Агрегатор

Сервис предназначенный для распараллеливания HTTP запросов

Введение

Обоснование

Поскольку стандартными средствами PHP не получается добиться безопасного и надежного распараллеливания запросов к поставщикам, было принято решение написать отдельный сервис который получит один запрос, вызовет трансляторы для отдельных поставщиков и будет передавать результаты по мере их поступления.

Минимизировав количество бизнес-логики на стороне **агрегатора** будет проще поддерживать сервис.

Агрегатор должен унифицировано поддерживать работу с различными поставщиками.

Агрегатор располагается по адресу https://github.com/theaspect/aggregator

Бизнес модель

Клиент ищет деталь по артикулу в доступных ему поставщиках. Некоторые поставщики позволяют искать по аналогам деталей и требуют несколько запросов.

Клиент должен максимально быстро получать данные, по мере их поступления т.е. не ждать пока сформируется ответ от всех поставщиков.

У клиента будет ограниченное количество одновременных сессий, для каждой сессии после успешного логина генерируется новый ключ, который и используется для авторизации.

Виды поставщиков

1. Возвращает сразу все товары с аналогами;
2. Возвращает товары без аналогов или с аналогами, в зависимости от параметра "Искать по аналогам" переданного с фронтенда;
3. Возвращает товары без аналогов. Для поиска аналогов нужно передать пару артикул+бренд. После этого делаются отдельные запросы для каждого аналога

Границы проекта

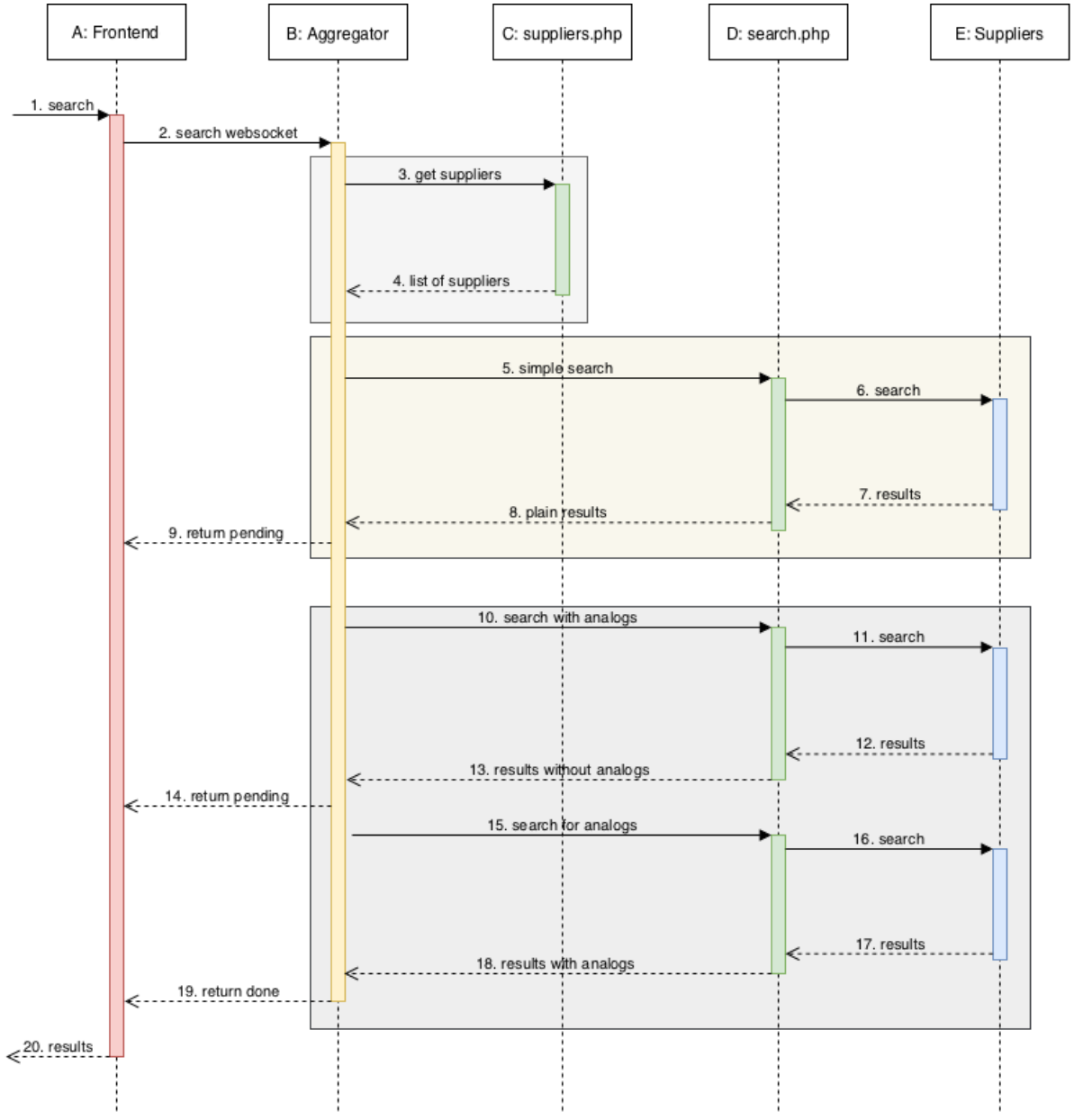
1. **Агрегатор** не занимается запросом данных у **поставщиков** и преобразованием ответов – это ответственность **скриптов**
2. **Агрегатор** не должен кэшировать результаты, по крайней мере уже готовые

