# FS3 - Esercizio per le vacanze

> *NB: Il seguente esercizio va eseguito solamente dopo aver terminato il modulo React, e di conseguenza dopo aver consegnato tutti i suoi esercizi obbligatori .*

Le crypto o cryptovalute sono un interessante strumento finanziario di recente creazione. A livello finanziario, ogni cripto è associata ad un mercato. Portiamo un piccolo esempio: BTCUSDT è un mercato in cui si compra BTC con USDT e si vende BTC per USDT. In particolare:

- BTC è il `base_asset`;
- USDT è il `quote_asset`;
- il nome completo del mercato è base_asset+quote_asset;

**Si richiede la realizzazione di una piccola applicazione sviluppata in React, in grado di poter visualizzare informazioni utili rispetto al mercato delle criptovalute.**

L'applicazione richiesta, deve essere di tipo multipagina con le seguenti caratteristiche:

- pagina `/markets`:
  - Renderizza attraverso una tabella (`datatable`, vedi sezione successiva), i dati relativi ai mercati. In modo particolare, sono richieste le seguenti colonne:
    - Nome mercato;
    - Base asset;
    - Quote asset;
    - Prezzo;
    - [Opzionale] Variazione 24h;
  - Tramite input e select dedicate, permette filtraggio e ordinamento nel modo seguente:
    - Ricerca: attraverso un input dedicato, filtrare i risultati in maniera da mostrare solo quelli il cui nome combaci con quello digitato dall'utente (case-insensitive);
    - Ordinamento: Realizzare un sistema capace di gestire un ordinamento dinamico, ossia gestibile sulla base dell'utente. In modo particolare, necessiteremo di select multipli. La direzione di ordinamento:
      - Crescente;
      - Decrescente;
    - E il campo da usare per ordinare:
      - Nome;
      - Cambiamento di prezzo 24h;
  - La pagina, accetta `base_asset` come `query_parameter` :
    - Se l'indirizzo della pagina include un `base_asset` i dati devono essere filtrati di conseguenza;
    - Filtri e ordinamento della pagina devono continuare a funzionare regolarmente;
    - Ex. /markets?base_asset=BTC dovrà mostrare solamente tutti i mercati con `base_asset=BTC` (case-insensitive);
- pagina `/assets`:

    - Come la pagina sopra, attraverso una tabella deve mostrare tutti i base_assets in **maniera univoca** , le colonne richieste sono:
        - Asset;
        - Numero mercati (Tutti i mercati in cui l'asset compare come base_asset);
    - Al click su un item, deve portare l'utente nella pagina markets, attaccando nei query_parameter il `base_asset` selezionato;

### Requisiti tecnici

- Gestire il routing dell'applicazione attraverso `react-router` , le pagine non esistenti devono renderizzare l'utente su /assets, come lo deve fare la pagina /;
- Creare un `custom-hook` `useFetchApi` in grado di gestire le nostre richieste API in maniera smart ed elegante, evitando ripetizioni di codice. In particolare:
    - Deve gestire loading, errori e dati di risposta della chiamata API passata come parametro;
    - La soluzione migliore, vede l'utilizzo di un reducer per la gestione dello stato interno (loading, error,data);
    - tutte le chiamate API, dovranno essere fatte utilizzando questo hook;
- I dati che dovremmo renderizzare in `/markets` e `/assets` sono molti: occorre renderizzarli utilizzando una libreria di tipo `datatable` dedicata per evitare il crash dell'applicazione. Alcuni esempi sono:
    - react-data-table-component oppure
    - material-table.
- Tutti i dati devono essere caricati al loading dell'applicazione: è fondamentale gestirli in maniera `globale` in modo da evitare chiamate API inutili ogni volta che cambiamo pagina. Per fare questo, possiamo utilizzare un `context` oppure il meccanismo di `HOC`
- Utilizzare tailwind come libreria css;

### Api da utilizzare

È richiesto l'utilizzo delle api di Binance per la realizzazione dell'esercizio. Guardate solo le sezioni `introduction` , `generalInfo` e `marketDataEndpoints` . In modo particolare, gli endpoint utili nell'esecuzione dell'esercizio sono i seguenti

- `GET /api/v3/exchangeInfo` - Restituisce le configurazioni dell'exchange, oltre che tutti i symbols (mercati disponibili). Da utilizzare in `/assets` e `/markets/
- `GET /api/v3/ticker/price` - Restituisce i prezzi di uno o più mercati, da utilizzare in `/markets`
- `GET /api/v3/ticker/24hr` - Restituisce le variazioni di prezzo nelle ultime 24h di un o più mercati, da utilizzare in `/markets` ;

Attenzione! Le API di Binance hanno un meccansimo interno di `rate-limit` , che va a `pesare` ogni vostra richiesta. Nel caso Binance vedesse troppe richieste in un breve arco temporale, vi bloccherà l'IP per qualche minuto. Maggiori info nella documentazione.