JAM

Unofficial PBWorks Offline Editor

Documentation

Table of Contents

Introduction

Welcome to Jam. This is a browser extension for Firefox that allows the user to download PBWorks pages, edit them offline, create pages offline, and synchronize changes with the server. It requires you to have an account on the PBWorks workspace you are downloading from, as well as admin privileges. PBWorks is a wiki and real-time collaborative editing system.

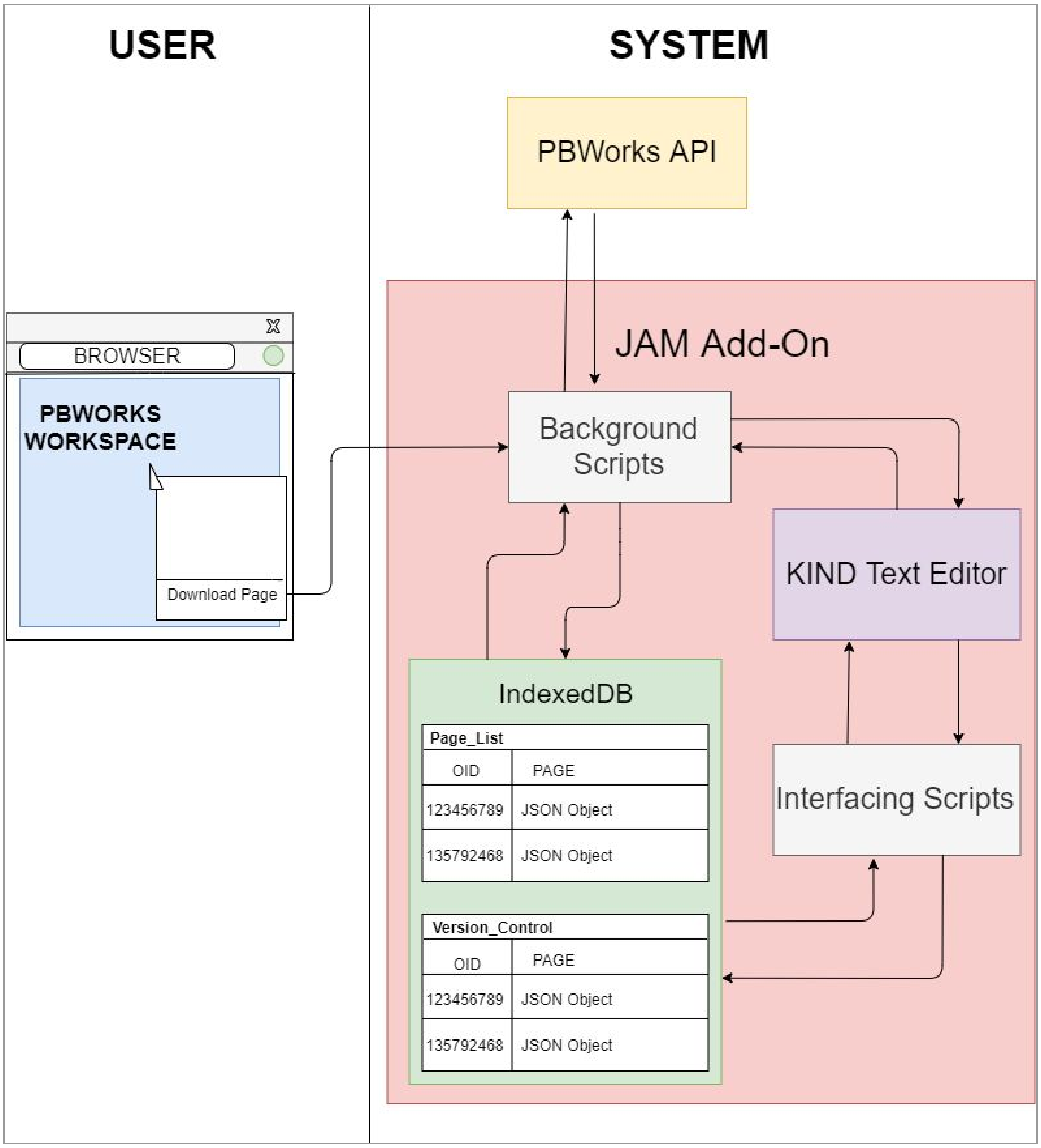
This addon is released via Github, and does not have an entry in Firefox Add-ons Manager. This is intentional, as it is currently a minimum viable product, and some important features are missing. It is assumed that users have some knowledge of software development when using or changing this add-on.

Architecture Overview

Our add-on fulfills the following requirements:

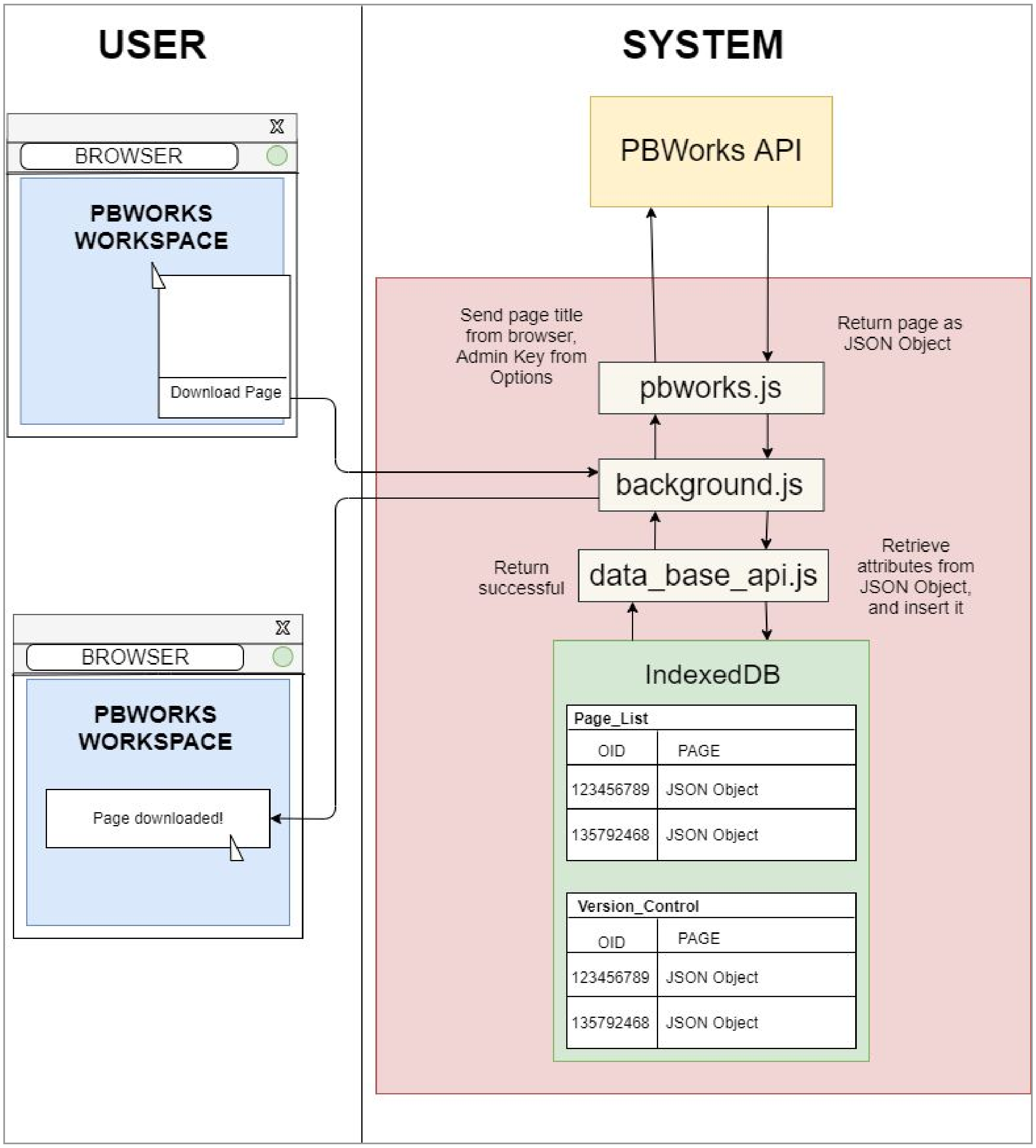
- **Local storage of page objects** - while downloading pages from PBWorks is available, there is no formalised way of storing and accessing them. We needed to consider
    - The type of object being returned
    - The amount of storage available in the browser
    - How much of the local system Firefox Add-ons can reach
    - What database options are available
- **Editing pages offline** - downloaded pages can be edited without the add-on, however this is not a streamlined process, and involves software outside of the browser environment. We considered
    - Offline, open source text editors using JavaScript
    - The size and scope of text editors
    - The format of the page object
- **Search function for locally stored pages** - this is not available through PBWorks, as this feature is strictly local. When developing this, we considered
    - Our file structure
    - The page objects
    - Scripts that use the general file explorer
    - Possible database options
- **Offline creation of new pages** - this is also not available through PBWorks, although manually creating and uploading a JSON object can do the same thing.
- **Solution based in Firefox Addon**
    - This is the biggest influence on the structure of our solution, as it impacts the manner that objects and files can be stored, downloaded, and accessed.

