Documentação Técnica — Calendário de Audiências

1. Integração com a extensão PJe+r

O calendário de audiências foi desenvolvido como uma funcionalidade nativa da extensão PJe+r, aproveitando toda a infraestrutura já existente dessa extensão. Essa decisão estratégica proporcionou dois grandes benefícios:

- Facilidade de implementação: a estrutura interna da PJe+r já possui mapeamentos de acesso e autenticação para os principais endpoints da API do PJe, o que reduz significativamente o esforço técnico necessário para integrar novos recursos, como o calendário.
- Maior aceitação pelo usuário final: ao ser incorporado a uma extensão já amplamente utilizada e de instalação simplificada, o calendário se beneficia da familiaridade e confiança dos usuários com a PJe+r, aumentando as chances de adesão e uso contínuo da nova funcionalidade.

Essa abordagem não apenas otimizou o processo de desenvolvimento, como também consolidou o calendário como uma extensão natural do ecossistema PJe+r.

2. Dificuldades que o ambiente de desenvolvimento da extensão PJe+r solucionam

Alguns dos desafios que a integração com o sistema PJe+r resolvem estão descritos a seguir, no trecho retirado da documentação da extensão.

Desafios das Extensões Cross-Browser

A criação de extensões _cross-browser_ envolve diversos desafios. Os principais deles, conforme descrito aqui, são diferenças quanto ao seguinte:

- *namespace* da API de cada navegador
- forma como a API de cada navegador lida com eventos assíncronos
- métodos implementados na API de cada navegador
- atributos do manifesto
- modo de empacotamento (*packaging*)
- modo de publicação

Além dessas diferenças, é necessário acrescentar, ainda, aquelas decorrentes da versão do manifesto.

O ambiente de desenvolvimento criado para o PJe+R tem por finalidade facilitar o trabalho dos desenvolvedores, contornando todos esses desafios de forma automática, de modo a evitar a necessidade de ajustes manuais ou de duplicação de código.

As principais ferramentas que nos permitem fazer isso são o NodeJS e o Typescript, porque com elas conseguimos separar o código em desenvolvimento do código de produção e aplicar, no processo de transpilação, quaisquer tipos de transformação que entendermos necessárias, quer por meio de scripts que nós mesmos elaboramos, quer por meio de ferramentas auxiliares e arquivos de configuração.

Além disso, o Typescript nos permite usar o padrão de módulos nativos do ES6, com import e export. Quando utilizamos esse padrão juntamente com um bundler (falamos do uso de bundlers mais à frente), conseguimos assegurar a reutilização do código de forma consistente e ao mesmo tempo atender às exigências dos navegadores de que determinadas partes da extensão estejam contidas em um só arquivo, em um determinado formato. Exemplo: nos códigos de conteúdo, em que não podemos utilizar os módulos do ES6, o bundler permite que todos os arquivos relacionados ao código de conteúdo sejam reunidos em um arquivo único e transformados automaticamente em uma IIFE.

A seguir, explicamos em maior detalhe como enfrentamos cada um dos desafios relacionados ao desenvolvimento de uma extensão *cross-browser*.

Diferenças no Namespace

No Firefox, a API do navegador é acessada pelo namespace browser, ao passo que no Chrome o namespace é chrome. Um namespace nada mais é do que o nome do objeto global pelo qual acessamos os métodos expostos pela API.

Como os content scripts fazem parte do mesmo contexto de execução das páginas acessadas pelo usuário no navegador, o namespace do navegador faz parte do objeto window. Assim, por exemplo, o namespace chrome equivale a window.chrome.

A mesma lógica se aplica às páginas popup e options, porque, embora não compartilhem do mesmo contexto de execução das páginas acessadas pelo usuários, elas têm seus próprios objetos window.

