```
const commonInfo: ChainInfo = {
  rpc: "undefined",
  rest: "undefined",
  chainId: "base",
  chainName: "XION Testnet",
  bip44: {
    coinType: 118,
  },
  bech32Config: {
    bech32PrefixAccAddr: "xion",
    bech32PrefixValAddr: "xionvaloper",
    bech32PrefixValPub: "xionvaloperpub",
    bech32PrefixAccPub: "xionpub",
    bech32PrefixConsAddr: "xionvalcons",
    bech32PrefixConsPub: "xionvalconspub",
  },
  stakeCurrency: xionCoin,
  currencies: [xionCoin],
  feeCurrencies: [xionCoin],
  features: ["cosmwasm"],
};

export const mainnetChainInfo: ChainInfo = {
  ...commonInfo,
  rpc: "https://rpc.xion-mainnet-1.burnt.com:443",
  rest: "https://api.xion-mainnet-1.burnt.com:443",
  chainId: "xion-mainnet-1",
  chainName: "XION Mainnet",
};
`
```

Ejemplo

En la aplicación de demostración proporcionada por la biblioteca xion.js, se importa ***AbstraxionContextProvider y se envuelve alrededor del componente raíz, lo que significa que toda la aplicación front-end puede usar la información de configuración de AbstraxionContext*** , por ejemplo, para que sepamos que la aplicación está configurada para conectarse a la **red de prueba.**

```
export default function RootLayout({
  children,
}: {
  children: React.ReactNode;
}): JSX.Element {
  return (
    <html lang="en">
      <body className={inter.className}>
        <AbstraxionProvider config={treasuryConfig}>
          {children}
        </AbstraxionProvider>
      </body>
    </html>
  );

}
```

Interfaz de AASigner

¿Qué es una firma?

En los sistemas de cadena de bloques, la **firma** es una tecnología crucial que garantiza la seguridad y la autenticidad de las transacciones. Es similar a firmar un contrato en papel, que sirve como prueba de que un usuario en particular inició una transacción.**Proceso de firma** : cuando inicia una transacción, el contenido de la misma se cifra con su clave privada, lo que genera una firma única. Esta firma garantiza que el contenido de la transacción no ha sido alterado y que solo el titular de la clave privada podría haberlo generado.**Verificación de firma** : otras partes (como nodos o mineros) utilizan su clave pública para verificar la transacción. La clave pública puede confirmar que la firma fue efectivamente generada por la clave privada, lo que garantiza que la transacción fue realmente iniciada por usted y no falsificada.
Para obtener más detalles sobre las firmas, consulte la sección 2.5 de “Conceptos básicos de Web3”.

Interfaz AASigner exclusiva de XION

En la biblioteca xion.js, hay un paquete relacionado con las firmas, ubicado en el directorio packages/signers . Proporciona varias implementaciones de firmas y utilidades relacionadas con **las Cuentas Abstractas (AA)** . Estas herramientas manejan diferentes tipos de lógica de firma y verificación. La biblioteca admite varios métodos de firma, como la firma directa (AADirectSigner), la firma de billetera Ethereum (AAEthSigner) y la firma JWT

(AAJWTSigner), y ofrece clases correspondientes para cada uno. Su relación se ilustra en el diagrama a continuación:

En este paquete, el archivo AASigner.ts proporciona una interfaz de firma unificada para gestionar la lógica de firma de distintos tipos de cuentas abstractas. Veamos el contenido del archivo AASigner.ts, ubicado en packages/signers/src/interfaces/AASigner.ts . Los componentes principales de este archivo incluyen:

```
import { AccountData, DirectSignResponse, OfflineDirectSigner }
from "@cosmjs/proto-signing";
import { SignDoc } from "cosmjs-types/cosmos/tx/v1beta1/tx";

export interface AAccountData extends AccountData {
  algo: AAAlgo;
}

export interface AASigner {
  readonly algo: AAAlgo;
  getAccounts(): Promise<readonly AAccountData[]>;
  getAccount(address: string): Promise<AAccountData>;
  signDirect(address: string, signDoc: SignDoc):
Promise<DirectSignResponse>;
}

// ... other methods
```

Desglosándolo:

1.**Importaciones** : el archivo comienza importando los tipos e interfaces necesarios, principalmente de la biblioteca @cosmjs. Estas importaciones proporcionan tipos de firmas y cuentas que son compatibles con el SDK de Cosmos.

2.**Interfaz** AAccountData : esta interfaz extiende la base AccountData, agregando una propiedad algo, que indica el algoritmo criptográfico utilizado por la cuenta para generar y verificar firmas.

3.**Interfaz** AASigner : este es el núcleo del archivo y define los métodos que cualquier firmante de cuenta abstracta debe implementar:

●algo: Una propiedad de solo lectura que especifica el algoritmo utilizado por el firmante.

●getAccounts(): un método asincrónico que devuelve una matriz de todas las cuentas disponibles.

●getAccount(address): un método asincrónico que devuelve información de la cuenta para una dirección específica.

●signDirect(address, signDoc) : un método asincrónico que firma un documento de transacción (SignDoc) utilizando la clave privada de la cuenta.

Resumen

La interfaz AASigner proporciona una abstracción unificada para diferentes tipos de firmantes (por ejemplo, AADirectSigner, AAEthSigner y AAJWTSigner). Simplifica la reutilización y la flexibilidad del código al ofrecer una estructura consistente y, al mismo tiempo, respaldar la lógica única de cada método de firma. Este diseño permite que las aplicaciones cambien fácilmente entre diferentes métodos de firma sin alterar el código subyacente que interactúa con el firmante.

El siguiente código muestra el contenido completo de la interfaz AASigner :

```
1import { AccountData, DirectSignResponse } from
"@cosmjs/proto-signing";
2import { SignDoc } from "cosmjs-types/cosmos/tx/v1beta1/tx";
3import { AAAlgo } from "./smartAccount";
4
5/**
6 * @extends AccountData
7 */
8export interface AAccountData extends AccountData {
9  readonly authenticatorId: number;
10  // This is the signer account address. Not the AA address. This
11  // address is recognized by the wallet and used to sign the
transaction
12  readonly accountAddress: string;
13  // The AA algorithm type
14  readonly aaalgo?: AAAlgo;
15}
16export abstract class AASigner {
17  /// The abstract account address of the signer
18  /// must be set by implementing class
19  abstractAccount: string | undefined;
20  accountAuthenticatorIndex: number | undefined;
21
22  constructor(abstractAccount: string) {
23    this.abstractAccount = abstractAccount;
24  }
25  /**
26   * This method is to be implemented by every class that
implements this interface
27   * it will be used by the client to create the transaction AA
signature
28   * required to verify the transaction on the chain
29   * This method should return a DirectSignResponse object but
only the signature field is required
```

```
30    * to be set
31    * @param _signerAddress the abstract account address to be used
as the signer
32    * @param signDoc the sign doc to be signed
33    * @returns
34    */
35   signDirect(
36     _signerAddress: string,
37     signDoc: SignDoc,
38   ): Promise<DirectSignResponse> {
39     // default
40     return Promise.resolve({
41       signed: signDoc,
42       signature: {
43         signature: "",
44         pub_key: {
45           type: "tendermint/PubKeySecp256k1",
46           value: new Uint8Array(),
47         },
48       },
49     });
50   }
51
52   /**
53    * This method is to be implemented by every class that
implements this interface
54    * it will be used by the client to get the account data of the
current abstract account
55    * the pubKey of the account data should be set to an empty
Uint8Array since it's not required
56    * and to declare it an AA
57    * @returns {AAccountData} of length 1
58    */
59   abstract getAccounts(): Promise<readonly AAccountData[]>;
60}
61
62// Default implementation for a signer class
63export class AADefaultSigner extends AASigner {
64   constructor(abstractAccount: string) {
65     super(abstractAccount);
66   }
67
68   getAccounts(): Promise<readonly AAccountData[]> {
69     throw new Error("Cannot get accounts from default signer");
70   }
71}
```

Implementación de firma directa

packages/signers proporciona implementaciones y funciones de utilidad relacionadas con los firmantes de cuentas abstractas (AA), incluida la interfaz AASigner, que maneja varias lógicas de firma y verificación. En esta sección, veamos una de las implementaciones de la interfaz AASigner: AADirectSigner .

AADirectSigner de XION

AADirectSigner se utiliza principalmente para la firma directa en el lado del cliente, sin necesidad de solicitudes de red adicionales. Su método signDirect toma una dirección (la cuenta de firma) y un objeto SignDoc (los datos que se van a firmar) y luego utiliza el objeto de firmante interno para ejecutar la firma.

```
export class AADirectSigner implements AASigner {
  constructor(
    private readonly signer: OfflineDirectSigner,
    private readonly client: AAClient
  ) {}

  // ... other methods ...

  async signDirect(address: string, signDoc: SignDoc):
Promise<DirectSignResponse> {
    const account = await this.getAccount(address);
    return this.signer.signArbFn(account.address, signDoc);
  }
}
```

AADirectSigner utiliza la función SignArbitraryFn de la interfaz OfflineDirectSigner para realizar firmas, y es compatible con **cuentas normales** y **abstractas** . Es adecuado para la mayoría de las billeteras y firmantes compatibles con Cosmos SDK.

Ejemplo de firma de AADirectSigner

Para probar AADirectSigner, podemos utilizar la aplicación de demostración de la biblioteca oficial xion.js desde el repositorio apps/demo-app . Los pasos para comenzar son:

1.**Clonar el repositorio** : git clone https://github.com/burnt-labs/xion.js.git

2.**Instalar dependencias** : Ejecute pnpm install en el directorio raíz.

3.**Construya los módulos** : ejecute pnpm run build --filter="*" en el directorio raíz.

4.**Inicie el proyecto** : navegue a apps/demo-app y ejecute pnpm run dev , luego acceda al proyecto a través de localhost:3001 .

