UXifier - CV
Domain Specific Languages

Team ADSL2

BARNA Anthony
BURETTE Leo
DEFENDINI Lara
SAVORNIN Guillaume
VAN DER TUIJN Anton

ACADEMIC YEAR 2021-2022

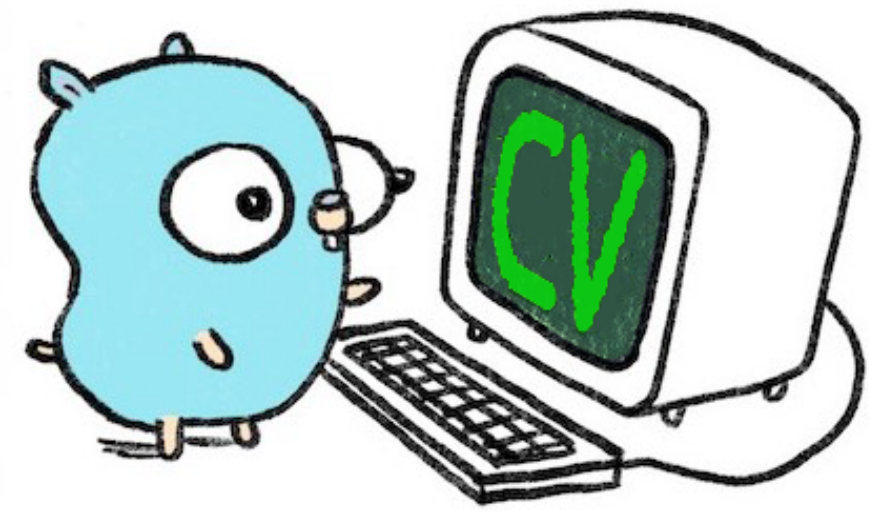

Figure 1: *The Gopher CV - Illustration image – non-contractual pictures (you are not a Gopher)*

Contents

The objective for this project is to define a Domain Specific Language (DSL) allowing the definition of a complex web app user interface. The language we decided to create focuses on the definition of a Curriculum Vitae website. We have determined the aspects of the domain model applicable to our subject, based on existing websites such as LinkedIn or doyoubuzz.

This document presents the work produced by our group for the DSL course. For this project we decided to implement a solution using an external DSL: JetBrains MPS.

The application generated by MPS is a React JavaScript file that uses the Material UI Library to get beautiful UI components.

The source code is hosted on GitHub at this address: https://github.com/GuillaumeSavornin/UXifier-Language.

1 Domain Model

Based on existing curriculum vitae websites, we defined the aspects of the domain model applicable to our subject. We deduced that a Curriculum Vitae is composed of multiple sections: Presentation, Additional information's, Skills, Experiences, Education, Activities, Contact, Social networks, Languages, Projects, and Image. Each of them has multiple common and unique characteristics.

We started the project by designing the Domain Model diagram to ensure that on the one hand our choices were consistent and, on the other hand, that we could share a reference document on the upcoming changes with the rest of the team.

On the following diagram we represented the different types of elements using colors:

- Yellow: Interfaces
- Blue: Classes
- Green: Enumerations

We didn't represent the *TextStyling* with arrows on the diagram, but directly in the list of attributes to keep the domain model readable for all the stylings. Those attributes are meant to be read as references.

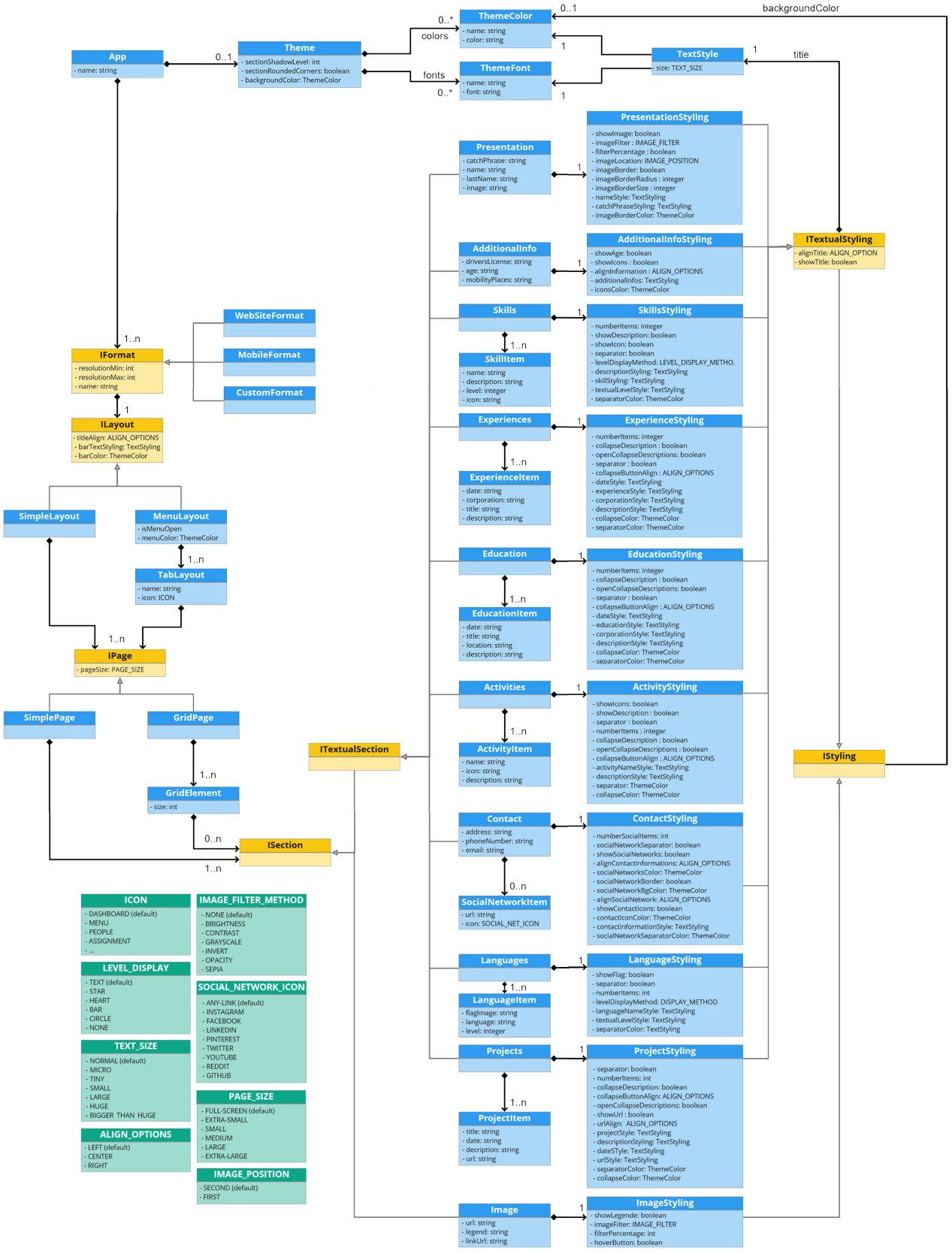

Figure 2: *Domain Model diagram*

1.1 Implementation and Domain Model description

In this section we will describe our Domain Model and how we implemented our language, by focusing on each component of the domain.

