(Public) App Home Design

Authors: liqining.keeling@bytedance.com
Last Update: Aug 2022
Status: In development

Table of content

One page overview

Background

App Home is a place where users can manage their apps like uninstalling, granting permission, etc. With more and more PWAs, users are more likely to want a place to manage their apps, so app home will play a crucial role in improving the user experience of PWAs.

Currently, |chrome://apps| page serves as a launch surface, but it's simple and doesn't offer enough functionality like setting app shortcuts, etc. To make |chrome://apps| have better UX, we plan to re-design the UI and add more functionality to chrome://apps.
What's more, as the framework of |chrome://apps| is a bit outdated, so we decided to try polymer framework to get a maintainable and modular implementation.

In addition, there's no convenient place for users to open frequently using apps, so we propose to add an app home module on the chrome new tab page, and it will help users quickly access their favorite apps. In future, there would be more features added to the app home module. This document will focus on the detailed design about refactoring the |chrome://apps| page, so future app home modules and other new features are not discussed here.

Also, refer to (Public) App Home intro for more introduction details about app home. The document here mainly focuses on app home technical design which comprises frontend UI design and backend implementation.

Tracking Bug

Implementation bug cr/1350406

Goals

- Besides current existing app managements, we will add more new management to enhance, such as creating shortcuts for apps, changing the permission setting, etc.
- Modernize the |chrome://apps| page by using WebUI polymer framework, and refactor the UI to enhance users' engagement with apps and provide a good experience in app management journey.

Non-Goals

- Implement an app home module which will be placed on the chrome new tab page. The app home module will be a new entry for users to access the apps management surface and allow users to open frequently used apps.
- Implement new apps discovery to make users easily find new apps to install.

Design overview

For frontend, the chrome://apps page will be rewritten via polymer framework to follow the material UI design specification, and to get a better UX experience. Besides the frontend framework changes under the chrome://apps, there also have UI changes in it. Refer to (PUBLIC) dPWA launchpad UX for the UX design specification of app home.

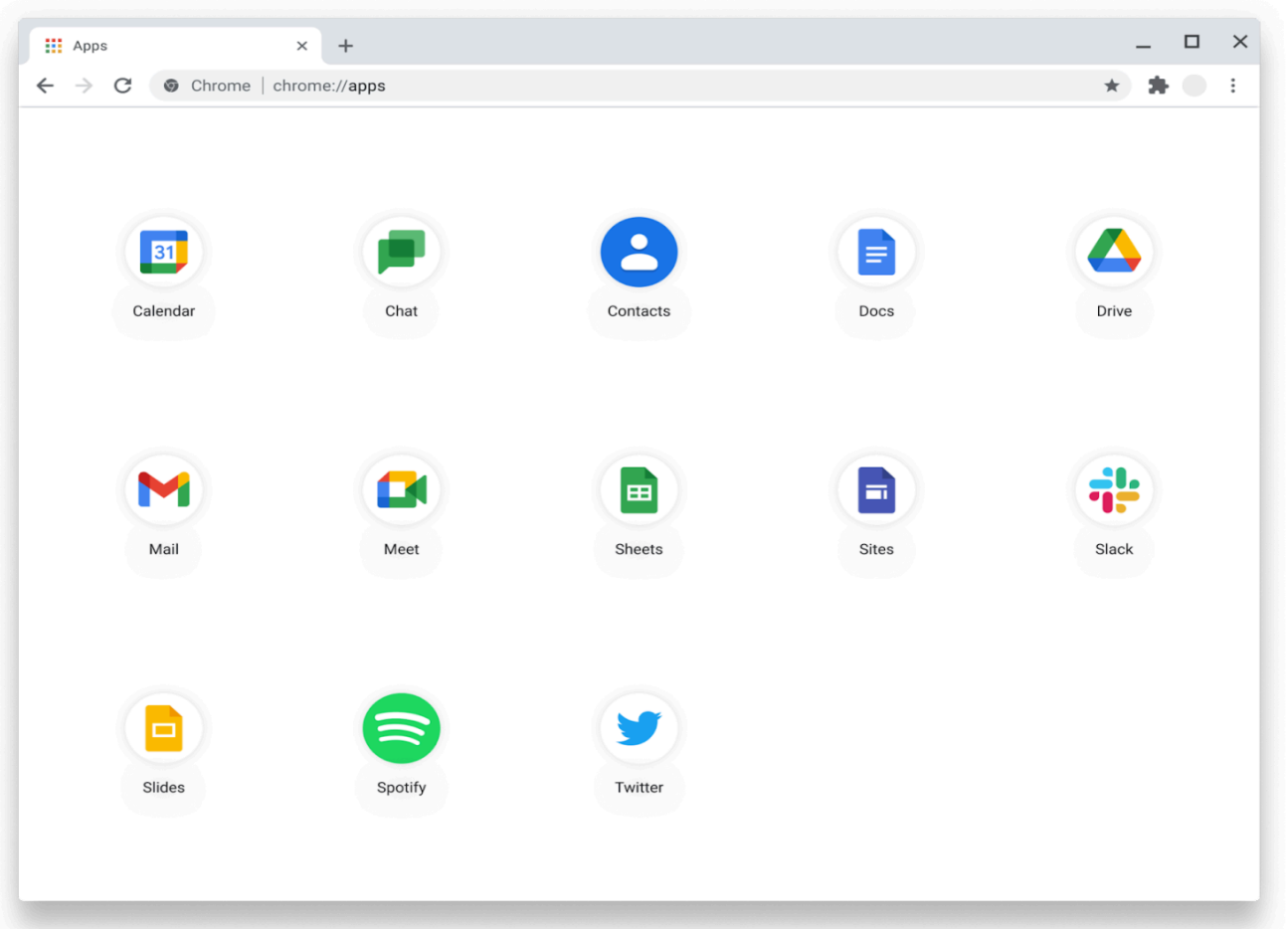
As for backend, we refactor the implementation by just changing the way of message handling between c++ and frontend, and use the |mojom| message handling. In such way, most origin backend logic remain the same, but the API exported for frontend as well as return value are refactored in a |mojom| way.

