КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра інтелектуальних програмних систем

Кваліфікаційна робота
на здобуття освітнього рівня бакалавра

за спеціальністю 121 Інженерія програмного забезпечення
на тему:

РОЗРОБКА ПЛАТФОРМИ ДЛЯ ПРОВЕДЕННЯ ДИСТАНЦІЙНИХ ОЛІМПІАД ТА ЗМАГАНЬ ЗІ СПОРТИВНОГО ПРОГРАМУВАННЯ

Виконав студент 4-го курсу
Кирило КОЦЬКО _____
(підпис)

Науковий керівник:
доцент, кандидат фіз.-мат. наук
Оксана ШКІЛЬНЯК _____
(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
« 29 » травня 2023 р.,
протокол № 11
Завідувач кафедри
Олександр ПРОВОТАР _____
(підпис)

Київ – 2023

РЕФЕРАТ

Обсяг роботи 51 сторінок, 29 ілюстрацій, 18 джерел посилань, 1 додаток.

ВЕБЗАСТОСУНОК, МІКРОСЕРВІСНА АРХІТЕКТУРА, КОМПІЛЯЦІЯ, МОВА ПРОГРАМУВАННЯ, SPRING BOOT, MVC, ANGULAR, DOCKER.

Об’єктом роботи є процес розробки на основі мікросервісної архітектури клієнт-серверного застосунку для проведення дистанційних змагань зі спортивного програмування.

Метою роботи є створення легкого у використанні вебзастосунку для створення, редагування та керування задач з програмування, а також організації відповідних публічних або приватних турнірів.

Предметом роботи є клієнт-серверний застосунок, серверна (back-end) та мікросервісна частина якого створена за допомогою мови програмування Java (Java EE), програмного каркасу (framework) Spring Framework, інструментарію для управління ізольованими Linux-контейнерами Docker, засобу відображення між об’єктами та реляційними структурами (ORM) Hibernate, об’єктно-реляційної системи керування базами даних (СУБД) PostgreSQL, а візуальна (front-end) частина якого створена за допомогою мови програмування TypeScript та front-end фреймворку Angular.

Під час виконання роботи були розглянуті різні підходи до розробки інформаційних систем, спрямованих на веборієнтовані середовища, зокрема розробка на основі мікросервісної архітектури. Було проведено загальний аналіз аналогічної системи та надано опис процесу розробки даної системи. Спроєктовано серверний, багатокористувацький додаток на базі фреймворку Spring, який був інтегрований з мікросервісом віддаленого компілятора інкапсульованого у Docker контейнері. Розроблений додаток під назвою Goslinger призначений для створення та проведення публічних або приватних турнірів зі спортивного програмування з обмеженою кількістю участників.

Система функціонує в режимі тестування в локальному середовищі на персональному комп’ютері.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

ACM ICPC – Association for Computing Machinery International Collegiate Programming Contest; Асоціація комп'ютерних машин Міжнародного студентського конкурсу програмування;

ORM – Object-Relational Mapping; Об'єктно-реляційне відображення;

Java EE – Java Enterprise Edition; Java для розробки корпоративних додатків;

СУБД – Системи керування базами даних;

UOI – Українська олімпіада з інформатики;

API – Application Programming Interface; Інтерфейс програмування додатків

BF – Brainf**k; мова програмування;

GUI – Grafical User Interface; графічний користувацький інтерфейс;

JDK – Java Development Kit; Набір засобів розробки Java;

JVM – Java Virtual Machine; віртуальна Java-машина;

DI – Dependency Injection; впровадження залежностей;

HTTP – Hypertext Transfer Protocol; Протокол передачі гіпертексту;

JWT – JSON Web Token; Маркер веб-токена у форматі JSON.

ВСТУП

З давніх часів людина відчувала потребу у навчанні та здобутті нових знань. Однак, ще більш важливим завданням для людини є ефективне використання набутих знань у цілях розвитку суспільства та здатності конкурувати за переваги та прогрес у цій суспільній системі. Досягненню другого аспекту сприяють різноманітні події, такі як конкурси, олімпіади, турніри та змагання. І хоча подібні заходи не лише спрямовані на визначення лідерів у сферах розумових здібностей, а й на виявлення спортивних та творчих талантів, найбільш адаптованими для комп’ютерної інтеграції є саме розумові змагання.

Комп’ютерна інтеграція у свою чергу є необхідною для автоматизації рутинних процесів, що значно полегшує роботу судів та членів журі олімпіад, турнірів та змагань. Це дозволяє ефективно зменшити часові, розумові та фізичні зусилля людей, сприяючи більш швидкому та точному проведенню змагань. Програмування, зокрема, призначене для автоматизації рутинних процесів, навіть якщо мова йде про процес змагань зі спортивного програмування.

Саме тому, розробка платформи для проведення дистанційних олімпіад та змагань зі спортивного програмування є **актуальною** задачею. Дана платформа дозволить учасникам змагань з усього світу брати участь у розумових змаганнях, використовуючи свої навички у програмуванні та розв’язуванні складних завдань.

Метою дослідження є розробка платформи для проведення дистанційних олімпіад та змагань зі спортивного програмування. Дослідження спрямоване на автоматизацію рутинних процесів, покращення точності та швидкості проведення змагань, забезпечення зручного інтерфейсу для учасників та організаторів, а також забезпечення безпеки та надійності системи.

Об’єктом дослідження є процес розробки клієнт-серверного застосунку на основі мікросервісної архітектури для проведення дистанційних змагань зі спортивного програмування.

Предметом дослідження даної роботи є розробка платформи, яка використовує клієнт-серверну архітектуру та мікросервіси для проведення

дистанційних олімпіад та змагань зі спортивного програмування. Серверна (back-end) частина платформи реалізована з використанням мови програмування Java (Java EE), фреймворка Spring Framework, Docker для управління ізольованими Linux-контейнерами, Hibernate для взаємодії з базою даних PostgreSQL. Візуальна (front-end) частина платформи реалізована з використанням мови програмування TypeScript та фреймворку Angular.

Дослідницькі завдання:

- вивчити сучасні тенденції та особливості проведення дистанційних змагань зі спортивного програмування;
- проаналізувати існуючі рішення та платформи для проведення дистанційних олімпіад та змагань з програмування;
- розробити архітектуру клієнт-серверного застосунку Goslinger з використанням мікросервісної архітектури;
- розробити та оптимізувати функціональні модулі застосунку, включаючи реєстрацію учасників, завантаження та перевірку коду, оцінку результатів та відображення рейтингів;
- дослідити та інтегрувати віддалений компілятор з відкритих джерел для перевірки коду учасників;
- розробити зручний та ергономічний інтерфейс для учасників та організаторів змагань.

Практична значимість роботи полягає в тому, що створена в її рамках платформа під назвою Goslinger може бути використана вчителями та викладачами середніх, або вищих навчальних закладів для проведення внутрішніх олімпіад/змагань з програмування серед учнів та студентів.

РОЗДІЛ 1 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ПЛАТФОРМ

1.1. Огляд наявних видів змагань з інформаційних технологій

Дослідження варто почати з огляду вже існуючих змагань з інформаційних технологій. Це може надати більше уявлень про те, як саме повинна виглядати система Goslinger, та на які аспекти слід зосередитися в першу чергу.

Для проведення огляду будуть розглянуті кілька відомих змагань у сфері інформаційних технологій:

- ACM ICPC (Association for Computing Machinery International Collegiate Programming Contest): це одне з найпопулярніших змагань зі спортивного програмування у світі;
- Ukrainian Olympiad in Informatics (Українська олімпіада з інформатики): Це національне змагання з програмування для учнів та студентів;
- All-Ukrainian Student Olympiad in Programming (Всеукраїнська студентська олімпіада з програмування): Це змагання з програмування, яке проводиться для студентів усіх університетів України;
- Hackathon (Хакатон) інтенсивне змагання або подія, де учасники протягом обмеженого часу працюють над інноваційними проєктами або розв’язаннями за допомогою програмування, дизайну та технологій.

ICPC (International Collegiate Programming Contest) є щорічним змаганням з програмування між університетами з усього світу (рис. 1.1). Під керівництвом професора Уільяма Б. Пучера з Університету Бейлор, ICPC організовує регіональні змагання на шести континентах, які завершуються щорічним світовим фіналом.

Рисунок 1.1 – Логотип конкурсу ICPC

ICPC змагання представляють собою командні змагання. За поточними правилами, кожна команда повинна складатися з трьох студентів. Учасниками можуть бути студенти університетів, які на момент змагання навчаються не більше п’яти років. Студенти, які вже брали участь в двох світових фіналах або п’яти регіональних змаганнях, не мають права повторно брати участь у змаганні [1].

Під час кожного змагання командам, що складаються з трьох учасників, надається 5 годин, щоб вирішити від восьми до п’ятнадцяти задач програмування. Зазвичай на регіональних етапах це восьми задачі, а на фінальних - дванадцять. Учасники повинні надіслати свої рішення у вигляді програм, написаних на мовах програмування C, C++, Java, Ada, Python або Kotlin. Потім програми запускаються на тестових даних. Якщо програма не дає правильної відповіді, команда отримує повідомлення про це і може надіслати іншу програму на перевірку.

Перемагає команда, яка набирає найбільшу кількість балів після розв’язання задач у встановлений термін. Команда, яка досягає найвищого результату в змаганні, вважається переможцем ICPC. Оцінки надаються на основі кількості задач, які команда успішно вирішила, а також часу, витраченого на їх розв’язання. Час не враховується за невирішену задачу. Крім цього, зазвичай враховується і правильність розв’язків для задач, які були вирішені командою [2].

У 2018 році в змаганні брали участь 52 709 студентів з 3 233 університетів з

110 країн. ICPC діє під егідою фонду ICPC і згідно з угодами з приймаючими університетами та неприбутковими організаціями відповідно до політик та процедур ICPC [3]. За період з 1977 року до 2017 року, ICPC було проведено під патронатом ACM і відоме як ACM-ICPC [4].

Українська олімпіада з інформатики (UOI) працює за схожим принципом. Олімпіада має на меті підтримку та розвиток обдарованої молоді в галузі інформатики. Участь українських олімпіад з інформатики відкрита для школярів та студентів різних рівнів освіти. Змагання зазвичай включають в себе декілька етапів, починаючи з місцевих відбіркових туров і закінчуючи національним фіналом. Учасники розв’язують завдання з різних областей інформатики, таких як алгоритми, структури даних, програмування, графи та інші. Кожен новий етап олімпіади зазвичай називають контестом.

Так само, як і в ICPC - українська олімпіада з інформатики надає можливість розв”язку задачі на різних мовах програмування. Після завершення контесту, організатори зазвичай викладають детальний розбір розв’язків задач та самий варіант можливого розв’язку на мові програмування C++ - флагманської мови цього конкурсу. Олімпіада проходить за допомогою платформи Eolymp [5], зокрема про яку ми поговоримо трохи згодом.

Перейдемо до Хакатону (рис. 1.2). Хакатон - це захід, де люди взаємодіють у швидкій та спільній розробці протягом відносно короткого періоду часу, наприклад, одну чи дві доби. Вони часто використовують гнучкі практики розробки програмного забезпечення, такі як спринтовий дизайн, де програмісти та інші учасники, такі як графічні дизайнери, дизайнери інтерфейсів, менеджери продукту, проєктні менеджери, експерти в галузі та інші, інтенсивно співпрацюють над інженерними проєктами, такими як розробка програмного забезпечення. Слово hackathon походить від “hack” і “marathon”, де слово hack репрезентує саме дослідницький підхід до програмування, а не його конвенційне значення.

Рисунок 1.2 – Один з живих Hackathon подій

Головною метою хакатону є створення працюючого програмного або апаратного забезпечення до завершення події. Хакатони зазвичай мають конкретну спрямованість, яка може включати обмеження на використання певної мови програмування, операційної системи, додатка, API або стосуватися конкретної тематики та цільової аудиторії програмістів.

Існують різні типи хакатонів:

- З використання певної мови програмування, API або фреймворку: присвячені розробці додатків, які використовують певну мову або фреймворк, такі як Ruby on Rails, JavaScript та Node.js. Такі хакатони зосереджуються на створенні додатків, які використовують API (інтерфейс програмування додатків) певної компанії або джерела даних. Open Hack, публічна подія, що проводиться Yahoo! з 2006 року, фокусується на використанні API Yahoo!, а також API вебсайтів, що належать Yahoo!, наприклад, Flickr. На Open Hack India 2012 компанії Yahoo! взяли участь понад 700 учасників [6].
- Для певного типу додатка: деякі хакатони спеціалізуються на конкретній платформі, такій як мобільні додатки, операційні системи для персональних комп’ютерів, веброзробка або розробка відеоігор. Наприклад, мобільні хакатони, як от Over the Air, що проходять в Фенікс Парку, Ірландія, зазвичай отримують значну корпоративну підтримку та викликають великий

інтерес [7].

Оглянувши та проаналізувавши дані змагання, можна зробити висновок, що найбільш підходящими для нашої платформи є підхід ICPC. Саме аспекти підтримки багатьох мов, обмеженість по часу виконання та підхід розподілення задач по тематикам і буде використано при проєктувані платформи Goslinger.

1.2. Огляд та аналіз платформи Eolymp

Eolymp - це онлайн-платформа для підготовки і участі у змаганнях з програмування та вирішення завдань. Вона надає широкий вибір завдань різної складності, включаючи алгоритми, структури даних, графи, математику та багато іншого [5]. Емблема платформи продемонстрована на рис. 1.3.

Рисунок 1.3 – Емблема сайту Eolymp

На платформі Eolymp ви можете знайти завдання для тренування і розвитку своїх навичок програмування, а також брати участь у онлайн-змаганнях. Ви можете вибрати завдання, написати код для його вирішення та надіслати своє рішення на перевірку. Після перевірки платформа повідомить вам про правильність або неправильність вашого рішення, а якщо воно неправильне, надасть інформацію про помилку.

