Resume Builder

The test assessment overview

Chosen implementation way

I researched different ways how to implement client-side PDF document generation.
Unfortunately, popular solutions like jspdf and html2pdf have some limitations and issues with page CSS (a PDF document looks different than an HTML page).
I decided to use the “print to pdf” built-in browser feature (convert HTML page to PDF document via browser print feature).

Therefore, the “Resume Builder” front-end app should generate an HTML page of a CV template ready to “print” by a browser from a generic CV template.

Concept overview

The main idea of this “Resume Builder” implementation is reusable blocky CV templates.
These templates should allow changing blocks’ data for each unique candidate on a CV generation step.

The CV template is a tree data structure.
Nodes (blocks) of this tree can be of two types:

1) Layout blocks

In the POC application, only one block of this type is implemented - the “layout” block.
This block doesn’t display any data. It’s a container for reserving some space or wrapping other blocks. This block manages content alignment, background color, and other display options.
This block can have child blocks, but may be a CV template tree’s leaf too.

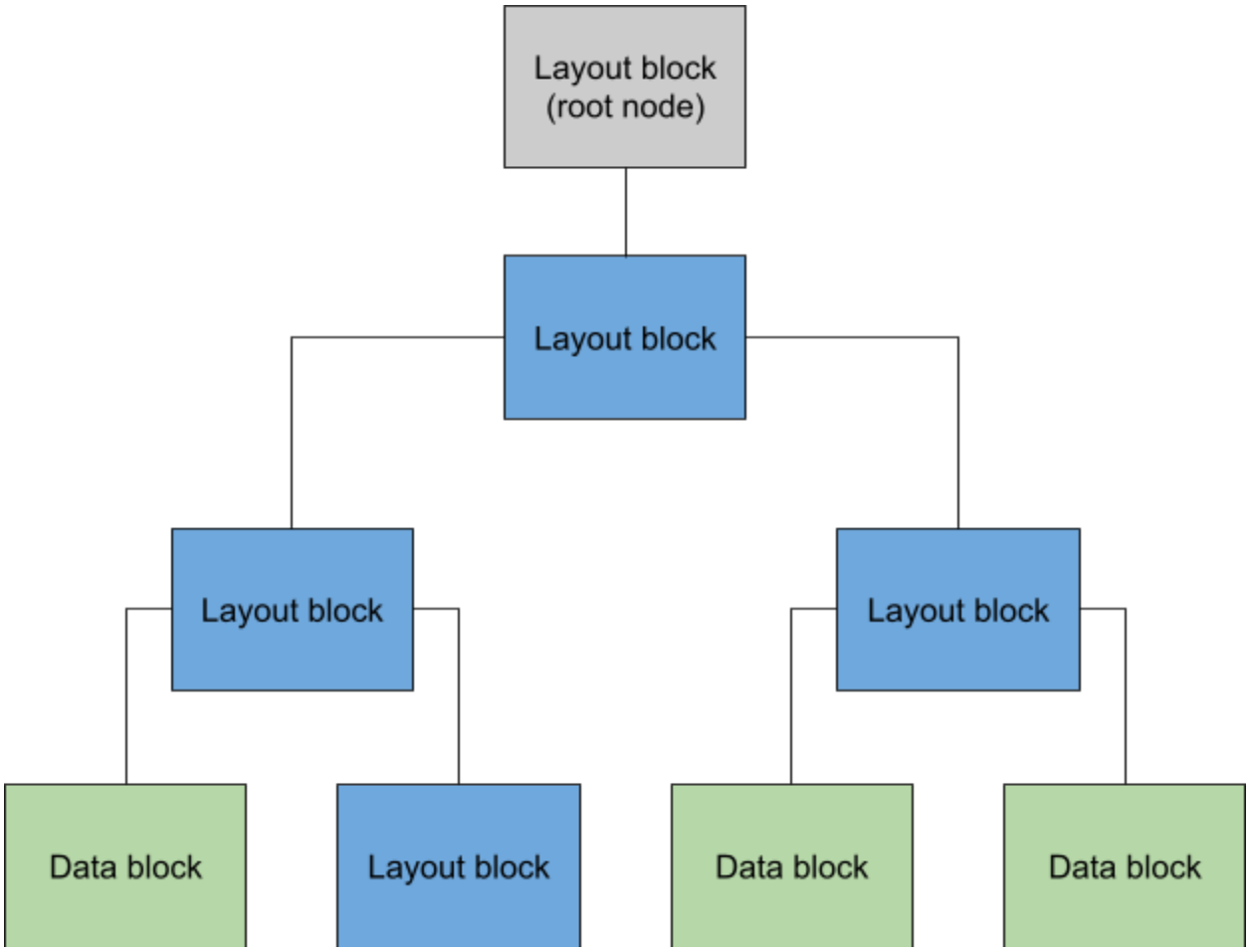
2) Data blocks

In the POC application there are two implementations of this block type:

a) “line text” block - simple line of text (<h> HTML tag)
b) “description” block (block with title and multiline markdown text)

These blocks display some candidate data in a generated CV.
These blocks cannot have children's blocks.
In other words - these blocks are always a leaf of the CV template tree.
Configuring various display options for these blocks is also available.

