Blackfort

Cryptocurrency Wallet

To develop a custom wallet on Flutter framework for Blackfort users. The main goal is to raise people’s trust in the project and provide them with a useful and convenient tool for managing their assets, including the usage of exclusive features provided by Blackfort opportunities.

Text

Vision and Scope

1. Business requirements

The purpose of this Vision and Scope document is to provide an overall understanding of the Project challenges, business needs and application scope.

1.1. Background

Cryptocurrency wallet. For better understanding you can install trust wallet app:
https://trustwallet.com/
Can be taken as standard. In many ways it will be similar to our app.

The trust wallet core is used to work with various blockchains —
https://github.com/trustwallet/wallet-core/ (C++ library with good documentation. Just configure, connect and use it)

1.2. Business Opportunity

It was decided and planned to develop a custom wallet for Blackfort users. The main goal is to raise people's trust in the project and provide them with a useful and convenient tool for managing their assets, including the usage of exclusive features provided by Blackfort opportunities. As long as smartphones became an essential part of everyday life, the team agreed to begin with the development of the mobile application.

1.3. Business Objectives

#	Description
BO-1	The MVP version is intended to demonstrate the concept of the project to potential members and investors

1.4. Business Ricks

#	Description
BR-1	The solution includes a sufficient number of third-party services, which increases the dependence on external factors associated with these services, which are difficult to control and foresee.

Vision of the Solution

User roles

User	Description
Wallet owner	A user who created an account in the mobile application and uses wallets.

Assumptions and Dependencies

In this subchapter, all assumptions that were made when conceiving the project and writing this Vision & Scope document will be defined. Major dependencies of the project must rely upon for success, such as specific technologies, third-party vendors, development partners, or other business relationships will be listed in this subchapter as well.

#	Description
D-1	The trust wallet core is used to work with various blockchains — https://github.com/trustwallet/wallet-core/
D-2	The fiat equivalent must be obtained from a third-party service: Coinbase, CoinGecko or similar
D-3	The cryptocurrency price charts must be obtained from a third-party service: Coinbase, CoinGecko or similar
D-4	For KYC identification it is supposed to use the service: Basis iD. BASIS ID API integration
D-5	Chat widget requires integration with a third-party service: Intercom or similar

Scope of Initial Release

In this subchapter, the major features that will be included in the initial release of the product are described.

Epic	Story As a User, I want to...	Acceptance Criteria: A User should be able to...
User Application		
Start loading screen & Onboarding screens	get introductory information about the wallet	▪ See wallet logo and name ○ Logo in the center of the screen on a light or dark background depending on the app theme ▪ See a carousel of onboarding screens ▪ Switch to the next/previous onboarding screen ▪ Skip the onboarding
Registration (Wallet creation or Import)	Python Backend — Accounts API (boyard) Blackfort Group database, Trust Core SDK, Twilio Verify API SDK (for future).	
	Input user info	Fields: ▪ First name ▪ Last name ▪ Email ▪ Phone — not MVP ▪ Country (select from the drop-down list of countries) Button for the next step.

	Phone Confirmation	We take this functionality out of MVP, it will be necessary when we will use the e-money feature ▪ Message with SMS code. ▪ Input code field.
	Email Confirmation	▪ Message with a link is sent to the user. ○ the link must only be valid for 1 hour
	Wallet creation page	▪ Secret phrase generation. ▪ Copy button for secret phrase. ▪ Continue button.
	Secret phrase confirmation	This step is optional, the user can skip it ▪ Input field. ▪ List of words from the secret phrase in random order. ▪ Ability to drag or tap words into the input field.
	Import secret phrase	1. Input field for secret phrase. 2. Import button.
	Enabling or disabling biometrics	The user should be able to enable or disable biometric login instead of a username and password Face ID or fingerprint, depending on the device
Navigation bar		(icons only): portfolio, browser, scanner, swap, setting

Wallet portfolio	Trust Core SDK Some blockchain APIs (to get data about transactions, prices, charts, etc.) Backend for Fiat	
	Wallet balance	Total portfolio balance (sum of all crypto and fiat balances in preferred currency)
	Button for adding new Token	Open next page — Add Token Page: Fields: 1. Contract Address (QR-scanner and Paste button) 2. Name (optional, should be fetched by address) 3. Symbol (optional, should be fetched by address) 4. Decimals (optional, should be fetched by address) 5. Add Token Button. SHOULD BE APPLICABLE FOR ALL BXN, ETH, AS WELL AS THE ADDITIONS OF NEW BLOCKCHAINS ETC SHOULD BE A DATA FETCH IF AVAILABLE FROM A REPUTABLE SOURCE
	List of cryptocurrency and fiat tiles, tiles contain	Symbol Name Rate (by rate of currency which is set in app settings — preferred) Rate change in percentage User balance in current cryptocurrency (or current fiat) User balance in Fiat (by preferred currency rate)

		On Click: 1.3.1. Current Balance 1.3.2. Buy Button (Transak for crypto) 1.3.3. Deposit Button (for fiat only) 1.3.4. Swap (moves to swap screen with pre-selected current FROM currency) 1.3.5. Receive Button 1.3.6. Send Button 1.3.7. Withdraw (fiat only, withdraw to a bank account) 1.3.8. Transfer Buttons (fiat only, send to other user by some internal identifier)
	see transactions history	See the list of transactions for each asset with the info: ▪ transaction date and time ▪ amount ▪ type: ○ send ○ receive ○ swap ▪ sender address ▪ receiver address
Send/Receive	Send transaction	Send transaction (assume we've chosen crypto to send) 1. Ask for the destination address and amount 1.4.2. ENS domains Switch between input crypto amount or fiat one

		Send MAX button 1.4.3. Confirm transaction screen Amount From To Network fee Button to set custom fees on transaction send Example of this option can be found in the Trust wallet on the send transaction page a) Button that can enable or disable the setting of fees in transactions. b) If Button is enable the button “fee setting” appears in the transaction sending window, and opens page with: ▪ Input field for Gas Price (Gwei) ▪ Input field for Gas Limit ▪ Input field Transaction data (optional) ▪ Miner tip (on availability like ETH blockchain) ▪ Input field Nonce ▪ Button “Save” and Button “Return to transaction”
	Receive crypto	1.5. Receive crypto (assume we’ve chosen crypto to receive) ▪ QR-code to scan address. Logo in the center of QR or outside like a card (check Trust Wallet) ▪ Copy Address ▪ Share Address

		▪ Ask for the custom amount (in base currency or selected cryptocurrency)
Price chart to fiat	Price chart to fiat	Current Price (last update time) ▪ 24h volume ▪ Market Cap ▪ Blockchain Explorer ▪ Short Description ▪ Chart with availability to choose date/time frame and percent change
BlackFort Scan	Interact with: camera, Trust Core SDK	
	QR-code scanner	To scan an address in the form of a QR-code. Scan QR wallet seed TO DO functionality Scan payment Wallet connect
	Wallet connect	2.2.1 Connection to following blockchains: 1. Ethereum 2. Binance Smart Chain 3. BlackFort X Network 4. All supported smart chains

Swap	Swap is merged with fiat buy/sell. It means that on the same screen you can exchange crypto-to-crypto, fiat-to-crypto (from fiat balance), crypto-to-fiat (to fiat balance). We'll need a bit designed asset of cryptocurrency icons (should be checked on a small amount of cryptocurrencies) just to see what it will look like. Interacts with: Python Backend for Swaps, Python Backend for Fiat Swaps.	
		1. Upper-right corner — Button to display the history of exchanges. On click — Open Transaction List: List of transactions tiles, which contains: a) Status: Processing: Waiting symbol and yellow lettering. Failed: Failure symbol and red lettering There will be a button that pops up in the list of swaps on the right. This opens an email with a prefilled text to send to swap support. Success: Success symbol and green lettering. b) Two names of the exchanged currency c) Date of swap (DD-MM-YYYY HH:MM:SS) d) Amount of (rounded to 8 decimal places) e) Amount of received (rounded to 8 decimal places)

		2. The ability to select 2 currencies (crypto or fiat) from the drop-down lists. 3. Input field for a number of tokens or fiat. 4. Field with Auto calculation Receive tokens. 5. Swap Button.
BXN Node dashboard / Economy	BXN Node dashboard	Interacts with BXN backend and BXN Nodes and validators a) Welcome text b) Dashboard Button: BXN Dashboard opening c) Cards: Total Value (amount of BXN Value) Your BXN (amount of user Token) Available Bonus (amount in Earned currency and Fiat) Self Buy (amount in Crypto and Fiat) Buy Price (amount in Crypto and Fiat) Sell Price (amount in Crypto and Fiat) Tokenized Assets (Crypto tokenized and BXN Value) d) Buttons: STAKE DELEGATE VALIDATOR TOKENIZE WITHDRAW
Settings	The Main Settings screen should have icons. Further settings should have short descriptions.	