3. **Агрегатор** не занимается авторизацией клиентов, авторизацией занимаются скрипты по полученному с фронтенда ключу
4. **Агрегатор** работает только через **websocket** (см. **Приложение 2 Возможные режимы работы фронтенда**)
5. **Агрегатор** не управляет расходом памяти **трансляторов**, память ограничивается внутренними средствами php
6. **Агрегатор** взаимодействует со **скриптами** через stdin и stdout, формат данных JSON
7. **Агрегатор** валидирует все передаваемые данные в формате JSON-а

Требования

Функциональные

Схема работы



Вход

(2) С фронтенда через вебсокет на агрегатор `ws://localhost:PORT/ws` отправляется запрос

```
{
      "code": "3310",
```

```
	"brand": "ctr",
	"apikey": "111111",
	"analog": 1
}
```

В конфигурации указана опция валидные параметры **aggregator.fields.request** где указан список обязательных параметров, которые будут передаваться скрипту.

Получение списка поставщиков

Агрегатор вызывает скрипт `php suppliers.php` **(3)** из конфига **aggregator.script.suppliers** передавая в stdin разрешенные параметры полученные с frontend предварительно проверив валидность JSON и наличие всех параметров.

```
{
	"code": "3310",
	"brand": "ctr",
	"apikey": "111111",
	"analog": 1
}
```

В ответ **(4)** получает список поставщиков доступных данному пользователю в виде JSON. Поле suppliers конфигурируется параметром **aggregator.fields.suppliers**. Полученный массив в поле suppliers добавляется в **очередь** на параллельный поиск.

```
{
	"suppliers": [{
			"class": "CManAutotrade",
			"params": {
				"login": "log1234",
				"password": "1234"
			},
			"code": "3310",
			"brand": "ctr",
			"analog": "1",
			"info": {
				"foo": "bar"
			}
		},
		{
			"class": "CManRossko",
			"params": {
				"login": "log1234",
				"password": "1234",
				"domain": "msk"
			},
			"code": "3310",
```

```
                "brand": "ctr",
                "analog": "1",
                "info": {
                    "foo": "bar"
                }
            },
            {
                "class": "CManArmtek",
                "params": {
                    "apikey": "4d44cbtf14130d2fsdftpq024kd"
                },
                "code": "3310",
                "brand": "ctr",
                "analog": "1",
                "info": {
                    "foo": "bar"
                }
            }
        ]
}
```

Очередь

Очередь запросов на оба скрипта (**suppliers.php** и **items.php**) общая, глубина очереди определяется параметром **aggregator.pool.executor**. На один запрос к поставщикам отводится **aggregator.timeout.script** секунд. Как только истечет время **aggregator.timeout.session**, все элементы очереди относящиеся к одному запросу удаляются из очереди и вебсокет закрывается с ошибкой **E_SESSION_TIMEOUT**.

