SmartApp SDK

Репозиторий: https://github.com/ExpressApp/smartapp-sdk

Библиотека предлагает:

Методы библиотеки SmartApp Bridge https://github.com/ExpressApp/smartapp-bridge

- Отправка ивента клиенту:

```
bridge?.sendClientEvent({
  method: string,
  params: object,
  timeout?: number // default 30s
})
```

- Отправка ивента боту:

```
bridge?.sendBotEvent({
   method: string,
   params: object,
   timeout?: number // default 30s
})
```

- `bridge?.enableLogs()` - включить сбор логов SmartApp (выключено по умолчанию);
- `bridge?.disableLogs()` - выключить сбор логов SmartApp (выключено по умолчанию);
- `bridge?.disableRenameParams()` – выключить переименование полей ивентов SmartApp из `camelCase` в `snake_case` при отправке ивента боту или клиенту, и наоборот, при получении (включено по умолчанию);
- `bridge?.enableRenameParams()` – включить переименование полей ивентов SmartApp из `camelCase` в `snake_case` при отправке ивента боту или клиенту, и наоборот, при получении (включено по умолчанию);

Ивент ready

Метод, отправляющий ивент должен вызываться первым после загрузки SmartApp

```
await ready({ timeout: number }) // default timeout is 30s
```

Редирект в другой смартапп

```
await openSmartApp({
  appId: string // уникальный идентификатор SmartApp e.g. "feature-smartapp",
})
```

Редирект в другой смартапп, передача поля meta

```
await openSmartApp({
  appId: string, // уникальный идентификатор SmartApp e.g. "feature-smartapp",
  meta: any, // "meta" может содержать любую информацию, которую необходимо передать
})
```

Пример редиректа в другой смартапп и передачи информации из поля meta

- SmartApp 1 отправляет ивент клиенту:

```
await openSmartApp({
  appId: "feature-smartapp",
  meta: {
    route: "/route-in-feature-smartapp"
  }
})
```

- Клиент получает ивент, сохраняет значение поля meta.
- Клиент открывает SmartApp 2 с appId ==="feature-smartapp"
- SmartApp 2 шлет ивент ready:

```
const response = await ready()
```

- В ответе на ready SmartApp 2 проверяет наличие поля openSmartAppMeta:

```
const meta = response?.payload?.openSmartAppMeta

if (meta) {
  history.push(`${meta?.route}`)
}
```

- SmartApp 2 выполняет необходимые действия с meta, которую SmartApp 1 отправляет клиенту, а клиент возвращает в ответе на ивент ready SmartApp 2.

Выход из смартапп на каталог

```
await exitSmartAppToCatalog()
```

Метод осуществляет выход из смартапп на каталог смартапп.

Получение параметров url SmartApp

```
const urlParams = useQuery() // react hook
```

Хук возвращает объект типа:

```
{
    platform: "web" | "ios" | "android",
    theme: "default" | "dark"
}
```

Загрузка файлов

- Загрузка одного файла через файловый менеджер Express:

```
const response = await bridge?.sendClientEvent({
     method: "upload_file",
     params: {
       type: string, // MIME type, pass "" for any
     },
   })
```

Метод возвращает объект с метаданными файла:

```
interface File {
  type?: string
  file?: string
  fileMimeType?: string
  fileName?: string
  filePreview?: string
  filePreviewHeight?: number
  filePreviewWidth?: number
  fileSize: number
  fileHash?: string
  fileEncryptionAlgo?: string
  chunkSize?: number
  fileId?: any
  key?: object
}

const file: File = { ...response.payload.record }
```

- Загрузка множества файлов через файловый менеджер Express:

```
const response = await bridge?.sendClientEvent({
      method: "upload_files",
      params: {
        type: string, // MIME type, pass "" for any
      },
    })
```

Метод возвращает массив объектов с метаданными файла:

```
const files: File[] = response.payload.records
```

Предпросмотр и скачивание файлов

```
await bridge?.sendClientEvent({
      method: "open_file",
      params: file: File,
    })
```

Ответа от клиента не приходит. Происходит открытие файлового менеджера Express.

