Битрикс24 представляет собой полный набор инструментов для организации работы компании. В нём присутствует все необходимое для создания коммуникаций внутри коллектива, для управления задачами и проектами, для работы с различной документацией, систему управления взаимоотношениями с клиентами (CRM), средства связи и многое другое.
Иногда возникают ситуации, когда стандартными средствами облачного сервиса не получается решить ту или иную бизнес-задачу.
С помощью открытого API клиенты и разработчики могут легко адаптировать «облако» под свои конкретные задачи, настраивая бизнес-логику.

Инструментарий Битрикс24 можно расширять с помощью самостоятельно разрабатываемых приложений. Разрабатываемые приложения условно делят на два типа: приложения разрабатываемые в рамках конкретного проекта и тиражные решения.

Приложения разрабатываемые в рамках конкретного проекта:
При разработке приложений в рамках конкретного проекта есть два варианта расширения функциональных возможностей Битрикс24 на основе REST API - это локальные приложения и пользовательские вебхуки.

Локальные приложения:
Локальные приложения поддерживают все возможности REST API и вызывают REST по протоколу авторизации OAuth 2.0, но устанавливаются только на конкретном Битрикс24, без публикации в каталоге решений. Для добавления локального приложения требуется административный доступ.

Локальные приложения лучше подходят для тех задач, которые требуют создания пользовательского интерфейса:

- различные отчеты;
- дополнительные автообработчики в рамках специфической бизнес-логики;
- решения, требующие управления доступом пользователей;
- чат-боты и приложения, расширяющие функционал мессенджера Битрикс24;
- дополнительные операции для автоматических бизнес-процессов Битрикс24.

Пользовательские вебхуки:
Использование вебхуков сильно упрощает техническую реализацию, поскольку не требует реализации протокола OAuth 2.0. Каждый пользователь может добавлять вебхуки "для себя" и REST API, вызываемый в рамках таких вебхуков, будет ограничиваться правами конкретного пользователя-владельца.
Вебхуки делятся на два вида:

1. Входящие – вебхуки обрабатывающие запрос пользователя и возвращающие результат выполнения запроса. Их предназначение — предоставление запрашиваемых данных.
2. Исходящие — вебхуки отвечающие за вставку и изменение данных.

Тиражные решения:
При разработке приложений в рамках тиражного решения для Битрикс24 приложения загружаются в каталог приложений и становятся доступными для установки неограниченному кругу пользователей и, более того, вы можете делать их платными, продавая их функционал по модели подписки.

Необходимость публикации в каталоге накладывает на приложения ряд дополнительных технических и организационных требований:

1. Необходимо стать технологическим партнером1С-Битрикс или быть участником стандартной партнерской программы;

2. Приложение, которое вы собираетесь опубликовать, должно будет пройти модерацию на соответствие требованиям к функционалу и оформлению;

3. У приложения всегда есть версии, каждая из которых может запрашивать свои права на доступ к модулям Битрикс24. В связи с этим, модерация запрашивается для каждой новой версии приложения, начиная с 1-й;

4. Автор решения несет ответственность за сохранность и конфиденциальность персональной информации, к которой получает доступ его приложение;

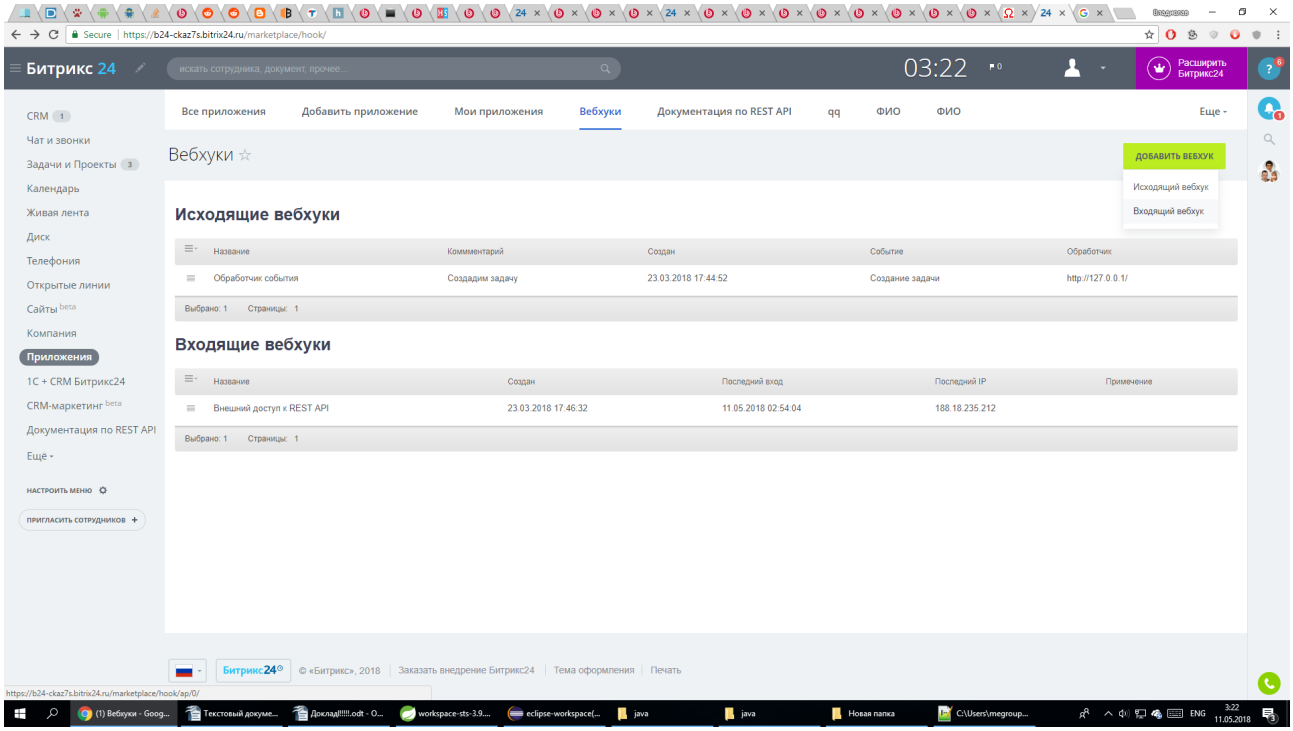
В отличие от разработки локальных приложений для конкретного проекта или заказчика, когда отношения между разработчиком и пользователем регулируются их двусторонними договоренностями, публикация тиражных решений в каталоге Приложений24 регулируется правилами Битрикс24, с которыми вам потребуется внимательно ознакомиться.

Рассмотрим процесс разработки пользовательского приложения с использованием вебхуков.

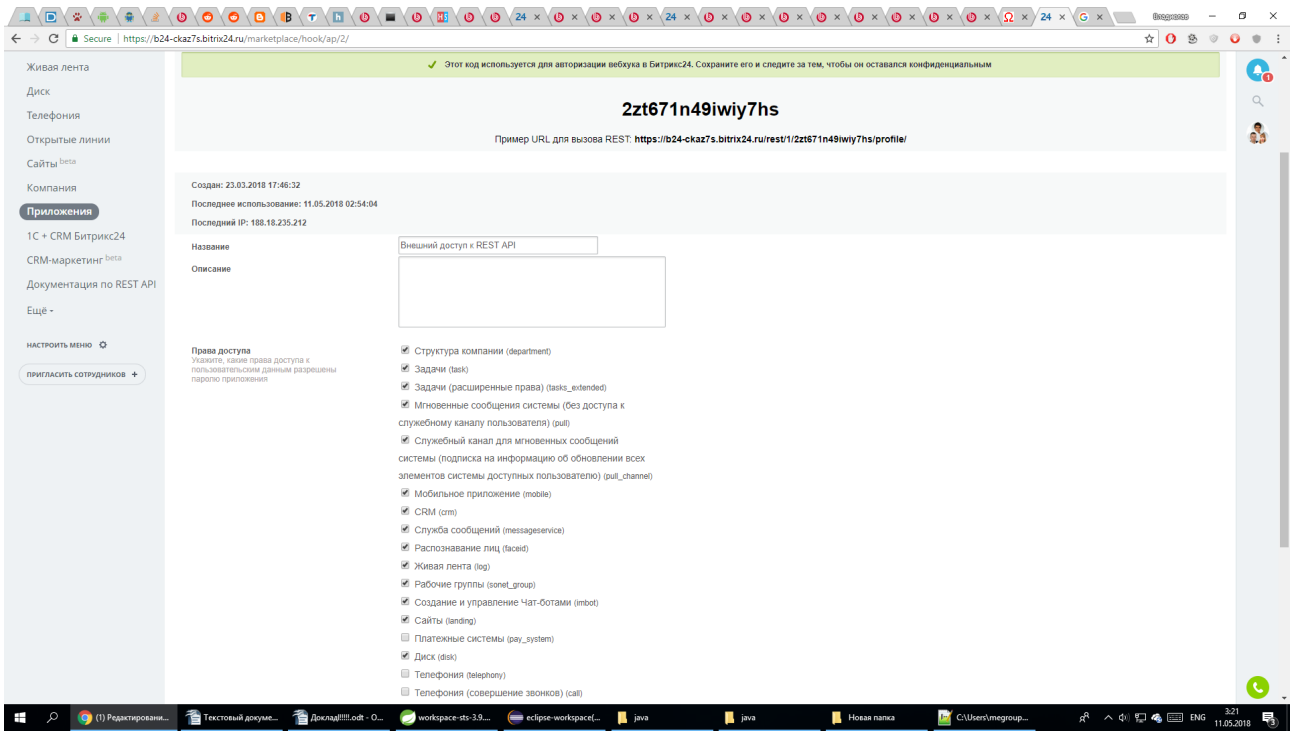
Разработаем Java приложение, которое будет получать информацию обо всех имеющихся на данный момент задачах.

