Простой Web-интерфейс для робота

AsciidocFX

Решаем задачу "сделать веб-морду" робота за пару часов.

Сия инструкция пишется одновременно с производимыми действиями, не обессудьте за краткость.

1. Выбираем MiddleWare

Чтобы заработало в Web, нам нужен промежуточный сервер. В жопу все node.js с их фреймворками, на них просто нет времени.

Обойдёмся по простому - берём промежуточный сервер, который хранит документы (произвольные json) и имеет HTTP RestAPI.

Тогда схема взаимодействия будет простой: робот по http обновляет базу, клиент в виде index.html+скрипты_js читает и отображают веб.

Из подходящих баз: CouchDB (https://couchdb.apache.org) и ArangoDB (https://www.arangodb.com/)

ArangoDB с одной стороны изобилует фичами, мульти-модельная база с развитым языком запросов. Но на мой взгляд излишне тяжёлый для такой простой задачи.

Берём самое простое: CouchDB

2. Вкратце про CouchDB

База хранит много-много json, которые в ней идентифицируются по уникальным _id. Можно обращаться сразу по ним, но для правильного фен-шуя в базу добавляются view - js скрипты которые делают выборки для конечного пользователя.

Для примера мы можем класть в базу json описание открытых ордеров и делаем _view который возвращает коллекцию ордеров.

3. Устанавливаем CouchDB

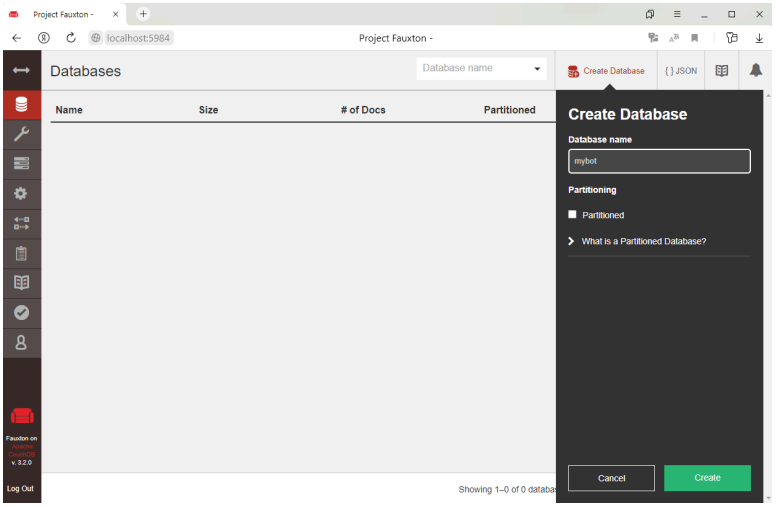
скачиваем с оф.сайта, запускаем инсталлятор.

в процессе установки задаём логин и пароль для администва. И оставляем "галочку" устанавить как Windows-сервис.

после завершения установки - открываем броузер и вбиваем в адресной строке : http://localhost:5984/_utils

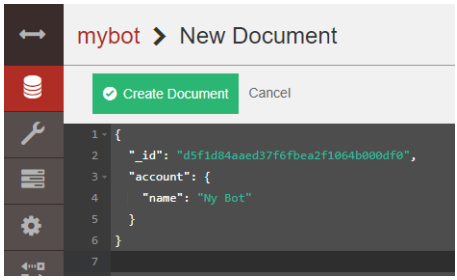
4. делаем и проверяем базу

в интерфейсе CouchDB сразу делаем базу, пусть будет mybot

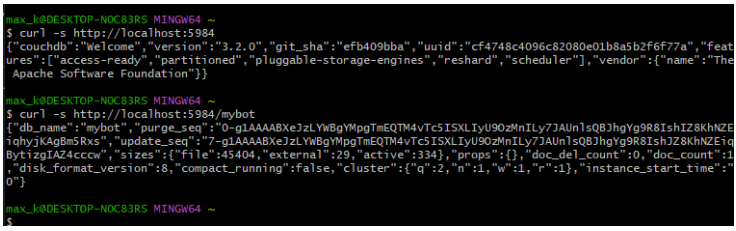


по умолчанию базы создаются с ролью _admin и для доступа будет требоваться авторизация. Для тестовых нужд, просто удаляю роль _admin.