Открытие настроек профиля

```
await openClientSettings() // only for iOS and Android
```

Метод отправляет клиенту ивент типа:

```
{
    "ref": string,
    "handler": "express",
    "type": "open_client_settings",
    "payload": {},
    "files": []
}
```

Ответа от клиента не приходит. Происходит открытие Настроек профиля пользователя Express.

Запрос чатов

```
const response = await getChats({ filter }: { filter: string | null })
```

Метод отправляет клиенту запрос типа:

```
{
    "ref": string,
    "handler": "express",
    "type": "get_chats",
    "payload": {
        "filter": string | null,
    },
    "files": []
}
```

И получает ответ типа:

```
{
    "ref": string,
    "status": "success",
    "data": {
      "chats": [
        {
          "group_chat_id": Uuid,
          "name": string,
          "avatar": string | null,
          "members_type": "cts|rts|hybrid",
          "is_trusted": boolean,
          "chat_type": "chat" | "group_chat" | "botx" | "channel",
        },
      ],
    }
}
```

Поиск по контактам (включая трастовые)

```
const response =
  await const searchCorporatePhonebook = ({ filter: 'andrey' }: { filter: string }) // 3 symbols min
```

Метод отправляет клиенту запрос типа:

```
{
    "ref": string,
    "handler": "express",
    "type": "get_chats",
    "payload": {
        "filter": string | null,
    },
    "files": []
}
```

И получает ответ типа:

```
{
  "ref":"9180b77e-2074-4a0c-bf37-c2bca61b144a",
          "type":"search_corporate_phonebook",
          "payload":{
    "status":"success",
            "data":{
      "corpPhonebookEntries":[
        {
          "avatar":null,
          "name":"Andrey Gerasimenko",
          "company":null,
          "company_position":null,
          "office":null,
          "department":null,
          "server_name":"cts1dev.ccsteam.ru (Имя сервера)",
          "contacts":[
            {
              "active":true,
              "contact":"brave@test.ccsteam.ru",
              "contactType":"ad_login",
              "ctsId":"a619fcfa-a19b-5256-a592-9b0e75ca0896",
              "userHuid":"b4d24e05-c65a-5846-970b-3bb8893c1082",
              "userKind":"cts_user",
              "isBot":false
            }
          ]
        }
      ]
    },
    "trustSearchEntries":[
      {
        "avatar":null,
        "name":"Andrey Gerasimenko",
        "company":null,
        "companyPosition":null,
        "office":null,
        "department":null,
        "serverName":"cts2dev.ccsteam.ru (Aaxaxax)",
        "contacts":[
          {
            "active":true,
            "contact":"brave@test.ccsteam.ru",
            "contactType":"ad_login",
            "ctsId":"4b97e07d-c295-5779-b3a6-02abaa179636",
            "userHuid":"999f4590-fa4e-5b65-914b-c6c3dfcd7f0c",
            "userKind":"cts_user",
            "isBot":false
          }
        ]
      }
    ]
  }
}
```


Открытие чата

```
await openGroupChat({ groupChatId }: { groupChatId: string })
```

Метод отправляет клиенту ивент типа:

```
{
    "ref": "fea1b2fb-2b01-4edf-b778-fd543f81dfae",
    "type": "smartapp_rpc",
    "handler": "express",
    "payload": {
        "groupChatId": "79108386-f4c5-09a5-1a01-b05ec8bfa1fd"
    },
    "method": "open_group_chat"
}
```

Ответа от клиента не приходит. Происходит открытие чата (параметр groupChatId имеют и персональные чаты, а так же, чаты с ботами)

Отправка скрытой команды боту

Команда позволяет только передать боту информацию. Обработка информации должна быть реализована на стороне бота.

```
await sendBotCommand({
    userHuid,
    body,
    data
  }: {
    userHuid: string
    body: string
    data: { command: string } | null
  }
)
```

Метод отправляет клиенту ивент типа:

```
{
    "ref": "22c6144a-d500-4523-baaf-1f6b02fa35cb",
    "type": "smartapp_rpc",
    "handler": "express",
    "payload": {
        "userHuid": "0754b198-1a97-55d3-a04b-d0a1e2e44458",
        "message": {
            "body": "hello",
            "data": {
                "command": "/test"
            }
        }
    },
    "method": "send_bot_command"
}
```