Платформа надає можливість створення власних змагань і запрошення інших учасників. Це може бути корисно для організації змагань у навчальних закладах або для перевірки навичок програмування в командній роботі.

Платформа Eolymp доступна на декількох мовах програмування, включаючи C++, Java, Python, Pascal та інші. Вона пропонує широкий спектр завдань, починаючи від базових і простих до більш складних і високорівневих.

Сайт надає базу завдань, навчальні матеріали та автоматичну систему перевірки рішень. Користувачам сайту пропонується вирішити завдання написавши невелику програму-рішення на одній з мов програмування і відправити її в систему для перевірки.

Головна сторінка платформи продемонстрована на рис. 1.4. На головній сторінці розміщенно перші 5 найпопулярніших змагань та останні новини пов’язані з сайтом Eolymp.

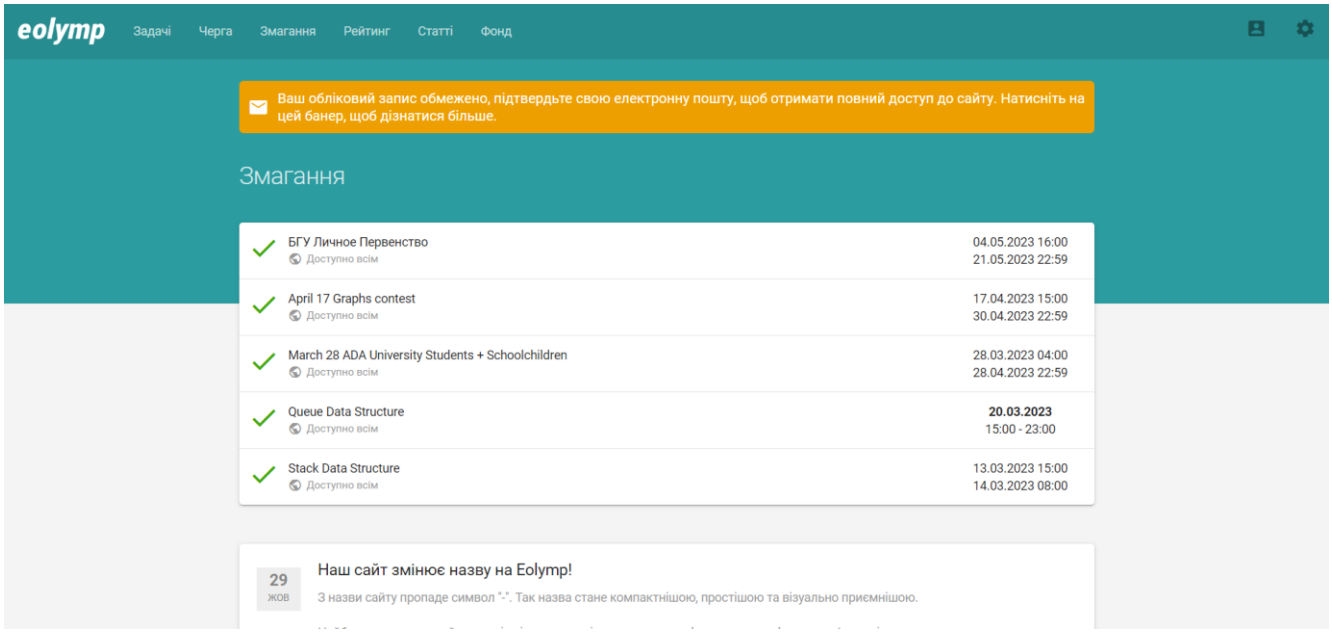

Рисунок 1.4 – Головна сторінка сайту Eolymp

Сайт має шість основних розділів:

- Задачі – репрезентує список усіх доступних для розв’язання задач
- Черга – список поточних відправок на перевірку завдань, з відповідними статусами перевірки
- Змагання – перелік усіх доступних змагань/турнірів
- Рейтинг – репрезентує рейтинг користувачів за кількістю розв”язаних

задач

- Статті – невеличкий портал, який здебільшого призначений для саморозвитку та навчання
- Фонд – окремий благодійний портал для підтримки проєкту [8]

Платформа працює у системі з авторизованими користувачами. Користувач може побачити свій ранг(рейтинг) у системі, кількість відправок та розв’язаних задач. Крім того можна побачити історію відправок, зі статусами, оцінками та вхідними кодами відправок. Доступна функція редагування профілю. Приклад такого профілю можна побачити на рис. 1.5.

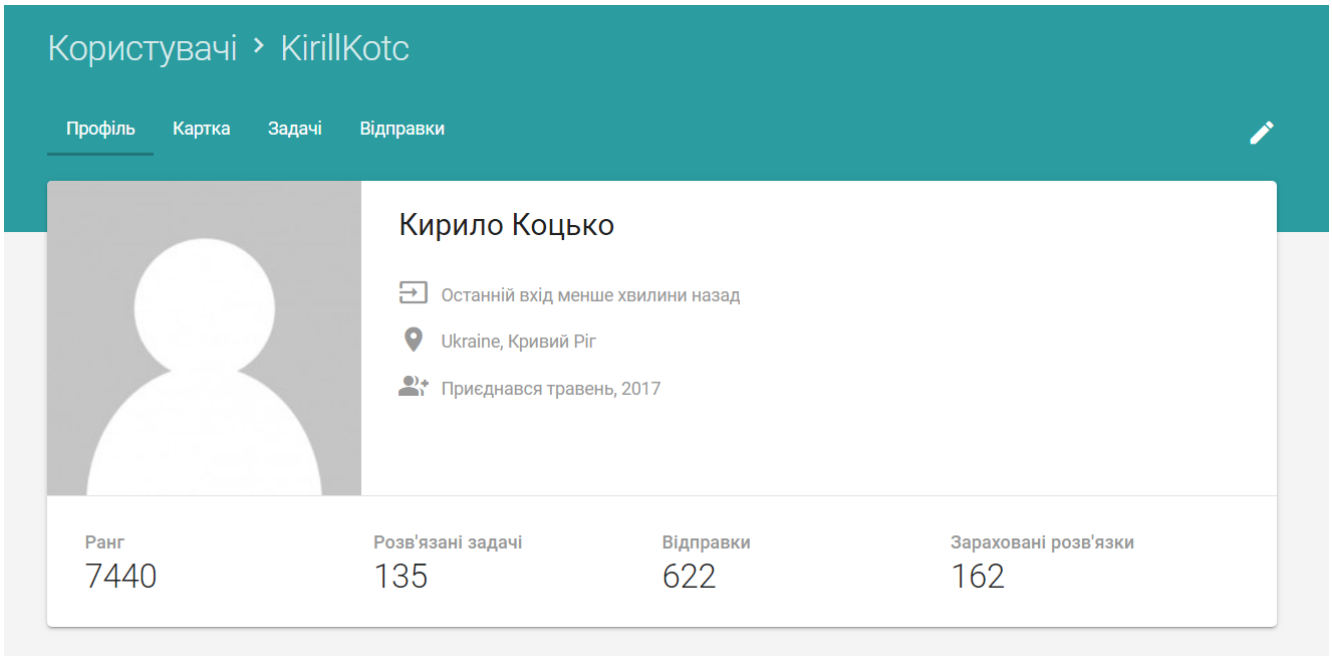

Рисунок 1.5 – Вигляд сторінки профілю користувача

Сайт дає змогу налаштувати мову інтерфейсу. Серед можливих мов інтерфейсу:

- українська
- англійська
- азербайджанська
- російська

Крім цього можна обрати улюблену мову програмування, часовий пояс для виконання завдань та цікавий параметр під назвою “Сполучення клавіш”. Важко сказати, за що саме відповідає цей параметр, адже документація до цієї опції

відсутня. Скоріш за все дана опція відповідає за hotkeys, які працюють на цьому сайті. Але які саме hotkey’і активні на цій платформі, також невідомо.

Перейдемо до задач. Приклад сторінки з умовою задачі можна побачити на рис. 1.6. Умова задачі складається з таких елементів:

- Заголовок задачі: Заголовок чітко вказує на тему або мету задачі.
- Умова задачі: У цьому розділі подається текстовий опис самої задачі. Він може містити вихідні дані, вхідні обмеження, приклади вхідних та вихідних даних, пояснення до завдання, необхідні вимоги та умови, яким повинно задовольняти рішення.
- Вхідні дані: Цей розділ описує формат та тип вхідних даних, які будуть надані для задачі. Наприклад, це можуть бути числа, рядки, масиви або інші типи даних.
- Вихідні дані: Тут вказується формат та тип очікуваних вихідних даних, які повинне повернути рішення задачі.
- Пояснення до завдання: Цей розділ може містити додаткові пояснення, правила чи інструкції, які допоможуть у більш розумінні задачі та її вимог.
- Приклади: В розділі прикладів наводяться конкретні приклади вхідних і вихідних даних для демонстрації того, як повинно працювати рішення задачі. Це допомагає краще зрозуміти очікувані результати та правильність розв’язку.
- Обмеження: У цьому розділі можуть бути наведені обмеження на вхідні дані, наприклад, максимальна довжина рядка, максимальне значення числа, обмеження на час виконання або використання пам’яті.

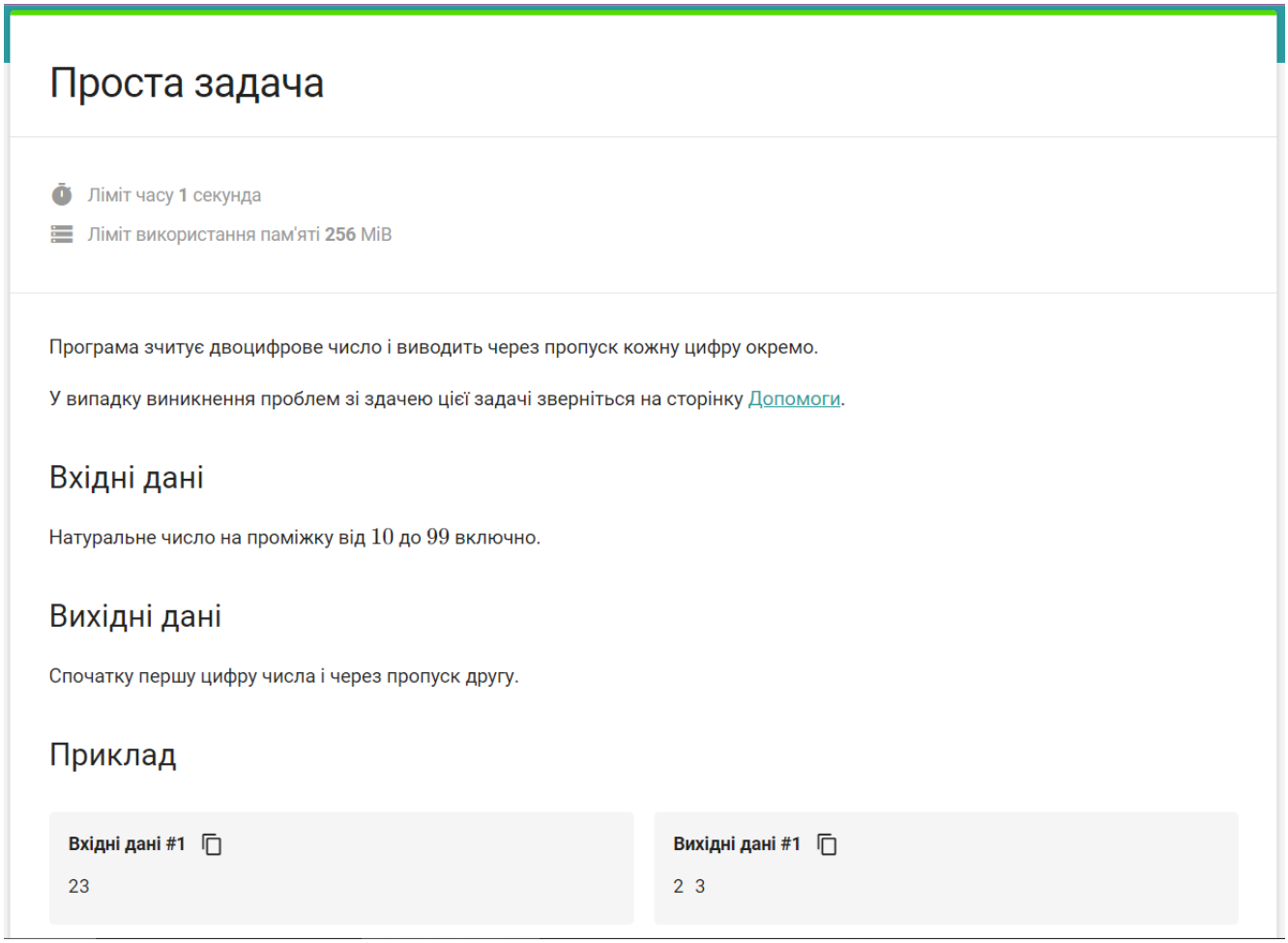

Рисунок 1.6 – Приклад сторінки “Умова задачі”

Для відправки розв’язків доступні такі мови програмування: BF, C, C#, C++, D, Dart, Go, Haskell, Java, JavaScript, Kotlin, Lua, Pascal, Perl, PHP, Python, Ruby, Rust, Swift. Крім того для деяких мов доступна можливість відправки на різних версіях компілятора.

На сторінці "Submissions" можна переглянути історію рішень, які ви надсилали для вирішення завдань (рис. 1.7). Ця сторінка надає інформацію про кожне надіслане вами рішення, його стан та результати перевірки. Основні елементи сторінки "Submissions" включають наступне:

- Список надісланих рішень: На цій сторінці ви зазвичай побачите список усіх рішень, які ви надіслали раніше. Кожен рядок представляє окреме надіслане рішення і містить інформацію, таку як час надсилання, статус та результат перевірки;
- Статус рішення: Для кожного надісланого рішення ви зазвичай

побачите його поточний статус. Це може бути "В черзі" (In Queue), "Оцінюється" (Evaluating), "Прийнято" (Accepted) або "Відхилено" (Rejected), в залежності від стану перевірки;

- Результати перевірки: сторінка "Submissions" зазвичай надає додаткову інформацію про результати перевірки кожного надісланого рішення. Це може включати інформацію про час виконання, використану пам’ять, кількість набраних балів або повідомлення про помилку, якщо рішення містить помилки або не відповідає вимогам задачі;
- Додаткові дії: Залежно від функціональності платформи Eolymp, на сторінці "Submissions" можуть бути доступні додаткові дії. Наприклад, ви можете переглянути деталі рішення, перевірити рішення знову або переглянути коментарі від перевіряючого;
- Код програми: Перегляд надісланого рішення. Бонусом являється “highlighting” спеціальних слів у коді, що значно полегшує читання та розуміння написаного коду.