Después de iniciar sesión con un correo electrónico, puede firmar cualquier contenido

```
import {
  DirectSignResponse,
  OfflineDirectSigner,
```

```
4  makeSignBytes,
5} from "@cosmjs/proto-signing";
6import { SignDoc } from "cosmjs-types/cosmos/tx/v1beta1/tx";
7import { AAccountData, AASigner } from "../interfaces/AASigner";
8import { getAAccounts } from "./utils";
9import { StdSignature } from "@cosmjs/amino";
10
11export type SignArbitraryFn = (
12  chainId: string,
13  signer: string,
14  data: string | Uint8Array,
15) => Promise<StdSignature>;
16
17/**
18 * This class is an implementation of the AASigner interface using
the DirectSecp256k1HdWallet
19 * or any other signer that implements the AASigner interface
20 * This class use would generally be with a wallet since it's method
of signing is the same as the
21 * DirectSecp256k1HdWallet. The only difference is that it makes sure
to replace the signer address
22 * with a valid wallet address (as to the abstract account address)
before signing the transaction.
23 *
24 * Note: instance variable abstractAccount must be set before any
signing
25 * @abstractAccount the abstract account address of the signer
26 * @signer the signer to be used to sign the transaction
27 * @implements AASigner
28 */
29export class AADirectSigner extends AASigner {
30  signer: OfflineDirectSigner;
31  accountAuthenticatorIndex: number;
32  indexerUrl: string;
33  signArbFn: SignArbitraryFn;
34
35  constructor(
36    initializedSigner: OfflineDirectSigner,
37    abstractAccount: string,
38    accountAuthenticatorIndex: number,
39    signArbFn: SignArbitraryFn,
40    indexerUrl?: string,
41  ) {
42    super(abstractAccount);
43    this.signer = initializedSigner;
44    this.accountAuthenticatorIndex = accountAuthenticatorIndex;
45    this.signArbFn = signArbFn;
46    this.indexerUrl =
47      indexerUrl ||
"https://api.subquery.network/sq/burnt-labs/xion-indexer";
48  }
```

```
49
50  async signDirect(
51    signerAddress: string,
52    signDoc: SignDoc,
53  ): Promise<DirectSignResponse> {
54    const signBytes = makeSignBytes(signDoc);
55    const signature = await this.signArbFn(
56      signDoc.chainId,
57      signerAddress,
58      signBytes,
59    );
60    return {
61      signed: signDoc,
62      signature: {
63        pub_key: {
64          type: "tendermint/PubKeySecp256k1",
65          value: "", // This doesn't matter. All we need is signature
below
66        },
67        signature: signature.signature,
68      },
69    };
70  }
71
72  async getAccounts(): Promise<readonly AAccountData[]> {
73    const accounts = await this.signer.getAccounts();
74    if (accounts.length === 0) {
75      return [];
76    }
77    if (this.abstractAccount === undefined) {
78      // we treat this class a a normal signer not an AA signer
79      return accounts.map((account) => {
80        return {
81          ...account,
82          authenticatorId: 0,
83          accountAddress: account.address,
84        };
85      });
86    }
87    return await getAAccounts(accounts, this.abstractAccount,
this.indexerUrl);
88  }
89}
```

Implementación del firmante JWT

¿Qué es JWT (JSON Web Token)?

JWT es un formato de token compacto, seguro y de estándar abierto basado en JSON, que se utiliza principalmente para transmitir datos firmados entre el **cliente** y **el servidor** . Se utiliza ampliamente en escenarios como autenticación, intercambio de información y gestión de sesiones.

En la autenticación, por ejemplo, un usuario inicia sesión con los datos de su cuenta y el **servidor** genera un JWT, que se devuelve al **cliente** . En cada solicitud posterior, el cliente envía el JWT en el encabezado de la solicitud. El servidor no necesita almacenar datos de la sesión, sino que simplemente verifica la firma JWT para confirmar la identidad del usuario. Esto reduce la carga de gestión de las sesiones de usuario en el servidor, lo que lo convierte en una solución madura y escalable para la autenticación de usuarios.

JWTSigner en XION

En XION, AbstractAccountJWTSigner aprovecha los JWT para las operaciones de verificación de identidad y firma dentro del entorno de la cadena de bloques. Así es como funciona:

●Un usuario inicia sesión utilizando un servicio de terceros como Google, Meta o Apple.

●El servidor de autenticación de terceros genera un JWT que contiene la información y los permisos del usuario.

●Este JWT se pasa a AbstractAccountJWTSigner.

●Cuando es necesario realizar una operación de blockchain, AbstractAccountJWTSigner utiliza el JWT para **verificar la identidad del usuario, verificar sus permisos** o **solicitar la ejecución de una acción de blockchain específica** en el backend.

●La red blockchain recibe esta solicitud validada y autorizada por JWT y ejecuta la operación correspondiente.

A diferencia de los métodos de firma tradicionales, la función signDirect de esta configuración no firma los datos localmente. En su lugar, la solicitud de firma se envía a un servidor remoto y el sessionToken se utiliza para autenticarse con el JWT.

```
export abstract class AbstractAccountJWTSigner implements AASigner
{
  constructor(
    protected readonly client: AAClient,
    protected readonly jwtToken: string
  ) {}

  // ... other methods ...

  async signDirect(address: string, signDoc: SignDoc):
Promise<DirectSignResponse> {

      // ...
    const authResponse = await fetch(
      `${this.apiUrl}/api/v1/sessions/authenticate`,
      {
```

```
        method: "POST",
        headers: {
          "content-type": "application/json",
        },
        body: JSON.stringify({
                          // this.sessionToken is used to obtain a
new JWT.           session_token: this.sessionToken,
          session_duration_minutes: 60 * 24 * 30,
          session_custom_claims: {
            transaction_hash: message,
          },
        }),
      },
    );

    // ...
    return {
      signed: signDoc,
      signature: {
        pub_key: {
          type: "",
          value: new Uint8Array(),
        },
        signature: Buffer.from(
                    // Convert JWT response to blockchain signature
format
          authResponseData.data.session_jwt,
          "utf-8",
        ).toString("base64"),
      },
    };
  }
}
```

Escenarios de aplicación

Integración de autenticación de terceros: los usuarios pueden iniciar sesión en una DApp mediante servicios como Google, Meta o Apple. El proceso es el siguiente:

- El usuario inicia sesión con una cuenta de terceros.
- El servidor de autenticación genera un JWT.
- AbstractAccountJWTSigner utiliza este JWT para actuar en nombre del usuario en operaciones de blockchain.

Gestión de permisos granular: si una DApp necesita un control de acceso detallado, el JWT puede incluir roles de usuario e información de permisos. AbstractAccountJWTSigner puede decidir si se permiten acciones específicas de la cadena de bloques en función de estos detalles.**Autorización temporal:** para brindar acceso limitado a la cadena de bloques

a aplicaciones de terceros, se puede crear un JWT con permisos específicos y un tiempo de vencimiento. AbstractAccountJWTSigner utiliza este JWT para realizar operaciones restringidas en nombre del usuario.

Preguntas frecuentesDoblar todo

¿Iniciar sesión en la DApp de XION con un correo electrónico de Google utiliza AAJWTSigner?

No, aunque los usuarios pueden iniciar sesión por correo electrónico en plataformas como Staking o Dashboard , utilizan **códigos de verificación por correo electrónico** , no JWT. Un JWT típico tiene la estructura: Header.Payload.Signature, similar al ejemplo siguiente:

Proveedor de Abstraxion

Biblioteca de abstracciones

Abstraxion es una biblioteca de herramientas de desarrollo frontend personalizada para XION, diseñada para servir como una solución de abstracción de cuentas, ayudando a los desarrolladores a integrar metacuentas en aplicaciones React.Las principales características de la biblioteca Abstraxion incluyen:

- **Abstracción de cuentas:** proporciona una forma simplificada y sencilla de crear cuentas abstractas, reduciendo la complejidad de la interacción de la cadena de bloques.
- **Firma de transacciones:** encapsula la lógica de firma de transacciones, lo que facilita el envío de transacciones.
- **Integración con Graz :** integra perfectamente la biblioteca Graz (una biblioteca React Hooks para el ecosistema Cosmos), conservando la funcionalidad tradicional de la billetera Cosmos.
- **Componentes y ganchos de React:** ofrece una serie de componentes de React y ganchos personalizados, como ***AbstraxionProvider*** , para ayudar a los desarrolladores a administrar estados de cuentas y realizar interacciones de blockchain en sus aplicaciones.

Proveedor de Abstraxion

AbstraxionProvider es uno de los componentes principales de la biblioteca xion.js, diseñado para proporcionar a los desarrolladores el contexto y las funcionalidades necesarias para interactuar con XION al crear aplicaciones descentralizadas (DApps).

```
export function AbstraxionProvider({
  children,
  config,
}: {
  children: React.ReactNode;
  config: AbstraxionConfig;
}): JSX.Element {
```

```
  return (
    <AbstraxionContextProvider
      contracts={config.contracts}
      rpcUrl={config.rpcUrl}
      restUrl={config.restUrl}
      stake={config.stake}
      bank={config.bank}
      callbackUrl={config.callbackUrl}
      treasury={config.treasury}
    >
      {children}
    </AbstraxionContextProvider>
  );
}
```

Principales funcionalidades de AbstraxionProvider:

1.**Gestión de configuración global:**

Permite configurar parámetros como URL RPC , URL REST , direcciones de contrato , etc.***AbstraxionProvider*** acepta un objeto AbstraxionConfig como parámetro de configuración que define cómo interactuar con XION.La configuración incluye:

```
export interface AbstraxionConfig {
  contracts?: ContractGrantDescription[];
  rpcUrl?: string;
  restUrl?: string;
  stake?: boolean;
  bank?: SpendLimit[];
  callbackUrl?: string;
  treasury?: string;
}
```

2.**Proporcionar contexto:**

AbstraxionProvider utiliza internamente AbstraxionContextProvider para crear un contexto React que contiene todos los estados y funciones necesarios para interactuar con XION.AbstraxionContextProvider ofrece :

- Gestión del estado de la conexión
- Manejo de errores
- Gestión de información de cuentas
- Configuración de la instancia XION
- Persistencia del estado de autenticación
- Funcionalidad de cierre de sesión

```
export function AbstraxionContextProvider({
  children,
  contracts,
  rpcUrl = testnetChainInfo.rpc,
  restUrl = testnetChainInfo.rest,
```

```
  stake = false,
  bank,
  callbackUrl,
  treasury,
}: {
...
  return (
    <AbstraxionContext.Provider
      value={{
        isConnected,
        setIsConnected,
        isConnecting,
        setIsConnecting,
        abstraxionError,
        setAbstraxionError,
        abstraxionAccount,
        setAbstraxionAccount,
        granterAddress,
        showModal,
        setShowModal,
        setGranterAddress,
        contracts,
        dashboardUrl,
        setDashboardUrl,
        rpcUrl,
        restUrl,
        stake,
        bank,
        treasury,
        logout,
      }}
    >
      {children}
    </AbstraxionContext.Provider>
  );
}
```

3.**Gestión estatal:**

AbstraxionContextProvider administra estados clave, incluidos el estado de conexión, la información de la cuenta y el manejo de errores. Los componentes secundarios pueden acceder a estos estados a través del gancho useContext .

¿Cómo utilizar AbstraxionProvider?

