راهنمای افزودن پروژه وب به پروژه Appsan

- راهنمای افزودن پروژه وب به پروژه Appsan
- راهنمای تنظیم پروژه Angular برای استفاده در Appsan
- راهنمای تنظیم پروژه React برای استفاده در Appsan
- استفاده از کتابخانه JavaScript اپسان (AppsanWeb) برای توسعه‌دهندگان
- تنظیم امکان بازگشت و رفتار دگمه بازگشت در اپسان
- AppsanHTTPClient
- اتصال سایت به اپسان با ورود از طریق توکن موقت

راهنمای افزودن پروژه وب به پروژه Appsan

در این راهنما، نحوه افزودن یک پروژه وب به پروژه Appsan به صورت گام‌به‌گام توضیح داده می‌شود. با استفاده از این راهنما، شما می‌توانید پروژه‌های وب خود را به‌طور مؤثری درون Appsan ادغام کرده و از قابلیت‌های جدید عنصر `web-page` بهره‌مند شوید.

۱. ایجاد پوشه‌های مورد نیاز

1. **ایجاد پوشه `web`**:
   - ابتدا در مسیر اصلی پروژه Appsan خود، یک پوشه به نام `web` ایجاد کنید.
   - این پوشه قرار است حاوی کد منبع پروژه‌های وب شما باشد.
2. **ایجاد پوشه `webdist`**:
   - در کنار پوشه `web`، یک پوشه دیگر به نام `webdist` ایجاد کنید.
   - این پوشه قرار است فایل‌های ساخته شده (build) پروژه‌های وب شما را نگهداری کند.

۲. اضافه کردن پروژه وب به پوشه `web`

1. **ایجاد پوشه برای پروژه وب**:
   - درون پوشه `web`، یک پوشه جدید به نام پروژه وب خود ایجاد کنید. به عنوان مثال، اگر پروژه وب شما "MyWebProject" نام دارد، یک پوشه به نام `MyWebProject` بسازید.
2. **افزودن کد منبع**:
   - تمام فایل‌های مرتبط با پروژه وب خود را درون این پوشه کپی کنید. این شامل HTML، CSS، JavaScript و هر فایل دیگری است که برای پروژه وب شما نیاز است.

۳. ساخت پروژه وب

1. **ساخت پروژه**:
   - با استفاده از ابزار ساخت مناسب (مانند Webpack، Parcel، یا ابزارهای دیگر)، پروژه وب خود را بسازید.
   - نتیجه ساخت (فایل‌های build شده) باید شامل فایل‌های HTML، CSS، و JavaScript بهینه شده باشد.
2. **کپی فایل‌های ساخته شده به `webdist`**:
   - پس از ساخت پروژه، تمامی فایل‌های ساخته شده را درون پوشه‌ای با همان نام پروژه در `webdist` کپی کنید.
   - برای مثال، اگر پروژه شما "MyWebProject" نام دارد، فایل‌های ساخته شده باید در مسیر `webdist/MyWebProject` قرار گیرند.

۴. استفاده از عنصر `webpage` در پروژه Appsan

1. **افزودن عنصر `webpage` به فایل XML**:
   - حال می‌توانید از عنصر `webpage` در فایل XML پروژه Appsan خود استفاده کنید.
   - ویژگی `src` را به نام پوشه پروژه در `webdist` تنظیم کنید و ویژگی `title` را نیز به دلخواه مقداردهی کنید.

   مثال:

   ```
   <webpage src="MyWebProject" title="پروژه وب من"/>
   ```

2. **تست و اشکال‌زدایی**:
   - پروژه Appsan خود را اجرا کرده و بررسی کنید که صفحه وب به درستی بارگذاری می‌شود.
   - در صورت وجود هر گونه اشکال، به مسیرهای پروژه در پوشه‌های `web` و `webdist` و همچنین تنظیمات ساخت پروژه وب خود مراجعه کنید.

نکات مهم

- اطمینان حاصل کنید که نام پوشه‌ها در `web` و `webdist` کاملاً یکسان باشد.
- برای لینک‌های خارجی از ویژگی `src` با یک لینک HTTPS معتبر استفاده کنید.
- در صورت تغییر در کد منبع پروژه وب، پروژه را مجدداً ساخته و فایل‌های ساخته شده را به پوشه `webdist` کپی کنید.

این راهنما به شما کمک می‌کند تا پروژه‌های وب خود را به‌طور مؤثر در پروژه‌های Appsan ادغام کنید و از قابلیت‌های جدید آن بهره‌مند شوید.

راهنمای تنظیم پروژه Angular برای استفاده در Appsan

راهنمای تنظیم پروژه Angular برای استفاده در Appsan

در این راهنما، نحوه تنظیم پروژه Angular به گونه‌ای که فایل‌های ساخته شده (build) آن به پوشه `webdist` در پروژه Appsan منتقل شوند و همچنین پروژه Angular در پوشه `web` مدیریت شود، توضیح داده می‌شود. این تنظیمات به شما امکان می‌دهند که به راحتی از پروژه Angular خود در Appsan استفاده کنید.

۱. ایجاد پوشه‌های مورد نیاز

1. **ایجاد پوشه `web`**:
   - ابتدا در مسیر اصلی پروژه Appsan خود، یک پوشه به نام `web` ایجاد کنید.
   - این پوشه قرار است حاوی کد منبع پروژه‌های وب شما باشد.
2. **ایجاد پوشه برای پروژه Angular**:
   - درون پوشه `web`، یک پوشه جدید به نام پروژه Angular خود ایجاد کنید. به عنوان مثال، اگر پروژه Angular شما "MyAngularProject" نام دارد، یک پوشه به نام `MyAngularProject` بسازید.