Basic Architecture Overview



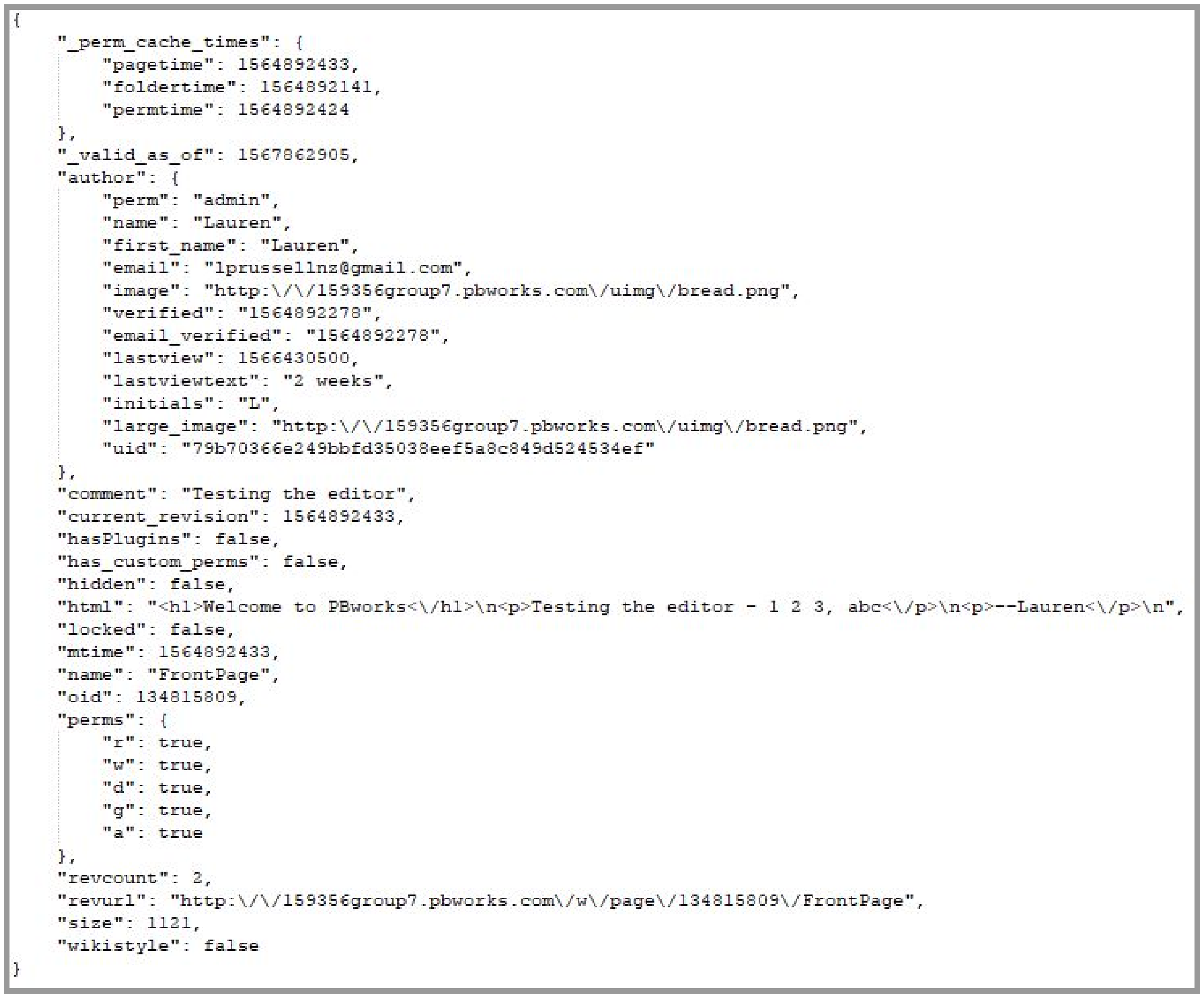
Our add-on contains no servers, and exists only on the front end. Thus, the majority of our project is done in Javascript, interfacing different elements such as the PBWorks API, IndexedDB, and KindEditor.

Downloading and Storing Architecture



When downloading a PBWorks page, the background script first checks the URL of the tab the user is on. If the URL matches the PBWorks pattern (containing the page ID and the title), the background script sends both the page ID (also called the object ID, or OID) and the admin key to the PBWorks API.

