Ну что начнем!

План:

Вступление (введение в тему)

- опять двадцать пять

В предыдущих сериях

- что было хорошо, а что было плохо

- реновация

Властелин колец: 2 башни или курсовая: 2 репозитория (постановка задачи)

- идея разделения проекта на 2 части

(1) Ваша Светлость a.k.a. AtomicJudge

- концепт

- реализация

- алгоритм работы

- использование, тестирование и проблемы

- дальнейшая судьба

(2) Сайт или тяп ляп и в продакшн

- концепт платформы age-of-python

- реализация, как её видят пользователи

- реализация, как её видит разработчик

- Выход в свет и тестирование

Заключение

will return in coursework 3: infinity mind

литература и использованные источники

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

**Факультет информатики, математики и компьютерных наук**

**Программа подготовки бакалавров по направлению   
(09.03.04) Программная инженерия**

*Дыряев Даниил Александрович*

**C O U R S E W O R K II**

**Age of python**

Разработка соревновательной образовательной системы с автоматической оценкой решений

|  |  |
| --- | --- |
|  | Научный руководитель  Ст. Преподаватель  Бычков Илья Сергеевич |

Нижний Новгород, 2021

**Вступление**

*“Убийцы и архитекторы всегда возвращаются на место преступления. “*

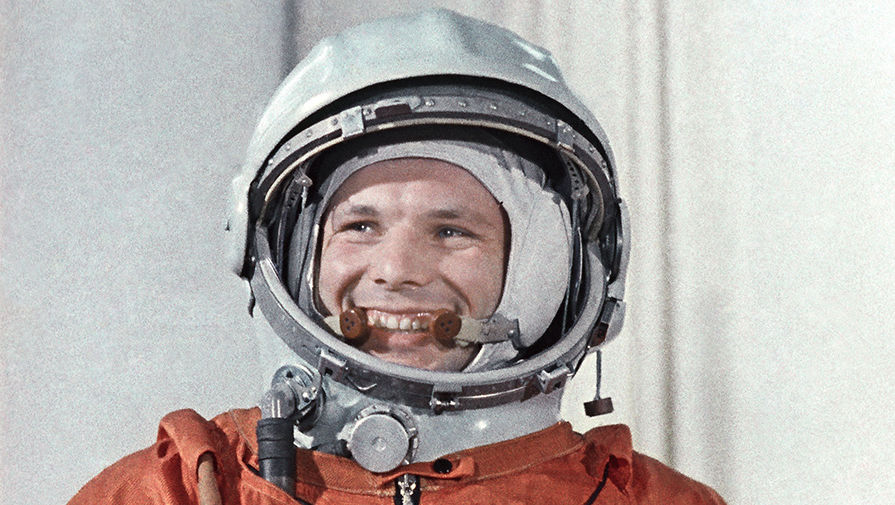
© Питер Устинов

Начнем на веселой ноте – мы и убийцы, и архитекторы одновременно. Дело в том, что я невольно убил двух зайцев, выбрав тему для своей курсовой работы. Случилось это в далеком 2019 году, на заре моего обучения в ВШЭ. Тогда я и представить себе не мог, что стану заниматься ею два года подряд. Утвердившись в своем намерении реализовывать то, о чем вы сейчас читаете, я не только обеспечил себя интересным занятием на пару лет, но и легким росчерком пера расписался в документе на получение кучи практических навыков практически задаром во время его выполнения.

С убийствами разобрались, давайте поговорим об архитекторах, точнее, о том, чем они обычно занимаются. “Архитектор - специалист, который занимается проектированием зданий, кварталов и интерьеров” - прочитать в интернете несложно, придать нужный смысл – сейчас попробуем. Дело в том, что у человека разумного, коими мы все являемся, есть поразительная способность – мыслить абстрактно, отвлеченно, не цепляясь за конкретный объект из нашей вселенной. Давайте взглянем на определение. Может показаться, что здесь определенно есть что-то лишнее, и это правда. Для нашей задачи стоит выкинуть все существительные, и заменить их тем, чем можно обобщить то, что они имеют под собой ввиду. “Архитектор - специалист, который занимается проектированием объектов, группы объектов и их содержанием”. Звучит уже ближе к тому, что мы хотим. А теперь давайте пойдем в обратном направлении, и заменим абстрактные понятия тем, к чему наша деятельность имеет непосредственное отношение. “Архитектор - специалист, который занимается проектированием классов, баз данных и их программным обеспечением” – немного натянуто, но передаёт суть того, что я хочу сказать. Программист – это архитектор, который только на первый взгляд пишет всем непонятные символы и заставляет работать мешки с микросхемами так, как его душе угодно. Любое разрабатываемое приложение требует продуманной структуры его написания и построения, иначе оно как некрасивое здание – будет стоять, но никто не посмотрит на него с восхищением. Именно поэтому создание архитектуры программного обеспечения – прямая ответственность его создателя, который, в случае успешного выполнения своей задачи сможет ностить негласный, но гордый титул – Архитектор.