۲. ایجاد پروژه Angular در پوشه `web`

1. **ایجاد پروژه Angular در مسیر مناسب**:
   - به پوشه‌ای که در مرحله قبل در `web` ایجاد کردید بروید (`web/MyAngularProject`) و پروژه Angular جدید خود را در این مسیر ایجاد کنید.
   - برای ایجاد پروژه در مسیر خاص، می‌توانید از دستور زیر استفاده کنید:

```
ng new MyAngularProject --directory web/MyAngularProject
```

2. **تست پروژه**:
   - قبل از انجام تغییرات، پروژه را به صورت محلی با استفاده از `ng serve` تست کنید تا از صحت عملکرد آن مطمئن شوید.

۳. تنظیم مسیر خروجی (outputPath) در `angular.json`

1. **باز کردن فایل `angular.json`**:
   - در ریشه پروژه Angular خود، فایل `angular.json` را باز کنید.
2. **یافتن تنظیمات `outputPath`**:
   - در بخش `architect` و زیر بخش `build`، تنظیماتی به نام `outputPath` را پیدا کنید. این بخش مشخص می‌کند که فایل‌های ساخته شده پروژه در کجا ذخیره شوند.
3. **تغییر مسیر خروجی**:
   - مقدار `outputPath` را به `../../webdist/[web-folder-name]` تغییر دهید. به جای `[web-folder-name]`، نام پوشه‌ای که برای پروژه وب خود در `webdist` استفاده خواهید کرد را قرار دهید.
   - به عنوان مثال، اگر نام پروژه شما "MyAngularProject" است، تنظیمات باید به شکل زیر باشد:

```
"outputPath": "../../webdist/MyAngularProject"
```

۴. تنظیم `baseHref` برای پروژه Angular

برای اطمینان از اینکه منابع پروژه به درستی بارگذاری می‌شوند، باید `baseHref` را به `./` تغییر دهید. این کار را می‌توانید به دو روش انجام دهید:

1. **تنظیم `baseHref` از طریق خط فرمان**:
   - هنگام ساخت پروژه، می‌توانید از دستور زیر استفاده کنید:

```
ng build --base-href ./
```

2. **تنظیم `baseHref` در `angular.json`**:
   - به جای استفاده از خط فرمان، می‌توانید `baseHref` را به طور دائم در فایل `angular.json` تنظیم کنید.
   - به بخش `architect` و زیر بخش `build` بروید و مقدار `baseHref` را به `./` تغییر دهید:

```
"build": {
  "options": {
    "outputPath": "../../webdist/MyAngularProject",
    "baseHref": "./"
  }
}
```

۵. ساخت پروژه Angular

1. **ساخت پروژه**:
   - پس از اعمال تغییرات در `angular.json` و تنظیم `baseHref`، با استفاده از دستور `ng build` پروژه خود را بسازید.

- فایل‌های ساخته شده باید در پوشه‌ای با نام پروژه در `webdist` ذخیره شوند.

2. **بررسی فایل‌های ساخته شده**:
   - به پوشه `webdist` در پروژه Appsan خود بروید و بررسی کنید که فایل‌های ساخته شده به درستی در مسیر `webdist/[web-folder-name]` قرار گرفته باشند.

۶. استفاده از پروژه Angular در Appsan

1. **افزودن عنصر `webpage` در فایل XML**:
   - از عنصر `webpage` در پروژه Appsan خود استفاده کنید و ویژگی `src` را به نام پوشه‌ای که پروژه Angular در `webdist` ذخیره شده است، تنظیم کنید.

   مثال:

   ```
   <webpage src="MyAngularProject" title="پروژه Angular من"/>
   ```

2. **اجرای پروژه Appsan**:
   - پروژه Appsan خود را اجرا کنید و مطمئن شوید که پروژه Angular به درستی بارگذاری می‌شود.

نکات مهم

- اطمینان حاصل کنید که پروژه Angular خود را در مسیر `web/[project-name]` ایجاد کرده‌اید.
- مسیر `outputPath` در `angular.json` باید به درستی به پوشه مناسب در `webdist` اشاره کند.
- حتماً `baseHref` را به `./` تنظیم کنید تا منابع به درستی در پروژه Appsan بارگذاری شوند.
- هر بار که تغییراتی در پروژه Angular خود ایجاد کردید، پروژه را مجدداً بسازید تا فایل‌های به‌روز شده در `webdist` قرار گیرند.
- نام پوشه در `webdist` باید با مقداری که در ویژگی `src` عنصر `webpage` استفاده می‌کنید، مطابقت داشته باشد.

این راهنما به شما کمک می‌کند تا پروژه‌های Angular خود را به‌طور کارآمد در پروژه‌های Appsan ادغام کنید و از قابلیت‌های جدید این چارچوب به‌رهمند شوید.

راهنمای تنظیم پروژه React برای استفاده در Appsan

در این راهنما، نحوه تنظیم یک پروژه React به‌گونه‌ای که فایل‌های build آن به پوشه `webdist` در پروژه Appsan منتقل شوند و همچنین پروژه در پوشه `web` مدیریت شود، آموزش داده می‌شود. این مراحل مشابه با تنظیمات پروژه Angular بوده ولی با در نظر گرفتن تفاوت‌های React و Vite تهیه شده است.

---

۱. ایجاد پوشه‌های مورد نیاز

ایجاد پوشه `web`:

در مسیر اصلی پروژه Appsan خود، یک پوشه به نام `web` ایجاد کنید. این پوشه محل قرارگیری پروژه‌های وب شماست.