При добавлении в очередь на отправляется сообщение вида

```
{
    "_session": "11",
    "_task": "ITEMS",
    "_id": 123,
    "_status": "PENDING"
}
```

Где `_id` уникальный номер запроса к поставщику.

Возможные статусы

Для поля `_status` допустимые значения

- `PENDING` задача поставлена в очередь
- `SENDING` ответ от поставщика
- `SUCCESS` все элементы успешно отправлены
- `ERROR` произошла ошибка. Сама ошибка будет в поле `_error`

Получение списка товаров простое

Для тех поставщиков, что позволяют сразу запросить товары с аналогами или без. Агрегатор вызывает скрипт `php suppliers.php` **(5)** передавая в stdin один элемент и массива, полученные на шаге **(4)**. Имя скрипта конфигурируется параметром **aggregator.script.suppliers**

```
{
	"class": "CManAutotrade",
	"params": {
		"login": "log1234",
		"password": "1234"
	},
	"code": "3310",
	"brand": "ctr",
	"analog": "1",
	"info": {
		"foo": "bar"
	}
}
```

В ответ приходит список товаров **(8)** с аналогами или без. На фронтенд передается каждый элемент в отдельном сообщении **(9)**.

```
{
	"items": [{
			"code": "3311",
			"brand": "ctr",
			"name": "Товар1",
			"price": "123",
			"quantity": "30",
			"_session": "11",
			"_task": "ITEMS",
			"_id": 2042,
			"_status": "SENDING",
			"_info": {
				"foo": "bar"
			}
		},
		{
			"code": "3302",
			"brand": "bermo",
			"name": "Товар2",
			"price": "120",
			"quantity": "10",
			"_session": "11",
			"_task": "ITEMS",
```

```
                "_id": 2042,
                "_status": "SENDING",
                "_info": {
                     "foo": "bar"
                }
           },
           {
                "code": "3333",
                "brand": "kama",
                "name": "Товар3",
                "price": "200",
                "quantity": "4",
                "_session": "11",
                "_task": "ITEMS",
                "_id": 2042,
                "_status": "SENDING",
                "_info": {
                     "foo": "bar"
                }
           }
     ]
}
```

Получение списка товаров вложенное

Для тех поставщиков, что требуют запрашивать аналоги отдельными запросами. Агрегатор вызывает скрипт `php suppliers.php` **(10)** передавая в stdin один элемент и массива, полученные на шаге **(4)**. Имя скрипта конфигурируется параметром **aggregator.script.suppliers**

```
{
     "class": "CManAutotrade",
     "params": {
          "login": "log1234",
          "password": "1234"
     },
     "code": "3310",
     "brand": "ctr",
     "analog": "1",
     "info": {
          "foo": "bar"
     }

}
```

В ответ приходит список товаров **(13)**. Если в ответе имеется не пустой массив suppliers который по формату совпадает с **(4)** то эти элементы добавляются в **очередь** на параллельный поиск **(15)**. Из ответа **(13)** возвращается на фронтенд каждый

элемент в отдельном сообщении как есть. Обратите внимание, что тут может появиться бесконечный цикл, который будет ограничен только таймаутом сессии.


```
{
      "items": [{
                  "code": "3311",
                  "brand": "ctr",
                  "name": "Товар1",
                  "price": "123",
                  "quantity": "30"
            },
            {
                  "code": "3302",
                  "brand": "bermo",
                  "name": "Товар2",
                  "price": "120",
                  "quantity": "10"
            },
            {
                  "code": "3333",
                  "brand": "kama",
                  "name": "Товар3",
                  "price": "200",
                  "quantity": "4"
            }
      ],
      "suppliers": [{
                  "class": "CManRossko",
                  "params": {
                        "login": "log1234",
                        "password": "1234",
                        "domain": "msk"
                  },
                  "code": "3311",
                  "brand": "ctr",
                  "analog": "0",
                  "info": {
                        "foo": "bar"
                  }
            },
            {
                  "class": "CManRossko",
                  "params": {
                        "login": "log1234",
                        "password": "1234",
                        "domain": "msk"
```

```
                },
                "code": "3302",
                "brand": "bermo",
                "analog": "0",
                "info": {
                     "foo": "bar"
                }
           },
           {
                "class": "CManRossko",
                "params": {
                     "login": "log1234",
                     "password": "1234",
                     "domain": "msk"
                },
                "code": "3333",
                "brand": "kama",
                "analog": "0",
                "info": {
                     "foo": "bar"
                }
           }
     ]
}
```