Interacts with: Python Backend Accounts API, KYC/AML API (basisID or separate own backend), Intercom (?) (for Support Chat)	
Wallets	Drop-down list with user wallets. Ability to switch between wallets. Ability to add a new wallet address. Ability to view in Wallet NFT's owned in various protocols
Base currency	Drop-down list of currencies. Ability to choose the main currency of the wallet.
Mode	Only Dark mode for MVP Additional: dark mode for AMOLED displays
Security — Passcode or biometric	Passcode or biometric (required) ▪ Open App ▪ Crypto Send (Withdraw) ▪ Reveal seed phrase ▪ Private key view
Security — 2FA	2FA (toggled individually) for: Login Withdraw
Chat with support	Support: Chat with support. Preferably this solution: https://developers.intercom.com/

	Languages	List of available application languages. (We can translate all strings) English German Spanish French Italian Croatian Slovenian The ability to change the language of the application. (Uses the system's default language but lets the user change it.)
	About	About info. Contacts. Link to Video Tutorials. App version.
	Screenshots on/off	▪ Enable or disable the ability to take screenshots in the application (in the wallet) ○ If the option is disabled, then the wallet application should not be able to take a screenshot on the user's device. ○ The user should be shown a message about it
User profile	Manage personal data	Input: First Name(s), Last Name, Email, Phone number, Address line 1

		Address line 2 City: Post code: Country:
	Logout Button	Logout
	Delete Wallet Button	Deletes data from the server
	List of running sessions	Kill all sessions (suggestion)
	Delete account	Delete account
Notifications	Notifications	When user: Sent transaction Received transaction Failed transaction Smart contract call executed Smart contract call failed Custom notifications for updates (in the language set by the user for the app) Other stuff in case we want to send something
OPTIONS for integrating new blockchains	OPTIONS for integrating new blockchains	OPTIONS for integrating / adding new blockchains or tokens especially for our BXN Network BXRC721 BXRC20 High priority (Top of the list are the highest priorities):

		1. Ethereum / Ethereum Tokens, including ERC-20 2. Bitcoin / BTC 3. BXN / Blackfort Tokens 4. Tron / Tron Tokens 5. Binance Smart chain / BSC Tokens 6. Polygon / Matic Low priority: ▪ Bep2 ▪ Solana ▪ XRP
NFT section	View the list of collectibles	▪ View separate tabs for NFT ▪ Choose a tab ▪ View the list of collectibles ○ The NFT list should be shown as a tile (link to an example screenshot) ○ The NFTs in the list should be grouped by collection. ■ Each NFT in the list must include: ▫ Collection image ▫ Collection name ▫ NFT counts (the number of tokens of the collection that the user already has)

            - Blockchain logo
            - The Blackfort NFT collection should be highlighted by a special icon. (“If holding a certain Blackfort NFT, the wallet should display a design feature to be provided. Such as an emoji or small design element somewhere on the main screen”)
        - Clicking on a collection should drop down to the level below where the list of NFTs in that collection should be displayed
    - Sort — Alphabetical. MVP does not allow to change the sorting
- Each NFT in the list must include:
    - NFT image
    - NFT name
    - Purchase price
    - Fiat equivalent
    - Blockchain logo
    - Blackfort NFT should be highlighted by a special icon.
- By clicking on the image there should open a card of NFT with additional detailed information about it such as:
    - NFT image
    - NFT name
    - Purchase price
    - Fiat equivalent
    - Description

		○ Blockchain logo ○ Blackfort NFT should be highlighted by a special icon. ▪ Filters and search are not planned in the MVP version
	Add NFT	▪ Add purchased NFTs. To do this, the address of the NFT and its ID must be entered
	Send/Receive NFT	▪ To perform Send and Receive transactions with NFT ○ Similar to fungible tokens
Decentralized browser	Mint NFT	▪ The user should be able to go to the project website or to the marketplace (e.g. OpenSea) via dApps browser and connect his wallet there via wallet connect and perform an NFT mint there. By mint we mean the first purchase of NFT (often it is either within the whitelist, or at the premint or mint stage)
	View a list of dApps	▪ View the list of dApps ○ each dApps in the list must consist of: ▪ dApps image ▪ dApps name ▪ dApps description ▪ dApps category ▪ Connect user wallet to the selected dApps via WalletConnect ○ It is planned to use the WalletConnect SDK ▪ The list of dApps will be clarified before the start of the sprint in which the development of this functionality will go

	Confirm/reject transactions	Confirm or reject transactions on third-party websites through the browser by scanning a QR code or through WalletConnect
	Additional functions	▪ Clearing history and cookies ▪ Refresh page ▪ Share the link ▪ Bookmarks (similar to the MetaMask functionality)
Improvement (Additionals)	Loading signs	When the app calculates something or awaiting a request or updating something — tell the user about it using loading percentages or loading signs. The system should inform the user that synchronization, installation or restore is now in progress
	Go to wallet switch screen on hold	Check out the bottom left hold button in Trust Wallet App
	Limited access before verification	Access to all backend stuff is restricted until both email and phone number will be verified
	Hide elements on the screen	Hide certain elements on the application screen. For example, clicking on the "eye" icon should hide the balance on the portfolio screen. The full list of hidden elements will be specified later

Features of Subsequent Release

Feature	Description
Mode	Choice of light or dark app theme Additional: dark mode for AMOLED displays
Send crypto to contacts	List of contacts which are imported from the device and filtered by availability to send crypto by phone number. Otherwise, contacts can be added by username only, phone number is additional
Help text to User on transaction send page	How to unstuck, if your transaction is stuck: You can send a transaction to yourself with the same nonce as the stuck one, then it will cancel sending. We need also button "Speed up" like in MetaMask app to make it simpler for user
KYC	KYC (using Basis ID / upload system for BXN) CERTAIN TURNOVERS TRIGGER AML CHECK, WILL BE PROVIDED through KYC provider. Integrate Basis ID in the app https://crm.basisid.com/documentationv2.html AML check: Limits and documents needed are as follows: 1) For onboarding, One Proof of Identity & One Proof of address from above. 2) For transactions over €15,000 in a month for individuals (natural person) or €25,000 for legal entities. (Limit resets monthly). Text could be displayed: Please Upload a scan or a good photoscan of your most recent Proof of Income. This could be:

Feature	Description
	i) Bank Statement showing Income or balance stamped and signed by bank representative ii) Financial contracts between You and third parties iii) Most recent Tax return from your tax authority iv) Property Title deeds or Savings bonds/ accounts These apply to Buy with Bank only and DeFi interaction. Not to swap nor Credit Card buy. When 1) is not fulfilled, restricted FIAT options, no Global CryptoReward is issued, no referral link for DeFi and no commissions can be earned in DeFi. Apart from the smart contract-enabled ones. When the limits for 2) are reached, a permanent prompt listing required docs and imposed restrictions and what is required to be submitted in the app forcing the user to submit documents by opening an email to support@blackfort.exchange with the subject “Documents for Enhanced Due Diligence”. The user can either press accept and the email opens or they can press “Cancel” and they return to their wallet however with all the following restrictions; restricted FIAT options, no Self CryptoReward is issued, no referral link for DeFi and no commissions can be earned in DeFi just aggregated. We would also need a Reject document and resend prompt to user to reupload due to… not clear image or unacceptable documents provided. Like sending a tax return from 2010 in 2021 as an example. In the backend we should have a “Block” and “Unblock” user button, for the regular it can be auto blocked when the conditions are satisfied. The block can otherwise be useful for spot checks and for removing T&C breachers. Every time they enter the app the same prompt shows up to remind them.

Feature	Description
	Maybe this could open an input mask to upload the documents and send them to us with all the user IDs, however an email can be a solution too as written above. Whichever is easier and most efficient to implement. Both methods should be sent to support@blackfort.exchange as that opens a support ticket for us to monitor if completed or not which we can then know to press Block or unblock button.
User profile — Bank Accounts	▪ List of user Bank Account: ▪ List of tiles ▪ Display Name of account and Description ▪ On click open page to editing ▪ Button to add a new Account. Open page — Add new bank account: ○ Text-1 ○ Fields to input: ◦ Description ◦ Name of account (prefilled with user's full name) ◦ IBAN ◦ BIC ◦ Bank name ◦ Address ○ Save Button (confirmation email)

Limitations and Exclusions

#	Description
L-1	In MVP it is required to implement only dark mode
L-2	
L-3	

Operating Environment

Mobile app: Android, iOS
Web app: Out of MVP Scope
Design: Design prepared

Appendix 1 — Wireframes

Not relevant, since there is already a finished design.

Main screen (for start we need only tokens section)
You can choose what currencies to display and you can remove them from the screen with swipe
Order by fiat amount of asset

BlackFort :: Tech Vision

General
Blackfort mobile app is designed to work with a crypto wallet that supports a large number of blockchains. The planned to be developed mobile application uses possibilities provided by Blackfort backend. The main functionality of Blackfort client can be divided into several components:

- Crypto Wallet;
- Authentication;
- Decentralized browser;
- NFT Section;
- Support Chat.

Blackfort mobile application will have a multi-platform implementation with support for Android and iOS.
Minimum Supported System Versions:

- iOS 13 - At the moment, it is> 90% of all acting devices;
- Android 6.0 (API 23) - At the moment, it is 96.2% of all acting devices.

Application stack and architecture

General information

Inside the application, six modules can be distinguished based on the tasks that they solve:

- **Application module** - the main application module that implements the Presentation, Domain and Data layers. Coordinates the work of the rest of the application modules;
- **Auth module** - responsible for working with user authentication and authorization in KYC and Boyard;
- **Wallet module** - work with the user's private keys, providing general interfaces for cryptocurrencies (including BXN);
- **NFT module** - responsible for working with user's NFT collections;
- **DApps module** - decentralized browser to interact with decentralized applications (DApps);
- **Support module** - integration with Intercom via SDK for chat support.