ایجاد پوشه برای پروژه React:

در داخل پوشه `web`، یک پوشه به نام دلخواه برای پروژه React ایجاد کنید. مثال:

```
web/MyReactProject
```

---

۲. ایجاد پروژه React در مسیر منا‌سب

با استفاده از Vite و دستور زیر پروژه React را در مسیر دلخواه ایجاد کنید:

```
npm create vite@latest web/MyReactProject -- --template react
```

سپس وارد پوشه پروژه شوید و پکیج‌ها را نصب کنید:

```
cd web/MyReactProject
npm install
```

---

۳. تنظیم مسیر خروجی (build output)

برای ذخیره فایل‌های build در پوشه `webdist`، فایل `vite.config.js` را باز کرده و تنظیمات زیر را اضافه کنید:

```
export default defineConfig({
  base: './',
  build: {
    outDir: '../../webdist/MyReactProject',
    emptyOutDir: true,
  },
});
```

توجه: به جای `MyReactProject` در مسیر `outDir`، نام پوشه‌ای که می‌خواهید در `webdist` بسازید را قرار دهید.

---

۴. تنظیم مسیر (Routing) برای استفاده از مسیرهای دایناموی در Appsan

در صورتی که اپلیکیشن شما از مسیر دایناموی (مانند `domain.com/x/y`) استفاده می‌کند، باید از `HashRouter` به جای `BrowserRouter` استفاده شود.

مراحل:

۱. نصب React Router:

```
npm install react-router-dom
```

۲. در فایل `App.jsx` به‌صورت زیر تغییر دهید:

```
import { HashRouter, Routes, Route } from 'react-router-dom';

function App() {
  return (
    <HashRouter>
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/results" element={<Results />} />
        {/* مسیرهای دیگر */}
      </Routes>
    </HashRouter>
  );
}
```

با این کار، آدرس‌ها در مرورگر به صورت `/#/results` نمایش داده می‌شوند که باعث می‌شود پروژه در Appsan حتی با مسیرهای داینامیک نیز درست اجرا شود.

---

۵. ساخت پروژه

پس از اعمال تغییرات بالا، دستور زیر را برای ساخت پروژه اجرا کنید:

```
npm run build
```

با اجرای این دستور، فایل‌های نهایی در مسیر `webdist/MyReactProject` قرار می‌گیرند.

---

۶. استفاده از پروژه React در Appsan

در فایل XML مربوط به Appsan، عنصر `<webpage>` را به شکل زیر اضافه کنید:

```
<webpage src="MyReactProject" title="پروژه React من"/>
```

مطمئن شوید که `src` با نام پوشه‌ای که در `webdist` ساخته‌شده مطابقت داشته باشد.

نکات مهم

- پروژه React باید در مسیر `web/[project-name]` ایجاد شود.
- مسیر خروجی (`outDir`) در `vite.config.js` باید به `../../webdist/[project-name]` تنظیم شود.
- مقدار `base` باید به `'./'` تغییر یابد.
- برای جلوگیری از مشکلات در مسیرهای دینامیک، از `HashRouter` استفاده کنید.
- هر بار که تغییراتی در پروژه ایجاد می‌کنید، مجدداً دستور `npm run build` را اجرا نمایید.
- در فایل XML Appsan مسیر `src` باید دقیقاً با نام خروجی در `webdist` یکسان باشد.

استفاده از کتابخانه JavaScript اپسان (AppsanWeb) برای توسعه‌دهندگان

کتابخانه JavaScript اپسان وب با هدف تسهیل ارتباط بین صفحات وب و پلتفرم اپسان طراحی شده است. این مستند به شما کمک می‌کند تا از این کتابخانه در وب‌سایت‌های خود استفاده کنید.

نصب کتابخانه

برای نصب کتابخانه اپسان، کافی است از **npm** استفاده کنید:

```
npm i @appsan-web/mini-web-sdk
```

[لینک کتابخانه]

راه‌اندازی کتابخانه

در ابتدا نیاز است که کتابخانه را در پروژه خود وارد کنید:

```
import {AppsanWeb} from '@appsan-web/mini-web-sdk';
```

استفاده از کلاس `AppsanWeb`

کلاس اصلی برای تعامل با اپسان، `AppsanWeb` است. این کلاس مسئول مدیریت ارتباطات بین وب‌سایت شما و صفحه اپسان است.

1. **راه‌اندازی اولیه**:

برای راه‌اندازی کتابخانه کافی است تابع `initialize` را فراخوانی کنید. این تابع به صورت خودکار کتابخانه را بعد از بارگذاری کامل صفحه وب شما فعال می‌کند.

```
AppsanWeb.initialize();
```

2. وضعیت آماده بودن اپسان:

می‌توانید با استفاده از `getStatus()` وضعیت آماده بودن اپسان را بررسی کنید. این تابع یک `BehaviorSubject` از نوع استرینگ باز می‌گرداند که می‌توانید به تغییرات آن گوش دهید:

```
const status$ = AppsanWeb.getStatus();
status$.subscribe(status => {
  console.log("Appsan Status:", status);
});
```

:وضعیت می‌تواند یکی از مقادیر زیر باشد

```
"not ready"
"ready"
"not available"
"error"
```

3. اجرای اکشن در اپسان:

تابع runAction برای اجرای اکشن در اپسان استفاده می شود.

```
AppsanWeb.runAction('myAction').subscribe();
```

4. مدیریت دکمه بازگشت:

برای کنترل رفتار دکمه بازگشت از متد `setCustomBackHandler` استفاده کنید در حالت پیشفرض اپسان هیستوری مینی وب شما را مدیریت میکند و این تابع تنها برای مدیریت پیشرفته ی بازگشت استفاده می شود.

```
AppsanWeb.setCustomBackHandler(() => {
  // Custom back behavior
});
```

5. ارسال درخواست HTTP:

برای ارسال درخواست HTTP می‌توانید از متد `httpCall` استفاده کنید. به این صورت:

```
AppsanWeb.httpCall('https://api.example.com', 'GET', {}, {}).subscribe(response => {
  console.log('HTTP Response:', response);
});
```

6. دریافت اطلاعات بایند (Bind)

متد `getBind` برای دریافت اطلاعات یک بایند (Bind) خاص از اپسان استفاده می‌شود. این متد یک `Subject` برمی‌گرداند که می‌توانید به آن subscribe کنید تا اطلاعات بایند را دریافت کنید:

```
AppsanWeb.getBind('myBind').subscribe(data => {
  console.log('Bind data:', data);
});
```

7. تنظیم مقدار متغیر

با استفاده از متد `setVariable` می‌توانید مقدار یک متغیر را در اپسان تنظیم کنید.

```
AppsanWeb.setVariable('variableId', 'newValue').subscribe();
```

8. دریافت مقدار متغیر

برای دریافت مقدار یک متغیر از اپسان از متد `getVariable` استفاده می‌کنید. این متد نیز یک `Subject` برمی‌گرداند که می‌توانید مقدار متغیر را با آن دریافت کنید:

```
AppsanWeb.getVariable('variableId').subscribe(variable => {
  console.log('Variable:', variable);
});
```

تنظیم امکان بازگشت و رفتار دگمه بازگشت در اپسان

تنظیم امکان بازگشت و رفتار دگمه بازگشت در اپسان

مقدمه

در این راهنما، به نحوه‌ی مدیریت رفتار دکمه بازگشت در اپسان وب پرداخته می‌شود. اپسان وب به توسعه‌دهندگان اجازه می‌دهد تا امکان بازگشت و عملکرد دکمه بازگشت را سفارشی کنند و یا از رفتار پیش‌فرض آن استفاده کنند.

استفاده از رفتار پیش‌فرض

رفتار پیش‌فرض اپسان بر اساس **تعداد صفحات موجود در تاریخچه** عمل می‌کند:

1. **بسته شدن اپلیکیشن**: اگر تعداد صفحات موجود در تاریخچه برابر یا کمتر از یک باشد، اپلیکیشن بسته می‌شود.
2. **بازگشت به صفحه قبلی**: اگر تاریخچه بیشتر از یک صفحه داشته باشد، به صفحه قبلی بازمی‌گردد. در صورت تعریف customBackHandler این تابع بجای رفتار پیش‌فرض بازگشت به صفحه ی پیشین اجرا می‌شود.

پاک کردن تاریخچه

برای پاک کردن تاریخچه و تنظیم تعداد صفحات موجود، می‌توانید از تابع پیش‌فرض `clearBackStack` استفاده کنید:

1. این تابع تمام صفحات را حذف کرده و تاریخچه را به حالت اولیه تنظیم می‌کند.
2. برای استفاده از این تابع، می‌توانید آن را در هر تابعی که نیاز دارید فراخوانی کنید.

```
AppsanWeb.getAvailableHistoryCountProvider()?.clearBackStack();
```

قابلیت customBackHandler در AppsanWeb

`AppsanWeb` شامل یک متغیر و متد برای مدیریت بازگشت سفارشی است:

1. **تعریف customBackHandler**: این متغیر می‌تواند یک تابع تعریف شده توسط شما را ذخیره کند که هنگام فشار دادن دکمه بازگشت، اجرا خواهد شد.
2. **متد setCustomBackHandler**: برای تنظیم یک تابع سفارشی استفاده می‌شود:

```
AppsanWeb.setCustomBackHandler(() => {
  // handle back here your way
});
```

در صورت استفاده از تابع شخصی سازی شده عملکرد بازگشت پیش فرض اجرا نمی شود و بازگشت می‌توند با استفاده از اکشن app/forceback انجام شود. دقت کنید تابع customBackHandler تنها در صورتی که مینی اپ امکان بازگشت داشته باشد صدا می شود. در صورتی که مقدار getAvailableBackHistoryCount() یک یا کم تر باشد مینی وب بدون اجرای تابع customBackHandler بسته می شود. توضیحات تکمیلی تابع getAvailableBackHistoryCount در بخش بعدی آمده‌است.

سفارشی‌سازی امکان بازگشت

اگر نیاز دارید که رفتار خاصی برای امکان بازگشت تعریف کنید، می‌توانید از طریق **پیاده‌سازی یک کلاس جدید** که از رابط `AvailableBackCountProvider` ارث‌بری می‌کند، اقدام کنید. این رابط شامل دو متد زیر است:

1. `getAvailableHistoryCount()`: برای دریافت تعداد صفحات موجود در تاریخچه.
2. `clearBackStack()`: برای پاک کردن تاریخچه و بازنشانی به مقدار اولیه.

نمونه کد برای غیر فعال کردن بازگشت در مینی وب و بسته شدن مینی اپ در صورت بازگشت.

برای غیرفعال کردن دکمه بازگشت و بستن مینی وب، میتوانید از یک **Custom BackCountProvider** استفاده کنید که بهطور مستقیم تعداد تاریخچه را به 1 تنظیم کرده و سپس اپلیکیشن را میبندد.

