Adding new micro frontends

Step-by-step guide to add a new micro frontend on Portal, from locally to deployed

Prerequisites:

- Clone Portal.
- Create new front-end app project (skip to next step if you already have a project) `npm create vite@latest`. Select React & Typescript on the terminal prompts (other types may need modification to make it work with portal current state)

After we have Portal and our frontend app ready, let's start configuring the frontend app first =>

1. **Add `.npmrc` file on root directory, and place this content, to provide necessary registry credentials for installing packages from our JFrog artifactory**

```
registry=https://artifacts.rbi.tech/artifactory/api/npm/npmjs-org-npm-proxy/
@rbal-modern-luka:registry=https://artifacts.rbi.tech/artifactory/api/npm/rbal-modern-luka-npm-host/
legacy-peer-deps=true
```

2. **Add necessary packages (ui-library, portal-shell, eslint-codestyle and luka-ufront-starter)**

```
"dependencies": {
    "@rbal-modern-luka/luka-portal-shell": "0.0.0-main-bb9e8eae", // check latest versions of these
packages
    "@rbal-modern-luka/ui-library": "0.0.0-main-06c7c76c",
        ...
}
"devDependencies": {
    "@rbal-modern-luka/eslint-codestyle": "0.0.0-main-7387d65f",
    "@rbal-modern-luka/luka-ufront-starter": "0.0.0-main-61b78ae2",
        ...
}
```

3. **Modify `tsconfig.json`**

```
{
  "compilerOptions": {
    "target": "ESNext",
    "useDefineForClassFields": true,
    "lib": ["DOM", "DOM.Iterable", "ESNext"],
    "allowJs": false,
    "skipLibCheck": true,
    "esModuleInterop": false,
    "allowSyntheticDefaultImports": true,
    "strict": true,
    "forceConsistentCasingInFileNames": true,
    "module": "ESNext",
    "moduleResolution": "Node",
    "resolveJsonModule": true,
    "isolatedModules": true,
    "noEmit": true,
    "jsx": "react-jsx",
    "jsxImportSource": "@emotion/react",
    "baseUrl": ".",
    "paths": {
      "~/*": ["./src/*"]
    }
  },
  "include": ["src"],
  "references": [{ "path": "./tsconfig.node.json" }]
}
```

4. **Modify `vite.config.ts`**

```
import { createUFrontViteBuild } from "@rbal-modern-luka/luka-ufront-starter";

export default createUFrontViteBuild({
  name: "{name-of-your-microfrontend}", // replace this placeholder with actual name of your micro
frontend
  serverPort: 15719, // may need to check available ports in portal/your pc
});
```

5. **Add a new `singleSpaEntry.ts` file on root directory. This file will act as the entry point of our app, and add this content**

```
import React from "react";
import { createRoot } from "react-dom/client";
import { ReactAppOrParcel } from "single-spa-react";
import { Root, RootProps } from "./Root";

interface MountProps extends RootProps {
  domElement: HTMLElement;
}

const lifecycles: ReactAppOrParcel<Record<string, never>> = {
  bootstrap() {
    return Promise.resolve();
  },
  mount(props: MountProps) {
    if (!root) {
      root = createRoot(props.domElement);
    }
    root.render(React.createElement(Root, props));
    return Promise.resolve();
  },
  unmount() {
    if (root) {
      root.unmount();
      root = null;
    }
    return Promise.resolve();
  },
  update() {
    return Promise.resolve();
  },
};

export const { bootstrap, mount, unmount, update } = lifecycles;
```

6. **Create root app component. Below example also has necessary context providers needed for translation and global portal context**

```
import { useMemo, useState } from "react";
import reactLogo from "./assets/react.svg";
import viteLogo from "/vite.svg";
import {
  I18nContext,
  I18nContextValue,
  PortalContext,
  PortalStore,
  translationsFactory,
  usePortalContext,
} from "@rbal-modern-luka/luka-portal-shell";
import { QueryClient, QueryClientProvider } from "react-query";

export interface RootProps {
  store: PortalStore;
}

const queryClient = new QueryClient();

export const Root: React.FunctionComponent<RootProps> = (props) => {
  const [count, setCount] = useState(0);

  const portalContext = usePortalContext(props.store);

  const i18nContext: I18nContextValue = useMemo(() => {
    const { preferredLanguage } = portalContext;

    return {
      tr: translationsFactory(preferredLanguage),
      lang: preferredLanguage,
    };
  }, [portalContext]);

  return (
    <QueryClientProvider client={queryClient}>
      <PortalContext.Provider value={portalContext}>
        <I18nContext.Provider value={i18nContext}>
          <div>
            <a href="https://vite.dev" target="_blank">
              <img src={viteLogo} className="logo" alt="Vite logo" />
            </a>
            <a href="https://react.dev" target="_blank">
              <img src={reactLogo} className="logo react" alt="React logo" />
            </a>
          </div>
          <h1>Vite + React</h1>
          <div className="card">
            <button onClick={() => setCount((count) => count + 1)}>
              count is {count}
            </button>
            <p>
              Edit <code>src/App.tsx</code> and save to test HMR
            </p>
          </div>
          <p className="read-the-docs">
            Click on the Vite and React logos to learn more
          </p>
        </I18nContext.Provider>
      </PortalContext.Provider>
    </QueryClientProvider>
  );
};
```

7. Now we are done of configurations needed from our new micro frontend. Now switch to the PORTAL project. Go to `proxyConfigs.ts` file, and add necessary proxy configurations needed for DEVELOPMENT mode only

```
const defaultConfigs: Record<string, "local" | "test" | "pilot"> = {
        ..., // existing configs
        myNewMicroFrontEnd: "test" // If you plan on having this deployed, leave it "test". If you only
need locally, set it "local"
}
```

