Austin Slaughter

Abstract

Visual JSON Editor is a Visual Studio Code extension that allows you to open JSON (JavaScript Object Notation) files in a “What You See is What You Get” editor tab.

SRS: Visual JSON Editor

# 1. Introduction (Purpose)

Visual JSON Editor (the Extension) is a Visual Studio Code (VS Code) extension that allows you to open JSON (JavaScript Object Notation) files in a “What You See is What You Get” editor tab. With its heavy use of braces, quotes, and nesting, JSON can often be a pain to read/write by hand, especially if it’s unformatted. With this extension, I aim to create a useful visualizer and editor for JSON files that is superior to handwriting it, especially for beginners.

# 2. Requirements

## 2.1. Mandatory Requirements

1. **The Extension shall provide a custom editor for JSON files in VS Code.**
   1. “Custom editors allow extensions to create fully customizable read/write editors that are used in place of VS Code's standard text editor for specific types of resources” (*Custom Editor Api*).
   2. **Success:** Visual JSON Editor will be available on the VS Code Extension Marketplace.
2. **The Extension shall allow users to edit the values of the six types of items available in a** JSON file: object, array, Boolean, number, string, and null.
   1. **Success:** Once a JSON file has been opened in an editor, a user will be able to click on a value to edit it (or otherwise interact with it in an appropriate fashion).
3. **The Extension shall allow the names of items to be edited as well.**
   1. **Success:** Clicking on the name of an item will allow it to be edited.
4. **The Extension shall provide a way to change the type of an item, albeit limited to** conversions that can be done safely.
   1. **Success:** Clicking on the type of an item (which is displayed near the name) will allow a different type to be picked if there are types it can be converted to.
5. **The Extension shall allow the addition of new items to objects, arrays, and the root (top-level) object.**
   1. **Success:** Each of the mentioned types of item will have an “add” button in a suitable location that creates a new child item.
6. **The Extension shall allow the deletion of any item in the object.**
   1. **Success:** All items will have a “delete” button in a suitable location.

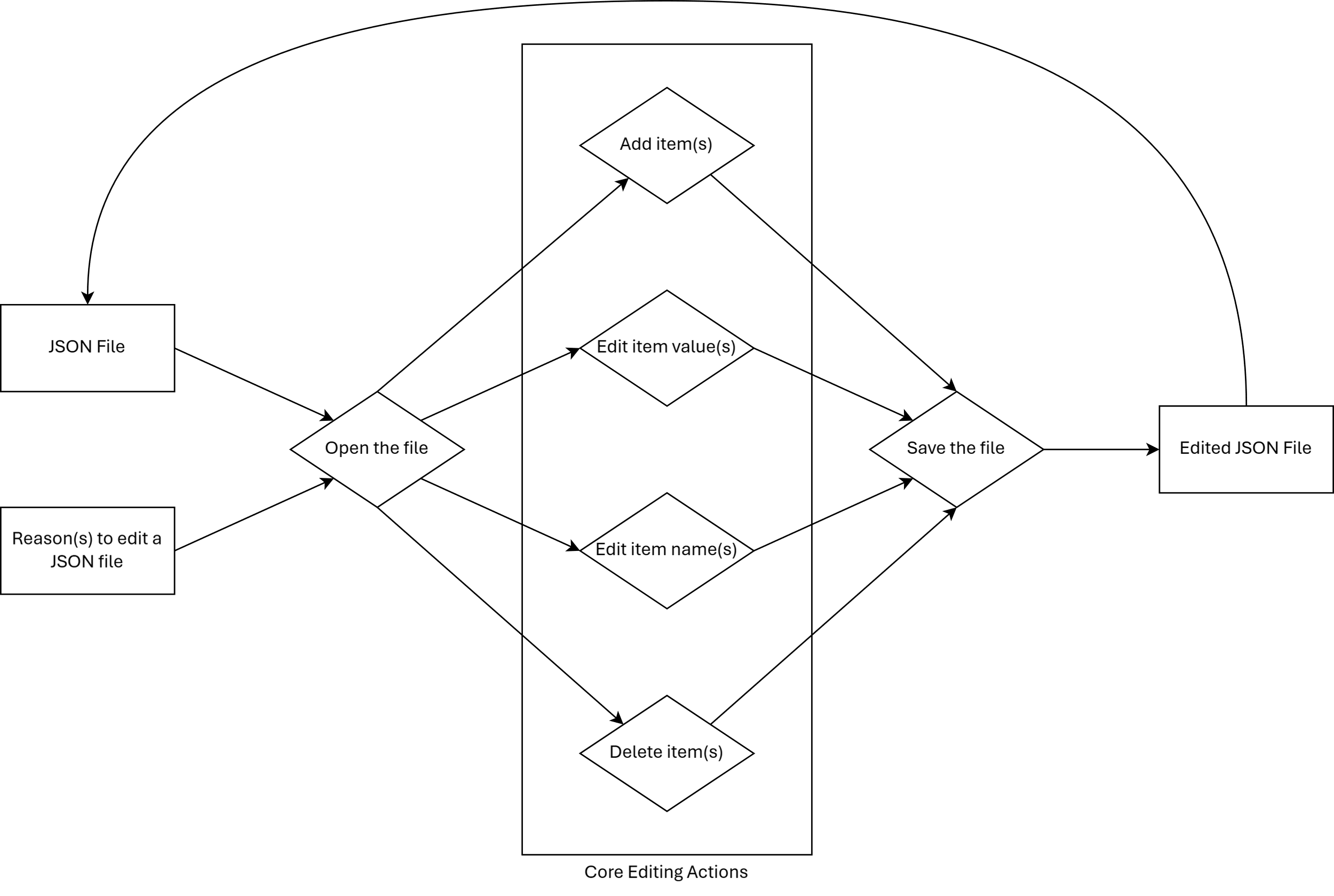
## 2.2. Optional (“Stretch”) Requirements