مراحل پیادهسازی

1. یک کلاس سفارشی ایجاد کنید که از رابط `AvailableBackCountProvider` ارثبری کند.
2. متد `getAvailableHistoryCount()` را به گونهای پیادهسازی کنید که همیشه مقدار `1` برگرداند.

کد نمونه

```
import { AppsanWeb, AvailableBackCountProvider } from "./AppsanWeb";

export class DisableBackAndCloseProvider implements AvailableBackCountProvider {
    private parent: any;

    constructor() {
        this.parent = window.parent;
        const glob = globalThis as any;
        if (glob['Appsan']) {
            this.parent = glob['Appsan'] as any;
        }
    }

    getAvailableHistoryCount(): number {
        return 1; // همیشه مقدار 1 برگرداندن میتواند تاریخچه بازگشت غیرفعال باشد
    }

    clearBackStack(): void {

    }
}
```

توضیحات

- **غیرفعال کردن بازگشت**: با برگرداندن مقدار `1` در متد `getAvailableHistoryCount()`، اپسان وب تصور میکند که تاریخچهای برای بازگشت وجود ندارد و با کلیک شدن دکمه بک مینی وب را میبندد..

نمونه پیاده‌سازی یک ارائه‌دهنده سفارشی بازگشت

در این مثال، از `customBackHandler` برای نمایش یک دیالوگ تأیید قبل از بازگشت استفاده می‌کنیم. پیاده‌سازی این کلاس به گونه‌ای است که امکان بازگشت را کاملاً سفارشی کرده و تعامل با تاریخچه صفحات را کنترل می‌کند.

```
import { AppsanWeb, AvailableBackCountProvider } from "./AppsanWeb";

export class ConfirmableBackCountProvider implements AvailableBackCountProvider {
    private availableHistoryCount = 1;
    private parent: any;

    constructor() {
        this.parent = window.parent;
        const glob = globalThis as any;
        if (glob['Appsan']) {
            this.parent = glob['Appsan'] as any;
        }

        const self = this;

        // شنود برای تغییرات تاریخچه
        (function (history: any) {
            const pushState = history.pushState;
            history.pushState = function (state: any) {
                console.log('Push state:', state);
                self.onPushState(state);
                return pushState.apply(history, arguments);
            };
        })(window.history);

        window.addEventListener("popstate", (event) => {
            self.onPopState(event.state);
        });

        // تنظیم customBackHandler
        AppsanWeb.setCustomBackHandler(() => {
            self.showConfirmationDialog();
```

```
        });
    }

    onPopState(state: any): void {
        this.availableHistoryCount--;
        (AppsanWeb as any).instance.post((AppsanWeb as
any).MESSAGE_TYPE_AVAILABLE_HISTORY_COUNT, this.availableHistoryCount);
    }

    onPushState(state: any): void {
        this.availableHistoryCount++;
        (AppsanWeb as any).instance.post((AppsanWeb as
any).MESSAGE_TYPE_AVAILABLE_HISTORY_COUNT, this.availableHistoryCount);
    }

    getAvailableHistoryCount(): number {
        return this.availableHistoryCount;
    }

    clearBackStack(): void {
        this.availableHistoryCount = 1;
        (AppsanWeb as any).instance.post((AppsanWeb as
any).MESSAGE_TYPE_AVAILABLE_HISTORY_COUNT, this.availableHistoryCount);
    }

    private showConfirmationDialog(): void {
        const confirmed = confirm("آیا مطمئن هستید که می‌خواهید به صفحه قبلی بازگردید؟");
        if (confirmed) {
            history.back();
        } else {
            console.log("کاربر بازگشت را لغو کرد.");
        }
    }
}
```

نحوه استفاده از کلاس سفارشی

برای استفاده از این کلاس در اپلیکیشن خود، مراحل زیر را دنبال کنید:

1. **ایجاد نمونه از کلاس**:

```
const backHandler = new ConfirmableBackCountProvider();
```

2. **ثبت رفتار بازگشت سفارشی**: کلاس `ConfirmableBackCountProvider` به صورت خودکار از طریق `AppsanWeb.setCustomBackHandler` رفتار بازگشت را تنظیم می‌کند.
3. **مدیریت تاریخچه**: این کلاس با متدهای `onPushState` و `onPopState` تعداد صفحات موجود در تاریخچه را مدیریت می‌کند.

AppsanHTTPClient

کلاس `AppsanHTTPClient`

کلاس `AppsanHTTPClient` یک واسط ساده برای ارسال درخواست‌های HTTP در مینی‌وب‌ها (Mini-Webs) است که به سوپرایپ Appsan متصل هستند. این کلاس از قابلیت‌های داخلی Appsan (از جمله احراز هویت، تنظیمات پیش‌فرض، و ارسال درخواست از طریق `AppsanWeb`) استفاده می‌کند تا ارسال درخواست به API‌ها ساده و ایمن باشد.

وارد کردن کلاس

```
import { AppsanHTTPClient } from '@appsan-web/mini-web-sdk';
```

تنظیمات درخواست

برای تنظیم رفتار درخواست‌های HTTP می‌توان از گزینه‌های موجود در `HttpOptions` استفاده کرد. این تنظیمات را می‌توان به‌صورت پیش‌فرض برای تمامی درخواست‌ها یا به‌صورت جداگانه برای هر درخواست تعیین کرد.