1.1.1 Formats

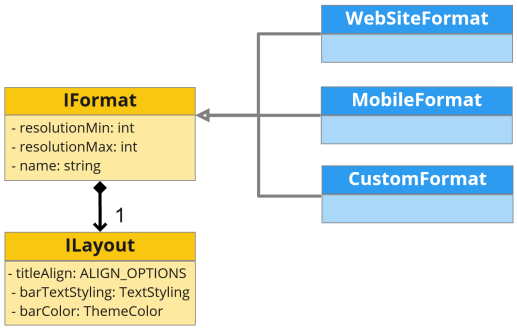

Figure 3: *Domain Model focus on Format*

The domain-specific language we developed allows you to write a CV for a variety of screen sizes; by default, the website and mobile formats are supported. The user can develop a variety of custom formats to handle a certain range of screen sizes defined by the user using pixels as the unit of measurement (min and max resolution). This makes it simple to design responsive curriculum vitae, with the option of employing different sections or stylistic attributes to get totally distinct displays between two formats.

1.1.2 Layouts

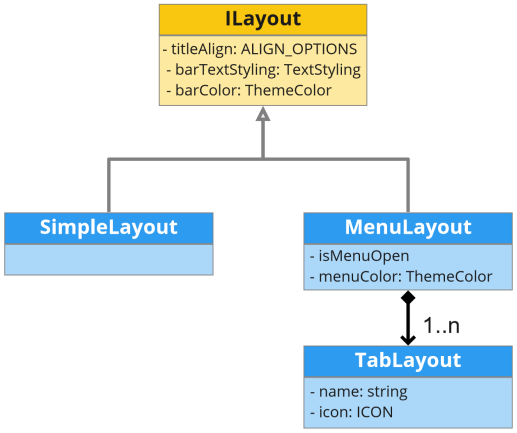

Figure 4: *Domain Model focus on Layout*

Each format is made up of a *SimpleLayout* or a *MenuLayout*, which is made up of a menu with tabs that allows you to travel between several *TabLayouts*. The tabs enable you to distribute your CV data across numerous tabs in order to decrease and arrange the data. You may also construct custom/categorize tabs, such as a photo gallery, by utilizing only the *ImageSection*. To avoid annoying customers with the menu, we recommend using the *SimpleLayout* for the mobile format. However, the *MenuLayout* can be used for the mobile format if that is the UX designer's preference.

1.1.3 Pages

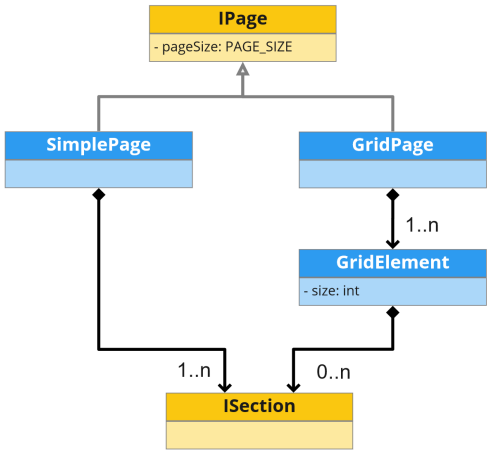

Figure 5: *Domain Model focus on Page*

Layouts are made up of one or more pages that are displayed one below the other, and they can be *SimplePages* or *GridPages*. *SimplePage* allows you to stack Sections one after the other with the same width, and you may choose from a variety of predefined sizes for your *SimplePage*. *GridLayout*, on the other hand, is more difficult to utilize since it is represented by a grid of 12 columns. The width of each *GridElement* may be defined by the UX designer when creating a *GridPage*. An error will be thrown if the total width of the *GridElement* exceeds 12. (See more warnings and errors in section 3.1. Constraints and Scope).

1.1.4 Sections

The sections represent the real content of the CV. They all contain a personalized *Styling* for their use. The styling is used to customize how the Section is going to look on the web page, that's why they have a lot of attributes.

Giving predetermined but configurable parts allows the UX Designer to focus on generating CV designs rather than having to create and position each small portion of each section. This would give the DSL the capacity to construct anything other than a CV, which is not what our DSL is for.

Presentation

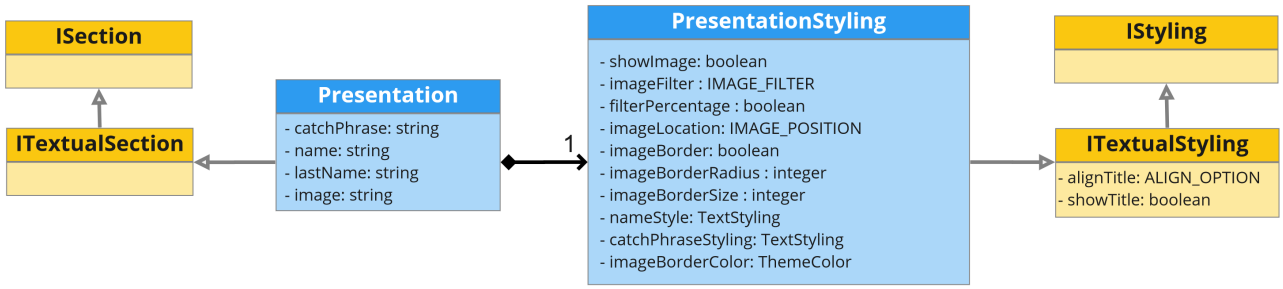

Figure 6: *Domain Model focus on Presentation*

The *Presentation* section aims to provide basic information about the person along with a picture. The design allows you to position the image within the presentation while also hiding it to support other standards, such as in North America or the United Kingdom, where it could be considered discriminatory.

Additional Information

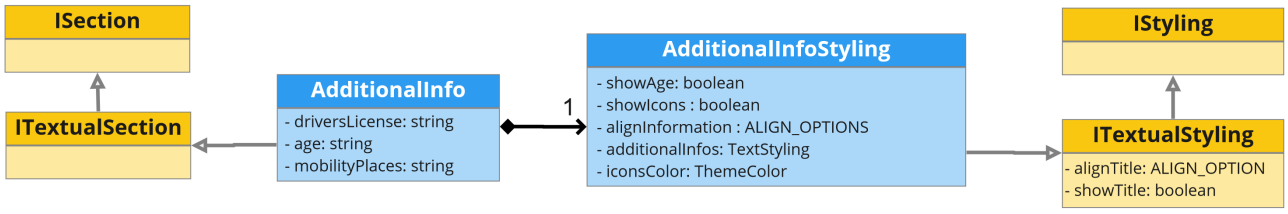

Figure 7: *Domain Model focus on Additional Information*

AdditionalInfo completes the presentation section by adding the age (which can be disable if you want), your driving license and mobility places. You can show the social medias icons too.

Skills

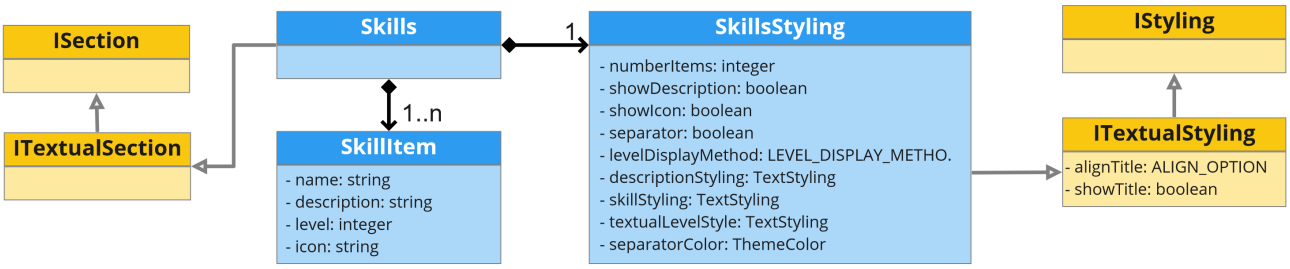

Figure 8: *Domain Model focus on Skills*

Skills are displayed as a list with an optional icon and description, in addition it allows to rate the skill level. Multiple graphical solutions are offered to rate such as stars, text, hearts, circles, or a bar.

