1. Целью НИР было спроектировать и разработать основу для платформы, при помощи которой можно развивать навык решения алгоритмических задач с любого устройства и в любое время.

2) Сейчас почти на всех собеседованиях в крупные компании для начинающих разработчиков обязательно присутствует решение каких-либо алгоритмических задач.

Постоянная практика в решении алгоритмических задач не только развивает умение быстро и эффективно находить оптимальные решения, но также формирует аналитическое мышление и готовность к промышленной разработке. Практическое владение алгоритмами и структурами данных позволяет использовать языки программирования и структуры более эффективно; библиотеки и фреймворки не как зависимости, задающие весь процесс разработки, а как эффективные инструменты для решения сложных задач в реальных проектах.

1. Структура задачи

* Типы данных для входных параметров и результата (мы хотим чтобы решение сводилось к реализации метода. На себя берем считывание и вывод в консоль)
* Тестовые данные, проверяющие правильность решения.
* Шаблон решения для каждого возможного языка решения

А так же всякое типо название задачи, название метода, допустимые языки, уровень сложности

4) Платформа берет на себя ввод и вывод данных то есть мы не можем просто запустить код пользователя и отправить тестовые данные.

* Для чтения, измерения времени, преобразования типов входных данных и вывода результатов решения в консоль используется специальный заготовленный код – драйвер.
* Все эти усложнения нужны для того, чтобы стандартизировать формат входных и выходных данных, благодаря чему процесс тестирования не будет отличаться от ЯП к ЯП, а так же относительно просто добавить не стандартный тип.

5)

1. Место куда вставляется код пользователя.
2. Функции чтения данных определенных типов из консоли.
3. Секция для чтения входных параметров (вызов функций их п. 2)
4. Начало отсчета времени
5. Вызов кода, отправленного пользователем, с передачей параметров
6. Конец отсчета времени
7. Подсчет памяти
8. Вывод результатов в консоль

6) Для автоматизации этих процессов может использоваться скрипт, или приложение, которое получает запрос с кодом, входными данными и языком, а возвращает результат выполнения.

Для осуществления этого необходимо, чтобы в окружении, где выполняется приложение были установлены нужные компиляторы и интерпретаторы.

Однако, поскольку приложение может запускаться в разных окружениях при разработке, тестировании и промышленном выполнении, было решено использовать Docker.

Коротко говоря, при помощи Docker можно настроить систему, сделать «снимок» - образ, и опубликовать его в специальном хранилище. После чего можно запускать экземпляр этого образа – контейнер, и везде будет настроенное окружение.

7) Каждая задача содержит набор тестовых данных, который проверяет правильность решений, отправляемых пользователем.

* Тест состоит из массива входных данных и соответствующего ответа.
* Может быть несколько вариантов ответов для одних входных данных.
* Входные и выходные данные это просто строки, что упрощает тестирование.

8) Архитектура. Веб приложение взаимодейтсвует с шлюзом, который перенаправляет запросы на различные сервисы, а так же распределяет нагрузку.

Так же подключен стэк ELK (elasticsearch, logstash, kibana) для сбора данных, не относящихся к бизнес данным, поиска по ним, визуализации всего этого. В моем случае идет сбор только логов.

В итоге был реализован минимум – Веб приложение, сервис task, executor, БД и ELK.