



Tonex

FreeTon browser extension

STAGE 1

tonexwallet.com



Tonex Wallet

FreeTON browser extension



Phillip

DEVELOPER



Serge

DESIGNER

We are a small team of designer and developer and we are happy to present our browser extension solution for interacting with the FreeTON blockchain — Tonex. This document describes the basic principles of work and features of the project.

We have developed base for a future full-fledged project. Special attention was paid to the architecture and design of the application.

Main Principles:

💎 Simple interface and basic flows coverage:

- creating/ restoring a wallet
- activating a wallet
- making transactions
- browsing recent activity

💎 Accessibility to a wide audience.

The main idea of the extension is to provide to user simple way for using FreeTON blockchain. There are many useful tips in the app and FAQ page on the website.

💎 Store as little as possible.

We do not store redundant information which can be obtained from the blockchain. The extension is just a way to communicate between the browser and the blockchain.

💎 Manifest V3 support

Browsers are actively implementing Manifest V3 for extensions. Extensions supporting Manifest V3 are more efficient and secure, but they have some limitations. For example, there is no way to have a permanent script in the background.

More about Manifest V3:

<https://developer.chrome.com/docs/extensions/mv3/intro/>

Extension Architecture

There are three basic parts of our extension: Popup UI, Background Service Worker and Storage.

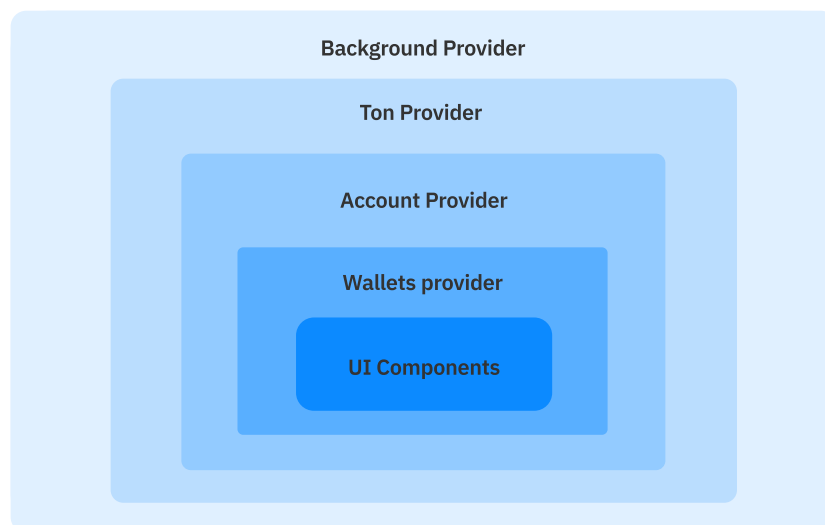


Popup UI

Popup UI is a window that opens when user click on extension icon. We decided to use React to build the interface, so it's easy to scale and maintain the product. This part runs only if extension popup is opened and destroys on close popup. To deliver data we use React Context providers connected to background service worker via [browser.runtime.connect](#).

Data providers:

BackgroundProvider — provides Background worker state and methods for messaging between extension Popup UI and Service Worker.



TonProvider — provides data about active network and methods for changing active network.

AccountProvider — provides account state and main methods like creating account, seed phrase generation, locking/unlocking and removing account.

WalletsProvider — provides list of available wallets and method for changing active wallet.

