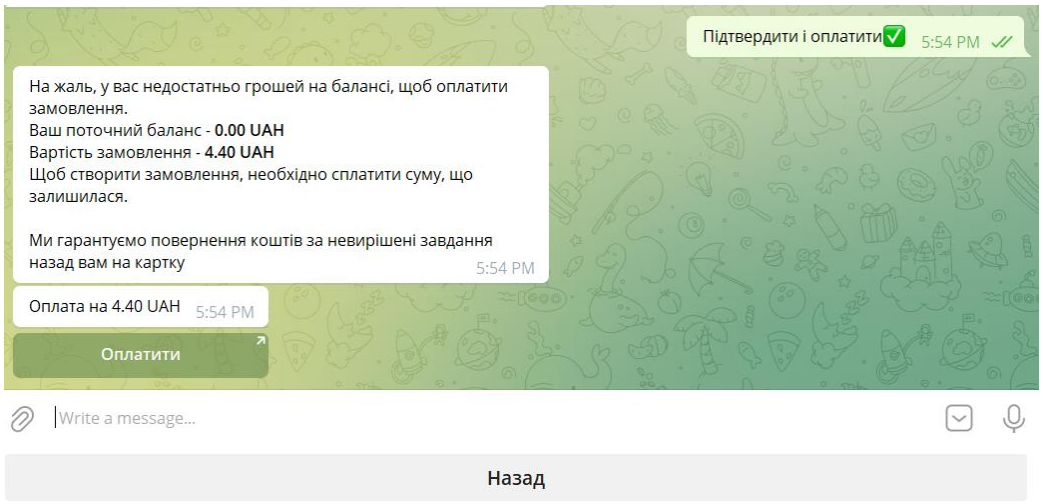

Рисунок 2.3.10 Оплата замовлення

Як бачимо, у нас на балансі немає грошей. Нам потрібно оплатити замовлення. Оплата проходить за допомогою LiqPay.

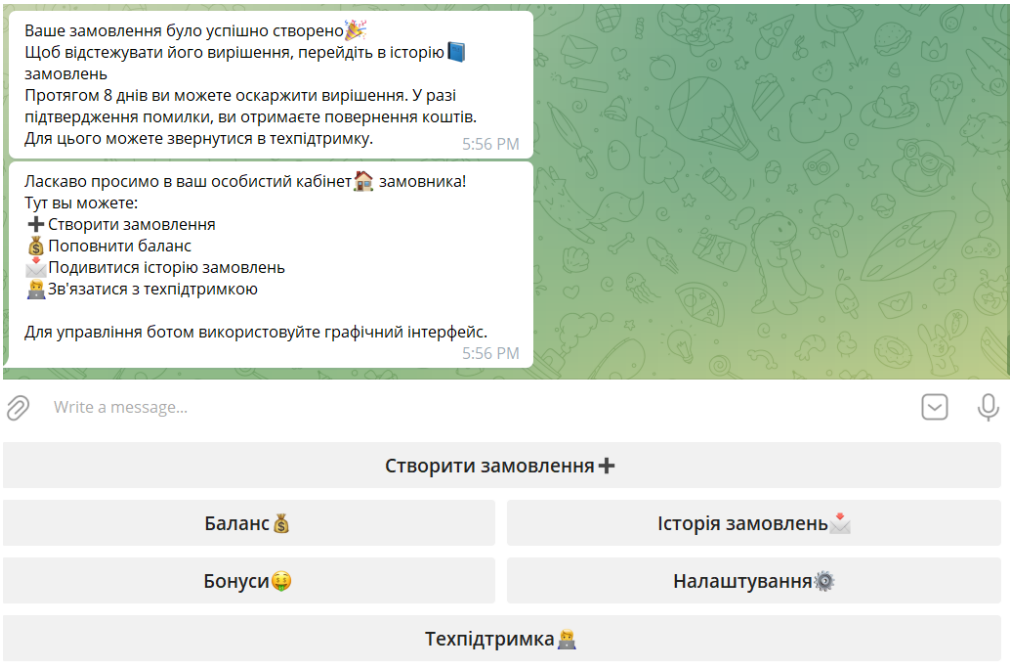

Рисунок 2.3.11 Замовлення створено

Як тільки ми оплатили замовлення, воно створилося. Тепер виконавці зможуть розв’язувати мої задачі. Щоб подивитися розв’язок замовлення, ми переходимо до історії замовлень.