Результаты поиска по аналогам **(18)** совпадают с простым поиском **(8)** и возвращаются на фронтенд по мере поступления ответов.


```
{
     "items": [{
                "code": "3311",
                "brand": "ctr",
                "name": "Товар1",
                "price": "123",
                "quantity": "30",
                "_session": "11",
                "_task": "ITEMS",
                "_id": 2042,
                "_status": "SENDING",
                "_info": {
                     "foo": "bar"
                }

           },
           {
                "code": "3302",
```

```
                "brand": "bermo",
                "name": "Товар2",
                "price": "120",
                "quantity": "10",
                "_session": "11",
                "_task": "ITEMS",
                "_id": 2042,
                "_status": "SENDING",
                "_info": {
                     "foo": "bar"
                }

           },
           {
                "code": "3333",
                "brand": "kama",
                "name": "Товар3",
                "price": "200",
                "quantity": "4",
                "_session": "11",
                "_task": "ITEMS",
                "_id": 2042,
                "_status": "SENDING",
                "_info": {
                     "foo": "bar"
                }

           }
     ]
}
```

Наличие непустого массива **suppliers** единственное отличие **вложенного поиска товаров** от **простого**.

Успешное получение данных от поставщика

После отправки всех данных от поставщика, на фронтенд будет отправлено сообщение вида

```
{
     "_session": "11",
     "_task": "ITEMS",
     "_id": 2043,
     "_status": "SUCCESS",
     "_info": {
          "foo": "bar"
     }
```

```
}
```

Ошибки

В случае ошибки, если это возможно передаются все имеющиеся параметры из списка **FAULTY**.
E_ILLEGAL_REQUEST не пришли необходимые параметры на вход
E_ITEMS_JSON ответ items.php не является валидным json
E_ITEMS_RESPONSE в ответе items.php отсутствует обязательное поле items
E_REQUEST_JSON входящий запрос на агрегатор не является валидным json
E_SCRIPT_CANCELLED таймаут на выполнение скрипта подошел до его запуска
E_SCRIPT_ERROR скрипт вернул не нулевой код
E_SCRIPT_START скрипт не найден
E_SCRIPT_TIMEOUT таймаут скрипта
E_SESSION_CLOSED сессия закрыта браузером (фронтенд не увидит эту ошибку)
E_SESSION_REUSED в существующей сессии пришел повторный запрос
E_SESSION_TIMEOUT таймаут сессии
E_SUPPLIERS_JSON ответ suppliers.php не является валидным json
E_SUPPLIERS_RESPONSE в ответе suppliers.php отсутствует обязательное поле suppliers
E_UNKNOWN неизвестная ошибка (фронтенд не должен получать эту ошибку)

При возникновении ошибки, на фронтенд будет передано сообщение вида

```
{
      "_params": {
            "code": "mixed",
            "brand": "foo",
            "analog": "baz"
      },
      "_session": "3",
      "_task": 10,
      "_error": "E_SCRIPT_TIMEOUT"
}
```

В сообщение будет указан список полей, если это применимо, относящихся к сессии из списка который конфигурируется через **aggregator.fields.faulty**. Дополнительно будет указан идентификатор сессии **session** и идентификатор таски **task**. Код ошибки будет в поле **error**.

Валидация ответа

Трансформер должен передать **валидный** JSON, который будет вставлен в ответ в противном случае на фронтенд будет передана ошибка **E_ITEMS_JSON или E_SUPPLIERS_JSON**. Если будут отсутствовать обязательные поля, то вернутся ошибки **E_ITEMS_RESPONSE** и **E_SUPPLIERS_RESPONSE** соответственно.