А теперь стоит объяснить, кто куда вернулся, и при чем тут преступление. Дело в том, что на первом курсе, как я писал выше, я выбрал тему, которой занимаюсь и сейчас. Тогда я думал, что завершу работу летом, и больше к ней не притронусь. Я очень сильно ошибался. То, что у меня тогда вышло – нельзя назвать плохой работой, но количество ошибок, которое я совершал по ходу её выполнения росло экспоненциально. Думаете почему? Я не знаю. Словно нейронная сеть, обученная делать что-то одно, я с каждым шагом, отдаляясь от преднаписанной и тривиальной последовательности действий клонился в сторону, давая все более сомнительный результат. Размышления над проделанной работой дали свои плоды, и вот я здесь, готовый исправить совершенные ошибки и реализовать свою идею лучше. Я вернулся на место преступления.

Поехали.



**В предыдущих сериях**

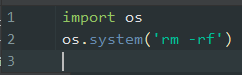
*Что было хорошо, а что было плохо?*

Давайте совершим небольшое путешестивие во времени, на год назад. Не в силах влиять на будущее, не станем менять прошлое, сыграем роль зрителей.

Взглянем на первую версию работы. Пробежимся по плюсам и минусам, стараясь выделить главное. И начнем мы, разумеется, с недостатков.

Первое, что бьёт по глазам – это дизайн. Нет, я не хочу сказать, что он плох или отвратителен, напротив, немалое количество людей выразило мне свои симпатии по отношению к внешнему виду сайта, чему я, конечно, не был удивлен. Я сделал его таким, каким мне он видился, и я был доволен результатом, но потерпел поражение, и вот почему. Я написал дизайн так, что он сносно смотрится на больших широких мониторах, как у меня, например, и совершенно отвратительно выглядит на экранах с соотношением 4:3 и на мобильных устройствах. Надписи и блоки контента порой уезжают в пятое измерение, успешно миновав четвертое. Коротко говоря и выражаясь языком разработчиков – он неадаптивный.

Продолжая складывать кучу из камней в собственном огороде, нельзя не упомянуть про безопасность. На проект у меня ушло пару месяцев, из которых жесткого кодинга 10 часов/день было около двадцати пяти дней. При этом проект выполнял сразу несколько предназначений, и для такого уровня задачи времени было, скажем, маловато. Сайт можно сломать, просто постояв рядом и пошевелив извилинами в голове. Можно даже этого не делать, лучше не станет. Уронить сайт очень легкая задача, не говоря уже о том, что можно просто сделать кучу контента, который нагрузит и без того не мощное железо, на котором стоит проект. Самый легкий способ щелкнуть пальцами и стереть ~~половину~~ весь проект – это послать к решению любой задачи код на языке python с таким содержанием:



И всё, жди пять лет и звони капитану Америка, в лучшем случае.

Хотя, стоит заметить, что у моего друга не получилось это сделать, потому что он забыл считать входные данные, и скрипт закончил выполнение с ошибкой, что конечно и является странным поведением, но особо доставляет, когда облом произошел все-таки не на твоей стороне.

Третий облом и, пожалуй, последний, это то, как работает сайт. Он просто запущен в режиме отладки на сервере, и получить к нему доступ можно просто по IP-адресу, что я, собственно и сделал во время написания этого текста, дабы вспомнить, что я вообще написал год назад. Знаете, привычка забывать плохое. И при чём тут моя курсовая?

А было ли что-то хорошее? Да, думаю было. Опять же, это моё субъективное мнение, но это неважно.

Мне нравится то, как сделана система управления контентом. Её еще называют CMS (Content Management System). Она простая как швейцарские часы и надёжная как три копейки. Именно в таком порядке, не путать! Реализация не сложная, а возможностей достаточно, чтобы покрыть 95% запросов современного образования, в частности обучения программированию.