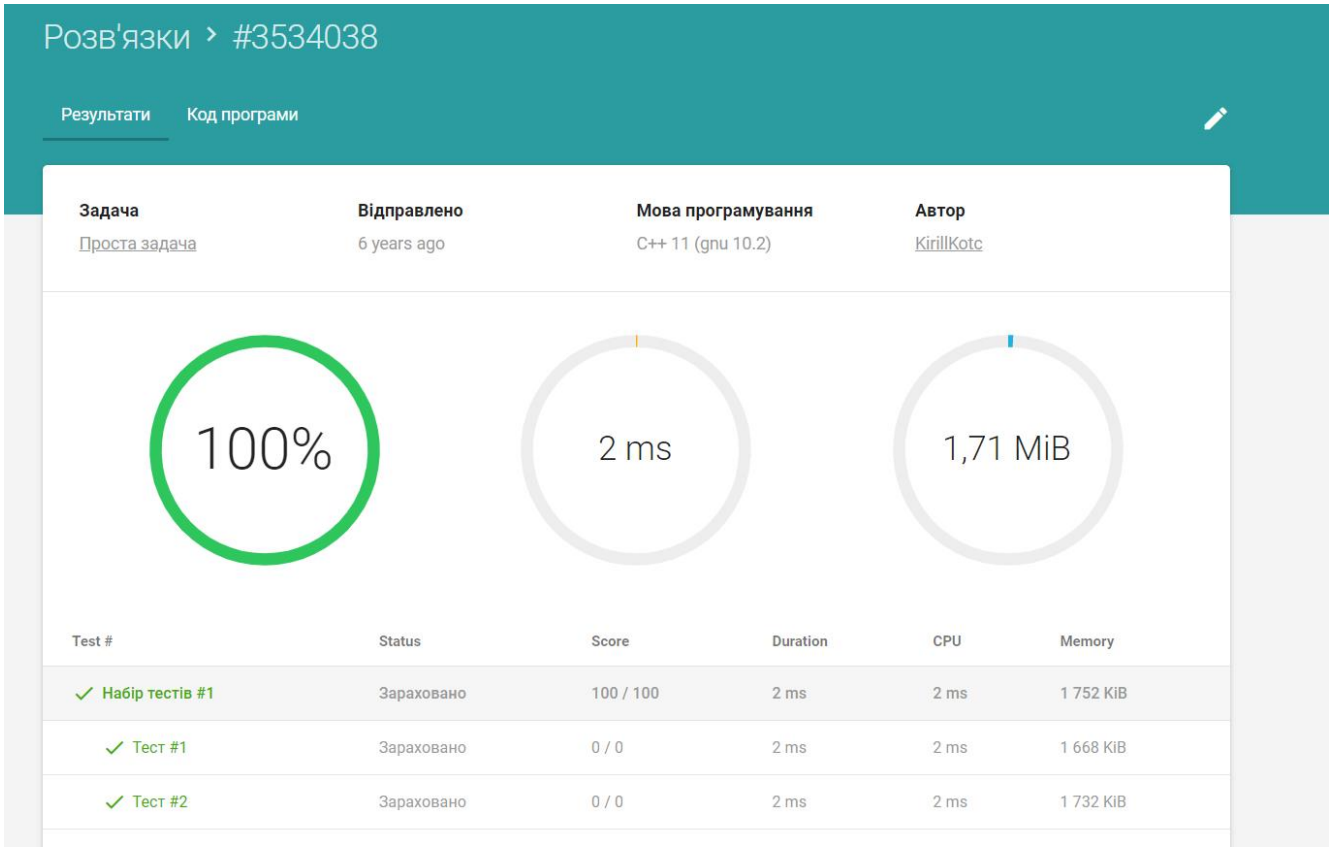

Рисунок 1.7 – Приклад сторінки "Submissions"

Дослідивши наявну систему Eolymp, та взявши до уваги основні елементи сайту можна переходити до наступного етапу огляду методів проєктування веборієнтованих систем та дослідження основних технологій, необхідних для створення платформи Goslinger.

РОЗДІЛ 2 АРХІТЕКТУРА ТА ТЕХНОЛОГІЇ РОЗРОБКИ

2.1. Клієнт-серверна архітектура на основі мікросервісів

Клієнт-серверна архітектура, яка схематично зображена на рис. 2.1, складається з двох компонентів: клієнта і сервера.

Клієнт - це комп’ютер, розташований на стороні користувача, який відправляє запити до сервера з метою отримання інформації, або виконання певних дій.

Сервер виступає як потужний комп’ютер або спеціальне обладнання, яке виконує певні завдання, включаючи виконання програмного коду, надання сервісних функцій за запитом клієнтів, забезпечення доступу користувачів до ресурсів, зберігання інформації і баз даних [9].

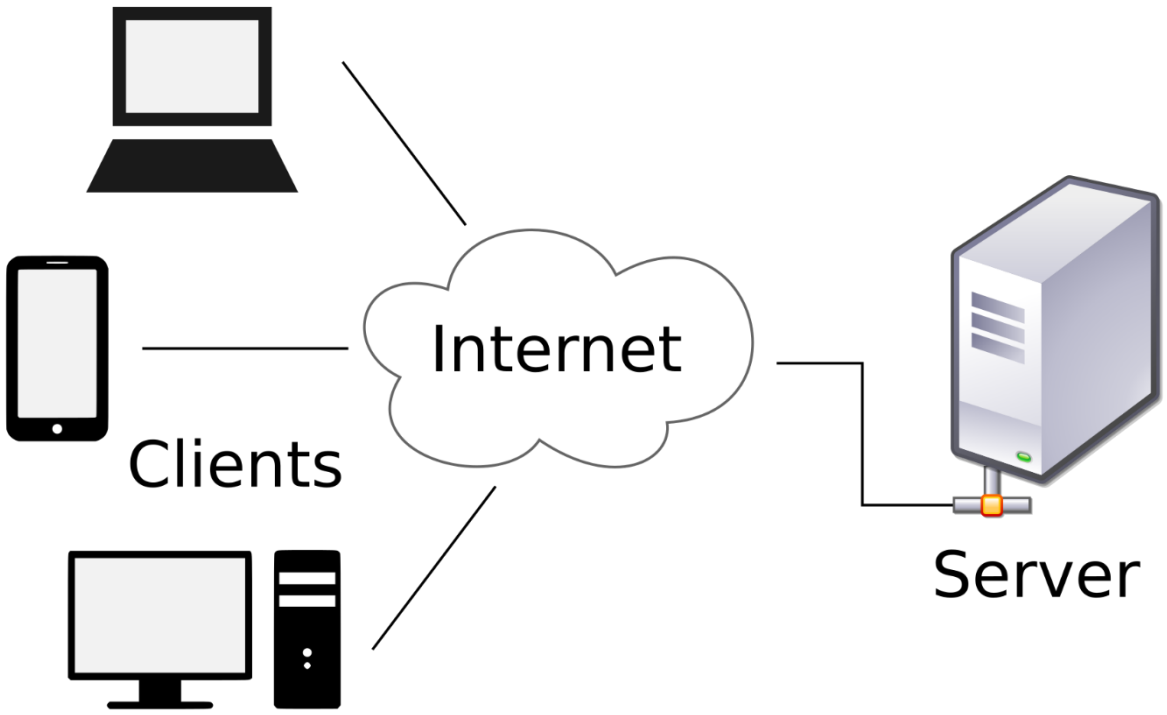

Рисунок 2.1 – Діаграма комп’ютерної мережі, на якій клієнти спілкуються з сервером через Інтернет

Взаємовідношення співпрацюючих програм у додатку описується через характеристику "клієнт-сервер". Серверний компонент забезпечує функцію або сервіс одному або кільком клієнтам, які ініціюють запити для отримання таких

послуг. Сервери класифікуються відповідно до виду послуг, які вони надають. Наприклад, вебсервер обслуговує вебсторінки, а файловий сервер надає доступ до комп’ютерних файлів. Спільні ресурси можуть включати будь-яке програмне забезпечення та електронні компоненти серверного комп’ютера - від програм і даних до процесорів та пристроїв зберігання. Спільне використання ресурсів сервера становить його сервіс.

Визначення того, чи комп’ютер є клієнтом, сервером або обома одночасно, залежить від характеру додатка, який потребує сервісних функцій. Наприклад, один комп”ютер може одночасно виконувати функції вебсервера та файлового сервера, надаючи різні дані клієнтам, які звертаються з різними типами запитів. Клієнтське програмне забезпечення також може спілкуватися з серверним програмним забезпеченням на тому ж комп’ютері. Комунікація між серверами, наприклад, для синхронізації даних, іноді називається міжсерверною або сервер-серверною комунікацією.

Клієнти та сервери взаємодіють, використовуючи шаблон запит-відповідь. Клієнт ініціює запит, а сервер повертає відповідь. Цей обмін повідомленнями є прикладом комунікації між процесами. Для успішної взаємодії комп’ютери повинні мати загальну мову та дотримуватися правил, щоб клієнт та сервер знав, що очікувати. Мова та правила комунікації визначаються в протоколі зв’язку, який працює на рівні додатку. Протокол рівня додатку встановлює основні шаблони діалогу. Для ще більшої формалізації обміну даними сервер може мати реалізований інтерфейс програмування додатків (API) [10]. API є абстракційним рівнем для доступу до сервісу. Шляхом обмеження комунікації до конкретного формату вмісту, API полегшує обробку даних. Шляхом абстрагування доступу, API сприяє обміну даними між різними платформами.

Частіше всього, клієнт-серверна архітектура представляється у форматі frontend та backend.

Фронтенд і бекенд - це два основних терміни, що використовуються у веброзробці. Фронтенд - це те, що відображається перед користувачем і з чим він взаємодіє, тоді як бекенд - це те, що відбувається позаду сцени. Обидва компоненти

повинні ефективно співпрацювати, щоб забезпечити оптимальну функціональність вебсайту.

Фронтенд охоплює ту частину вебсайту, яку користувачі можуть бачити та з якою вони можуть взаємодіяти. Це включає графічний інтерфейс користувача, командний рядок, дизайн, навігаційні меню, текст, зображення, відео і т.д. З іншого боку, бекенд є невидимою частиною вебсайту, з якою користувачі не мають прямого контакту.

Фронтенд відповідає за візуальні аспекти вебсайту, які сприймають користувачі. Зі свого боку, бекенд забезпечує роботу всіх процесів, які відбуваються в тлі.

Бекенд виконується на стороні сервера. Вона відповідає за збереження та організацію даних, а також за забезпечення належної роботи всього, що стосується клієнтської сторони вебсайту. Бекенд - це скрита від користувачів частина програмного забезпечення, яка не взаємодіє безпосередньо з ними. Користувачі отримують доступ до розроблених бекенд-розробниками функцій та характеристик через фронтенд-додаток. Бекенд також включає такі дії, як створення API, розробку бібліотек та роботу з компонентами системи, які не мають інтерфейсу користувача, або навіть системи наукового програмування.

Типовими мовами та фреймворками frontend’у вважаються: JavaScript, TypeScript, React.js, AngularJS, JQuerry, Flutter. Типовими мовами та фреймворками backend”у вважаються: Java, PHP, C++, Python, C#, Node.js, Django, Spring, Ruby on Rails, Laravel [11]. Деякі з них ми розглянемо у наступному розділі. Схематичний розподіл цих технологій можна побачити на рис. 2.2.

Рисунок 2.2 – Frontend'ові та backend'ові технології програмування

Повернемось до самої клієнт-серверної архітектури. Існують дворівневі та багаторівневі клієнт-серверні архітектури.

Дворівнева архітектура складається з двох вузлів:

- клієнт, що представляє користувацький інтерфейс;
- сервер, який відповідає за отримання запитів і відправку відповідей клієнту, використовуючи при цьому лише власні ресурси.

Робочий принцип полягає в тому, що сервер приймає запит, обробляє його і надає відповідь безпосередньо, без залучення зовнішніх ресурсів [9]. Дворівнева архітектура схематично зображена на рис. 2.3.

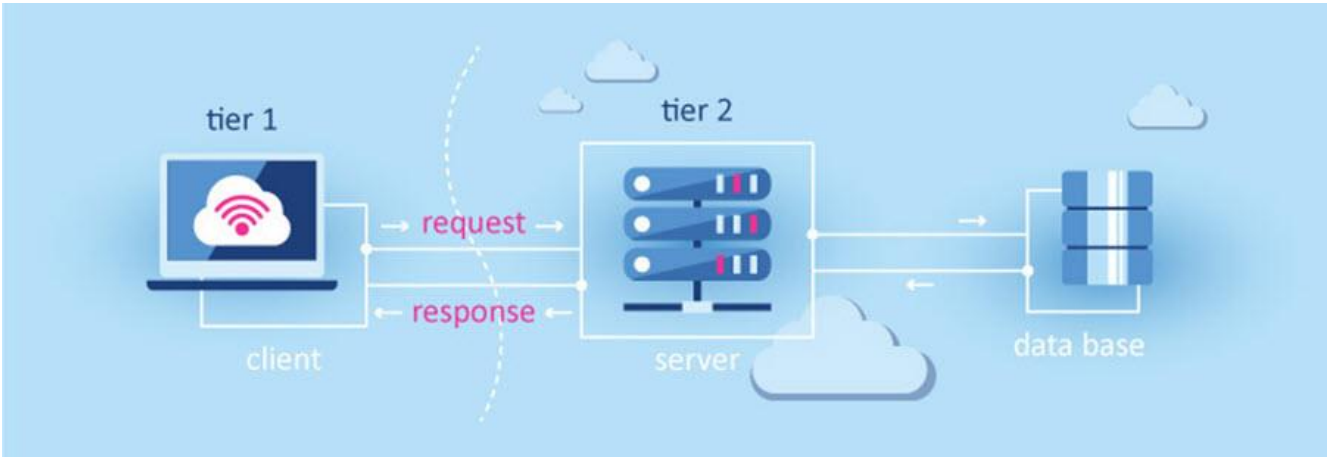

Рисунок 2.3 – Дворівнева архітектура

Багаторівнева архітектура (рис. 2.4) складається з наступних компонентів:

- представлення даних – призначений для користувача інтерфейс;
- прикладний компонент – сервер додатків;
- N-tier сервери – мікросервіси, які слугують для додаткової обробки даних.