Experience and Education

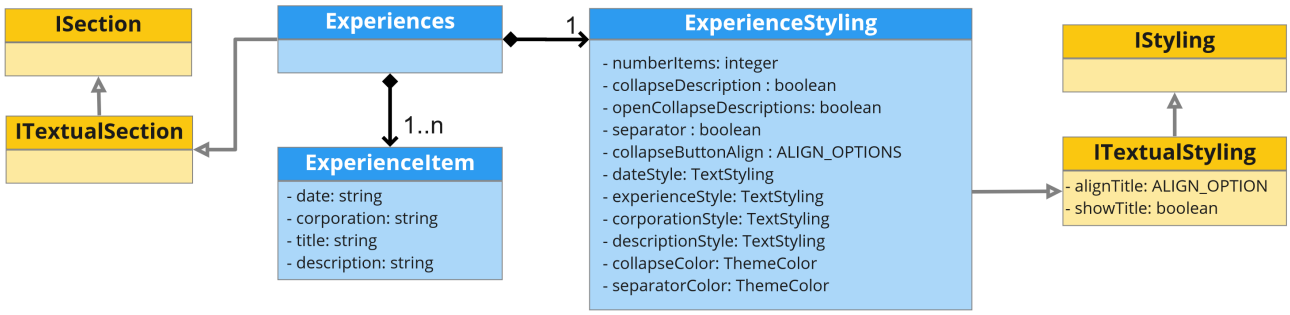

Figure 9: *Domain Model focus on Experiences*

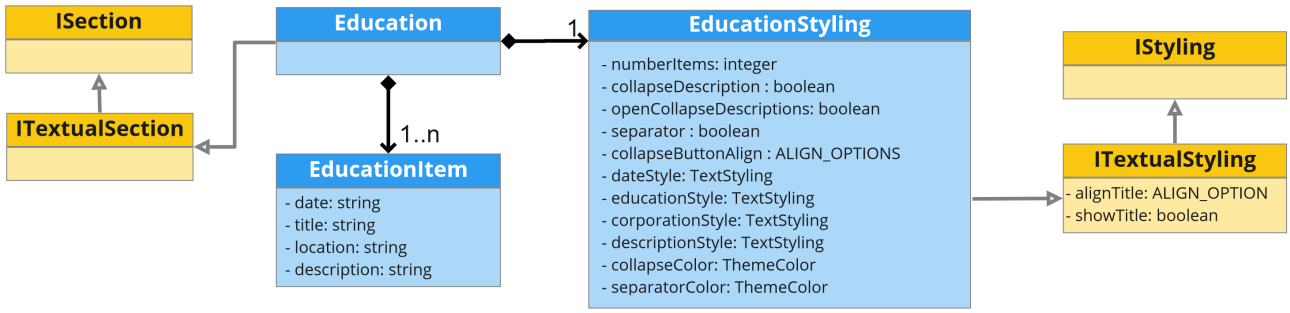

Figure 10: *Domain Model focus on Education*

These two sections have a similar styling because their representation is the same. We split into two specific sections to be closest possible to the Domain model. *ALIGN_OPTIONS* can be placed on *LEFT*, *RIGHT* or *CENTER*. We can customize each text part with the *TextStyling* component, and we can give the number of items and if we want to *collapseDescription* (with the default behavior) and the separator.

Activities

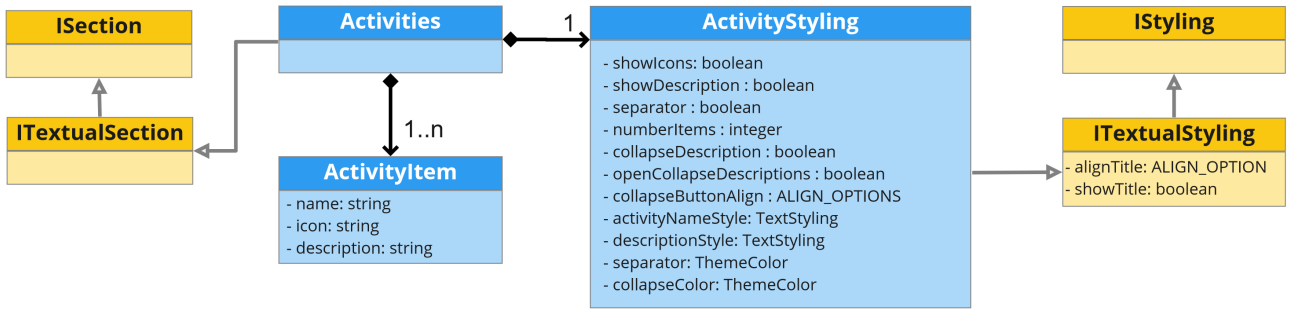

Figure 11: *Domain Model focus on Activity*

This part allows the user to give information about his passion, hobby, or activities in general. You can add a description and an icon to customize the part. Like on the above parts, we can customize each text part with the *TextStyling* component, and we can give the number of items and if we want to *collapseDescription* (with the default behavior) and the separator.

Contacts

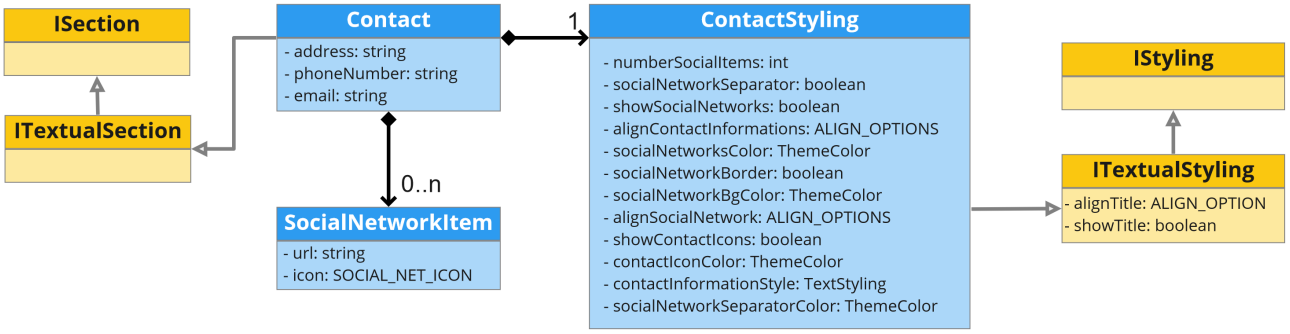

Figure 12: *Domain Model focus on Contact*

Three types of personal data are displayed in the *Contact* section, personal address, phone number and email address, all trees are mandatory. In option we can add social networks represented by an icon that can be colored. At the moment, 8 social networks icons can be automatically generated, with a default one.

Languages

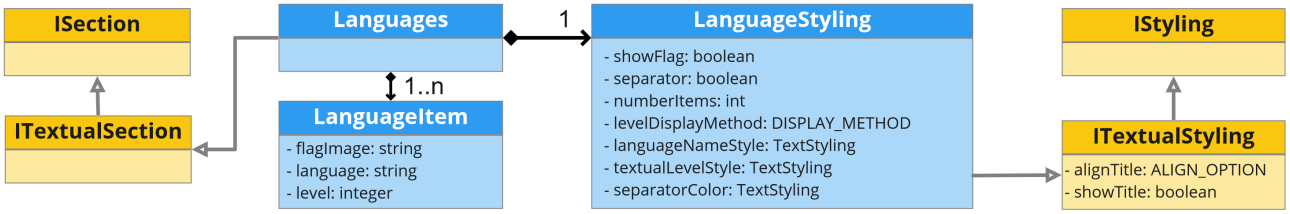

Figure 13: *Domain Model focus on Language*

Similarly to the *Skills* section, the *Language* section is displayed as a list with an optional flag and a rating level. The same graphical solutions available for the skill can be used for the language section.