Еще я хочу приплести к плюсам простой и понятный интерфейс и в целом отсутствие перегруженности приложения лишним функционалом. Звучит сомнительно, но при использовании чувствуется, и чувствуется приятно.

В целом понятно. А что дальше? А дальше только вперёд.

*Реновация!*

*“И чтобы камня на камне здесь не осталось!”* ©



Пора браться за дело. На основе минусов, которые были в предыдущей части, уже можно осуществить планирование для довольно большого объёма работы, что, собственно, и намечается. Кроме того, следует сменить концепт сайта, и сместить приоритет с обслуживаия личных задач в сторону рядового пользователя приложения.

Концепт в этот раз такой: кольцо всевластия мы оставляем у себя, а реплики с таким же количеством возможностей раздаём налево и направо. Что с ними сделают и как их используют, нас уже не волнует. Что это значит, спросите вы? Всё расскажем, всё покажем, но позже.

**Властелин колец: два репозитория**

*Разделение проекта на части*

*“Это невероятно, теперь их стало уже двое”*

© Нут Ганрей

Перейдём непосредственно к структуре приложения и реализации. В этот раз проект напрашивался на логичное и рациональное разделение. Что это значит?

Давайте вспомним, что мы хотим – образовательную платформу для широкого круга лиц, направленную на обучение программированию. Загвоздка здесь в программировании. Одна из задач – это создать автоматизированную систему проверки решений, которая будет покрывать запрос на проверку результатов обучения и каким-то образом интегрировать её в Web-приложение.

Дело в том, что автоматическая проверка программного кода — это задача, которая не вписывается в стандартное поведение аппаратной части интернет-сайта. Рациональный ход в данной ситуации – разделить проект и сделать функционал каждой части в рамках того класса программ, к которому она принадлежит. Пусть сайт остаётся сайтом, а не превращается в мутанта с непонятной структурой. Более того, вынесение задачи по автоматической проверке программного кода в отдельный блок дарит нам гарантию того, что неожиданное поведение не скажется на работе пользовательского интерфейса, так как попросту не будет знать, что он существует.

К чему мы пришли? К тому, что репозитория на github у нас будет два. Проект будет состоять из двух частей, каждая из которых будет отвечать за выполнение определенного класса задач и никак не будет связана друг с другом напрямую. Взглянув на программное обеспечение серверной части Web-приложения, даже не будет понятно, где именно и каким образом проверяются решения пользователей. Это стало возможно благодаря хитрому способу интеграции частей друг в друга, о котором я расскажу немного попозже.

**Часть I  
Ваша светлость a.k.a. AtomicJudge***Концепт*

*“Проблема — это не проблема. Проблема — это твоё отношение к проблеме. Смекаешь?”*

© Капитан Джек Воробей

Начнём мы, пожалуй, с так называемого “Онлайн-судьи”. Так принятно называть программы, которые берут на себя ответственность за автоматическую проверку задач пользователей. Оставив попытки опереться на опыт существующих решений, подумаем, что нужно лично нам.

Я вижу хорошее решение этой проблемы таким: Это самостоятельный модуль, для которого мы определим интерфейс взаимодействия с внешним миром. Внутри он будет представлять из себя “черный ящик”, который будет выдавать данные из зараннее известного множества результатов выполнения программы пользователя. Эти данные будут обработаны и интерпретированы на стороне сервера для отображения конечному пользователю.

Более того, модуль будет являться самостоятельным приложением, выполняющим набор программных инструкций, никак напрямую не влияющий на ПО сайта. Модуль должен работать автономно и в изолированной среде, чтобы избежать нанесения вреда серверу, на котором тот выполняет свою работу.

Так как выполнение чужого программного кода – это очень опасная задача, модуль не будет хранить никаких важных данных, он будет их только обрабатывать и менять. Приложение должно быть взаимозаменяемым и быстроразвертываемым на сервере, в случае утери работающего экземпляра, поэтому следует обеспечить максимально простой процесс установки и начала работы приложения на серверной машине.

*Реализация*

Настало время реализовать наш концепт. Давайте определимся с парой тонкостей, необходимых для понимания того, как это сделать.

До этого момента в тексте фигурировало такое понятие, как “решение пользователя”. Звучит абстрактно. Давайте разберемся, что оно представляет из себя на практике.

