# Functional Requirements Document: HashPack Wallet Integration

**Author:** Manus AI
**Date:** August 5, 2025

## 1. Introduction

This Functional Requirements Document (FrD) outlines the technical specifications and integration guidelines for connecting a decentralized application (dApp) with the HashPack wallet on the Hedera network. The primary objective is to enable seamless wallet detection, connection, and subsequent navigation to a user-specific profile page within the dApp. This document is intended for developers with a foundational understanding of web development and blockchain concepts, providing detailed instructions to ensure correct and secure implementation.

## 2. Hedera Mainnet Architecture and Fundamentals

The Hedera mainnet is a public distributed ledger designed for high-throughput, low-latency, and fair transaction ordering. Unlike traditional blockchain architectures that rely on sequential blocks, Hedera utilizes a unique Directed Acyclic Graph (DAG) data structure known as the hashgraph, combined with the Gossip about Gossip protocol and virtual voting for consensus. This innovative approach ensures asynchronous Byzantine Fault Tolerance (aBFT), providing a secure, fast, and efficient network for decentralized applications [1].

### 2.1 Hashgraph Consensus Mechanism

The core of Hedera's architecture is the hashgraph consensus algorithm. This mechanism operates through a process called 'Gossip about Gossip,' where nodes randomly exchange information about their transactions and the transactions they've heard from others. This information is then organized into a hashgraph, a DAG that records the causal relationships between events. Virtual voting, a deterministic process, allows each node to determine the consensus order of transactions without requiring actual voting rounds, leading to high transaction throughput and finality [2].

### 2.2 Network Structure

The Hedera network comprises a set of consensus nodes, which are responsible for processing and ordering transactions. These nodes are currently permissioned but are on a

path to permissionless' decentralization model, aiming for broader participation over time. In addition to consensus nodes, the network also includes mirror nodes. Mirror nodes are not involved in consensus but store a complete history of transactions and states, providing a publicly accessible and queryable data source for dApps without burdening the consensus network [3].

### 2.3 Hedera Consensus Service (HCS)

The Hedera Consensus Service (HCS) is a key component that allows applications to achieve verifiable timestamps and ordering of events. HCS acts as a decentralized, trust-agnostic messaging layer, enabling applications to submit messages to the Hedera network for consensus. These messages are then ordered and timestamped by the consensus nodes, providing an immutable and auditable log of events. This service is particularly useful for applications requiring verifiable data integrity, such as supply chain tracking, token issuance, and verifiable logging [4].

## 3. HashPack Wallet Integration Methods

HashPack is a popular non-custodial wallet for the Hedera network, providing users with a secure way to manage their HBAR and other Hedera-based assets. For dApp developers, HashPack offers seamless integration through WalletConnect, a widely adopted open protocol for connecting decentralized applications to mobile wallets with QR code scanning or deep linking. HashPack's integration with WalletConnect allows for secure transaction signing and interaction with Hedera dApps without exposing private keys [5].

There are two primary methods for integrating HashPack with a dApp:

### 3.1 Using the `@hashgraph/hedera-wallet-connect` Library Directly

This method involves directly utilizing the `@hashgraph/hedera-wallet-connect` npm package, which serves as a messaging relay between dApps and Hedera wallets via the WalletConnect network. This library supports both Hedera native JSON-RPC and Ethereum JSON-RPC specifications, offering flexibility for developers building on Hedera's EVM-compatible layer or its native services.

### 3.1.1 Installation

To begin, install the necessary packages using npm:

Bash

```bash
npm install @hashgraph/hedera-wallet-connect@2.0.0-canary.811af2f.0
@hashgraph/sdk @walletconnect/modal
```

- `@hashgraph/hedera-wallet-connect` : The core library for connecting to Hedera wallets via WalletConnect.
- `@hashgraph/sdk` : The official Hedera JavaScript SDK, essential for interacting with the Hedera network.
- `@walletconnect/modal` : A UI library for displaying the WalletConnect modal, allowing users to scan a QR code or click a deep link to connect their wallet.

### 3.1.2 Initialization of DAppConnector

The `DAppConnector` class is central to establishing and managing the connection. It requires metadata about your dApp, the Hedera network ledger ID, a WalletConnect `projectId` , and lists of supported JSON-RPC methods and session events. The `projectId` is obtained from the WalletConnect Cloud and is crucial for routing messages through the WalletConnect network.