Background Service Worker

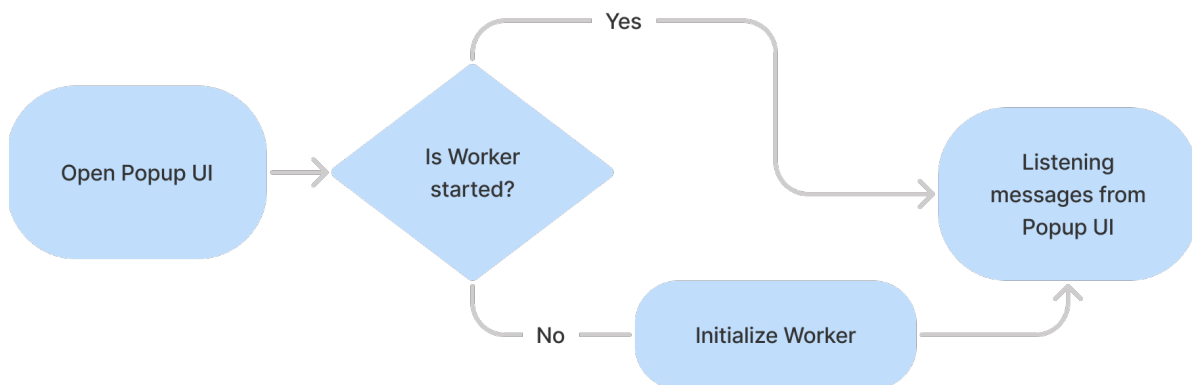
This is “backend” of extension. Since we use manifest V3 we don’t have persistent background script. Instead we use [Service Worker](#). All interactions with ton blockchain happens here.

i Currently TON SDK can not be started from another worker so we had to modify the TON SDK web library. We are going to make a more universal solution later.

Since we do not have a permanent background script we can’t receive updates from blockchain when the worker is stopped. However, we made notifications about recently submitted transactions even if extension is closed (but not locked) and worker is active.

However, this limitation of service worker can be an advantage for security. As soon as the Worker “goes to sleep” or user locks the wallet all in-memory data like keys and transactions data are removed from memory. Thus, the extension does not leave any information on which to trace the interaction with the wallet.

Worker starts when user opens popup and can be stopped by browser in any time when popup is closed. After starting the Worker initializes ton client and waits for messages from Popup UI.



Worker watches for wallet updates and provide actual data to Popup UI with messages. We will consider below all interaction flows between Worker and Popup UI.

Storage

Account data is stored by browser storage.local API. Sensitive data is encrypted with AES algorithm. Extension doesn't store wallet addresses, keys and activity history. This information derives dynamically when user decrypt or restore their seed phrase.

Here is an example of encrypted account object structure. Our principle of storing as little information as possible works here.

```
{
  "seedPhrase": "sniff orbit brand process arrest much ...",
  "paths": [
    "m/44'/396'/0'/0/0",
    "m/44'/396'/0'/0/1"
  ],
  "activeWalletPath": "m/44'/396'/0'/0/0"
}
```

When a user creates an account we save generated Seed Phrase.

We also keep a list of derived `paths`, so that we can then generate a list of user's wallets. When a user changes active wallet we save his choice like `activeWalletPath`. This data is not sensitive, but we encrypt it anyway for easier extraction.