Принцип роботи полягає в тому, що кілька серверів спільно обробляють запити від клієнтів, що дозволяє розподілити навантаження і зменшити навантаження на окремі сервери. Мультирівнева архітектура сприяє покращенню продуктивності інформаційної системи, а також оптимізує розподіл її програмних та апаратних ресурсів [9].

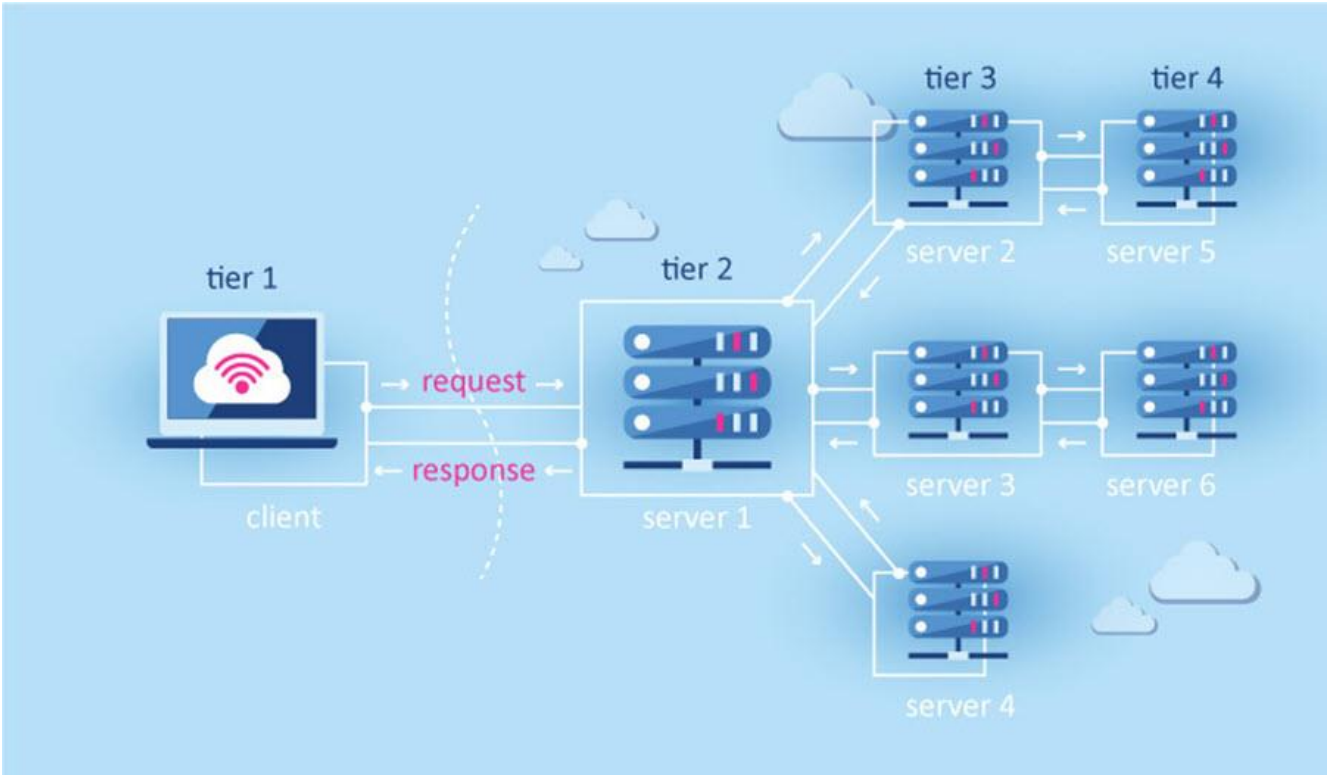

Рисунок 2.4 – Багаторівнева архітектура

Мікросервіс – невеликий незалежний сервіс або функція, яка приймає запити і виконує певні дії. Це певна частина функціоналу, яка реалізована як окремий сервіс і доступна через мережу за допомогою API. Він може бути доступний як постійно активний бекенд-сервіс або викликатись при виникненні певної події. Мікросервіси, відмінно від монолітних додатків, є незалежними сервісами, які спілкуються між собою через комп’ютерну мережу. Вони розгортаються на різних

серверах.

Мікросервіси зазвичай організовані за допомогою різних ярусів або компонентів. Немає жорстких правил про кількість або типи ярусів, але існують рекомендації щодо найкращих практик. На рис. 2.5 можна побачити найпоширеніші яруси, які використовуються в подібних архітектурах. Назви ярусів можуть варіюватися, але загальна структура схожа на класичну багатоярусну архітектуру [12].

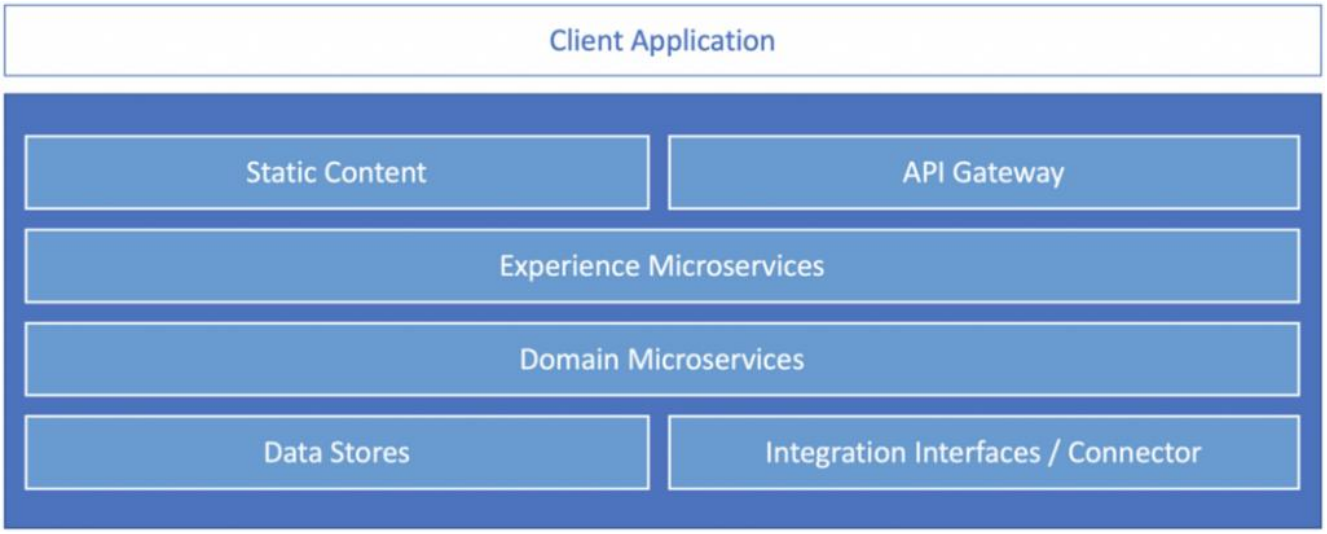

Рисунок 2.5 – Архітектура мікросервісного застосунку

У розробці використовуватися різні логічні яруси залежно від потреб проєкту. Однак, основні логічні яруси, які часто зустрічаються, включають:

- Презентаційний ярус (Presentation Layer): Відповідає за представлення даних користувачу. Цей ярус включає компоненти, що стосуються графічного інтерфейсу користувача (GUI), відображення даних та взаємодії з користувачем.
- Логічний або додатковий ярус (Logic or Service Layer): Відповідає за бізнес-логіку додатку. Цей ярус містить компоненти, які обробляють логіку, правила та функціональність додатку.
- Шлюз API (API Gateway): центральний вхід для всіх публічних API. Він виступає як єдина точка доступу до "Experience APIs", які можуть бути використані клієнтськими програмами, і він є важливою складовою управління API (API Management) за найкращими практиками. В основному, його роль полягає у передачі запитів до справжніх бекенд-сервісів та прийнятті рішень. API Gateway

виконує такі функції, як: Request handling, Upstream (або downstream), data transfer (або transformation), Access control, Security policies, Throttling, Rate limiting, Analytics, Caching [12].

- Джерело даних або персистентний ярус (Data Source or Persistence Layer): Відповідає за збереження та доступ до даних. Цей ярус включає компоненти, які взаємодіють з базами даних або іншими джерелами даних.
- та інші…

2.2. Огляд технологій розробки: Spring Framework, Docker, Angular

Розглянемо 3 основних концепти на основі яких буде побудований застосунок клієнт-серверний Goslinger, а саме:

- Spring Framework;
- Angular;
- Docker.

Розпочнемо основного backend фреймворку для побудови основної серверної та мікросервісної частини – Spring Framework.

Spring спрощує розробку корпоративних додатків на мові Java, надаючи повний набір інструментів для їх створення. Він підтримує використання мови Java у підприємницькому середовищі і також дозволяє використовувати Groovy та Kotlin як альтернативні мови програмування на віртуальній машині Java (JVM) [18]. Крім того, Spring дозволяє створювати різні архітектурні рішення залежно від потреб конкретного додатку.

Емблема Spring зображена на рис. 2.6. Починаючи з версії Spring Framework 6.0, для роботи з Spring потрібно мати Java версії 17 або вище [13].

Рисунок 2.6 – Емблема Spring Framework

Spring є проєктом з відкритим вихідним кодом і має активну спільноту розробників. Ця спільнота постійно надає зворотний зв’язок із реальними випадками використання, що допомагає Spring успішно розвиватися на протязі тривалого періоду.

Spring забезпечує підтримку широкого спектру сценаріїв використання. Великі підприємства часто мають додатки, які існують тривалий час і повинні працювати на конкретній версії JDK (Java Development Kit) та сервера додатків, які оновлюються відповідно до внутрішнього циклу оновлень. Інші додатки можуть бути розповсюджені як один JAR-файл з вбудованим сервером, наприклад, у хмарному середовищі. Деякі додатки можуть бути самостійними, без необхідності використання сервера, наприклад, пакетні або інтеграційні роботи.

Основні принципи Spring Framework включають:

- Сильна сумісність з попередніми версіями: Еволюцію Spring управляють таким чином, щоб мінімізувати кількість розбиваючих змін між версіями. Spring підтримує певний набір версій JDK та сторонніх бібліотек для полегшення підтримки додатків та бібліотек, що залежать від Spring.
- Увага до дизайну API: Команда Spring вкладає багато часу та зусиль у створення інтуїтивно зрозумілих API, які зберігають свою актуальність протягом багатьох версій та років.
- Врахування різноманітних підходів: Spring підтримує гнучкість і не нав”язує жорстких правил щодо того, як слід робити речі. Він підтримує широкий спектр потреб додатків з різними підходами.

- Надання вибору на кожному рівні: Spring дозволяє вам відкладати прийняття проєктних рішень до пізнішого часу. Наприклад, ви можете змінювати постачальників персистентності через конфігурацію, не змінюючи код. Те саме стосується інших інфраструктурних питань та інтеграції зі сторонніми API.
- Встановлення високих стандартів якості коду: Spring Framework надає велику увагу значущим, актуальним і точним javadoc. Він є одним з небагатьох проєктів, які мають чисту структуру коду без циклічних залежностей між пакетами.

Якщо ви починаєте роботу з Spring, ви можете використати Spring Boot для створення свого першого додатка. Spring Boot надає швидкий спосіб створення готового до використання додатка на базі Spring. Він ґрунтується на Spring Framework, використовує анотації замість складної конфігурації і призначений для швидкого запуску вашого додатка.

Перейдемо до основного фреймворку frontend частини, а саме до Angular. Angular працює у связці з мовою програмування TypeScript: обгорткою над мовою програмування JavaScript. Основна відмінність цих мов програмування полягає у строгій типізації TypeScript на відміну від JS.

Angular розробляється та підтримується командою Google. Емблему Angular можна побачити на рис. 2.7.

Рисунок 2.7 – Емблема Angular Framework

Щоб розуміти можливості фреймворка Angular, потрібно ознайомитись з такими елементами:

- Компоненти (Components): Компоненти в Angular представляють розділи користувацького інтерфейсу та включають логіку та представлення цих розділів. Вони використовуються для організації ієрархії додатків та забезпечення повторного використання коду;
- Шаблони (Templates): Шаблони використовуються для відображення інформації та взаємодії з користувачем. Вони містять HTML-код з додатковими синтаксичними конструкціями Angular, які дозволяють зв’язувати дані та керувати відображенням;
- Директиви (Directives): Директиви в Angular розширюють можливості HTML і дозволяють додавати нові функціональні можливості до елементів DOM. Вони можуть бути використані для маніпулювання відображенням, додавання анімації, роботи з подіями та інших операцій;
- Залежність ін’єкції (Dependency Injection): Механізм залежностей Angular дозволяє ефективно керувати залежностями між компонентами та іншими сервісами. Він дозволяє легко внедрювати залежності та забезпечувати ефективну роботу додатка.

