Aurox Wallet Extension Code Style

Commit messages, git branches name formatting

Please refer to https://github.com/angular/angular.js/blob/master/DEVELOPERS.md#-git-commit-guidelines

Terminology

Entity: Refers to any of the components of the extension, such as service_worker, content_script, and all the ui frames such as popup, connect, hardware, etc

Frame: Refers to UI html pages that are rendered within the context of the extension, such as popup, hardware, etc…

Application file structure

```
src/
  /common  ...............  Holds code that are shared across all
entities of the extension
  /contentScript
  /injectScript
  /serviceWorker
  /ui
    /common  .............  Common code for the ui goes here, such as
utils, formatters, form helpers, etc...
    /components  .........  Components that are reused much be put in
this folder
    /frames  .............  Holds the code for each frame's entry point
    /hooks  ..............  All hooks must be placed in here
    /pages  ..............  The pages that frames will reference for
internal navigation should go here
    types  ...............  A temporary folder to hold the types used
only by the ui frames, it must be moved to common eventually, but
requires a lot of polishing
```

React Component structure

```
SomeComponent/
  SomeComponentBigPart/  .........  If there is a component that is
ONLY used by this component and it's very complex, it's worth putting
in it's own folder
    index.ts
    SomeComponentBigPart.tsx
    SomeComponentBigPartSection1.tsx
    SomeComponentBigPartSection2.tsx
  index.ts  ......................  Must re-export SomeComponent as
default and ONLY SomeComponent. If SomeComponentPart1 is going to be
needed elsewhere, then it should be moved to common. type.ts may be re-
exported
  SomeComponent.tsx
  SomeComponentPart1.tsx
  SomeComponentPart2.tsx
  types.ts  ......................  If there are types that are shared
across the parts of this component then they can be put in this file.
Moving to common type defs is preferable, only use in exceptional cases
  utils.ts  ......................  If there are util methods that will
only ever be used by this component then this is a good place for this
```

Imports for the component/page sequence

```
// ---------- Global imports (from node_modules) ----------
import React, { memo, useState, useEffect, useCallback } from "react";
import Decimal from "decimal.js";
import clsx from "clsx";

// ---------- MUI imports ----------
// ----- styles -----
import makeStyles from "@mui/styles/makeStyles";
import { Theme } from "@mui/material/styles";

// ----- components -----
import { Divider, Tooltip, Typography } from "@mui/material";

// ----- icons -----
import {
  Twitter as TwitterIcon,
  Facebook as FacebookIcon,
} from "@mui/icons-material";

// ---------- App-level imports e.g. config/data/utils/hooks ----------
import { NETWORKS, NETWORK_IDENTIFIERS } from "config";
import {
  GraphQLMarketsAPICoinQueryIdentificationType,
  SupportedNetworkIdentifier,
} from "types";

// ---------- Library (app-level) components ----------
import EqualCols from "pages/Library/components/EqualCols";
import InfoPair from "pages/Library/components/InfoPair";

// ---------- Local component/page imports e.g. hooks/components/utils
----------
import { formatAbbreviated } from "./utils";
import {
  useCoin,
  useAccountTokenHoldings,
  useTokenPriceChangeData,
} from "./hooks";
```

Component logical part sequence

```
// ALWAYS export the props interface
export interface ComponentProps {
  value?: string;
  onClick?: () => void;
  rootClassName?: string;
}

// ALWAYS use this syntax, if you need to memoize then do: export
default memo(function Component...
// NEVER destructure props in place
// NEVER use this syntax: export const Component = (props:
ComponentProps) => {...}
export default function Component(props: ComponentProps) {
  const { value, onClick, rootClassName } = props;

  const classes = useStyles();

  // Local state declaration;
  const [price, setPrice] = useState<Nullable<string>>("0");

  // App-level and custom hooks execution;
  const params = useParams();

  // Event handlers;
  const handlePriceChange = (newPrice: string) => {
    setPrice(newPrice);
  };

  // Local effects;
  useEffect(() => {
    setPrice(value);
  }, [value]);

  // Render;
  return (<></>)
}
```

Styling

- use theme values as wide as possible from `src/theme/index.ts`. Feel free to add your values for colors, sizes, etc.
- you may use recommended MUI styling with styled: https://mui.com/system/styled/
- use the `$+$sibling` selector if you need to reference the sibling element

```
const useStyles = makeStyles((theme: Theme) => ({
  section: {
    "&+$section": {
      marginTop: 30,
    },
  },
  title: {
    color: theme.palette.text.primary,
  },
}));
```

Other guidelines

- use `<Box />` with `sx` styles widely

```
<Box component="span" ml={0.5} sx={{ width: "100%" }}>
  <CopyToClipboard text="https://etherscan.io/address
/0xc02aaa39b223fe8d0a0e5c4f27ead9083c756cc2" />
</Box>
```