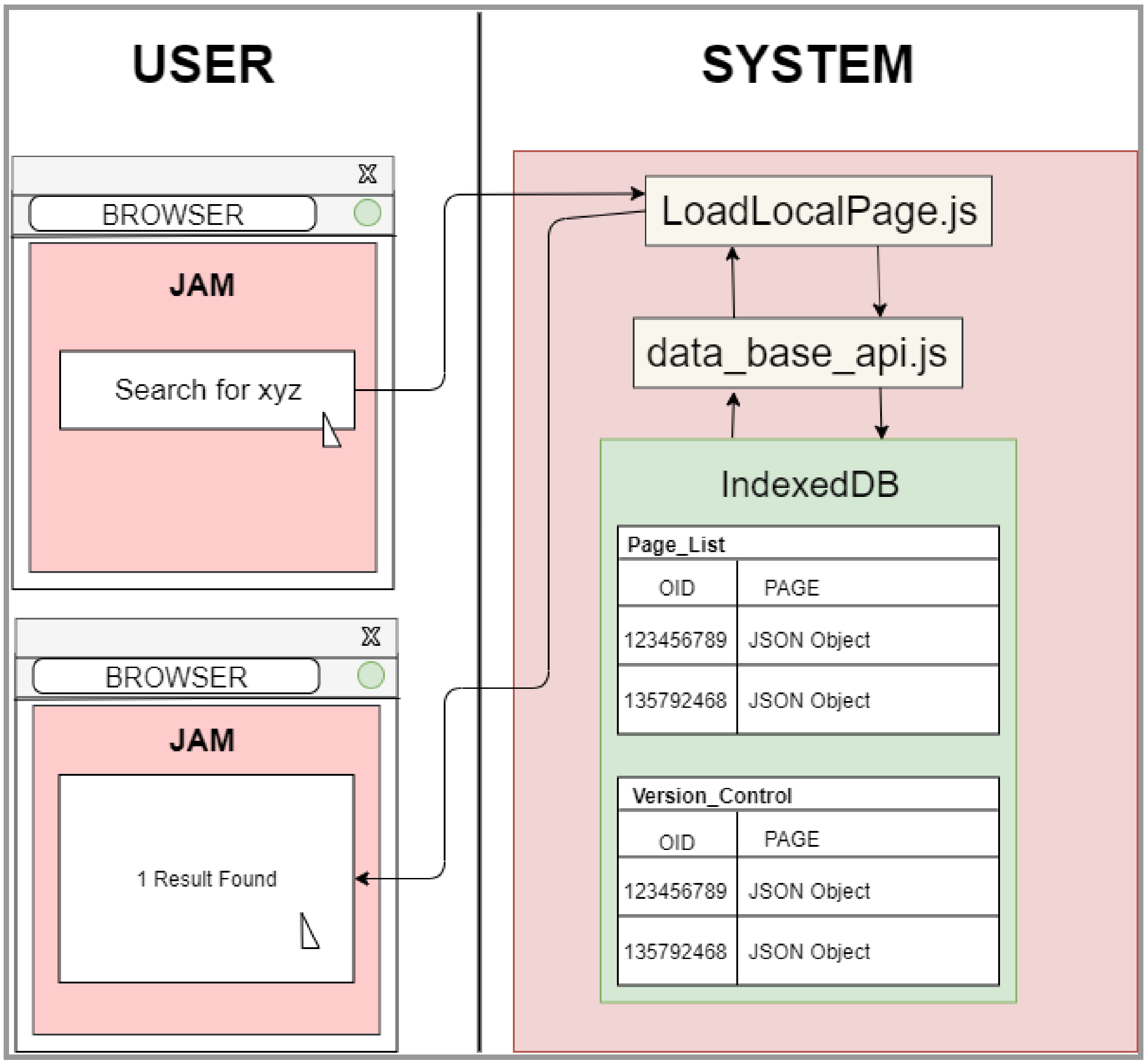
Single pages downloaded from PBWorks are returned in a JSON format. This includes attributes about the author, author email, last online edit, and the HTML for the page content.
This object is saved into two tables, the ‘page_list’ and the ‘version_control’ (more on the Database Structure)



```
{
    "_perm_cache_times": {
        "pagetime": 1564892433,
        "foldertime": 1564892141,
        "permtime": 1564892424
    },
    "_valid_as_of": 1567862905,
    "author": {
        "perm": "admin",
        "name": "Lauren",
        "first_name": "Lauren",
        "email": "lprussellnz@gmail.com",
        "image": "http:\/\/159356group7.pbworks.com\/uimg\/bread.png",
        "verified": "1564892278",
        "email_verified": "1564892278",
        "lastview": 1566430500,
        "lastviewtext": "2 weeks",
        "initials": "L",
        "large_image": "http:\/\/159356group7.pbworks.com\/uimg\/bread.png",
        "uid": "79b70366e249bbfd35038eef5a8c849d524534ef"
    },
    "comment": "Testing the editor",
    "current_revision": 1564892433,
    "hasPlugins": false,
    "has_custom_perms": false,
    "hidden": false,
    "html": "<h1>Welcome to PBworks<\/h1>\n<p>Testing the editor - 1 2 3, abc<\/p>\n<p>--Lauren<\/p>\n",
    "locked": false,
    "mtime": 1564892433,
    "name": "FrontPage",
    "oid": 134815809,
    "perms": {
        "r": true,
        "w": true,
        "d": true,
        "g": true,
        "a": true
    },
    "revcount": 2,
    "revurl": "http:\/\/159356group7.pbworks.com\/w\/page\/134815809\/FrontPage",
    "size": 1121,
    "wikistyle": false
}
```

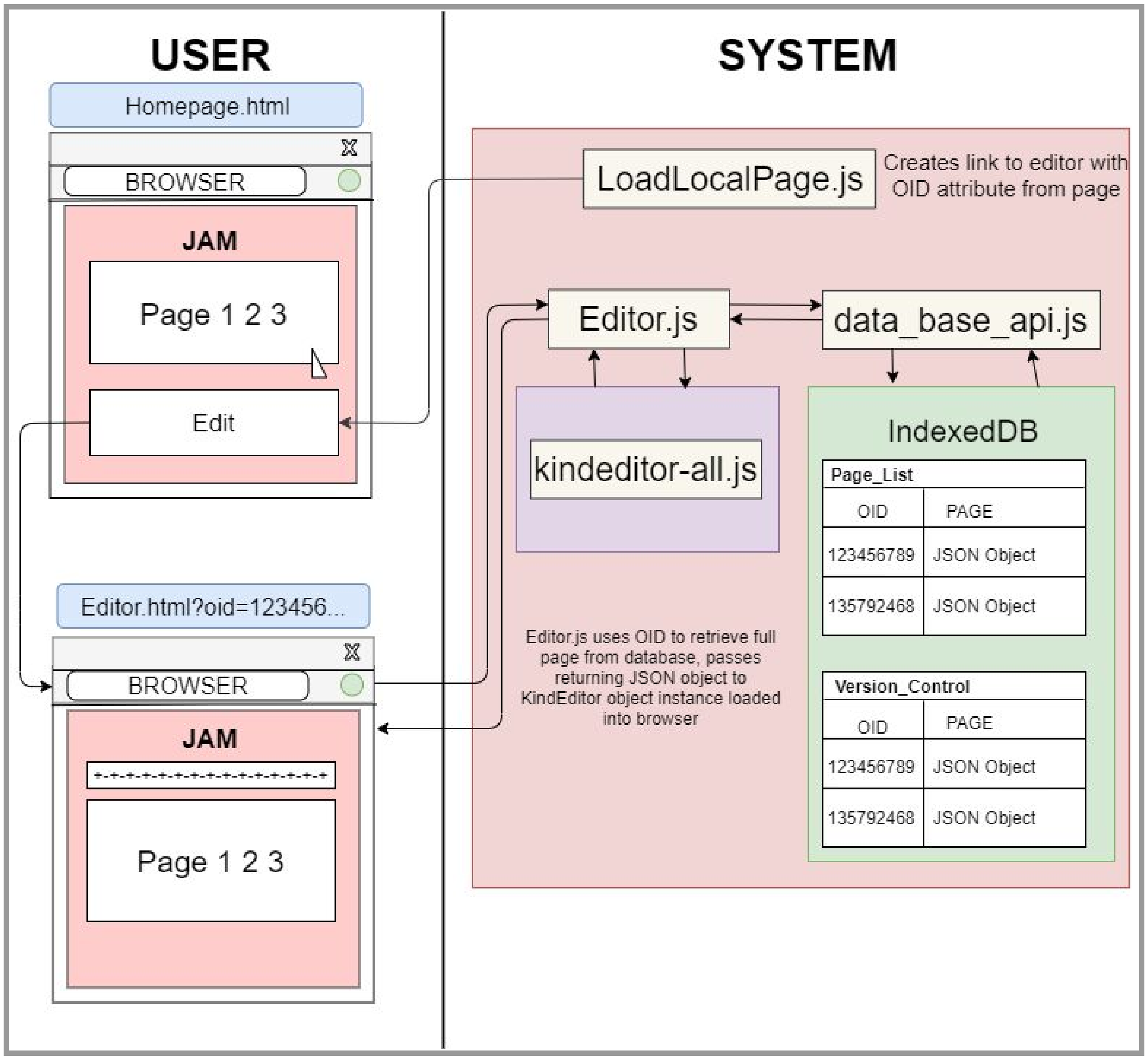
An example of a JSON object returned from PBWorks