Há uma diferença importante, contudo, no que se refere ao código de fundo. Enquanto nas extensões baseadas na versão 2 do manifesto o background script recebe tratamento similar ao acima descrito, porque tem seu próprio objeto window, nas extensões baseadas na versão 3 do manifesto o tratamento é completamente diverso. O código de fundo passa a ser um service worker, sem acesso ao objeto window, e o seu contexto de execução passa a ter um objeto global do tipo ServiceWorkerGlobalScope. Além disso, o código de fundo torna-se efêmero, ou seja, não persiste na memória, sendo carregado a cada nova execução. Assim, o namespace não pode ser armazenado em uma variável global; precisa ser verificado a cada nova execução.

Para abstrair essas dificuldades e tornar mais simples e uniforme o acesso à API do Firefox e do Chrome, utilizamos uma técnica de "fachada", que consiste em verificar qual é o namespace presente em cada contexto de execução e entregá-lo sempre sob o mesmo nome (browser) ao código solicitante, independentemente do navegador em que a extensão esteja

sendo executada. Isso nos permite também, em certas condições, entregar, no lugar do objeto nativo, um polyfill, que torna uniformes outros aspectos das APIs (para mais detalhes sobre isso, veja abaixo o tópico relacionado aos eventos assíncronos).

Na prática, isso significa que, no lugar de acessar diretamente a variável global browser ou chrome, o desenvolvedor deve importar o objeto browser da pasta utils, conforme ilustrado no exemplo a seguir:

```
import   browser   from '~/utils' // '~/utils' equivale a
'src/utils'

browser runtime onInstalled addListener   : void =>

  // ...
```

Eventos Assíncronos

Nas extensões baseadas na versão 2 do manifesto, Chrome e Firefox divergem também quanto à forma de processar eventos assíncronos. A API do Chrome exige o uso de callbacks, ao passo que a API do Firefox utiliza-se de Promises.

Para tornar uniforme o uso de Promises, utilizamos o webextension-polyfill, um polyfill criado pela Mozilla para auxiliar no desenvolvimento de extensões cross-browser.

O polyfill é automaticamente acionado, quando necessário, sempre que utilizamos o padrão de importação descrito no item anterior.

A partir da versão 3 do manifesto, a API do Chrome também passará a adotar Promises, o que poderá tornar desnecessário o uso do polyfill.

Atributos do Manifesto

O manifesto pode ter atributos diferentes a depender do navegador. Além disso, os atributos sofreram mudanças consideráveis na passagem da versão 2 para a 3.

Atualmente, para desenvolver uma extensão cross-browser, não há como deixar de contemplar as duas versões do manfesto, pois, enquanto a Mozilla ainda não apresentou previsão de migrar o Firefox da versão 2 para a 3, a Google já definiu um cronograma, que você pode ver aqui:

- A partir de 17 de janeiro de 2022 extensões baseadas na versão 2 não são mais aceitas pela Chrome Web Store.
- A partir de janeiro de 2023 o navegador Chrome não executará mais extensões baseadas na versão 2.

Para lidar com as duas versões do manifesto e tornar mais fácil a migração, fizemos o seguinte:

1. No lugar de trabalhar diretamente em um arquivo manifest.json, optamos por gerar esse arquivo dinamicamente, a partir do arquivo manifest.ts, inserido na raiz da pasta src. A vantagem dessa abordagem é que ela permite também ajustar os atributos do manifesto conforme o navegador para o qual a extensão se destina. Assim, resolvemos de uma só vez o problema das versões e o problema da diferença de atributos entre os navegadores. Além disso, ao criar o arquivo manifesto a partir de um código em Typescript, podemos contar também com autocomplete e verificação de tipo.
2. Procuramos planejar a extensão tendo em vista todas as restrições da versão 3, sem contar com funcionalidades que existem somente na versão 2. Assim, por exemplo, o nosso código de fundo (background) já foi projetado para funcionar desde logo como um service worker, mesmo que seja executado como um script comum na versão 2 do manifesto.

Empacotamento (packaging) e Publicação

As diferenças quanto às formas de empacotamento e publicação se resolvem com a adoção de uma esteira de building/bundling, que conseguimos implementar graças ao ambiente em NodeJS. Para tanto, utilizamos os scripts que estão na pasta scripts e do Vite, uma ferramenta de desenvolvimento construída por cima do rollup.js e do esbuild.