Para utilizar AbstraxionProvider , los desarrolladores deben incluirlo en el componente raíz de su aplicación frontend.He aquí un ejemplo de uso:

1.Instalar la dependencia: npm i @burnt-labs/abstraxion

2.Importe AbstraxionProvider y los estilos requeridos en el componente raíz:

```
import { AbstraxionProvider } from "@burnt-labs/abstraxion";
import "@burnt-labs/abstraxion/dist/index.css";
```

3.Configure el objeto treasuryConfig para especificar la dirección del contrato XION:

```
const treasuryConfig = {
  treasury:
"xion17ah4x9te3sttpy2vj5x6hv4xvc0td526nu0msf7mt3kydqj4qs2s9jhe90",
// Example of XION treasury contract address
  // Optional parameters for configuring mainnet information
  // rpcUrl: "https://rpc.xion-mainnet-1.burnt.com:443",
  // restUrl: "https://api.xion-mainnet-1.burnt.com:443",
};
```

4.Envuelva toda la aplicación en AbstraxionProvider dentro del componente RootLayout , pasando treasuryConfig como propiedad de configuración .

```
export default function RootLayout({
  children,
}: {
  children: React.ReactNode;
}): JSX.Element {
  return (
    <html lang="en">
      <body>
        <AbstraxionProvider config={treasuryConfig}>
           {children}
         </AbstraxionProvider>
      </body>
    </html>
  );
}
```

Aquí hay un ejemplo completo del uso de ***AbstraxionProvider*** en desarrollo.

```
import { AbstraxionProvider } from "@burnt-labs/abstraxion";
import "@burnt-labs/abstraxion/dist/index.css";

const treasuryConfig = {
  treasury: "xion17ah4x9te3sttpy2vj5x6hv4xvc0td526nu0msf7mt3kydqj4qs2s9jhe90", // Example of XION treasury contract address
    // Optional parameters for configuring mainnet information
  // rpcUrl: "https://rpc.xion-mainnet-1.burnt.com:443",
  // restUrl: "https://api.xion-mainnet-1.burnt.com:443",
};

export default function RootLayout({
  children,
}: {
  children: React.ReactNode;
}): JSX.Element {
  return (
    <html lang="en">
      <body>
        <AbstraxionProvider config={treasuryConfig}>
          {children}
        </AbstraxionProvider>
      </body>
    </html>
  );
}
```

Abstraxión

Abstraxion es un componente central de XION que se utiliza para implementar la abstracción de cuentas. Proporciona una interfaz modal que permite a los usuarios conectarse y administrar sus cuentas abstractas existentes.

```
export function Abstraxion({ onClose }: ModalProps): JSX.Element | null {
  const {
```

```
    abstraxionAccount,
    abstraxionError,
    isConnected,
    showModal,
    setShowModal,
  } = useContext(AbstraxionContext);

  const closeOnEscKey = useCallback(
    (e: KeyboardEventInit): void => {
      if (e.key === "Escape") {
        onClose();
        setShowModal(false);
      }
    },
    [onClose, setShowModal],
  );

  useEffect(() => {
    document.addEventListener("keydown", closeOnEscKey);
    return () => {
      document.removeEventListener("keydown", closeOnEscKey);
    };
  }, [closeOnEscKey]);

  if (!showModal) return null;

  return (
    <Dialog onOpenChange={onClose} open={showModal}>
      <DialogContent>
        {abstraxionError ? (
          <ErrorDisplay />
        ) : abstraxionAccount || isConnected ? (
          <Connected onClose={onClose} />
        ) : !abstraxionAccount ? (
          <AbstraxionSignin />
        ) : null}
      </DialogContent>
    </Dialog>
  );
```

```
42}
```

1.Definición del componente Abstraxion

Abstraxion se define como un componente funcional de React que acepta una devolución de llamada `onClose` como propiedad.

```
1export function Abstraxion({ onClose }: ModalProps): JSX.Element | null {
2  const {
3    abstraxionAccount,
4    abstraxionError,
5    isConnected,
6    showModal,
7    setShowModal,
8  } = useContext(AbstraxionContext);
```

2.Uso del contexto

El componente utiliza el gancho `useContext` de React para recuperar el estado y las funciones de `AbstraxionContext` .

```
1  const {
2    abstraxionAccount,
3    abstraxionError,
4    isConnected,
5    showModal,
6    setShowModal,
7  } = useContext(AbstraxionContext);
```

3.Manejo del cierre modal con la tecla Escape

La función `closeOnEscKey` maneja la lógica para cerrar el modal cuando se presiona la tecla Esc.

```
1  const closeOnEscKey = useCallback(
2    (e: KeyboardEventInit): void => {
3      if (e.key === "Escape") {
4        onClose();
5        setShowModal(false);
6      }
7    },
8    [onClose, setShowModal],
9  );
```

4.Manejo de los efectos secundarios

`useEffect` se utiliza para agregar y eliminar el detector de eventos del teclado para la tecla Escape.

```
useEffect(() => {
  document.addEventListener("keydown", closeOnEscKey);
  return () => {
    document.removeEventListener("keydown", closeOnEscKey);
  };
}, [closeOnEscKey]);
```

5.**Representación condicional**

El contenido modal solo se representa si `showModal` es verdadero.

```
if (!showModal) return null;
```

6.**Estructura modal**

La estructura modal se construye utilizando los componentes `Dialog` y `DialogContent` .

```
return (
  <Dialog onOpenChange={onClose} open={showModal}>
    <DialogContent>
      {abstraxionError ? (
        <ErrorDisplay />
      ) : abstraxionAccount || isConnected ? (
        <Connected onClose={onClose} />
      ) : !abstraxionAccount ? (
        <AbstraxionSignin />
      ) : null}
    </DialogContent>
  </Dialog>
);
```

7.**Contenido condicional**

El componente representa contenido diferente según los estados de `abstraxionError` , `abstraxionAccount` y `isConnected` .

```
  {abstraxionError ? (
  <ErrorDisplay />
) : abstraxionAccount || isConnected ? (
  <Connected onClose={onClose} />
) : !abstraxionAccount ? (
  <AbstraxionSignin />
) : null
}
```

¿Cómo utilizar Abstraxion?

1.Primero, importe los componentes necesarios a su aplicación frontend de React.

```
"use client";
import { Abstraxion } from "@burnt-labs/abstraxion";
import { useState } from "react";
```

2.Crea un estado para controlar la visibilidad del modal **Abstraxion**.

```
export default function Home() {
  const [isOpen, setIsOpen] = useState(false);
```

3.Dentro de la función de render, agregue el componente **Abstraxion** y un botón para activar el modal.

```
  return (
    <div>
      <Abstraxion onClose={() => setIsOpen(false)} isOpen={isOpen} />
      <button onClick={() => setIsOpen(true)}>Connect Wallet</button>
    </div>
  );
```

Abstraxion Hooks

Abstraxion offers three custom hooks: **useAbstraxionAccount**, **useAbstraxionSigningClient**, and **useModal**.

useAbstraxionAccount

The **useAbstraxionAccount** hook provides information about the currently connected account.

```
export const useAbstraxionAccount = (): AbstraxionAccountState => {
  const { isConnected, granterAddress, isConnecting } =
    useContext(AbstraxionContext);
    useContext(AbstraxionContext);
  return {
    data: {
      bech32Address: granterAddress,
    },
```

```
9     isConnected,
10     isConnecting,
11   };

12   };
```

Implementation logic:

1.The `useContext` hook is used to get the connection state from `AbstraxionContext`.

The values `isConnected`, `isConnecting`, and `granterAddress` are destructured from the context.

- `isConnected`: Indicates if the account is connected.
- `isConnecting`: Shows whether the connection is in progress.
- `granterAddress`: Represents the address of the fee granter.

```
1export const useAbstraxionAccount = (): AbstraxionAccountState =>
{
2   const { isConnected, granterAddress, isConnecting } =
3     useContext(AbstraxionContext);

4     useContext(AbstraxionContext);
```

2.These values are returned to the hook caller.

```
1   return {
2     data: {
3       bech32Address: granterAddress,
4     },
5     isConnected,
6     isConnecting,
7   };
```

```
8  };
```

useAbstraxionSigningClient

The **useAbstraxionSigningClient** hook provides a signing client for interacting with the blockchain.

Code:

Key points:

1.Hook definition and return type:

Define the Hook return object, which includes the three properties: `client`, `signArb`, and `logout`.

```
1export const useAbstraxionSigningClient = (): {
2  readonly client: GranteeSignerClient | undefined;
3  readonly signArb:
4    | ((signerAddress: string, message: string | Uint8Array) =>
Promise<string>)
5    | undefined;
6  readonly logout: (() => void) | undefined;

7} => {
```

2.State and context handling:

The hook uses `useContext` and `useState` to manage the signing client state.

```
1  const { isConnected, abstraxionAccount, granterAddress, rpcUrl,
logout } =
2    useContext(AbstraxionContext);
3  const [signArbWallet, setSignArbWallet] = useState<
4    SignArbSecp256k1HdWallet | undefined
5  >(undefined);
6
7  const [abstractClient, setAbstractClient] = useState<
```

```
8     GranteeSignerClient | undefined

9   >(undefined);
```

3.Async Signer retrieval:

The signer is fetched using the useEffect hook.

```
1   useEffect(() => {

2     async function getSigner() {
```

4.Error handling:

Errors are caught and managed during the Signer retrieval process.

```
1try {
2   if (!abstraxionAccount) {
3             throw new Error("No account found.");
4   }
5
6   if (!granterAddress) {
7             throw new Error("No granter found.");

8   }
```

5.Retrieve granteeAddress:

Retrieve granteeAddress from abstraxionAccount.

```
1const granteeAddress = await abstraxionAccount
2           .getAccounts()
3           .then((accounts) => {
4                 if (accounts.length === 0) {
5                       throw new Error("No account found.");
6       }
7       return accounts[0].address;

8     });
```

6.GranteeSignerClient creation:

The client is instantiated with granterAddress and granteeAddress.

```
const directClient = await GranteeSignerClient.connectWithSigner(
    // Should be set in the context but defaulting here just in
case
    rpcUrl || testnetChainInfo.rpc,
    abstraxionAccount,
    {
          gasPrice: GasPrice.fromString("0uxion"),
   granterAddress,
   granteeAddress,
    },

);
```

7.Local keypair retrieval:

A local keypair is retrieved and assigned to signArbWallet.

```
const wallet = await abstraxionAuth.getLocalKeypair();
    if (wallet) {
          setSignArbWallet(wallet);

    }
```

8.Return values:

The hook returns an object containing client, signArb, and logout.

```
  return {
    client: abstractClient,
    signArb: signArbWallet?.signArb,
    logout,

  } as const;
```

useModal

useModal is used to manage the display state of the Abstraxion modal, aiming to provide a simple way to control the opening and closing of the modal.Specifically, **useModal** encapsulates the modal state logic within a Hook, offering a reusable way to manage the state of the Abstraxion modal across the application without needing direct access to the AbstraxionContext.

```
1export const useModal = (): [
2  boolean,
3  React.Dispatch<React.SetStateAction<boolean>>,
4] => {
5  const { showModal, setShowModal } =
useContext(AbstraxionContext);
6  return [showModal, setShowModal];

7};
```

1.**useModal** is defined as a function that returns a tuple containing a boolean and a setter function.

```
1export const useModal = (): [
2  boolean,
3  React.Dispatch<React.SetStateAction<boolean>>,

4] => {
```

2.Provide Context

Access `AbstraxionContext` and destructure `showModal` and `setShowModal` from it.

```
1const { showModal, setShowModal } =
useContext(AbstraxionContext);
```

3.Return Value:

The hook returns an array containing `showModal` (a boolean) and `setShowModal` (a function), allowing users to directly access and modify the modal's state.

```
return [showModal, setShowModal];
```

How to Use Abstraxion Hooks?