تنظیمات Timeout:

- `connectTimeout`: حداکثر زمان مجاز برای برقراری اتصال با سرور. مقدار این گزینه بر حسب **میلی‌ثانیه (ms)** است.
- `readTimeout`: حداکثر زمان انتظار برای دریافت داده از سرور پس از برقراری اتصال. مقدار این گزینه بر حسب **میلی‌ثانیه (ms)** است.
- `writeTimeout`: حداکثر زمان مجاز برای ارسال داده درخواست به سرور. مقدار این گزینه بر حسب **میلی‌ثانیه (ms)** است.

برای مثال:

```
AppsanHTTPClient.setDefaultOptions({
  connectTimeout: 10000,
  readTimeout: 30000,
```

```
  writeTimeout: 30000
});
```

در مثال بالا، مقدار `connectTimeout` برابر ۱۰ ثانیه و مقادیر `readTimeout` و `writeTimeout` برابر ۳۰ ثانیه هستند.

متدها

`setDefaultOptions(options: HttpOptions): void`

تنظیم گزینه‌های پیش‌فرض برای تمامی درخواست‌ها.

ورودی‌ها:

- `options`: یک شیء از نوع `HttpOptions` که شامل هدرها، تنظیمات سفارشی و گزینه‌های Timeout است.

گزینه‌های Timeout قابل استفاده در `HttpOptions` عبارتند از:

- `connectTimeout`: زمان مجاز برای برقراری اتصال، بر حسب میلی‌ثانیه.
- `readTimeout`: زمان مجاز برای خواندن پاسخ، بر حسب میلی‌ثانیه.
- `writeTimeout`: زمان مجاز برای ارسال داده، بر حسب میلی‌ثانیه.

`get(url: string, options?: any): Observable<any>`

ارسال درخواست GET.

ورودی‌ها:

- `url`: آدرس API.
- `options` (اختیاری): تنظیمات سفارشی برای این درخواست خاص. این تنظیمات نیز باشند. می‌توانند شامل `connectTimeout`، `readTimeout` و `writeTimeout`

خروجی:
یک `Observable` که نتیجه پاسخ API را بازمی‌گرداند.

`post(url: string, body: any, options?: any): Observable<any>`

ارسال درخواست POST.

ورودی‌ها:

- `url`: آدرس API.
- `body`: داده‌ای که باید در بدنه درخواست ارسال شود.
- `options` (اختیاری): تنظیمات اضافی برای این درخواست، از جمله تنظیمات Timeout.

`put(url: string, body: any, options?: any): Observable<any>`

ارسال درخواست PUT برای به‌روزرسانی داده‌ها.

ورودی‌ها:

- `url`: آدرس API.
- `body`: داده‌ای که باید به‌روزرسانی شود.
- `options` (اختیاری): تنظیمات اضافی برای این درخواست، از جمله تنظیمات Timeout.

`delete(url: string, options?: any): Observable<any>`

ارسال درخواست DELETE برای حذف داده‌ها.

ورودی‌ها:

- `url`: آدرس API.
- `options` (اختیاری): تنظیمات اضافی برای این درخواست، از جمله تنظیمات Timeout.

`patch(url: string, body: any, options?: any): Observable<any>`

ارسال درخواست PATCH برای تغییر جزئی داده‌ها.

ورودی‌ها:

- `url`: آدرس API.
- `body`: داده‌ای که باید تغییری کند.
- `options` (اختیاری): تنظیمات اضافی برای این درخواست، از جمله تنظیمات Timeout.

مثال تنظیم Timeout برای یک درخواست

علاوه بر تنظیم مقادیر پیش‌فرض، می‌توان Timeoutها را برای هر درخواست به‌صورت جداگانه مشخص کرد.

```
AppsanHTTPClient.get('/api/example', {
  connectTimeout: 5000,
  readTimeout: 15000,
  writeTimeout: 15000
});
```

تمام مقادیر Timeout بر حسب **میلی‌ثانیه (ms)** هستند.

نکات مهم

- این کلاس از `AppsanWeb.httpCall` برای ارسال درخواست‌ها استفاده می‌کند، که توسط سوپرپاپ Appsan مدیریت می‌شود.
- داده‌های مربوط به احراز هویت، به صورت خودکار توسط سوپرپاپ اضافه می‌شوند.
- خروجی تمامی متدها از نوع `Observable` است و می‌توان با استفاده از RxJS آن‌ها را مدیریت کرد.
- مقادیر `connectTimeout`، `readTimeout` و `writeTimeout` همگی بر حسب **میلی‌ثانیه** هستند.
- Timeoutها را می‌توان با استفاده از `setDefaultOptions` برای تمامی درخواست‌ها تنظیم کرد یا در `options` هر متد، برای یک درخواست خاص مقدار متفاوتی در نظر گرفت.

اتصال سایت به اپسان با ورود از طریق توکن موقت

برای نمایش سایت موجود خود در اپسان به‌صورت مینی‌اپ، دو مسیر وجود دارد: استفاده از کتابخانه JavaScript اپسان و ارسال درخواست‌ها از طریق آن، یا ایجاد نشست سایت با یک توکن موقت. این راهنما مسیر دوم را توضیح می‌دهد؛ پس از ورود، سایت با نشست و APIهای معمول خودش کار می‌کند و برای همین جریان ورود نیازی به جایگزینی همهٔ فراخوانی‌های API با SDK ندارد.

راهنمای روش اول: نصب و راه‌اندازی AppsanWeb و ارسال درخواست‌ها با AppsanHTTPClient.