Компоненти є основним будівельним блоком для додатків на Angular [14]. Кожен компонент складається з таких елементів:

- HTML-шаблону, який визначає, що відображатиметься на сторінці;
- TypeScript-класу, який визначає поведінку компонента;
- CSS-селектора, який визначає, як компонент використовується у шаблоні;
- Опціонально, CSS-стилі, які застосовуються до шаблону.

Такі елементи дозволяють створювати потужні та розширювані компоненти, які можна використовувати для створення складних додатків на Angular.

Розширювати HTML-словник додатка можна за допомогою спеціального синтаксису Angular у шаблонах. Наприклад, Angular дозволяє динамічно

отримувати та встановлювати значення елементів DOM (Document Object Model) за допомогою вбудованих функцій шаблону, змінних, прослуховування подій та зв’язування даних.

Більшість синтаксису HTML може бути використана в шаблоні. Однак, оскільки шаблон Angular є частиною загальної вебсторінки, а не її повною версією, не потрібно включати елементи, такі як <*html*>, <*body*> або <*base*>, і можете зосередитися лише на тій частині сторінки, над якою працюєте [14].

Під час розробки меншої частини системи, наприклад модуля або класу, можуть виникнути потреби у функціональності інших класів. Наприклад, потрібен HTTP-сервіс для здійснення запитів до серверної частини. Впровадження залежностей (DI) - це патерн проєктування та механізм, який дозволяє створювати та передавати частини додатка іншим частинам, які залежать від них. Angular підтримує цей патерн та дає можливість використовувати його в своїх додатках для підвищення гнучкості та модульності.

У Angular залежності, як правило, представлені сервісами, але можуть бути також значеннями, такими як рядки або функції. Інжектор додатка (який створюється автоматично під час завантаження) створює екземпляри залежностей при потребі, використовуючи налаштований постачальник сервісу або значення.

Сукупність усіх елементів допомагають побудувати потужні, гнучкі та легко підтримувані додатки за допомогою фреймворка Angular.

Перейдемо до технології Docker, технології інкапсуляції додатків. Емблему Docker’у можна побачити на рис. 2.8.

Docker - це сервіс для запуску додатків у контейнерах, який можна використовувати як у процесі розробки додатків, так і у процесі їх запуску у production. Проте, для запуску додатків у production, Docker часто використовує надбудову під назвою Kubernetes, яка дозволяє керувати запуском різних контейнерів на різних серверах. Отже, які завдання вирішує Docker і чому варто використовувати Docker:

- Додатки запускаються в ізольованому середовищі, яке називається контейнером. У контейнері додаток працює в ізоляції від інших контейнерів та

зовнішнього середовища, включаючи комп’ютер, на якому запускаються контейнери.

- Додаток легко запускати на різних серверах. Ви можете взяти контейнерний образ Docker, перенести його з одного комп’ютера на інший і запустити контейнери на новому комп’ютері. Контейнери на різних комп’ютерах будуть поводитися абсолютно однаково [15].
- Всі залежності додатків встановлюються всередині контейнерів, тобто немає потреби встановлювати будь-які залежності поза контейнерами на серверах, де запускаються контейнери.
- Додаток, що запущений всередині контейнерів, можна легко масштабувати шляхом збільшення кількості контейнерів. Іншими словами, одну й ту ж саму програму можна запустити у різних контейнерах.

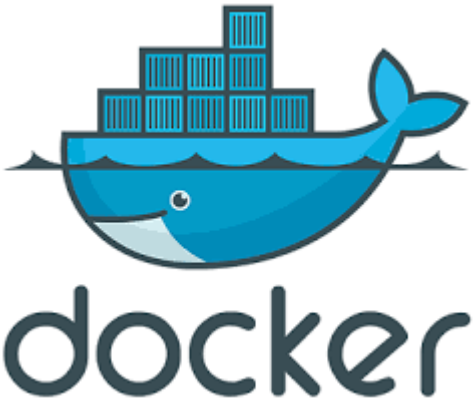

Рисунок 2.8 – Емблема Docker

Контейнер - це виконавче середовище, яке упаковує всі необхідні компоненти для запуску програмного додатку [15]. Він включає в себе усі залежності, бібліотеки, конфігурацію та сам додаток, що дозволяє йому бути переносним та працювати незалежно від операційної системи та конфігурації сервера.

Docker складається з трьох основних компонентів: Docker Client, Docker Daemon та Docker Host.

Docker Client: Це інтерфейс, який дозволяє користувачам взаємодіяти з Docker. Він надає командний рядок та API, за допомогою яких можна виконувати різноманітні операції з контейнерами та образами. Клієнт може бути запущений на

тому ж комп”ютері, де працює Docker Daemon, або віддалено підключатися до нього.

Docker Daemon: Це фоновий процес, який слухає запити від Docker Client та керує контейнерами, образами, мережами та іншими аспектами Docker. Він відповідає за створення, запуск, зупинку та видалення контейнерів, а також за управління образами та ресурсами системи. Docker Daemon також взаємодіє з оперативною системою та обладнанням сервера для забезпечення роботи контейнерів.

Docker Host: Це фізичний або віртуальний сервер, на якому встановлений Docker. Він надає необхідні ресурси, такі як процесор, пам”ять та диск, для роботи контейнерів. Docker Host може бути локальним комп’ютером або віддаленим сервером, на якому запускаються контейнери.

Контейнери створюються на комп’ютері, що працює під управлінням операційної системи Linux. У будь-якої операційної системи Linux є ядро Linux Kernel, а також оперативна пам’ять RAM, процесор CPU та мережеві ресурси Network, такі як мережевий адаптер. Крім того, є диск, на якому зберігаються файли та папки.

Для створення контейнерів вам потрібен Docker Engine, який запускає Docker Daemon - сервіс Docker. Отже, цей мінімальний набір, необхідний для створення контейнерів та зображений на рис. 2.9.

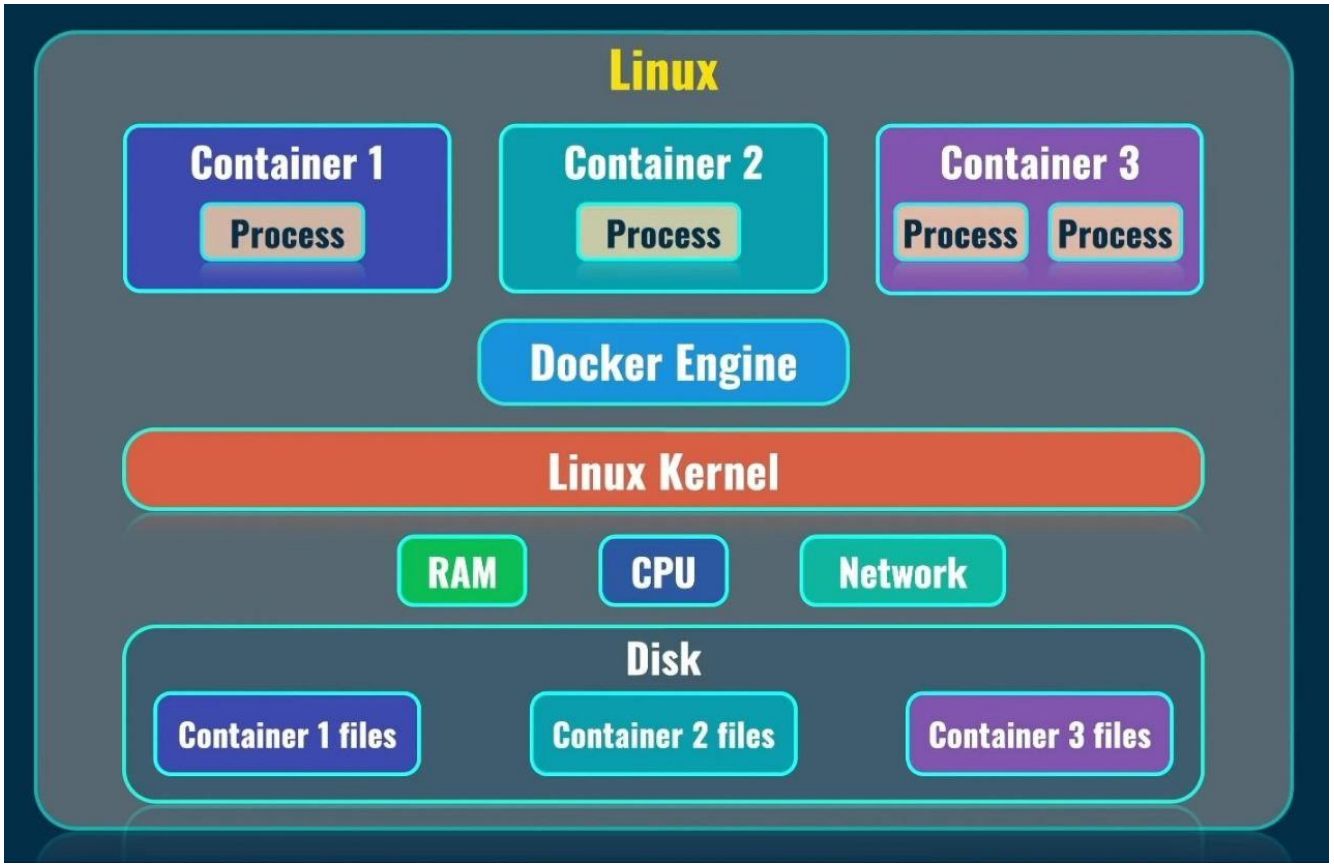

Рисунок 2.9 – Як працюють контейнери

З одного образу можна створити кілька контейнерів. Давайте розглянемо це детальніше. Отже, образ фактично є набором файлів, і образ (Image) є статичним. З цього статичного образу можна створювати контейнери. Наприклад, ви можете завантажити офіційний образ з Docker Hub, і цей образ буде зберігатися в кеші Docker на вашому локальному комп”ютері. На основі цього образу ви можете створити кілька різних контейнерів, запускаючи в них різні додатки.

Контейнер відрізняється від образу тим, що в ньому запущені процеси, і контейнери працюють і виконують певні завдання. Інформація про процес, який запускається в контейнері, що створюється на основі певного образу, зберігається в образі. Це одна з інструкцій, яка просто знаходиться у вигляді текстового рядка в образі. Для кожного контейнера призначено місце на жорсткому диску, і кожен контейнер може записувати або зчитувати файли з жорсткого диску. Отже, виходить, що образи є статичними, а контейнери є динамічними, і контейнери розгортаються з образу.

Усі операції з Docker’ом відбуваються за допомогою команд. Docker має

дуже детальну документацію по використанню цих команда [15].

Отже, після дослідження архітектури та технологій розробки, можна переходити до наступного етапу, а саме до проєктування клієнт-серверного застосунку.

РОЗДІЛ 3 ПРОЄКТУВАННЯ КЛІЄНТ-СЕРВЕРНОГО ЗАСТОСУНКУ

3.1. Дослідження та використання мікросервісу "віддаленого" компілятора

Для перевірки коду учасників, як вже було проаналізовано в першому розділі роботи, будемо використовувати відкритий репозиторій RemoteCodeCompiler [16]. Автором компілятора є Maaraki Zakaria (рис. 3.1).

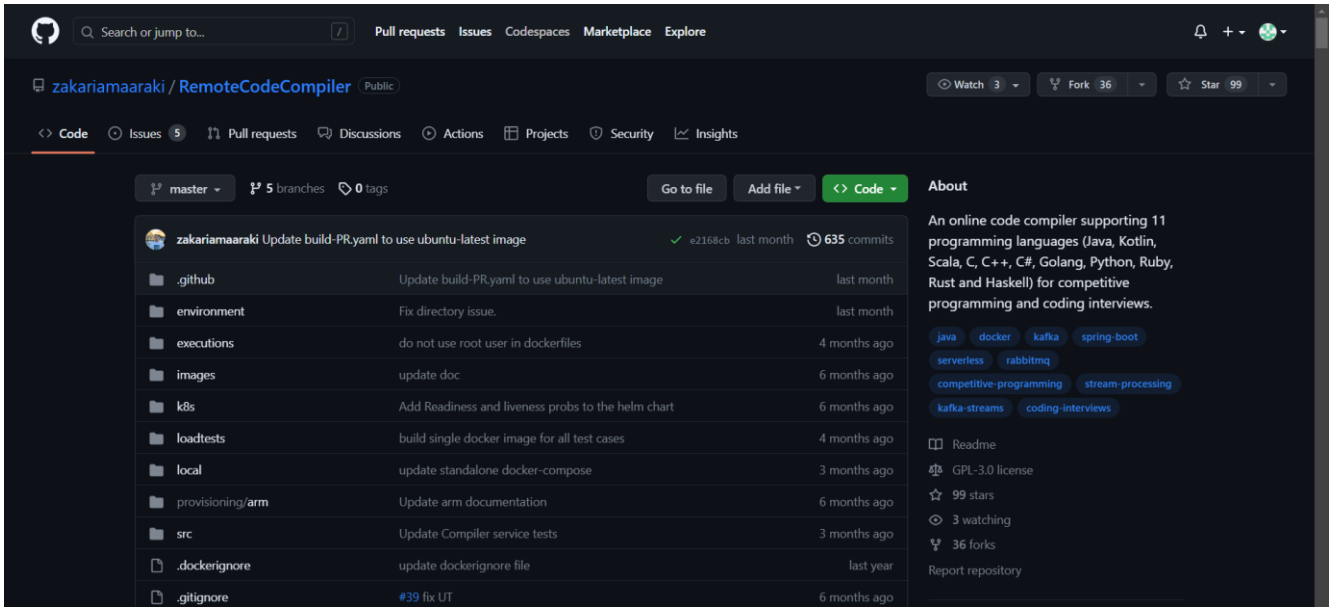

Рисунок 3.1 – Репозиторій з кодом

Онлайн-компілятор коду, що підтримує 11 мов програмування (Java, Kotlin, C, C++, C#, Golang, Python, Scala, Ruby, Rust та Haskell), який можна використовувати для змагального програмування, або технічних співбесід. Цей мікросервіс виконує код віддалено за допомогою контейнерів Docker для розділення середовищ виконання.