The CV template tree always has the layout block as the root block and may look like this:



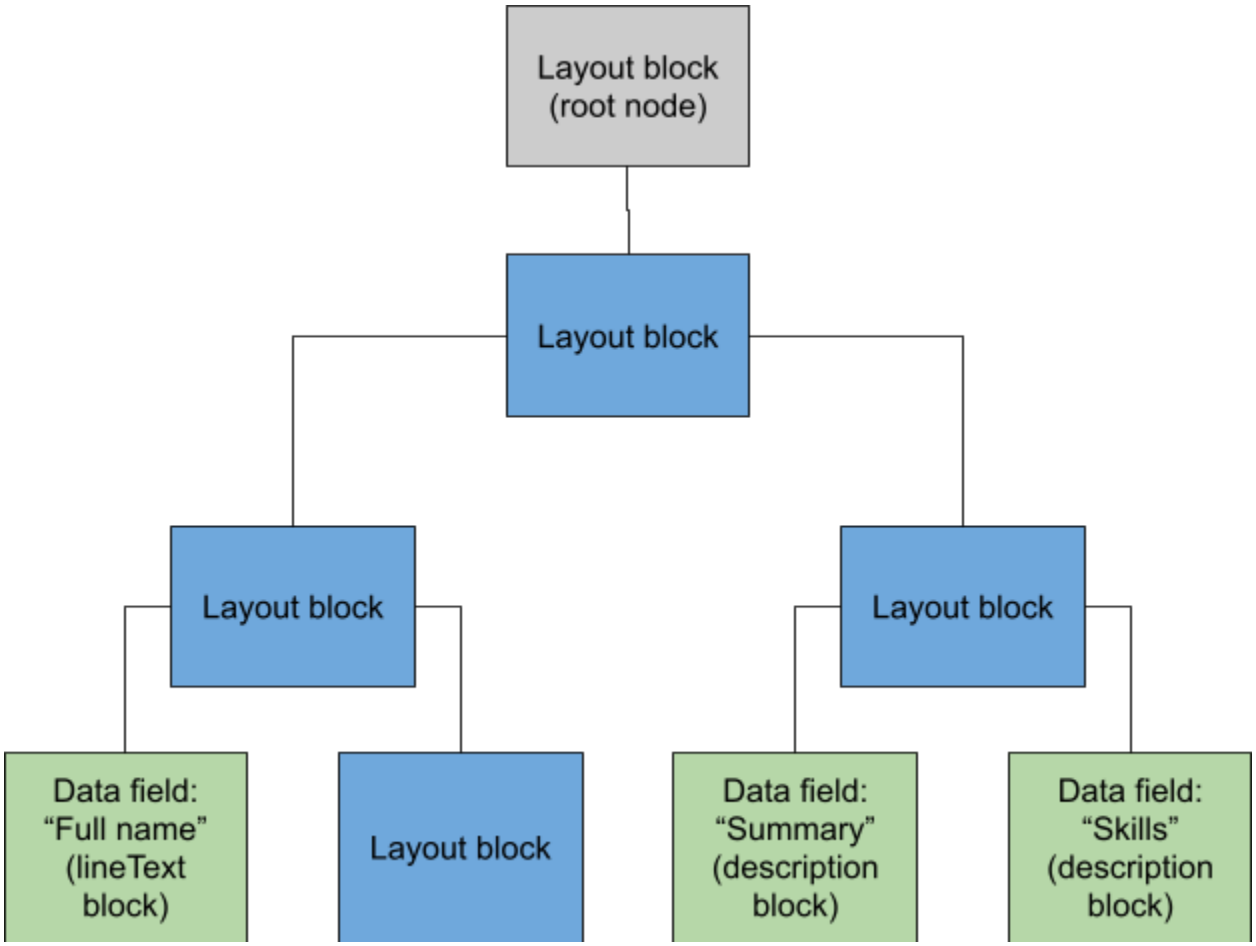
Data blocks (“DataField” in the code)

One data block can display various types of candidate data.
The “description” block can be used for displaying a candidate’s skills or a candidate’s summary, etc. To avoid the creation of a unique data block type for each candidate’s data, from data blocks user can create “data fields” (it’s called “data blocks in UI).
The “data field” is a data block with a name and a unique ID.
It also may contain some preview data to display on the preview CV template UI (general template without candidate data).
These “data fields” will help map a candidate's data to a CV template on the generate CV step.