Required integrations and services:

Crypto gateway
A mobile phone cannot act as a full-fledged participant in the blockchain network and keep the nodes of different blockchains up to date. Therefore, we need access to a node or service capable of proxying our transactions to the network.
The second important point is working with history. CRYPTO_GATEWAY_NAME supports indexing history by address, which speeds up its receipt on the client without the need for synchronization.

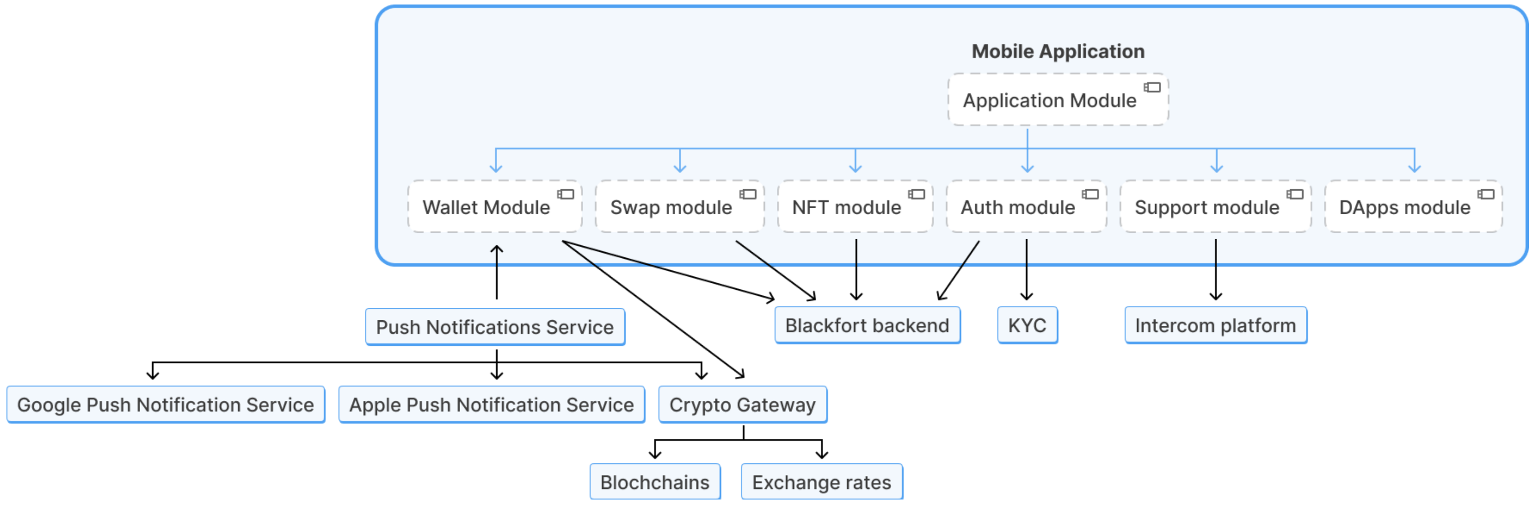
Firebase Cloud Messaging (FCM)
For push notifications to work, you need a backend that monitors changes in the network at the address and sends notifications.

Intercom
Intercom provides its native SDKs. Before integrating, you will need to follow the steps from this tutorial: Intercom SDK installation guides

Tech stack

Languages	Frameworks	Libraries	Delivery and Deployment
Multi-platform: Dart Android: Kotlin, Java iOS: Swift, Objective-C	Flutter, Firebase	Wallet Core Flutter, Intercom Flutter, Firebase Messaging, http, socket_io_client	Git, Fastlane, Google Play (Android), AppStore/TestFlight (iOS)

Architecture



Backend stack and architecture

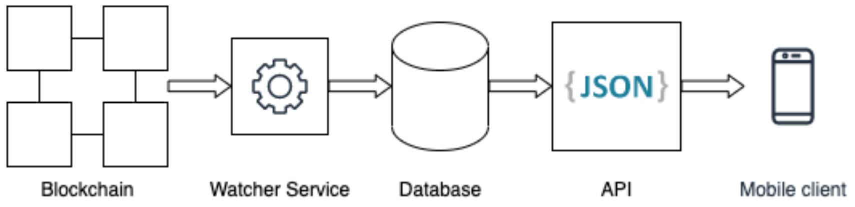
General information

To implement the full functionality of the application, the development of an additional backend will be required. The main tasks of this component:

- Obtaining the user's transaction history, including the transfer of ERC20 tokens
- Getting the user's NFT, including their purchase history to display the price
- Sending push notifications
- Ability to send a transaction

Unfortunately, there is no easy way for blockchains to get a history of transactions. Working with the blockchain comes down to running separate services that work with the database.

It looks something like this - the blockchain notifies a specific service about a new block, and the service parses the information. It saves it to the database, and a separate service retrieves this information from the database and transfers the already indexed data to the client.

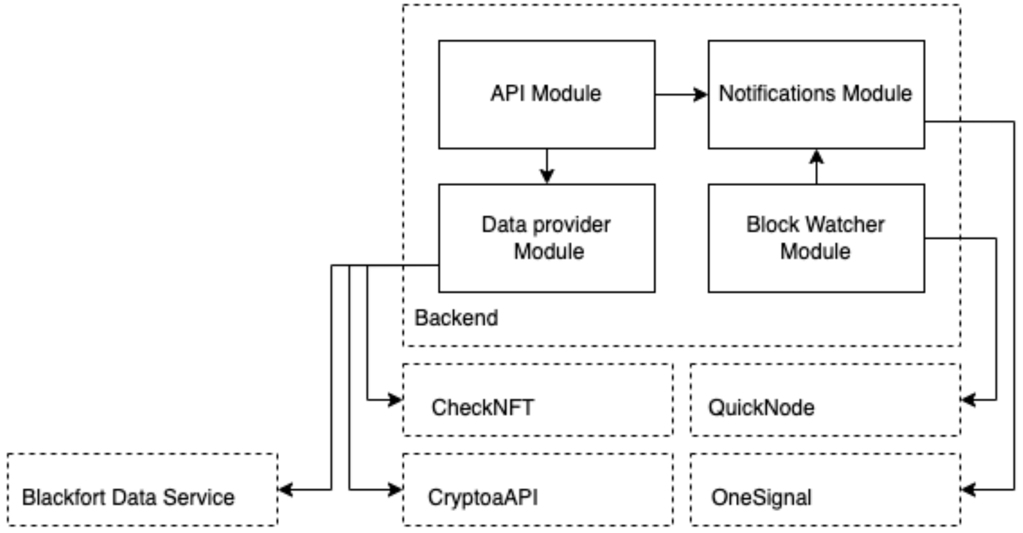


As a result, to provide each user with the opportunity to obtain information about his address, the database must always contain all the data about all accounts, which is quite expensive, since the database size for only one blockchain will be measured in terabytes.

Instead, for the first versions of the application, we propose an approach in which third-party services will be used to obtain data about the user's transaction history and NFT. At the same time, a separate, internal service will be used to watch blockchains to watch new account transactions and send notifications to them.

It is assumed that the wallet will support different blockchains. Each of the blockchains has its own data sources, so the application architecture should include a system of adapters with the ability to adapt data from different sources to one type.

The general structure of the blockchain module system is as follows:



API Module - a module responsible for the external interface of the backend. The mobile application accesses it to obtain transaction history, NFTs and to register a new address to receive notifications.

Data provider Module - responsible for providing the requested data. Accesses external services through adapters, providing unified data to the API module. The module uses different services for different blockchains and data. Subsequently, as such an external service, its own additional module responsible for indexing data can also be used.

Block Watcher Module - the module subscribes to different blockchains and watches blocks. Upon receipt of a new block, it checks for the presence of transactions with the addresses of interest to it and, if any, sends a request to send a notification

Notifications Module - responsible for sending notifications and storing subscribers to these notifications.