Searching for Pages



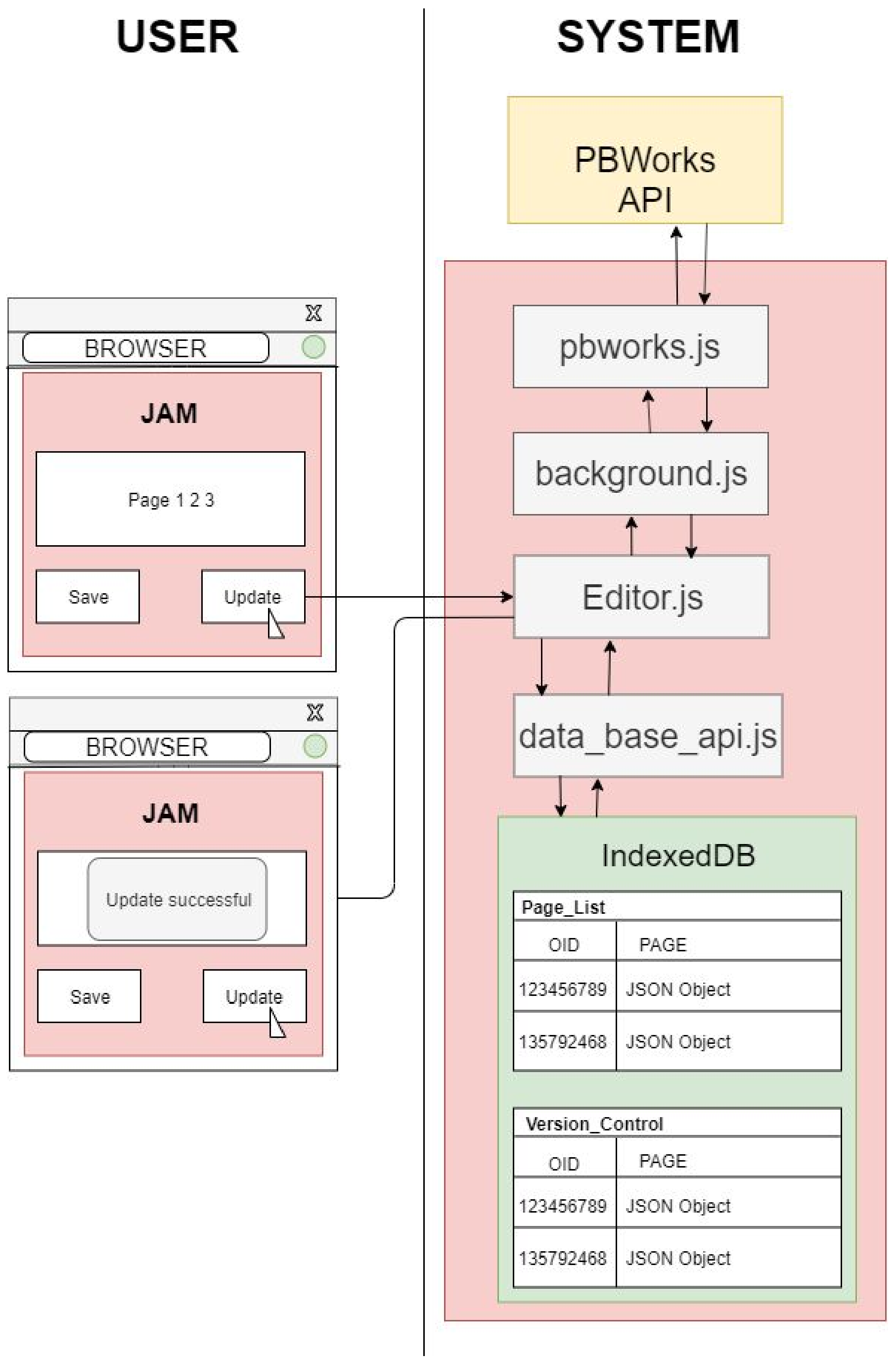
When searching for pages stored locally, pages can be searched for via the OID, the page's last editor, the page title, and the page version (although versioning is not fully implemented yet).
Because of limitations in IndexedDB (seen here), fuzzy searches (approximate search patterns) and multi criteria searches cannot be directly accomplished through IndexedDB. Fuzzy searches are implemented in the LoadLocalPage.js script, when searching for page titles.

Editing a Page



When a user selects a page to view, an edit button is autogenerated that contains a link to Editor.html, and passes the page OID as an argument. Thus, the page object is not directly passed from page to page, only its reference. This OID is used to retrieve the full page object from IndexedDB. While the preliminary workings of version control is included, fully expanding it will require reconfiguring this connection between these pages. The OID will no longer be the primary key, as it will need to be duplicated across multiple entries to track different versions of pages.

Updating/Uploading a Page to PBWorks



Page_List	
OID	PAGE
123456789	JSON Object
135792468	JSON Object

Version_Control	
OID	PAGE
123456789	JSON Object
135792468	JSON Object

When uploading, pages are saved to IndexedDB first, before pushing them to the PBWorks API. In the event that this is a new page created locally, the original OID that will be saved is -1, this allows users to distinguish between pages from PBWorks and ones that are not available online. The OID is generated by PBWorks, and will return when the page is successfully uploaded. The only page attributes that are specifically changed when uploading are 1) the author’s name, and 2) the HTML content. Other attributes are automatically created or adjusted by PBWorks, once the upload is complete.

When uploading the author’s name, the add-on uses the user’s email, workspace name, and admin key to retrieve author details from PBWorks. This leads to the unusual situation where admins are able to impersonate other contributors - this is not a quirk or issue introduced by our add-on, but is a part of the PBWorks API itself. Admins have access to all user details on their workspace, and can use their own credentials plus a user’s email to search for their ID. No verification or login is required to access or use someone else’s email, so users of the add-on should be aware of their personal responsibilities when uploading and accessing admin information.

If a page has been deleted, and user’s upload a local version of it, it will save in the same process as any new page.

Database - IndexedDB

For the storage of our pages, we chose IndexedDB, a kind of NoSQL database. IndexedDB is a way to store large amounts of data using a browser. The data it creates can be queried and used offline. IndexedDB is a very effective solution for programs that need to store large amounts of data or need to be used offline. Our other primary reasons for choosing this database type include:

- As a front-end project, we need front-end storage and don't want to request data from the background. This database is within the Firefox browser, so we don’t need to run the database in the background.
- Key-value pair storage. We can store the object directly.
- Compare to web storage, like LocalStorage, IndexedDB has more storage space. The exact amount of storage depends on your hard drive, so we can say that it is actually unlimited. As for web SQL database, it doesn’t support Firefox.
- Asynchronous. The IndexedDB operation does not lock the browser, and the user can still perform other operations, in contrast to LocalStorage, which operates synchronously. Asynchronous design is to prevent reading and writing of large amounts of data and slow down the performance of web pages.
- Support for indexing. Some other databases, such as LocalStorage, do not provide search functionality and cannot build custom indexes.

Despite the obvious advantages, there are some limitations when it is used:

- Due to the characteristics of the key-value pair storage, we can't establish a connection between two object stores with a foreign key. What’s more, it does not support composite keys.
- Very limited in query function. According to the official API documentation, IndexedDB only supports accurate query. To implement a fuzzy search, the functions must be directly written (in our add-on, these are found in ‘LoadLocalPage.js’. In addition, multi criteria query is not supported.

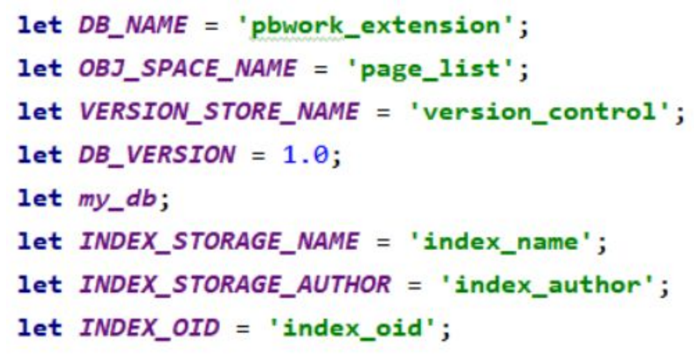
Initializing IndexedDB

Our extension accesses IndexedDB through the data_base_api.js. This includes the initialization of the database and the creation of object stores. When users first download a page, or if searching for a specific page, we will call the function in the API to automatically complete the database establishment and initialization. The specific function is init_database().

Installation of object stores

This part is also in the function init_database(), after we open the database.

The attributes for our extension database are defined in the data_base_api.js., such as the database name, table names, username and so forth. These can be adjusted as needed, and require no password to do so.



```
let DB_NAME = 'pbwork_extension';
let OBJ_SPACE_NAME = 'page_list';
let VERSION_STORE_NAME = 'version_control';
let DB_VERSION = 1.0;
let my_db;
let INDEX_STORAGE_NAME = 'index_name';
let INDEX_STORAGE_AUTHOR = 'index_author';
let INDEX_OID = 'index_oid';
```

Our current database consists of two tables, page_list and version_control. Page_list stores two attributes per page, the object ID (OID) as the key, and the JSON page object.

