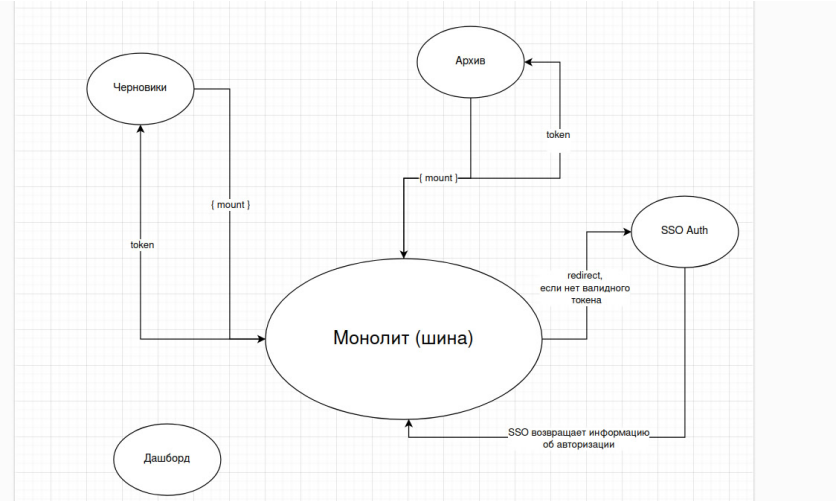
Микрофронты v1

Данная статья описывает первую, "сырую" версию архитектуры микрофронтов. Ожидаются доработки и обновления.

Что такое микрофронт?



Микрофронт - это приложение, которое может поддерживаться и разрабатываться отдельно, но существует только как дочернее, в рамках проекта-"шины". В нашем случае, "шина" - это microws-ervu, "ребёнок" - это микрофронт (дальше как пример будем рассматривать mfe-drafts aka "черновики" aka "ручной ввод"; его можно лицезреть тут).

Микрофронты представляют из себя обычные js-модули. Из одного микрофронта может быть экспортировано несколько файлов (модулей). Модули декларируются в конфигах приложений (см. ниже)

Пока что отсутствует техническая возможность запускать и вести качественную разработку микросервисов отдельно от проекта microws-ervu. Это обусловлено тем, что имеются проблемы с изоляцией стилей (в нашем случае, фреймворка Quasar), из-за чего наблюдаются проблемы с вёрсткой.

Как общаются микросервисы с "шиной" и друг с другом?

Поскольку микрофронты - это js-модули, мы можем экспортировать из них всё, что угодно. В том числе, мы можем вызывать функции, экспортируемые из "детей", с параметрами, существующими только в "шине". Ровно так сейчас и реализовано монтирование микрофронтов в microws-ervu и общение между ним и детьми.

Например, мы можем в шине объявить функцию, которая возвращает токен для авторизации запросов на бэкенд:

```
// microws-ervu/src/recruits/components/mfe/DraftsApp.vue - ""

...
'token-request': async function () {
        const { auth } = await import('@services/AuthService');
        if (auth) {
                const Authorization = await auth.getAccessToken();
                return Authorization;
        }
},
...
```

и запросить этот токен в дочернем приложении (см. token-request):

```
// mfe-drafts/src/utils/api.ts - ""

...
export function createApi(config: CreateAxiosDefaults): AxiosInstance {
  const instance = axios.create(config);

  instance.interceptors.request.use(
    async (config) => {
      const token = await fireMfeEventToContainer('token-request');
      config.headers.Authorization = `Bearer ${token}`;
      return config;
    },
    (error) => {
      return Promise.reject(error);
    },
  );

  return instance;
}
...
```

Теперь наши запросы в ребёнке будут авторизованы, и мы сможем их отправлять через axios.

Поскольку ребёнок может слать сообщение шине, то шина может сохранять это состояние, и к ней может обратиться уже другой ребёнок - механизм общения микрофронтов друг с другом.

Событие 'navigate' зарезервировано. Оно используется для синхронизации роутеров ребёнка и шины. Смотри файлы:

- **mfe-drafts/src/mfe-app-tools.js:75**
- **microws-ervu/src/recruits/mixins/mfe-wrapper.ts:41**

Как запустить проект на микрофронтах?

Внимание! Зависимости в микрофронтах нужно установить на node v20.*. Если версия будет ниже, то возможны ошибки при сборке.