В чат с ботом придет значение параметра `{ "body": "hello" }`, бот получит объект `{ "command": "/test" }`

Подписка на ивенты от клиента или бота (на примере redux-saga)

```
export function subscribeClientEvents(): EventChannel<AppEvent> {
  return eventChannel(emit => {
    // подписка на ивенты от клиента и бота
    bridge?.onReceive((event) => emit(event as any))
    return () => {
    }
  })
}

export function* watchClientEvents() {
  const channel: EventChannel<AppEvent> = yield call(subscribeClientEvents)

  while (true) {
    const event: AppEvent = yield take(channel)

    switch (event.type) {
      case 'back_pressed':
        yield call(handleClientBackPressedEvent) // ивент придет при свайпе назад на мобильных устройствах
        break
      case 'clean_cache':
        yield call(handleCleanCache) // ивент придет при нажатии кнопки Clear cache в меню с тремя точками
        break
      default:
        break
    }
  }
}
```

```
export function locationChangeSaga(
  action: { type: typeof '@@router/LOCATION_CHANGE', payload: string }
) {
  const isRoot = action.payload === '/'

  bridge?.sendClientEvent({
    method: 'routing_changed',
    params: {
      location: isRoot ? 'root' : 'nested',
    }
  })
}

export function handleClientBackPressedEvent() {
  history.back()
}

export function* handleCleanCache() {
  const registrations: ServiceWorkerRegistration[] = yield navigator.serviceWorker.getRegistrations()
  const unregisterPromises = registrations.map(registration => registration.unregister())

  const allCaches: string[] = yield caches.keys()
  const cacheDeletionPromises = allCaches.map(cache => caches.delete(cache))

  yield Promise.all([...unregisterPromises, ...cacheDeletionPromises])

  localStorage.clear()
}

export function* rootRouterSaga() {
  yield all([
    takeEvery('@@router/LOCATION_CHANGE', locationChangeSaga), // we use '@@router/LOCATION_CHANGE' event from connected-react-route
  ])
}

// include rootRouterSaga in your root saga
```

Кеширование статики с помощью WorkboxWebpackPlugin

Если приложение было создано с помощью `create-react-app`, добавляем строчку в `package.json`:

```
"scripts": {
"eject": "react-scripts eject",
}
```

В зависимости приложения добавляем `smartapp-sdk` версии `1.1.6` или выше:

```
"dependencies": {
"@unlimited/smartapp-sdk": "^1.1.6",
}
```

Устанавливаем пакет и выполняем команду `npm run eject`.

Далее, делаем следующие изменения в файлах:

Добавляем код в `index.tsx`:

```
if (module.hot) module.hot.accept()

if ("serviceWorker" in navigator) {
window.addEventListener("load", () => {
navigator.serviceWorker.register("./sw.js")
})
}
```

Добавляем код в файл `webpack.config.js`:

```
plugins: [
new WorkboxWebpackPlugin.InjectManifest({
swSrc: "@unlimited/smartapp-sdk/workers/workbox.js", // path to worker
swDest: "sw.js"
}),
```

Удаляем в файле `webpack.config.js` следующий код:

```
// Generate a service worker script that will precache, and keep up to date,
// the HTML & assets that are part of the webpack build.
isEnvProduction &&
fs.existsSync(swSrc) &&
new WorkboxWebpackPlugin.InjectManifest({
swSrc,
dontCacheBustURLsMatching: /\.[0-9a-f]{8}\./,
exclude: [/\.map$/, /asset-manifest\.json$/, /LICENSE/],
// Bump up the default maximum size (2mb) that"s precached,
// to make lazy-loading failure scenarios less likely.
// See <https://github.com/cra-template/pwa/issues/13#issuecomment-722667270>
maximumFileSizeToCacheInBytes: 5 _ 1024 _ 1024,
}),
```

Запускаем приложение, проверяем регистрацию сервис-воркера.