Як працює даний мікросервіс? Сервіс працює на основі http запитів. Коли отримується запит, компілятор створює контейнер, який відповідає за компіляцію вихідного коду, що був наданий (цей контейнер має спільний том з основним додатком). Після успішної компіляції для кожного набору тестових випадків створюється окремий контейнер виконання з власним середовищем, який повністю ізольований від інших контейнерів (рис. 3.2).

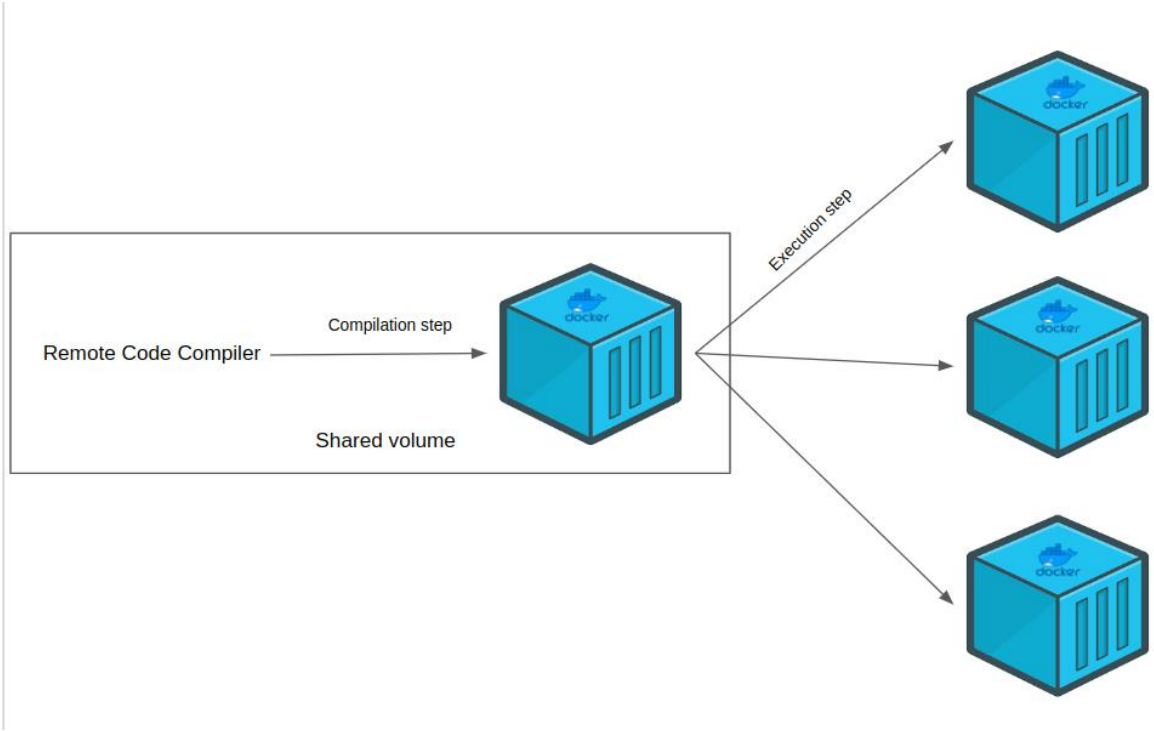

Рисунок 3.2 – Візуалізація процесу виконання коду мікросервісом

Основний endpoint для використання “/api/compile/json”. Цей POST запит представлений на рис. 3.3.

Цей ендпоінт приймає запит на виконання вихідного коду на кількох тестових випадках.

Параметри запиту:

- “request”: JSON-об’єкт, який містить вихідний код, вхідні та вихідні файли, обмеження на час виконання та обмеження на пам’ять;

- “userId” (необов’язковий): ідентифікатор користувача;

- “prefer” (необов’язковий): вказує бажану операцію (наразі доступне лише “prefer-push”);

- “url” (необов’язковий): URL, на який повинна бути відправлена відповідь у разі використання опції “prefer-push”.

Повертає:

- “ResponseEntity<Object>“: відповідь, яка містить один з статусів виконання (Accepted, Wrong Answer, Time Limit Exceeded, Memory Limit Exceeded, Compilation Error, RunTime Error).

За допомогою методу “ExecutionFactory.createExecution” створюється об’єкт

“Execution”, який містить інформацію про виконання, таку як вихідний код, тестові випадки, обмеження на час виконання та обмеження на пам’ять.

Після створення об’єкту “Execution”, звільняється пам’ять, щоб звільнити місце, оскільки виконання може зайняти багато часу.

Залежно від значення “prefer”, виконується компіляція виконання за допомогою “compiler.compile”. Якщо “prefer” має значення “prefer-push”, результат виконання буде відправлений на вказаний URL.

Технологія prefer-push: використовується при потребі отримати відповідь і уникнути таймауту HTTP-запиту. Для цього ми можемо використовувати пуш-сповіщення. Щоб це зробити, нам потрібно передати два значення заголовків (URL, куди ви хочете отримати відповідь, та встановити значення preferPush на prefer-push).

На рис. 3.3 зображений частина класу Requst. Клас “com.cp.compiler.models.Request” представляє об’єкт запиту і має наступні характеристики:

- Поле “sourcecode” представляє вихідний код програми і є обов’язковим. Воно міститься в JSON-полі “"sourcecode"“.

- Поле “language” представляє мову програмування, в якій написаний вихідний код, і є обов'язковим. Воно міститься в JSON-полі “"language"“.

- Поле “timeLimit” представляє обмеження на час виконання програми (в секундах) і є обов’язковим. Воно міститься в JSON-полі “"timeLimit"“.

- Поле “memoryLimit” представляє обмеження на використання пам’яті програмою і є обов’язковим. Воно міститься в JSON-полі “"memoryLimit"“.

- Поле “testCases” представляє тестові випадки для виконання програми і є обов’язковим. Воно міститься в JSON-полі “"testCases"“. Тестові випадки зберігаються у вигляді “LinkedHashMap<String, TestCase>“, де ключ - це ідентифікатор тестового випадку, а значення - об”єкт “TestCase”.

- Метод “getSourcecodeFile()” повертає вихідний код у форматі “MultipartFile”. Цей метод використовується для отримання вихідного коду як файлу, який можна передати при компіляції. Він створює “MockMultipartFile” з

іменем файлу, відповідним мові програмування, та байтовим потоком, що містить вихідний код.

- Метод “getConvertedTestCases()” повертає список перетворених тестових випадків у форматі “List<ConvertedTestCase>“. Цей метод використовується для отримання тестових випадків у відповідному форматі, який може бути використаний при виконанні програми. Він використовує “TestCaseMapper” для перетворення тестових випадків у відповідний формат.

Знання усіх елементів класу Request важливий, адже саме ці параметри ми повинні передавати в POST запиті до мікросервісу. Приклад такого POST запиту можна побачити на рис. 3.3.

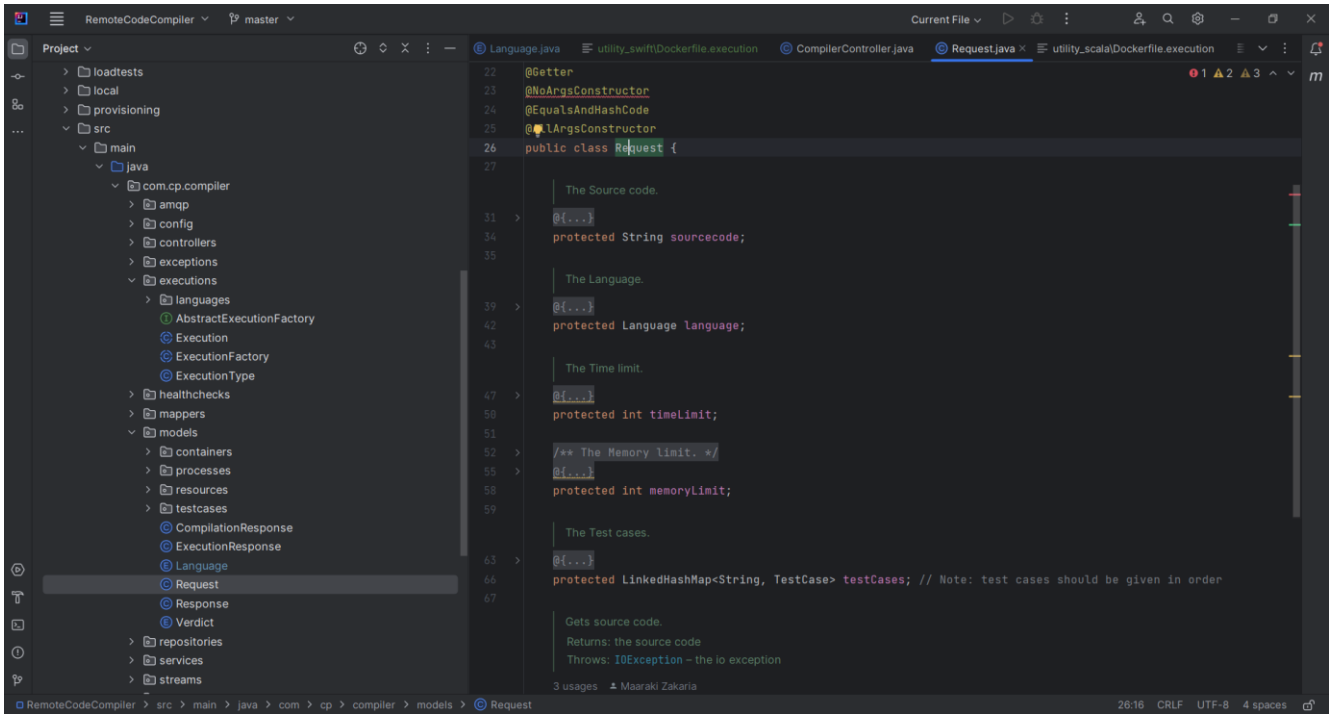

Рисунок 3.3 – Клас Request

Рисунок 3.4 – Приклад body для POST запиту /api/compile/json

Основна логіка створення Image”ів, безпосереднє виконання коду, та побудова команд знаходиться у біні DockerContainerService. Інша бізнес логіка обробки запитів, слідкування за часом виконання та відслідковування пам’яті та оформлення відповіді (Response’у) знаходиться у пакеті com/cp/compiler/services/businesslogic.

Побудова основного Image”у мікросервіса знаходиться у файлі Dockerfile, зображеного на рис. 3.5.

```dockerfile
# Build stage

FROM maven:3.6.0 AS BUILD_STAGE
WORKDIR /compiler
COPY . .
RUN ["mvn", "clean", "install", "-Dmaven.test.skip=true"]


# Run stage

FROM openjdk:11.0.6-jre-slim
WORKDIR /compiler

USER root

COPY --from=BUILD_STAGE /compiler/target/*.jar ../compiler.jar

RUN apt update && apt install -y docker.io

ADD executions ../executions

ADD entrypoint.sh ../entrypoint.sh

RUN chmod a+x ../entrypoint.sh

EXPOSE 8082

ENTRYPOINT ["../entrypoint.sh"]
```

Рисунок 3.5 – Основний Dockerfile

Цей Dockerfile складається з двох етапів: етапу збірки (Build stage) та етапу запуску (Run stage).

Етап збірки (Build stage):

- Використовується базовий образ “maven:3.6.0”.
- Задається робочий каталог “/compiler”.
- Копіюються всі файли і каталоги з поточного контексту в робочий каталог “/compiler”.
- Виконується команда “mvn clean install -Dmaven.test.skip=true”, яка збирає проєкт Maven і пропускає виконання тестів.

Етап запуску (Run stage):

- Використовується базовий образ “openjdk:11.0.6-jre-slim”.
- Задається робочий каталог “/compiler”.

- Встановлюється користувач “root”.
- Копіюється скомпільований JAR-файл з етапу збірки в каталог “../compiler.jar”.
- Виконується оновлення пакетів та встановлення пакету “docker.io”.
- Додаються каталог “executions” та файл “entrypoint.sh”.
- Задається виконавча (executable) дозвільна модель для файлу “../entrypoint.sh”.
- Використовується команда “EXPOSE” для відкриття порту 8082.
- Встановлюється точка входу (“ENTRYPOINT”) для запуску команди “../entrypoint.sh”.

Він здійснює збірку проєкту Maven, підготовку середовища для виконання, копіювання необхідних файлів і налаштувань, встановлення необхідних пакетів, відкриття порту і встановлення точки входу для запуску виконавчого скрипту “entrypoint.sh”.

Як ми вже знаємо, цей онлайн компілятор підтримує з коробки 11 мов програмування. Додамо підтримку ще одної мови програмування, а саме Swift. Це можна зробити, дотримуючись таких кроків:

1. Створити клас у папці "execution", який розширює імплементує Execution стратегію (SwiftExecution).
2. Зареєструвати цей клас в файлі LanguageConfig.java.
3. Додати усі необхідні константи для оперуванням мовою програмування Swift у Docker контейнерах та в самому коді безпосередньо.
4. Створити Dockerfile (у папці "executions"), який містить офіційний образ мови. Dockerfile призначений для виконання компіляції Swift-коду.