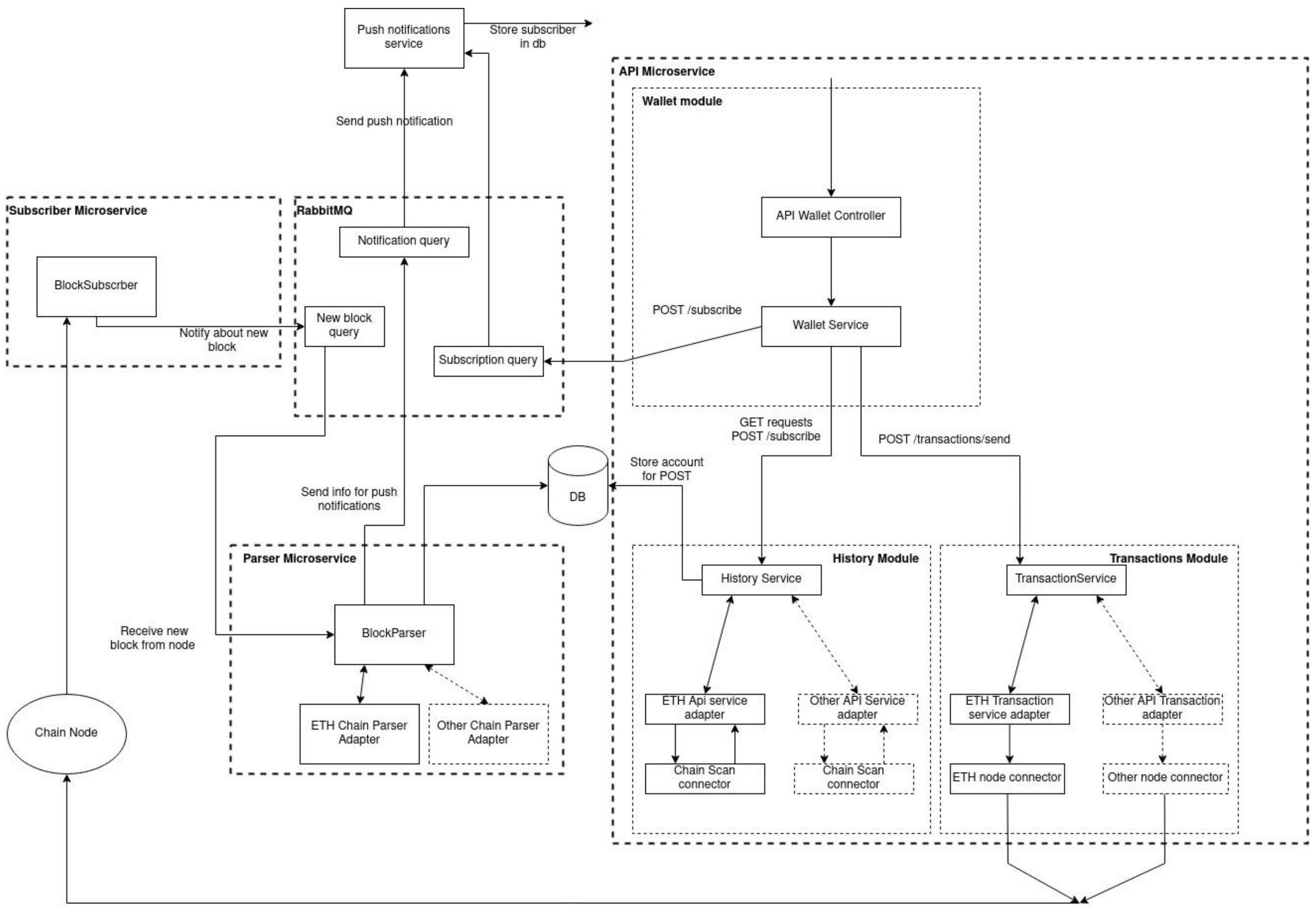
To obtain the history of transactions and NFT, it is proposed to use CryptoAPI and CheckNFT as third-party services. The services provide access to data from blockchains such as Ethereum and Bitcoin, including the ability to get all the details about NFTs and their transfer history.

To obtain data about Blackfort transactions, it is supposed to use the Blackford explorer API, however, if expanding the API will be necessary, we are also considering the option of launching a separate Blackfort Data Service module based on Blockscout.

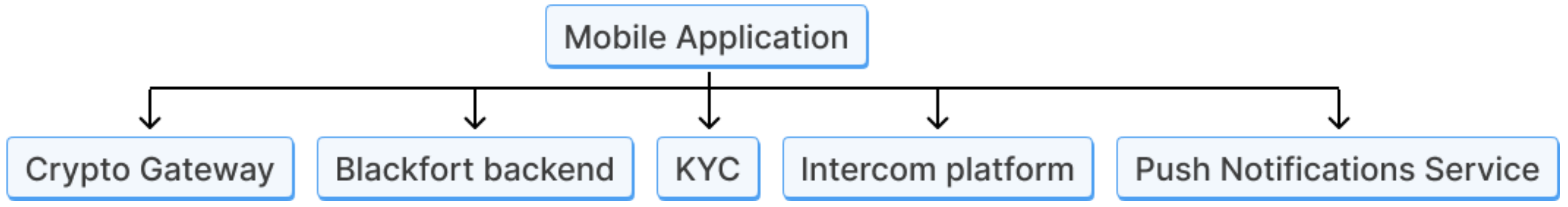
Tech stack

Languages	Frameworks and Standarts	Database	Message Broker	Monitoring
JavaScript, TypeScript, Node.js	Nest.js, Express, TypeORM, REST API, JWT	PostgreSQL	RabbitMQ	Prometheus, Grafana, Sentry

Architecture



Data flow



Profile and settings

Blackfort Mobile App works with data from users of the service through the backend (Blackfort Boyard).

Minimum set of required endpoints:

- Creating a user profile (registration in the system);
- List of running sessions;
- List of user bank accounts;
- Delete account.

Blockchain data

Blackfort mobile app works with blockchain data through Crypto Gateway, such as CRYPTO_GATEWAY_NAME.

This is necessary since the application is not a full-fledged participant in the network and cannot contain the nodes of all the required blockchains. But, it remains safe for users.

The minimum required to work with any blockchain from a mobile application:

- Ability to send a transaction;
- Ability to get the history of transactions and/or balance at the address.

Sending transactions

Sending transactions through the Crypto Gateway remains secure because it cannot interfere with the transaction data. All work with user keys remains strictly local. The assembly and signing of transactions is strictly local. Therefore, the task of the Crypto Gateway is to proxy the signed transaction to the network. This can be done without Crypto Gateway by sending transaction requests directly to public nodes. But this approach has its disadvantages:

- You need to make sure that the nodes always remain synchronized;
- We must know in advance the IP addresses of the nodes and have stable access to them;
- IP addresses of nodes can be attacked;

- Difficulties in working with commissions. To work correctly with commissions in some blockchains, you need to monitor commissions in transactions and blocks. Most Crypto Gateway services provide the ability to get current commission values.

Getting history

To work with history in bitcoin like blockchains, it is necessary to track all changes by address. CryptoAPI indexes history by the user's public address, simplifying and speeding up the user's work with the application.

Push notifications

Special Apple and Google services are responsible for delivering push notifications to the mobile application:

- Apple Push Notification Service (APNS) - responsible for delivering push notifications to iOS;
- Google Cloud Messaging (GCM) and Firebase Cloud Messaging (FCM) deliver push notifications to Android.

The algorithm for working with notifications is as follows:

1. Special keys are created in Firebase Console (for Android) and Apple Developer Portal (for iOS) to identify the mobile application in APNS and FCM; These keychains are used in both the app and the mobile backend;

2. The application gets the user rights to work with notifications;
3. The application sends a request for registration in APNS or FCM services and receives a unique push token;
4. The app sends a unique push token to the mobile backend. So that in the future, you can find push tokens that correspond to a specific user of the system;
5. The mobile backend waits for events to appear in the system that requires sending push notifications;
6. Mobile backend generates push notifications and sends them to APNS and FCM;
7. APNS and FCM deliver notifications to users.

In our case, Crypto Gateway should act as a mobile backend, so it monitors changes in the network even when the client's application is not running.
CRYPTO_GATEWAY_NAME supports sending push notifications. Together with the push token, the application transmits a list of addresses for monitoring changes in the network.

Security of the solution

Private key storing

During development, we minimize key storage in any form within the application. Therefore, the application uses a password that encrypts the mnemonic / passphrase, from which a private key is subsequently generated. In doing so, we do not store the password, private key and the original passphrase. Remembering the password remains the user's task, and if it is necessary to sign a transaction, we ask the user for the password (in order to decrypt the passphrase and generate a private key from it, see transaction signing below).
Instead of a password, the phone's biometric capabilities (FaceID, Fingerprint) with the protection implementation at the operating system level (iOS, Android) can be used, while we still do not store the private key and the original passphrase.
Besides not storing the password and keys between sessions, we minimize their placement in the phone RAM during execution. After each operation that requires a private key, we forcibly overwrite it in RAM.
An additional degree of protection is the storage of encrypted data in special system storages of private data. (Keychain: iOS, Android). Access to them is provided and checked by the system itself and introduces additional protection levels, such as encryption and access restriction from the outside.

Transaction processing

The process of assembling a transaction happens strictly locally in the application. Transaction signature also happens locally before sending the signed transaction on the network. Therefore, the entire process does not depend on third-party services and can take place offline.
But to send a signed transaction to the network, you need a blockchain node or a Crypto Gateway such as CRYPTO_GATEWAY_NAME.
In order to sign a transaction, we ask the user to enter the application password. We take his hash, and compare with the password hash, which we have saved. If they are the same, then this is our password.
Next, we retrieve the encrypted mnemonic / passphrase and decrypt it with a password. The next step is signing the transaction. Then we forcibly nullify the open mnemonic / passphrase and the in-memory password. We send the transaction to the network.

Services integration

An essential condition for working with third-party services is a TLS / SSL connection. This is the only acceptable way to ensure the integrity, correctness and security of the transmitted data.
TLS / SSL enables client-server applications to communicate over the network so that no sniffing or unauthorized access can be made.

Scalability

The increase in the number of users does not directly affect the resiliency of applications.

End-user applications are delivered as compiled binaries through the system app stores. Compatibility of new and old versions of applications is carried out through the versioning of the API’s with which this application works.