۱. جریان ورود و مسئولیت هر بخش

1. کاربر مینی‌اپ را در سوپراپ باز می‌کند.
2. صفحهٔ شروع مینی‌اپ، endpoint صدور توکن سایت را با HTTPClient فراخوانی می‌کند.
3. سرور سایت، درخواست و هویت کاربر را اعتبارسنجی کرده و یک توکن موقت به همراه آدرس صفحهٔ ورود سایت برمی‌گرداند.
4. مینی‌اپ توکن را در پارامتر `token` آدرس قرار می‌دهد و آن را در المان `webpage` باز می‌کند.
5. سرور سایت توکن را اعتبارسنجی و مصرف می‌کند، نشست کاربر را می‌سازد و کاربر را به آدرس بدون توکن هدایت می‌کند.

الزامات سرور در این راهنما قرارداد پیشنهادی برای پیاده‌سازی مشتری هستند و باید با تیم اپسان نهایی شوند.

۲. پیش‌نیازها و اطلاعات موردنیاز

- یک سایت و API قابل دسترسی با HTTPS معتبر و امکان ایجاد نشست در سمت سرور.
- آدرس endpoint صدور توکن، روش درخواست، ده‌های موردنیاز و شناسهٔ مینی‌اپ.
- روش مورد توافق برای دریافت و اعتبارسنجی هویت کاربر سوپراپ و نگاشت آن به حساب سایت. هدر `x-api-app` فقط شناسهٔ برنامه است و به‌تنهایی هویت کاربر را اثبات نمی‌کند.
- آدرس ثابت و مجاز صفحهٔ مصرف توکن، مسیر مقصد پس از ورود، زمان اعتبار توکن و سیاست دسترسی کاربران.

- یک پروژه مینی‌اپ با صفحات شروع، وب و خطا که در فایل اصلی پروژه include شده باشند.

۳. قرارداد endpoint صدور توکن

در مثال این راهنما، درخواست از نوع GET است. مسیر زیر فرضی است؛ آن را با مسیر واقعی جایگزین کنید:

```
GET https://api.example.com/api/v1/auth/sso/appsan
Accept: application/json
x-api-app: customer_miniapp
```

ممکن است endpoint به هدر `x-api-key` نیاز داشته باشد؛ نیاز به این هدر و نحوه تأمین آن به قرارداد اتصال وابسته است. کلید مشترک جایگزین احراز هویت کاربر نیست. کلید محرمانه دائمی را در کد فرانت‌ای بسته قابل دریافت مینی‌اپ قرار ندهید؛ در صورت نیاز، آن را از طریق مسیر سروری مورد توافق تأمین کنید. مثال پایین فرض می‌کند احراز هویت معتبر درخواست از قبل در مسیر اتصال فراهم شده است؛ این endpoint نباید برای درخواست ناشناس توکن کاربر صادر کند.

بدنه پاسخ موفق مورد انتظار:

```
{
  "token": "ONE_TIME_TEMP_TOKEN",
  "url": "https://site.example.com/auth/appsan"
}
```

در callback کتابخانه HTTPClient، این دو مقدار به شکل `res.data.token` و `res.data.url` خوانده می‌شوند؛ بنابراین پوشش اضافی `data` در JSON بالا لازم نیست. توکن باید به کاربر و برنامه معتبر متصل باشد. پاسخ توکن با `Cache-Control: no-store` ارسال شود و خطای احراز هویت، دسترسی ای صدور توکن با پاسخ خطای مناسب برگردد.

۴. پیاده‌سازی سمت مینی‌اپ

کد زیر را می‌توانید در فایل `layout/app.xml.appsan` قرار دهید، ای سه بخش را در فایل‌های جدا بنویسید و از فایل اصلی include کنید. دامنه‌های نمونه، شناسه برنامه و قرارداد احراز هویت درخواست را متناسب با اتصال واقعی تنظیم کنید. این مثال فقط آدرس دقیق `https://site.example.com/auth/appsan` را به‌عنوان مقصد مجاز قبول می‌کند؛ در هر دو کنترل آدرس، آن را جایگزین کنید.

```
<page id="start" title="ورود به سامانه" background="white"
      onshown="script/starter-script:init()" clearBackStack="true">
    <frame width="match_parent" gravity="center" layout="frame" margin_top="64">
        <progressbar></progressbar>
    </frame>
    <script id="starter-script">
        function init() {
            HTTPClient.get('https://api.example.com/api/v1/auth/sso/appsan', {
                headers: {
                    Accept: 'application/json',
                    "x-api-app": "customer_miniapp"
                }
            }).subscribe(res => {
                let token = res.data.token;
                let url = res.data.url;
                if (typeof token !== 'string' || !token ||
                    url !== 'https://site.example.com/auth/appsan') {
                    Appsan.runAction('page/error');
                    return;
                }
                Appsan.setVariable('qpToken', token);
                Appsan.setVariable('url', url);
                Appsan.runAction('page/web');
            }, err => {
                Appsan.runAction('page/error');
            });
        }
    </script>
</page>

<webpage id="web" title="سامانه من" clearBackStack="true"
         onshown="script/webscript:init()">
    <script id="webscript">
        function init() {
            let token = Appsan.getVariable('qpToken')?.value;
            let url = Appsan.getVariable('url')?.value;
            if (typeof token !== 'string' || !token ||
                url !== 'https://site.example.com/auth/appsan') {
                Appsan.runAction('page/error');
```

```
                return;
            }
            Appsan.setProperty('web', 'src',
                url + '?token=' + encodeURIComponent(token) +
                String.fromCharCode(38) + 'utm_source=superapp_app');
        }
    </script>
</webpage>

<page id="error" title="خطا در ورود" layout="linear_vertical"
      gravity="center" background="white" clearBackStack="true">
    <text textalignment="center">ورود به سایت انجام نشد.</text>
    <button onclick="page/start">تلاش مجدد</button>
</page>
```

در این مثال، URL برگشتی باید همان آدرس ثابت و بدون query string یا fragment باشد. اگر قرارداد شما URL دارای پارامتر می‌خواهد، پارامتر `token` را با حفظ پارامترهای موجود و پیش از fragment اضافه کنید؛ به آدرسی که از قبل `?` دارد، دوباره `?` اضافه نکنید. مقدار توکن حتماً URL-encode شود. پارامتر `utm_source=superapp_app` اختیاری و صرفاً برای تشخیص منبع ورود است.