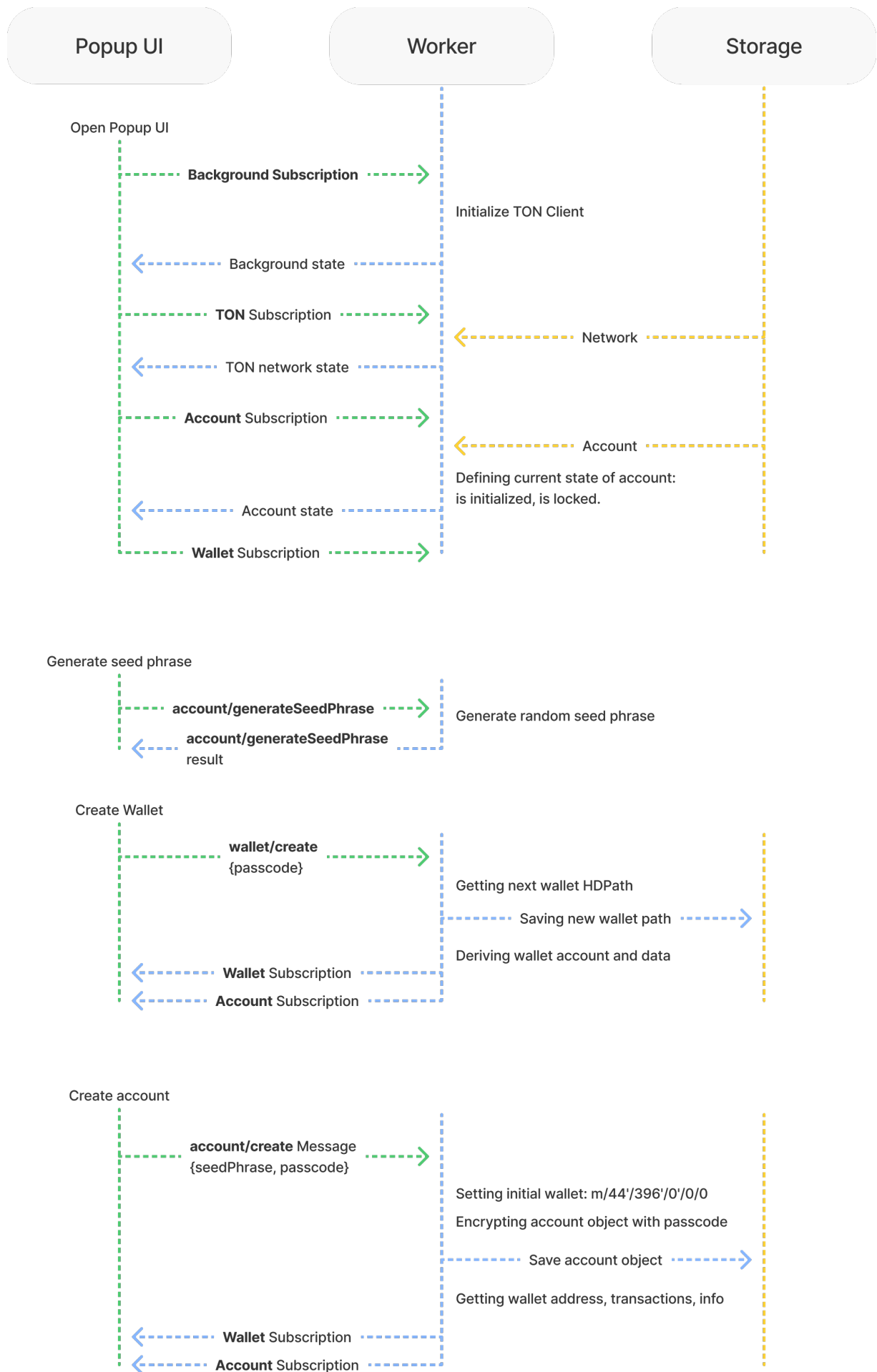
Other unencrypted data in browser storage.local can be used to store Popup UI state for UX optimization (for example to show last opened page, to keep form data etc.). Access to this data is provided directly from Popup UI avoiding Worker.