Projects

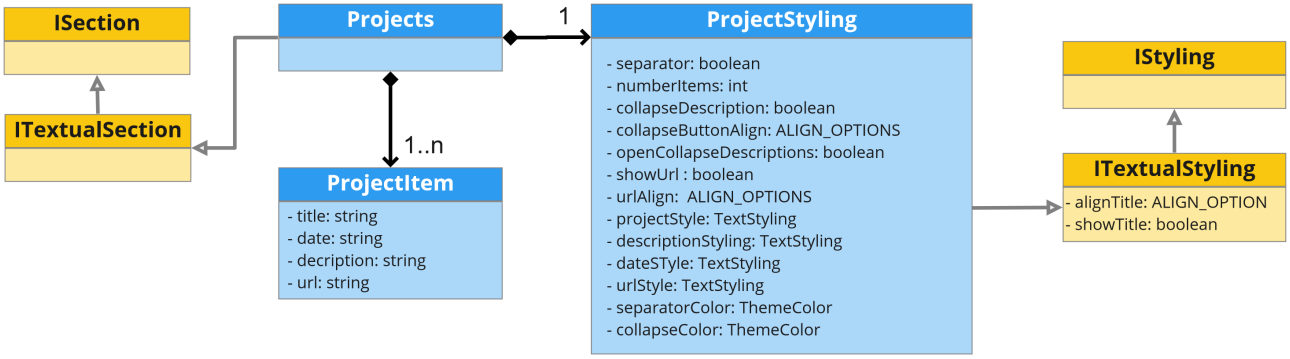

Figure 14: *Domain Model focus on Project*

The *Project* section is designed to display projects alongside an URL so that you can easily see more of it. The number of presented projects can directly be set in the DSL.

Images

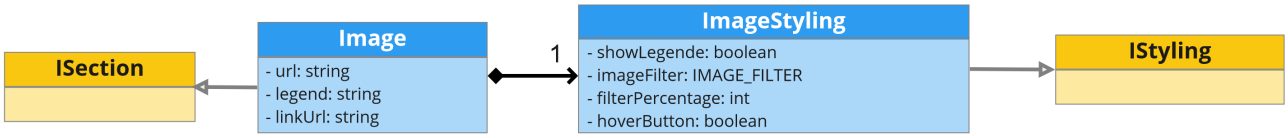

Figure 15: *Domain Model focus on Image*

This section has an URL to find the image, a legend and a *linkUrl*. The *showLegend* styling displays the legend, *imageFilter* and *filterPercentage* can add a filter to the image among: *BRIGHTNESS*, *CONTRAST*, *GRAYSCALE*, *INVERT*, *OPACITY* and *SEPIA*. The *hoverButton* option adds a button on the image and adds a blur effect. The button redirects to the *linkUrl*.

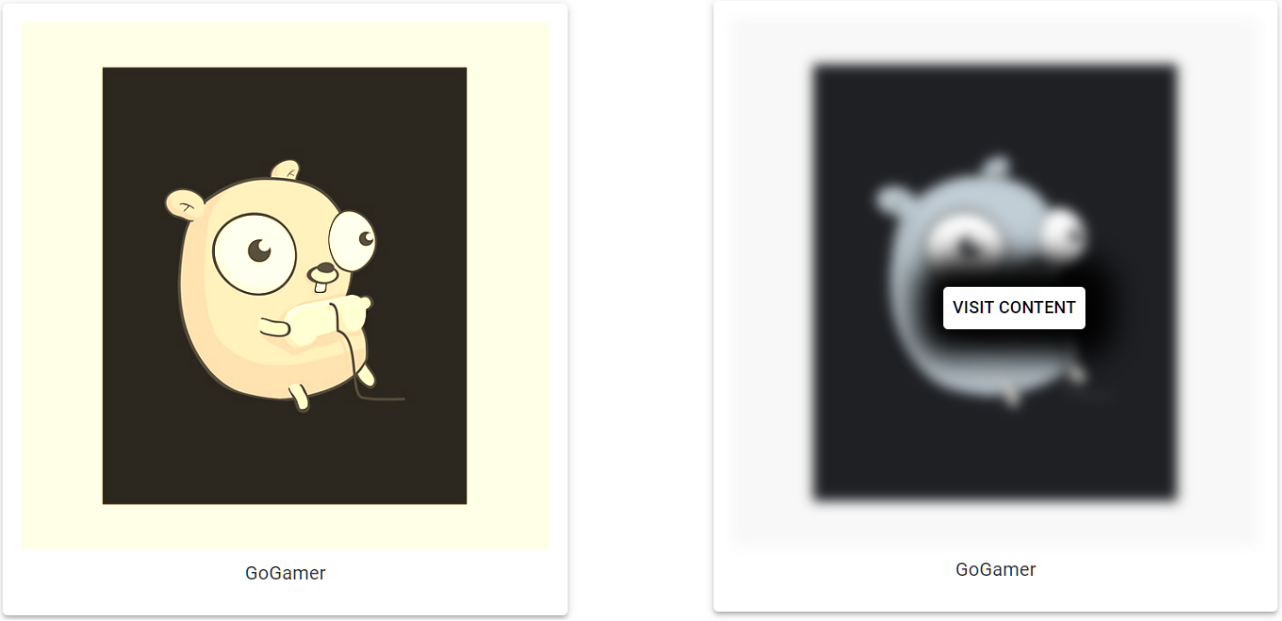

Figure 16: *Example of hoverButton. On the right when the mouse is hover.*

1.1.5 Themes

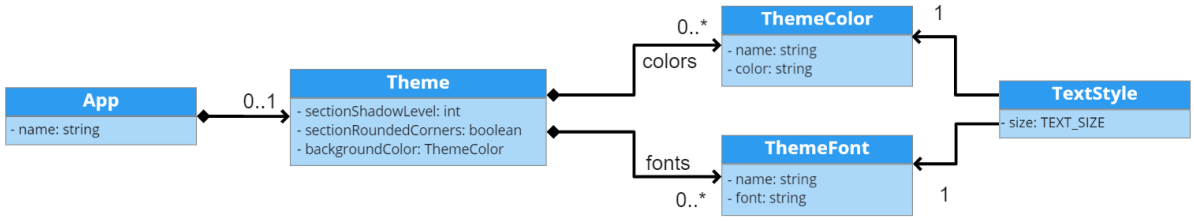

Figure 17: *Domain Model focus on Theme*

Our domain specific language features a *Theme*, defining global styling properties such as shadow level, section zone corners style and background color. Those properties will be applied across all the *Formats* to each *Section*.

The Theme also allows you to create a list of colors and fonts that may be used to color and style *TextStyle* and other components. *TextStyle* also offers the possibility to define the text size. *TextStyle* style is applied to all strings in our language, allowing the UX designer to tweak them as he wishes. The *ThemeColor* stylizing feature may be used by the designer to customize most of the the UI's icons and colorable components.