A vantagem do Vite está em seu excelente servidor de desenvolvimento, que conta com um sistema de HMR (Hot Module Replacement) super rápido, baseado nos módulos ES6 nativos. Isso nos permite ver as alterações aparecerem, em tempo real, no navegador, enquanto escrevemos e alteramos o código.

O rollup.js e o esbuild são bundlers, que cuidam do processo de ler o código em Typescript, analisar todas as suas dependências e transformar esse código em Javascript puro, contido em um único arquivo. No processo de bundling, conseguimos introduzir alterações adicionais com o uso de plugins e/ou scripts.

3. Manutenção do projeto

3.1 Preparando do ambiente, obtendo o código e compilando

Para colaborar no projeto, sua estação de trabalho deve estar configurada com os seguintes aplicativos:

1. Navegadores de internet: Google Chrome, Mozilla Firefox, e, opcionalmente, Microsoft Edge.
2. `Git` - É um sistema de controle de versões que possue a função de registrar quaisquer alterações feitas em arquivos simplificando o processo de compartilhamento de um projeto com um time.
3. `NodeJS` - É um ambiente de execução Javascript *server-side*. [Todo: explicar uso no projeto]

4. pNpm - Gestor de pacotes rápidos
5. Visual Studio Code / VS Code (opcional, mas recomendado)

O ambiente deve funcionar igualmente bem em Windows ou Linux.

Para simplificar a configuração no ambiente Microsoft Windows 10 ou superior podemos seguir o roteiro a seguir:

1. Abrir um terminal do Powershell (Tecla do Windows + X e depois I)
2. Instalar Scoop digitando os seguintes comandos no terminal do Powershell:

   Set-ExecutionPolicy RemoteSigned -Scope CurrentUser
   irm get.scoop.sh | iex

   ---

3. Instalar os requisitos necessários (Git, VS Code, NodeJS na versão 18, pNpm)
   No windows você pode instalar pelo terminal usando os seguintes comandos:

   `scoop bucket add extras`
   `scoop install git vscode pnpm nodejs`

   ---

4. Para usuários de Linux, é preciso instalar manualmente o git-flow:

   ---

   sudo apt-get update
   sudo apt-get install git-flow

Caso seja a primeira vez que esteja usando o Git, defina o user.name e user.email primeiro. Veja como fazer isso aqui. Isso deve ser suficiente para você usar o Git pelo terminal. Se além dos comandos user.name e user.email, o Git solicitar a senha, o usuário deve seguir os passos abaixo:

1.0 Acesse a página de preferências do seu perfil do Git no link:
https://git.cnj.jus.br/-/profile/preferences
1.1 Ao entrar na opção acima, você poderá criar um novo token de acesso, clicando em "Access Tokens"

2.0 Ao entrar na opção 1.1 você irá criar um novo token de acesso
2.1.0 Preencher 'Token name'
2.1.1 Preencher 'Expiration date'. Caso este campo fique em branco o token não irá expirar.
2.1.2 Selecionar o scopo de acordo com sua necessidade
2.1.3 Após realizar os passos 2.1.0 até 2.1.2 o usuario deve clicar em "Create personal access token"

3.0 Copiar o código gerado e informar na linha de comando do git

Para obter o código, clone o repositório e faça a primeira compilação com os seguintes comandos:

```
git clone https://git.cnj.jus.br/git-jus/pjemaisr.git
cd pjemaisr
pnpm install

# Instalação automatizada
pnpm setup:extension
```

Para testar a instalação, rode o comando para levantar o ambiente de desenvolvimento:

```
pnpm play:chromium
```

Esse comando abre a página inicial do ambiente de desenvolvimentos com links (documentação, links, mocks…).