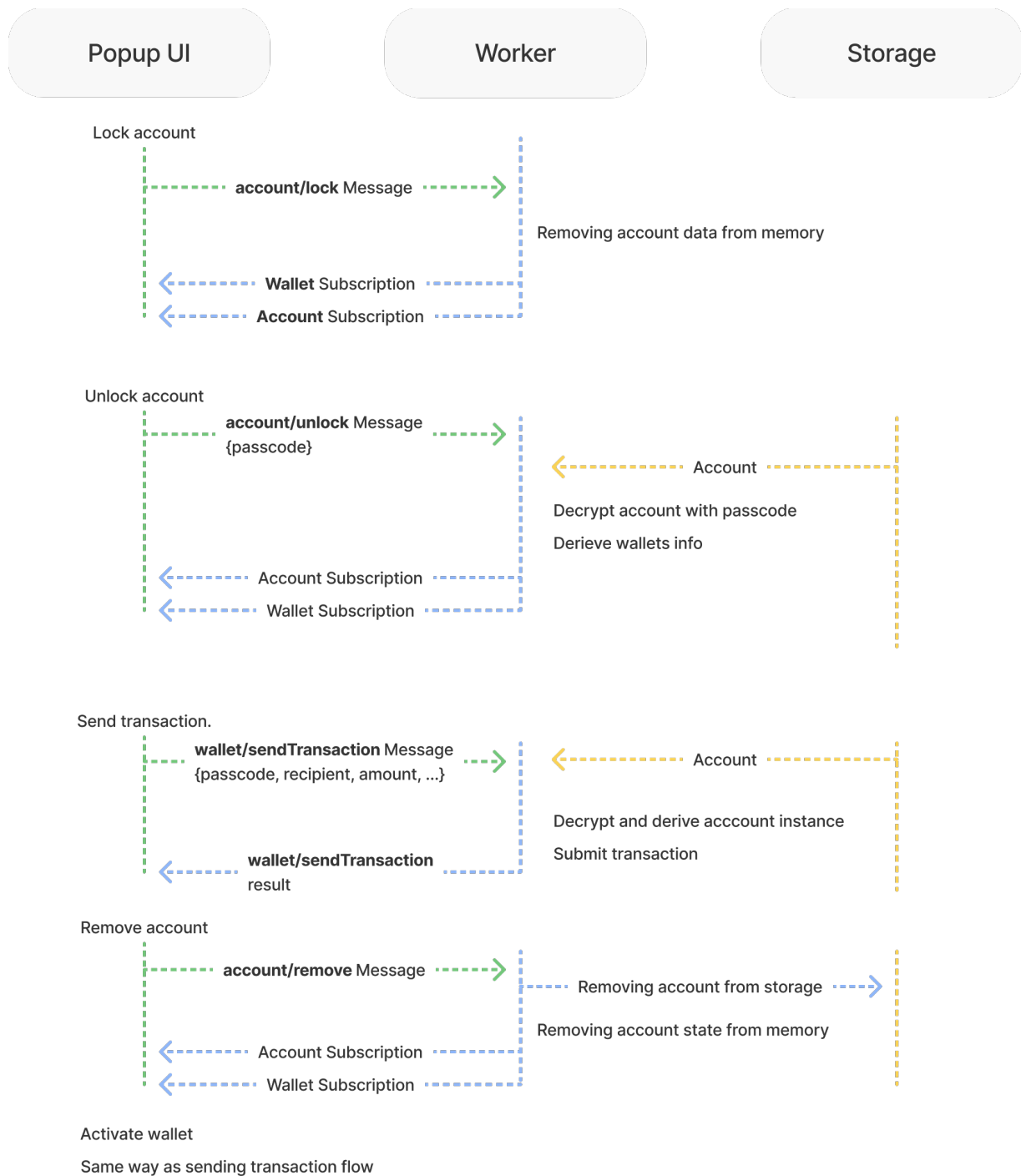
User flows

Now we will take a closer look at how these parts interact. As mentioned earlier we use browser.runtime.connect to communicate between Popup UI and Worker.

We have two types of events - subscriptions and post messages:

- the Popup UI subscribes to wallet updates, transactions etc. This is how actual data comes to Popup UI.
- using post messages Popup UI can initiate actions such as creating a wallet, sending transactions and other.





Installation

Releases of the extension are placed in our [GitHub](#) repository. Installation instructions can also be found there.

We are in the process of publishing an extension to the extension stores. Not all browsers supported and has tested yet. But it will be done as soon as possible.

Links and contacts



Write me in Telegram

<https://t.me/pavophilip>



Website

<https://tonexwallet.com>



Telegram Channel

<https://t.me/tonexwallet>



Design in Figma

<https://www.figma.com/file/l7T0tE97jK66GohwgFzbmm/Tonex>



Source code on GitHub

<https://github.com/TonexWallet/tonex-extension>

Wallet



0:92440067d283e136c170655609bf4ebf959696077ddf3af8f9dbb591d9305f1d