1. **The Extension should provide several settings to customize its functionality.**
   1. **Success:** VS Code provides a settings screen that all extension settings can be accessed from.
2. **The editor should look very similar to the rest of the VS Code user interface.**
   1. **Success:** Clearly visible if an editor is open; similarity judged by the user
3. **The editor should provide special item editors for the following:**
   1. **Colors** – at least 6-8-digit hexadecimal format
   2. **Dates and times** – at least ISO 8601 formatted
   3. **“Raw” JSON**, allowing for the conversion of objects and arrays between UI elements and plain JSON (for importing existing JSON or getting an alternate view of the item hierarchy)
   4. **Arbitrary files** to be converted to Base64
      1. Images added this way will be displayed in the editor
   5. **Success:** Each special editor, if implemented, will be available in the type selection dropdown menu for at least null items. Ideally, items in compatible formats will be auto-detected and set to these types automatically.
4. **The editor should provide a way to change the order of items in objects, arrays, and the root** object.
   1. **Success:** UI elements (a drag-and-drop handle or up/down buttons) will exist to implement this functionality.
5. **The editor should support undo, redo, and revert functionality as expected by VS Code.**
   1. **Success:** Undo (Ctrl+Z), Redo (Ctrl+Y), and “File: Revert File” (VS Code command) will work when an editor is focused.

# 3. Design Overview

## 3.1. User Workflow

A user starts with a JSON file and some reasons to edit it. They open it in Visual JSON Editor, which allows them to add items, edit values, edit names, and delete items. They can then save the file, resulting in an edited JSON file.



## 3.2. Resources – Dependencies

* **Visual Studio Code**: Program the Extension extends the functionality of; also, the IDE being used to develop it
* **Major NPM Packages:**
  + **Codicons** ([@vscode/codicons](https://www.npmjs.com/package/@vscode/codicons)): VS Code’s icon library, used in editor views for quick identification of item type. Webviews and custom editors must depend on this manually; VS Code provides few of its own resources to them.
  + **ESBuild** ([esbuild](https://www.npmjs.com/package/esbuild)): Builder, bundler, and minifier used to compile and package the extension.
  + **Fast HTML Parser** ([node-html-parser](https://www.npmjs.com/package/node-html-parser)): Used in the extension backend to read data from the HTML of editor views when saving files.
  + **TypeScript** ([typescript](https://www.npmjs.com/package/typescript)): Provides the TypeScript to JavaScript transpiler.
  + **VS Code API type definitions** ([@types/vscode](https://www.npmjs.com/package/@types/vscode)): Type information for the VS Code API, necessary during development of the extension backend.