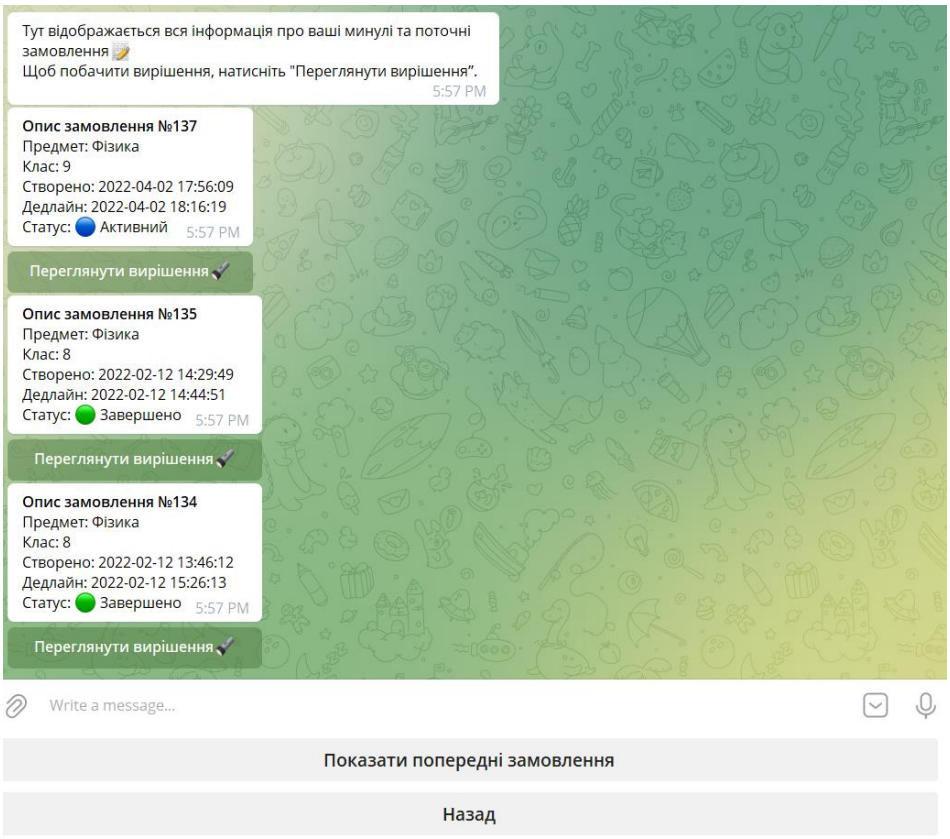

Рисунок 2.3.12 Історія замовлень

Можемо побачити, що раніше я вже створював замовлення. Вони мають завершений статус. І згори бачимо активне замовлення. Натискаємо на перегляд вирішення.