以 useAbstraxionSigningClient 为例，如何使用 client 执行交易。

1.Import and use the `useAbstraxionSigningClient` hook in your component:

```
import {
  useAbstraxionAccount,
  useAbstraxionSigningClient,
} from "@burnt-labs/abstraxion";
const { data: account } = useAbstraxionAccount();

const { client, signArb, logout } = useAbstraxionSigningClient();
```

We destructure the `client`, `signArb`, and `logout` values.`client` is used to execute transactions.`signArb` is used to sign messages.`logout` is used to log out the user.

2.Execute a transaction using the `client`:

```
const claimRes = await client?.execute(
    account.bech32Address,
    seatContractAddress,
    msg,
    {
          amount: [{ amount: "0", denom: "uxion" }],
          gas: "500000",
    },
          "",
          [],

    );
```

Here, `client.execute` is used to send the transaction with the following parameters:Method Parameters:

- Sender address: account.bech32Address
- Contract address: seatContractAddress
- Message object: msg
- Transaction options: including the amount and gas limit.

Using `useAbstraxionSigningClient` simplifies the process by providing a pre-configured client for executing transactions.

Here is a complete example using **Abstraxion Hooks** for development:

```
"use client";
import Link from "next/link";
import { useState } from "react";
import {
  Abstraxion,
  useAbstraxionAccount,
  useAbstraxionSigningClient,
  useModal,
} from "@burnt-labs/abstraxion";
import { Button } from "@burnt-labs/ui";
import "@burnt-labs/ui/dist/index.css";
import type { ExecuteResult } from "@cosmjs/cosmwasm-stargate";
import { SignArb } from "../components/sign-arb.tsx";

const seatContractAddress =

"xion1z70cvc08qv5764zeg3dykcyymj5z6nu4sqr7x8vl4zjef2gyp69s9mmdka";

type ExecuteResultOrUndefined = ExecuteResult | undefined;

export default function Page(): JSX.Element {
  // Abstraxion hooks
  const { data: account } = useAbstraxionAccount();
  const { client, signArb, logout } =
useAbstraxionSigningClient();

  // General state hooks
```

```
  const [, setShowModal]: [
    boolean,
    React.Dispatch<React.SetStateAction<boolean>>,
  ] = useModal();
  const [loading, setLoading] = useState(false);
  const [executeResult, setExecuteResult] =
    useState<ExecuteResultOrUndefined>(undefined);

  const blockExplorerUrl =
`https://explorer.burnt.com/xion-testnet-1/tx/${executeResult?.tra
nsactionHash}`;

  function getTimestampInSeconds(date: Date | null): number {
    if (!date) return 0;
    const d = new Date(date);
    return Math.floor(d.getTime() / 1000);
  }

  const now = new Date();
  now.setSeconds(now.getSeconds() + 15);
  const oneYearFromNow = new Date();
  oneYearFromNow.setFullYear(oneYearFromNow.getFullYear() + 1);

  async function claimSeat(): Promise<void> {
    setLoading(true);
    const msg = {
      sales: {
        claim_item: {
          token_id: String(getTimestampInSeconds(now)),
          owner: account.bech32Address,
          token_uri: "",
          extension: {},
        },
      },
    };

    try {
      const claimRes = await client?.execute(
        account.bech32Address,
        seatContractAddress,
        msg,
        {
          amount: [{ amount: "0", denom: "uxion" }],
          gas: "500000",
        },
        "",
        [],
      );
```

```
72
73        setExecuteResult(claimRes);
74      } catch (error) {
75        // eslint-disable-next-line no-console -- No UI exists yet
to display errors
76        console.log(error);
77      } finally {
78        setLoading(false);
79      }
80    }
81
82    return (
83      <main className="m-auto flex min-h-screen max-w-xs flex-col
items-center justify-center gap-4 p-4">
84        <h1 className="text-2xl font-bold tracking-tighter
text-white">
85          ABSTRAXION
86        </h1>
87        <Button
88          fullWidth
89          onClick={() => {
90            setShowModal(true);
91          }}
92          structure="base"
93        >
94          {account.bech32Address ? (
95            <div className="flex items-center justify-center">VIEW
ACCOUNT</div>
96          ) : (
97            "CONNECT"
98          )}
99        </Button>
100       {client ? (
101         <>
102           <Button
103             disabled={loading}
104             fullWidth
105             onClick={() => {
106               void claimSeat();
107             }}
108             structure="base"
109           >
110             {loading ? "LOADING..." : "CLAIM SEAT"}
111           </Button>
112           {logout ? (
113             <Button
114               disabled={loading}
115               fullWidth
```

```
116                onClick={() => {
117                  logout();
118                }}
119                structure="base"
120              >
121                LOGOUT
122              </Button>
123            ) : null}
124            {signArb ? <SignArb /> : null}
125          </>
126        ) : null}
127        <Abstraxion
128          onClose={() => {
129            setShowModal(false);
130          }}
131        />
132        {executeResult ? (
133          <div className="flex flex-col rounded border-2
border-black p-2 dark:border-white">
134            <div className="mt-2">
135              <p className="text-zinc-500">
136                <span className="font-bold">Transaction
Hash</span>
137              </p>
138              <p
className="text-sm">{executeResult.transactionHash}</p>
139            </div>
140            <div className="mt-2">
141              <p className=" text-zinc-500">
142                <span className="font-bold">Block Height:</span>
143              </p>
144              <p className="text-sm">{executeResult.height}</p>
145            </div>
146            <div className="mt-2">
147              <Link
148                className="text-black underline
visited:text-purple-600 dark:text-white"
149                href={blockExplorerUrl}
150                target="_blank"
151              >
152                View in Block Explorer
153              </Link>
154            </div>
155          </div>
156        ) : null}
157      </main>
158    );
159}
```

Creando una experiencia sin gas mediante contratos de tesorería

/ Unidad 3 - Implementación de Gas-less en Contratos de Tesorería

/ Módulo 3.1 - Authz

Módulo 3.1 - Authz

autorización

En esta lección, aprenderemos cómo un usuario puede autorizar privilegios de operación a un contrato para un tipo específico de transacción.Esto se logra a través del módulo authz en Cosmos.

Autorización

Este módulo permite otorgar derechos de operación arbitrarios de una cuenta a otra, es decir, el usuario puede hacer que un contrato realice una operación específica en su nombre.

Actualmente existen tres tipos de autorización:

●**GenericAuthorization** : Autorización genérica, donde el usuario puede otorgar cualquier privilegio de ejecución de mensajes a un contrato.

●**SendAuthorization : autorización de transferencia, el usuario puede especificar a qué direcciones** se puede transferir el contrato y hasta **qué cantidad** de transferencia.

●**StakeAuthorization** : Autorización de compromiso, los usuarios pueden elegir permitir o denegar el contrato con el que comprometerse con qué autenticadores.

Uso

Cosmos ha implementado el módulo lógico correspondiente, podemos usarlo directamente:

1.Establecer la configuración correspondiente según los tipos definidos.

2.Llame a cosmos_sdk e ingrese a la configuración.

x/authz

Abstract

`x/authz` is an implementation of a Cosmos SDK module, per ADR 30, that allows granting arbitrary privileges from one account (the granter) to another account (the grantee). Authorizations must be granted for a particular Msg service method one by one using an implementation of the `Authorization` interface.

Contents

Concepts

Authorization and Grant

The `x/authz` module defines interfaces and messages grant authorizations to perform actions on behalf of one account to other accounts. The design is defined in the ADR 030.

A *grant* is an allowance to execute a Msg by the grantee on behalf of the granter. Authorization is an interface that must be implemented by a concrete authorization logic to validate and execute grants. Authorizations are extensible and can be defined for any Msg service method even outside of the module where the Msg method is defined. See the `SendAuthorization` example in the next section for more details.

Note: The authz module is different from the auth (authentication) module that is responsible for specifying the base transaction and account types.

```
x/authz/authorizations.go
type Authorization interface {
     proto.Message

     // MsgTypeURL returns the fully-qualified Msg service
method URL (as described in ADR 031),
     // which will process and accept or reject a request.
     MsgTypeURL() string

     // Accept determines whether this grant permits the
provided sdk.Msg to be performed,
     // and if so provides an upgraded authorization instance.
     Accept(ctx sdk.Context, msg sdk.Msg) (AcceptResponse,
error)

     // ValidateBasic does a simple validation check that
     // doesn't require access to any other information.
     ValidateBasic() error
}
```

See full example on GitHub

Built-in Authorizations

The Cosmos SDK `x/authz` module comes with following authorization types:

GenericAuthorization

`GenericAuthorization` implements the `Authorization` interface that gives unrestricted permission to execute the provided Msg on behalf of granter's account.

```
proto/cosmos/authz/v1beta1/authz.proto
// GenericAuthorization gives the grantee unrestricted
permissions to execute
// the provided method on behalf of the granter's account.
message GenericAuthorization {
```

```
 option (amino.name)                          =
"cosmos-sdk/GenericAuthorization";
 option (cosmos_proto.implements_interface) =
"cosmos.authz.v1beta1.Authorization";

 // Msg, identified by it's type URL, to grant unrestricted
permissions to execute
 string msg = 1;
}
```

See full example on GitHub

```
x/authz/generic_authorization.go
// MsgTypeURL implements Authorization.MsgTypeURL.
func (a GenericAuthorization) MsgTypeURL() string {
     return a.Msg
}

// Accept implements Authorization.Accept.
func (a GenericAuthorization) Accept(ctx sdk.Context, msg
sdk.Msg) (AcceptResponse, error) {
     return AcceptResponse{Accept: true}, nil
}

// ValidateBasic implements Authorization.ValidateBasic.
func (a GenericAuthorization) ValidateBasic() error {
     return nil
}
```

See full example on GitHub

`msg` stores Msg type URL.

