**УДК 004.42**

**РАЗРАБОТКА**

**ПЛАТФОРМЫ ДЛЯ ТРЕНИРОВКИ РЕШЕНИЯ АЛГОРИТМИЧЕСКИХ ЗАДАЧ**

***Вашкулатов Н.А.***

***Рязанский государственный радиотехнический университет  
имени В.Ф. Уткина***

В данной работе разработана платформа для тренировки решения алгоритмических задач на нескольких языках с взаимодействием через WEB-интерфейс.

***Ключевые слова:*** *WEB, Java, Kotlin, Spring Framework, алгоритмы, задачи, Kafka, Python, языки программирования.*

**Введение**

Современное информационное общество сталкивается с постоянным ростом потребности в квалифицированных IT-специалистах. С увеличением числа кандидатов, стремящихся занять вакансии в области программирования, выявление наилучших талантов и их эффективная оценка становятся важнейшими задачами для компаний.

Сейчас на большинстве собеседований обязательно присутствуют вопросы про различные алгоритмы или решение задач прямо во время собеседования. В связи с этим большинству начинающих программистов необходимо постоянно усовершенствовать свои навыки в решении классических алгоритмических задач. Для этой задачи идеально подходит WEB платформа на которой пользователь сможет находить задачи, писать код и проверять решения с любого устройства с WEB браузером, без установки компиляторов, сред разработки и других программ.

В данной работе разрабатывается платформа с WEB-интерфейсом на которой можно писать код, отправлять решение на сервер, проверять ошибки компиляции, выполнения и тестирования.

**Разработка платформы для тренировки решения алгоритмических задач**

Основными сущностями на нашей платформе являются: пользователь, решение, задача, тест, пример обрабатываемых данных. Получим следующую ER диаграмму (рисунок 1).

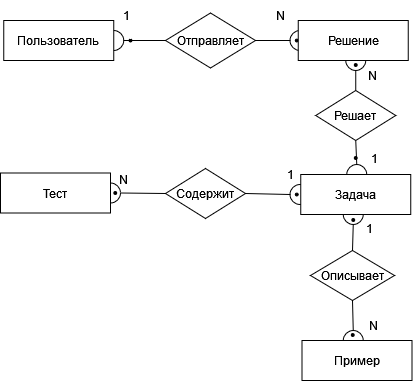


Рисунок 1 – ER диграмма

Из ER диаграммы составим схему базы данных с дополнительными таблицами (рисунок 2).