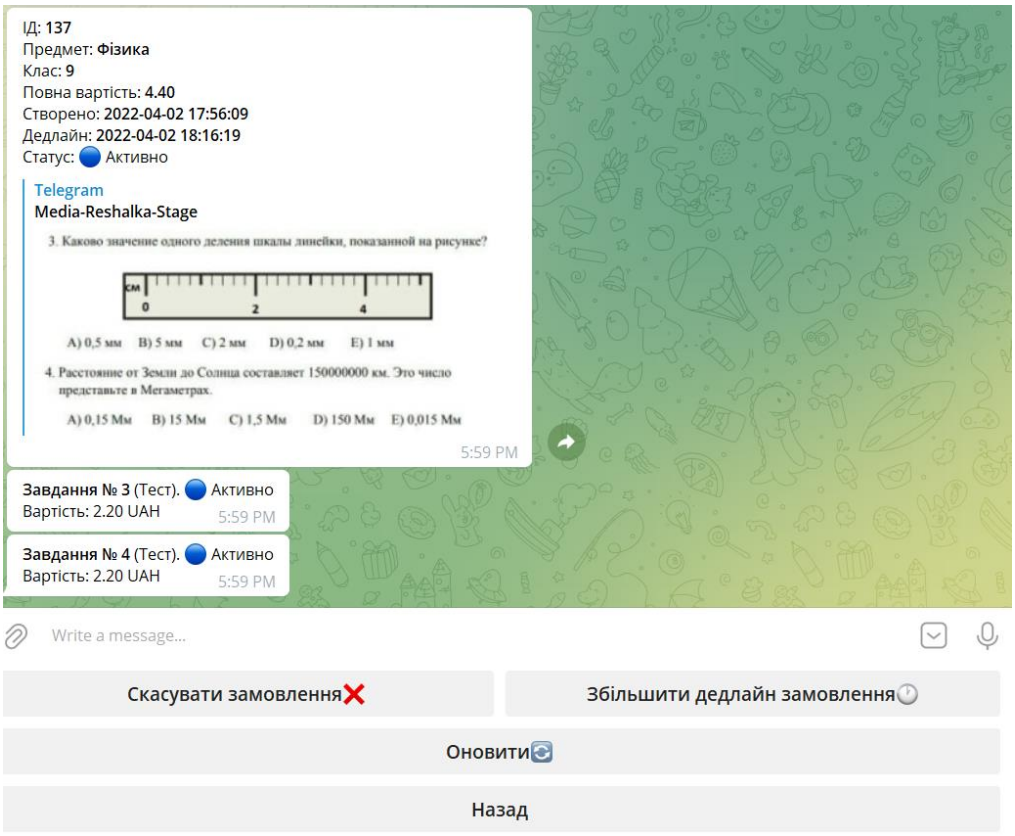

Рисунок 2.3.13 Детальна інформація про замовлення

На цій сторінці ми можемо побачити детальну інформацію про замовлення, скасувати замовлення, збільшити дедлайн замовлення. А також можна побачити статус по кожному завданню. В даний момент всі завдання мають активний статус, що означає що поки що завдання ніхто не вирішує. Коли виконавець починає вирішувати моє завдання, його статус змінюється на “Вирішується”. Як тільки завдання було вирішено, ми отримаємо сповіщення про це.

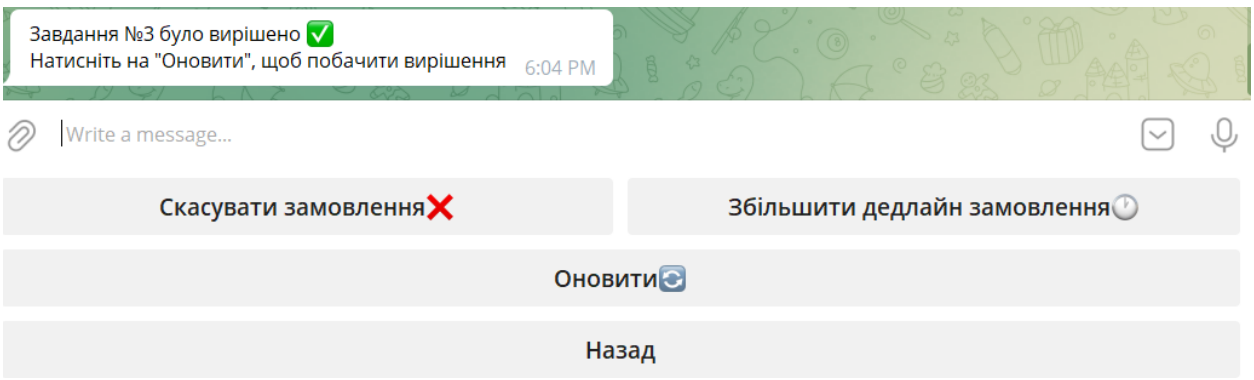

Рисунок 2.3.14 Сповіщення про вирішене завдання

Після оновлення, можна побачити розв’язок:

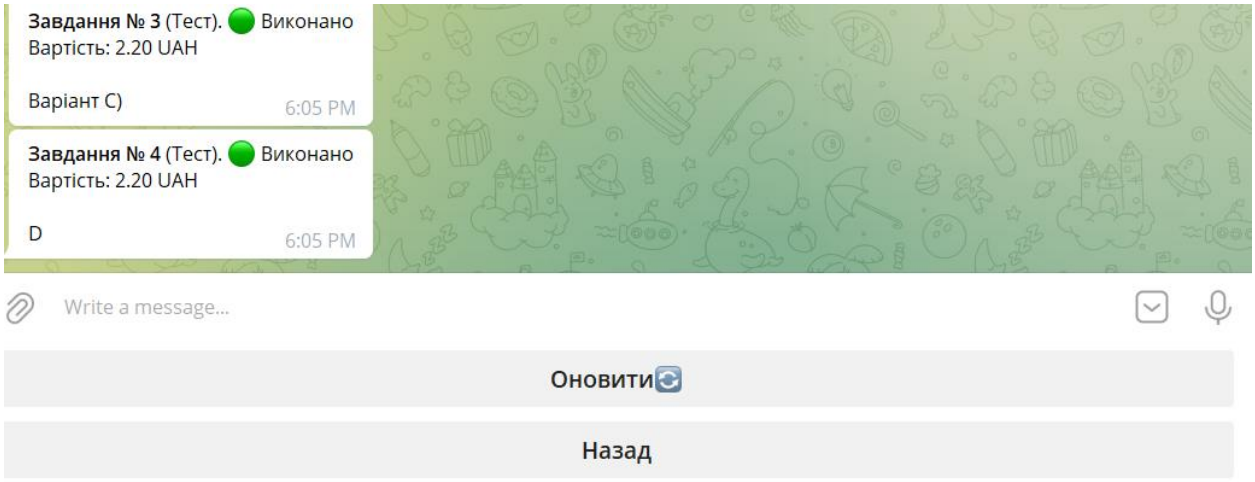

Рисунок 2.3.15 Розв’язок завдання

Можемо побачити, що наше замовлення вже вирішили та розв’язок до третього завдання варіант C, а для четвертого – D.

Взаємодія виконавця з платформою. Виконавці мають доступ як до режиму замовника, так і до режима виконавця. Ось так виглядає головна сторінка виконавця:

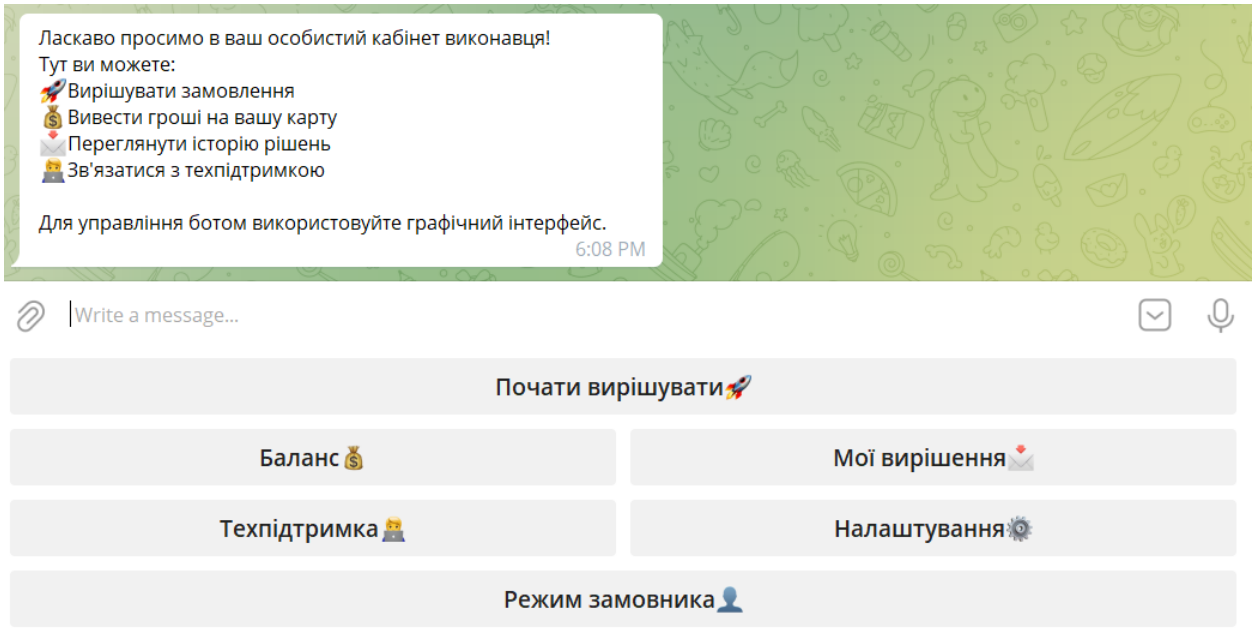

Рисунок 2.3.16 Головна сторінка виконавця

На цій сторінці ми можемо почати вирішувати замовлення, подивитися поточний баланс, свої вирішення, звернутися до техпідтримки, змінити налаштування та перейти до режиму замовника.

Для того, щоб почати вирішувати замовлення, натискаємо на “Почати вирішувати”.

