```typescript
//+--------------------------------------------------------------------------------------+
//|                                                                  account.ts |
//|                                               Copyright 2018, Dennis Jorgenson |
//+--------------------------------------------------------------------------------------+
"use strict";

import type { ISession } from "module/session";
import type { IAccess, TAccess } from "db/interfaces/state";
import type { IPublishResult, TResponse } from "api";

import UserToken, { setUserToken } from "cli/interfaces/user";

import { Select, Insert, Update } from "db/query.utils";
import { hashHmac } from "lib/crypto.util";
import { PrimaryKey } from "api";
import { isEqual } from "lib/std.util";
import { Session } from "module/session";

import * as States from "db/interfaces/state";
import * as Users from "db/interfaces/user";
import * as Brokers from "db/interfaces/broker";
import * as Environments from "db/interfaces/environment";

export interface IAccount {
  account: Uint8Array;
  alias: string;
  broker: Uint8Array;
  broker_name: string;
  owner: Uint8Array;
  owner_name: string;
  state: Uint8Array;
  status: TAccess;
  environment: Uint8Array;
  environ: string;
  margin_mode: "cross" | "isolated";
  hedging: boolean;
  broker_image_url: string;
  broker_website_url: string;
  owner_email: string;
  owner_image_url: string;
  rest_api_url: string;
  private_wss_url: string;
  public_wss_url: string;
  total_equity: number;
  isolated_equity: number;
  currency: Uint8Array;
  account_currency: string;
  currency_image_url: string;
  currency_state: Uint8Array;
  currency_status: string;
  balance: number;
  currency_equity: number;
  currency_isolated_equity: number;
  available: number;
  available_equity: number;
  equity_usd: number;
  frozen: number;
  order_frozen: number;
  borrow_frozen: number;
  unrealized_pnl: number;
  isolated_unrealized_pnl: number;
  coin_usd_price: number;
  margin_ratio: number;
  spot_available: number;
  liability: number;
  update_time: Date;
}

//+--------------------------------------------------------------------------------------+
//| Returns promises for all or 'verified new' accounts from .env;                   |
//+--------------------------------------------------------------------------------------+
export const Available = async (filter: "New" | "All"): Promise<Array<Partial<ISession>>> => {
  const accounts = process.env.APP_ACCOUNT;
  if (!accounts) return [];

  let sessions: Array<Partial<ISession>>;
  try {
    sessions = JSON.parse(accounts);
  } catch (e) {
    console.error("-> [Error] Failed to parse account keys");
    return [];
  }

  if (filter === "All") return sessions;

  const missing = await Promise.all(
    sessions.map(async (account) => {
      const { api, secret, phrase } = account;
      const isNew = (await Key({ api, secret, phrase })) === undefined;
      return isNew ? account : undefined;
    }),
  );

  return missing.filter((acc): acc is Partial<ISession> => acc !== undefined);
};

//+--------------------------------------------------------------------------------------+
//| Adds all new accounts recieved from ui or any internal source to the database;      |
//+--------------------------------------------------------------------------------------+
export const Add = async (props: Partial<IAccount>, session: Partial<ISession>): Promise<IPublishResult<IAccount>> => {
  const exists = await Key(session);

  if (exists) {
    setUserToken({ error: 312, message: `Duplicate account ${props.alias} exists.` });
    return { key: undefined, response: { success: false, code: 312, response: `error`, rows: 0, context: "Account.Add" } };
  }

  const hmac = await hashHmac(session);

  if (hmac) {
    const slot = Math.floor(Math.random() * 82 + 1);
    const hash = Buffer.from([slot, hmac.charCodeAt(slot), hmac.charCodeAt(slot + 1)]);
    const account: Partial<IAccount> = {
      account: hash,
      alias: props.alias,
      owner: props.owner || (await Users.Key({ username: props.owner_name })) || undefined,
      broker: props.broker || (await Brokers.Key({ name: props.broker_name })) || undefined,
      state: props.state || (props.status ? await States.Key<IAccess>({ status: props.status }) : undefined) || undefined,
      environment: props.environment || (await Environments.Key({ environ: props.environ })) || undefined,
      total_equity: 0,
      isolated_equity: 0,
      rest_api_url: props.rest_api_url,
      private_wss_url: props.private_wss_url,
      public_wss_url: props.public_wss_url,
    };
    const result = await Insert<IAccount>(account, { table: `account` });
    return { key: PrimaryKey(account, ["account"]), response: result };
  }
  setUserToken({ error: 315, message: `Invalid session credentials.` });
  return { key: undefined, response: { success: false, code: 315, response: `error`, rows: 0, context: "Account.Add" } };
};

//+--------------------------------------------------------------------------------------+
//| Updates the account (master) from the API (select fields) or the UI;              |
//+--------------------------------------------------------------------------------------+
export const Publish = async (props: Partial<IAccount>): Promise<IPublishResult<IAccount>> => {
  if (!isEqual(Session().account!, props.account!)) {
    setUserToken({ error: 315, message: `Unauthorized account publication; invalid session account` }, props);
    throw new Error(UserToken().message);
  }