Логирование

Агрегатор логирует информацию о работе и ошибках в папку **LOGS**
Ротация логов обеспечивается самим Агрегатором
10 файлов по 10 МБ каждый

Мониторинг

По адресу `http://localhost:8000/stats` будет возвращаться JSON:

```
{
      "uptime": "34m 45s",
      "sessionTotal": 15,
      "sessionAlive": 0,
      "sessionFinished": 15,
      "taskTotal": 2034,
      "taskPending": 0,
      "taskSucceeded": 23,
      "taskFailed": 2011
}
```

`uptime` общее время работы агрегатора
`sessionTotal` общее количество сессий, должно быть суммой **sessionAlive** + **sessionFinished**
`sessionAlive` количество сессий открытых в данный момент
`sessionFinished` количество завершенных сессий
`taskTotal` общее количество скриптов поставленных в очередь, должно быть суммой **taskPending** + **taskSucceeded** + **taskFailed**
`taskPending` количество запросов на выполнение скрипта в очереди
`taskSucceeded` количество успешно завершенных скриптов
`taskFailed` количество неуспешно завершенных скриптов или отмененных до начала выполнения

Демонстрация

По адресу `http://localhost:PORT` располагается демонстрационная страница, на которой можно нажав соответствующую кнопку увидеть демонстрацию работы и различные ошибки.

На этой же странице можно нажать кнопку Send All Requests чтобы одновременно запустить все доступные сценарии.

На странице отображается статистика производительности, которая обновляется раз в секунду (в случае большой нагрузки может быть больше).

Два демонстрационных скрипта **`suppliers.php`** и **`items.php`**, которые отдают тестовые ответы или возвращают ошибки в зависимости от запроса, находятся в папке **`script`**, относительно `aggregator.jar`.

Управление процессом (**Этого Нет**)

Java должна стартовать как системная служба при запуске сервера на
CentOS 7+
Ubuntu 16.04+

Предусмотреть стандартные системные скрипты для запуска, перезапуска, остановки, проверки состояния процесса

При стандартном завершении агрегатора или посылании сигнала **SIGTERM** (15) очередь должна быть очищена и все выполняющиеся PHP процессы должны быть завершены. Сигнал **SIGKILL** (9) невозможно обработать.

Конфигурируемые параметры

В папке с приложением можно создать файл **`application.properties`** со следующими параметрами:
`server.port=8000` порт на котором приложение будет слушать
`logging.level.me.blzr.aggregator=DEBUG` уровень логирования, можно повысить до **`INFO`**

`aggregator.script.items=/usr/bin/php -d display_errors=stderr script/items.php` строка запуска скрипта items.php
`aggregator.script.suppliers=/usr/bin/php -d display_errors=stderr script/suppliers.php` строка запуска скрипта suppliers.php

`aggregator.timeout.script=5` максимальное время выполнения одного скрипта
`aggregator.timeout.session=30` максимальное время на одну сессию

`aggregator.pool.executor=20` максимальное количество запущенных php процессов

`aggregator.fields.request=code,brand,apikey,analog` обязательные поля в запросе (через запятую, без пробела)
`aggregator.fields.items=items` обязательное поле для списка деталей
`aggregator.fields.suppliers=suppliers` обязательное поле для списка поставщиков
`aggregator.fields.faulty=code,brand,analog` в случае ошибки эти поля из запроса будут переданы вместе с кодом ошибки (через запятую, без пробела)

Также конфигурационные параметры можно переопределить через параметры командной строки, например `java -Dserver.port=8000`

`-Dlogging.level.me.blzr.aggregator=DEBUG -jar aggregator.jar`. Другие варианты указаны в [10]

Потребляемая память указывается параметрами:
`-Xms64m` начальная память
`-Xmx64m` максимальная память
`-Xss1m` память на один тред

```
java -Xms64m -Xmx64m -Xss1m -jar aggregator.jar
```

Общая память будет примерно равна: **`xmx + xss * (2 + pool.executor + 2 * pool.watchdog)`**