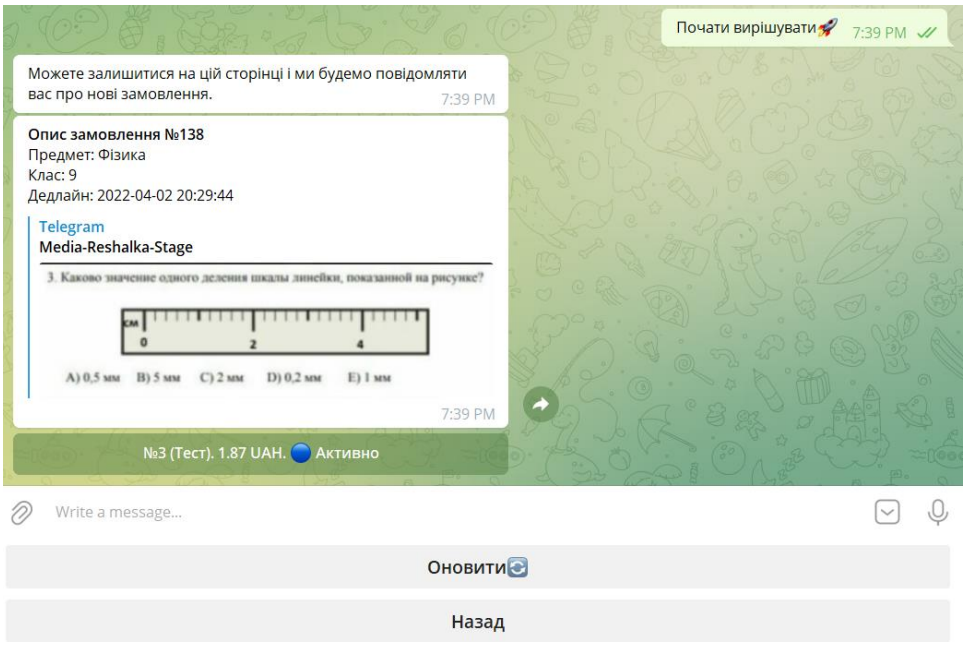

Рисунок 2.3.17 Список доступних замовлень

Для нас відображається список доступних замовлень. В даний момент доступне одне замовлення: вирішити тест. Також вказано скільки грошей ми замовимо якщо вирішимо тест. Для того, щоб почати виконувати завдання, натискаємо на кнопку з тестом.

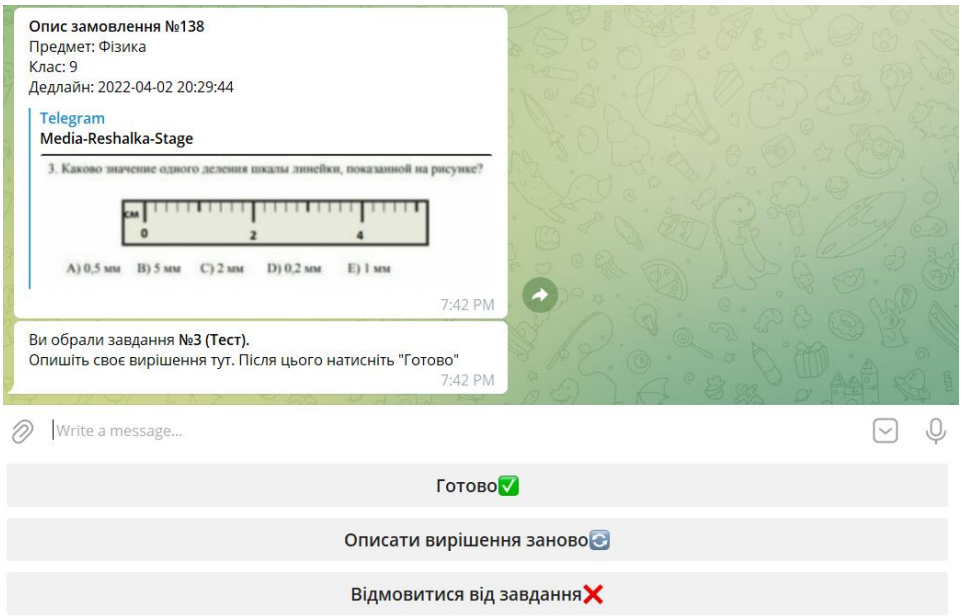

Рисунок 2.3.18 Вирішення завдання

Тепер ми описуємо своє вирішення і натискаємо “Готово”.