CV template

As described above, the CV template is a tree.
But with “data field” leaves instead of two types of data blocks.

Therefore the final CV template may look like this:



Users can create as many CV templates (trees) as needed.

Candidate

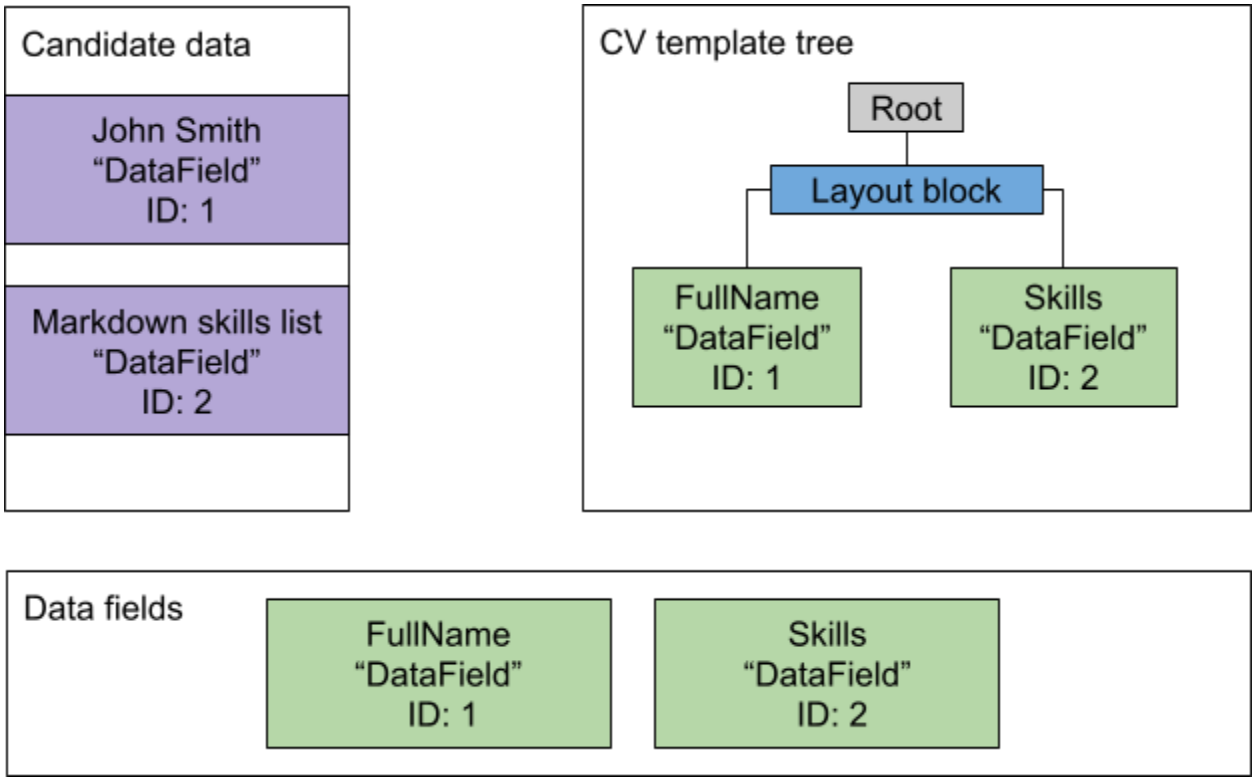
It’s convenient to collect and store candidate’s data separately and not related to any CV template. And apply any kind of CV template to this data without its modification to get the CV of this candidate.
Therefore, the candidates' data is collected and stored separately.
Parts of this data are “data field” blocks (e.g. candidate’s full name, summary, skills, etc.)
In other words, when we create a candidate’s data - we fill previously created “data field” blocks.

CV generation

After the creation of “data field” blocks, some CV templates, and fill a few candidate’s data, CVs for these candidates can be generated.
In the final step, the POC application fills a CV template with the candidate’s data.
It’s easy because a CV template tree has “data field” blocks and a candidate’s data is split into the same type of “data field” blocks (id’s of a tree’s “data field” blocks and id’s of a candidate’s

data “data field” blocks are same).
After filling the local CV template copy with a candidate’s data, it renders a separate page without additional UI controls with a CV template tree filled with the candidate’s data.
This page is ready to be printed to PDF by browser built-in tools.

Big picture



Before rendering the final CV page, the local copy CV template’s “data fields” is replaced with a candidate’s “data fields”.