3.2 Manutenção na funcionalidade Calendário de Audiências

1. Vá até a branch **feature/calendario-de-audiencias-v2**
2. Navegue até o local da funcionalidade em /apps/extension/src/content/calendario-de-audiencias-v2

O diretório da funcionalidade tem a seguinte estrutura:

```
calendario-de-audiencias
├ components
|  └ CalendarioDeAudiencias.vue
├ services
|  └ dados-audiencias.ts
|  └ dados-cargo-judicial.ts
├ utils
|  └ injetar-style-no-shadow.ts
|  └ obter-lotacao.ts
├ index.ts
├ principal.ts
├ README.md
└ tipos.ts
```

1. **README.md**: arquivo contendo instruções e detalhamentos da funcionalidade.
2. **index.ts**: Este arquivo é o ponto de entrada da funcionalidade. Nele é instanciado e instalado a funcionalidade passando como parâmetros as opções de instalação e de ativação a partir de um instalador geral para todas a funcionalidades.
3. **principal.ts**: esse arquivo contém, por padrão, o código principal da funcionalidade, porém, como o calendário de audiências é montado por meio de um componente vue, o código principal se encontra dentro do próprio componente CalendarioDeAudiencias.vue.
4. **tipos.ts**: esse arquivo contém as interfaces utilizadas pela funcionalidade.
5. **components**
    a. **CalendarioDeAudiencias.vue**: esse é o componente vue de fato, responsável por adicionar o botão do calendário ao DOM da página, abrir o calendário quando o botão for acionado e exibir as datas e horários disponíveis e ocupados, segundo os dados retornados por dados-audiencias.ts.
6. **services**
    a. **dados-audiencias.ts**: esse é o serviço responsável por chamar os endpoints da API do PJe e retornar as audiências com o número e id do processo, data e horário da audiência e cargo judicial do encarregado pelo processo.
    b. **dados-cargo-judicial.ts**: essa é a classe responsável por detectar o cargo judicial do processo selecionado pelo usuário e ajustar o filtro de cargos do calendário de acordo.
7. **utils**
    a. **injetar-style-no-shadow.ts**: é uma funcionalidade utilizada para injetar os temas visuais do v-calendar diretamente dentro do shadow-dom onde a funcionalidade é instalada, assim evitando erros renderização de estilos do calendário.
    b. **obter-lotacao.ts**: esse é um serviço para obtenção da lotação do usuário atual com fallback e usando cache quando disponível. A lotação do usuário se faz necessária para chamar o endpoint de tarefas disponíveis para aquela lotação.

3.3 Testando alterações no código e gerando o build para testes

Para testar qualquer alteração no código, basta rodar o comando para subir o ambiente de desenvolvimento:

```
pnpm play:chromium
```

Atualmente o script abre o navegador Edge, então é importante garantir que o mesmo esteja no path, para evitar erros.

Para gerar o build de um arquivo zip para teste no navegador, siga os seguintes passos:

1. Na raiz do projeto, execute:

   ```
   npm config set registry {url do repositório do nexus do CNJ}
   ```

2. Execute:

   ```
   pnpm install
   ```

3. Execute:

   ```
   pnpm build:shared-utils
   ```

4. Se tiver erro no shared-utils, tente rodar novamente o passo 2 e 3 e depois vá até \packages\shared-utils e execute:

   ```
   pnpm build
   ```

5. Finalmente, para montar a versão de teste, vá até /apps/extension e execute:

   ```
   pnpm build-chromium
   ```

3.4 Ponto de observação

No dia 13/06/2025, o sistema do TRF1 saiu do ar e após ser restabelecido, foi necessário alterar o código da funcionalidade para ela voltar ao seu funcionamento normal.

Na classe dados-audiencias.ts, os IDs das tarefas "[JEF] Aguardando audiência cível - Analisar petição" e "[JEF] Aguardando audiência cível" precisaram ser alterados de **7776778300** e **7776778337** para **7844535881** e **7844535855**, respectivamente.

As mudanças dos IDs de certas tarefas e eventos é um ponto a ser vigiado para manutenções futuras da funcionalidade.

Como medida de segurança, foi adicionado uma verificação com redundância, usando os nomes das tarefas em conjunto com os IDs.