В первую очередь – это какая-то часть программного кода, которую нужно выполнить. Также важно знать язык программирования, чтобы понимать, каким образом запускать блок программного кода. Хорошо, уже что-то есть.

Чтобы что-то решать, надо знать семантику получения ответа, или простыми словами, задачу, которую мы решаем. На стороне пользователя смысл придается условием задачи. Для машины же нужен конкретный алгоритм действий.   
Как это понимать? Автор решения пишет алгоритм для того, чтобы получить определенный ответ согласно определенному условию. Условие в рамках алгоритма – это входные данные. Ответ – это выходные данные. Как правило, чтобы объективно оценить решение, нужно проверить его на разных случаях, на разных вариантах “условия” i.e. входных данных. Вывод – входных данных больше одной единицы, соответственно, выходных столько же. (Здесь не стоит вдаваться в определения инъекции и сюрьекции, для нашего алгоритма удобнее представить test-case парой вход/выход, не экономя на занимаемом месте.)

Когда мы получаем какой-то результат выполнения программы пользователя, нам необходимо решить прявильный ли это ответ, или насколько он правильный, неважно, главное его каким-то образом оценить.

Здесь в игру вступает автор задачи. На него ложится роль проверки ответов. Но не считайте его за идиота, не будет же он вручную проверять. Для выполнения данного этапа алгоритма оценивания решения пользователя от автора задачи требуется одно из двух:

1. Эталонное решение собственной задачи, в соответствии с которым будут сгенерированы ответы на каждый тест.
2. Скрипт, который самостоятельно оценит ответ пользователя и выдаст вердикт.

И последнее, ради чего, собственно всё это происходит – это вердикт.

К чёрту слова, переходим к делу.

Будем писать приложение на языке программирования python. Высокая скорость разработки, стильный, модный, молодёжный, что ещё нужно?

У нас будет очень много данных, их следует где-то хранить. Будем использовать мощную и надежную базу данных – PostgreSQL.

Работать приложение будет на сервере с операционной системой ubuntu. Данная ОС хороший выбор для такого рода задач.

Для изолированности и безопасности работы приложения будем использовать его внутри Docker-контейнера, образ для которого предварительно сами соберем.   
Пару слов об образе. В нем будут установлены все языки программирования, поддерживаемые онлайн-судьёй, в том числе и python, а также пакеты, необходимые для работы самого приложения. В нашем случае это psycopg2, который обеспечит работу с базой данных внутри программного кода на языке python. Стоит уточнить, что версия языка – 3.6. Как по мне, она самая стабильная и с ней у меня никогда не возникало проблем, в отличие от остальных.

Когда всё готово, стоит приступить к выполнению. В процессе разработки будем использовать объектно-ориентированный подход, который в разы упростит построение архитектуры приложения и его написания.

Мы обсудили, какие данные нам необходимы для выполнения алгоритма проверки. Это:

* Решение пользователя
* Набор тестов
* Способ оценивания
* Решение Жюри (автора)

Если смотреть со стороны пользователя, то естественным образом возникают следующие элементы:

* Задача
* Решение
* Тесты

В базе данных эти объекты будут храниться в таблицах. Чтобы взаимодействовать с ними в приложении, создадим следующий набор классов:

* Solution (Решение пользователя)
* Test (Единичный тест)
* Task (Задача и способ оценивания)
* CodeFile (Вспомогательный класс, хранящий информацию о коде пользователя)

Для каждого класса реализуем интерфейс, который позволит обновлять их поля в базе данных. Получение всех данных будет происходить в диспетчере, который будет являться по своей сути абстрактной фабрикой. После формирования очереди решений, диспетчер будет их по одному отправлять в класс TaskManager, который является самой весомой частью программы и занимает около 600 строк кода. Этот класс будет ответственен за дальнейшее взаимодействие с решением пользователя.

Подробнее о классе TaskManager:

* Берет на себя ответственность за проверку решения пользователя и рассчет конечного результата.
* Только он обновляет поля в базе данных, остальное приложение лишь обращается к БД с целью данные получить.
* Имеет поддержку компиляции кода на разных языках программирования.
* Использует отдельные системные процессы для управления ходом запуска решений.