When searching for a page to edit or view, searches are directed to the page_list table. As discussed earlier, IndexedDb does not allow for foreign or composite keys. Because of this, we were unable to fully establish versioning as a feature in our add-on. As a solution, future developers should look at:

- Trying to model a non-relational database containing a new identification or key
- Using an array as a key - while technically a composite key, this would likely have to be scripted outside of the IndexedDB scope as there are quirks in how keys are defined and handled

Some other important factors to consider when adjusting the database are:

- In PBWorks, titles are used as the primary key for pages. The object identification (OID) can often be used in its place, or to retrieve the page and page title (and vice versa). This is an unusual set up, and we have not been able to fully investigate why.
- Our extension specifically queries PBWorks with the OID because it is also featured in the URL, although page titles could be accessed and used this way as well.
- OIDs are generated by PBWorks, thus to store new pages (not available on PBWorks), we give them the OID -1.

Installation

Distributing Add-ons

Unless using the Developer's edition of Firefox, permanently installing add-ons requires the add-on to be 'signed', before it can be installed. This 'signing' is primarily done by Mozilla to regulate extensions distributed through the Add-ons Mozilla site (AMO). However, it is possible for developers to sign their add-ons, and to self-distribute outside of the AMO environment.

Without signing the add-on, it is not possible to formally install it in the Firefox browser (although some unconventional and unrecommended methods to do so are available online, we have not tried these, nor would we encourage using them).

Before attempting to distribute the add-on for permanent installation, we recommend reading the Mozilla documentation, which covers these aspects in more detail.

- Package your extension
- Submitting an add-on
- Firefox Add-on Distribution Agreement
- Add-on Signing

GitHub - Checking out and Building the Project

For temporary installation, this is a very straightforward process. Download the project from our GitHub repository, and follow our Temporary Installation guide.

If you want to permanently install the add-on, this add-on must be submitted and signed via the Mozilla Developer Hub. Code is verified, and the project file format returned from signing is .xpi.

On installation, our extension requests permission to "Access your data for all websites" and to "Access browser tabs". Our manifest requires the following permissions:

"All_urls", "tabs", "activeTab" : This allows the add-on to detect the page that users are on, to retrieve the URL for downloading pages. It also allows us to run Javascript on the Jam homepage.

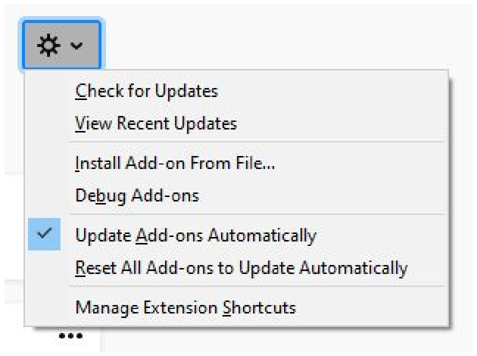
"contextMenus": This is so users can right-click on a page to download, by adding a new option to the context menu.

"Storage": Allowing storage means that pages can be stored in IndexedDb.

Permanent Installation

If you are permanently installing the add-on, you must make sure you have

1. A stable version of the add-on
2. The full add-on code in a compressed .xpi file, signed via the Mozilla Developer Hub
3. The most recent version of Firefox



Check for Updates
View Recent Updates
Install Add-on From File...
Debug Add-ons
Update Add-ons Automatically
Reset All Add-ons to Update Automatically
Manage Extension Shortcuts

With Firefox open, press (Alt + t > a) to open your add-ons manager.

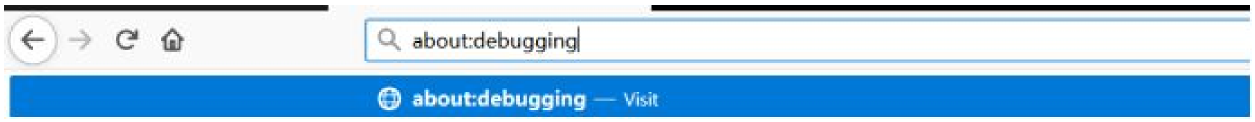
Click the cog in the top right of the screen, and select "Install Add-on From File", and allow the requested permissions (without accepting permissions, the Add-on will not install or work correctly).

The add-on will now be permanently installed in your Firefox Browser.

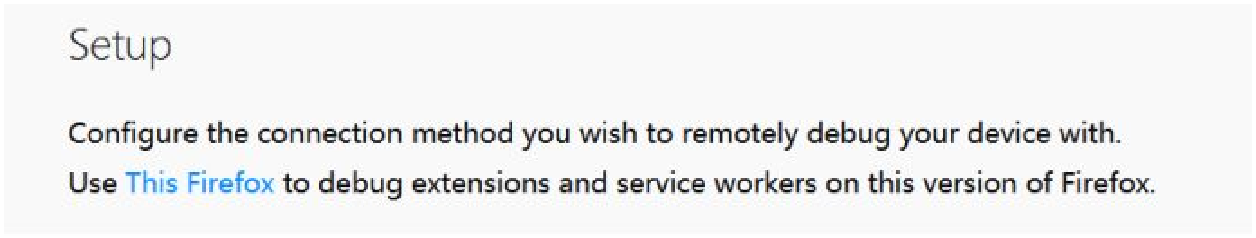
Temporary Installation

If you are making changes to the addon, are testing new changes, or otherwise do not want the add-on to remain on closing Firefox, a temporary installation will work for you.

After downloading the project folder, open '*about:debugging*' in Firefox.



Occasionally, this may redirect to the '*about:debugging#/setup*'. Simply click "This Firefox" to proceed (as shown below)



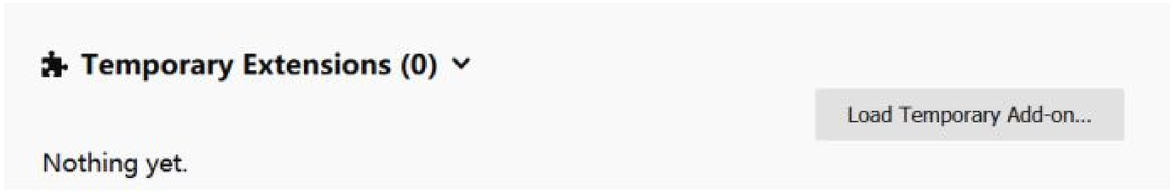
Setup

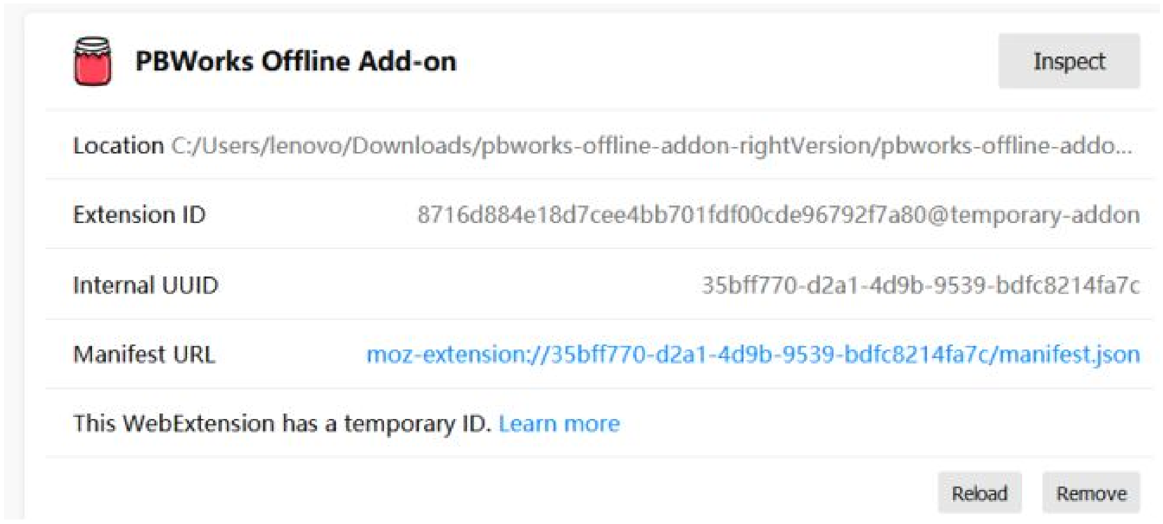
Configure the connection method you wish to remotely debug your device with.