1.2 Text generation

The Application structure is at the top of our domain model and is in charge of generating the JavaScript file in MPS. We create the base of the React application using a text generator for the App, referencing the HTML file and adding all the imports.
As a result, we build a function for each *Format* based on what the expert has defined in the DSL: mobile, website, or a variety of custom formats. Depending on the width of the page, a single format represented by a function is then invoked when rendering the page.
The function generated for each format provides practically all the required data; just the theme color and font variables are referenced. That means, if we have a Presentation section that is the same in the Mobile and Website format, it will create the code for it once in each of the function representing the formats.
The *Theme* creates an object at the root of the JavaScript file that is then referenced in other styling structure to get access to the colors and fonts. Those variables' value could have been directly printed in the code without being referenced, but it's easier for the expert.

1.3 JSON Loader

To have a usable final product, we made a pattern that we called *JSONLoader*. The idea is to check if we have data in the **data.json** file next to out generated JavaScript code. If we have data with corresponding label, we load this in our page. Otherwise we use the default value defined in MPS behaviors files.

```
{
  (()=>{
        if(data." + section + "){
             return data." + section + "." + attribut + "
        }
        return " + defaultValue + "
  })()
}
```

Thanks to this function, we can generate a CV with default value and the final user of the CV can replace the placeholders with his data using the **data.json** file.

2 DSL Syntax

2.1 Extended Backus–Naur Form

```
<Presentation> = "Presentation section:" <PresentationStyling>;
<AdditionalInfo> = "Additional Information Section:" <AdditionalInfoStyling>;
<Skills> = "Skills Section:" <SkillsStyling>;
<Experiences> = "Experiences Section:" <ExperiencesStyling>;
<Education> = "Education Section:" < Education Styling>;
<Activities>   = "Activities Section:" < ActivitiesStyling>;
<Contact> = "Contact Section:" < ContactStyling>;
<Languages> = "Languages Section:" < LanguagesStyling>;
<Projects> = "Image Section:" <ImageStyling>;
<Image> = "Additional Information Section:" <AdditionalInfoStyling>;
<TextualSection> = <Presentation> | <AdditionalInfo> | <Skills> | <Experiences> | <Education>
    | <Activities> | <Contact> | <Languages> | <Projects>;
<Section> = <TextualSection> | <Image>;

<PresentationStyling> =
    "show title:" <Boolean> <String>
        "align title:" <ALIGN_TITLE>
    "background color:" <String>
    "name style:" <String>
    "catch phrase style:" <String>
    "show image:" <Boolean>
        "image position:" <IMAGE_LOCATION>
        "image filter:" <IMAGE_FILTER>
            "filter percentage:" <Integer>
        "image border:" <Boolean>
            "border color:" <String>
            "border size (px):" <Integer>
            "border radius (px):" <Integer>;
<AdditionalInfoStyling> =
    "show title:" <Boolean> <String>
        "alignment:" <ALIGN_TITLE>
    "background color:" <String>
    "additional informations style:" <TextStyling>
    "show icons:" <Boolean> "[C]:" <String>
    "show age:" <Boolean>
    "align additional information:" <ALIGN_TITLE>;
<SkillsStyling> =
    "show title:" <Boolean> <String>
        "alignment:" <ALIGN_TITLE>
    "background color:" <String>
    "skill name style:" <TextStyling>
    "show icon:" <Boolean>
    "show description:" <Boolean> <TextStyling>
    "show separator:" <Boolean> "[C]:" <String>
    "level display method:" <Integer> <TextStyling>
    "number of skills:" <Integer>;
<ExperiencesStyling> =
    "show title:" <Boolean> <String>
        "alignment:" <ALIGN_TITLE>
    "background color:" <TextStyling>
    "experience name style:" <TextStyling>
    "date style:" <TextStyling>
    "corporation name style:" <TextStyling>
    "description style:" <TextStyling>
    "collapse description:" <Boolean> "[C]:" <String>
        "open all:" <Boolean>
        "button alignment:" <ALIGN_OPTIONS>
    "show separator:" <Boolean>  "[C]:" <String>
    "number of experience:" <Integer>;
<EducationStyling> =
    "show title:" <Boolean> <String>
        "alignment:" <ALIGN_TITLE>
    "background color:" <TextStyling>
    "education name style:" <TextStyling>
    "date style:" <TextStyling>
```

```
    "school name style:" <TextStyling>
    "description style:" <TextStyling>
    "collapse description:" <Boolean> "[C]:" <String>
        "open all:" <Boolean>
        "button alignment:" <ALIGN_OPTIONS>
    "show separator:" <Boolean> "[C]:" <String>
    "number of educations:" <Integer>;
<ActivitiesStyling> =
    "show title:" <Boolean> <String>
        "alignment:" <ALIGN_TITLE>
    "background color:" <TextStyling>
    "education name style:" <TextStyling>
    "date style:" <TextStyling>
    "school name style:" <TextStyling>
    "description style:" <TextStyling>
    "collapse description:" <Boolean> "[C]:" <String>
        "open all:" <Boolean>
        "button alignment:" <ALIGN_OPTIONS>
    "show separator:" <Boolean>  "[C]:" <String>
    "number of activities:" <Integer>;
<ContactStyling> =
    "show title:" <Boolean> <String>
        "alignment:" <ALIGN_TITLE>
    "background color:" <TextStyling>
    "contact element style:" <TextStyling>
    "align contact information:" <ALIGN_TITLE>
    "show contact icons:" <Boolean> "[C]:" <String>
    "show social networks:" <Boolean>
        "icon color:" <String>
        "background color:" <String>
        "border:" <Boolean>
        "alignment:" <ALIGN_TITLE>
        "show separator:" <Boolean> "[C]:" <String>
        "number of social networks:" <Integer>;
<LanguagesStyling> =
    "show title:" <Boolean> <TextStyling>
        "alignment:" <ALIGN_OPTIONS>
    "background color:" <String>
    "language name style:" <TextStyling>
    "show flags:" <Boolean>
    "show separator:" <Boolean> "[C]:" <String>
    "level display method:" <LEVEL_DISPLAY_METHOD>
    "number of languages:" <Integer>;
<ProjectsStyling> =
    "show title:" <Boolean> <TextStyling>
        "alignment:" <ALIGN_OPTIONS>
    "background color:" <String>
    "project name style:" <TextStyling>
    "date style:" <TextStyling>
    "description style:" <TextStyling>
    "collapse description:" <Boolean> "[C]:" <String>
        "open all:" <Boolean>
        "button alignment:" <ALIGN_OPTIONS>
    "show button  go  to project :" <Boolean> <TextStyling>
        "alignment:" <ALIGN_OPTIONS>
    "show separator:" <Boolean> "[C]:" <String>
    "number of projects:" <Integer>;
<ImageStyling> =
    "background color:" <String>
    "show legend:" <Boolean> <TextStyling>
    "hover image button:" <Boolean>
    "align additional informations:" <ALIGN_OPTIONS>;
<TextStyling> = <PresentationStyling> | <AdditionalInfoStyling> | <SkillsStyling> | <
    ExperiencesStyling> | <EducationStyling> | <ActivitiesStyling> | <ContactStyling> | <
    LanguagesStyling> | <ProjectsStyling>;
<Styling> = <TextualStyling> | <ImageStyling>;

<GridElements> =
    "Size" <Integer> "/12"
```

```
    (<Section>)*;
<GridPage> =
    "Grid page:    width:" <Integer>
        (<GridElements>)+;
<SimplePage> =
    "Simple page:    width:" <Integer>
    (<Section>)+;
<Page> = <GridPage> | <SimplePage>;

<TabLayout> =
    "Tab "<String>     "icon:" <Icon>
    (<Page>)+;
<MenuLayout> =
    "Menu layout:    menu open:" <Boolean>
        "style:"
            "top bar:"
                "color:" <String>
                "text:" <TextStyling>
                "alignment:" <ALIGN_OPTIONS>
            "menu:" <String>
        (<TabLayout>)+;
<SimpleLayout> =
    "Layout:"
        "style:"
            "color:" <String>
            "text:" <TextStyling>
            "alignment:" <ALIGN_OPTIONS>
        (<Page>)+;
<Layout> = < MenuLayout> | <SimpleLayout>;
<WebSiteFormat> = "Website format:" <Layout>;
<MobileFormat> = "Mobile format:" <Layout>;
<CustomFormat> = <String> "format: resolution range: [" <Integer> "," <Integer> "]" <Layout>;
<Format> = <WebSiteFormat> | <MobileFormat> | <CustomFormat>;
<ThemeColor> = "name:" <String> "color:" <String>;
<ThemeFont> = "name:" <String> "font:" <String>;
<TextStyling> = "[F]:" <String> "[C]:" <String> "[S]:" < String>;
<Theme> =
    "colors:" (<ThemeColor>)*
    "fonts:" (<ThemeFont>)*
   "section:"
        "shadow level:" <Integer>
        "rounded corners" <Boolean>
    "background color:" <ThemeColor>;
<App> =
    "Application" <String>
        "Theme": (<Theme>)?
        (<Format>)+;

<ALIGN_OPTIONS> = "LEFT" | "CENTER" | "RIGHT";
<ICON> = "DASHBOARD" | "MENU" | "PEOPLE" | "ASSIGNMENT" | "ARROW-RIGHT-1" | "ARROW-RIGHT-2" |
    "FACE" | "FILE" | "FLOWER" | "PICTURE" | "SPORT" | "WORK" | "STAR";
<IMAGE_FILTER> = "NONE" | "BRIGHTNESS" | "CONTRAST" | "GRAYSCALE" | "INVERT" | "OPACITY" | "
    SEPIA";
<IMAGE_POSITION> = "SECOND" | "FIRST";
<LEVEL_DISPLAY_METHOD> = "TEXT" | "STAR" | "HEART" | "BAR" | "CIRCLE" | "NONE";
<PAGE_SIZE> = "FULL-SCREEN" | "EXTRA-SMALL" | "SMALL" | "MEDIUM" | "LARGE" | "EXTRA-LARGE";
<SOCIAL_NETWORK_ICON> = "ANY-LINK" | "INSTRAGRAM" | "FACEBOOK" | "LINKEDIN" | "PINTEREST" | "
    TWITTER" | "YOUTUBE" | "REDDIT" | "GITHUB";
<TEXT_SIZE> = "NORMAL" | "MICRO" | "TINY" | "SMALL" | "LARGE" | "HUGE" | "BIGGER_THAN_HUGE";
```