К слову о языках программирования. Я решил сделать поддержку:

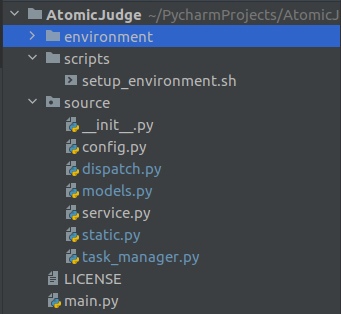
* GNU Assembly
* C (99, 11)
* C++ (11, 14, 17, 20)
* Python (2.7, 3.6)
* Java 8

На мой взгляд, этого базового набора достаточно. При необходимости добавить любой другой язык не составит труда.

Думаю, с реализацией всё. Давайте посмотрим на алгоритм работы этого чуда.

*Алгоритм работы*

Давайте перейдём непосредственно к принципу действия. Вот структура проекта. Она не перегружена и довольно проста для восприятия. В коде несложно разобраться, потому что он самодокументирующийся.



Начнем разбираться, как это работает.

Определим классы для работы с данными. Сделаем обертку над SQL запросами в виде класса DataBase для обобщения работы с ней. Для подключения к бд используем пакет psycopg2. Также определим класс как многопоточно-безопасный синглтон для того, чтобы можно было обратиться к базе данных из любого потока и из любой части программы.

Сначала приложение получает список всех посылок, которые нужно проверить. Это делается с помощью SQL запроса из базы данных с указанием условия, что объект не проверен. При получении множества объектов в виде строк мы создаем местные экземпляры объектов для удобной работы с ними, которые работают по принципу абстрактной обертки над данными для обобщения и абстрагирования от конкретных деталей конкретной модели (объекта).

Дальше, при создании экземпляра посылки, у нее сразу меняется статус, который указывает на то, что посылка в очереди, и ее не требуется брать из бд снова. После этого объект ставится в очередь. На этом работа первого потока закончена.

Второй поток, в свою очередь, проверяет очередь с некоторым периодом, и, если она не пуста, отправляет посылку на проверку. Она делает это синхронно, потому что одно приложение разработано ровно на проверку одной посылки в момент времени.

Перейдем к классу TaskManager, который является оберткой над низкоуровневыми действиями по проверке посылки.

Рассмотрим шаги действия:

1. Подготовка среды:

Создается пустая папка, там создается файл с кодом пользователя и решением жюри при необходимости.

1. Сборка решения жюри и пользователя  
   В зависимости от языка, решение компилируется, или, в случае с интерпритируемыми языками, ничего не происходит.
2. Получение набора тестов.  
   Из базы данных получаются все тесты с правильными ответами и на основе них создаются обертки – классы-модели.
3. После этого исполняемый файл прогонятеся на всех тестах, в отдельном процессе, с контролем времени и кодом возврата. В соответствии с поведением кода, алгоритм собирает данные.
4. По окончании тестирования, алгоритм решает, какой вердикт выставить пользователю. В соответствии с ним обновляются данные в базе данных.

*Использование, тестирование и проблемы*

Приложение предназначено для работы в docker контейнере, потому что это обеспечивает безопасность проверки.

Одновременная проверка только одной посылки даёт ряд преимуществ:

* Возможность ограничения оперативной памяти для проверки пользовательского кода.
* Как сказано выше, полная безопасность для машины, на которой запущено приложение.
* Мгновенное развертывание на машине с Windows или Linux.

Если приложение столкнулось с неожиданным поведением кода пользователя, его можно просто удалить, и не беспокоиться о проблемах, которые возникли во время выполнения.

В процессе создания и использования приложения было пофикшено множество уязвимостей. Также планируется выявить оставшиеся проблемы при использовании системы в “боевых условиях”. (Планируется провести соревнование, но уже после срока сдачи курсовой работы. Об этом подробшнее будет написано ниже).

Из имеющихся проблем можно выделить следующее:

* Нет абсолютной уверенности в том, что код пользователя не навредит системе (docker) – реализация абсолютной безопасности внутри приложения – нерациональна.
* Нет поддержки более универсальной базы данных. Поддержка подстроена под ту модель, которая существует в текущем проекте.

**Сайт, собственной персоной**

*Концепт платформы age-of-python*

Давайте поговорим о концепте проекта в целом.

Ключевая идея состоит в том, чтобы сделать платформу максимально доступной и понятной для использования широкого круга лиц. Под широким кругом лиц подразумевается аудитория потенциальных преподавателей и создателей контента, которые ищут возможность преобразования своих идей и наработок в реальный продукт для потребителей.