В наших проектах используется технология Module Federation. Перед запуском нужно убедиться в том, что конфиги настроены следующим образом:

```
// microws-ervu/vue.config.js - ""

...
plugins: [
    new (require('circular-dependency-plugin'))({ exclude: /node_modules/ }),
    new ContextReplacementPlugin(/moment[/\\]locale$/, /ru|en/),
    // new BundleAnalyzerPlugin(),
        new ModuleFederationForWebpack4Plugin({
                name: 'microws',
        remotes: {
          drafts: "drafts@/mfe/drafts/remoteEntry.js",
          archive: 'archive@/mfe/archive/remoteEntry.js'
                },
        }),
],
...
```

```
// mfe-drafts/vue.config.js - ""

...
configureWebpack: (config) => {
    config.plugins.push(
      new ModuleFederationPlugin({
        name: 'drafts',
        filename: 'remoteEntry.js',
        exposes: {
          './drafts': './src/bootstrap',
        },
      }),
    );
},
...
```

В дочернем приложении мы экспоузим ("показываем вовне") файл **`src/bootstrap`**, который неймспейсим как **`drafts`**. В шине объявляем микросервис как **`remotes.drafts`** и в нём обращаемся к **`drafts@/mfe/drafts/remoteEntry.js`**, где **`drafts`** - это неймспейс в дочернем приложении, а **`/mfe/drafts/remoteEntry.js`** - путь до нужного нам файла в ребёнке (разделены "собакой" - особенность технологии).

Настройка devServer'ов.

URI /mfe/drafts/* используется для доступа к черновикам на проде, т.к. там они лежат в этой директории. При разработке необходимо проверить наличие следующих настроек в конфигах приложений:

```
// microws-ervu/vue.config.js - ""

...
devServer: {
    ...
    proxy: {
      '/mfe/drafts': {
        target: 'https://localhost:8081',
        changeOrigin: true,
        pathRewrite: path => path.replace('^mfe/drafts', '')
      },
      '/mfe/archive': {
        target: 'https://localhost:8082',
        changeOrigin: true,
        pathRewrite: path => path.replace('^mfe/archive', '')
      },
        ...
...
```

```
// mfe-drafts/vue.config.js - ""
...
devServer: process.env.PROXY_HOST
    ? {
        https: true,
        port: 8081,
        allowedHosts: [process.env.PROXY_HOST],
        webSocketServer: false,
        proxy: {
          '/': {
            bypass: function (req) {
              if (req.url.startsWith('/mfe/drafts')) {
                //        /mfe/drafts(.*)
                return req.url.replace('/mfe/drafts', '');
              }
            },
            target: `https://${process.env.PROXY_IP}`,
            headers: {
              Host: process.env.PROXY_HOST,
            },
          },
        },
      }
    : {
        port: 8081,
        allowedHosts: ['mfe-drafts', process.env.VUE_APP_HOST], // devServer  'Invalid Host Header'
      },
...
```

Для корректного запуска ребёнка требуется добавить в .env-файл пару PROXY_HOST=0.0.0.0 - дев-сервер проксирует localhost.

Мы видим, что в шине у нас есть прокси для **`localhost:8081`**, в котором подменяется строка '/mfe/drafts' на ''. Это нужно для корректной работы шины, т.к. веб-сервера с **`basePath === /mfe/drafts`** при разработке не запускается.

В ребёнке нам также следует добавить прокси для обработки запроса **`localhost:8081/mfe/drafts`** уже на веб-сервере ребёнка. Там мы также подменяем uri на пустую строку, чтобы отдавать нужные файлы.

Как запустить?

- **Остановить все текущие проекты, если они запущены**
- В проекте-шине перейти на ветку **mfe**: в ней хранится актуальное состояние разработки с использованием микрофронтов
- Установить зависимости (на 14-й или 16-й ноде), если ещё не установлены; цель - установить плагин для Module Federation
- Перейти в папку проекта-ребёнка, также предварительно установив зависимости, и запустить его в режиме разработки
- ПОСЛЕ ЭТОГО: запустить проект-шину в режиме разработки

На момент запуска проекта-шины должны быть запущены все микрофронты, т.к. webpack подгружает их до старта сборки шины.

Заключение.

На момент написания статьи происходит вывод релиза 1.5.3 на dev и test-контуры, в котором впервые на микрофронтах будет реализован Ручной ввод. После вывода релиза и актуализации ветки master необязательно нужно будет переходить на ветку mfe, достаточно будет вести разработку по стандартному флоу.

В процессе разработки инфраструктуры будут выходить статьи, описывающие изменения и обновления.

По troubleshooting'y обращаться в личку тг или на почту.