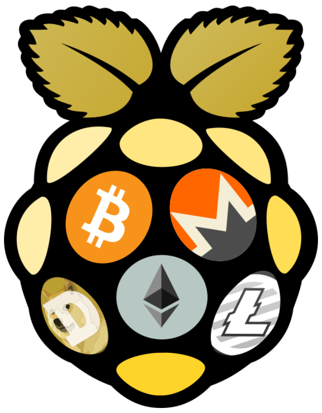
RaspberryWallet / **Backend**

Backend containing REST HttpServer, Manager daemon and modules of Raspberry Wallet

#kotlin #java #bitcoinj #raspberry-pi-gpio #ktor #bitcoin #bitcoin-wallet #websockets #coroutines #raspberry-pi-zero-w Manage topics

275 commits **6** branches **1** release **4** contributors

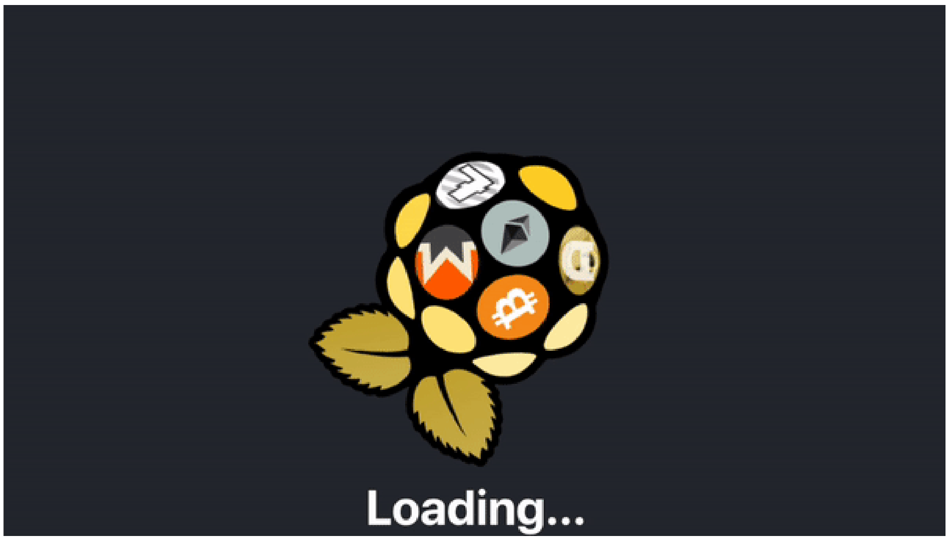
README.md



Backend

PASSED

This module is responsible for main logic on the whole wallet. It contains HTTP Server, that is accessible by client, internal Manager logic, which locks and unlocks modules and also it contains built in Modules logic or interfaces that provide access to them.



Requirements

- Maven 3+
- Java 8
- Kotlin 1.3.0

Installation

```
# get repo
git clone https://github.com/RaspberryWallet/Backend.git
cd Backend

# build jars and install them into your local repository
mvn clean install

# run Manager, modules option specifies directory path of Modules classes
java -jar target/manager_artifact_name.jar [-modules <arg>]
```

Table of contents

Modularity

Our architecture is focused on modularity. We want to give user a choice, if he wants to have very secured wallet, with a lot of modules, where every module stores a little piece of the whole secret. Or if you are just enthusiastic user and you are happy with basic security, then single module with PIN is fine for you.

It's all about you, how you decide to configure your Wallet.

By that, it's possible to use our standard modules, that are ready to use, or add your custom modules, that are imported as jar packages.

Module implementation

It's overall look about "how to implement module". You can find out more technical details in Adding custom modules section.

Every single module have to implement abstract class `Module`:
https://github.com/RaspberryWallet/Backend/blob/master/Manager/src/main/java/io/raspberrywallet/manager/modules/Module.java

It may look complicated, but in fact, most of the logic is implemented by default as protected methods. You just have to fill abstract methods.

Module also, should follow package naming convention:

- Your module class name must contains postfix `Module` e.g. `PinModule`, `PushButtonModule`
- YourModule class must be placed in `io.raspberrywallet.manager.modules.<name>` e.g `io.raspberrywallet.manager.modules.pin`

Otherwise it won't be loaded.