## 3.3. Data at Rest

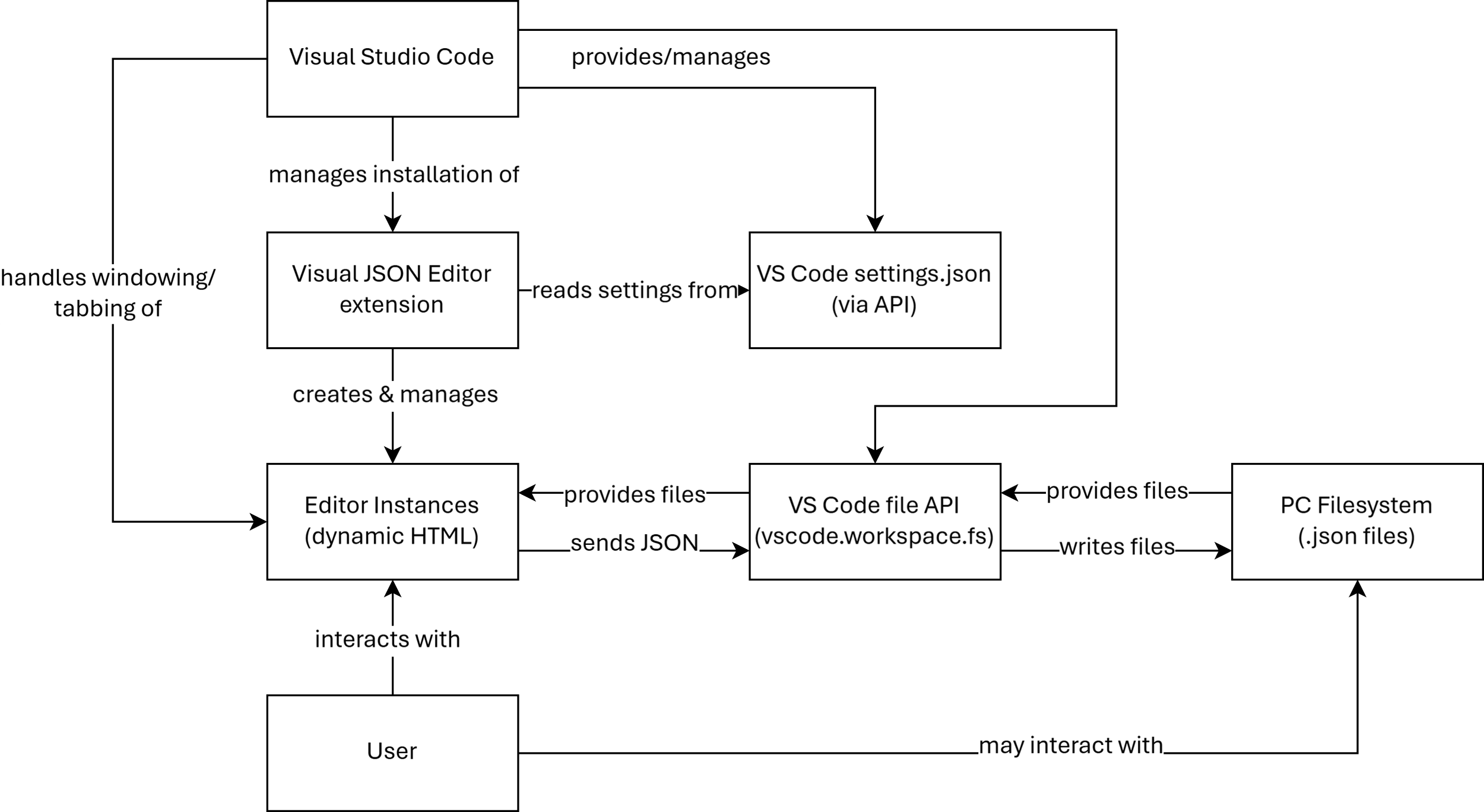
The Extension’s single piece of persistent data is its settings, which are managed by VS Code: “VS Code extensions store their settings in your user or workspaces settings files and they are available through the Settings editor UI… or via IntelliSense in your settings.json file” (*Visual Studio Code User and Workspace Settings*). As of writing, there is one setting.

As the Extension uses VS Code APIs to interface with the files it opens and edits, it does not directly handle the storage of user-generated application data.

## 3.4. Data on the Wire

Saved information is stored on disk. All file operations are done through the VS Code API and locally on the machine that is running VS Code.

## 3.5. Data State (Flow Diagram)



## 3.6. HMI/HCI/GUI

A screenshot of a computer

Description automatically generated

# 4. Verification

## 4.1. Demonstration

While the codebase is set up for unit tests to be added, the easiest and most obvious way to demonstrate that it works is to test it manually (by using it). Any existing JSON file can be opened in it to verify the file opening functionality, and the Extension provides everything necessary for a user or tester to generate their own data. The following section will define how to verify each requirement.