Since most of the code in |chrome://apps| page currently lies in |NTP4&NTP| folder which served for several types of page, we decide moving the code of |chrome://apps| to an isolated fold named |app_home|.

Detail design

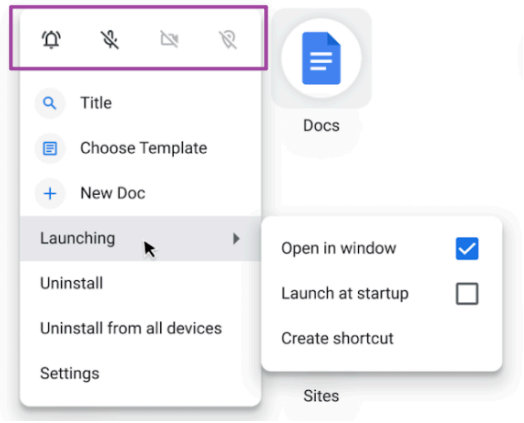
Frontend

Each app item displayed on the chrome://apps page will be implemented as an individual polymer component, which binds to the underlying app data model. The app data source (like app icon, app name, permission status..) for data binding can be obtained by Mojo WebUI handlers, which communicate the underlying apps system.

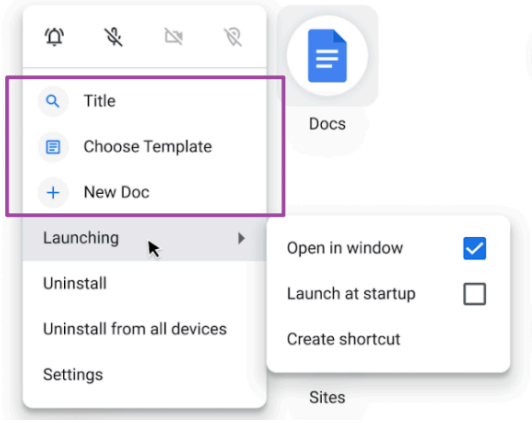


For the function of the right-clicking context menu, we decided to implement it by cascading selectors, similar to a multi-level selector. There are many elements in this part, they are divided into three parts.