SendAuthorization

`SendAuthorization` implements the `Authorization` interface for the `cosmos.bank.v1beta1.MsgSend` Msg.

It takes a (positive) `SpendLimit` that specifies the maximum amount of tokens the grantee can spend. The `SpendLimit` is updated as the tokens are spent.
It takes an (optional) `AllowList` that specifies to which addresses a grantee can send token.

`proto/cosmos/bank/v1beta1/authz.proto`

```
// SendAuthorization allows the grantee to spend up to
spend_limit coins from
// the granter's account.
//
// Since: cosmos-sdk 0.43
message SendAuthorization {
 option (cosmos_proto.implements_interface) =
"cosmos.authz.v1beta1.Authorization";
 option (amino.name)                        =
"cosmos-sdk/SendAuthorization";

 repeated cosmos.base.v1beta1.Coin spend_limit = 1 [
   (gogoproto.nullable)     = false,
   (amino.dont_omitempty)   = true,
   (gogoproto.castrepeated) =
"github.com/cosmos/cosmos-sdk/types.Coins"
 ];

 // allow_list specifies an optional list of addresses to whom
the grantee can send tokens on behalf of the
 // granter. If omitted, any recipient is allowed.
 //
 // Since: cosmos-sdk 0.47
 repeated string allow_list = 2;
}
```

See full example on GitHub

```
x/bank/types/send_authorization.go
// Accept implements Authorization.Accept.
func (a SendAuthorization) Accept(ctx sdk.Context, msg
sdk.Msg) (authz.AcceptResponse, error) {
     mSend, ok := msg.(*MsgSend)
     if !ok {
          return authz.AcceptResponse{},
sdkerrors.ErrInvalidType.Wrap("type mismatch")
     }

     toAddr := mSend.ToAddress

     limitLeft, isNegative :=
a.SpendLimit.SafeSub(mSend.Amount...)
     if isNegative {
          return authz.AcceptResponse{},
sdkerrors.ErrInsufficientFunds.Wrapf("requested amount is more
than spend limit")
```

```
	}
	if limitLeft.IsZero() {
		return authz.AcceptResponse{Accept: true, Delete:
true}, nil
	}

	isAddrExists := false
	allowedList := a.GetAllowList()

	for _, addr := range allowedList {
		ctx.GasMeter().ConsumeGas(gasCostPerIteration, "send
authorization")
		if addr == toAddr {
			isAddrExists = true
			break
		}
	}

	if len(allowedList) > 0 && !isAddrExists {
		return authz.AcceptResponse{},
sdkerrors.ErrUnauthorized.Wrapf("cannot send to %s address",
toAddr)
	}

	return authz.AcceptResponse{Accept: true, Delete: false,
Updated: &SendAuthorization{SpendLimit: limitLeft, AllowList:
allowedList}}, nil
}
```

See full example on GitHub

`spend_limit` keeps track of how many coins are left in the authorization. `allow_list` specifies an optional list of addresses to whom the grantee can send tokens on behalf of the granter.

StakeAuthorization

`StakeAuthorization` implements the `Authorization` interface for messages in the staking module. It takes an `AuthorizationType` to specify whether you want to authorise delegating, undelegating or redelegating (i.e. these have to be authorised separately). It also takes an optional `MaxTokens` that keeps track of a limit to the amount of tokens that can be delegated/undelegated/redelegated. If left empty, the amount is unlimited. Additionally, this Msg takes an `AllowList` or a

`DenyList`, which allows you to select which validators you allow or deny grantees to stake with.

proto/cosmos/staking/v1beta1/authz.proto

```
// StakeAuthorization defines authorization for
delegate/undelegate/redelegate.
//
// Since: cosmos-sdk 0.43
message StakeAuthorization {
 option (cosmos_proto.implements_interface) =
"cosmos.authz.v1beta1.Authorization";
 option (amino.name)                        =
"cosmos-sdk/StakeAuthorization";

 // max_tokens specifies the maximum amount of tokens can be
delegate to a validator. If it is
 // empty, there is no spend limit and any amount of coins can
be delegated.
 cosmos.base.v1beta1.Coin max_tokens = 1
[(gogoproto.castrepeated) =
"github.com/cosmos/cosmos-sdk/types.Coin"];
 // validators is the oneof that represents either allow_list
or deny_list
 oneof validators {
   // allow_list specifies list of validator addresses to whom
grantee can delegate tokens on behalf of granter's
   // account.
   Validators allow_list = 2;
   // deny_list specifies list of validator addresses to whom
grantee can not delegate tokens.
   Validators deny_list = 3;
 }
 // Validators defines list of validator addresses.
 message Validators {
   repeated string address = 1 [(cosmos_proto.scalar) =
"cosmos.AddressString"];
 }
 // authorization_type defines one of AuthorizationType.
 AuthorizationType authorization_type = 4;
}
```

See full example on GitHub

x/staking/types/authz.go

```
// NewStakeAuthorization creates a new StakeAuthorization
object.
```

```
func NewStakeAuthorization(allowed []sdk.ValAddress, denied
[]sdk.ValAddress, authzType AuthorizationType, amount
*sdk.Coin) (*StakeAuthorization, error) {
     allowedValidators, deniedValidators, err :=
validateAllowAndDenyValidators(allowed, denied)
     if err != nil {
          return nil, err
     }

     a := StakeAuthorization{}
     if allowedValidators != nil {
          a.Validators =
&StakeAuthorization_AllowList{AllowList:
&StakeAuthorization_Validators{Address: allowedValidators}}
     } else {
          a.Validators =
&StakeAuthorization_DenyList{DenyList:
&StakeAuthorization_Validators{Address: deniedValidators}}
     }

     if amount != nil {
          a.MaxTokens = amount
     }
     a.AuthorizationType = authzType

     return &a, nil
}
```

See full example on GitHub

Gas

In order to prevent DoS attacks, granting `StakeAuthorization`s with `x/authz` incurs gas. `StakeAuthorization` allows you to authorize another account to delegate, undelegate, or redelegate to validators. The authorizer can define a list of validators they allow or deny delegations to. The Cosmos SDK iterates over these lists and charge 10 gas for each validator in both of the lists.

Since the state maintaining a list for granter, grantee pair with same expiration, we are iterating over the list to remove the grant (incase of any revoke of paritcular `msgType`) from the list and we are charging 20 gas per iteration.

State

Grant

Grants are identified by combining granter address (the address bytes of the granter), grantee address (the address bytes of the grantee) and Authorization type (its type URL). Hence we only allow one grant for the (granter, grantee, Authorization) triple.

> Grant: `0x01 | granter_address_len (1 byte) | granter_address_bytes | grantee_address_len (1 byte) | grantee_address_bytes | msgType_bytes -> ProtocolBuffer(AuthorizationGrant)`

The grant object encapsulates an `Authorization` type and an expiration timestamp:

```
proto/cosmos/authz/v1beta1/authz.proto
// Grant gives permissions to execute
// the provide method with expiration time.
message Grant {
 google.protobuf.Any authorization = 1
[(cosmos_proto.accepts_interface) =
"cosmos.authz.v1beta1.Authorization"];
 // time when the grant will expire and will be pruned. If
null, then the grant
 // doesn't have a time expiration (other conditions  in
`authorization`
 // may apply to invalidate the grant)
 google.protobuf.Timestamp expiration = 2 [(gogoproto.stdtime)
= true, (gogoproto.nullable) = true];
}
```

See full example on GitHub

GrantQueue

We are maintaining a queue for authz pruning. Whenever a grant is created, an item will be added to `GrantQueue` with a key of expiration, granter, grantee.

In `EndBlock` (which runs for every block) we continuously check and prune the expired grants by forming a prefix key with current blocktime that passed the stored expiration in `GrantQueue`, we iterate through all the matched records from `GrantQueue` and delete them from the `GrantQueue` & `Grant`s store.

```
x/authz/keeper/keeper.go
func (k Keeper) DequeueAndDeleteExpiredGrants(ctx sdk.Context)
error {
```

```
	store := ctx.KVStore(k.storeKey)

	iterator := store.Iterator(GrantQueuePrefix,
sdk.InclusiveEndBytes(GrantQueueTimePrefix(ctx.BlockTime())))
	defer iterator.Close()

	for ; iterator.Valid(); iterator.Next() {
		var queueItem authz.GrantQueueItem
		if err := k.cdc.Unmarshal(iterator.Value(),
&queueItem); err != nil {
			return err
		}

		_, granter, grantee, err :=
parseGrantQueueKey(iterator.Key())
		if err != nil {
			return err
		}

		store.Delete(iterator.Key())

		for _, typeURL := range queueItem.MsgTypeUrls {
			store.Delete(grantStoreKey(grantee, granter,
typeURL))
		}
	}

	return nil
}
```

See full example on GitHub

GrantQueue: `0x02 | expiration_bytes | granter_address_len (1 byte) | granter_address_bytes | grantee_address_len (1 byte) | grantee_address_bytes -> ProtocalBuffer(GrantQueueItem)`

The `expiration_bytes` are the expiration date in UTC with the format `"2006-01-02T15:04:05.000000000"`.

`x/authz/keeper/keys.go`

```
// GrantQueueKey - return grant queue store key. If a given
grant doesn't have a defined
// expiration, then it should not be used in the pruning
queue.
```

```
// Key format is:
//
//   0x02<expiration><granterAddressLen (1
Byte)><granterAddressBytes><granteeAddressLen (1
Byte)><granteeAddressBytes>: GrantQueueItem
func GrantQueueKey(expiration time.Time, granter
sdk.AccAddress, grantee sdk.AccAddress) []byte {
     exp := sdk.FormatTimeBytes(expiration)
     granter = address.MustLengthPrefix(granter)
     grantee = address.MustLengthPrefix(grantee)

     return sdk.AppendLengthPrefixedBytes(GrantQueuePrefix,
exp, granter, grantee)
}

// GrantQueueTimePrefix - return grant queue time prefix
func GrantQueueTimePrefix(expiration time.Time) []byte {
     return append(GrantQueuePrefix,
sdk.FormatTimeBytes(expiration)...)
}
```

See full example on GitHub

The `GrantQueueItem` object contains the list of type urls between granter and grantee that expire at the time indicated in the key.

Messages

In this section we describe the processing of messages for the authz module.