Системные

- Операционная система Ubuntu >=16.04, CentOS >=7
- Количество свободной памяти 64 Java + 64 * 20 PHP >= 1344 МБ
- Standalone PHP (не модуль апача) версии 7.0, 7.1, 7.2 с ограничением по памяти 64 МБ. При запуске скрипт может вызвать команду ini_set и установить переменную memory_limit в нужное значение [5] [6]
  Либо установить ограничение в отдельном php.ini специально для скриптов
- Java 1.8 [8] [9]
  Ограничение по памяти 64 МБ (предварительно, уточнить позже)
- Минимальная версия браузера: IE10, Edge 12, современные Chrome, FF, Safari. Все поддерживают вебсокеты.
- Если в конфиге не указан параметр **PORT**, приложение запускается на порту 8000 и слушает только localhost, иначе используется значение параметра **PORT**
- Nginx версии ??
- Во внешний мир приложение смотрит через Nginx
- Nginx настроен на прокидывание вебсокетов на порт **агрегатора** [4]

Нефункциональные

Документация

С агрегатором должна быть предоставлена следующая документация:

1. по установке
2. по управлению сервисом
3. по конфигурированию
4. по мониторингу

Этапы (25:11)

1 Proof-of-Concept (2:14)

1. **На Nginx настроен проброс вебсокетов**
2. **При открытии урла http://localhost:8000/ отображается простая форма с несколькими полями и кнопкой которая посылает вебсокет запрос**
3. **Фронтенд открывает вебсокет на http://localhost:8000/ws?echo=xyz**
4. **Из каталога с агрегатором вызывается скрипт script.php и передает через stdin все пришедшие параметры:** `php script.php`
5. **Скрипт просто отвечает в stdout json-ом куда вложен ответ** `{"php": {"echo": "xyz"}}`
6. **Агрегатор вкладывает ответ еще раз** `{"aggregator": {"php": {"echo": "xyz"}}}`
7. **Возвращает полученный результат через вебсокет**
8. **Полученный ответ отображается под формой**
9. **После ответа вебсокет закрывается**

2 Агрегатор (6:44)

1. **Распарсить вход и прочитать список параметров**
2. **Создать класс сессии который будет оборачивать ws сессию и параметры**
3. **В задание добавить проверку что сессия еще жива**
4. **Создать реестр сессий**
5. **Поместить задание в реестр сессий**
6. **Создать очередь экзекуторов**
7. **Создать базовый класс задания**
8. **Создать класс для задания suppliers**
9. **Создать класс для задания items**
10. **Создать новое задание и поместить в очередь**
11. **Создать класс для выполнения процессов**
12. **Создать класс для выполнения скрипта suppliers**
13. **Получить результаты и добавить в очередь задания items**
14. **Создать класс для выполнения скрипта items**
15. **Получить результаты и вернуть в ws-сессию**
16. **Обработать таймаут одного задания и вернуть ошибку**
17. **Обработать таймауты всей сессии, остановить все связанные задания и вернуть в сессию все результаты**
18. **В случае инвалидации сессии зачистить задания, потому что возвращать некуда**
19. **Добавить логгер с ротацией**
20. **Обработать ошибки и вернуть в сессию**
21. **Вынести конфигурацию в файл**

3 Тесты (13:11)

1. **Сделать страницу с тестами**
2. **Написать два скрипта которые будут имитировать ответы со случайными задержками**
3. **Отправить запрос который вернет несколько правильных результатов**
4. **Отправить неправильный запрос**
5. **Отправить запрос который упадет с таймаутом на suppliers**
6. **Отправить запрос который вернет ненулевой код на suppliers**
7. **Отправить запрос который не вернет suppliers**
8. **Отправить запрос который вернет ошибочные результаты для items**
9. **Отправить запрос который вернут несколько неправильных items, несколько правильных, несколько с таймаутами**
10. **Отправить запрос который завершится с таймаутом сессии на items вернув несколько правильных результатов**
11. **Отправить запрос который не вернет items**
12. **Отправить запрос который завершится с таймаутом на items не вернув ничего**
13. **Отправить запрос который вернет поиск по аналогам и найдет их**
14. **Добавить кнопку которая одновременно пошлет все запросы**
15. **Написать тесты для каждого этапа трансляции json-а**