- use fewer ternary operators for rendering:

```
// good;
return (
  <>
    {coin && <Header title={coin.fn} />}
  </>
);

// bad;
return (
  <>
    {coin ? <Header title={coin.fn} /> : null}
  </>
);
```

- always return a component instead of `null`:

```
// good;
return TIME_UNITS ? (
  <div className={classes.controls}>
    {Object.keys(TIME_UNITS).map((periodId) => (
      <PeriodButton
        key={periodId}
        period={periodId as Period}
        active={selectedPeriod === periodId}
        onClick={handleChangePeriod}
      />
    ))}
  </div>
) : <></>;

// bad;
return TIME_UNITS ? (
  <div className={classes.controls}>
    {Object.keys(TIME_UNITS).map((periodId) => (
      <PeriodButton
        key={periodId}
        period={periodId as Period}
        active={selectedPeriod === periodId}
        onClick={handleChangePeriod}
      />
    ))}
  </div>
) : null;
```

- use `memo`, `useMemo`, `useCallback` when it is really needed:

```
// good;
const handleImageError = () => setImageError(true);

// good (re-initializes only if `isFavorite` changes);
const handleToggleFavorite = useCallback(() => {
  setIsFavorite(!isFavorite);
}, [isFavorite]);

// bad (not needed, redundant);
const handleImageError = useCallback(() => {
  setImageError(true);
}, []);
```

- avoid unnecessary <></> fragment wrappers:

```
// good;
const tooltip = (
  <ul>
    <li>
      Volume{" "}
      <span className={classes.tooltipPrice}>
        {formatToAverage(coin.v24)}
      </span>
    </li>
  </ul>
);

// bad;
const tooltip = (
  <>
    <ul>
      <li>
        Volume{" "}
        <span className={classes.tooltipPrice}>
          {formatToAverage(coin.v24)}
        </span>
      </li>
    </ul>
  </>
);
```

- avoid installing new `npm` packages if something you need is already installed:

```
  "throttle-debounce": "5.0.0",
  "lodash": "4.17.15",
```

here above `lodash` has `lodash/throttle` and `lodash/debounce` and you use them instead of installing `throttle-debounce`.

- separate dependencies from devDependencies:

```
"dependencies": {
  "randomcolor": "0.6.2",
  "react": "17.0.2",
  "react-dom": "17.0.2",
},
"devDependencies": {
  "css-loader": "6.7.1",
  "eslint": "8.14.0",
}
```

- fix always install the exact npm package version:

```
// good;
"react": "17.0.2",
"react-dom": "17.0.2",

// bad;
"react": "^17.0.2",
"react-dom": "^17.0.2",
```

read about why it is important.

- use project `.eslintrc.json` and `.editorconfig` to format your files before committing them
- use props destructuring:

```
function CoinRow(props: CoinRowProps) {
  const { coin, onClick } = props;
}
```

- insert newlines after logical parts of a function body, e.g. variables declarations, `if`-clauses, function calls, `return` statements, etc:

```
// good;
useEffect(() => {
  const params = new URLSearchParams(location.search);
  const tag = params.get("tag");
  const tags = coinTags?.map((coin) => coin.name);

  if (tag && tags?.includes(tag)) {
    setTag(tag);
  }

  return null;
}, [location, coinTags]);

// bad;
useEffect(() => {
  const params = new URLSearchParams(location.search);
  const tag = params.get("tag");
  const tags = coinTags?.map((coin) => coin.name);
  if (tag && tags?.includes(tag)) {
    setTag(tag);
  }
  return null;
}, [location, coinTags]);
```

- avoid contextual event handlers:

```
// good;
const handleTabChange = (event: React.SyntheticEvent, tabName: string)
=> {
  setTab(tabName);
};

<Tabs onClick={handleTabChange} />

// bad;
<Tabs onClick={(event, tabName) => setTab(tabName)} />
```

- make sure default export and naming export is supported for the component:

```
import { Search, SearchProps } from "./Search";

export { SearchProps, Search };

export default Search;
```

- if some subfolder contains only one functional file, prevent using just a file instead:

```
// structure contains a single fucntion file;
hooks/
  index.ts
  useForm.ts

// use file instead;
hooks.ts

import { useForm } from "./hooks";
```

- move all constants, helpers, hooks, utils out from the component as it may be reused somewhere else:

```
// good;

import { clearText } from "./helpers";

export function Search() {
  clearText();
}

// bad;
export const clearText = (value: string) => value.replace('"', "");

export function Search() {
  clearText();
}
```

- consider breaking down large components with a lof or lines of code to smaller ones