В первую очередь создадим непосредственно сам вебхук:

Нажмём на кнопку добавить вебхук и выберем входящий вебхук.



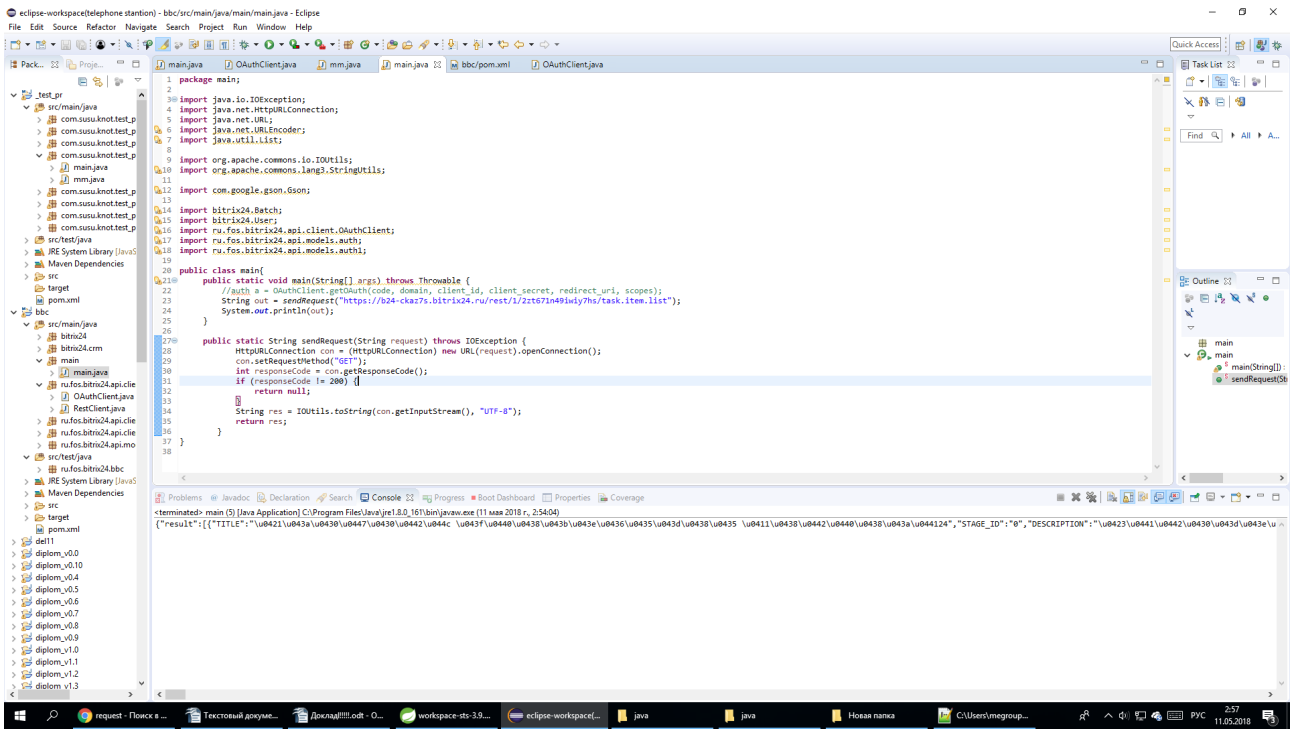
Укажем название создаваемого вебхука и отметим права доступа и нажмём кнопку создать, которая находится в нижней части страницы.



Далее напишем код программы на языке программирования Java.

```
package main;


import java.io.IOException;
import java.net.HttpURLConnection;
import java.net.URL;
import java.net.URLEncoder;
import java.util.List;

import org.apache.commons.io.IOUtils;
import org.apache.commons.lang3.StringUtils;

import com.google.gson.Gson;

import bitrix24.Batch;
import bitrix24.User;
import ru.fos.bitrix24.api.client.OAuthClient;
import ru.fos.bitrix24.api.models.auth;
import ru.fos.bitrix24.api.models.auth1;

public class main{
	public static void main(String[] args) throws Throwable {
		String out = sendRequest("https://b24-
ckaz7s.bitrix24.ru/rest/1/2zt671n49iwiy7hs/task.item.list");
		System.out.println(out);
	}

	public static String sendRequest(String request) throws IOException {
		HttpURLConnection con = (HttpURLConnection) new
URL(request).openConnection();
		con.setRequestMethod("GET");
		int responseCode = con.getResponseCode();
		if (responseCode != 200) {
		    return null;
		}
		String res = IOUtils.toString(con.getInputStream(), "UTF-8");
		return res;
	  }
}
```

Запустим приложение.

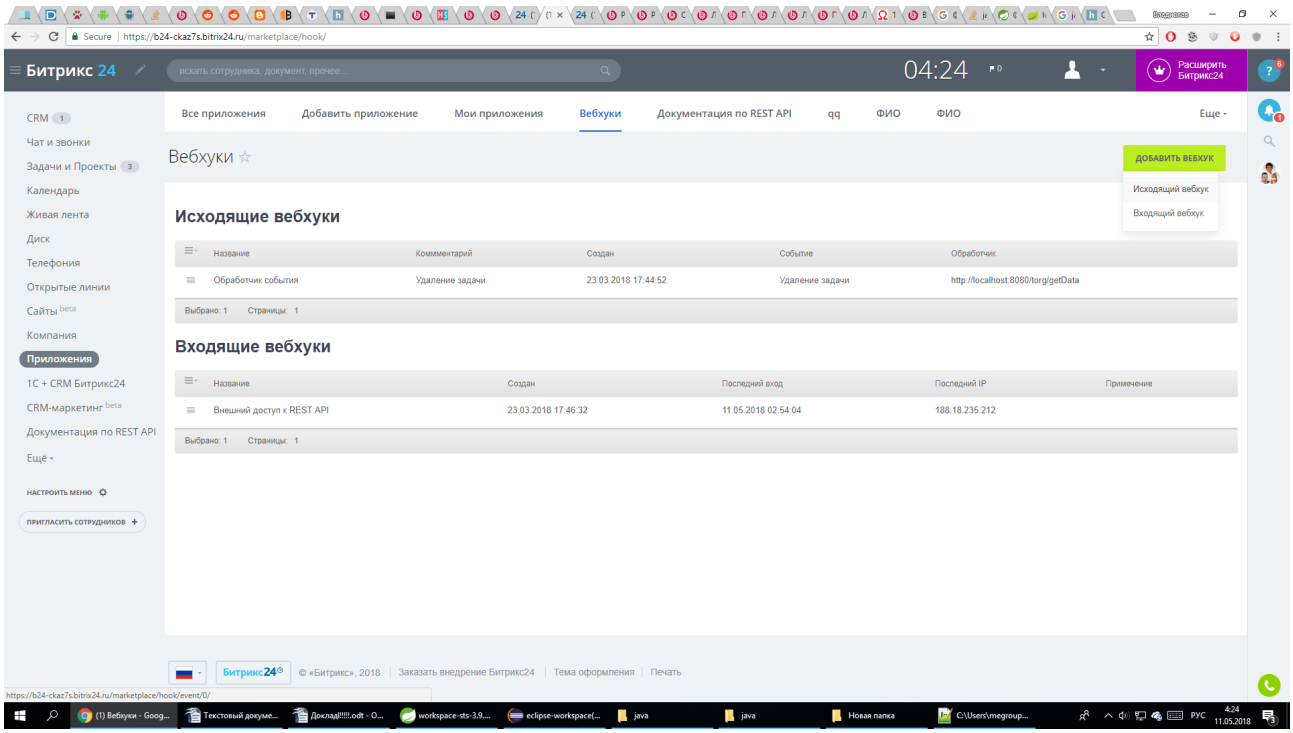


В ответ на запрос получили данные в формате json.

Далее рассмотрим исходящие вебхуки.
При возникновении определённого события исходящий вебхук инициализирует отправку данных обработчику расположенному на стороннем сервере.

Напишем сервер который будет получать сообщения от исходящего вебхука.
Писать код будем на языке программирования Java. Для работы с WEB используем фреймворк Spring.

Для начала создадим исходящий вебхук.
В выпадающем меню Добавить вебхук выберем Исходящий вебхук.



В открывшейся форме заполним поля:

Адрес обработчика - страница на стороннем ресурсе, куда будет обращаться вебхук.