MsgGrant

An authorization grant is created using the `MsgGrant` message. If there is already a grant for the `(granter, grantee, Authorization)` triple, then the new grant overwrites the previous one. To update or extend an existing grant, a new grant with the same `(granter, grantee, Authorization)` triple should be created.

proto/cosmos/authz/v1beta1/tx.proto

```
// MsgGrant is a request type for Grant method. It declares
authorization to the grantee
// on behalf of the granter with the provided expiration time.
message MsgGrant {
 option (cosmos.msg.v1.signer) = "granter";
 option (amino.name)           = "cosmos-sdk/MsgGrant";

 string granter = 1 [(cosmos_proto.scalar) =
"cosmos.AddressString"];
```

```
 string grantee = 2 [(cosmos_proto.scalar) =
"cosmos.AddressString"];

 cosmos.authz.v1beta1.Grant grant = 3 [(gogoproto.nullable) =
false, (amino.dont_omitempty) = true];
}
```

See full example on GitHub

The message handling should fail if:

- both granter and grantee have the same address.
- provided `Expiration` time is less than current unix timestamp (but a grant will be created if no `expiration` time is provided since `expiration` is optional).
- provided `Grant.Authorization` is not implemented.
- `Authorization.MsgTypeURL()` is not defined in the router (there is no defined handler in the app router to handle that Msg types).

MsgRevoke

A grant can be removed with the `MsgRevoke` message.

```
proto/cosmos/authz/v1beta1/tx.proto
// MsgRevoke revokes any authorization with the provided
sdk.Msg type on the
// granter's account with that has been granted to the
grantee.
message MsgRevoke {
 option (cosmos.msg.v1.signer) = "granter";
 option (amino.name)           = "cosmos-sdk/MsgRevoke";

 string granter      = 1 [(cosmos_proto.scalar) =
"cosmos.AddressString"];
 string grantee      = 2 [(cosmos_proto.scalar) =
"cosmos.AddressString"];
 string msg_type_url = 3;
}
```

See full example on GitHub

The message handling should fail if:

- both granter and grantee have the same address.
- provided `MsgTypeUrl` is empty.

NOTE: The `MsgExec` message removes a grant if the grant has expired.

MsgExec

When a grantee wants to execute a transaction on behalf of a granter, they must send `MsgExec`.

```
proto/cosmos/authz/v1beta1/tx.proto
// MsgExec attempts to execute the provided messages using
// authorizations granted to the grantee. Each message should
have only
// one signer corresponding to the granter of the
authorization.
message MsgExec {
 option (cosmos.msg.v1.signer) = "grantee";
 option (amino.name)           = "cosmos-sdk/MsgExec";

 string grantee = 1 [(cosmos_proto.scalar) =
"cosmos.AddressString"];
 // Execute Msg.
 // The x/authz will try to find a grant matching
(msg.signers[0], grantee, MsgTypeURL(msg))
 // triple and validate it.
 repeated google.protobuf.Any msgs = 2
[(cosmos_proto.accepts_interface) =
"cosmos.base.v1beta1.Msg"];
```

See full example on GitHub

The message handling should fail if:

provided `Authorization` is not implemented.
grantee doesn't have permission to run the transaction.
if granted authorization is expired.

Events

The authz module emits proto events defined in the Protobuf reference.

Client

CLI

A user can query and interact with the `authz` module using the CLI.

Query

The `query` commands allow users to query `authz` state.

```
simd query authz --help
```

grants

The `grants` command allows users to query grants for a granter-grantee pair. If the message type URL is set, it selects grants only for that message type.

```
simd query authz grants [granter-addr] [grantee-addr]
[msg-type-url]? [flags]
```

Example:

```
simd query authz grants cosmos1.. cosmos1..
/cosmos.bank.v1beta1.MsgSend
```

Example Output:

```
grants:
- authorization:
    '@type': /cosmos.bank.v1beta1.SendAuthorization
    spend_limit:
    - amount: "100"
      denom: stake
  expiration: "2022-01-01T00:00:00Z"
pagination: null
```

Transactions

The `tx` commands allow users to interact with the `authz` module.

```
simd tx authz --help
```

exec

The `exec` command allows a grantee to execute a transaction on behalf of granter.

```
 simd tx authz exec [tx-json-file] --from [grantee] [flags]
```

Example:

```
simd tx authz exec tx.json --from=cosmos1..
```

grant

The `grant` command allows a granter to grant an authorization to a grantee.

```
simd tx authz grant <grantee>
<authorization_type="send"|"generic"|"delegate"|"unbond"|"rede
legate"> --from <granter> [flags]
```

> The `send` authorization_type refers to the built-in `SendAuthorization` type. The custom flags available are `spend-limit` (required) and `allow-list` (optional) , documented here

Example:

```
   simd tx authz grant cosmos1.. send --spend-limit=100stake
--allow-list=cosmos1...,cosmos2... --from=cosmos1..
```

The `generic` authorization_type refers to the built-in `GenericAuthorization` type. The custom flag available is `msg-type` (required) documented here.

Note: `msg-type` is any valid Cosmos SDK `Msg` type url.

Example:

```
   simd tx authz grant cosmos1.. generic
--msg-type=/cosmos.bank.v1beta1.MsgSend --from=cosmos1..
```

The `delegate`,`unbond`,`redelegate` authorization_types refer to the built-in `StakeAuthorization` type. The custom flags available are `spend-limit` (optional), `allowed-validators` (optional) and `deny-validators` (optional) documented here.
Note: `allowed-validators` and `deny-validators` cannot both be empty. `spend-limit` represents the `MaxTokens`

Example:

```
simd tx authz grant cosmos1.. delegate --spend-limit=100stake
--allowed-validators=cosmos...,cosmos...
--deny-validators=cosmos... --from=cosmos1..
```

revoke

The `revoke` command allows a granter to revoke an authorization from a grantee.

```
simd tx authz revoke [grantee] [msg-type-url] --from=[granter]
[flags]
```

Example:

```
simd tx authz revoke cosmos1.. /cosmos.bank.v1beta1.MsgSend
--from=cosmos1..
```

gRPC

A user can query the `authz` module using gRPC endpoints.

Grants

The `Grants` endpoint allows users to query grants for a granter-grantee pair. If the message type URL is set, it selects grants only for that message type.

```
cosmos.authz.v1beta1.Query/Grants
```

Example:

```
grpcurl -plaintext \
   -d
'{"granter":"cosmos1..","grantee":"cosmos1..","msg_type_url":"
/cosmos.bank.v1beta1.MsgSend"}' \
```

```
   localhost:9090 \
   cosmos.authz.v1beta1.Query/Grants
```

Example Output:

```
{
 "grants": [
   {
     "authorization": {
       "@type": "/cosmos.bank.v1beta1.SendAuthorization",
       "spendLimit": [
         {
           "denom":"stake",
           "amount":"100"
         }
       ]
     },
     "expiration": "2022-01-01T00:00:00Z"
   }
 ]
}
```

REST

A user can query the `authz` module using REST endpoints.

```
/cosmos/authz/v1beta1/grants
```

Example:

```
curl
"localhost:1317/cosmos/authz/v1beta1/grants?granter=cosmos1..&
grantee=cosmos1..&msg_type_url=/cosmos.bank.v1beta1.MsgSend"
```

Example Output:

```
{
 "grants": [
   {
     "authorization": {
       "@type": "/cosmos.bank.v1beta1.SendAuthorization",
       "spend_limit": [
         {
           "denom": "stake",
           "amount": "100"
         }
       ]
     },
     "expiration": "2022-01-01T00:00:00Z"
   }
 ],
```

```
  "pagination": null
}
```

subvención de honorarios

Como mencionamos en las lecciones anteriores, los administradores pueden establecer la cantidad y el período de tiempo durante el cual pagarán las tarifas en su nombre, lo que se logra a través del módulo feegrant en Cosmos.

subvención de honorarios

Este módulo permite que una cuenta otorgue a otras cuentas la capacidad de ejecutar transacciones con sus propios fondos, de manera similar a otorgarle a otro usuario una “permiso” de transacción.

Actualmente existen tres tipos de asignaciones:

- **BasicAllowance** : otorga al usuario permiso básico para utilizar fondos hasta una cierta cantidad.

💡

Si no especifica una asignación, ¡el usuario autorizado puede incluso usar todos sus fondos!

- **PeriodicAllowance** : establece la cantidad de fondos que el usuario puede usar dentro de un período específico y le permite establecer un tiempo de reinicio para la cantidad.
- **AllowedMsgAllowance** : Permite al usuario utilizar un tipo específico de mensaje para las transacciones, restringiendo su uso al tipo de mensaje especificado por el administrador.

Implementación del contrato

Podemos ver que la estructura FeeConfig que necesita configurarse está definida en grant.rs

```
#[cw_serde]
pub struct FeeConfig {
    description: String,
    pub allowance: Option<Any>,
    pub expiration: Option<u32>,
}

#[cw_serde]
pub struct Any {
     pub type_url: String,
     pub value: Binary,
}
```

●descripción: Este es un valor de cadena que representa la descripción de la configuración que se mostrará a nuestros usuarios.
●allowance: Esta es una configuración de autorización de tarifa opcional con type_url y value, dependiendo del tipo de módulo **feegrant** , esta url corresponde al identificador de tipo de los tres tipos de configuraciones anteriores:
●/cosmos.feegrant.v1beta1.BasicAllowance
●/cosmos.feegrant.v1beta1.PeriodicAllowance
●/cosmos.feegrant.v1beta1.AllowedMsgAllowance

Uso

Cosmos ha implementado el módulo lógico correspondiente, por lo que podemos usarlo directamente:
1.Establecer la configuración correspondiente según los tipos definidos.
2.Llame a cosmos_sdk e ingrese a la configuración.

`x/feegrant`

Abstract

This document specifies the fee grant module. For the full ADR, please see Fee Grant ADR-029.

This module allows accounts to grant fee allowances and to use fees from their accounts. Grantees can execute any transaction without the need to maintain sufficient fees.