TypeScript

```typescript
import {
  HederaSessionEvent,
  HederaJsonRpcMethod,
  DAppConnector,
  HederaChainId,
} from '@hashgraph/hedera-wallet-connect';
```

```
import { LedgerId } from '@hashgraph/sdk';

const metadata = {
  name: 'Your DApp Name',
  description: 'A brief description of your dApp',
  url: 'https://your-dapp-url.com', // Must match your dApp's origin and
subdomain
  icons: ['https://your-dapp-url.com/logo.png'], // URL to your dApp's icon
};

// Replace 'YOUR_WALLETCONNECT_PROJECT_ID' with your actual Project ID from
WalletConnect Cloud
const projectId = 'YOUR_WALLETCONNECT_PROJECT_ID';

const dAppConnector = new DAppConnector(
  metadata,
  LedgerId.Mainnet, // Or LedgerId.Testnet for development
  projectId,
  Object.values(HederaJsonRpcMethod), // All supported Hedera JSON-RPC
methods
  [HederaSessionEvent.ChainChanged, HederaSessionEvent.AccountsChanged], //
Events to listen for
  [HederaChainId.Mainnet, HederaChainId.Testnet], // Supported Hedera chain
IDs
);

await dAppConnector.init({ logger: 'error' });
```

**Explanation of Parameters:**

- `metadata` : An object containing information about your dApp, which will be displayed to the user in their wallet during the connection process. This helps users identify the dApp they are connecting to.
- `LedgerId.Mainnet` (or `LedgerId.Testnet` ): Specifies the Hedera network your dApp intends to interact with. It's crucial to match this with the network the user's wallet is configured for.
- `projectId` : A unique identifier obtained from the WalletConnect Cloud. This ID is essential for establishing a connection through the WalletConnect relay servers.
- `Object.values(HederaJsonRpcMethod)` : Includes all Hedera-specific JSON-RPC methods that your dApp might use for transactions and queries. This informs the wallet about

the types of operations your dApp can perform.

- `[HederaSessionEvent.ChainChanged, HederaSessionEvent.AccountsChanged]` : An array of session events your dApp will subscribe to. `ChainChanged` notifies your dApp if the user switches networks in their wallet, and `AccountsChanged` notifies if the user switches accounts.
- `[HederaChainId.Mainnet, HederaChainId.Testnet]` : Defines the Hedera chain IDs your dApp supports. This allows the wallet to ensure compatibility and guide the user if they are on an unsupported network.

### 3.1.3 Connecting to a Wallet

Once the `DAppConnector` is initialized, you can trigger the connection process by calling `openModal()` . This will typically open a WalletConnect modal, presenting a QR code for mobile wallet users or a list of compatible desktop wallets.

```typescript
await dAppConnector.openModal();
```

Upon successful connection, the `dAppConnector` instance will manage the session, allowing your dApp to send transaction requests and receive responses from the user's HashPack wallet.

### 3.1.4 Handling Sessions, Events, and Payloads

After a connection is established, your dApp needs to handle various aspects of the WalletConnect session, including:

- **Session Management:** Maintaining the active session, handling disconnections, and re-establishing connections if necessary.
- **Event Listeners:** Subscribing to events like `HederaSessionEvent.AccountsChanged` and `HederaSessionEvent.ChainChanged` to react to user actions within their wallet (e.g., switching accounts or networks).

- **Payload Processing:** Sending transaction payloads to the wallet for signing and receiving signed transaction responses. The `DAppConnector` object provides methods for these interactions.

For detailed information on handling these aspects, refer to the official `@hashgraph/hedera-wallet-connect` documentation and examples, particularly the `DAppConnector` class methods.

## 3.2 Using Reown's AppKit

Reown's AppKit provides a higher-level abstraction for integrating with various wallets, including HashPack, by building on top of WalletConnect. This can simplify development for dApps that need to support multiple blockchain ecosystems.

### 3.2.1 Installation

Bash

```bash
npm install @hashgraph/hedera-wallet-connect@2.0.1-canary.24fffa7.0
@hashgraph/sdk @walletconnect/universal-provider
```

- `@walletconnect/universal-provider`: A WalletConnect package that provides a universal provider for various blockchain networks.

### 3.2.2 Configuration with `createAppKit`

When using Reown's AppKit, you configure it with adapters for the specific blockchains you want to support. For Hedera, you'll use `HederaAdapter`.