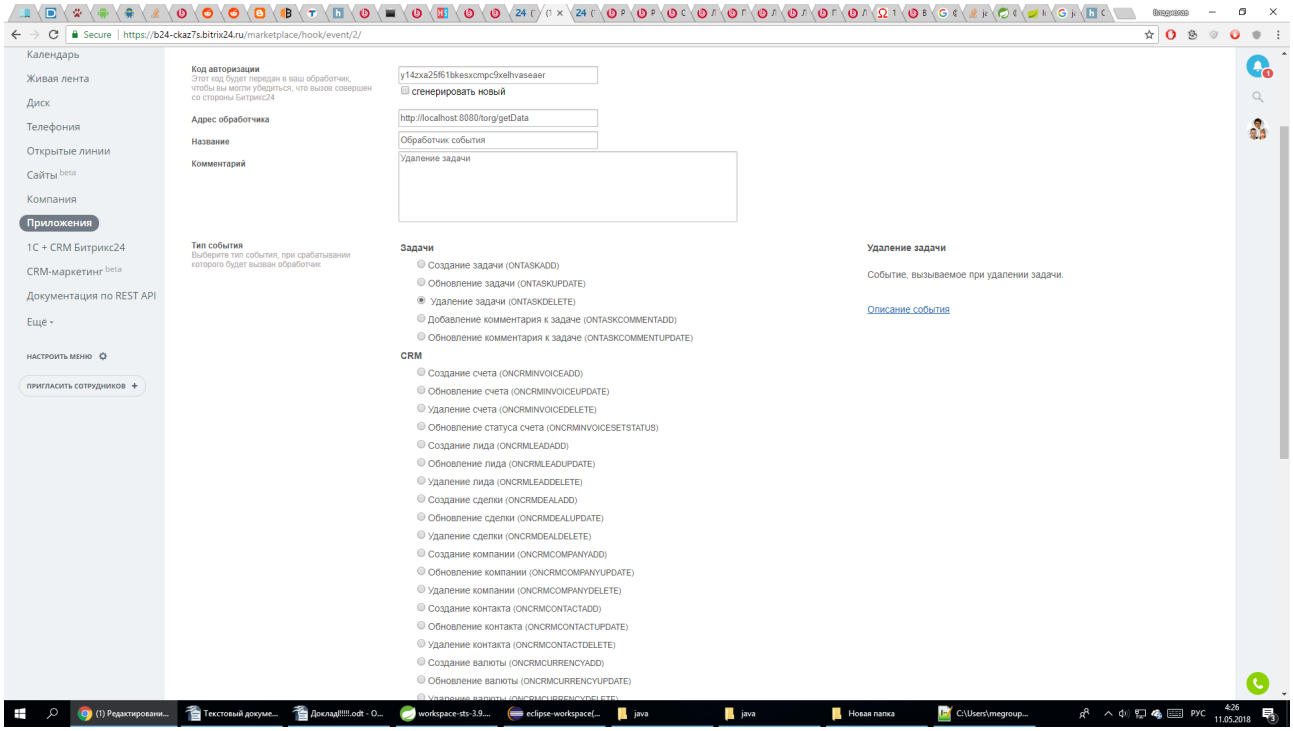
Название и Описание - произвольные данные.

Тип события - нужно указать событие, на которое будет инициализироваться вебхук.

В качестве адреса обработчика укажем: **http://localhost:8080/torg/getData.**

Зададим название: Обработчик события.

Выберем тип события: Удаление задачи (ONTASKDELETE).



После сохранения вебхука будет выведен Код авторизации в виде строки из случайных знаков. Этот код позволит внутри обработчика проверить, действительно ли обработчик вызван

вашим Битрикс*24*. Для изменения кода авторизации достаточно перейти в режим редактирования вебхука и отметить пункт сгенерировать новый. После сохранения изменений будет выдан новый код авторизации.

Далее напишем код сервера:

```
package hello;

import java.util.ArrayList;
import java.util.Enumeration;
import java.util.Iterator;
import java.util.List;

import javax.servlet.http.HttpServletRequest;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
@RequestMapping(path="/torg")
public class MainController {
	@GetMapping(path="/getData")
	public @ResponseBody String tovarInfo1(HttpServletRequest request) {
		//Массив значений параметров
		ArrayList<String> list = new ArrayList<String>();
		//Получим имена параметров
		Enumeration<String> v = request.getParameterNames();
		//Получим значения параметров и сохраним в list
		for (Enumeration<String> e = v; e.hasMoreElements();)
		{
			list.add(request.getParameter(e.nextElement()));
		}
		return "success";
	}
}
```

Разрабатываемые приложения требующие регестрации на Bitrix24 делятся на два типа:

1. **Статичные приложения на основе HTML/JS**. Фактически, на основе этих технологий вы можете реализовывать приложения-одностраничники, обращаясь к REST API Битрикс24 при помощи SDK на JS. Приложение будет представлено в интерфейсе Битрикс24 в виде отдельной страницы (со ссылкой из левого меню). Статичные приложения не могут получать события Битрикс24. Обычно загружаются в виде архива, который содержит в себе весь необходимый html, стили, javascript, картинки. Точкой входа такого приложения считается файл **index.html**. Инсталлятором - **install.html**, при его наличии.

2. **Серверные приложения на любых подходящих языках программирования** (PHP, Python, и другие). Они могут обращаться к REST API используя протокол OAuth 2.0, а также получать события от Битрикс24 в свои обработчики. Серверные приложения могут быть представлены в интерфейсе Битрикс24 в виде отдельной страницы, а также в виде встраиваемых popup-диалогов в доступных для встраивания объектах Битрикс24. Есть вариант, когда приложение никак не проявляется себя в интерфейсе Битрикс24, но использует REST API для обмена данными или какой-то автоматической обработки данных Битрикс24. Такие приложения размещаются на сторонних серверах. При добавлении такого приложения в качестве тиражного в личном кабинете или в качестве локального на конкретном Битрикс24, указываются прямые ссылки на точку входа и инсталлятор этого приложения, которые будут открыты в интерфейсе Битрикс24.

Статичные приложения:

При написании внутренних приложений используются технологии такие как: HTML, JavaScript, CSS. Есть возможность использования различных JavaScript-фреймворков, например, jQuery. Так же доступны различные удалённые библиотеки такие как: Google Charts, ChartJS, Highcharts JS, Flot, Bounce.js и многие другие.

Средства такие как PHP, mySQL и cron не могут быть использованы при разработке приложения.

Рассмотрим пример создания статичного приложения:

Приложение предоставляет отчёт о текущих задачах в виде таблицы.

В отчёте содержится информация о следующих полях:

1. ID – порядковый номер задачи.
2. TITLE — название задачи.
3. PRIORITY — приоритет задачи.
4. STATUS — статус задачи.

Разработку начнём с написания кода приложения.

Приложения будет состоять из двух файлов Index.html и GetTask.html.

В файле Index.html реализуем главное меню программы.

В файле GetTask.html реализуем основной функционал.

Index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Quick start. Local static application</title>
</head>
<body>
  <script src="//api.bitrix24.com/api/v1/"></script>
  <a href="getTask.html">getTask</a><br/>
</body>
</html>
```

GetTask.html

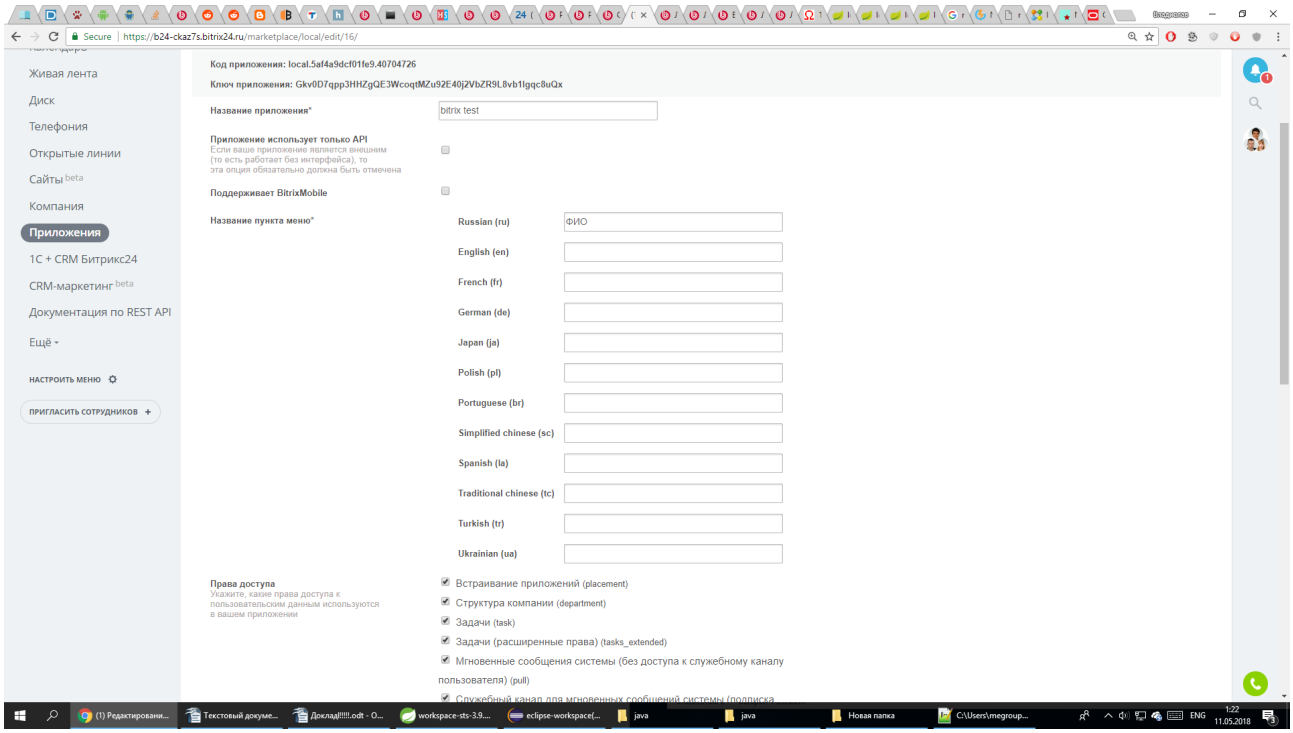
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Quick start. Local static application</title>
</head>
<body>
    <div id="task"></div>
    <script src="//api.bitrix24.com/api/v1/"></script>
    <script>
        // Make a call to REST when JS SDK is loaded
                BX24.init(function(){
                        BX24.callMethod('task.item.list',
                                [{ID : 'desc'},// Сортировка по ID  по убыванию.
```

```
                                {}],
            function(result){
                var name = document.getElementById('task');
                var dat =result.data()
                var txt = '<table border="1"><caption>Отчёт по текущим
задачам</caption><tr><th>id</th><th>Наименование</th><th>Приоритет</th><t
h>Статус</th></tr>';
                for (var i = 0; i < dat.length; i++) {
                    var task = dat[i];
                    txt += '<tr><td>'+ task.ID +'</td><td>'+ task.TITLE
+'</td><td>'+ task.PRIORITY  +'</td><td>'+ task.STATUS +'</td></tr>';
                }
                name.innerHTML = txt + '</table>';
                console.info(result.data());
                console.log(result);
            }); });
    </script>
</body>
</html>
```