## 4.2. Testing

### 4.2.1. Mandatory Requirements

1. **Custom Editor**
   1. **Verification:** Ensure the Extension creates a unique, interactive JSON editor that opens JSON files in a format other than VS Code's text editor. This should include a user interface that differs from the standard editor.
   2. **Criteria:**
      1. Users can open JSON files with this editor, which should become the default for JSON files on installation.
      2. The editor must appear in VS Code’s extension marketplace under “Visual JSON Editor.”
   3. **Unfulfilled if:** JSON files still open in the standard text editor after installation, or if the Extension does not appear in the marketplace when searched for.
2. **Editable Item Values**
   1. **Verification:** Confirm that users can select and modify values directly within the editor for each data type: object, array, Boolean, number, string, and null.
   2. **Criteria:**
      1. A user clicks on a value and can alter it without switching to raw JSON text.
      2. Each of the six item types should be identifiable and editable.
   3. **Unfulfilled if:** Any data type (besides null) is not editable, or values cannot be modified in the editor.
3. **Editable Item Names**
   1. **Verification:** Ensure users can select and modify the names (keys) of items directly within the editor.
   2. **Criteria:**
      1. A user can click on a key/name, which then becomes editable.
   3. **Unfulfilled if:** Keys/names are unclickable or cannot be modified directly in the editor.
4. **Changeable Item Types (Safe Conversions Only)**
   1. **Verification:** Check that users can convert between item types where safe conversions apply (e.g., number to string but not array to Boolean).
   2. **Criteria:**
      1. Users can select a type for each item, with appropriate types available based on existing data.
      2. Incompatible types should not be selectable.
   3. **Unfulfilled if:** No type change option is available, or conversions lead to data corruption.
5. **Add New Items to Objects, Arrays, and Root Object**
   1. **Verification:** Ensure there’s an option to add new items to objects, arrays, and the root object.
   2. **Criteria:**
      1. Each type (object, array, root) has an “add” button positioned logically for the user.
   3. **Unfulfilled if:** Missing “add” buttons or new items cannot be created within any of the specified types.
6. **Deletion of Items**
   1. **Verification:** Check for a delete option for each item, ensuring users can remove entries from the JSON structure.
   2. **Criteria:**
      1. A “delete” button or similar UI control should be visible and functional for each item.
   3. **Unfulfilled if:** No delete functionality exists, or deletion does not fully work.

### 4.2.2. Optional Requirements

1. **Extension Settings for Customization**
   1. **Verification:** Confirm that users can customize settings for the editor within VS Code’s standard settings screen (if searched for).
   2. **Criteria:**
      1. The settings screen displays extension options, and changes apply to the editor’s behavior.
   3. **Unfulfilled if:** No settings appear in VS Code, or changes do not impact the editor.
2. **Visual Consistency with VS Code UI**
   1. **Verification:** Evaluate if the editor visually aligns with VS Code’s native UI.
   2. **Criteria:**
      1. Fonts, icons, and layout should appear similar, ensuring a seamless experience.
   3. **Unfulfilled if:** The editor’s appearance clashes with or does not resemble the VS Code UI, resulting in a jarring user experience.
3. **Special Item Editors for Colors, Dates, JSON, and Files**
   1. **Verification:** Confirm the editor allows handling specific formats (hex colors, ISO dates, JSON strings, files for Base64 conversion).
   2. **Criteria:**
      1. Users can select special editors for these formats, and compatible types are auto-detected where possible.
   3. **Unfulfilled if:** No special editor options appear, or compatible items are not detected.
4. **Reordering Items in Objects and Arrays**
   1. **Verification:** Ensure there is a mechanism (drag-and-drop or buttons) to reorder items inside arrays, objects, and the root object.
   2. **Criteria:**
      1. Reordering controls are present, and items can be moved without errors.
   3. **Unfulfilled if:** Reordering is impossible, or controls do not function consistently.
5. **Undo, Redo, and Revert**
   1. **Verification:** Confirm that the editor supports undo, redo, and revert functionalities using standard VS Code shortcuts and commands.
   2. **Criteria:**
      1. Users can undo and redo changes with shortcuts and reset the editor to the last saved state.
   3. **Unfulfilled if:** Undo, redo, or revert do not function within the editor.

# Works Cited

*Custom Editor Api*. <https://code.visualstudio.com/api/extension-guides/custom-editors>. Accessed 28 Oct. 2024.

*Visual Studio Code User and Workspace Settings*. <https://code.visualstudio.com/docs/getstarted/settings>. Accessed 28 Oct. 2024.