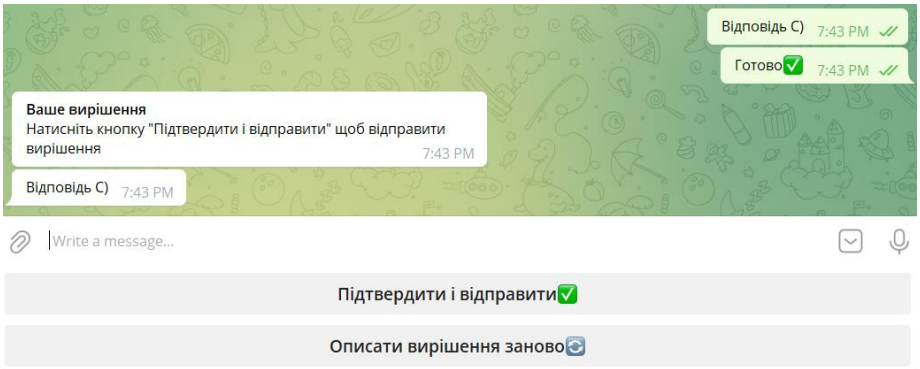

Рисунок 2.3.19 Перевірка вирішення

Далі перевіряємо правильність нашого вирішення і відправляємо його.

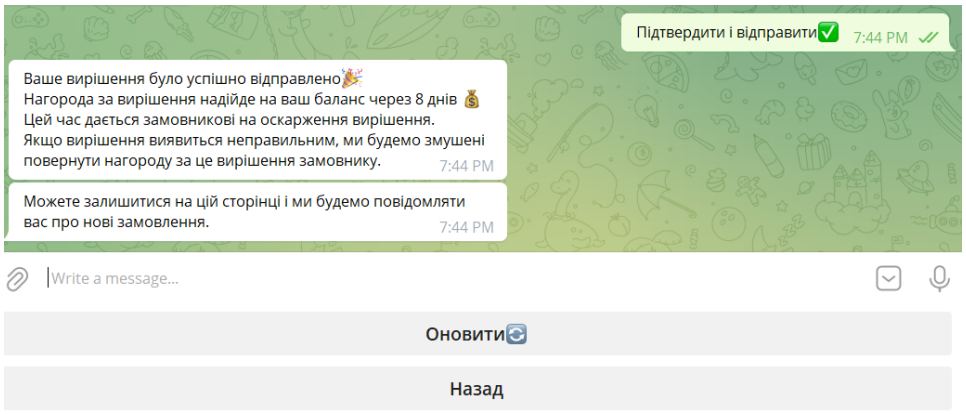

Рисунок 2.3.20 Замовлення вирішено

Замовлення було вирішено. Нагорода за нього надійде на баланс через 8 днів. В цей час замовник може оскаржити наше рішення і повернути кошти.

Тепер якщо в головному меню ми перейдемо до меню “Баланс”, ми зможемо подивитися, скільки коштів ми можемо вивести.

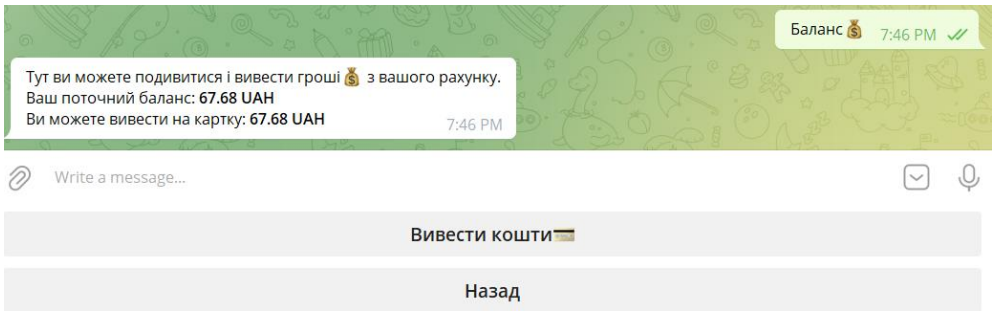

Рисунок 2.3.21 Меню баланса

2.4 Опис взаємодії адміністратора з платформою

На даний момент для адміністратора не передбачено графічного інтерфейсу. Всю активність адміністратор може проводити за допомогою нашого API.