Основной код написан.

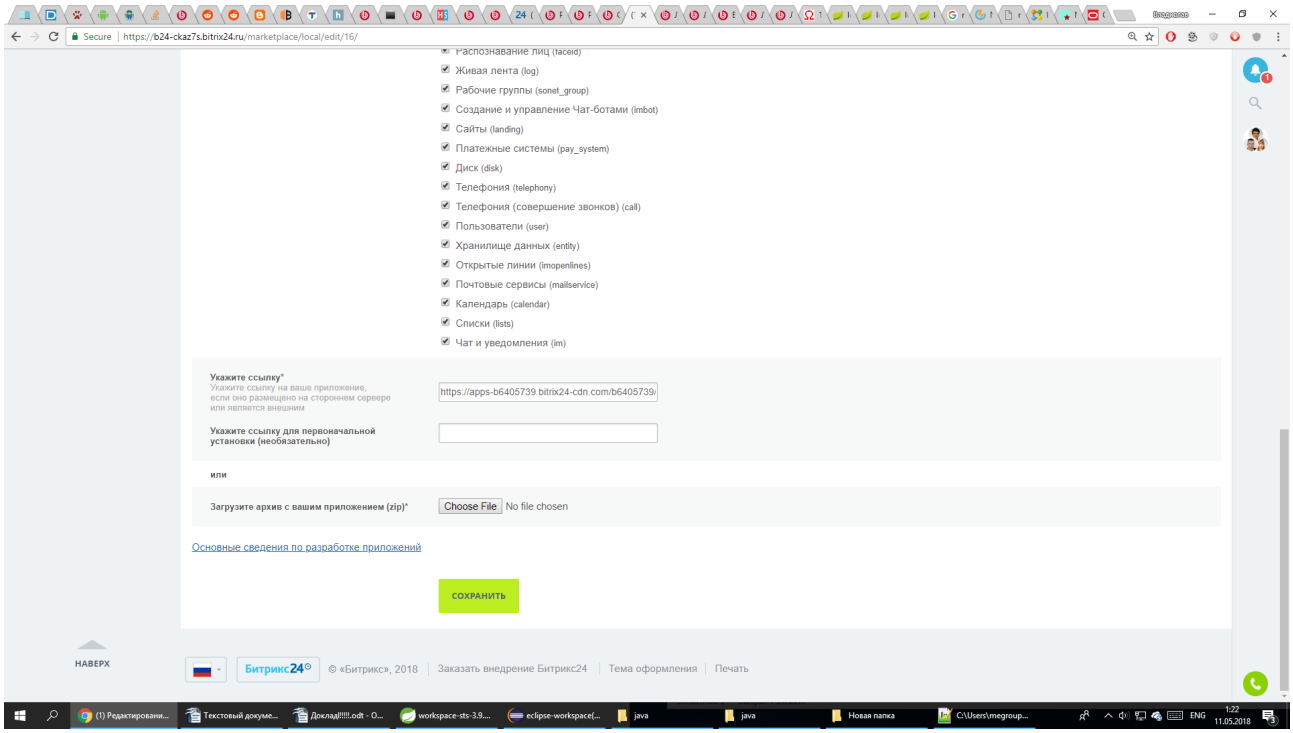
Далее перейдём к добавлению приложения.

В открывшейся форме заполним базовые поля и укажем необходимые для приложения права.



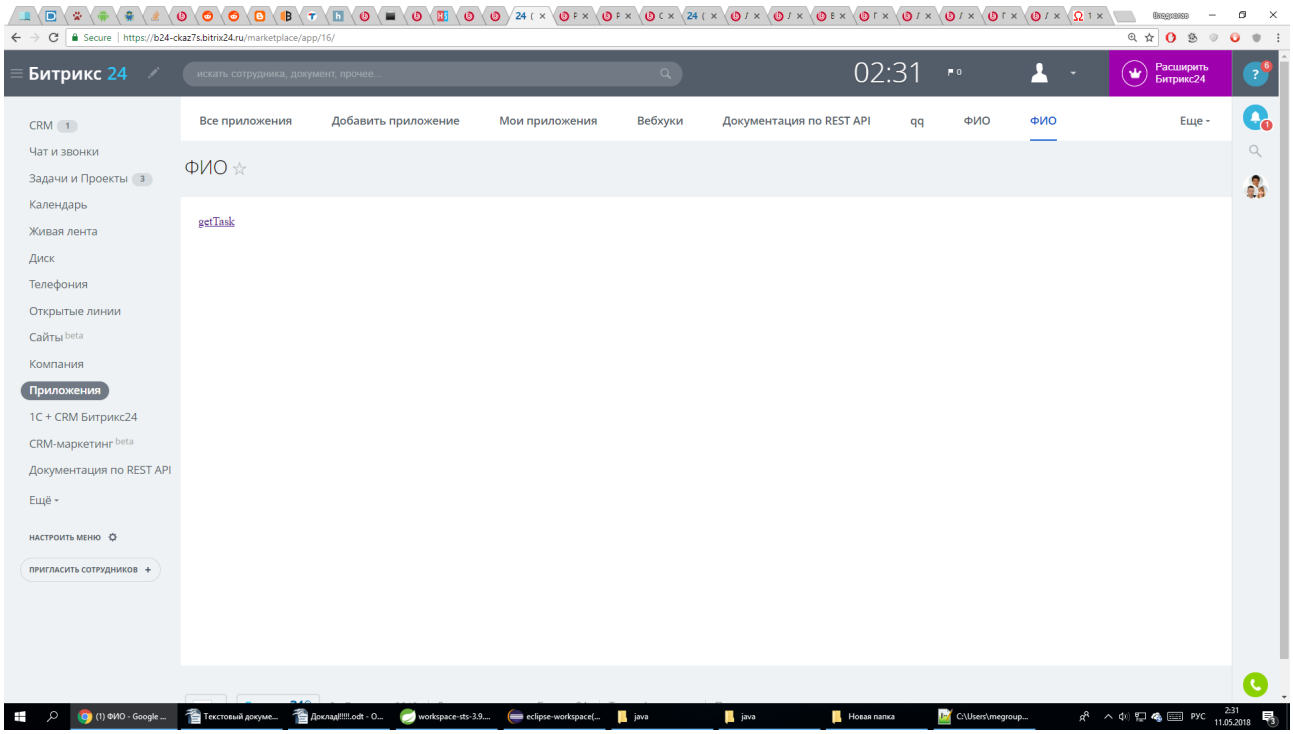
Создадим архив с приложением.

Для создания архива требуется собрать все файлы проекта в одном каталоге и затем архивировать его. Перед архивацией убедитесь в наличии файла **index.html,** который является точкой входа в приложение.

