Resume Generator Configuration App Conceptual Design

Jon Weatherspoon

December 28, 2020

Contents

1 High Level Flow

Generating a new resume shall be done using the configuration app. If the user does not have a resume shell file (template file) ready, they will have to create one.

Once they have a template file, the user will load it into the configuration app through the file menu (or through the button in the preview pane). Upon loading the file, the application will grab the template and theme ids from it and attempt to find a matching JSX file / theme configuration respectively. If both are found (or just the JSX file if the theme is an empty string), the file parsing will continue. If there are any regions with sections present, they will be parsed into the "editor" section of the configuration app.

The editor section will be presented as a hierarchy following the same structure as the opened file. The base of the hierarchy will be the template / configuration's name, followed by a list of named regions. Each region may have zero or more components and each component may have zero or more editable properties.

The user shall be able to click on a region to add a new component to it. When doing this, a dialog should appear, listing out the available components. These will be pulled from some data provider (likely a json file at first for simplicity, database in the future).

The user shall be able to expand a component to show its properties. The user shall be able to edit those properties depending on their data types. The editors displayed in the application will depend on the data type in the property definition. See 3 for details.

2 Creating a New Resume Configuration File

If no resume configuration file has been loaded, the preview / editor sections will be replaced by a canvas with the text "You must load or create a resume configuration file to view content." Underneath the text will be two buttons. The first will spawn a dialog to create a new resume configuration file and automatically load it into the application. The second button will spawn a file browse dialog and allow the user to select an existing configuration file.

2.1 Creation Process

When creating a new resume configuration file, a dialog will spawn asking for the following information:

- The desired filename (required)
- The location to save the file to (required)
- The resume template to use (required)
- The theme definition to use (optional)

The user will have to select the resume template from a list retrieved by some provider (likely a JSON mapping at first, then database in the future).

Once the information has been entered and the user clicks okay, the file will be created and loaded into the application. The resume's template cannot be changed at this point. If the user wishes to change the template, they will have to create a new resume and select that template.

3 Building the Editor Section

When loading an existing resume configuration file, the application will have to build the editor / preview sections to match the properties defined in the file. To do this, we will follow these steps:

1. Load the configuration file as a javascript object
2. Verify the configuration's template id is valid. Fail if it is unknown
3. If present, attempt to verify the configuration's theme id is valid.
    - If not, set it to the empty string
4. Add a node to the editor section containing the configuration name
5. Under it, add a ThemeEditor node for the theme
6. For each region defined in the configuration:
    (a) Add a new region node with the corresponding name
    (b) For each component in the region:
        i. Look up the component definition from the component id
            - If not found, fail?
        ii. For each property in the component definition:
            A. Add a new property editor node based on the property's data type.
            B. If the property exists on the component defined in the resume configuration file, set up the new editor with that value. Otherwise, use the default value from the component definition.

3.1 ComponentFactory

The ComponentFactory will be used to transform a component id to a JSX element that can be added to the virtual DOM. We shall create a IResumeComponent interface that will include a component id (string) and an implementation of the factory method. (probably more?)

A ComponentFactory class shall be created and it shall import all existing IResumeComponents upon initialization. When parsing a configuration file, we will query the factory for a component with the given name. If found, the factory will create it and pass it back to us. Our job is to then add it to the preview section of the application.

Each new type of component will have to implement the interface and the dev will add an import for the component into the ComponentFactory definition. Or perhaps, the ComponentFactory class will be a singleton or something and each component type will just register to it?

3.2 PropertyEditorFactory

3.3 Editor Section Layout

The resume editor section will be a hierarchical control that uses nested MUI lists. The root node of the editor will contain the name of the loaded resume template file. Under it will be an expandable "Regions" node and an editable "Theme" property. Expanding the "Regions" node will show all regions defined by the selected template.

3.4 Base Editor Implementations

3.4.1 TextEditor

3.4.2 DateEditor

3.4.3 ListEditor¡T¿

3.5 Specialized Editor Implementations

3.5.1 ThemeEditor

3.6 Saving a Configuration