# VHcontest

(Сейчас для авторизации достаточно ввести что угодно в форму авторизации, будет исправленно к 3му чекпоинту)

**VHcontest** - это система для проверки решений задач по программированию, написанных на разных языках программирования.

Система будет состоять из нескольких подсистем:

1. **Web-client** - веб клиент, запускающийся в браузере, взаимодействует с **API** путем **https**-запросов. Будет написан с использованием фреймворка **vuejs.** Верстка будет сделана при помощи css-фреймворка **bootstrap**, чтобы сэкономить время на верстке, и при этом чтобы она была адаптивной. Система сборки - webpack
2. **Web-API-Server** - будет написан на веб фреймворке **flask + SQLAlchemy**
3. **Testing-Server** - сервер тестирования решений - будет написан на **python**
4. **Push-Server** - сервер рассылки уведомлений. Будет написан на **nodejs.** По сути реализуется паттерн наблюдатель, только взаимодействуют в нем разные сервера.
5. **Database** - база данных, общая для всех подсистем. Хранит информацию о пользователях и о решении задач. Будет использоваться СУБД **MySQL**

**Как же будут запускаться пользовательские решения?**

Эта часть может сильно зависеть от платформы + можно придумать различные вариации реализовать это (перспектива для экспериментов).

Текущая идея - запускать решения от имени какого-то пользователя, не имеющего прав ни к чему кроме директории с исполняемым файлом. Лимиты будут выставляться и отслеживаться при помощи небольшой программы на **C**., с использованием системных вызовов.

Схема работы:
запускается программа на C, она создает два процесса - наблюдатель, который ждет завершения дочернего процесса, и исполнитель, на исполнителя выставляются жесткие ограничения при помощи системного вызова rlimit. Также перенаправляются все потоки во временные файлы. (stdin, stdout, stderr)Когда дочерний процесс закончит работу, наблюдатель смотрит, вложился ли в разрешенные ресурсы процесс или он их перебрал.

UPD: в дальнейшем это все можно перенести на виртуальную машину (чтобы наверняка) . **Эта часть будет зависеть от системных вызовов Linux (а именно: fork, exec, wait, itimer, rlimit и возможно других)**

**UPD:**
В процессе разработки удалось избавиться от этого модуля, так как все системные вызовы которые есть в **linux** можно совершать при помощи **python**.

## Преимущества архитектуры

1. Можно запустить сколько угодно серверов для тестирования (Горизонтальное масштабирование)
2. Благодаря реализации **WEB-API** можно писать клиенты под любую платформу (WEB, Android, IOS и т д)
3. **Push-Server** - вынесен на отдельный сервер, поэтому другие компоненты системы от него не зависят. Также рассылка оповещений не будет создавать дополнительной нагрузки на основной сервер.
4. У клиентов есть только доступ к **WEB-API** и возможность подписаться на уведомления **Push-Server** => при реализации клиента не нужно вдаваться в детали реализации всей системы и отдельных ее подсистем, что по сути реализует паттерн Фасад.

## Недостатки архитектуры

1. Общий сервер БД **MySQL** который довольно сложно масштабировать(большинство методов требует дополнительных изменений в код проекта, и иногда серьезных).