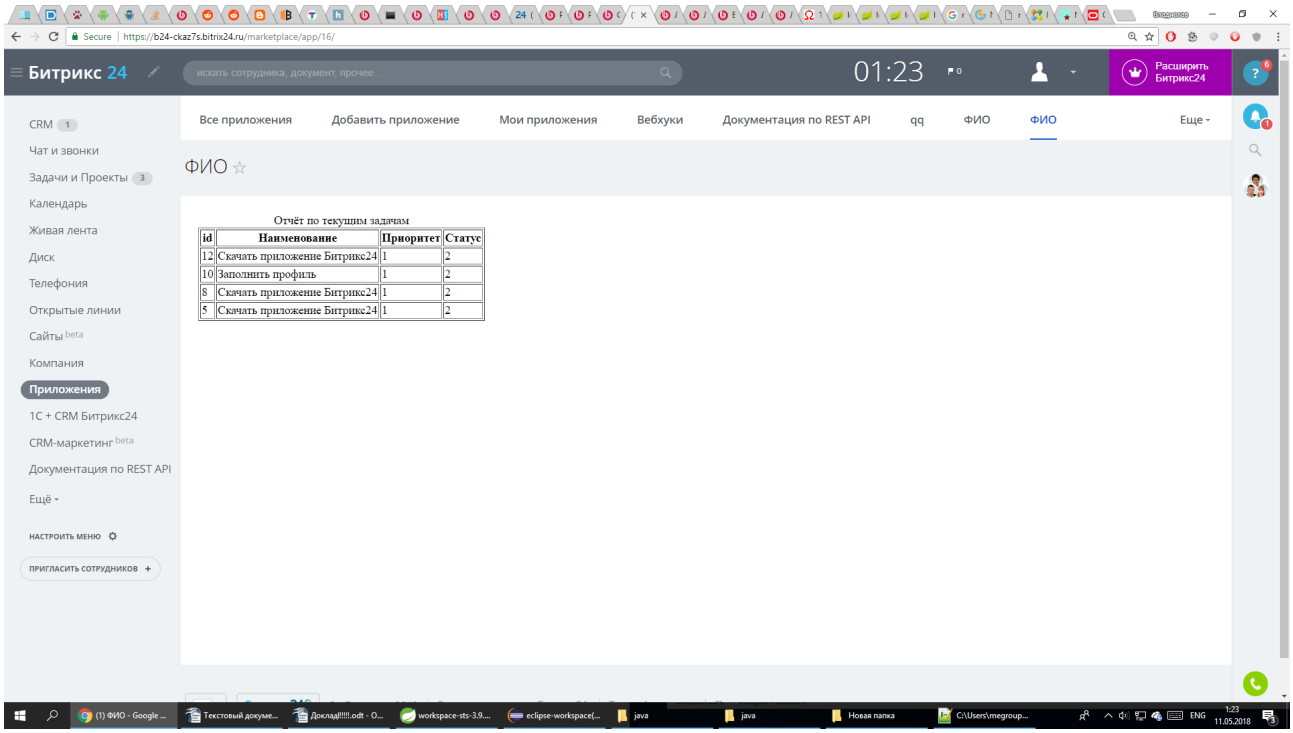


Загрузим созданный архив на портал.

Запустим наше приложение и убедимся в его работоспособности.



Перейдём по ссылке.



Запрошенные данные были получены и отображены в виде таблицы.

Серверные приложения:

Приложения данного типа — это как правило скрипты, располагающиеся на сервере разработчика. При реализации таких приложений разработчик может использовать все доступные ему инструменты (библиотеки, базы данных, специальные сервисы, и.т.д.) для решения поставленых задач.

Серверные приложения условно делятся на два подтипа:

1. Работающие внутри интерфейса «Битрикс24».
2. Работающие во внешней среде.

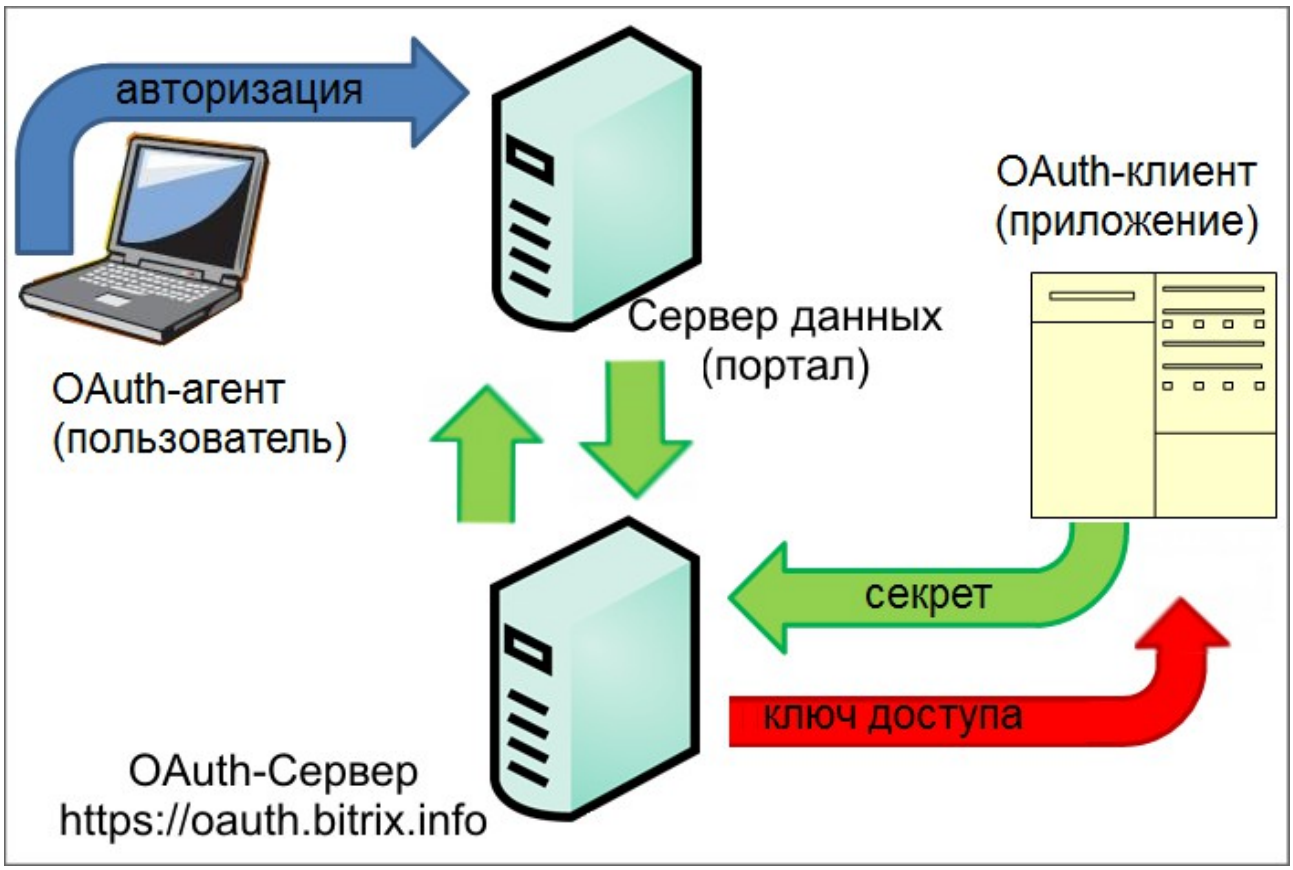
С точки зрения реализации внутренней логики принципиальной разницы между этими подтипами нет. Они отличаются способом подключения к «Битрикс24» и продления доступа к данным.

Приложения, работающие внутри интерфейса подключаются посредством установки . Для работы с данными таким приложениям требуется периодически обновлять ключ доступа, что осуществляется автоматически.

Приложения работающие во внешнейво время подключения проходят процесс авторизации, который осуществляется по протоколу Oauth.

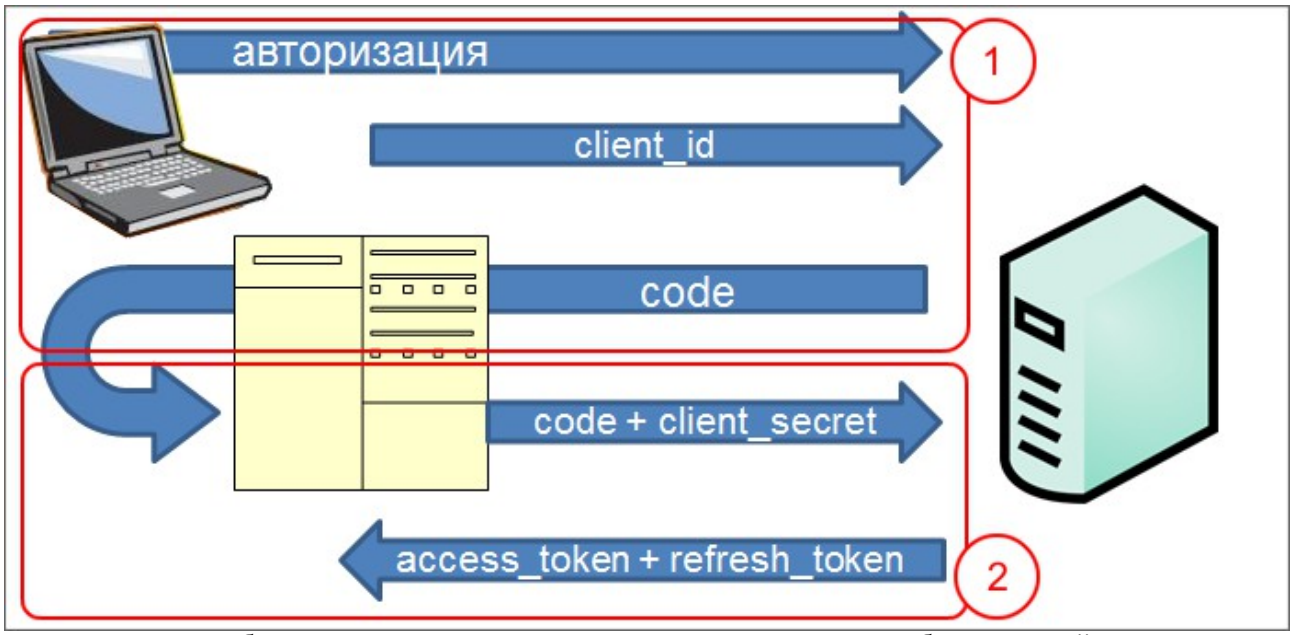
Oauth – это открытый протокол авторизации, который позволяет предоставить третьей стороне ограниченный доступ к защищенным ресурсам пользователя без необходимости передавать ей (третьей стороне) логин и пароль.