Modules by default reads configuration from their configuration class that implements `ModuleConfig` interface. You don't have to implement anything, `ModuleConfig` is just for generalization and more like class metadata.

You can find out more about configuration in Configuration section.

The last thing you have to do, is signing jars, that must be done, due to verification needed for security. It's easy to imagine situation, where Wallet loads attacker's jar Module, that may cause catastrophic problems to your funds. 😼

Adding custom modules

As we said before, your `CustomModule.java`

- must implement `io.raspberrywallet.manager.modules.Module` interface
- must be in package `io.raspberrywallet.manager.modules.<your_package_name>`

Manager loads modules (in `jar` format) from `/opt/wallet/modules` or specified by `-modules path/to/modules/` relative to your current directory.

In order to compile your `CustomModule.java` follow these steps:

```
# run Manager and specify correct path to your module and other modules
javac -cp Manager/target/Manager-<version>-jar-with-dependencies.jar /path/to/your/CustomModule.java -d modules/
```

```
# create jars
jar cvf CustomModule.jar CustomModule.class

# sign jar
jarsigner -keystore RaspberryWallet.keystore -signedjar CustomModule.jar CustomModule.jar signModules
```

And after these, your module can be loaded on startup 🎉

There is also helper script in `Scripts/modules/compileJarSignAndCopyModules.py` that automate everything.

It will be automatically loaded and verified on startup

```
ℹ[INFO][23:45:55][] Successfully verified module io.raspberrywallet.manager.modules.pin.PinModule
ℹ[INFO][23:45:55][] Successfully verified module io.raspberrywallet.manager.modules.example.ExampleModule
ℹ[INFO][18:12:39][ModuleClassLoader] Loaded 2 modules
ℹ[INFO][18:12:39][] Module {
        name: ExampleModule
        id: io.raspberrywallet.manager.modules.ExampleModule
        description: An example waiting and xoring module to show how things work.
}
ℹ[INFO][18:12:39][] Module {
        name: PinModule
        id: io.raspberrywallet.manager.modules.PinModule
        description: Module that require enter 4 digits code
}
```

Configuration

RaspberryWallet can be configured with YAML configuration. It depends on modules, what configuration do they need. Every module should be documented by itself.

Configuration example

```
# default global configuration
version: 0.5.0
base-path-prefix: /opt/wallet/
autolock-seconds: 60

bitcoin:
  network: testnet
  user-agent: RaspberryWallet

server:
  keystore-name: RaspberryWallet.keystore
  keystore-password: raspberrywallet
  key-alias: ssl
  port: 80
  secure-port: 433

# every module has his own configuration
modules:

  # configuration is fully customisable by Module author
  # module name must match the name given in code (class name)
  PinModule:
    # JSON nodes with configuration, that does directly map to class fields
    # you specify node name in code with annotation @JsonAlias("name")
    max-retry: 5

  AuthorizationServerModule:
    host: https://127.0.0.1
    port: 8443
    accept-untrusted-certs: true

    # configuration can be nasted, can contain lists etc. matches JSON possibilities
    endpoints:
      set-secret: /authorization/secret/set
      overwrite: /authorization/secret/overwrite
```

Modules by default reads configuration from their configuration class that implements `ModuleConfig` interface. You don't have to implement anything, `ModuleConfig` is just for generalization and more like class metadata.

Module configuration example

Let the code explain how it works. We've got the following configuration for `PinModule`:

```
modules:
  PinModule:
    max-retry: 5
```

Fields are specified in `PinModuleConfig` class like this:

```
public class PinModuleConfig implements ModuleConfig {

    @JsonAlias("max-retry")
    public int maxRetry = 3;

    @JsonAlias("min-length")
    public int minLength = 4;

    @JsonAlias("max-length")
    public int maxLength = 9;
}
```

As you can see, `max-retry: 5` is mapped by Wallet to a field `public int maxRetry`.
You can assign values to other fields implemented in `PinModuleConfig` class analogically.

Documents

More documents can be found in dedicated repository here

Authors

Name	email
Stanisław Barański	stasbar1995 at gmail.com
Patryk Milewski	patryk.milewski at gmail.com
Piotr Pabis	rotworm0 at yahoo.pl

Changelog

Version	Is backward- compatible	Changes	Commit ID
			There are no released versions