کاربرد متدها: `Appsan.setVariable` توکن و آدرس را بین صفحات منتقل می‌کند؛ `Appsan.getVariable(...).value` آن‌ها را می‌خواند؛ `Appsan.runAction('page/web')` صفحه وب را باز می‌کند و `Appsan.setProperty('web', 'src', ...)` آدرس آن را تعیین می‌کند. نام `web` باید با id المان webpage یکسان باشد. دکمه تلاش مجدد به صفحه شروع برمی‌گردد تا توکن جدید دریافت شود.

۵. پیاده‌سازی مصرف توکن در سرور سایت

مسیر `/auth/appsan` یک مسیر سمت سرور سایت است؛ صرف خواندن token در JavaScript مرورگر، ورود معتبر ایجاد نمی‌کند. منطق پیشنهادی:

```
GET /auth/appsan?token=...
1. Read token; reject missing or malformed values.
2. Validate expiry, intended application and linked user.
3. Atomically consume token; reject replay or invalid tokens.
4. Check user status and permissions; create/rotate site session.
5. Set session cookie and redirect to a clean URL such as /dashboard.
```

توکن بهتر است کوتاه‌عمر و یک‌بارمصرف باشد؛ مثلاً ۶۰ ثانیه فقط یک مقدار پیشنهادی است و مقدار واقعی باید توافق شود. اگر توکن opaque است، سرور می‌تواند نگاشت امن توکن به کاربر و زمان انقضا را نگهداری کند؛ اگر امضاشده است، امضا و claims باید اعتبارسنجی شوند و یک‌بارمصرف بودن نیز در سرور اعمال شود. نام کاربر یا شناسه‌ای که فقط از query گرفته شده، مبنای ورود نباشد.

پس از اعتبارسنجی، از سازوکار نشست معمول سایت استفاده کنید و برای cookie تنظیمات Secure و HttpOnly را اعمال کنید. تنظیم SameSite باید با نحوه نمایش در سوپراپ سازگار باشد. در نمایش iframe بین دامنه‌های متفاوت، محدودیت کوکی‌های شخص ثالث را بررسی کنید؛ SameSite=None نیازمند Secure است و به‌تنهایی محدودیت همه مرورگرها را رفع نمی‌کند. سیاست‌های CSP/frame-ancestors و X-Frame-Options نیز باید نمایش سایت در میزبان مجاز سوپراپ را پشتیبانی کنند.

توکن موقت را در لاگ، ابزار تحلیل، پیام خطا یا ذخیره‌سازی دائمی مرورگر ثبت نکنید. صفحه مصرف توکن پیش از بارگذاری منابع ثالث، توکن را مصرف و به URL پاک هدایت کند؛ Referrer-Policy مناسب مانند no-referrer و حذف یا پوشاندن token در لاگ reverse proxy نیز در نظر گرفته شود. خطای توکن نامعتبر یا منقضی باید صفحه قابل فهم با امکان شروع دوباره ورود نشان دهد.

۶. بررسی اتصال و رفع اشکال

- **ورود موفق:** کاربر درست وارد شود، URL پس از ورود فاقد token باشد و APIهای سایت با نشست ساخته‌شده کار کنند.
- **توکن منقضی، مصرف‌شده یا دست‌کاری‌شده:** ورود در شود و دریافت توکن جدید امکان‌پذیر باشد؛ استفاده همزمان از یک توکن بیش از یک نشست ایجاد نکند.
- **خطای دریافت توکن:** درها، اعتبار هویت کاربر، دسترسی برنامه و ساختار پاسخ بررسی شوند؛ صفحه خطا و تلاش مجدد نمایش داده شود.
- **نمایش صفحه ورود سایت به‌جای ورود خودکار:** مسیر مصرف توکن، ثبت cookie، redirect، انقضای توکن و نگاشت کاربر بررسی شوند.
- **صفحه خالی یا بارگذاری ناموفق:** HTTPS، آدرس src، مجاز بودن دامنه و تنظیمات iframe/CSP بررسی شوند.
- **دو حساب متفاوت:** با دو کاربر مستقل آزمایش کنید تا نشست یا توکن یک کاربر به دیگری منتقل نشود؛ تغییر کاربر و خروج نیز بررسی شوند.
- **محیط‌ها:** جریان کامل را در کلاینت‌های هدف، از جمله وب و اندرویدی در صورت استفاده، آزمایش کنید؛ سیاست cookie و نمایش وب ممکن است متفاوت باشد.

برای تحویل اتصال، مشتری قرارداد endpoint و نحوه احراز هویت آن، URL مصرف توکن، سیاست انقضا و مصرف مجدد، و نتیجه آزمایش ورود را ارائه کند؛ تیم اپسان آدرس‌ها و شناسه برنامه را در مینی‌اپ تنظیم و اجرای جریان کامل را بررسی می‌کند.