Как работает протокол **OAuth**



Для сервера авторизация - это указание на то, что пользователь дал доступ приложению, приложение предоставляет свой секрет. Портал это всё объединяет и выдаёт приложению соответствующий тип доступа.

Протокол состоит из двух шагов:



Пользователь сообщает порталу, что он авторизован. Приложение добавляет свой идентификатор: **client id**. В ответ сервер передаёт пользователю, а через него и приложению, первый авторизационный код: **code**.

•Этот код приложение, скрыто от пользователя, передаёт обратно порталу, присоединяя к нему свой секретный ключ: **client secret**. Приложение таким образом подтверждает, что оно то самое приложение, которое известно порталу и которое может с ним работать. В ответ портал сообщает два параметра: `access_token` - параметр, который собственно требуется для доступа к авторизации и `refresh_token`- токен, который требуется для продления авторизации.

•После использования `refresh_token` и выданный вместе с ним `access_token` становятся недействительными. Для доступа к REST API следует использовать новый полученный `access_token`, а для продления доступа - новый `refresh_token`.

Общий порядок работы с **OAuth**

•добавляется и устанавливается тиражное приложение, либо локальное в своем отдельном Битрикс24;

•запрашиваются с удаленного сервера ключи;

•сервер перенаправляет браузер на зарегистрированный приложением URL;

•обрабатывается ответ;

•подписываются полученным ключом все запросы к Rest API.

В качестве сервера данных и держателя пользовательской авторизации выступает конкретный Битрикс24. В качестве держателя авторизации приложения служит сервер авторизации, доступный по адресу https://oauth.bitrix.info/.

Полный сценарий **OAuth-**авторизации проходит в несколько шагов**:**

Этап **1**

Отправим запрос на получение параметра code нашему битриксу.

https://portal.bitrix24.com/oauth/authorize/?
client_id=app.573ad8a0346747.09223434&state=JJHgsdgfkdaslg7lbadsfg

Параметры:

•**client_id** - код приложения, получаемый в партнерском кабинете при регистрации приложения (и действующий для любых Битрикс24), либо получаемый в конкретном Битрикс24 в случае локального приложения (будет действовать только для этого Битрикс24). Обязательный параметр.

•Portal – имя нашего битрикса.

•**state** - дополнительный параметр, позволяющий приложению передать произвольные дополнительные данные между шагами авторизации. Необязательный параметр.

По данной ссылке пользователю будет выведена форма авторизации. После авторизации (либо при наличии авторизованной сессии), если приложение с переданным **client_id** установлено на портале, портал вернет пользователя на **redirect_uri** приложения. Если же приложение на портал не установлено, пользователю будет выведено соответствующее сообщение об ошибке.

Для бхода формы авторизации перед отправкой запроса на получение параметра code можнго выполнить запрос следующего вида:

```
https://b24-ckaz7s.bitrix24.ru/crm/start/?
AUTH_FORM=Y&TYPE=AUTH&USER_LOGIN=***&USER_PASSWORD=***2&USER_REMEMBER=Y
```

Итогом успешной пользовательской авторизации должен стать возврат пользователя на зарегистрированный адрес приложения с дополнительными параметрами:

```
https://www.applicationhost.com/application/?
   code=avmocpghblyi01m3h42bljvqtyd19sw1
   &state=JJHgsdgfkdaslg7lbadsfg
   &domain=portal.bitrix24.com
   &member_id=a223c6b3710f85df22e9377d6c4f7553
   &scope=crm%2Centity%2Cim%2Ctask
   &server_domain=oauth.bitrix.info
```

Параметры:

- **code** - первый авторизационный код, см. ниже;
- **state** - значение переданное в первом запросе;
- **domain** - домен портала, на котором происходит авторизация;
- **member_id** - уникальный идентификатор портала, на котором происходит авторизация;
- **scope** - разделенный запятыми список прав доступа к REST API, которые портал предоставляет приложению;
- **server_domain** - домен сервера авторизации.

Этап 2

Получив тем или иным способом первый авторизационный код **code** приложение должно совершить второй шаг OAuth-авторизации и сделать скрытый от пользователя запрос вида:

```
https://oauth.bitrix.info/oauth/token/?
   grant_type=authorization_code
   &client_id=app.573ad8a0346747.09223434
   &client_secret=LJSl0lNB76B5YY6u0YVQ3AW0DrVADcRTwVr4y99PXU1BWQybWK
   &code=avmocpghblyi01m3h42bljvqtyd19sw1
```

Параметры (все - обязательны):

- **grant_type** - параметр, показывающий тип авторизационных данных, подлежащих валидации. Должен иметь значение *authorization_code*;
- **client_id** - код приложения, получаемый в партнерском кабинете при регистрации приложения либо на портале в случае локального приложения;
- **client_secret** - секретный ключ приложения, получаемый в партнерском кабинете при регистрации приложения либо на портале в случае локального приложения;
- **code** - значение параметра code, переданного приложению в конце предыдущего шага.

В ответ на такой запрос приложение получит *json* следующего содержания:

```
GET /oauth/token/
```

```
HTTP/1.1 200 OK

Content-Type: application/json

{
    "access_token": "s1morf609228iwyjjpvfv6wsvuja4p8u",
    "client_endpoint": "https://portal.bitrix24.com/rest/",
    "domain": "oauth.bitrix.info",
    "expires_in": 3600,
    "member_id": "a223c6b3710f85df22e9377d6c4f7553",
```

```
    "refresh_token": "4f9k4jpmg13usmybzuqknt2v9fh0q6rl",

    "scope": "app",

    "server_endpoint": "https://oauth.bitrix.info/rest/",

    "status": "T"

}
```

Значимые параметры:

- **access_token** - основной авторизационный токен, требуемый для доступа к REST API;
- **refresh_token** - дополнительный авторизационный токен, служащий для продления сохраненной авторизации;
- **client_endpoint** - адрес REST-интерфейса портала;
- **server_endpoint** - адрес REST-интерфейса сервера;
- **status** - статус приложения на портале.

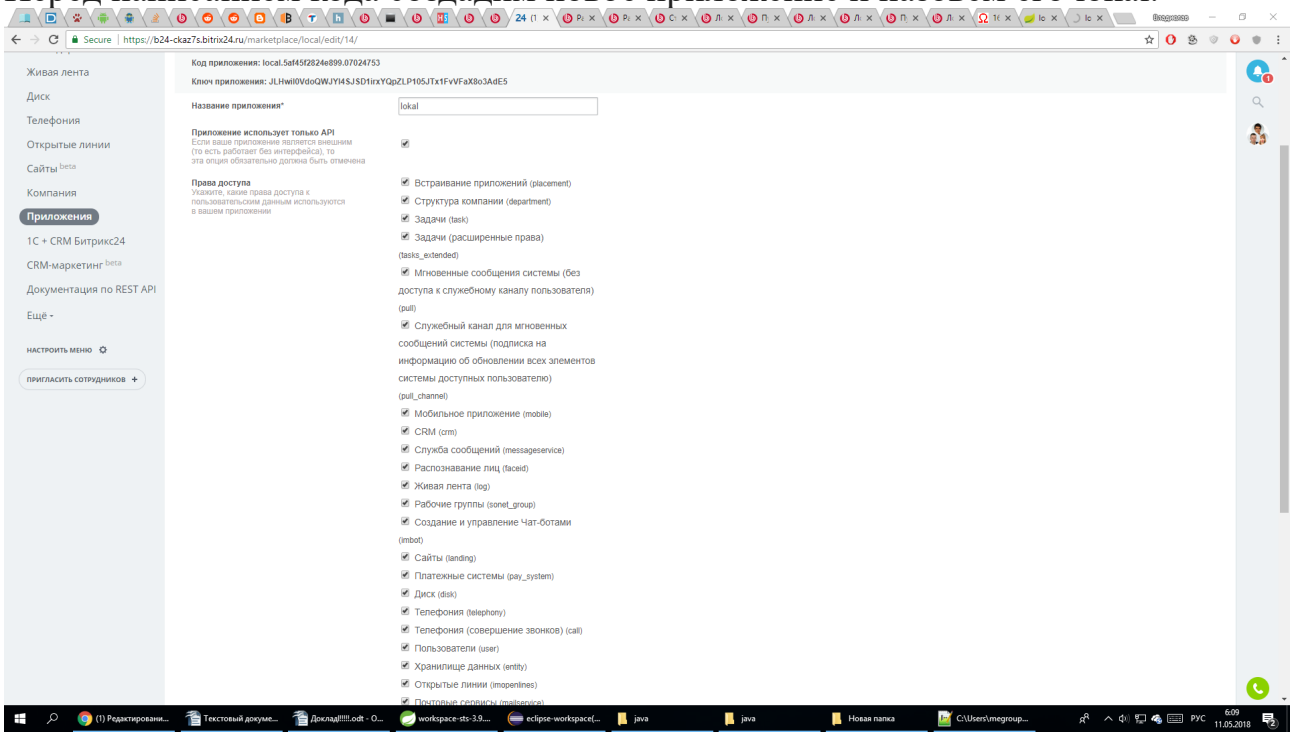
Также, на этом этапе приложение может получить ошибку авторизации, если, например, истек пробный или оплаченный период.

```
{

    "error": "PAYMENT_REQUIRED",

    "error_description": "Payment required"

}
```