Quality Assurance testing strategy

Testing stages

Iteration	Tasks	Documentation
Level 1 Exploring	▪ Introduction to the project ▪ Specification testing ▪ Defining test types ▪ Selection of testing tools ▪ Definition of acceptance criteria	
Level 2	▪ Creating a test suite in Testrail ▪ Providing test coverage	Link to scripts

Iteration	Tasks	Documentation
Preparation	▪ Designing test scripts	
Level 3 Main test	▪ Smoke testing in the developer branch ▪ Passing test runs in the master branch ▪ Test status indication ▪ Bug reporting in Jira	Link to testing status
Level 4 Test result analysis	▪ Upload test report ▪ Defect Analysis	Link to report
Level 5 Stabilization (master)	▪ Checking bug fixes ▪ Regression testing ▪ Acceptance testing	

Targets and goals

Testing tasks	Testing target
▪ Providing testing of intermediate versions of the developed software product ▪ Providing system testing ▪ Identification of inconsistencies of the implemented functionality with the declared requirements ▪ Minimizing the number of defects in the final product ▪ Providing information about the quality of software to the end customer	▪ Detailed study of the subject area ▪ Conducting an audit of the specification and requirements for the completeness and relevance of the data ▪ Development of test documentation ▪ Product testing: a. Performing checks based on pre-prepared test scenarios b. Intuitive simulation of situations that may arise in the operating environment of the software ▪ Classification of detected errors and their entry into the bug tracking system ▪ Team notification of all detected defects ▪ Control of the elimination process of identified defects ▪ Testing analysis. Drawing up reports on the results of inspections

Defect Severity Definitions

Blocking error	Critical	Major	Minor	Trivially

A problem which causes the product to crash and/or hang the system	A feature or function, which does not work correctly and has no obvious work around	A feature or function, which does not work correctly, but has an intuitive work around	A problem that has no significant impact on functionality or performance	Minor visual defect not noticeable to the user

Main test

Quality Assurance	
Specification	
Design	
Defect management method	The tools for registering and tracking the current state of defects are: ▪ Atlassian Jira ▪ Test Case Manager - TestRail
Testing types	1. Functional ▪ Smoke testing ▪ Critical Path testing ▪ New feature testing ▪ Confirmation testing ▪ Regression testing 2. Non-functional: ▪ Installation testing ▪ UI testing ▪ Usability testing ▪ Configuration testing ▪ Compatibility testing ▪ Localisation testing ▪ Acceptance testing

Quality Assurance	
By the positivity of the scenario	A. Positive test B. Negative test
Test configuration description	Devices ▪ Android ▪ iOS
Test Documentation	▪ Test Plan ▪ Test Cases ▪ Check Lists ▪ Bug Reports ▪ Test Reports ▪ Mind Maps
Test Tools	▪ Jira ▪ Testrail ▪ Charles Proxy ▪ Swagger ▪ Postman ▪ Android-Studio ▪ Vysor ▪ Firebase

Quality Assurance	
Testing methods	▪ Black Box testing. Functional and non-functional testing without access to the internal structure of system components ▪ Grey Box testing Testing using access to the internal structure and algorithms of the software
Product quality criteria	▪ The functionality must fully cover all the requirements set out by the client ▪ The software product must not have known defects with the status "Critical" and "Major" at the time of delivery of the software product to the customer

Test scripts

Story	Link (expect)
Start loading + Registration	Link
Wallet portfolio	Link
Send/Receive	Link
BlackFort Scan	Link
Swap	Link
BXN Node dashboard	Link
Settings	Link
User Profile	Link
Notifications	Link
OPTIONS for integrating new blockchains	Link
NFT Section	Link
Decentralized Browser	Link

Schedule of internal team events	- Mo-Fr 11:00 - Daily Meeting - Every second Tuesday of the month - Project Overview & Retrospective - Every second Wednesday of the month - Planning Meeting & Backlog Grooming

Types of tests and test environment

- Installation testing - Smoke testing - AD-HOC - Compatibility	Testing on DEV
- Installation testing - New feature testing - Running Tests by Case - Testing fixed bugs - Regression testing - UI testing - Usability testing - Localisation testing - Acceptance testing	Testing on Master
- Installation testing - Configuration testing - Regression testing - Beta ? (On the client side)	Test on PROD

BlackFort :: Tech design

Overview

Section description

This section contains a top-level description of the system. That is the description of the system as a whole without going into details. It includes the features of the system.

System description

The system is a mobile client application for the Blackfort platform. The tasks of the client application include creation/import of a crypto wallet, user registration in the system, the ability to send/receive cryptocurrencies, cryptocurrency exchange (swap), NFT collections, DApps, KYC (out of MVP scope), cryptocurrency rate display, scanning QR-code addresses, BXN dashboard, day/night mode, 2FA, confirmation of operations through biometrics, support chat, user profile management, support for transaction status notifications, support for multiple languages.

System features

The system includes the following features:

- Onboarding screen - familiarizing the user with the capabilities of the system;
- Wallet creation - creating a cryptocurrency wallet;
- Wallet import - importing the user's existing crypto wallet;
- Wallet portfolio - displaying wallet balance, displaying transaction history, adding tokens, displaying a list of cryptocurrencies;
- Blackfort account registration - registration of an account in the Blackfort system with cryptocurrency wallet linkage;
- Send crypto - sending cryptocurrency to the destination address;
- Receive crypto - receiving cryptocurrency by QR-code with the ability to specify the amount;
- Price chart - market cap, Blockchain Explorer;
- BlackFort scan - QR-code scanner and Wallet connect;
- Swap - cryptocurrency exchange with the possibility of fiat exchange;
- BXN dashboard;
- User profile - managing personal data, logout, deleting wallet, deleting account, listing bank accounts;
- Notifications - display of transaction statuses via push notifications;
- NFT section - viewing collection list, adding, sending, receiving, or minting NFT;
- Decentralized browser - viewing list of dapps, confirming/rejecting transactions;
- KYC (out of MVP scope) - verifying the user’s identity.

Architecture

Section description

Contains the architecture of the system (the relationship of the system modules). Also describes the interaction of the system with external services.

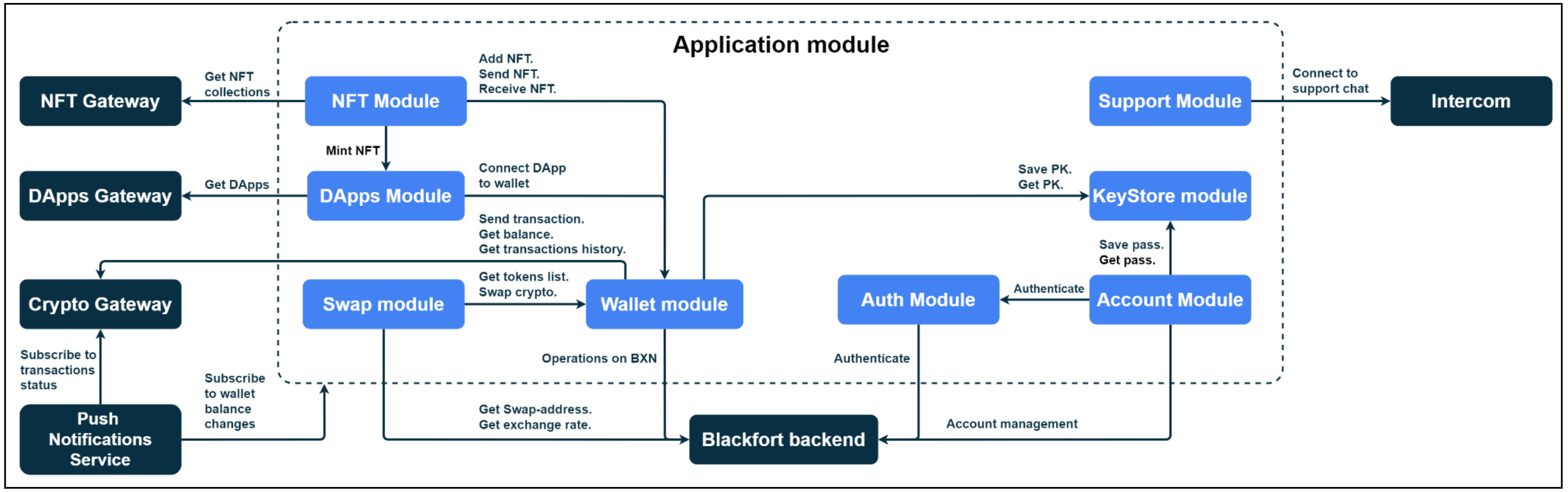
Rationale

For this project, a clean architecture with partitioning into modules was chosen. This architecture will allow weakly coupled code, which will eventually lead to the flexibility of changing and replacing defective modules. Also, such an architecture is relatively easy to test and maintain.

Notes

It is worth noting that in class diagrams, some methods for data layer implementation classes are omitted because they repeat methods of the parent class and do not carry additional information. Also, some connections to models are omitted due to the obvious dependence/use of them + not to clutter up the diagram.

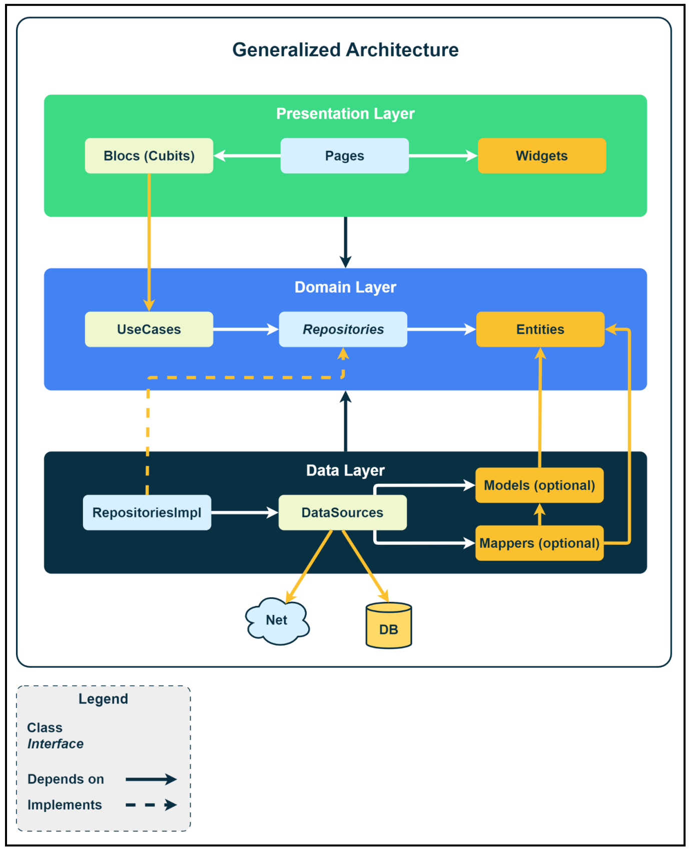
Modules diagram



Pic. 1 - Application modules diagram.