TypeScript

```typescript
import type UniversalProvider from '@walletconnect/universal-provider';

import {
  HederaProvider,
  HederaAdapter,
  HederaChainDefinition,
  hederaNamespace,
} from '@hashgraph/hedera-wallet-connect';

const metadata = {
  name: 'AppKit w/ Hedera Example',
  description: 'An example dApp using Reown AppKit with Hedera',
  url: 'https://example.com', // Must match your domain & subdomain
  icons: ['https://avatars.githubusercontent.com/u/179229932'],
};

const projectId = 'YOUR_WALLETCONNECT_PROJECT_ID'; // Your WalletConnect
Project ID

const hederaEVMAdapter = new HederaAdapter({
  projectId,
  networks: [
    HederaChainDefinition.EVM.Mainnet,
    HederaChainDefinition.EVM.Testnet,
  ],
  namespace: 'eip155',
});

const universalProvider = (await HederaProvider.init({
  projectId: projectId,
  metadata,
})) as unknown as UniversalProvider; // Type assertion to resolve potential
type mismatches

createAppKit({
  adapters: [ hederaEVMAdapter ],
  universalProvider,
  projectId,
  metadata,
  networks: [
    HederaChainDefinition.EVM.Mainnet,
    HederaChainDefinition.EVM.Testnet,
  ],
});

// Recommended: Add Hedera Native WalletConnect Adapter for full Hedera
support
const hederaNativeAdapter = new HederaAdapter({
  projectId,
  networks: [HederaChainDefinition.Native.Mainnet,
HederaChainDefinition.Native.Testnet],
  namespace: hederaNamespace, // 'hedera' as CaipNamespace,
});

createAppKit({
  adapters: [hederaEVMAdapter, hederaNativeAdapter],
  projectId,
  metadata,
  networks: [
    HederaChainDefinition.EVM.Mainnet,
    HederaChainDefinition.EVM.Testnet,
    HederaChainDefinition.Native.Mainnet,
    HederaChainDefinition.Native.Testnet,
  ],
});
```

**Explanation of AppKit Configuration:**

- `HederaProvider.init`: Initializes the WalletConnect Universal Provider specifically for Hedera.
- `HederaAdapter`: An adapter for Hedera, configured with the `projectId`, supported networks (EVM and/or Native), and the appropriate namespace ( `eip155` for EVM compatibility or `hederaNamespace` for native Hedera operations).
- `createAppKit`: The main function from Reown's AppKit to set up the multi-wallet integration. You pass the configured adapters, universal provider, project ID, metadata, and supported networks.

## 3.3 WalletConnect and User Authentication Flow

The user authentication flow with HashPack via WalletConnect is designed to be secure and user-friendly. The general steps are as follows:

1. **DApp Initiates Connection:** The dApp, upon a user action (e.g., clicking a

connect wallet button), initiates a connection request using the `DAppConnector` or `createAppKit`.
2. **WalletConnect Modal Presentation:** A WalletConnect modal or pop-up is displayed to the user. This modal typically presents a QR code for mobile wallet users to scan or offers deep linking options to open HashPack directly on their device. For desktop users, it might list compatible browser extensions.
3. **User Approval in HashPack:** The user interacts with their HashPack wallet (either the browser extension or mobile app) to approve the connection request. This is a critical security step where the user explicitly grants the dApp permission to interact with their Hedera account. HashPack ensures that private keys remain secure within the wallet and are never exposed to the dApp.

4. **Secure Session Establishment:** Upon user approval, a secure communication channel (session) is established between the dApp and HashPack via the WalletConnect relay servers. This session is encrypted and allows for the secure exchange of transaction requests and responses.
5. **Account Information Retrieval:** Once the session is established, the dApp can query the connected wallet for basic account information, such as the user's Hedera Account ID. This information is essential for personalizing the dApp experience.
6. **User Profile Data Access (HashPack Profile API):** To provide a richer, personalized user experience, the dApp can then leverage the HashPack Profile API. This API allows dApps to retrieve standardized user profile data that users have optionally set within their HashPack wallet. This data is token-gated, meaning it's associated with specific tokens, and is guaranteed to be unique, enhancing the authenticity and reliability of the profile information. The Profile API offers methods for:
* **Get Single Profile:** Retrieves the profile for a specific Hedera Account ID.
* **Get Multiple Profiles:** Retrieves profiles for a list of Hedera Account IDs.
* **Update Profile:** Allows the dApp to facilitate user updates to their profile information within HashPack, subject to user approval.
7. **Navigation to Profile Page:** After successfully retrieving the necessary account and profile data, the dApp can programmatically navigate the user to a dedicated profile page within the application. This page can then display the user's account ID, profile picture, username, and any other relevant information obtained from the HashPack Profile API.

### 3.3.1 Key Considerations for a Robust Authentication Flow

- **`projectId` Management:** Ensure your WalletConnect `projectId` is correctly configured and kept secure. This ID is fundamental for your dApp to communicate with the WalletConnect network.