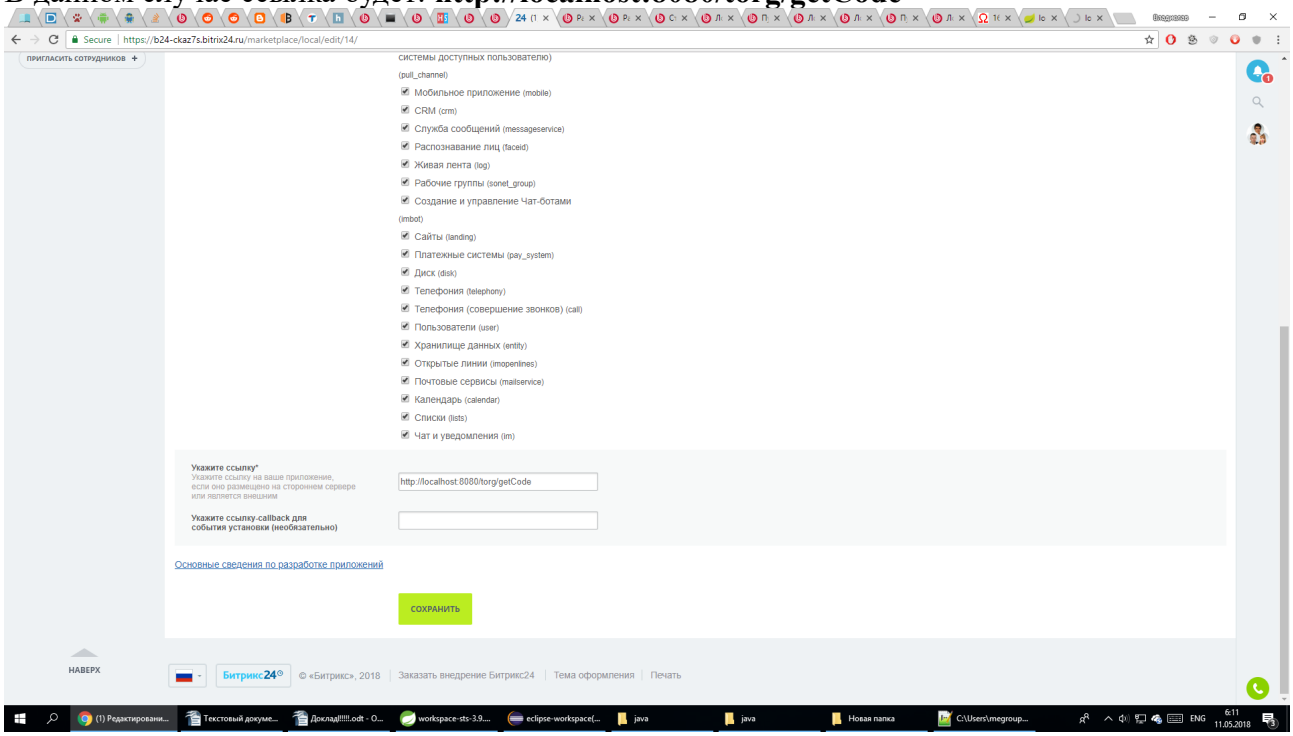
Напишем небольшое приложение использующее рассмотренный тип авторизации.

Писать код будем на языке программирования Java. Для работы с WEB используем фреймворк Spring.

Перед написанием кода создадим новое приложение и назовём его lokal.



Далее укажем ссылку на наше приложение.
В данном случае ссылка будет: **http://localhost:8080/torg/getCode**



Далее напишем код код сервера:

```
package hello;

import java.io.IOException;
```

```
import java.util.ArrayList;
import java.util.Enumeration;
import java.util.Iterator;
import java.util.List;

import javax.servlet.http.HttpServletRequest;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;

import com.susu.knot.test_pr.ru.fos.bitrix24.api.client.OAuthClient;
import com.susu.knot.test_pr.ru.fos.bitrix24.api.client.RestClient;
import com.susu.knot.test_pr.ru.fos.bitrix24.api.models.AppInfo;
import com.susu.knot.test_pr.ru.fos.bitrix24.api.models.Task;
import com.susu.knot.test_pr.ru.fos.bitrix24.api.models.auth;

@Controller
@RequestMapping(path="/torg")
public class MainController {
	@GetMapping(path="/getCode")
	public @ResponseBody String getData(@RequestParam(value="code") String code)
throws IOException {
		System.out.println("Code: "+code);
		//Авторизуемся и получим токены
		auth a = OAuthClient.getOAuth4(code, "oauth.bitrix.info",
"local.5af45f2824e899.07024753", "JLHwiI0VdoQWJYl4SJSD1irxYQpZLP105JTx1FvVFaX8o3AdE5");
		System.out.println("refresh_token:  "+a.access_token+"
refresh_token:"+a.refresh_token);
		//Запросим информацию о нашем приложении
		String str = RestClient.appInfoStr(a);
		//Выведем в консоль ответ
		System.out.println(str);
		return "success";
	}
}
```

Исходный код класса **OauthClient. (Данный класс ответственен за процесс авторизации и обновления токенов)**

```
package com.susu.knot.test_pr.ru.fos.bitrix24.api.client;

import com.google.gson.*;
import org.apache.commons.io.IOUtils;
import org.apache.commons.lang3.StringUtils;
import com.susu.knot.test_pr.ru.fos.bitrix24.api.models.auth;
import com.susu.knot.test_pr.ru.fos.bitrix24.api.models.auth1;

import java.io.IOException;
import java.io.UnsupportedEncodingException;
import java.lang.reflect.Type;
import java.net.HttpURLConnection;
import java.net.URL;
import java.net.URLEncoder;
import java.net.UnknownHostException;
import java.util.List;
```

```
public class OAuthClient {
       public static auth getOAuth4(String code, String domain, String client_id, String
client_secret) throws IOException {
        HttpURLConnection con = (HttpURLConnection) new URL("https://" + domain +
"/oauth/token/"
                + "?client_id=" + client_id
                + "&grant_type=authorization_code"
                + "&client_secret=" + client_secret
                + "&code=" + code).openConnection();
        con.setRequestMethod("GET");

        int responseCode = con.getResponseCode();
        if (responseCode != 200) {
            return null;
        }
        String res = IOUtils.toString(con.getInputStream(), "UTF-8");
        return new Gson().fromJson(res, auth1.class).toAuth();
    }



    public static String getOAuthRequestString(String domain, String client_id, String
redirect_uri) throws UnsupportedEncodingException {
        return "https://" + domain + "/oauth/authorize/?"
                + "response_type=code"
                + "&client_id=" + client_id
                + "&redirect_uri=" + URLEncoder.encode(redirect_uri, "UTF-8");
    }

    public static auth getOAuth(String code, String domain, String client_id, String
client_secret, String redirect_uri, List<String> scopes) throws IOException {
        HttpURLConnection con = (HttpURLConnection) new URL("https://" + domain +
"/oauth/token/"
                + "?client_id=" + client_id
                + "&grant_type=authorization_code"
                + "&client_secret=" + client_secret
                + "&redirect_uri=" + URLEncoder.encode(redirect_uri, "UTF-8")
                + "&code=" + code
                + "&scope=" + StringUtils.join(scopes, ",")).openConnection();
        con.setRequestMethod("GET");

        int responseCode = con.getResponseCode();
        if (responseCode != 200) {
            return null;
        }
        String res = IOUtils.toString(con.getInputStream(), "UTF-8");
        return new Gson().fromJson(res, auth1.class).toAuth();
    }

    public static auth refreshOAuth(auth oauth, String client_id, String client_secret,
String redirect_uri) throws IOException {
        if (oauth == null) {
            return null;
        }
        HttpURLConnection con = (HttpURLConnection) new URL("https://" + oauth.domain +
"/rest/app.info.json"
                + "?auth=" + oauth.access_token).openConnection();
        con.setRequestMethod("GET");

        try {
```

```
            int responseCode = con.getResponseCode();
            if (responseCode == 200) {
                return oauth;
            }
        } catch (UnknownHostException e) {
            return null;
        }
        con = (HttpURLConnection) new URL("https://oauth.bitrix.info/oauth/token/"
                + "?client_id=" + client_id
                + "&grant_type=refresh_token"
                + "&client_secret=" + client_secret
                + "&redirect_uri=" + redirect_uri
                + "&refresh_token=" + oauth.refresh_token).openConnection();
        con.setRequestMethod("GET");

        int responseCode = con.getResponseCode();
        if (responseCode != 200) {
            return null;
        }
        String response = IOUtils.toString(con.getInputStream(), "UTF-8");
        return new GsonBuilder().registerTypeAdapter(String[].class, new
JsonDeserializer<String[]>() {

            public String[] deserialize(JsonElement json, Type typeOfT,
                                        JsonDeserializationContext context) throws
JsonParseException {
                return json.getAsString().split(",");
            }
        }).create().fromJson(response, auth.class);
    }
}
```

Исходный код класса auth (Данный класс отвечает за хранение токенов)