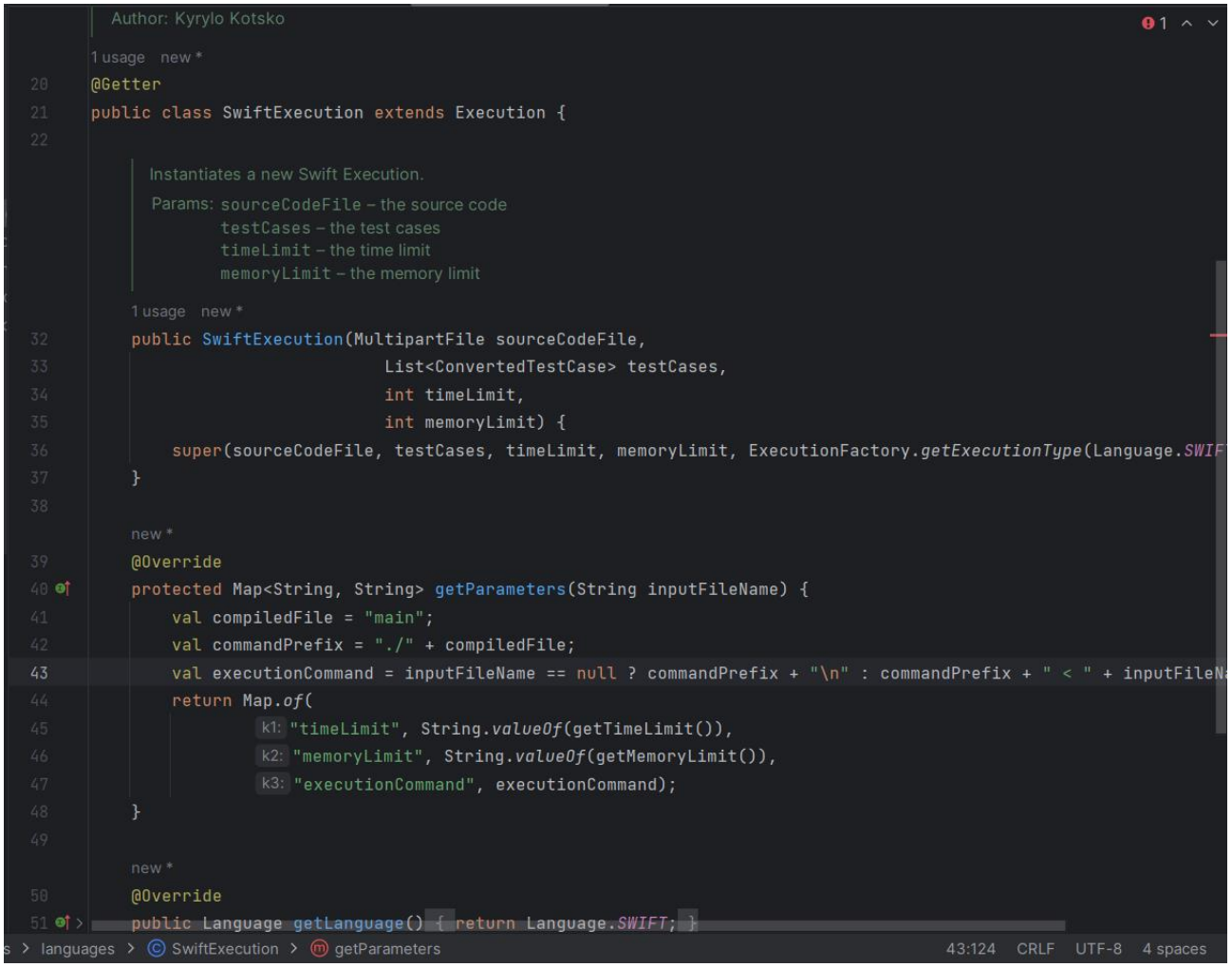

Рисунок 3.6 – Реалізація SwiftExecution стратегії

Реалізацію стратегії SwiftExecution можна побачити на рис. 3.6, а повну реалізацію додавання підтримки мови програмування Swift можна побачити у репозиторії проєкту Goslinger (посилання в додатоку А), хеш посилання комміту: 0968d3e667fc90b147fba45592cc143e9cf82e0c.

3.2 Проєктування платформи Goslinger

Проєкт "Goslinger" розробляється з метою створення потужної та функціональної платформи для проведення дистанційних олімпіад та змагань зі спортивного програмування. Головною метою проєкту є забезпечення зручного та ефективного середовища для участників, які бажають продемонструвати свої навички програмування та вирішувати складні завдання. Проєкт названо в честь творця мови програмування Java: Джеймса Гослінга.

Архітектура платформи Goslinger передбачає використання ряду сучасних

технологій. Для реалізації бекенд частини використовується фреймворк Spring, що надає потужні інструменти для розробки вебдодатків. Для реалізації фронтенду обрано фреймворк Angular, який забезпечує елегантний та дружній інтерфейс користувача.

Окрім того, для забезпечення перевірки коду учасників, використовується мікросервіс з відкритих джерел віддаленого компілятору. Цей компілятор дозволяє виконувати код, наданий учасниками, та повертати результат його виконання. Інтеграція цього сервісу з платформою Goslinger є необхідною для забезпечення надійності та об'єктивності перевірки коду.

Одним із ключових вимог до платформи є підтримка різних мов програмування. Goslinger повинна забезпечувати можливість виконання коду, написаного у 12 різних мовах програмування, а саме: Java, Kotlin, C, C++, C#, Golang, Python, Scala, Ruby, Rust, Haskell та Swift. Це дозволить учасникам вибирати мову програмування, з якою вони найбільш знайомі та впевнені в її використанні.

Крім того, платформа Goslinger передбачає підтримку ролей учасника та адміністратора. Учасники зможуть зареєструватися на платформі, виконувати завдання та переглядати свої результати. Адміністратори матимуть доступ до розширених можливостей управління платформою, включаючи налаштування олімпіад та змагань, перегляд результатів учасників та здійснення адміністративних функцій.

Усі дані, необхідні для функціонування платформи Goslinger, будуть зберігатися у системі керування базами даних PostgreSQL. Використання цієї СУБД забезпечить надійність, швидкодію та масштабованість зберігання даних платформи.

Платформа Goslinger надає можливість створення різноманітних задач для олімпіад та змагань з програмування. Адміністратори та викладачі матимуть зручний інтерфейс для створення та налаштування задач, що дозволить їм гнучко визначати складність, обмеження та критерії оцінювання для кожної задачі.

Однією з важливих можливостей платформи Goslinger є підтримка CSV

формату для імпорту тестових кейсів. Це дає змогу адміністраторам та викладачам зручно завантажувати тестові кейси зовнішніми файлами у форматі CSV. Цей формат дозволяє легко представляти тестові дані та очікувані результати у структурованому вигляді, що спрощує процес створення та управління тестовими наборами.

При імпорті тестових кейсів з файлу CSV, платформа Goslinger забезпечує автоматичне розпізнавання та обробку даних. Кожен рядок у CSV файлі може містити вхідні дані для тесту, очікуваний вихід або будь-які додаткові параметри, які необхідні для оцінки розв'язку задачі. Після імпорту, тестові кейси стають доступними для використання в процесі перевірки розв'язків учасників.

Цей підхід до створення задач та підтримка CSV формату для імпорту тестових кейсів дозволяють ефективно керувати завданнями та спрощують процес внесення та оновлення великого обсягу тестових даних. Завдяки цим можливостям, платформа Goslinger стає потужним інструментом для організації і проведення різноманітних програмувальних змагань та олімпіад.

Загалом, платформа Goslinger розробляється з урахуванням найновіших технологій та вимог у сфері спортивного програмування. Вона має на меті створення зручного та ефективного середовища для проведення дистанційних олімпіад та змагань, сприяючи розвитку програмування та залученню багатьох учасників з різних куточків світу.

РОЗДІЛ 4 РОЗРОБКА ПЛАТФОРМИ GOSLINGER

4.1 Розробка backend та frontend частини додатку

Розпочнемо зі структури проєкту. Репозиторій нашого додатку можна побачити за цим посиланням в додатку А. Увесь проєкт складається з 3 основних частин папок:

- FE: папка frontend частини проєкту;
- BE: папка backend частини проєкту;
- Microservice: папка з запозиченим та модифікованим мікросервісом віддаленого компілятора.

Для BE частини ми використовуємо на версії Java 17.0.5 та версії package'a Spring Boot 3.1.0. На поточний час Spring Boot 3.1.0 є найновішою версією пакета spring-boot-starter-parent, а отже будемо використовувати її. За інструмент для управління проєктом оберемо Maven. Це дозволить нам доволі просто та зручно керувати залежностями, який потребує наш проєкт. Початковий пакет проєкту згенеруємо за допомгою ресурсу Spring Initializr [17]. Початкові дані для генерації можна побачити на рис. 4.1.

Також, при початковій генерації були використані залежності:

- Spring Data JPA – для управління базами даних
- PostgreSQL Driver – драйвер для СУБД PostgreSQL
- Lombok – допоміжна утиліта, для автоматичної генерації конструкторів, геттерів, сеттерів і так далі
- Spring Security – безпековий додаток до проєкту
- Spring Web

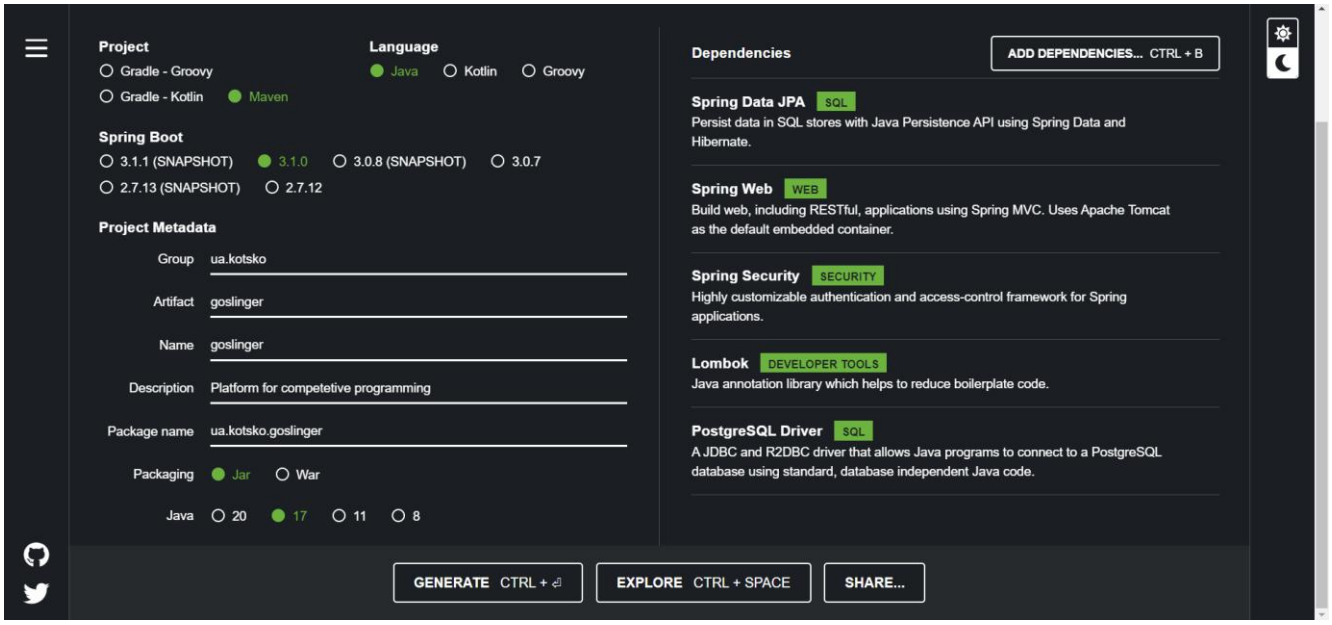

Рисунок 4.1 – Початкові дані для генерації

Для роботи CSV файлами (csv файли з test case’ами) використовуємо бібліотеку OpenCSV. Для зв’язку з мікросервісом будемо використовувати RESTful API, по якому будемо висилати на ендпоінт POST “/api/compile/json”, для первірки кода надісланого користувачем. Таким же чином будемо отримувати відповіді.

Основну бізнес логіку серверної частини можна побачити в пакеті ua/kotsko/goslinger/service. Конфігураційні файли знаходяться в папці ua/kotsko/goslinger/config, а основна конфігурація з під’єднанням бази даних, налаштування порту та збереженням приватного ключа шифрування токенів зображена на рис. 4.2. Перелік сутностей якими ми оперуємо, ми можемо побачити у пакеті ua/kotsko/goslinger/entity, а діаграмму усієї бази даних можна побачити на рис. 4.3.

Для аутентифікації будемо використовувати можливості протоколу JWT. JWT (JSON Web Token) є стандартом для передачі безпечної інформації між двома сторонами у форматі JSON. Він використовується для автентифікації та авторизації в розподілених системах, зокрема у вебдодатках та API.

```yaml
server:
  port: 8250
spring:
  jpa:
    hibernate:
      ddl-auto: create-drop
    show-sql: true
    properties:
      hibernate:
        format_sql: true
    database: postgresql
    database-platform: org.hibernate.dialect.PostgreSQLDialect
  datasource:
    driver-class-name: org.postgresql.Driver
    url: jdbc:postgresql://localhost:5432/goslinger
    username: postgres
    password: admin

jwt:
  secret-key: 24432646294A404E635266556A586E3272347537782141254 42A472D4B6150645367566B597033733676387 92F423F4528482
  access-token:
    expiration:
      days: 1
  refresh-token:
    expiration:
      days: 7
```