после создание базы, делаем в ней первый "документ" - просто произвольный json.



проверяем, из командной строки запрашиваем сервер и конкретную базу :



обращение к корню вернуло мета-данные сервера, обращение к конкретной базе соответственно её мета-данные

4.1. Уровни и тренды

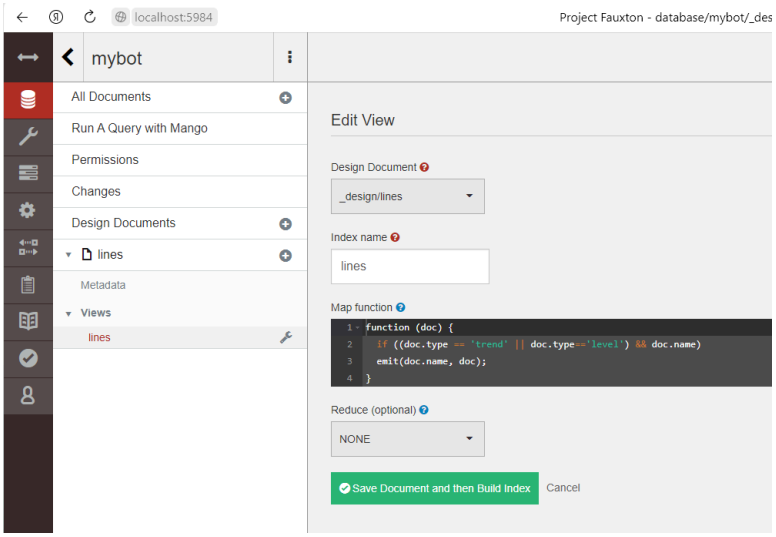
Так как сегодня выходной, торговли нет, то для тестово-демонстрационных целей робот будет показывать в веб уровни и тренды. То есть графические объекты чарта.

В базе про каждый OBJ_TREND OBJ_HLINE будем поддерживать подобный json :

```
{
    "type": "trend",
    "name": "objname",
    "price": [ 0.112 0.223 ],
    "time" : [ xxxx, 2222 ]
}
```

4.2. Полный список

делаем сразу view который просто отдаст всю коллекцию линий:



теперь можно в GUI сделать несколько документов json и посмотреть как оно работает, просто обратившись curl -s http://localhost:5984/mybot/_design/lines/_view/lines

то есть чтобы получить коллекцию наших линий надо будет обратится по /{db}/_design/{index}/_view/{view} ; Каждый документ имеет доп. поля _id - индентификатор назначаемый нам при создании и _rev - номер ревизии, меняется при каждом изменении документа.

потом можем добавить различных view, например "средний уровень всех горизонталей" и так далее.

4.3. добавление документа

каждый документ должен иметь уникальный id, вообще принято использовать OID (GID); В MT сгенерировать OID не прибегая к DLL или WinAPI нельзя. Хорошо что разработчики CouchDB об этом побеспокоились - можно сгенерировать уникальный id обратившись к базе:

```
curl -s -X GET http://127.0.0.1:5984/_uuids
```

images::couchdb_uuids.png[]

а дальше остаётся только задействовать метод PUT:

```
curl -s -X PUT http://127.0.0.1:5984/
mybot/d5f1d84aaed37f6fbea2f1064b0082b9 -d
 { "type":"trend","name":"object22" };
```

images::couchdb_put.png[]

то есть просто методом PUT создали документ с заданным _id и произвольным объектом json внутри. К объекту будут добавлены поля _id (его идентификатор) и автоматически созданный _rev идентификатор ревизии

4.4. изменение документа

изменение документа производится точно так-же, обращением PUT к документу в базе по его идентифкатору. За единственным отличием - в данных надо указать текущий _rev.

```
curl -s -X PUT http://127.0.0.1:5984/
mybot/d5f1d84aaed37f6fbea2f1064b0082b9 -d
 { "type":"trend","name":"object22","color":"red",_rev="77809" };
```

в ответ получим json вида:

```
{ "ok":true, id="273872",rev="384980" }
```

откуда заберём номер последней(актуальной) ревизии документа

удаление документа

не будет неожиданностью, что для удаления документа надо воспользоваться методом DELETE :-) Не без нюансов: надо указать id. последней ревизии документа

curl -s -X DELETE http://127.0.0.1:5984/mybot/d5f1d84aaed37f6fbea2f1064b0082b9?_rev=77809

номер ревизии можно передавать и через заголовок If-Match (см. документацию)

Со всеми требуемыми API вроде как разобрались.

4.5. Проблема с порт 80

Опять проблема откуда не ждали и кто-бы мог подумать, от MetaTrader и MQL: встроенные функции WebRequest не умеют обращаться к нестандартным портам. А у нас порт 5984.

Поэтому придётся поменять порт подключения CouchDB на 80 (или поставить простой WebProxy или настроить переадресацию портов, но сейчас делаем как проще и быстрее)

в настройках меняем адрес на 127.0.0.2 (будет как-бы "виртуальный хост" и не будет путаться с web-сервером если что) и порт на 80

Рестартуем виндовс сервис на всяк.случай

любуюмся результатом:

5. Пишем код

Исключительно в MQL, делая ремарки где-что подправить на C++.

Активно используем WebRequest к адресу 127.0.0.2, там наша база `mybot`.

Складываем/синхронизируем объекты которорые достойны быть в интерфейсе юзера. Помещаем в виде JSON, примерно в том виде что их видим в программе. Конечные представления для пользователя и выборки формируем через _view.

Для генерации UUID объектов опять-же обращаемся к базе. Потом это переделаем на C/C++,WinAPI чтобы было быстро.

Побежали..

5.1. UUID

с матом пополам получаем функцию генерации UUID:

```
bool UUID(string &ret) {
   string cookie="";
   string referer="";
   int timeout=500;
```

```
   char data[]={};
   int data_size=0;
   char result[]; // Ожидаемый результат
   string result_headers="";  // Заголовки результата
   if (!WebRequest("GET",CouchServer+"/
_uuids",cookie,referer,timeout,data,data_size,result,result_headers) ) {
      // обломались
      PrintFormat("UUID gen over couchdb faults");
      return false;
   }
   if ( ArraySize(result)<40 ) {
      // ответ заведомо неверный
      // прислали сообщение про ошибку
      PrintFormat("UUID gen over couchdb faults with result %d
 bytes",ArraySize(result) );
      return false;
   }
   // результат прилетел в UTF8
   // а мы такого не умеем - конвертируем
   string text=CharArrayToString(result);
   // выдираем поле UUID, тут бы надо cjson через DLL, нативные решения
 MQL не вызывают уверенности
   string match="{\"uuids\":[\"";
   int begin=StringFind(text,match);
   if (begin>=0) {
      // найден UUID в строке
      // берём подстроку до завершающей "]
      begin+=StringLen(match);
      int end=StringFind(text,"\"]",begin+1);
      string uuid=StringSubstr(text,begin,end-begin);
      ret=uuid;
      return true;
   }
   return false;
}
```

заодно проверили и работу подхода - к CouchDB обратились, результат получили

5.2. Документы

Сразу немного терминологии : объектами будем называть именно объекты MQL, экземпляры классов. А документами - их json которые сохраняются в базу.

классам добавляем поля _id - идентификатор его документа и _rev - актуальная ревизия.

```
class XxxObject {
....
string _id; // идентификтор документа
string _rev;// актуальная ревизия
};
```

документы соответсвенно будут выглядить как:

```
{
  "_id":"837984792",
  "_rev:"78797987987897"
  прочие поля
}
```

Создание документа

уже выше разобранно, просто вкратце:

- генеририруем идентификатор _id
- методом PUT обращаемся по адресу mybot/{_id} прикладывая json документа
- получаем ответ, если ok==true то берём из него текущую ревизию и запоминаем её в объекте

```
bool CreateDocument(ChartObject &obj) {
   string headers="Content-Type: application/json";
   int timeout=500;
   char data[]={};
   int data_size=0;
   char result[];              // Ожидаемый результат
   string result_headers="";  // Заголовки результата
   string uuid;
   // заводим _id для объекта
   if (!UUID(uuid)) return false;
   obj._id=uuid;    //
   obj._rev="";
   // делаем текст документа
   // обратить внимания - в тексте json нет _rev
```

```
   string text=obj.ToJSON();
   StringToCharArray(text,data,0,WHOLE_ARRAY,CP_UTF8);
   ArrayResize(data,ArraySize(data)-1);
   // отправляем на сервер
   if (!WebRequest("PUT",CouchServer+StringFormat("/mybot/
%s",obj._id),headers,timeout,data,result,result_headers)) {
      // обломались
      PrintFormat("PUT faults");
      return false;
   }
   if ( ArraySize(result)<40 ) {
      // ответ заведомо неверный
      // прислали сообщение про ошибку
      PrintFormat("PUT faults with result %d bytes",ArraySize(result) );
      return false;
   }
   // разбираем ответ
   text=CharArrayToString(result,0,WHOLE_ARRAY,CP_UTF8);
   CJAVal json;
   json.Deserialize(text);
   if ( json["ok"].ToBool() ) {
      obj._id=json["id"].ToStr();
      obj._rev=json["rev"].ToStr();
   }
   return true;
}
```

Обновление документа

Производится ровно так-же как и создание :-) Единственное отличие - у документа уже есть заполненный _id его не надо генерировать и непустой _rev, который поменяемтся после обновления на сервере (в ответ мы получим новый _rev и его запомним)

Удаление документа

Для удаления документа надо обратиться методом DELETE по адресу документа и указать номер ревизии mybot/{_id}?rev={_rev}

```
int DeleteDocument(ChartObject *obj)
{
   if (obj==NULL) {
      PrintFormat("NULL obj");
```

```
      return false;
   }
   if (obj._id=="" || obj._rev=="") {
      PrintFormat("fault _id=%s _rev=%s",obj._id,obj._rev);
      return false;
   }
   return DeleteDocument(obj._id,obj._rev);
}
int DeleteDocument(string _id,string _rev) {
   string headers="Content-Type: application/json";
   int timeout=500;
   char data[]={};
   int data_size=0;
   char result[]; // Ожидаемый результат
   string result_headers="";  // Заголовки результата
   PrintFormat("DELETE "+CouchServer+StringFormat("/mybot/%s?rev=
%s",_id,_rev));
   if (!WebRequest("DELETE",CouchServer+StringFormat("/mybot/%s?rev=
%s",_id,_rev),headers,timeout,data,data_size,result,result_headers) ) {
      // обломались
      PrintFormat("DELETE faults");
      return false;
   }
   CJAVal json;
   string text=CharArrayToString(result,0,WHOLE_ARRAY,CP_UTF8);
   json.Deserialize(text);
   if ( json["ok"].ToBool() ) {
      PrintFormat("Delete ok");
   }
   return true;
}
```

5.3. Синхронизация базы

Самый важный и ответсвенный момент - старт советника и восстановление связи если она прерывалась. За это время реальные объекты могли поменяться, появится новые и прежние удалиться. А база должна содержать актуальное состояние.

Поэтому делаем процедуру синхронизации:

- получаем (или заранее храним) объекты терминала
- получаем список документов с сервера

- сравниваем два списка :
  - новым объектам создаём документы
  - если объект изменился - обновляем его документ
  - если документ не находит соотв.объекта (объект был удалён в терминале) - удаляем документ

```
// синхронизация объектов с сервером
bool SyncChartToServer() {
   // ToDo: коллекцию объектов надо поддерживать постоянно
   // пока что она перечитывается при синхронизации

   // очищаем всё
   ClearObjects(ChartObjects);
   // получаем из терминала объекты в структуры
   int total=ObjectsTotal(0);
   for(int pos=0;pos<total;pos++) {
      string name = ObjectName(0,pos);
      if (name == "") {
         continue;   // не удалось получить объект
      }
      int type=(int)ObjectGetInteger(0,name,OBJPROP_TYPE);
      if (type!=OBJ_TREND && type!=OBJ_HLINE) continue;     // пока
 умеем только тренды и уровни
      ChartObject *obj=AddObject(ChartObjects,name);        // добавляем
 объект
      obj.SetFromChart();
   }
   // объекты ещё не имеют заполненных _id _rev
   // запрашиваем список от сервера
   ChartObject *remote[];           // коллекция сведений о объектах на
 сервере
   // получаем от сервера список объектов, ранее сохранённых там
   if (!GetRemoteObjects(remote)) {
      return false;
   }
   total=ArraySize(remote);
   for (int pos=0;pos<total;pos++) {
      if (remote[pos]==NULL) continue; // перестраховка
      ChartObject *obj=ObjectByName(ChartObjects,remote[pos].name); //
 ищём локальный объект по имени
      if (obj==NULL) {
         // нет такого объекта,
```

```
         // удаляем документ на сервере
         if (!DeleteDocument(remote[pos])) {
            ClearObjects(remote);
            return false;
         }
         delete remote[pos];
         remote[pos]=NULL;
      } else {
         // _id теперь известен
         obj._id=remote[pos]._id;
         obj._rev=remote[pos]._rev;
         // сравниваем данные
         if (obj.type!=remote[pos].type || !
Equal(obj.price,remote[pos].price) || !Equal(obj.time,remote[pos].time)
 || obj.clr!=remote[pos].clr || obj.width!=remote[pos].width) {
            // на сервере некорееектные данные - надо их обновить
            if (!UpdateDocument(obj)) {
               ClearObjects(remote);
               return false;
            }
         }
         // документ учтён
         delete remote[pos];
         remote[pos]=NULL;
      }
   }
   // Ещё раз проходим по коллекции объектов
   // те у которых нет _id - новые, их надо разместить
   //return true;
   total=ArraySize(ChartObjects);
   for(int pos=0;pos<total;pos++) {
      ChartObject *obj=ChartObjects[pos];
      if (obj==NULL) continue;
      if (obj._id=="") {
         if (!CreateDocument(obj)) {
            ClearObjects(remote);
            return false;
         }
      }
   }
   // в массиве remote[pos] остались документы которые есть на сервере,
 но нет в чарте
   // их надо поудалять
   total=ArraySize(remote);
   for(int pos=0;pos<total;pos++) {
      if (remote[pos]==NULL) continue;
```

```
      PrintFormat("----- DELETE 2");
      if (!DeleteDocument(remote[pos])) {
         ClearObjects(remote);
         return false;
      }
      delete remote[pos];
      remote[pos]=NULL;
   }
   ClearObjects(remote);
   return true;
}
```

с кодом практически ВСЁ. Остаются мелкие функции и OnChartEvent - при создании/изменении/удалении объекта чарта - внести соотв. правки в документы

С MQL закончили.

Можно запускать советник и через GUI CouchDB смотреть объекты

5.4. WebGUI

Сразу пилюля - надо запустить какой-никакой а веб-сервер. Можно произвольный, самый простой - нам надо хранить и отдавать в броузер всего 1 файл couch.html . Какой сервер использовать всё равно.

Второй момент - в сервере CouchDB надо разрешить CORS, чтобы к нему с запросами мог обращаться JS броузера. Заходим в настройки CouchDB и включаем. Рестартуем сервис.

Должно получится:

images::couchdb_cors.png[]

Пишем страничку

Страничка будет просто демонстрационная - отображаем общий список объектов и дополнительно свойства/детали отдельного объекта.

С дизайном вообще не заморачиваемя - фактически чуть адаптируем примеры из DataTables.net.

Для получение всего списка документов через AJAX обращаемся к {db}/_design/{index}/_view/{view} где db = mybot, index,view - выборка созданная на предыдущих шагах через GUI CouchDB

Для получения отдельного документа - обращаемся {db}/{_id} где _id - это идентификатор документа

```
<html>
  <head>
    <!-- jQUERY -->
    <script type="text/javascript" charset="utf8" src="https://
code.jquery.com/jquery-3.6.0.min.js"></script>
    <!-- dataTables -->
    <link rel="stylesheet" type="text/css" href="https://
cdn.datatables.net/1.11.5/css/jquery.dataTables.css" />
    <script type="text/javascript" charset="utf8" src="https://
cdn.datatables.net/1.11.5/js/jquery.dataTables.js"></script>

    <link rel="stylesheet" type="text/css" href="https://
cdn.datatables.net/buttons/2.2.2/css/buttons.dataTables.min.css" />
    <script type="text/javascript" charset="utf8" src="https://
cdn.datatables.net/buttons/2.2.2/js/dataTables.buttons.min.js"></script>
    <script type="text/javascript" charset="utf8" src="https://
cdn.datatables.net/buttons/2.2.2/js/buttons.html5.min.js"></script>
    <script type="text/javascript" charset="utf8" src="https://
cdn.datatables.net/buttons/2.2.2/js/buttons.colVis.min.js"></script>

    <script type="text/javascript" charset="utf8" src="https://
cdn.datatables.net/colreorder/1.5.5/js/dataTables.colReorder.min.js"></
script>
  </head>
  <body>
    <script>
    function escapeHtml(unsafe)
    {
        return unsafe
            .replace(/&/g, "&amp;")
            .replace(/</g, "&lt;")
            .replace(/>/g, "&gt;")
            .replace(/"/g, "&quot;")
            .replace(/'/g, "&#039;");
    }
    $(document).ready( function () {
        var table=$('#chart_objects').DataTable( {
```

```
            "columns": [
                {
                    className:      'dt-control',
                    orderable:      false,
                    data:           null,
                    defaultContent: ''
                },
                { "data":"_id" },
                { "data":"name" },
                { "data":"type" },
                { "data":"price.0" },
                { "data":"time.0" },
                { "data":"price.1" ,defaultContent: ''},
                { "data":"time.1",defaultContent: '' }
            ],
            colReorder: true,
            order: [[1, 'asc']],
            "ajax": {
                "url": "http://127.0.0.2/mybot/_design/lines/_view/
lines" ,
                "dataSrc":  function (json) {
                    ///
                    /// получаем список объектов
                    ///
                    var data=[];
                    for (record of json.rows) {
                        var value=record.value;
                        data.push(value);
                    }
                    return data;
                }
            },
            dom: 'Bfrtip',
            buttons: [
                'copy','csv','colvis',
                {
                text: 'Reload',
                action: function ( e, dt, node, config ) {
                        dt.ajax.reload();
                    }
                }
            ]
        } )
        $('#chart_objects tbody').on('click', 'td.dt-control', function
 () {
            console.log("click");
```

```
            var tr = $(this).closest('tr');
            var row = table.row( tr );

            if ( row.child.isShown() ) {
                row.child.hide();
                tr.removeClass('shown');
            }
            else {
                row.child( format(row.data()) ).show();
                tr.addClass('shown');
            }
        } );
        function format ( rowData ) {
            var div = $('<div/>')
                .addClass( 'loading' )
                .text( 'Loading...' );
            ///
            /// Получаем документ конкретного объекта
            ///
            var objurl='http://127.0.0.2/mybot/'+rowData._id.toString();
            console.log("ajax to "+objurl);
            $.ajax( {
                url: objurl,
                dataType: 'json',
                success: function ( json ) {
                    var content="<table>";
                    var num=0;
                    for(key in json) {
                        if (num==0) {
                            content+="<tr>";
                        } else if ((num%2)==0) {
                            content+="</tr><tr>";
                        }
                        content+="<td>";
                        content+=escapeHtml(key.toString());
                        content+="</td>";
                        content+="<td>";
                        content+=escapeHtml(json[key].toString());
                        content+="</td>";
                        //content+="</tr>";
                        num++;
                    }
                    while((num%2)!=0) {
                        content+="<td></td>";
                    }
                    if (num!=0) {
```

```
                    content+="</tr>";
                }
                content+="</table>";
                div
                    .html( content )
                    .removeClass( 'loading' );
            }
        } );
        return div;
    }
    //setInterval(function(tab) {
    //    tab.ajax.reload();
    //} , 1000, table );
} );    // dicument ready
</script>
<table id="chart_objects" class="display">
    <thead>
        <tr>
            <th></th>
            <th>id</th>
            <th>name</th>
            <th>type</th>
            <th>price[0]</th>
            <th>time[0]</th>
            <th>price[1]</th>
            <th>time[1]</th>
        </tr>
    </thead>

    <tbody>
    </tbody>

    <tfoot>
        <tr>
            <td></th>
            <th>id</th>
            <th>name</th>
            <th>type</th>
            <th>price[0]</th>
            <th>time[0]</th>
            <th>price[1]</th>
            <th>time[1]</th>
        </tr>
    </tfoot>

</table>
```

```
  </body>
</html>
```

результат работы страницы и всего проекта :

images::couch_html.png[]

видим список графических объектов (пока-что только горизонталей и трендов), по любому из них можно посмотреть детали. Если нажать кнопочку "reload" список будет обновлён на клиенте.

6. вместо ToDo

Что стоит добавлять в проект в первую очередь - динамическое обновление на клиента (без кнопки reload) - или js который периодически считаывает объекты из базы и выводить изменению куда надо или читает _changes из CouchDB (есть там такая фича - показывать изменения). Но это выходит за рамки MQL и вообще проблемы фронт-энд. Это не затрагивает функционирование робота.

По аналогии с графическими объектами - сделать такое-же с ордерами и позициями. Точно так-же : появился объект "ордер" - завели ему документ, обновился - поменяли, закрылся - удалили. И в CouchDB дабавить выборку ордеров.

И конечно "дело №0" - до всего этого,сделать паузу и через неделю провести ревизию кода. Там просто в коде много чего можно улучшить и обобщить. Но такое делается не по горячим следам.

Что осталось за кадром:

- Всё проделано на localhost`е и надо делать обратные движения:
  - вернуть базу на её стандартный порт
  - настроить reverse-proxy чтобы MQL мог общаться с базой
- Получение UUID всё-таки надо делать через WinAPI, DLL
- WebRequest лучше через DLL - так и к нестандартным портам можно обращаться и сделать из асинхронными. **Сейчас MQL останавливает**

исполнение до завершения запроса. Для торговых программ совершенно недопустимая вещь

7. ИТОГО

Из дивной связки MetaTrader-Субд-Бэк-Фронт мы вместо Субд+Бэк взяли CouchDB (можно другую базу документов) и сконцентировали разработку на MQL. Не особо задумывались над структурой базы, а сохраняли данные как они есть на самом деле. Web-Фронт сделали сугубо демонстрационным.

Сократили себе объём работы раза в два-три.

Что осталось за кадром:

- Всё проделано на localhost`е и надо делать обратные движения:
  - вернуть базу на её стандартный порт
  - настроить reverse-proxy чтобы MQL мог общаться с базой

Резюмируя, основные пункты:

- Для хранения документов (json) используем базу документов с интерфейсом REST. Для простых проектов CouchDB, а если мега-большое то смотрим на ArangoDB
- данные (документы) сохраняем в их естественном виде. Такими как мы их знаем. Выборки и представления - это уже дело базы. MQL вообще не должен задумываться над моделью данных базы.
- Чтобы MQL мог обращаться к базе, она должна быть на 80 порту. Или делаем соотв. настройки базы или настраиваем реверс-прокси.
- Базу документов держим на том-же хосте где MetaTrader. Для скорости MQL←→База. Для доступа снаружи настраиваем репликацию баз. (все современные базы это умеют)