Задля безпеки в нашому сервісі було створено повноцінну систему авторизації. Вона передбачає створення користувача, надання йому необхідних ролей та процес аутентифікації за допомогою JWT токена. Для адміністратора передбачено такий функціонал:

- Відмінити замовлення. Цей запит відміняє всі завдання в замовленні.
- Змінити інформацію про замовлення
- Отримати фінансову звітність. Запит повертає необхідну фінансову звітність за визначений період: скільки грошей було введено, загальна сума комісії, скільки грошей було виведено, тощо.
- Отримати звітність про користувачів. Запит повертає інформацію про користувачів: скільки користувачів приєдналося, скільки з них зробило хоча б одне замовлення, скільки користувачів має статус виконавця, тощо.
- Отримати звітність про замовлення. Запит повертає інформацію про замовлення: скільки замовлень було створено, скільки завдань було вирішено успішно, тощо.
- Створити притензію на вирішення. Він використовується, коли замовник невдоволений вирішенням. Цей запит заморожує кошти за виконання завдання поки притензія не буде розглянута. Таким чином, виконавець не зможе отримати кошти за вирішення.
- Відмінити притензію на вирішення. Він використовується, коли притензія була розглянута і помилок в вирішенні не було виявлено. Цей запит розморожує кошти за виконання завдання.

- Повернення коштів за завдання. Він використовується, коли притензія була розглянута і було виявлено помилки в вирішенні завдання. В цьому випадку кошти повертаються замовнику.
- Відміна завдання.
- Пошук завдань з фільтрацією
- Пошук користувачів з фільтрацією
- Зміна ролей користувача
- Збільшити баланс користувача
- Зменшити баланс користувача
- Відправити повідомлення від імені бота
- Забанити користувача
- Розбанити користувача

ВИСНОВОК

Отже, як результат виконання кваліфікаційної роботи, ми проаналізували ефективні способи побудови складного веб-додатку, проаналізували та використали інструменти для побудови його архітектури, та створили повноцінну платформу для вирішення шкільних завдань. Ця робота є досить унікальною та зможе бути корисною для школярів різних рівнів: як тих, хто хоче перевірити свій розв’язок та порівняти його з правильним, так і для тих, хто хоче заробити на своїх відмінних шкільних знаннях. Цей додаток підвищує мотивацію учнів вивчати шкільні предмети, адже вони розуміють, що за свої знання вони зможуть заробити свої перші гроші.

ДЖЕРЕЛА

1. Чистий код. Створення і рефакторинг за допомогою Agile. Роберт Сесіл Мартін. 448 с . ISBN 978-617-09-5285-1
2. Високонавантажених додатки. Програмування, масштабування, підтримка. Мартін Клеппман. 740 с. ISBN 978-5-4461-0512-0
3. Курс Udemy “Ultimate AWS Certified Developer Associate 2021 Masterclass
4. Документація LiqPay API [Електронний ресурс]. Режим доступу: https://www.liqpay.ua/documentation/api/home/
5. Документація Interkassa API [Електронний ресурс]. Режим доступу: https://www.interkassa.com/en/documentation-api/
6. Client–server model [Електронний ресурс]. Режим доступу: https://en.wikipedia.org/wiki/Client%E2%80%93server_model
7. Spring Framework [Електронний ресурс]. Режим доступу: https://spring.io/
8. AWS documentation [Електронний ресурс]. Режим доступу: https://docs.aws.amazon.com/
9. Побудова CI/CD процесу розроблення програмного забезпечення з використанням TeamCity та Go CD. Р. Дубленич, Є. Струк, 2017. Режим доступу: https://science.lpnu.ua/sites/default/files/journal-paper/2018/jul/13855/30.pdf
10. Model–view–controller [Електронний ресурс]. Режим доступу: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller
11. Monolith vs Microservices [Електронний ресурс]. Режим доступу: https://www.n-ix.com/microservices-vs-monolith-which-architecture-best-choice-your-business/
12. Telegram bot API [Електронний ресурс]. Режим доступу: https://core.telegram.org/bots/api

13. AWS [Електронний ресурс]. Режим доступу: https://aws.amazon.com/console/

14. Ngrok [Електронний ресурс]. Режим доступу: https://ngrok.com/

15. Continuous delivery [Електронний ресурс]. Режим доступу: https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment