#37 HTTP Notification Provider Contest

Code and technical details:
https://github.com/RSquad/notification-provider

Demonstration stand:
- **https://ton-provider.rsquad.io**
- **https://ton-provider.rsquad.io/explorer**

For questions:
- @AntonVarlamov
- @dialouliti
- @Ilya_axakov
- @PolyakovAlexei

RSquad-Notification-Module

Description of the contest solution

As part of the contest, it was required to implement an HTTP notification module for external applications and services. According to the requirements, this module must be able to send notifications via the HTTP protocol. Free TON users needed a module that would send messages and ensure anonymity of the blockchain users.

The **Queue Provider** does half the work and has already been created. It forwards encrypted messages with an identifier to Notification Provider which determines where the message should be sent next according to the received identifier.

Based on the requirements, a notification module has been implemented. Notification module consists of:

- a local database that stores users notifications which had not received yet;
- a set of API methods required to work with the module;
- Apache Kafka subscription;

- daemon, responsible for sending notifications to users;
- daemon, responsible for clearing the logs;

When the module is deployed, a database has been created (for simplicity and clarity of testing, a method of storing data in the form of a .json file is used. It can be easily replaced with another storing method, for example, with one of the relational databases) and a connection with Apache Kafka had been established. Alerts are recorded only for those users who have created a record about themselves in the module. The record stores the clientId and url to which the notification should be sent.

A user can create multiple records, i.e. notifications will be carried out on all of them. After receipt of a message from the Queue Provider, the notification module creates records with urls to send notifications to in the database, then saves the message itself and makes a record in the log. Two daemons are also started. The daemon, making sure that the table with logs does not grow uncontrollably, deletes records with creation time more than one week ago. The notification daemon makes an attempt to send notification at an equal time interval set in the module's configuration. If the message is delivered to a proper url (code 200 received), the daemon removes the record from the queue. After the notification is sent and delivered to all urls, its body is removed from the database, but not from the logs. If any of the url remains unavailable (or returns any other code except 200) for too long, the message also is removed from the queue. The time interval for deleting records is specified in the module configuration. The sending frequency is also configurable.

Implementation of module requirements

General information about the implementation of the module in accordance with the architecture of the system and the requirements of the contest.

Feature	Status	Comment
Add a unique identifier and notification parameters to the internal database	Implemented	The data is stored in a .json file for easier testing (the usage of a relational database will be implemented in the future)

Get the configuration API method	Implemented	The module configuration is stored in the database and is available for reading
Module information	Implemented	Displayed on the web-site
Get the structural input parameters for the current module	Implemented	Presented in the contest documentation
All HTTP API methods must return a 200 response if the requested operation is successful and corresponding HTTP error code otherwise	Implemented	
http-server with some UI	Implemented	
Guaranteed delivery of notifications within N time and repeated delivery of notifications if the delivery address is unavailable	Implemented	
Support for HTTPS protocol	Implemented	
Logging of events of http notifications for the possibility of displaying them in charts	Implemented	Logs are cleared of records older than one week
Availability of documentation with usage examples	Implemented	Presented in the contest documentation
Compiling, building, deploying, running and testing instructions with prerequisites	Implemented	Presented in the contest documentation

Within the contest, a module, which can be easily expanded to a full-fledged notification system, was implemented. Major components are presented and also can be expanded and scaled.

API Methods description

The table below describes the implemented API methods with parameters and the structure of return values. For convenience, a graphical interface had been deployed on https://ton-provider.rsquad.io/explorer

API method	**Params**	**Returns**

[PATCH] /configurations	{
	"resetPeriod": “number”,
	"deliveryPeriod": “number”
	}
		{
		"count": “number”
		}

[GET] /configurations		[
		{
		"id": "string",
		"resetPeriod": “number”,
		"deliveryPeriod": “number”
		}
		]

[GET] /logs/count		{
		"count": “number”
		}

[GET] /logs		[
		{
		"id": "string",
		"kafkaId": "string",
		"nonce": "string",
		"timestamp": “number”
		}
		]

[PATCH] /users/{id}	{
	"id": "string",
	"kafkaId": "string",
	"url": "string"
	}	

[GET]
/users/{id}
	{
	  "id": "string"
	}
		{
		  "id": "string",
		  "kafkaId": "string",
		  "url": "string"
		}

[DELETE]
/users/{id}
	{
	  "id": "string"
	}	

[POST] /users	{
	  "kafkaId": "string",
	  "url": "string"
	}
		{
		  "id": "string",
		  "kafkaId": "string",
		  "url": "string"
		}

Deployment instructions

It is recommended to use Docker to run the module. In the script, you need to pass the parameters of the values of the variables:

```
docker run --name notify-api \
  -v /var/docker/data/notify-api:/home/node/app/data \
  -p 3010:3000 \
  --read-only \
  -d \
  -e KAFKA_HOST='YOUR_VALUE'\
  -e KAFKA_CLIENT_ID='YOUR_VALUE' \
  -e SASL_MECHANISM='YOUR_VALUE' \
  -e SASL_USERNAME='YOUR_VALUE' \
  -e SASL_PASSWORD='YOUR_VALUE' \
  -e GROUP_ID='YOUR_VALUE' \
  -e TOPIC='YOUR_VALUE' \
  notify-api
```

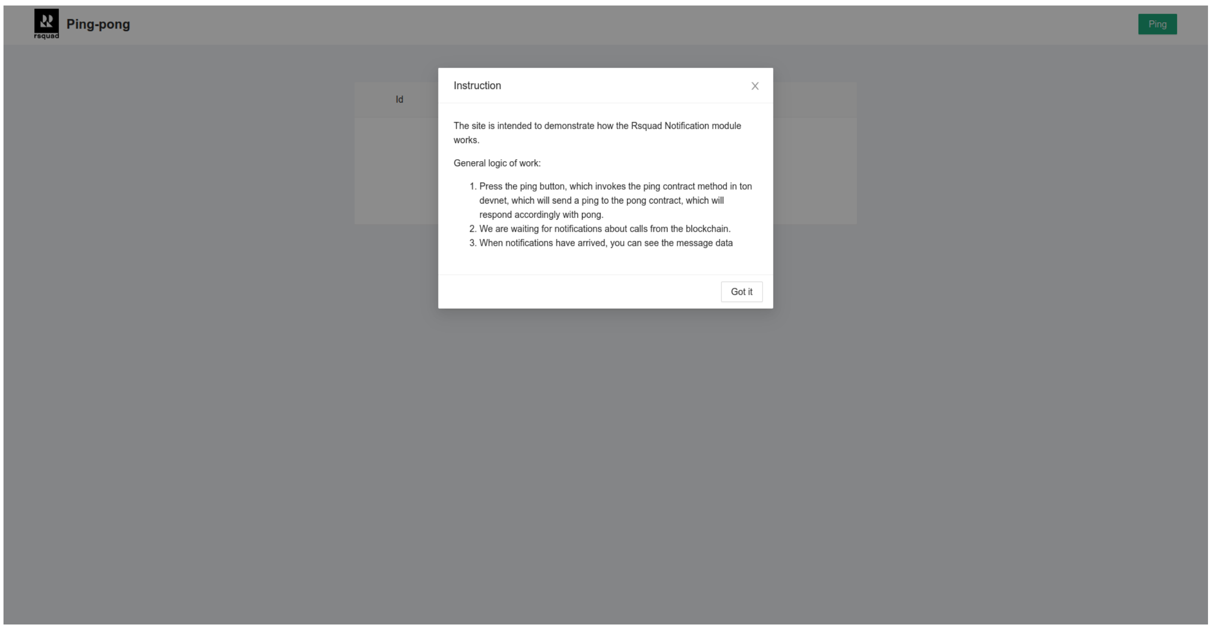
You can also start this module using **npm start**. In this case, the above variables must be placed in the **.env** file at the root of the project. Variable names are similar to the names in the bash script.

The repository contains bash-scripts for deploying the project through Docker. The **build-docker.sh** script is run first, followed by **create-docker.sh**. The create script must be modified to suit your settings.

To provide support for the **https** protocol, you need to **configure Nginx** or in another way.

Usage example

After deploying the module with our settings, we need to create a user record. The user must already be registered in **Queue Provider.**



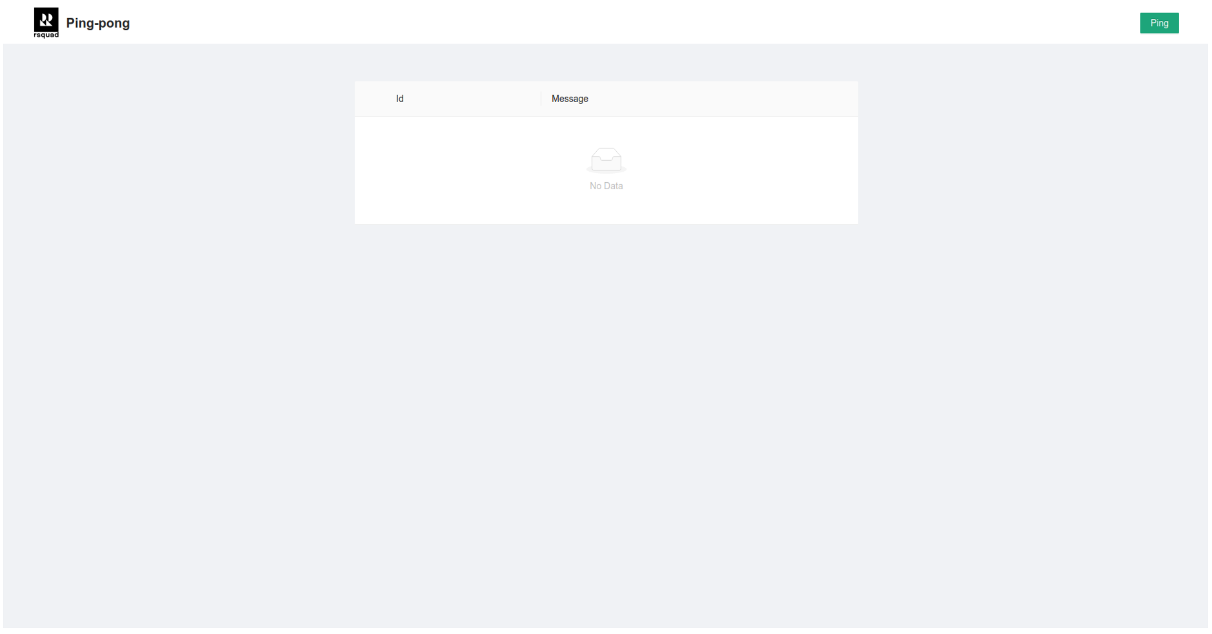
Instruction

The site is intended to demonstrate how the Rsquad Notification module works.

General logic of work:

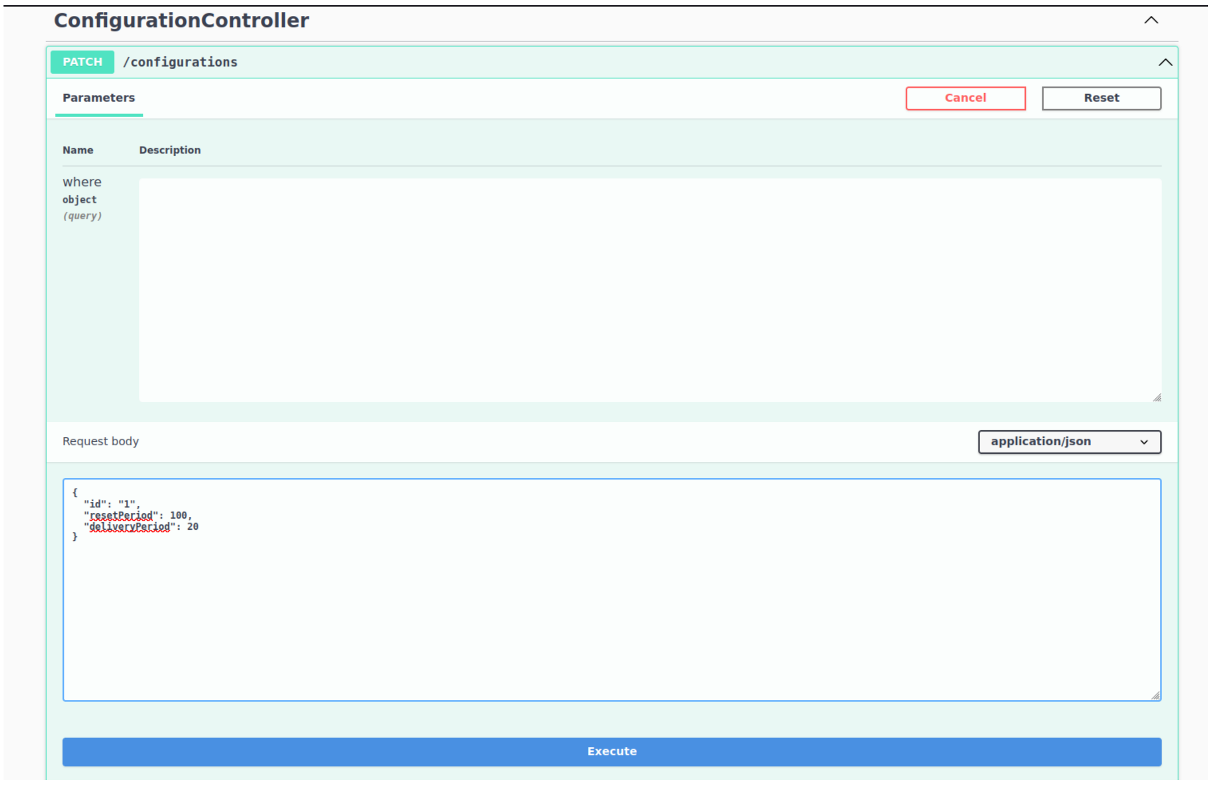
1. Press the ping button, which invokes the ping contract method in ton devnet, which will send a ping to the pong contract, which will respond accordingly with pong.
2. We are waiting for notifications about calls from the blockchain.
3. When notifications have arrived, you can see the message data

The module will begin to receive notifications and send them to the specified address in encrypted form.



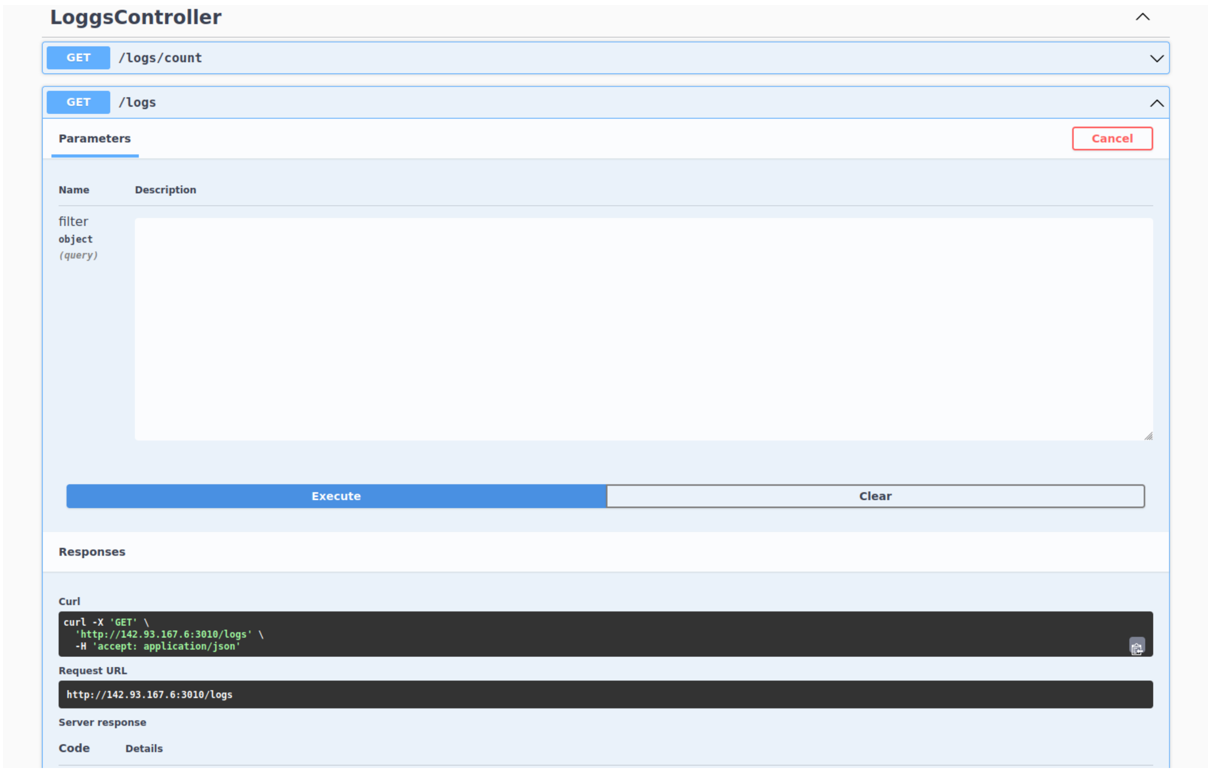
To configure the module, you can modify the message store period and the frequency of attempts to send notifications.

https://ton-provider.rsquad.io/explorer



```
{
  "id": "1",
  "resetPeriod": 100,
  "deliveryPeriod": 20
}
```

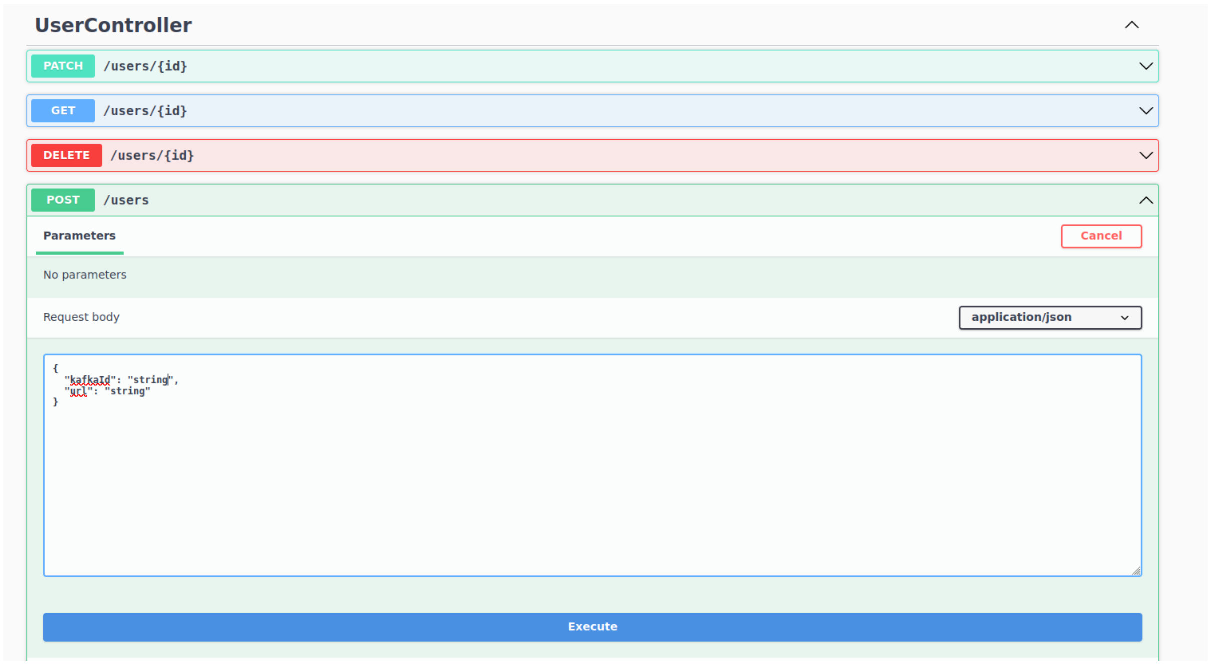
It is also possible to get logs from the database.



```
curl -X 'GET' \
  'http://142.93.167.6:3010/logs' \
  -H 'accept: application/json'
```

```
http://142.93.167.6:3010/logs
```

You can also create a user through explorer.



No parameters

Request body: application/json

```
{
  "kafkaId": "string",
  "url": "string"
}
```

Information on the instructions https://github.com/freeton-org/readme will be placed in the repository, since the application number is not known before publication.