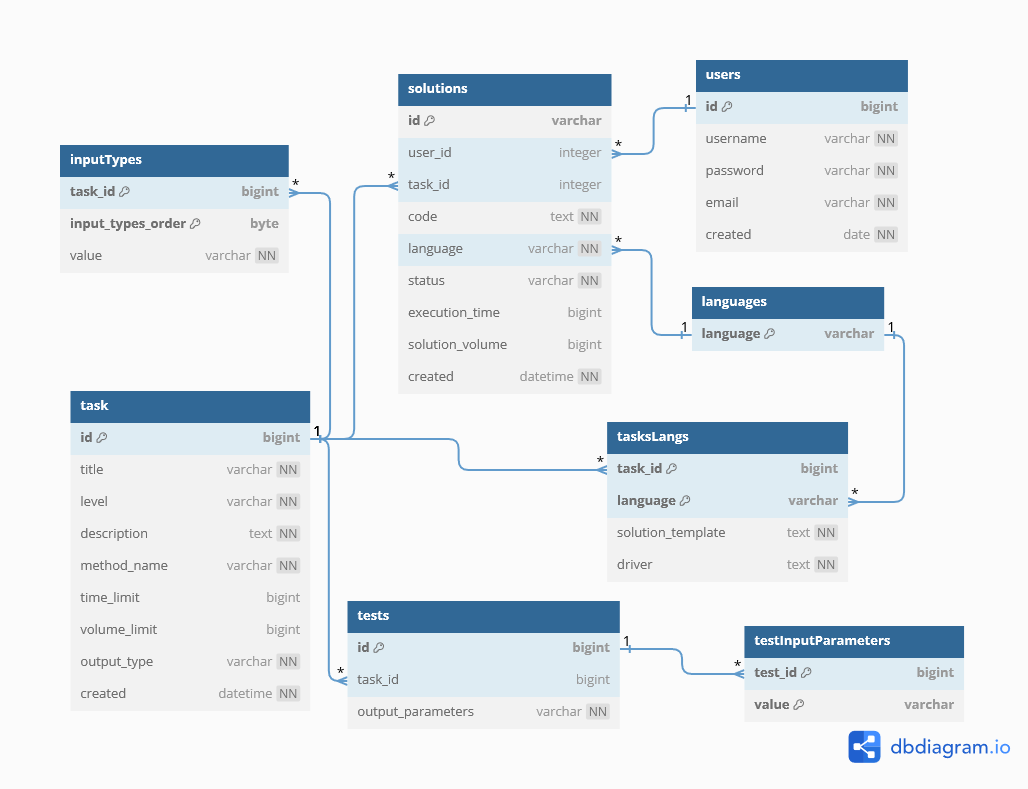


Рисунок 2 – Схема базы данных

Как видно из схемы базы данных реляционное отображение задачи является весьма непростым из-за технической составляющей и в связи с этим в дальнейшем задача, тесты и примеры будут вынесены в документно-ориентированную базу данных. Так же в ранних версиях убран пользователь.

Распределим операции над сущностями в раздельные сервисы, добавим дополнительные технические приложения, получим приблизительную архитектуру нашей платформы (рисунок 3).

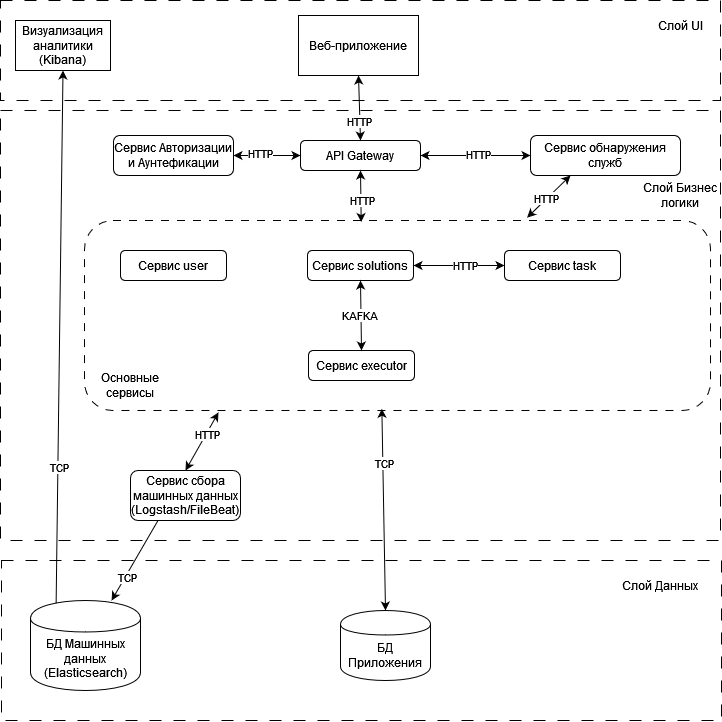


Рисунок 3 – Архитектура

Сервис task отвечает за работу над задачами такими как:

* добавление или удаление задач
* получение информации о задаче
* добавление или удаление тестов

Сервис solution отвечает за:

* формирование данных для отправки в сервис решения
* сохранение или обновление попыток
* формирование данных о попытке для пользователя

Сервис executor отвечает за:

* выполнение и оценку правильности оценки пользователя

Сервис user отвечает за работу с данными пользователя.

API Gateway служит точкой входа в серверную часть платформы и перенаправляет запросы между различными экземплярами сервисов, то есть выполняет роль балансировщика нагрузки. А также шлюз проверяет данные пользователя при доступе к защищенным методам сервисов.

Стек ELK необходим для сбора, хранения и поиска по логам и другим машинным данным производимых платформой.