App architecture diagrams

Generalized module architecture



Pic 2. - Diagram of a generalized module architecture.

Detailed diagram

Since a detailed diagram takes up a lot of space, it's not possible to place it here. You can see the diagram at the following link: It's also worth noting that the presentation layer isn't detailed because it's the most prone to change.

Modules

Section description

Contains a description of the modules (components) of the system to be implemented. The description includes the module's purpose, features and dependencies (packages, libraries, submodules).

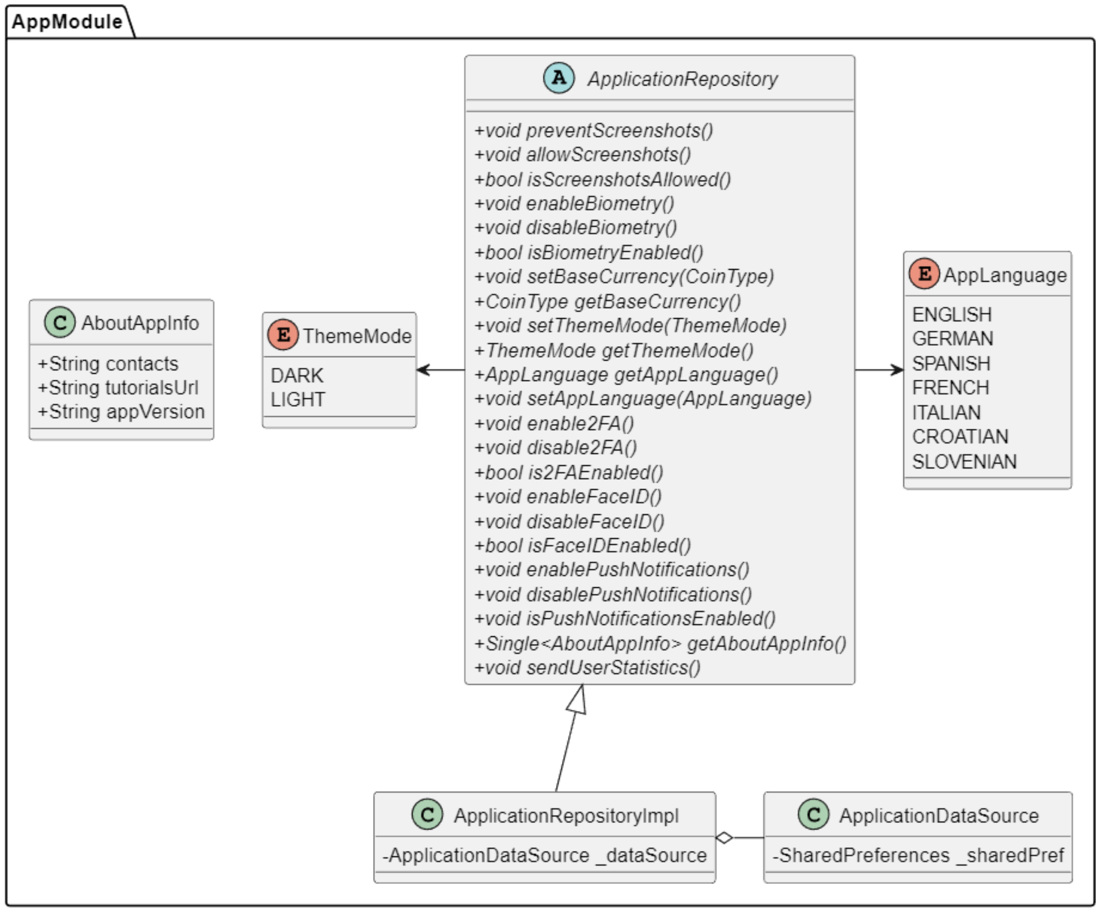
Application module

Description
The main task of this module is to coordinate the other modules of the application.

Features
The tasks of this module include the following:

- Dependency injection;
- Screen navigation;
- State management;
- Localization;
- Settings;
- Analytics.

Class Diagram


AppModule
ApplicationRepository
+void preventScreenshots()
+void allowScreenshots()
+bool isScreenshotsAllowed()
+void enableBiometry()
+void disableBiometry()
+bool isBiometryEnabled()
+void setBaseCurrency(CoinType)
+CoinType getBaseCurrency()
+void setThemeMode(ThemeMode)
+ThemeMode getThemeMode()
+AppLanguage getAppLanguage()
+void setAppLanguage(AppLanguage)
+void enable2FA()
+void disable2FA()
+bool is2FAEnabled()
+void enableFaceID()
+void disableFaceID()
+bool isFaceIDEnabled()
+void enablePushNotifications()
+void disablePushNotifications()
+void isPushNotificationsEnabled()
+Single<AboutAppInfo> getAboutAppInfo()
+void sendUserStatistics()
AboutAppInfo
+String contacts
+String tutorialsUrl
+String appVersion
ThemeMode
DARK
LIGHT
AppLanguage
ENGLISH
GERMAN
SPANISH
FRENCH
ITALIAN
CROATIAN
SLOVENIAN
ApplicationRepositoryImpl
-ApplicationDataSource _dataSource
ApplicationDataSource
-SharedPreferences _sharedPref

Auth module

Description

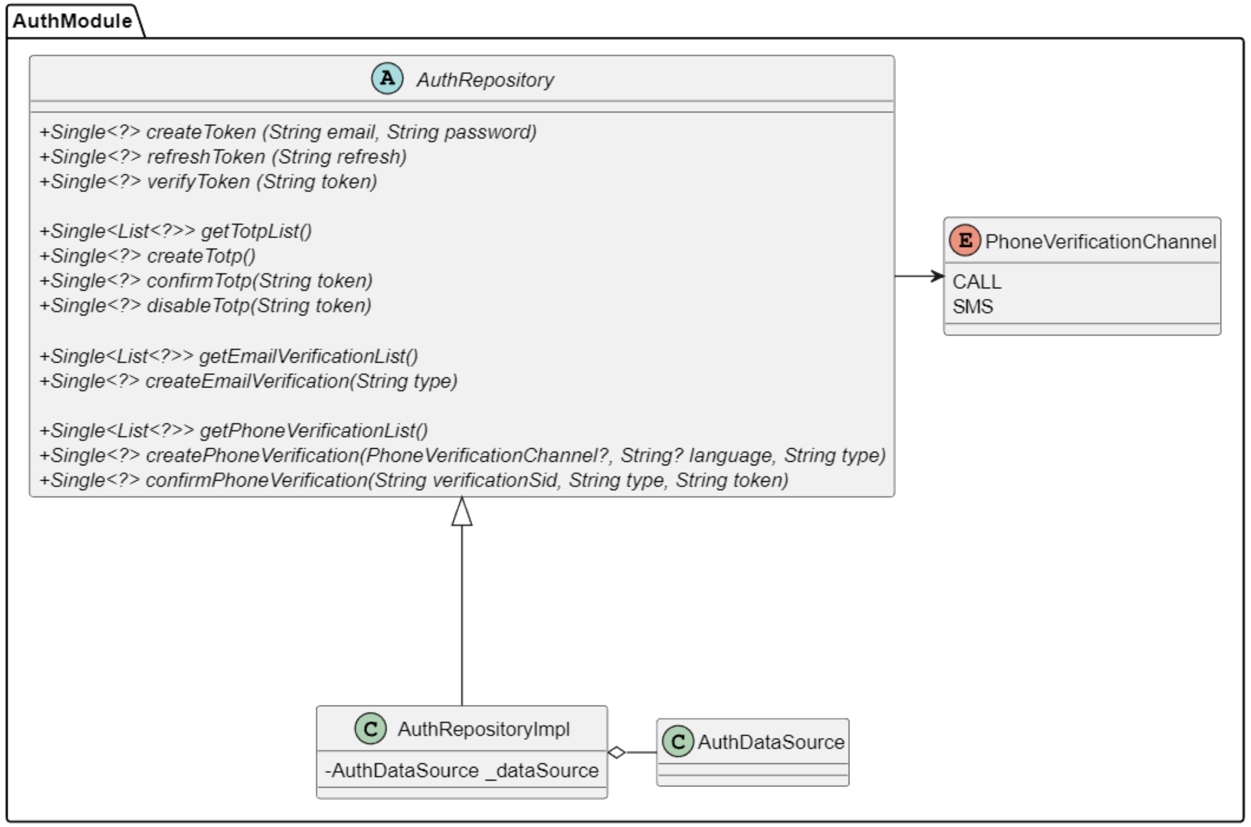
This module is designed for user registration, authentication and authorization.

Features

This module provides the following features:

- Authentication (Boyard);
- Authorization;
- KYC (out of MVP).