Contents

Concepts

Grant

`Grant` is stored in the KVStore to record a grant with full context. Every grant will contain `granter`, `grantee` and what kind of `allowance` is granted. `granter` is an account address who is giving permission to `grantee` (the beneficiary account address) to pay for some or all of `grantee`'s transaction fees. `allowance` defines what kind of fee allowance (`BasicAllowance` or `PeriodicAllowance`, see below) is granted to `grantee`. `allowance` accepts an interface which implements `FeeAllowanceI`, encoded as `Any` type. There can be only one existing fee grant allowed for a `grantee` and `granter`, self grants are not allowed.

proto/cosmos/feegrant/v1beta1/feegrant.proto

```
// Grant is stored in the KVStore to record a grant with full
context
message Grant {
 // granter is the address of the user granting an allowance
of their funds.
 string granter = 1 [(cosmos_proto.scalar) =
"cosmos.AddressString"];

 // grantee is the address of the user being granted an
allowance of another user's funds.
 string grantee = 2 [(cosmos_proto.scalar) =
"cosmos.AddressString"];

 // allowance can be any of basic, periodic, allowed fee
allowance.
 google.protobuf.Any allowance = 3
[(cosmos_proto.accepts_interface) =
"cosmos.feegrant.v1beta1.FeeAllowanceI"];
}
```

See full example on GitHub

`FeeAllowanceI` looks like:

x/feegrant/fees.go

```
// FeeAllowance implementations are tied to a given fee
delegator and delegatee,
// and are used to enforce fee grant limits.
type FeeAllowanceI interface {
     // Accept can use fee payment requested as well as
timestamp of the current block
     // to determine whether or not to process this. This is
checked in
     // Keeper.UseGrantedFees and the return values should
match how it is handled there.
     //
     // If it returns an error, the fee payment is rejected,
otherwise it is accepted.
     // The FeeAllowance implementation is expected to update
it's internal state
     // and will be saved again after an acceptance.
     //
     // If remove is true (regardless of the error), the
FeeAllowance will be deleted from storage
     // (eg. when it is used up). (See call to RevokeAllowance
in Keeper.UseGrantedFees)
     Accept(ctx sdk.Context, fee sdk.Coins, msgs []sdk.Msg)
(remove bool, err error)

     // ValidateBasic should evaluate this FeeAllowance for
internal consistency.
     // Don't allow negative amounts, or negative periods for
example.
     ValidateBasic() error

     // ExpiresAt returns the expiry time of the allowance.
     ExpiresAt() (*time.Time, error)
}
```

See full example on GitHub

Fee Allowance types

There are two types of fee allowances present at the moment:

```
BasicAllowance
PeriodicAllowance
AllowedMsgAllowance
```

BasicAllowance

`BasicAllowance` is permission for `grantee` to use fee from a `granter`'s account.

If any of the `spend_limit` or `expiration` reaches its limit, the grant will be removed from the state.

proto/cosmos/feegrant/v1beta1/feegrant.proto

```
// BasicAllowance implements Allowance with a one-time grant
of coins
// that optionally expires. The grantee can use up to
SpendLimit to cover fees.
message BasicAllowance {
 option (cosmos_proto.implements_interface) =
"cosmos.feegrant.v1beta1.FeeAllowanceI";
 option (amino.name)                        =
"cosmos-sdk/BasicAllowance";

 // spend_limit specifies the maximum amount of coins that can
be spent
 // by this allowance and will be updated as coins are spent.
If it is
 // empty, there is no spend limit and any amount of coins can
be spent.
 repeated cosmos.base.v1beta1.Coin spend_limit = 1 [
   (gogoproto.nullable)     = false,
   (amino.dont_omitempty)   = true,
   (gogoproto.castrepeated) =
"github.com/cosmos/cosmos-sdk/types.Coins"
 ];
```

See full example on GitHub

> `spend_limit` is the limit of coins that are allowed to be used from the `granter` account. If it is empty, it assumes there's no spend limit, `grantee` can use any number of available coins from `granter` account address before the expiration.
> `expiration` specifies an optional time when this allowance expires. If the value is left empty, there is no expiry for the grant.
> When a grant is created with empty values for `spend_limit` and `expiration`, it is still a valid grant. It won't restrict the `grantee` to use any number of coins from `granter` and it won't have any expiration. The only way to restrict the `grantee` is by revoking the grant.

PeriodicAllowance

`PeriodicAllowance` is a repeating fee allowance for the mentioned period, we can mention when the grant can expire as well as when a period can reset. We can also define the maximum number of coins that can be used in a mentioned period of time.

proto/cosmos/feegrant/v1beta1/feegrant.proto

```
// PeriodicAllowance extends Allowance to allow for both a
maximum cap,
// as well as a limit per time period.
message PeriodicAllowance {
 option (cosmos_proto.implements_interface) =
"cosmos.feegrant.v1beta1.FeeAllowanceI";
 option (amino.name)                        =
"cosmos-sdk/PeriodicAllowance";

 // basic specifies a struct of `BasicAllowance`
 BasicAllowance basic = 1 [(gogoproto.nullable) = false,
(amino.dont_omitempty) = true];

 // period specifies the time duration in which
period_spend_limit coins can
 // be spent before that allowance is reset
 google.protobuf.Duration period = 2
     [(gogoproto.stdduration) = true, (gogoproto.nullable) =
false, (amino.dont_omitempty) = true];

 // period_spend_limit specifies the maximum number of coins
that can be spent
 // in the period
 repeated cosmos.base.v1beta1.Coin period_spend_limit = 3 [
   (gogoproto.nullable)     = false,
   (amino.dont_omitempty)   = true,
   (gogoproto.castrepeated) =
"github.com/cosmos/cosmos-sdk/types.Coins"
 ];

 // period_can_spend is the number of coins left to be spent
before the period_reset time
 repeated cosmos.base.v1beta1.Coin period_can_spend = 4 [
   (gogoproto.nullable)     = false,
   (amino.dont_omitempty)   = true,
   (gogoproto.castrepeated) =
"github.com/cosmos/cosmos-sdk/types.Coins"
```

```
 ];

 // period_reset is the time at which this period resets and a
new one begins,
 // it is calculated from the start time of the first
transaction after the
 // last period ended
 google.protobuf.Timestamp period_reset = 5
     [(gogoproto.stdtime) = true, (gogoproto.nullable) =
false, (amino.dont_omitempty) = true];
}
```

See full example on GitHub

`basic` is the instance of `BasicAllowance` which is optional for periodic fee allowance. If empty, the grant will have no `expiration` and no `spend_limit`.
`period` is the specific period of time, after each period passes, `period_can_spend` will be reset.
`period_spend_limit` specifies the maximum number of coins that can be spent in the period.
`period_can_spend` is the number of coins left to be spent before the period_reset time.
`period_reset` keeps track of when a next period reset should happen.

AllowedMsgAllowance

`AllowedMsgAllowance` is a fee allowance, it can be any of `BasicFeeAllowance`, `PeriodicAllowance` but restricted only to the allowed messages mentioned by the granter.

proto/cosmos/feegrant/v1beta1/feegrant.proto

```
// AllowedMsgAllowance creates allowance only for specified
message types.
message AllowedMsgAllowance {
 option (gogoproto.goproto_getters)         = false;
 option (cosmos_proto.implements_interface) =
"cosmos.feegrant.v1beta1.FeeAllowanceI";
 option (amino.name)                        =
"cosmos-sdk/AllowedMsgAllowance";

 // allowance can be any of basic and periodic fee allowance.
```

```
 google.protobuf.Any allowance = 1
[(cosmos_proto.accepts_interface) =
"cosmos.feegrant.v1beta1.FeeAllowanceI"];

 // allowed_messages are the messages for which the grantee
has the access.
 repeated string allowed_messages = 2;
}
```

See full example on GitHub

`allowance` is either `BasicAllowance` or `PeriodicAllowance`. `allowed_messages` is array of messages allowed to execute the given allowance.

FeeGranter flag

`feegrant` module introduces a `FeeGranter` flag for CLI for the sake of executing transactions with fee granter. When this flag is set, `clientCtx` will append the granter account address for transactions generated through CLI.

```
client/cmd.go
if clientCtx.FeePayer == nil ||
flagSet.Changed(flags.FlagFeePayer) {
     payer, _ := flagSet.GetString(flags.FlagFeePayer)

     if payer != "" {
          payerAcc, err := sdk.AccAddressFromBech32(payer)
          if err != nil {
               return clientCtx, err
          }

          clientCtx = clientCtx.WithFeePayerAddress(payerAcc)
     }
}
```

See full example on GitHub

```
client/tx/tx.go
err = Sign(txf, clientCtx.GetFromName(), tx, true)
```

See full example on GitHub

```
x/auth/tx/builder.go
func (w *wrapper) SetFeeGranter(feeGranter sdk.AccAddress) {
     if w.tx.AuthInfo.Fee == nil {
          w.tx.AuthInfo.Fee = &tx.Fee{}
     }
```

```
	w.tx.AuthInfo.Fee.Granter = feeGranter.String()

	// set authInfoBz to nil because the cached authInfoBz no
longer matches tx.AuthInfo
	w.authInfoBz = nil
}
```

See full example on GitHub

```
proto/cosmos/tx/v1beta1/tx.proto
// Fee includes the amount of coins paid in fees and the
maximum
// gas to be used by the transaction. The ratio yields an
effective "gasprice",
// which must be above some miminum to be accepted into the
mempool.
message Fee {
 // amount is the amount of coins to be paid as a fee
 repeated cosmos.base.v1beta1.Coin amount = 1
     [(gogoproto.nullable) = false, (gogoproto.castrepeated) =
"github.com/cosmos/cosmos-sdk/types.Coins"];

 // gas_limit is the maximum gas that can be used in
transaction processing
 // before an out of gas error occurs
 uint64 gas_limit = 2;

 // if unset, the first signer is responsible for paying the
fees. If set, the specified account must pay the fees.
 // the payer must be a tx signer (and thus have signed this
field in AuthInfo).
 // setting this field does *not* change the ordering of
required signers for the transaction.
 string payer = 3 [(cosmos_proto.scalar) =
"cosmos.AddressString"];

 // if set, the fee payer (either the first signer or the
value of the payer field) requests that a fee grant be used
 // to pay fees instead of the fee payer's own balance. If an
appropriate fee grant does not exist or the chain does
 // not support fee grants, this will fail
 string granter = 4 [(cosmos_proto.scalar) =
"cosmos.AddressString"];
}
```

See full example on GitHub

Example cmd:

```
./simd tx gov submit-proposal --title="Test Proposal"
--description="My awesome proposal" --type="Text" --from
validator-key
--fee-granter=cosmos1xh44hxt7spr67hqaa7nyx5gnutrz5fraw6grxn
--chain-id=testnet --fees="10stake"
```

Granted Fee Deductions

Fees are deducted from grants in the `x/auth` ante handler. To learn more about how ante handlers work, read the Auth Module AnteHandlers Guide.

Gas

In order to prevent DoS attacks, using a filtered `x/feegrant` incurs gas. The SDK must assure that the `grantee`'s transactions all conform to the filter set by the `granter`. The SDK does this by iterating over the allowed messages in the filter and charging 10 gas per filtered message. The SDK will then iterate over the messages being sent by the `grantee` to ensure the messages adhere to the filter, also charging 10 gas per message. The SDK will stop iterating and fail the transaction if it finds a message that does not conform to the filter.

WARNING: The gas is charged against the granted allowance. Ensure your messages conform to the filter, if any, before sending transactions using your allowance.