Use This Firefox to debug extensions and service workers on this version of Firefox.

Click "Load Temporary Add-on" and choose the manifest.json file. The addon will now be installed, and will stay until you restart Firefox. If you change some codes, you need to reload the addon.

Alternatively, you can run the extension from the command line using the [web-ext](web-ext) tool.





PBWorks Offline Add-on

Location C:/Users/lenovo/Downloads/pbworks-offline-addon-rightVersion/pbworks-offline-addo...

Extension ID 8716d884e18d7cee4bb701fdf00cde96792f7a80@temporary-addon

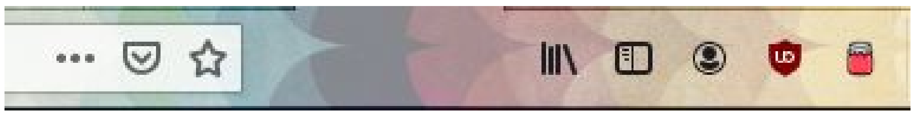
Internal UUID 35bff770-d2a1-4d9b-9539-bdfc8214fa7c

Manifest URL moz-extension://35bff770-d2a1-4d9b-9539-bdfc8214fa7c/manifest.json

This WebExtension has a temporary ID. Learn more

For more information on Firefox Web Extensions, please use the [Mozilla Firefox Guide](#).

After Installation

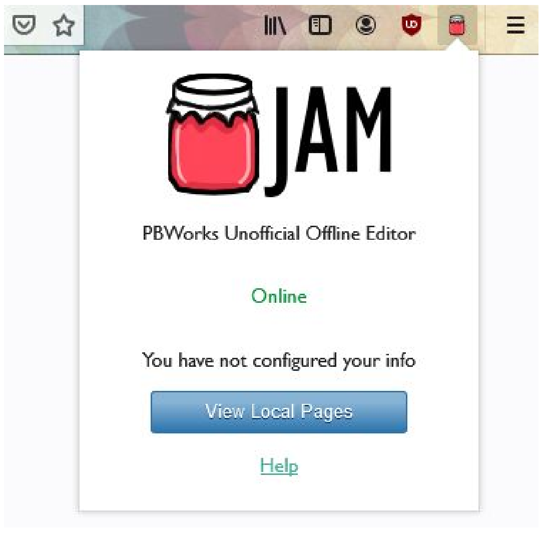


JAM is installed

Once installed, a small jam jar icon will appear in your hotbar. You are almost ready to start downloading pages - click on the JAM jar to view your current status.

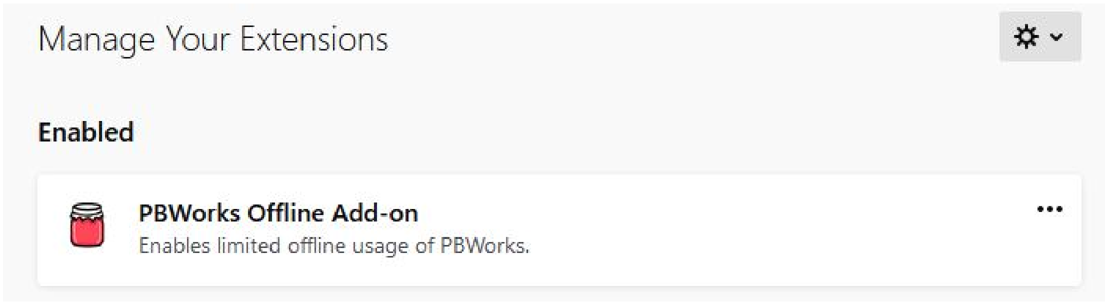
On first installation, you will be notified that your relevant information has not been configured. To set your user information, press (Alt + t > a) to open your add-ons manager.

NOTE: While you can use the editor to create pages at this stage, it is expected that you have set your user information.



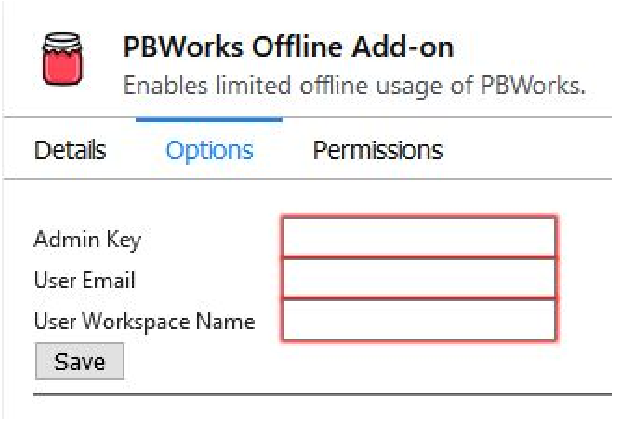
JAM
PBWorks Unofficial Offline Editor
Online
You have not configured your info
View Local Pages
Help

Popup - First Appearance



Select the add-on, and access the Options tab. Here, you will need to enter the Admin Key for your workspace, your email, and the workspace name. Once you have entered and saved your information, the popup will inform you that everything is saved.

NOTE: If any of your information is wrong, the addon will not detect this until it attempts to use your information to access PBWorks



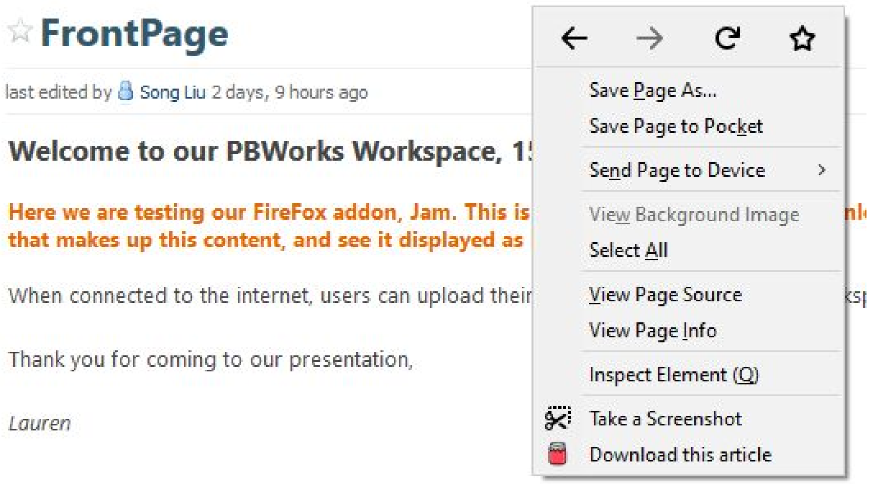
JAM options tab under 'about:addons'

Using the Addon

Download pages

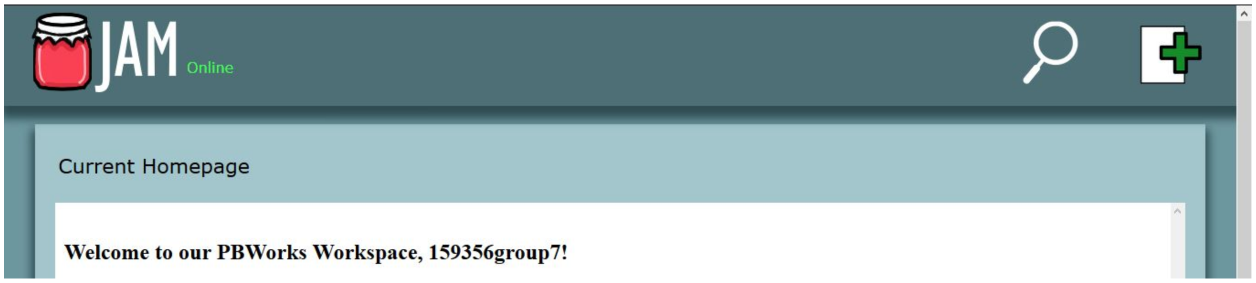
Now that Jam is installed, you should be able to download pages from your Workspace. To get started, find a page on your workspace that you would like to download. Right click on the page, and select "Download this article". If this is successful, you will be alerted that the page is stored in your database. Click on the Jam jar, and select "View Local Pages" to find all of your downloaded and created pages.