Class Diagram


AuthModule
AuthRepository
+Single<?> createToken (String email, String password)
+Single<?> refreshToken (String refresh)
+Single<?> verifyToken (String token)
+Single<List<?>> getTotpList()
+Single<?> createTotp()
+Single<?> confirmTotp(String token)
+Single<?> disableTotp(String token)
+Single<List<?>> getEmailVerificationList()
+Single<?> createEmailVerification(String type)
+Single<List<?>> getPhoneVerificationList()
+Single<?> createPhoneVerification(PhoneVerificationChannel?, String? language, String type)
+Single<?> confirmPhoneVerification(String verificationSid, String type, String token)
PhoneVerificationChannel
CALL
SMS
AuthRepositoryImpl
-AuthDataSource _dataSource
AuthDataSource

Dependencies

Authentication:

KYC: BasisID

Account module

Description

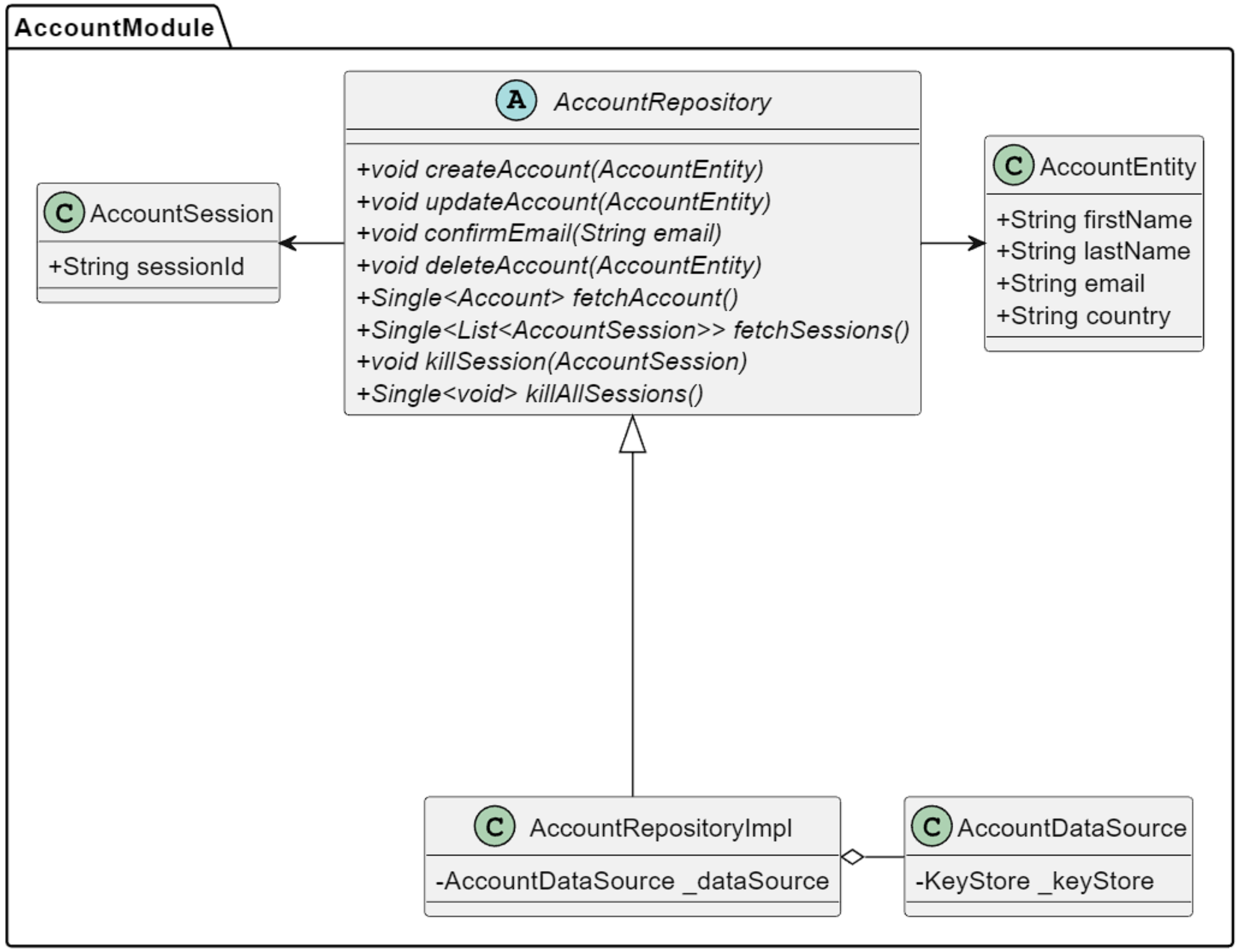
This module is designed for user account creation, editing and other management features.

Features

This module provides the following features:

- Account creation;
- Account management;
- Account deletion.

Class Diagram


AccountModule
AccountRepository
+void createAccount(AccountEntity)
+void updateAccount(AccountEntity)
+void confirmEmail(String email)
+void deleteAccount(AccountEntity)
+Single<Account> fetchAccount()
+Single<List<AccountSession>> fetchSessions()
+void killSession(AccountSession)
+Single<void> killAllSessions()
AccountSession
+String sessionId
AccountEntity
+String firstName
+String lastName
+String email
+String country
AccountRepositoryImpl
-AccountDataSource _dataSource
AccountDataSource
-KeyStore _keyStore

Wallet module

Description

The module is responsible for working with cryptocurrency wallets. And also for operations with cryptocurrencies.

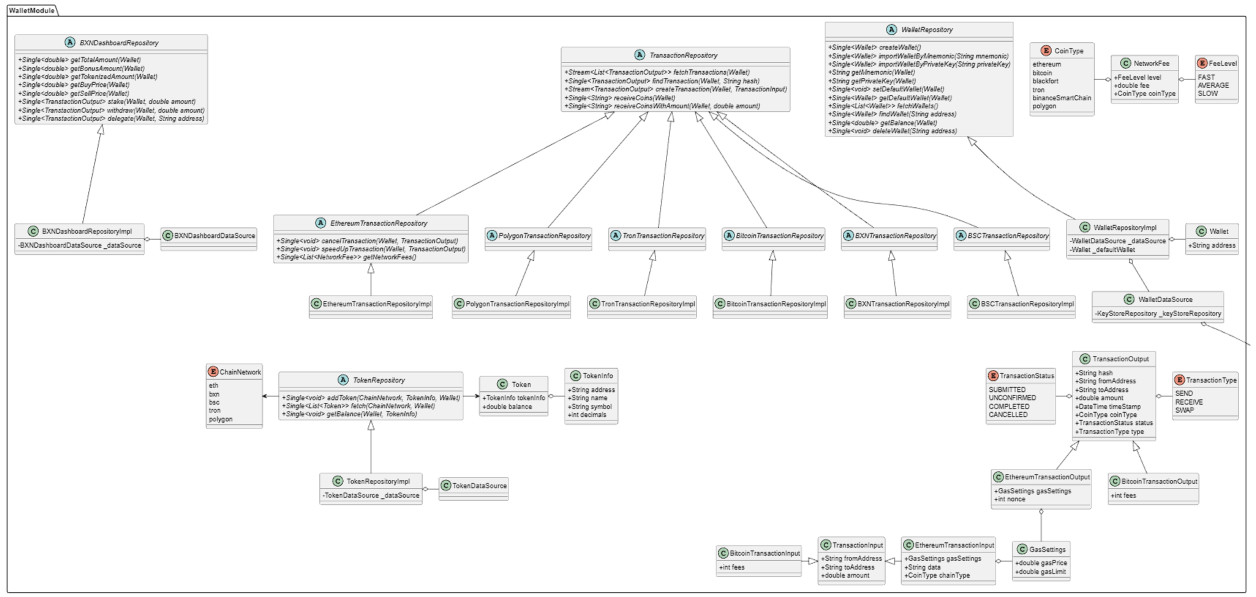
Features

This module provides the following features:

- Create wallet;
- Import wallet;
- Get wallet balance;
- Provide transactions history;
- Send crypto;
- Cancel transaction;
- Provide wallets list;
- Set default wallet;
- Get a default wallet.