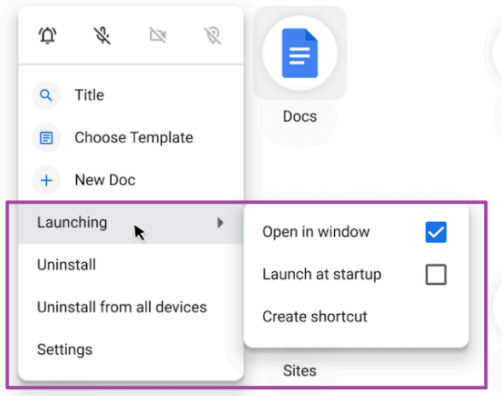
The first part is a set of icons to apply for permission.



The second part is an icon and a description for app shortcuts, which will be implemented based on a multi-row list.



The last part for general application management is based on a cascading selector + multiple checkboxes.



App management

AppInfo list

As for all installed apps, we provide |GetApps()| API which returns a list of |AppInfo| objects contained in the array. The |AppInfo| object defines app's information associated with each app like app's name, permission status, launch setting and so on.

The |AppInfo| is associated with PWA and extension based apps. In the future, extension based apps will be deprecated in |chrome://apps|, to easily remove code relevant to them after deprecation, we document the extension based app specifc fields here.

field in |AppInfo|	explanation
details_url	Url that links app detailed information
homepage_url	Url of the homepage for this extension app
options_url	Url link to options page that allows user to customize the behavior of an extension
version	The version of the extension app
is_deprecated_app	mark the extension app whether deprecated or not

Permission items

We provide |SetPermission(app_id, permission_type)| API to set permissions for an app with specific permission type. Permission type includes notification, location, microphone and camera. Also, each app's permissions status is included in |AppInfo|.

Launch setting

Launch setting defines how apps behave on startup. Currently, three launch configurations will be contained in the launch setting: opening apps in the window, launching at startup and creating the shortcut for apps.

Open in window

Opening in window in app display mode configuration makes the app open in a standalone window. By default, the display mode of apps is opening apps in a standalone window.

For PWAs, the app display mode value is stored in app database, and every time app launch will deserialize this value and decide the way how the app shows. So to enable app open in window, we need to set the display mode as |kStandalone|, and sync the change to the app database. Here, we call SetAppUserDisplayMode in |WebAppSyncBridge| to update the app's display mode in the app database.

For extension based app, call |SetLaunchType| to change the app's display mode.

Launch at startup

The configuration of launching apps at startup offers users a mechanism to allow PWAs auto-launch when users login into their operating system. To setup this configuration, call |InstallOsHook| API implemented in |OsIntegrationManager| and set the |OsHookType| for |kRunOnOsLogin| as true.

App shortcuts

App shortcuts allow users to quickly access common or recommended tasks within web applications. Shortcuts are defined in the manifest file, and each item represents the user intent of launching the app. Shortcuts items in manifest will be parsed in ManifestParser::Parse() and stored in the |WebApp| object. To get web app's shortcut information, we defined |**GetAppShortcutInfo(app_id)**| API, and for-each over all the apps stored in |WebAppRegistrar| and collect shortcut data by calling |shortcuts_menu_item_infos()| API. Each shortcut item comprises url, app icon and shortcut name.

Upon clicking shortcut item of application, we call |**LaunchAppWithUrl(app_id, url)**| to launch a web app and load the specific url passed into the function call.

API overview

The following table lists all backend API exported to frontend.

APIs	**Explanation**
GetApps() => (array<App> apps)	Get all installed apps in chrome
SetPermission(app_id, permission_type)	Set permission for the app specified by app_id. Permission type includes: 1. kNotification 2. kLocation 3. kMicrophone 4. kCamera.
SetDisplayMode(app_id, window_mode)	Set apps window display mode. Display mode includes: 1. kStandalone 2. kBrowser
SetRunOnOsLoginMode(app_id)	Enable apps auto-launch when users login into operation system
CreateShortcut(app_id)	Create shortcut link for app
Uninstall(app_id)	Uninstall app
GetAppShortcutInfo(app_id)	Get app's shortcut data
LaunchAppWithUrl(app_id, url)	Launch app by shortcut

LaunchApp(app_id)	Launch app
ShowAppSetting(app_id)	Open app's site setting page

Testing

[WIP] (public) App Home Test Plan

Metrics

1. Add |AppHomePage.Type| enum metric to track which type of |chrome://apps| page displays. This type includes:
    a. NTP based app home page.
    b. Polymer framework based app home page.
2. Add an enum metric to track all user actions on the new page.
3. Add a launch source to `LaunchSource` used for existing PWA launch metrics.