NOTE: This will not work for navigation pages such as in 'Pages & Files', as they are not retrievable by the PBWorks API


FrontPage
last edited by Song Liu 2 days, 9 hours ago
Welcome to our PBWorks Workspace, 1
Here we are testing our FireFox addon, Jam. This is
that makes up this content, and see it displayed as
When connected to the internet, users can upload their
Thank you for coming to our presentation,
Lauren
Save Page As...
Save Page to Pocket
Send Page to Device
View Background Image
Select All
View Page Source
View Page Info
Inspect Element (Q)
Take a Screenshot
Download this article

Open and Edit Pages

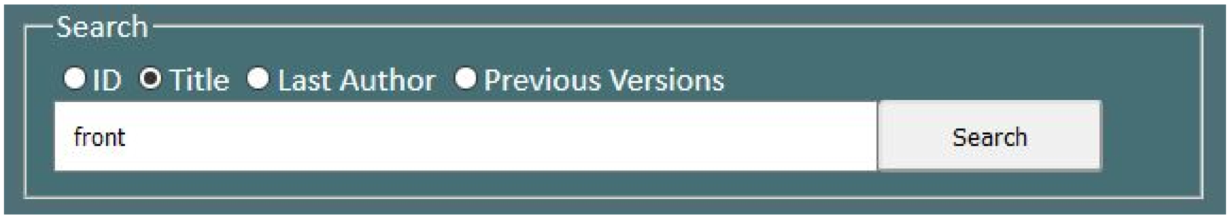
After clicking on “View Local Pages”, a new tab will open in Firefox with the main screen for this addon. If you have downloaded the front page of your workspace (that is, if it is named “FrontPage”) this will be set as your welcome screen.



Current Homepage

Welcome to our PBWorks Workspace, 159356group7!

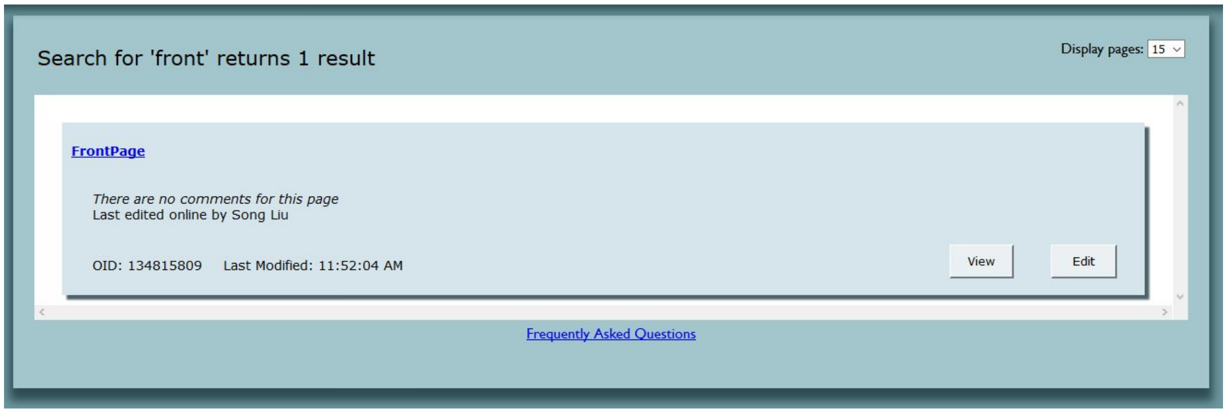
Click the search button, and type the title (or part of the title) that matches your desired page.



Search: ○ ID [✓ Title] ○ Last Author ○ Previous Versions

front

All of the matching searches will appear, with information about the last online author, the comments made from the last version available, and the options to view or edit this page.



Search for 'front' returns 1 result

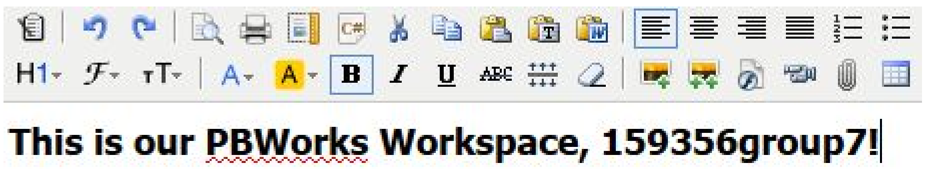
FrontPage

There are no comments for this page
Last edited online by Song Liu

OID: 134815809 Last Modified: 11:52:04 AM

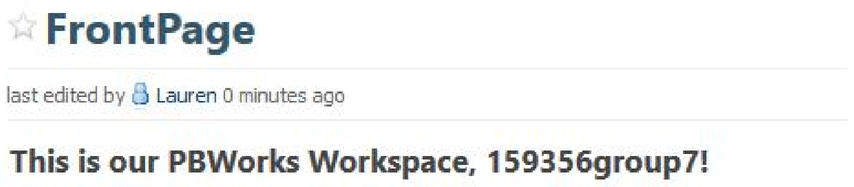
Selecting ‘edit’ will open the page in the local editor. If you are not connected to the internet, you will be unable to upload your page, however you can still save locally.

Save
Update



This is our PBWorks Workspace, 159356group7!

KindEditor works as a standard text editor, change your text as you like, and save. When you are online again, load the page in the editor, and click ‘Update’.
One it is saved and updated, you can see the new version in the editor, and on the PBWorks site.

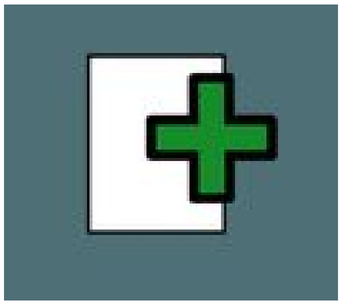


FrontPage

last edited by Lauren 0 minutes ago

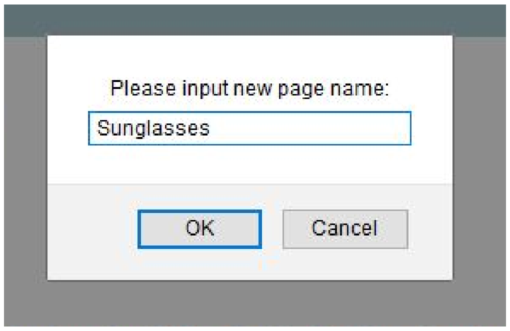
This is our PBWorks Workspace, 159356group7!

Create and Upload New Pages



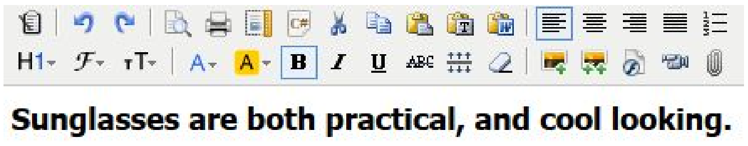
This follows almost the same process as editing any other page - to start, click the new page button on your homepage. Enter a new page name into the textbox, and click 'OK' to continue.

NOTE: Creating a new page does not check if the name has been used by another page on PBWorks, or stored in your IndexedDB. For more information, please see our Limitiations

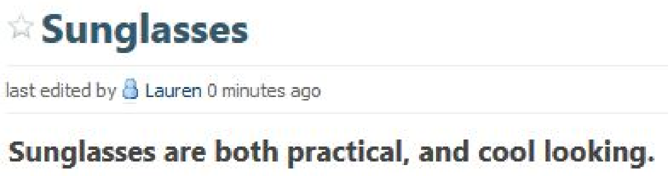


Here, you can save, upload, and edit your new page, the same way you can for any page downloaded from your workspace. As before, 'Save' stores it locally, and 'Update' stores it on your PBWorks workspace.