2.2 Syntax example

Figure 18: *Global Theme definition*

The syntax to define a *Theme* is the first one to be defined, as shown on the above example we defined multiple colors and fonts. When the reference name of the font is equals to the font they are collapsed into a single value.

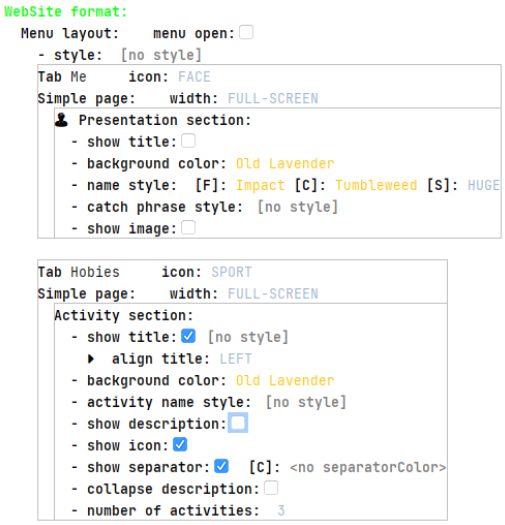

Figure 19: *Multiple tab layout*

Menu layout offers the possibility to create tabs, each tab can then be composed by multiple sections. In the script the tabs are displayed in a vertical list, one below the other.

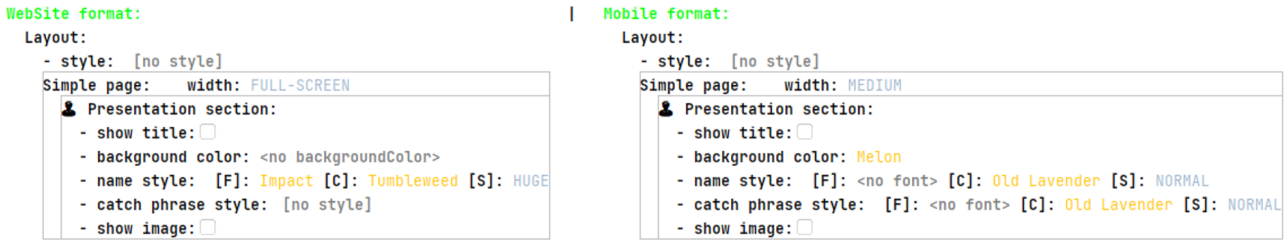

Figure 20: *Website and mobile format side by side*

When the UX designer defines multiple formats, they are displayed side by side. If a format takes too much space it will be displayed on a new line.

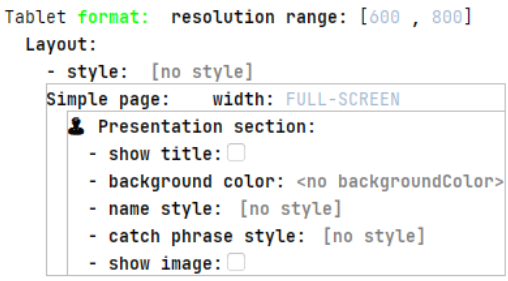

Figure 21: *Custom format*

Custom format resolution is expressed in pixels. When the user screen resolution enters the defined range, it will display the formats.

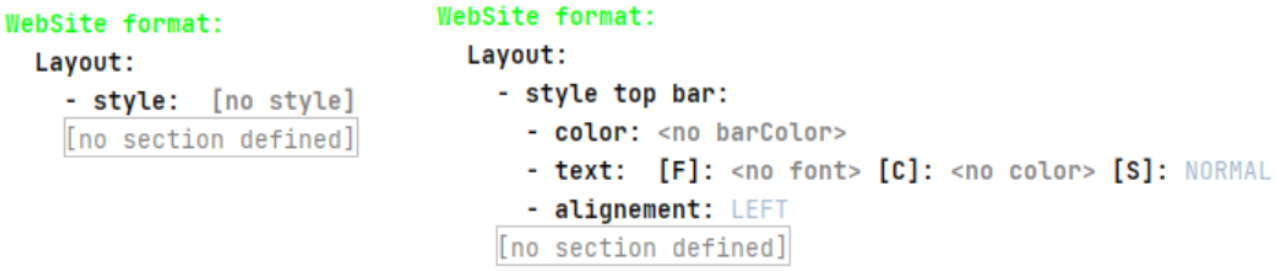

Figure 22: *Left: collapsed properties – Right: displayed properties*

On the left side we have a collapsed property, because no style wants to be defined only one line is displayed. However, if the user wants a style, it will display the style properties.