Не все сервисы платформы взаимодействую между собой, однако все взаимодействуют с API Gateway и базой данных.

Для взаимодействия с сервисом, который выполняет решение задач используется Apache Kafka.

Сервис обнаружения используется для регистрации и хранения адресов экземпляров сервисов платформы. Это позволяет запускать множество экземпляров сервисов на любых машинах по всему миру, увеличивая производительность платформы.

Очень важной составляющей процесса решения задачи является «Драйвер». Драйвер – это основа для решения любой задачи. Он содержит в себе код, который выполняет счет времени выполнения и занимаемой памяти. Так же драйвер отвечает за чтение данных с консоли, преобразование их к нужному типу, вызов метода, который реализовал пользователь с передачей нужных параметров и вывод необходимых данных в консоль, для того, чтобы сервис выполнения прочитал их и сравнил с тестами. Для каждого языка пишется свой драйвер. Для каждой задачи так же составляется свой драйвер на основе типов данных, используемых в решаемой задачи и их количества. Пример драйвера на языке Python:

import time

import psutil

from typing import List

${solution}

def READ\_STRING():

    return input()

def READ\_INTEGER():

    return int(input())

def READ\_STRING\_ARR():

    input\_str = input()

    if input\_str == "[]":

        return []

    input\_str = input\_str.replace("[", "").replace("]", "").replace(" ", "")

    elements = input\_str.split(",")

    arr = [str(element) for element in elements]

    return arr

def READ\_INTEGER\_ARR():

    input\_str = input()

    if input\_str == "[]":

        return []

    input\_str = input\_str.replace("[", "").replace("]", "").replace(" ", "")

    elements = input\_str.split(",")

    arr = [int(element) for element in elements]

    return arr

if \_\_name\_\_ == '\_\_main\_\_':

${paramsInputSection}

    s = Solution()

    start\_time = time.time()

    ret = Solution.${methodName}(s, ${paramList})

    end\_time = time.time()

    process = psutil.Process()

    memory\_info = memory\_info = process.memory\_info()

    used\_memory\_kb = memory\_info.rss / (1024\*1024)

    execution\_time = (end\_time - start\_time) \* 1e9

    print(ret)

    print(execution\_time)

    print(used\_memory\_kb)

После того, как пользователь отправляет решение на выбранном языке, запрос попадает на сервис solution. Он запрашивает данные, необходимые для решения (тесты, типы данных, название реализуемой функции) с сервиса task. Сервис solution сохраняет в базе новую попытку с статусом «Создано», отправляет сообщение на сервис execution и возвращает веб приложению id попытки. В сервисе execution на основе полученного запроса с драйвером создается и сохраняется код, готовый к компиляции/выполнению на выбранном языке. После чего executor создает процесс в системе компилируя и запуская код, передает туда тестовые параметры и считывает результат с консоли. Из консольного вывода формируется результат решения и отправляется на сервис solution. В сервисе solution сохраняется попытка с обновленным статусом. Решение завершено и веб приложение запросит попытку по полученному ранее id.

Для выполнения в промышленной среде каждый сервис собирается в образ Docker при помощи GitHub Actions и публикуется в реестре контейнеров GitHub. После чего на машине создается специальный файл, в котором описаны какие образы использовать, а также различные параметры и настройки сетевых взаимодействий.

**Заключение**

Таким образом, была разработана основа платформы для тренировки решения алгоритмических задач с WEB-интерфейсом. Исходных код доступен на GitHub.

**Список используемых источников**

1. Эккель Б. Философия Java, 4-e изд. – Спб.: Питер, 2021. – 1168c
2. Бек К. Экстремальное программирование: разработка через тестирование.– Спб.: Питер, 2022. – 224с.
3. jUnit5 About: [Электронный ресурс]. URL: https://junit.org/junit5/
4. GitHub Исходный код платформы [Электронный ресурс]. URL: https://github.com/R3nnyWeb/codest-open
5. Codeforces Олимпиадное программирование [Электронный ресурс]. URL: https://codeforces.com/
6. LeetCode Platform to enhance your skills [Электронный ресурс]. URL: https://leetcode.com/