The difference is in how the PBWorks API uses the page it is sent. This takes a little longer than overwriting a page, as it first checks for any matching pages, then creates one if there is no page with a matching title.



Sunglasses are both practical, and cool looking.



Sunglasses

last edited by Lauren 0 minutes ago

Sunglasses are both practical, and cool looking.

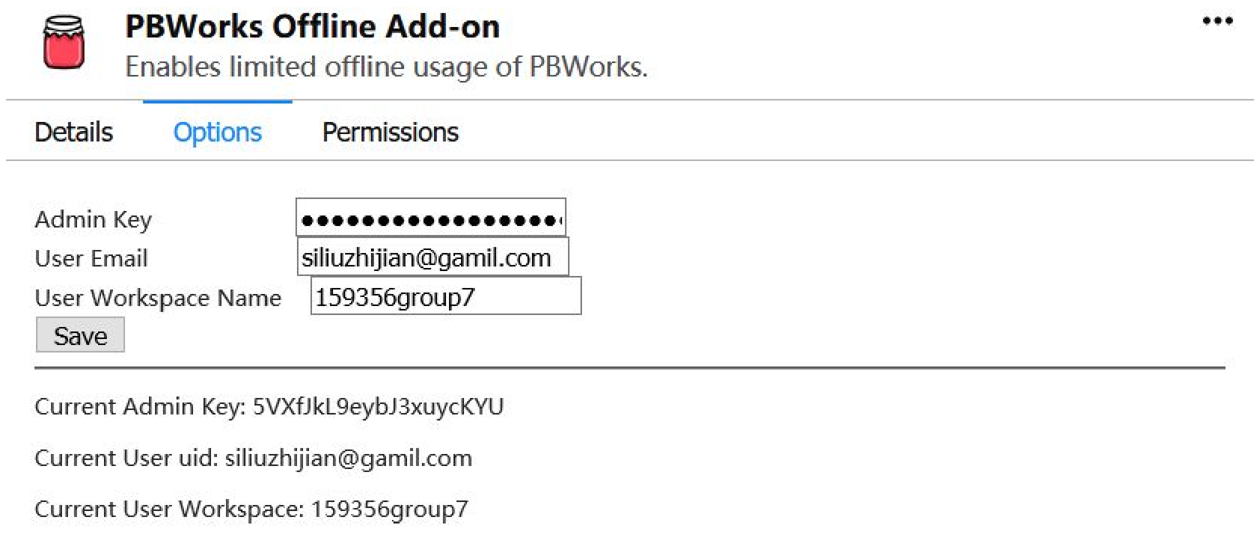
Customisation

Style

While the fundamental part of our project is customizing text and pages, our main user interface does not have specific customization functions (outside of editing our css pages). We have not allowed for style customizations outside of editing css pages - users can find our css files in pbworks-offline-addon-rightVersion\popup\CSS. There is a folder of CSS files, and most codes of the style are in the Homepage.css.

Authentication

When using our addon, users need to authenticate before we can operate.



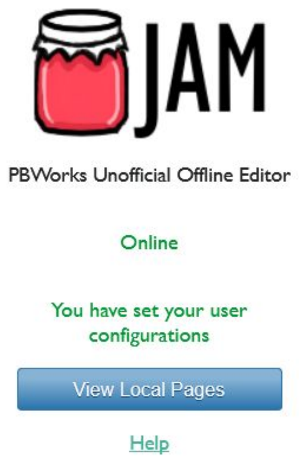
Admin Key ●●●●●●●●●●●●●●●●●●
User Email siliuzhijian@gamil.com
User Workspace Name 159356group7

Current Admin Key: 5VXfJkL9eybJ3xuycKYU

Current User uid: siliuzhijian@gamil.com

Current User Workspace: 159356group7

This is a necessary step. Without it, users can still get the pages, but they can’t install the database and do operations like downloading pages. In this step, we need to enter the workspace admin key, user email and the user workspace name. Only when we enter the correct workspace, we can download things from this workspace. The user email should be the one registered with PBWorks, because it will represent our identity. If we create a new page or edit a page, the author will be the name of the email when it was registered with PBWorks.



You have set your user configurations

Product Limitations

We have implemented the basic functionality required for this project, but there are still some parts that developers should start on for improvement:

1. **Download pages using the POST or PUT method instead of the GET method** - We cannot pass parameters to the PBWorks API using the HTTP POST method, so only the GET method can be used. Currently, this limits the number of characters that can be uploaded to PBWorks. Future developers should look at other ways these pages can be sent, including looking into the AppendPage method in PBWorks

2. **Support upload multimedia files such as photos, videos and so on -** We can upload text with formatting into an old page, or a new page, but currently the add-on cannot upload images, multimedia, or other files. The editor provides the function of uploading photos, but it doesn't work. This may be a limitation of the editor, which can hopefully be fixed.
   When connected to the internet, images that are on the PBWorks page can be viewed in the our extension, but these are not downloaded and accessed locally.

3. **Download all pages of the workspace at once** - We mentioned earlier the difficulties we had in downloading all pages, due to the differing formats that PBWorks returns pages in, and because of the limits to how many calls can be made to PBWorks at a time. Developers looking to implement this feature should consider both the file formats, and the limitations in place by the PBWorks API.

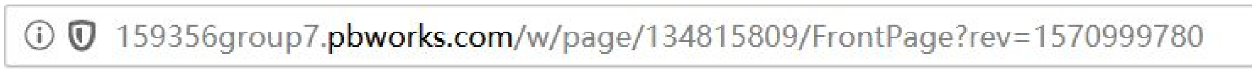
4. **Fully working version control -** This is both the largest, and most complex of the initial requirements future developers will want to investigate. When we started, we hoped to develop a way of seeing different versions and reverting to previous versions of pages, like in PBWorks. To try and accomplish this, we created another object store in the database. We can search a page for previous versions and get the information of the last author and modify time, but we can't edit them. One of the limitations is the database doesn't have foreign keys or composite keys, so it's hard to create connections. Another reason is when we open the editor and edit pages, we use oid to distinguish different pages. But if we want to edit different versions of a page (same oid), we should add another attribute to distinguish these pages.
   This really requires a reworking of how the tables are constructed in IndexedDB, or even the implementation of a new database format. Version control would have a large impact on the functionality of our add-on, and would require other changes such as

    a. How search works - as LoadLocalPage.js expects one array of page objects from IndexedDB, this would need reconfiguring
    b. What variable is passed to the editor URL, and how this variable is used to acquire the page object
    c. How downloading/uploading to and from PBWorks functions. Before storing a downloaded page, the add-on will need to check whether or not it has a version of this page already, whether it is the same version of any local page, and so forth. When uploading an updated page, the page itself will need to be accessed/downloaded to check the last time it was edited.

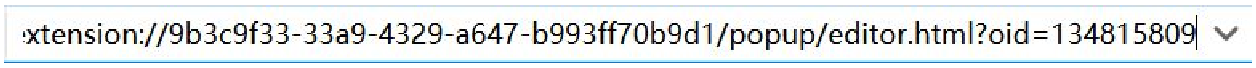
5. **Upload multiple pages at the same time according to users' selection** - This requires the version control to be functional, in order to distinguish what pages can be updated, which ones already match pages on PBWorks and so forth. This feature also requires a reworking of LoadLocalPage, which currently does all of the loading and updating on the homepage.html.

6. **Security issues & Authentication** - While we cannot change how PBWorks handles authentication, upgrades can be made to how our add-on handles JSON objects and the admin key, to provide more security for user data.

7. **Sort function for search result** - This function is also dependent on version control upgrades, as our add-on uses OIDs as the primary key in our tables when PBWorks tends to use the actual page title. How future developers handle this depends on the updated structure of the object storage/tables and the general database.


159356group7.pbworks.com/w/page/134815809/FrontPage?rev=1570999780

Above is a version of page in PBWorks


xtension://9b3c9f33-33a9-4329-a647-b993ff70b9d1/popup/editor.html?oid=134815809

Our editor only distinguish pages by oid