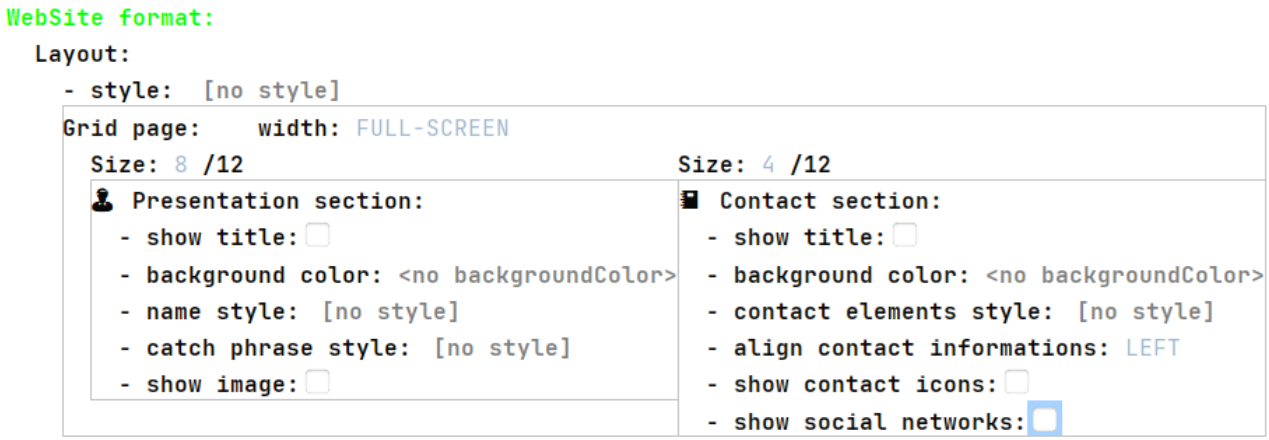

Figure 23: *Grid layout*

Above, you have an example of grid layout with two groups, the first one with a size of 8/12 and the second one a size of 4/12. They are displayed side by side.

3 Constraints and Scope

In MPS, all scripts share the same scope in case we want to reference attributes between them. To avoid the expert referencing a *ThemeColor* and *ThemeFont* from another CV, we limited the scope to the current CV. This can help in case the expert wants to name a color "primary" in all its CVs and that they have different color values.

We added constraints to complement the domain model limitations in a more advanced way. They help the expert to create the CV's UI without making mistakes that can be prevented.

There are some generic constraints on the values that can be entered like the *ThemeColor* that checks that the values entered are hexadecimal colors (6 or 8 digits).

The *GridPage* is separated into 12 sections, if the expert has multiple *GridElement* that sums below 12, then it displays a warning that the whole grid isn't filled and that there will be some blank spaces. This will help the expert to avoid simple mistakes, but not making this an error can still allow the expert to create innovative CVs.

Moreover, the *GridPage* could potentially be filled with a single *GridElement* of size 12, it's equivalent to creating a *SimplePage*. To inform the expert we added a warning and a "quick fix" that pops up to the expert in MPS with the shortcut *Alt+Enter*. This quick fix will replace the *GridPage* node with a *SimplePage* node while keeping all the sections that it contains.

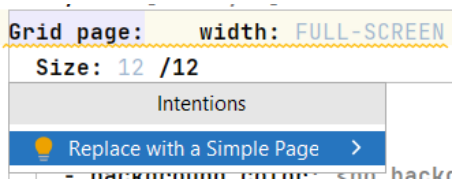

Figure 24: *Quick fix: 12/12 grid to simple page*

The expert can create multiple types of layouts, but only a single instance of the Mobile and Website layout. If the expert creates two of them, we display an error and add a quick fix that will change the wrong layout with a *Custom Layout* that will make sense domain wise.

Figure 25: *Quick fix duplicate Mobile or Website layout*

The *CustomLayout*, is configured through a range of pixels at which the Layout is displayed on the screen. We display an error in case multiple custom layouts are created that overlap each other. It can prevent unwanted behavior on the web page on the overlapping screen sizes.

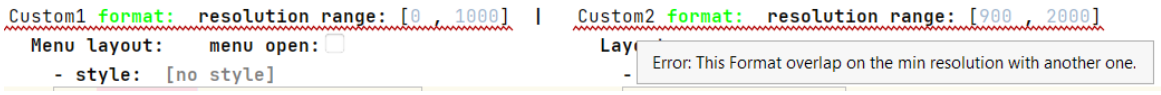

Figure 26: *Format overlapping*

A warning is added to *Sections* if it detects that there is more than one *Section* of each type in a layout. It alerts the expert that he has some duplicate sections, and he might want to delete one of them or move it to another Layout.

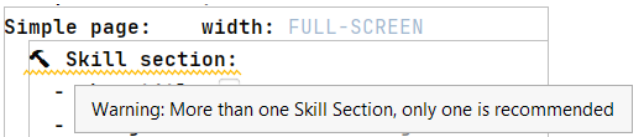

Figure 27: *Duplicate section*

Finally, another warning is displayed when the CV has a menu that contains more than 6 tabs, that could mean the expert has tabs without that much data in it that would lead the user to change tabs too much to see the whole content. So, we recommend the expert to group some tabs in case this happens.