4 Мониторинг (1:05)

1. **Добавить счетчики в класс бизнес-логики**
2. **Написать страницу мониторинга которая будет выводить текущие результаты**
3. **Сделать авторефреш страницы**

5 Скрипты управления (1:14)

1. **Написать скрипт который будет останавливать предыдущую версию и запускать новую**

6 Документация (0:40)

1. **Актуализировать документацию**

7 Рефакторинг сообщений (2:15)

1. **Отправлять дополнительные сообщения в начале и конце запроса для каждого из поставщиков**

Приложения

1 Терминология

Очередь единая FIFO очередь в которую добавляются задания на вызов скриптов
Поставщик или **Источник Данных**, к которому посылаются запросы при помощи HTTP запросов. Все поставщики имеют свой интерфейс, поэтому необходимо приводить результат к унифицированному виду
Скрипты два cli скрипта на php, для получения **списка поставщиков** и **поиска**
Транслятор скрипт на php, который получит на вход артикул, запросит у поставщика данные, преобразует к определенному виду и выдаст результат

2 Возможные режимы работы фронтенда

Приложение может работать в трех режимах:

1. **Polling** – Отправляется запрос с идентификатором сессии и раз в X секунд отправляется AJAX запрос на сервер пока не будет получен флаг окончания сессии.
   + Наиболее простое в реализации
   + Поддерживается всеми браузерами
   - Ответ приходит с задержкой в X секунд
2. **Long Polling** – Отправляется запрос с идентификатором сессии и коннект висит открытым до тех пор пока не будет получен первый результат или истечет таймаут коннекта, сразу же посылается следующий запрос. Запросы посылаются пока не будет получен флаг окончания сессии.
   + Поддерживается всеми браузерами
   + Ответ приходит без задержек по мере поступления данных
   - Сложный в реализации как фронтенд так и бекенд
   - Нагрузка на сервер по открытым коннектам
3. **Websocket** – Открывается коннект и в него посылаются данные по мере прихода. Закрывается по окончанию работы.
   + Ответ приходит без задержек
   - Доступно на 94% браузеров [1]

3 Справочная информация

1. Доступность вебсокетов на браузерах https://caniuse.com/#feat=websockets
2. Разбор командной строки в php http://php.net/manual/ru/function.getopt.php
3. Чтение данных из stdin в php http://us2.php.net/manual/en/features.commandline.io-streams.php
4. Настройка вебсокетов в Nginx https://nginx.org/ru/docs/http/websocket.html

5. Ограничение памяти в php http://php.net/manual/ru/function.ini-set.php
6. Ограничение памяти в php http://php.net/manual/ru/ini.core.php#ini.sect.resource-limits
7. Документация по вебсокетам https://developer.mozilla.org/ru/docs/WebSockets
8. Версия java в Ubuntu 16.04 https://packages.ubuntu.com/xenial/java/openjdk-8-jdk-headless
9. Версия java в Centos 7 http://mirror.centos.org/centos/7/os/x86_64/Packages/
10. Конфигурация приложения https://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-external-config.html

4 Конфигурация сервера

Установка jre `sudo apt-get install openjdk-8-jre-headless`
Клонирование репозитрия `git clone https://github.com/theaspect/aggregator.git`
Сборка приложения `./gradlew build`
Деплой приложения `scp build/libs/aggregator-0.4.jar root@X.X.X.X:/var/www/www-root/data/www/service/aggregator/`
Запуск сервера `ssh root@X.X.X.X /var/www/www-root/data/www/service/aggregator/start.sh`

В файле `/etc/nginx/vhosts/www-root/site.org.conf`

```
location ~ /ws/.* {
    rewrite /ws/(.*) /$1 break;
    proxy_pass http://127.0.0.1:8000;
    proxy_set_header Host $host;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "Upgrade";
}
```

Если вместо страницы открывается 404 от битрикса, создайте пустой файл в папке `/var/www/www-root/data/www/site.org/ws`