  const account = await Fetch({ account: props.account });

  if (!account) {
    setUserToken({ error: 315, message: `Unauthorized account publication; invalid session account` }, props);
    throw new Error(UserToken().message);
  }

  const [current] = account;
  const revised: Partial<IAccount> = {
    account: current.account,
    total_equity: isEqual(props.total_equity!, current.total_equity!) ? undefined : props.total_equity,
    isolated_equity: isEqual(props.isolated_equity!, current.isolated_equity!) ? undefined : props.isolated_equity,
    update_time: isEqual(props.update_time!, current.update_time!) ? undefined : props.update_time,
  };
  const result: TResponse = await Update(revised, { table: `account`, keys: [[`account`]] });
  return { key: PrimaryKey(current, ["account"]), response: result };
};

//+--------------------------------------------------------------------------------------+
//| Updates the account (detail) from the API (select fields) or the UI;              |
//+--------------------------------------------------------------------------------------+
export const PublishDetail = async (props: Partial<IAccount>): Promise<IPublishResult<IAccount>> => {
  if (!props.account || !props.currency) {
    setUserToken({ error: 315, message: `Unauthorized account publication; invalid session account` }, props);
    throw new Error(UserToken().message);
  }

  const account = await Fetch({ account: props.account, currency: props.currency });

  if (account) {
    const [current] = account;
    const revised: Partial<IAccount> = {
      account: current.account,
      currency: current.currency,
      balance: isEqual(props.balance!, current.balance!) ? undefined : props.balance,
      currency_equity: isEqual(props.currency_equity!, current.currency_equity!) ? undefined : props.currency_equity,
      currency_isolated_equity: isEqual(props.currency_isolated_equity!, current.currency_isolated_equity!) ? undefined : props.currency_isolated_equity,
      available: isEqual(props.available!, current.available!) ? undefined : props.available,
      available_equity: isEqual(props.available_equity!, current.available_equity!) ? undefined : props.available_equity,
      equity_usd: isEqual(props.equity_usd!, current.equity_usd!) ? undefined : props.equity_usd,
      frozen: isEqual(props.frozen!, current.frozen!) ? undefined : props.frozen,
      order_frozen: isEqual(props.order_frozen!, current.order_frozen!) ? undefined : props.order_frozen,
      borrow_frozen: isEqual(props.borrow_frozen!, current.borrow_frozen!) ? undefined : props.borrow_frozen,
      unrealized_pnl: isEqual(props.unrealized_pnl!, current.unrealized_pnl!) ? undefined : props.unrealized_pnl,
      isolated_unrealized_pnl: isEqual(props.isolated_unrealized_pnl!, current.isolated_unrealized_pnl!) ? undefined : props.isolated_unrealized_pnl,
      coin_usd_price: isEqual(props.coin_usd_price!, current.coin_usd_price!) ? undefined : props.coin_usd_price,
      margin_ratio: isEqual(props.margin_ratio!, current.margin_ratio!) ? undefined : props.margin_ratio,
      spot_available: isEqual(props.spot_available!, current.spot_available!) ? undefined : props.spot_available,
      liability: isEqual(props.liability!, current.liability!) ? undefined : props.liability,
      update_time: isEqual(props.update_time!, current.update_time!) ? undefined : props.update_time,
    };

    const result: TResponse = await Update(revised, { table: `account_detail`, keys: [[`account`], [`currency`]] });

    setUserToken({ error: result.code, message: result.success ? `Account details update applied.` : `Account details update failed.` });
    return { key: PrimaryKey(current, ["account", "currency"]), response: result } as IPublishResult<IAccount>;
  }

  const result = await Insert<IAccount>(props, { table: `account_detail` });

  setUserToken({ error: result.code, message: result.success ? `New Account details Imported.` : `Failed to import account details.` });
  return { key: PrimaryKey(props, ["account", "currency"]), response: result } as IPublishResult<IAccount>;
};

//+--------------------------------------------------------------------------------------+
//| Returns an account key for sessions provided (configured) in the .env;             |
//+--------------------------------------------------------------------------------------+
export const Key = async (props: Partial<ISession>): Promise<IAccount["account"] | undefined> => {
  const { account, api, secret, phrase } = props;

  if (account) {
    const result = await Fetch({ account });
    return result === undefined ? undefined : account;
  } else {
    if (api && secret && phrase) {
      const accounts = await Fetch({});

      if (accounts)
        for (const { account } of accounts) {
          const hmac = await hashHmac(props);

          if (hmac) {
            const slot = account![0];
          const hash = Buffer.from([slot, hmac.charCodeAt(slot), hmac.charCodeAt(slot + 1)]);

          if (isEqual(hash, account!)) return hash;
        }
      }
    }
  }
  return undefined;
};

//+-----------------------------------------------------------------------------------+
//| Returns accounts from supplied params; returns all on empty props set {};          |
//+-----------------------------------------------------------------------------------+
export const Fetch = async (props: Partial<IAccount>): Promise<Array<Partial<IAccount>> | undefined> => {
  const result = await Select<IAccount>(props, { table: `vw_accounts` });
  return result.length ? result : undefined;
};
```