API для интеграции с Почтовой кассой

Описание API

Команды API

Интеграция с личным кабинетом Почтовой кассы осуществляется с использованием `REST API`, описание команд которого доступно по ссылке **https://app.swaggerhub.com/apis-docs/postkassa/account/1.0.0**.

Адрес сервера для обращений по API: **https://account.postkassa.ru/api/v1**

Полный адрес команды формируется из адреса сервера + пути команды, указанной в документации. Например, для авторизации на сервере необходимо обращаться по адресу **https://account.postkassa.ru/api/v1/login-auto**

Авторизация на сервере

Для целей интеграции используется метод авторизации **https://app.swaggerhub.com/apis-docs/postkassa/account/1.0.0#/Вход%2C%20выход%20и%20сброс%20пароля/LoginAuto**

При успешной авторизации в отклике команды будет содержаться токен авторизации, который в дальнейшем необходимо передавать в каждом REST-запросе в поле заголовка `X-AUTH-TOKEN`.

Пример получения статуса почтового отправления

Авторизация на сервере

```
curl -X POST https://account.postkassa.ru/api/v1/login-auto -H 'Content-Type:
application/json' -d '{"email": "example@example.com","password": "password-example"}'

{
    "token":
"MTYzMjkzODAzNHxOd3dBTkVKRFRVaFhUazVSVWtReldVbE9UVmN5TlVVM1JWZ3lSa1UyU0ZkQ1dVMDBOMUZhVDBGRE1r
    "anti_csrf_token": "393b8f4d04c17cc0"
}
```

Запрос статусов почтовых отправлений

В строке запроса в значение **from** указываются идентификаторы РПО через запятую. В заголовок в поле **X-AUTH-TOKEN** заносится значение токена, полученного в результате выполнения команды **login-auto**.

Результатом выполнения запроса является JSON-объект, в поле **rpos** которого содержится массив почтовых отправлений. Статус каждого почтового отправления содержится в поле **status**. Значение **loaded** означает, что почтовое отправление было загружено в Почтовую кассу, значение **charged** означает, что почтовое отправлени было оплачено

```
curl -X GET 'https://account.postkassa.ru/api/v1/account/rpo?
field=rpo_id&from=123497494127,123497494219,123497494141,123497494257' -H 'X-AUTH-TOKEN:
MTYzMjkzODAzNHxOd3dBTkVKRFRVaFhUazVSVWtReldVbE9UVmN5TlVVM1JWZ3lSa1UyU0ZkQ1dVMDBOMUZhVDBGRE1rC


{
  "rpos": [
   {
    "rpo_id": "123497494127",
    "ext_rpo_id": "ext-1",
    "loaded_at": "2021-09-29T21:12:06.660125+03:00",
    "rpo_date": "2021-03-05",
    "amount": 2350,
    "espp": "112233",
    "vendor_id": "test",
    "vendor_name": "Тестовый платежный оператор",
    "status": "loaded",
    "source": "csv_by_user",
    "source_name": "rpos.csv",
    "pochta_incoming_id": "",
    "underpay_amount": 0,
    "registry_date": ""
   },
   {
    "rpo_id": "123497494219",
    "ext_rpo_id": "ext-2",
    "loaded_at": "2021-09-29T21:12:06.670668+03:00",
    "rpo_date": "2021-04-01",
    "amount": 3000,
```

```
    "espp": "112233",
    "vendor_id": "test",
    "vendor_name": "Тестовый платежный оператор",
    "status": "loaded",
    "source": "csv_by_user",
    "source_name": "rpos.csv",
    "pochta_incoming_id": "",
    "underpay_amount": 0,
    "registry_date": ""
   },
   {
    "rpo_id": "123497494141",
    "ext_rpo_id": "ext-3",
    "loaded_at": "2021-09-29T21:12:06.679534+03:00",
    "rpo_date": "2021-05-07",
    "amount": 1525.75,
    "espp": "112233",
    "vendor_id": "test",
    "vendor_name": "Тестовый платежный оператор",
    "status": "charged",
    "source": "csv_by_user",
    "source_name": "rpos.csv",
    "pochta_incoming_id": "11223344",
    "underpay_amount": 0,
    "registry_date": "2021-15-29T21:12:06.685406+03:00"
   },
   {
    "rpo_id": "123497494257",
    "ext_rpo_id": "ext-4",
    "loaded_at": "2021-09-29T21:12:06.685406+03:00",
    "rpo_date": "2021-06-20",
    "amount": 4870,
    "espp": "112233",
    "vendor_id": "test",
    "vendor_name": "Тестовый платежный оператор",
    "status": "charged",
    "source": "csv_by_user",
    "source_name": "rpos.csv",
    "pochta_incoming_id": "11223344",
    "underpay_amount": 0,
    "registry_date": "2021-15-29T21:12:06.685406+03:00"
   }
  ],
  "offset": 0,
  "total": 4,
  "amount": 11745.75,
  "underpay_amount": 0
 }
```


Завершение сессии на сервере

```
curl -X POST https://account.postkassa.ru/api/v1/logoff -H 'Content-Type: application/json'
-H 'X-AUTH-TOKEN:
MTYzMjkzODAzNHxOd3dBTkVKRFRVaFhUazVSVWtReldVbE9UVmN5TlVVM1JWZ3lSa1UyU0ZkQ1dVMDBOMUZhVDBGRE1rC
-d '"anti_csrf_token": "393b8f4d04c17cc0"'

{}
```

Двухфакторная авторизация для защиты специальных операций

Специальные операции

Для выполнения части операций требуется выполнить дополнительную авторизацию. К таким операциям относятся команды отправки денег на карты, изменение пароя, телефона и другие. При работе с личным кабинете **https://accpount.postkassa.ru** через браузер данные операции защищаются с использованием подтверждения операции через СМС на телефон, привязанный к аккаунту. При работе через интеграционное API данные операции защищаютс криптографическим способом путем подписывания команд электронно-цифровой подписью (ЭЦП).

Подготовка к работе с ЭЦП

Для того, чтобы команды можно было подписывать цифровой подписью, сперва необходимо загрузить в личный кабинет публичный ключ, который будет использоваться Почтовой кассой для проверки цифровой подписи. Загрузка публичного ключа осуществляется Личного кабинета на странице `Настройки - API` Личного кабинета: **https://account.postkassa.ru/settings**.

Публичный и приватный ключи могут быть сгенерированы, например, с использованием программы `OpenSSL`.

Сперва генерируется приватный ключ, которым будут подписываться команды, в файл `privkey.pem`:

```
openssl genrsa -out privkey.pem 2048
```

Затем создается публичный ключ:

```
openssl rsa -in privkey.pem -outform PEM -pubout -out pubkey.pem
```

После чего, файл публичного ключа `pubkey.pem` необходимо загрузить в личный кабинет Почтовой кассы по адресу **https://account.postkassa.ru/settings** на вкладке `API`.

Подписывание команд ЭЦП

Формирование подписи

1. Исходная строка для подписи получается путем конкатенации через `\n` следующих значений:
    - HTTP-метода в верхнем регистре (`POST` или `GET`)

- URI из описания команды в **https://app.swaggerhub.com/apis-docs/postkassa/account/1.0.0**, (*) причем URI берется без хоста, параметры должны быть UrlEncoded
- тела запроса (`Content`) запроса HTTP

2. Из строки формируется хэш-значение `SHA-256`, которое подписывается с помощью приватного RSA ключа.
3. Полученная подпись кодируется в `Base64`.
4. Итоговое значение записывается в параметр `X-POSTKASSA-SIGNATURE` заголовка HTTP-запроса.

Пример строки для подписи

Ниже приведен пример строки для полписи команды отправки денег на карту
https://app.swaggerhub.com/apis-docs/postkassa/account/1.0.0#/Управление%20аккаунтом/SendPayout

```
var rawString = 'POST\n/account/payout/send\n{"amount": 5000.75,"pan":
"1111111111111111","vendor_id": "bank123","details": "","client_id": "ext-100}'
```

Пример формирования подписи на PHP

```
function sign(string $method, string $uri, string $data, string $private_key_pem): string
{
    openssl_sign("{$method}\n{$uri}\n{$data}", $signature, $private_key_pem,
OPENSSL_ALGO_SHA256);

    return base64_encode($signature);
}
```

Пример формирования подписи на GO

```
func SignRequest(method string, uri string, data string, rsaPrivateKeyFile string)
(string,error) {

    key, err := LoadRsaPrivateKey(rsaPrivateKeyFile)
    if err!=nil {
        return "",err
    }

    str := PrepareString(method, uri, data)

    return RsaSign(key,str)
}

func PrepareString(method string, uri string, data string) string {
    return fmt.Sprintf("%v\n%v\n%v",method,uri,data)
}

func RsaSign(key *rsa.PrivateKey, data string) (string, error) {
```

```
 hashed := sha256.Sum256([]byte(data))

 signature, err := rsa.SignPKCS1v15(rand.Reader, key, crypto.SHA256, hashed[:])
 if err != nil {
  return "", err
 }

 b64 := base64.RawStdEncoding.WithPadding(base64.StdPadding).EncodeToString(signature)
 return b64, nil
}

func LoadRsaPrivateKey(rsaPrivateKeyLocation, rsaPrivateKeyPassword string)
(*rsa.PrivateKey, error) {
 if rsaPrivateKeyLocation == "" {
  return nil, errors.New("No RSA Key given")
 }

 priv, err := ioutil.ReadFile(rsaPrivateKeyLocation)
 if err != nil {
  return nil, errors.New("No RSA private key found")
 }

 privPem, _ := pem.Decode(priv)
 var privPemBytes []byte
 if privPem.Type != "RSA PRIVATE KEY" {
  return nil, errors.New("RSA private key is of the wrong type")
 }

 if rsaPrivateKeyPassword != "" {
  privPemBytes, err = x509.DecryptPEMBlock(privPem, []byte(rsaPrivateKeyPassword))
  if err != nil {
   return nil, errors.New("unable to decrypt passphrase")
  }
 } else {
  privPemBytes = privPem.Bytes
 }

 var parsedKey interface{}
 if parsedKey, err = x509.ParsePKCS1PrivateKey(privPemBytes); err != nil {
  if parsedKey, err = x509.ParsePKCS8PrivateKey(privPemBytes); err != nil {
   return nil, errors.New("Unable to parse RSA private key")
  }
 }

 var privateKey *rsa.PrivateKey
 var ok bool
 privateKey, ok = parsedKey.(*rsa.PrivateKey)
 if !ok {
  return nil, errors.New("Unable to parse RSA private key")
 }

 return privateKey, nil
}
```