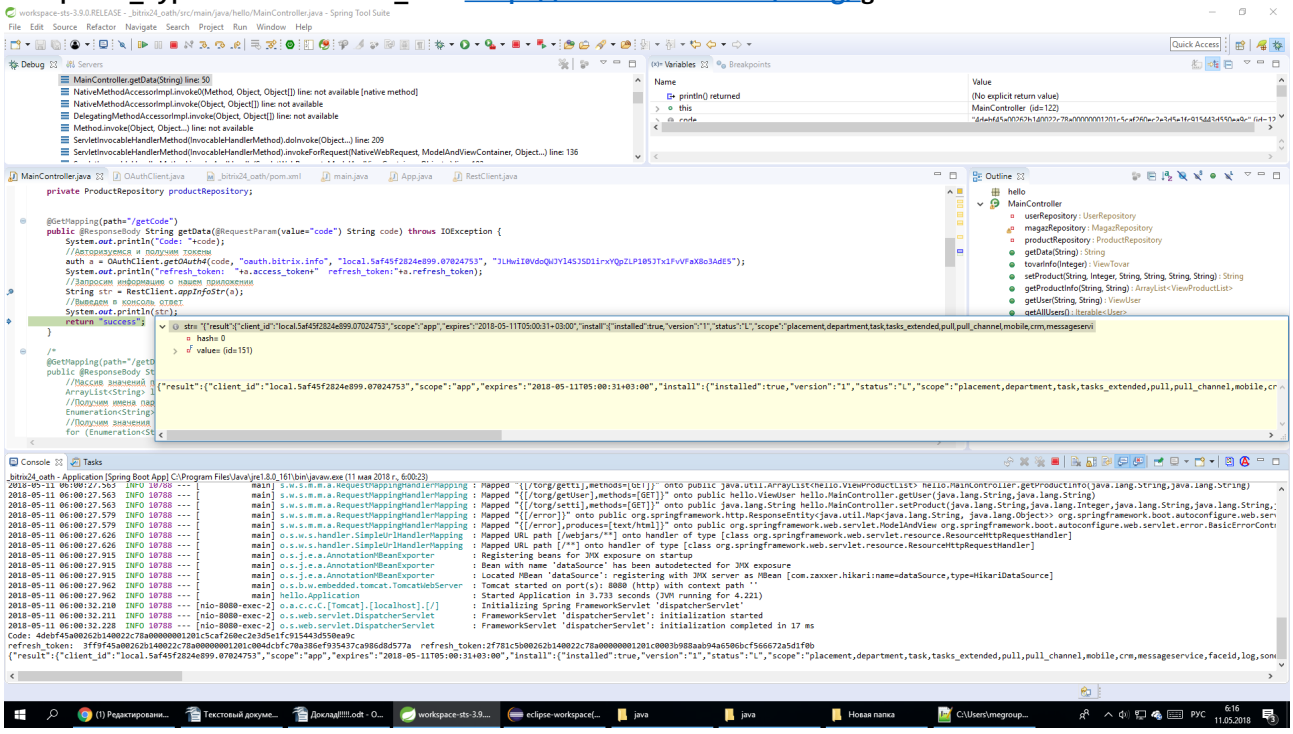
```
package com.susu.knot.test_pr.ru.fos.bitrix24.api.models;

import org.springframework.data.annotation.Id;

public class auth {

    @Id
    public String domain;

    public String member_id;

    public String access_token;

    public long expires_in;

    public String[] scope;

    public String refresh_token;
}
```

Исходный код класса auth1 (Данный класс выступает в качестве посредния при приведении получаемых данных формата json по завершении второго этапа авторизации к типу auth)

```
package com.susu.knot.test_pr.ru.fos.bitrix24.api.models;

import org.springframework.data.annotation.Id;

public class auth {
```

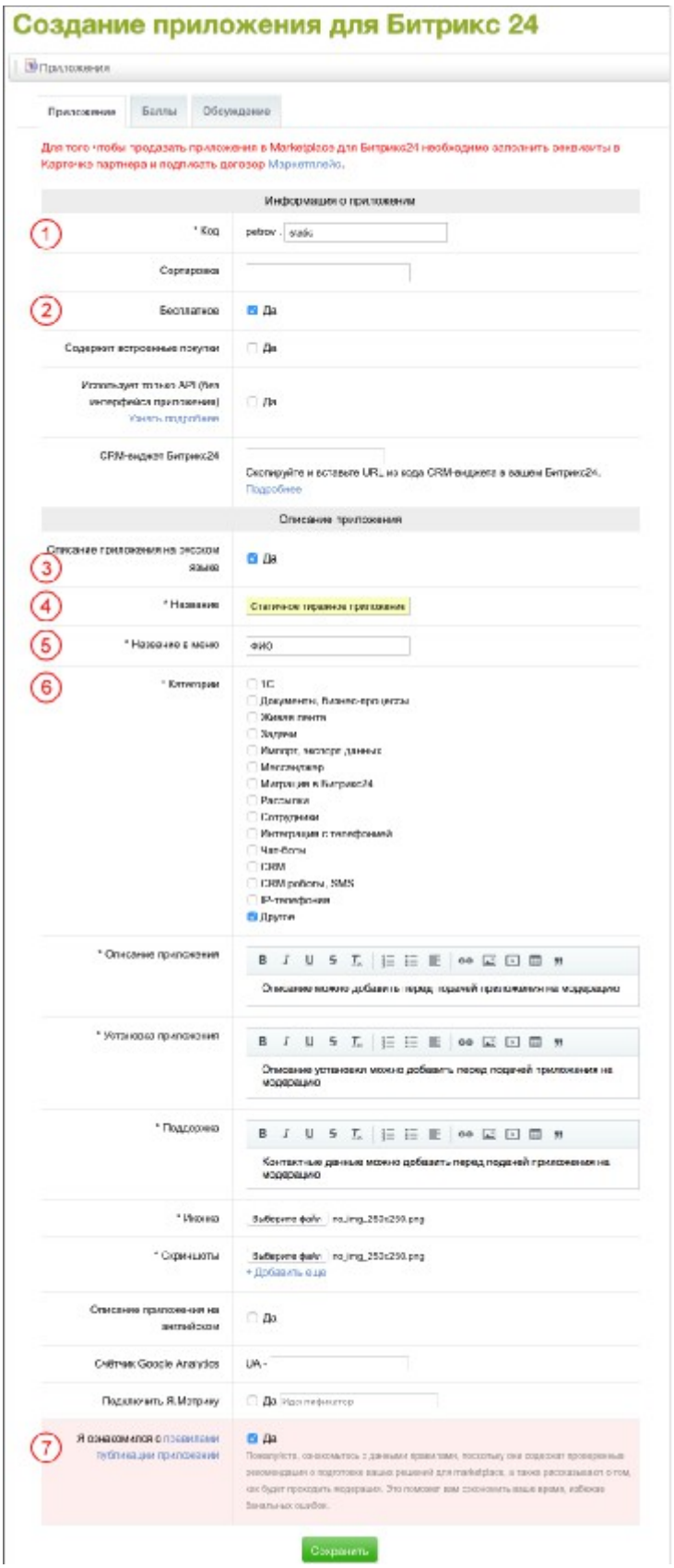
```
    @Id
    public String domain;

    public String member_id;

    public String access_token;

    public long expires_in;

    public String[] scope;

    public String refresh_token;
}
```

Запустим сервер в режиме отладки и убедимся что оба этапа авторизации прошли успешно и был получен ответ на наш запрос.
Для проверки работоспособности перейдём по ссылке http://b24-ckaz7s.bitrix24.ru/oauth/authorize/?client_id=local.5af45f2824e899.07024753 &response_type=code&redirect_uri=http://localhost:8080/torg/getCode.



Публикация Битрикс приложений:

Для создания решения, которое в дальнейшем может быть опубликовано в каталоге Приложения24 необходимо в партнерском кабинете перейти по цепочке Кабинет партнера > Приложения*24* для Битрикс*24* > Добавить приложение. В открывшейся форме необходимо заполнить обязательные поля:



В отличие от локальных приложений, у каждого тиражного решения необходимо задать уникальный символьный код. Код состоит из двух частей: вашего уникального символьного кода партнера (указывается в партнерской карточке) и части, которую вы присваиваете конкретному создаваемому приложению. Именно этот символьный код в дальнейшем будет присутствовать в публичном адресе решения в каталоге Приложения и заменить его в дальнейшем уже не получится.

Если решение платное, то требуется указать один из режимов работы платного решения:

- Демо - если оплаченный пользователем период использования закончен, то решение не отключается, остается работоспособным, но внутри интерфейса вы сможете отключать какой-то функционал и выводить пользователю напоминание о покупке. Для этого вам пригодится функция app.info, возвращающая статус вашего решения, установленного на конкретном Битрикс24.
- Триал позволяет не заниматься отслеживанием статуса решения. Как только на конкретном Битрикс24 оплаченный пользователем срок закончится, Битрикс24 заблокирует пользовательский функционал и предложит пользователю оплатить продление.
- Нет пробной версии мы рекомендуем в крайнем случае. Он означает, что пользователь должен сначала оплатить использование решения, а только потом сможет его установить на свой Битрикс24. Такой сценарий монетизируется достаточно сложно.

Можно сразу привязать свое решение к нужной категории решений (6). Эти категории служат для навигации пользователей по каталогу решений.

Сохраните карточку приложения, заполнив все необходимые поля. Первый шаг сделан - фактически, вы подготовили описательную часть вашего будущего приложения. Осталось загрузить техническую реализацию.