Pruning

A queue in the state maintained with the prefix of expiration of the grants and checks them on EndBlock with the current block time for every block to prune.

State

FeeAllowance

Fee Allowances are identified by combining `Grantee` (the account address of fee allowance grantee) with the `Granter` (the account address of fee allowance granter).

Fee allowance grants are stored in the state as follows:

- Grant: `0x00 | grantee_addr_len (1 byte) | grantee_addr_bytes | granter_addr_len (1 byte) | granter_addr_bytes -> ProtocolBuffer(Grant)`

`x/feegrant/feegrant.pb.go`

```
// Grant is stored in the KVStore to record a grant with full
context
type Grant struct {
     // granter is the address of the user granting an
allowance of their funds.
     Granter string
`protobuf:"bytes,1,opt,name=granter,proto3"
json:"granter,omitempty"`
     // grantee is the address of the user being granted an
allowance of another user's funds.
     Grantee string
`protobuf:"bytes,2,opt,name=grantee,proto3"
json:"grantee,omitempty"`
     // allowance can be any of basic, periodic, allowed fee
allowance.
     Allowance *types1.Any
`protobuf:"bytes,3,opt,name=allowance,proto3"
json:"allowance,omitempty"`
}
```

See full example on GitHub

FeeAllowanceQueue

Fee Allowances queue items are identified by combining the `FeeAllowancePrefixQueue` (i.e., 0x01), `expiration`, `grantee` (the account address of fee allowance grantee), `granter` (the account address of fee allowance granter). Endblocker checks `FeeAllowanceQueue` state for the expired grants and prunes them from `FeeAllowance` if there are any found.

Fee allowance queue keys are stored in the state as follows:

> Grant: `0x01 | expiration_bytes | grantee_addr_len (1 byte) | grantee_addr_bytes | granter_addr_len (1 byte) | granter_addr_bytes -> EmptyBytes`

Messages

Msg/GrantAllowance

A fee allowance grant will be created with the `MsgGrantAllowance` message.

```
proto/cosmos/feegrant/v1beta1/tx.proto
// MsgGrantAllowance adds permission for Grantee to spend up
to Allowance
// of fees from the account of Granter.
```

```
message MsgGrantAllowance {
 option (cosmos.msg.v1.signer) = "granter";
 option (amino.name)           =
"cosmos-sdk/MsgGrantAllowance";

 // granter is the address of the user granting an allowance
of their funds.
 string granter = 1 [(cosmos_proto.scalar) =
"cosmos.AddressString"];

 // grantee is the address of the user being granted an
allowance of another user's funds.
 string grantee = 2 [(cosmos_proto.scalar) =
"cosmos.AddressString"];

 // allowance can be any of basic, periodic, allowed fee
allowance.
 google.protobuf.Any allowance = 3
[(cosmos_proto.accepts_interface) =
"cosmos.feegrant.v1beta1.FeeAllowanceI"];
}
```

See full example on GitHub

Msg/RevokeAllowance

An allowed grant fee allowance can be removed with the `MsgRevokeAllowance` message.

proto/cosmos/feegrant/v1beta1/tx.proto

```
// MsgGrantAllowanceResponse defines the
Msg/GrantAllowanceResponse response type.
message MsgGrantAllowanceResponse {}

// MsgRevokeAllowance removes any existing Allowance from
Granter to Grantee.
message MsgRevokeAllowance {
 option (cosmos.msg.v1.signer) = "granter";
 option (amino.name)           =
"cosmos-sdk/MsgRevokeAllowance";

 // granter is the address of the user granting an allowance
of their funds.
 string granter = 1 [(cosmos_proto.scalar) =
"cosmos.AddressString"];
```

```
 // grantee is the address of the user being granted an
allowance of another user's funds.
 string grantee = 2 [(cosmos_proto.scalar) =
"cosmos.AddressString"];
}
```

See full example on GitHub

Events

The feegrant module emits the following events:

Msg Server

MsgGrantAllowance

Type	Attribute Key	Attribute Value
message	action	set_feegrant
message	granter	{granterAddress}
message	grantee	{granteeAddress}

MsgRevokeAllowance

Type	Attribute Key	Attribute Value
message	action	revoke_feegrant
message	granter	{granterAddress}
message	grantee	{granteeAddress}

Exec fee allowance

Type	Attribute Key	Attribute Value
message	action	use_feegrant
message	granter	{granterAddress}

message	grantee	{granteeAddress}

Prune fee allowances

Type	Attribute Key	Attribute Value
message	action	prune_feegrant
message	pruner	{prunerAddress}

Client

CLI

A user can query and interact with the `feegrant` module using the CLI.

Query

The `query` commands allow users to query `feegrant` state.

```
simd query feegrant --help
```

grant

The `grant` command allows users to query a grant for a given granter-grantee pair.

```
simd query feegrant grant [granter] [grantee] [flags]
```

Example:

```
simd query feegrant grant cosmos1.. cosmos1..
```

Example Output:

```
allowance:
 '@type': /cosmos.feegrant.v1beta1.BasicAllowance
 expiration: null
 spend_limit:
 - amount: "100"
   denom: stake
grantee: cosmos1..
granter: cosmos1..
```

grants

The `grants` command allows users to query all grants for a given grantee.

```
simd query feegrant grants [grantee] [flags]
```

Example:

```
simd query feegrant grants cosmos1..
```

Example Output:

```
allowances:
- allowance:
```

```
    '@type': /cosmos.feegrant.v1beta1.BasicAllowance
    expiration: null
    spend_limit:
    - amount: "100"
      denom: stake
  grantee: cosmos1..
  granter: cosmos1..
pagination:
  next_key: null
  total: "0"
```

Transactions

The `tx` commands allow users to interact with the `feegrant` module.

```
simd tx feegrant --help
```

grant

The `grant` command allows users to grant fee allowances to another account. The fee allowance can have an expiration date, a total spend limit, and/or a periodic spend limit.

```
simd tx feegrant grant [granter] [grantee] [flags]
```

Example (one-time spend limit):

```
simd tx feegrant grant cosmos1.. cosmos1.. --spend-limit 100stake
```

Example (periodic spend limit):

```
simd tx feegrant grant cosmos1.. cosmos1.. --period 3600 --period-limit 10stake
```

revoke

The `revoke` command allows users to revoke a granted fee allowance.

```
simd tx feegrant revoke [granter] [grantee] [flags]
```

Example:

```
simd tx feegrant revoke cosmos1.. cosmos1..
```

gRPC

A user can query the `feegrant` module using gRPC endpoints.

Allowance

The `Allowance` endpoint allows users to query a granted fee allowance.

```
cosmos.feegrant.v1beta1.Query/Allowance
```

Example:

```
grpcurl -plaintext \
    -d '{"grantee":"cosmos1..","granter":"cosmos1.."}' \
    localhost:9090 \
    cosmos.feegrant.v1beta1.Query/Allowance
```

Example Output:

```
{
 "allowance": {
   "granter": "cosmos1..",
   "grantee": "cosmos1..",
   "allowance":
{"@type":"/cosmos.feegrant.v1beta1.BasicAllowance","spendLimit
":[{"denom":"stake","amount":"100"}]}
 }
}
```

Allowances

The `Allowances` endpoint allows users to query all granted fee allowances for a given grantee.

```
cosmos.feegrant.v1beta1.Query/Allowances
```

Example:

```
grpcurl -plaintext \
   -d '{"address":"cosmos1.."}' \
   localhost:9090 \
   cosmos.feegrant.v1beta1.Query/Allowances
```

Example Output:

```
{
 "allowances": [
   {
     "granter": "cosmos1..",
     "grantee": "cosmos1..",
     "allowance":
{"@type":"/cosmos.feegrant.v1beta1.BasicAllowance","spendLimit
":[{"denom":"stake","amount":"100"}]}
   }
 ],
 "pagination": {
   "total": "1"
 }
}
```

Contrato.rs

A continuación, analicemos el contrato central del proyecto.

instanciar

●Esta función se llama cuando se implementa el contrato por primera vez y se utiliza para configurar información básica, incluido el nombre del contrato, la versión, la dirección del administrador y la configuración de tarifas.

ejecutar

El contrato Cosmwasm ejecuta una transacción mediante una llamada de mensaje:

●El tipo de mensaje se define en el archivo msg.rs.

●La lógica de ejecución del mensaje se define enexecute.rs.

Por lo tanto, podemos hacer coincidir la lógica con el tipo de mensaje que pasa el usuario.

```
1#[entry_point]
2pub fn execute(
3    deps: DepsMut,
4    env: Env,
5    info: MessageInfo,
6    msg: ExecuteMsg,
7) -> ContractResult<Response> {
8    match msg {
9        ExecuteMsg::DeployFeeGrant {
10            authz_granter,
11            authz_grantee,
12        } => execute::deploy_fee_grant(deps, env, authz_granter,
authz_grantee),
13        ExecuteMsg::UpdateAdmin { new_admin } =>
execute::update_admin(deps, info, new_admin),
14        ExecuteMsg::UpdateGrantConfig {
15            msg_type_url,
16            grant_config,
17        } => execute::update_grant_config(deps, info,
msg_type_url, grant_config),
18        ExecuteMsg::RemoveGrantConfig { msg_type_url } => {
19            execute::remove_grant_config(deps, info,
msg_type_url)
20        }
21        ExecuteMsg::UpdateFeeConfig { fee_config } =>
update_fee_config(deps, info, fee_config),
22        ExecuteMsg::RevokeAllowance { grantee } =>
revoke_allowance(deps, env, info, grantee),
23        ExecuteMsg::UpdateParams { params } =>
update_params(deps, info, params),
24    }
25}
```

consulta

Dado que Cosmwasm utiliza las bibliotecas relevantes para la gestión del estado:

●El tipo de mensaje de consulta se define en el archivo msg.rs.

●La lógica de la función de consulta se define en query.rs.

Por lo tanto, también podemos hacer coincidir la lógica con el tipo de mensaje que pasa el usuario.

```
pub fn query(deps: Deps, _env: Env, msg: QueryMsg) -> StdResult<Binary> {
    match msg {
        QueryMsg::GrantConfigByTypeUrl { msg_type_url } => to_json_binary(
            &query::grant_config_by_type_url(deps.storage, msg_type_url)?,
        ),
        QueryMsg::GrantConfigTypeUrls {} => {
            to_json_binary(&query::grant_config_type_urls(deps.storage)?)
        }
        QueryMsg::FeeConfig {} => to_json_binary(&query::fee_config(deps.storage)?),
        QueryMsg::Admin {} => to_json_binary(&query::admin(deps.storage)?),
        QueryMsg::Params {} => to_json_binary(&query::params(deps.storage)?),
    }
}
```