8. Below in the same file, you will have `proxyConfigs` object, add proxy configuration there

```
myNewMicroFrontEnd: {
    pathPrefix: "/ufronts/{name-of-your-microfrontend}", // this will be the path the server routes to,
make sure it is the same in the Nginx configuration (helm chart, check step #13) when you have the app
deployed,
    test: defaultTestProxyConfig, // default test proxy config, where the target will be portal itself
    local: {
      ws: false,
      rewrite: localhostSingleSpaEntryRewrite, // function needed to rewrite `singleSpaEntry.ts` file
locally
      target: "http://localhost:15718", // Host and port of your running micro frontend
    },
  },
```

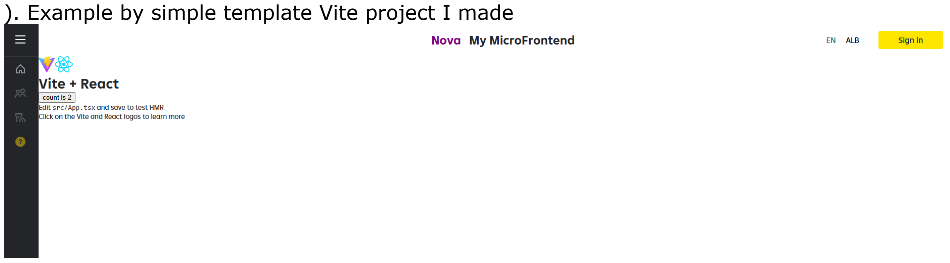
9. Now lastly, we need to display and route our micro frontend. You can do this by just adding this configuration on `appsettings.json` file under `public` folder

```
"my-new-microfrontend": {
      "id": "{name-of-your-microfrontend}", // should match name provided on `vite.config.ts` file on
your micro frontend
      "requiresAuthentication": false, // will be disabled if true, and the user is not logged in
      "onNavbar": true, // is displayed on left side navigation bar
      "onHome": true, // is displayed on Portal Home screen
      "allowSubModules": true, // whether it should render sub-modules on nav bar
      "title": { "en": "My MicroFrontend", "sq": "Mikrofrontendi im" }, // title of this micro frontend
in English and Albanian
      "category": "apps", // The category it belongs to. Default I have set all micro frontends to be
under "apps"
      "path": "/my-microfrontend", // path to access this micro frontend
      "icon": "", // icon for this module. Check UI-Library for available types.
      "description": {
        "en": "Retail customer and accounts", // description of this micro frontend in English and
Albanian
        "sq": "Klientët dhe llogaritë retail"
      }
    },
```

10. Now we are done. Just `npm start` both portal and micro frontend (micro front end should be started first), open portal, and you should be able to access micro frontend there (you need to either set default proxy config to "local", or override it by creating a `proxyOverrides.json` file, with json like this =>

```
{
        "myNewMicroFrontEnd": "local"
}
```

). Example by simple template Vite project I made



Edit `src/App.tsx` and save to test HMR
Click on the Vite and React logos to learn more

Adding the module in deployed environment, there are 2 aspects: Infrastructure (nginx proxying = /ufront/{name-of-your-microfrontend} => your-mfe.com) and Application (rendering module, registering app-level routes, options, etc...)

11. **[Infrastructure]** To properly **proxy** the MFE, you need to add config here rbal-charts/rbal-luka-portal/values.yaml at main · raiffeisen/rbal-charts. DevOps should add something like this

```
location ~ ^/ufronts/{name-of-your-microfrontend}/ { // must match the pathPrefix mentioned in
`proxyConfigs` object on portal "/ufronts/my-new-microfrontend"
        rewrite /ufronts/{name-of-your-microfrontend}/(.*) /$1 break;
        add_header Cache-Control 'must-revalidate';
        expires off;

        proxy_pass http://rbal-my-new-microfrontend; // the deployed url of your micro frontend
    }
```

12. **[Application]** To render an app, you need to add respective JSON configuration in the `appsettings.json` file, just like **step 11**.
    The appsettings.json file is mounted to **Portal** in runtime, from **AWS Secrets** of Portal. The name of secrets is `**rbal-luka-portal-secret-cm**`.
    The pod must be refresh so the new file gets updated.

Notes to keep in mind about CSS styles

The **PORTAL uses Emotion**, which injects styles at runtime with `<style>` tags.
By default, **Tailwind builds a separate `style.css` file**, loaded with a `<link>` tag.

If your app uses tailwind or similar static CSS libraries, in a micro-frontend setup this causes problems:

- CSS load **order is unpredictable** Emotion can override Tailwind (or vice versa).
- CSS files may **fail to load** if the host/portal doesn't serve them from the right path.

One solution is:

Add vite-plugin-css-injected-by-js with cssCodeSplit: false.
This bundles Tailwind CSS directly into the MF's JS and injects it as a <style> tag at runtime, making each MF fully self-contained and ensuring consistent style order with Emotion.

Some example how to do it, inside vite.config.ts file of your micro front end:

```
import { createUFrontViteBuild } from "@rbal-modern-luka/luka-ufront-starter";
import type { UserConfig } from "vite";
import cssInjectedByJsPlugin from "vite-plugin-css-injected-by-js";

const config = createUFrontViteBuild(
  {
    name: "{name-of-your-microfrontend}",
    serverPort: 14314,
    hasEmotionCss: false,
  },
  (config) => {
    const cfg = config as UserConfig;

    cfg.plugins = [...(cfg.plugins || []), cssInjectedByJsPlugin()];

    cfg.build = {
      ...cfg.build,
      cssCodeSplit: false,
    };

    return cfg;
  }
);

export default config;
```