Class diagram

wallet_module_class_diagram.png


WalletModule
BXNDashboardRepository
+Single<double> getTotalAmount(Wallet)
+Single<double> getBonusAmount(Wallet)
+Single<double> getTokenizedAmount(Wallet)
+Single<double> getBuyPrice(Wallet)
+Single<double> getSellPrice(Wallet)
+Single<TranstactionOutput> stake(Wallet, double amount)
+Single<TranstactionOutput> withdraw(Wallet, double amount)
+Single<TranstactionOutput> delegate(Wallet, String address)
BXNDashboardRepositoryImpl
-BXNDashboardDataSource _dataSource
BXNDashboardDataSource
TransactionRepository
+Stream<List<TransactionOutput>> fetchTransactions(Wallet)
+Single<TransactionOutput> findTransaction(Wallet, String hash)
+Stream<TransactionOutput> createTransaction(Wallet, TransactionInput)
+Single<String> receiveCoins(Wallet)
+Single<String> receiveCoinsWithAmount(Wallet, double amount)
WalletRepository
+Single<Wallet> createWallet()
+Single<Wallet> importWalletByMnemonic(String mnemonic)
+Single<Wallet> importWalletByPrivateKey(String privateKey)
+String getMnemonic(Wallet)
+String getPrivateKey(Wallet)
+Single<void> setDefaultWallet(Wallet)
+Single<Wallet> getDefaultWallet(Wallet)
+Single<List<Wallet>> fetchWallets()
+Single<Wallet> findWallet(String address)
+Single<double> getBalance(Wallet)
+Single<void> deleteWallet(String address)
CoinType
ethereum
bitcoin
blackfort
tron
binanceSmartChain
polygon
NetworkFee
+FeeLevel level
+double fee
+CoinType coinType
FeeLevel
FAST
AVERAGE
SLOW
EthereumTransactionRepository
+Single<void> cancelTransaction(Wallet, TransactionOutput)
+Single<void> speedUpTransaction(Wallet, TransactionOutput)
+Single<List<NetworkFee>> getNetworkFees()
PolygonTransactionRepository
TronTransactionRepository
BitcoinTransactionRepository
BXNTransactionRepository
BSCTransactionRepository
WalletRepositoryImpl
-WalletDataSource _dataSource
-Wallet _defaultWallet
Wallet
+String address
EthereumTransactionRepositoryImpl
PolygonTransactionRepositoryImpl
TronTransactionRepositoryImpl
BitcoinTransactionRepositoryImpl
BXNTransactionRepositoryImpl
BSCTransactionRepositoryImpl
WalletDataSource
-KeyStoreRepository _keyStoreRepository
ChainNetwork
eth
bxn
bsc
tron
polygon
TokenRepository
+Single<void> addToken(ChainNetwork, TokenInfo, Wallet)
+Single<List<Token>> fetch(ChainNetwork, Wallet)
+Single<void> getBalance(Wallet, TokenInfo)
Token
+TokenInfo tokenInfo
+double balance
TokenInfo
+String address
+String name
+String symbol
+int decimals
TokenRepositoryImpl
-TokenDataSource _dataSource
TokenDataSource
TransactionStatus
SUBMITTED
UNCONFIRMED
COMPLETED
CANCELLED
TransactionOutput
+String hash
+String fromAddress
+String toAddress
+double amount
+DateTime timeStamp
+CoinType coinType
+TransactionStatus status
+TransactionType type
TransactionType
SEND
RECEIVE
SWAP
EthereumTransactionOutput
+GasSettings gasSettings
+int nonce
BitcoinTransactionOutput
+int fees
BitcoinTransactionInput
+int fees
TransactionInput
+String fromAddress
+String toAddress
+double amount
EthereumTransactionInput
+GasSettings gasSettings
+String data
+CoinType chainType
GasSettings
+double gasPrice
+double gasLimit

DApps module

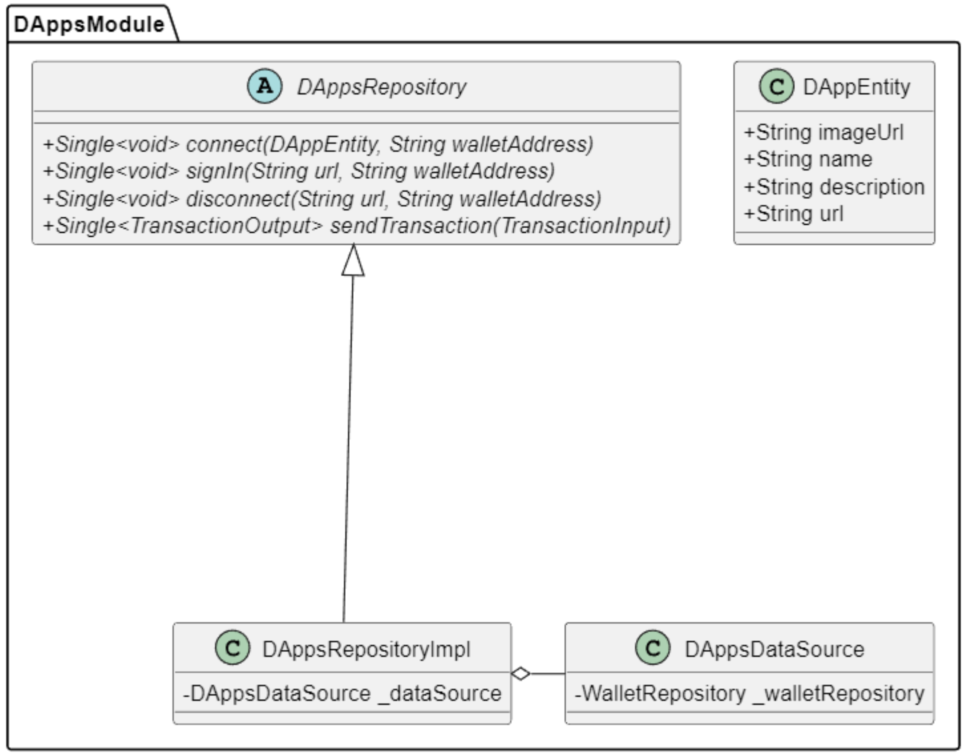
Description

Web3 browser to work with DApps.

Features

- View a list of DApps;
- Connect to DApps;
- Clear history and cookies;
- Refresh page;
- Share link;
- Bookmarks;
- Confirm/reject transactions.

Class Diagram


DAppsModule
DAppsRepository
+Single<void> connect(DAppEntity, String walletAddress)
+Single<void> signIn(String url, String walletAddress)
+Single<void> disconnect(String url, String walletAddress)
+Single<TransactionOutput> sendTransaction(TransactionInput)
DAppEntity
+String imageUrl
+String name
+String description
+String url
DAppsRepositoryImpl
-DAppsDataSource _dataSource
DAppsDataSource
-WalletRepository _walletRepository

Dependencies

Browser: web3dart, inappwebview, webview_flutter

NFT module

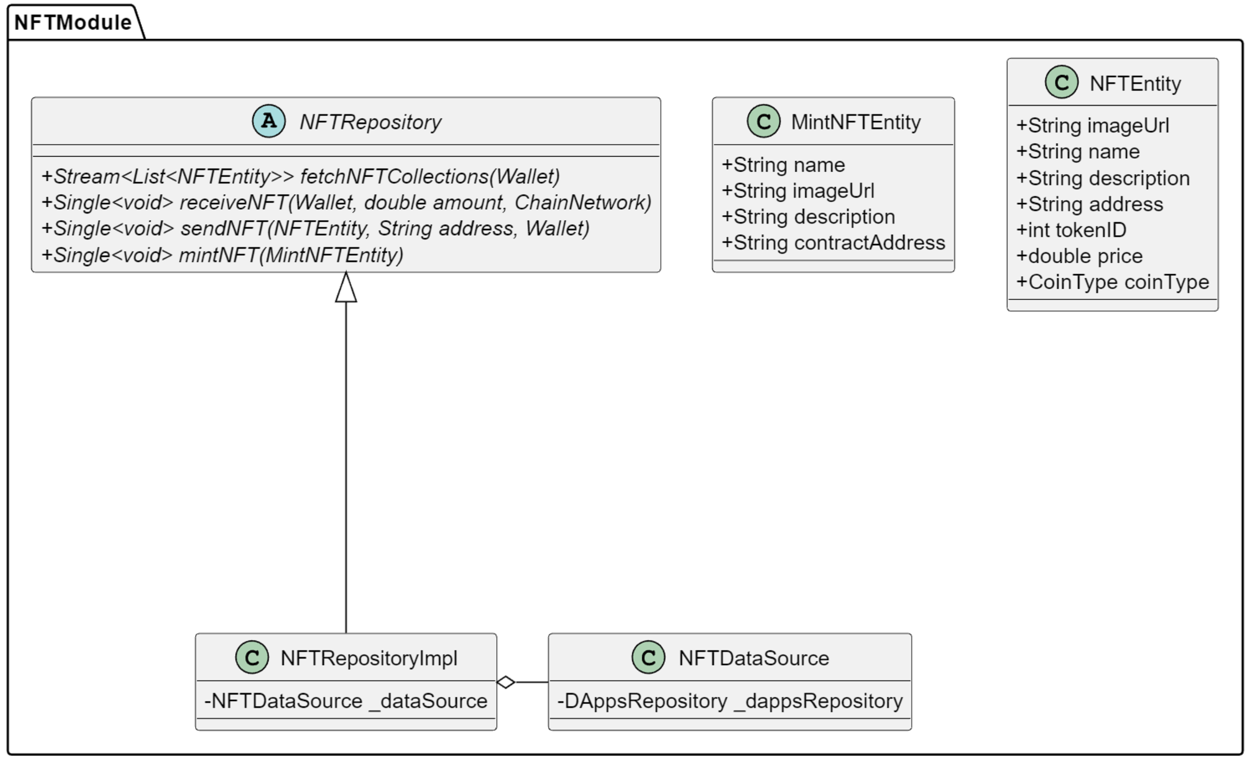
Description

The module is responsible for working with user NFT collections.

Features

- View the list of NFT collections;
- Send NFT;
- Receive NFT;
- Mint NFT.

Class Diagram


NFTModule
NFTRepository
+Stream<List<NFTEntity>> fetchNFTCollections(Wallet)
+Single<void> receiveNFT(Wallet, double amount, ChainNetwork)
+Single<void> sendNFT(NFTEntity, String address, Wallet)
+Single<void> mintNFT(MintNFTEntity)
MintNFTEntity
+String name
+String imageUrl
+String description
+String contractAddress
NFTEntity
+String imageUrl
+String name
+String description
+String address
+int tokenID
+double price
+CoinType coinType
NFTRepositoryImpl
-NFTDataSource _dataSource
NFTDataSource
-DAppsRepository _dappsRepository