Рисунок 4.2 – Файл application.yaml з основною конфігурацією

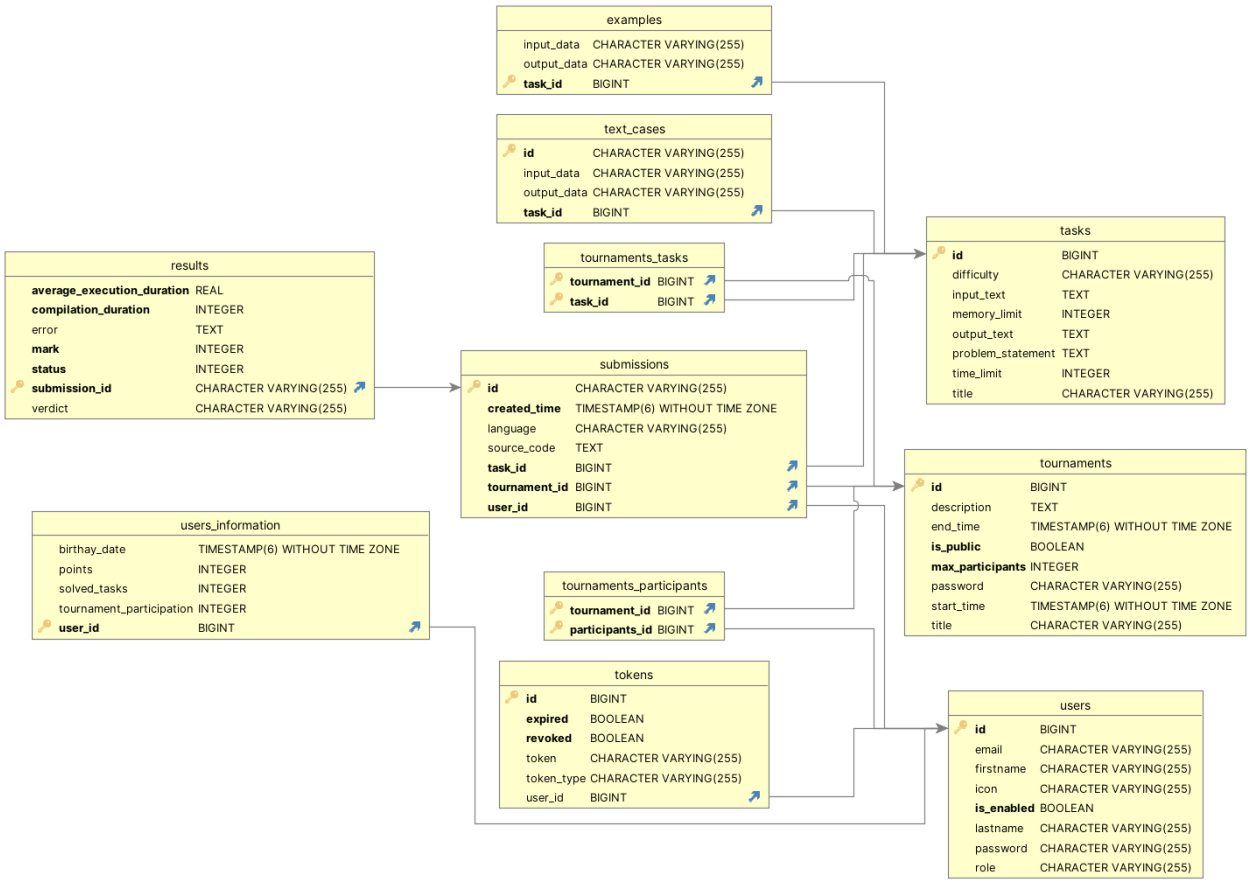

Рисунок 4.3 – Діаграма бази даних

Основна логіка процесу аутентифікації зображена на рис. 4.4. Вона інкапсульована у фільтрі ua.kotsko.goslinger.filter.JwtAuthenticationFilter. Частину коду цього фільтра можна побачити на рис. 4.5. Якщо коротко, то фільтр дістає access_token з заголовка запиту, та перевіряє його на валідність. Для оновлення токену використовується механізм оновлюючого токену: refresh_token. Час життя refresh_token'у: 7 діб.

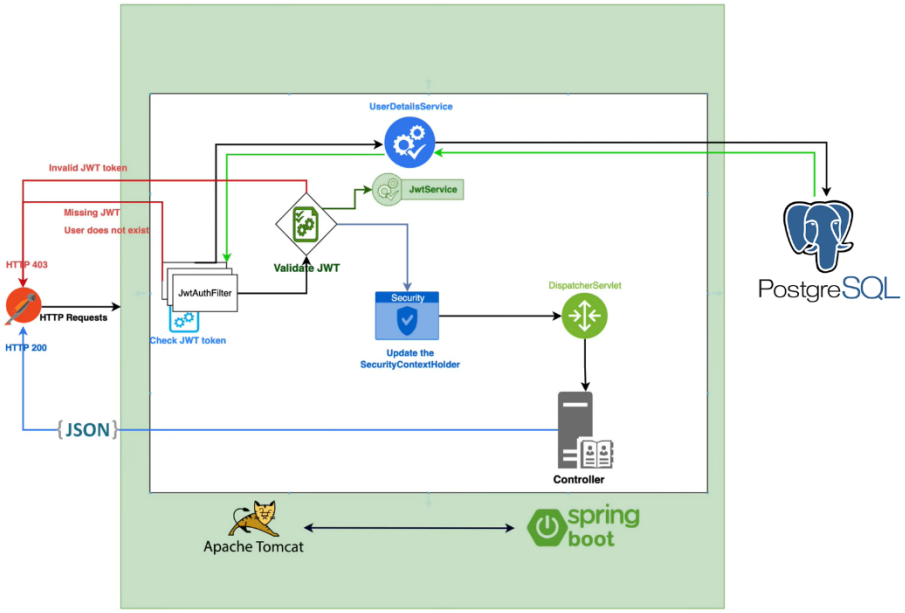

Рисунок 4.4 – Логіка процесу аутентифікації

```
2 usages  Kyrylo_Kotsko
@Component
@RequiredArgsConstructor
public class JwtAuthenticationFilter extends OncePerRequestFilter {

    1 usage
    public static final String GOSLINGER_AUTH_API_PATH = "/api/auth";
    1 usage
    public static final String AUTHORIZATION = "Authorization";
    1 usage
    public static final String BEARER = "Bearer ";

    private final JwtService jwtService;
    private final UserDetailsService userDetailsService;
    private final TokenRepository tokenRepository;

    no usages  Kyrylo_Kotsko
    @Override
    protected void doFilterInternal(@NonNull HttpServletRequest request, @NonNull HttpServletResponse response, @
    ) throws ServletException, IOException {
        if (request.getServletPath().contains(GOSLINGER_AUTH_API_PATH)) {
            filterChain.doFilter(request, response);
            return;
        }
        final String authHeader = request.getHeader(AUTHORIZATION);
        final String jwt;
        final String userEmail;

        if (authHeader == null || !authHeader.startsWith(BEARER)) {
            filterChain.doFilter(request, response);
```

Рисунок 4.5 – Частина коду JwtAuthenticationFilter

Код frontend частини можна знайти у папці FE репозиторію проєкту (додаток А). Емблему додатку можна побачити на рис. 4.6. Проєкт збудований на Angular

12.2.18 та Node.js версії 12.22.12. Інтерфейс додатку виконаний англійською – міжнародною мовою програмування. Девіз платформи “Unleash Your Inner Coding Champion with Goslinger!”, що в перекладі означає: Розкрийте свій потенціал програміста з Гослінгером!

Рисунок 4.6 – Емблема платформи Goslinger

Проєкт створений за допомогою команди “ng new goslinger”. Проєкт являє собо набір компонентів, які комунікують з BE частиною додатку та відображають необхідну інформацію (в залежності від сторінки додатку). Для підсвічування коду використовується залежність ngx-highlightjs, яку ми імпортували в проєкт за допомогою команди “ npm i ngx-highlightjs”.

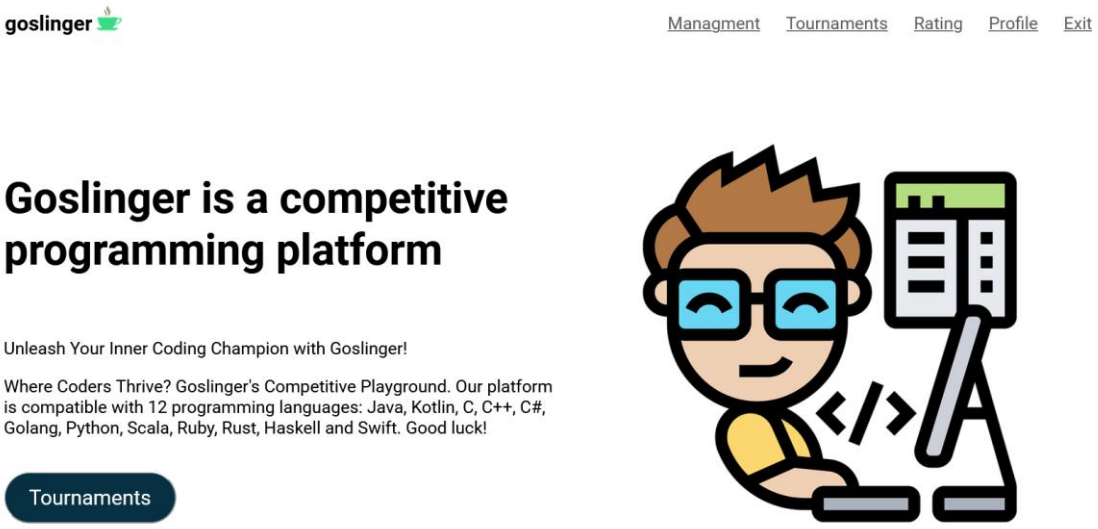

Рисунок 4.7 – Зовнішній вигляд головної сторінки платформи Goslinger

Інтерфейс додатку виконаний у мінімалістичному стилі, для того, щоб користувач фокусувався тільки на важливій інформації. Додаток має сторінки Management, для управління задачами, турнірами та користувачами(тільки для адміністраторів), та сторінки Tournaments(Турніри), Rating(Рейтинг), та Profile(Профіль). Зовнішній вигляд головної сторінки можна побачити на рис. 4.7.

ВИСНОВКИ

Результатом кваліфікаційної роботи стало створення на основі мікросервісної архітектури клієнт-серверної платформи Goslinger для проведення дистанційних олімпіад та змагань зі спортивного програмування. Комп'ютерна інтеграція та автоматизація рутинних процесів значно полегшують роботу судів, членів журі та організаторів, допомагаючи зменшити зусилля і покращити точність та швидкість проведення змагань.

Розробка платформи Goslinger забезпечує учасникам змагань з усього світу можливість брати участь у розумових змаганнях, використовуючи свої навички у програмуванні та розв'язуванні складних завдань. Платформа базується на клієнт-серверній архітектурі та використовує мікросервіси для забезпечення зручного інтерфейсу та безпеки системи.

У процесі дослідження були вирішені наступні завдання: вивчення сучасних тенденцій та особливостей проведення дистанційних змагань зі спортивного програмування, аналіз існуючих рішень та платформ, розробка архітектури клієнт-серверного застосунку, оптимізація функціональних модулів, інтеграція віддаленого компілятора, а також створення зручного та ергономічного інтерфейсу для учасників та організаторів змагань.

Отже, розробка платформи Goslinger є актуальною та важливою, оскільки вона сприяє автоматизації та полегшенню проведення дистанційних олімпіад та змагань зі спортивного програмування, забезпечує зручний інтерфейс для учасників та організаторів, а також підвищує точність, швидкість і безпеку цих заходів.

Результати дослідження та розробки платформи Goslinger можуть мати практичне застосування в сфері проведення розумових змагань та сприяти розвитку спортивного програмування.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Rules of the 2008 ICPC Regional Contests [Електронний ресурс]. – 2008. – Режим доступу до ресурсу: https://web.archive.org/web/20080616223212/https://icpc.baylor.edu/icpc/info/default.htm

2. ICPC Regional Rules for Regionals [Електронний ресурс]. – 2022. – Режим доступу до ресурсу: https://icpc.global/regionals/rules

3. ICPC Policies and Procedures [Електронний ресурс]. – 2017. – Режим доступу до ресурсу: https://web.archive.org/web/20170829081054/https://icpc.baylor.edu/compete/ICPC-Policies-and-Procedures.pdf

4. ICPC FACT SHEET [Електронний ресурс]. – 2017. – Режим доступу до ресурсу: https://web.archive.org/web/20220307093311/https://icpc.global/community/history/Factsheet-2017.pdf

5. Платформа Eolymp [Електронний ресурс] – Режим доступу до ресурсу: https://www.eolymp.com/uk/

6. Purple in Bangalore – Inside Yahoo! Open Hack India 2012 [Електронний ресурс]. – 2012. – Режим доступу до ресурсу: https://web.archive.org/web/20131021180616/http://developer.yahoo.com/blogs/ydn/purple-bangalore-inside-yahoo-open-hack-india-2012-52837.html

7. Hackers Get Hired At Bletchley Park [Електронний ресурс]. – 2011. – Режим доступу до ресурсу: https://web.archive.org/web/20110926233338/http://www.huffingtonpost.co.uk/2011/09/16/hackers-get-hired-at-blet_n_966246.html

8. Портал Благодійного фонду Eolymp [Електронний ресурс] – Режим доступу до ресурсу: https://fund.eolymp.com/uk/

9. КЛІЄНТ-СЕРВЕРНА АРХІТЕКТУРА [Електронний ресурс] // QATestLab. – 2020. – Режим доступу до ресурсу:

https://training.qatestlab.com/blog/technical-articles/client-server-architecture/

10. Буалем Б. Web service conversation modeling: a cornerstone for e-business automation / Б. Буалем, К. Феліче, Т. Фарука // IEEE Internet Computing / Б. Буалем, К. Феліче, Т. Фарука., 2004. – (IEEE). – (10.1109/MIC.2004.1260703). – С. 46 – 54.

11. Frontend vs Backend [Електронний ресурс] // Geeksforgeeks. – 2023. – Режим доступу до ресурсу: https://www.geeksforgeeks.org/frontend-vs-backend/

12. Гасімов О. Мікросервісна архітектура для початківців [Електронний ресурс] / Орхан Гасімов // GlobalLogic – Режим доступу до ресурсу: https://www.globallogic.com/ua/insights/blogs/microservices-architecture-for-beginners-part-one/

13. Spring Framework Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.spring.io/spring-framework/reference/

14. Angular Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://angular.io/docs

15. Docker Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.docker.com/

16. Zakaria M. Remote Code Compiler [Електронний ресурс] / Maaraki Zakaria – Режим доступу до ресурсу: https://github.com/zakariamaaraki/RemoteCodeCompiler/tree/master

17. Spring Initializr [Електронний ресурс] – Режим доступу до ресурсу: https://start.spring.io/

18. Craig W. Spring in Action, Sixth Edition / Walls Craig., 2022. – 520 с. – (Manning Publications Co.). – (ISBN 9781617297571).

ДОДАТОК А

Посилання на репозиторій проєкту Goslinger:
https://gitlab.com/kerchickd/Goslinger