- **`metadata` Accuracy:** The `metadata` object provided during `DAppConnector` initialization is crucial for user trust. It should accurately represent your dApp's name, description, URL, and icon, as this information is displayed to the user in their wallet during the connection prompt.

- **Network Handling:** Implement logic to detect and handle cases where the user's wallet is connected to a different Hedera network (e.g., Testnet instead of Mainnet) than your dApp expects. Prompt the user to switch networks if necessary.

- **Event Listeners:** Actively listen for `HederaSessionEvent.AccountsChanged` and `HederaSessionEvent.ChainChanged` events. These events are vital for maintaining a synchronized state between your dApp and the user's wallet. For example, if a user switches accounts in HashPack, your dApp should automatically update to reflect the new account.
- **Error Handling and User Feedback:** Provide clear and concise error messages to the user if the connection fails, is rejected, or if there are issues during transaction signing. Guide the user on how to resolve common problems.
- **Session Persistence:** Consider implementing session persistence to avoid requiring the user to reconnect their wallet on every visit. WalletConnect sessions can be re-established if the user's wallet is still connected.
- **Security Best Practices:** Always emphasize that your dApp does not have direct access to the user's private keys. All sensitive operations, such as transaction signing, occur securely within the HashPack wallet, with only the signed transaction being returned to the dApp.

## 4. Functional Requirements

This section details the functional requirements for integrating the HashPack wallet connect button and subsequent user profile navigation.

### 4.1 Wallet Connection

**FR-WC-001: Wallet Detection:** The system SHALL detect if the HashPack wallet extension (or mobile app via deep link) is installed and accessible in the user's environment.

**FR-WC-002: Connect Button Display:** A prominent

Connect Wallet" button SHALL be displayed to the user, prompting them to initiate the connection process.

**FR-WC-003: WalletConnect Modal:** Upon clicking the "Connect Wallet" button, the system SHALL display the WalletConnect modal, providing a QR code for mobile wallet users and a list of compatible desktop wallets, including HashPack.

**FR-WC-004: Connection Establishment:** The system SHALL establish a secure session with the user's HashPack wallet upon successful approval by the user.

**FR-WC-005: Account ID Retrieval:** Upon successful connection, the system SHALL retrieve the user's Hedera Account ID from the connected HashPack wallet.

**FR-WC-006: Disconnection:** The system SHALL provide a mechanism for the user to disconnect their wallet from the dApp.

### 4.2 User Profile Navigation

**FR-UPN-001: Profile Data Retrieval:** After successful wallet connection, the system SHALL use the retrieved Hedera Account ID to query the HashPack Profile API for the user's profile data.

**FR-UPN-002: Profile Page Navigation:** Upon successful retrieval of the user's profile data, the system SHALL automatically navigate the user to a dedicated profile page within the dApp.

**FR-UPN-003: Profile Data Display:** The profile page SHALL display the user's profile information, including but not limited to their username, profile picture, and any other available data from the HashPack Profile API.

**FR-UPN-004: Graceful Handling of Missing Profiles:** If a user has not set up a profile in their HashPack wallet, the system SHALL gracefully handle this scenario, for example, by displaying a default profile page with a prompt to create a profile in HashPack.

## 5. Non-Functional Requirements

**NFR-001: Security:** All communication between the dApp and HashPack SHALL be encrypted and secure, adhering to WalletConnect protocol standards. Private keys MUST never be exposed to the dApp.

**NFR-002: Performance:** The wallet connection and profile data retrieval process SHALL be fast and efficient, with minimal latency to ensure a smooth user experience.

**NFR-003: Usability:** The wallet connection process SHALL be intuitive and user-friendly, with clear instructions and feedback to the user.

**NFR-004: Reliability:** The wallet connection and profile data retrieval SHALL be reliable and handle potential errors gracefully, providing informative messages to the user.

## 6. References

[1] Hedera. (2025). *Mainnet*. Hedera Docs. Retrieved from https://docs.hedera.com/hedera/networks/mainnet

[2] Hedera. (2025). *Hashgraph Consensus Algorithm*. Hedera Docs. Retrieved from https://docs.hedera.com/hedera/core-concepts/hashgraph-consensus-algorithms

[3] Hedera. (n.d.). *What is the Hedera mirror network?* Hedera. Retrieved from https://hedera.com/learning/hedera-hashgraph/what-is-the-hedera-mirror-network

[4] Hedera. (n.d.). *Hedera Consensus Service*. Hedera. Retrieved from https://hedera.com/consensus-service

[5] HashPack. (n.d.). *Developers*. HashPack. Retrieved from https://www.hashpack.app/developers