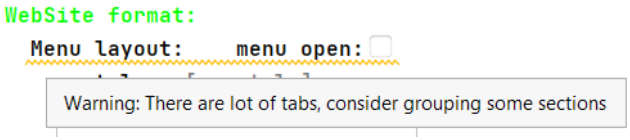

Figure 28: *Too many tabs*

4 Scenarios

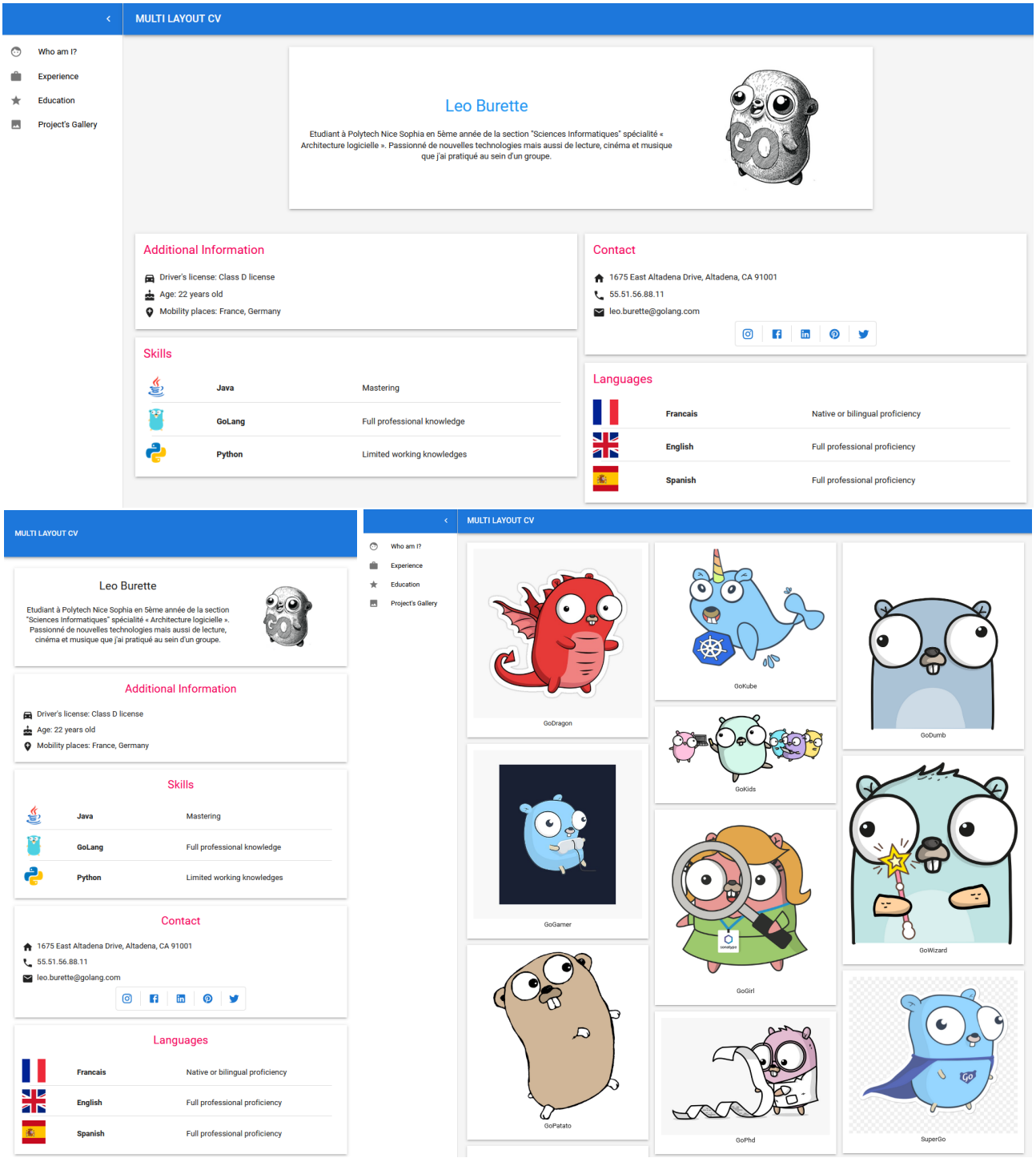

Figure 29: *Pictures of CV example 1*

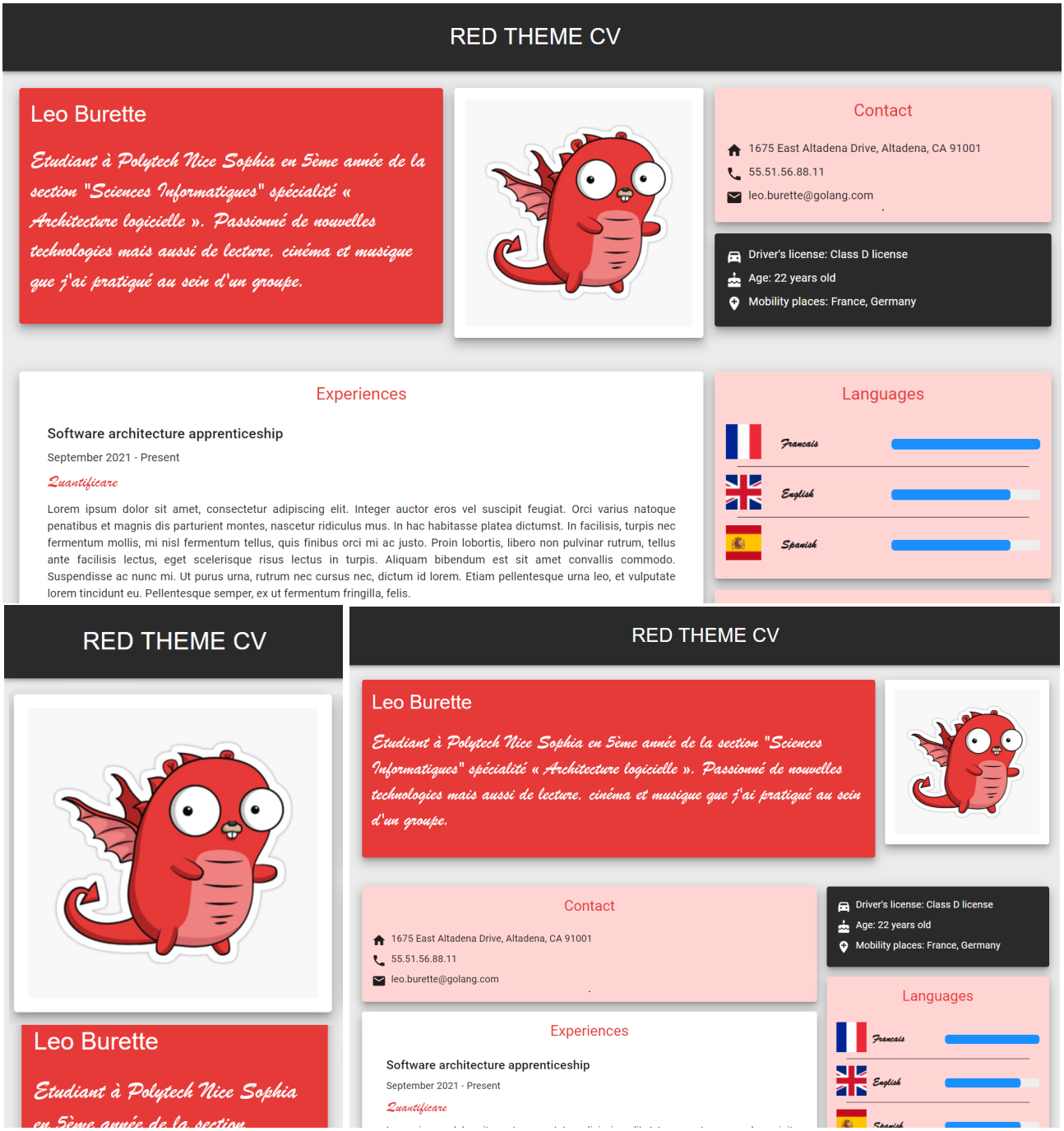

Figure 30: *Pictures of CV example 2*

5 Retrospective

When looking at the Domain Model, one could think that we could have done an abstraction of some of our Sections. We chose not to do it because it was simpler, and the code of the text generator is cleaner and more readable this way. It avoided a lot of confusion and mistakes. It also allowed us to better define the model and be closer to the business, because each section that the user has to use is clearly listed.

One issue that we had while developing is having to preview the generated **index.js** file from the CV's DSL to then copy its content to the react app so that it regenerates the web page. MPS documentation explains that the file can be generated in a custom folder, but it didn't work in our case. A potential improvement we could have made is to add a file watcher that can copy the file to the correct location, helping the expert create CVs more fluently.

When designing our Domain Model, we faced the choice of letting the user the ability to reuse multiple times section he creates and configure. We chose against giving the user this option since we believed he would be more likely to build the views differently and not reuse sections. However, the cost to implement this feature would be rather low. To do this, we should allow users to create identifiers for each section and then refer to them when the section needs to be reused.

To go further, some improvements could have been made to the descriptions styling characteristics. Descriptions are utilized in numerous sections; currently, the description can be collapsed or not. Nevertheless, because it collapses the entire description, this alternative is limited. We would like to add the ability to display portions of the description, such as the first two lines of the description and the remainder the rest collapsed. Because it is merely a style element, it will have a minor impact on our Domain Model, and we will be able to implement it with more time.

We decided to use the MPS tool to create our language since it gives a lot of guidance. The most significant drawback is that a language created with MPS is extremely reliant on the IDE. Because it cannot be opened by any other IDE, the language becomes far less practical without it. Constraints, on the other hand, are simple to create, and warnings and error checks are presented in real time to assist the user. The syntax coloring, which may be substantially adjusted, is another feature that makes MPS an excellent alternative to the other tools.

6 Work repartition

Anthony Barna:

- Experience section and Education section

Leo Burette:

- Presentation section, Image section and Json data loaded

Lara Defendini:

- Additional information, Activities section and convert the report to Latex

Guillaume Savornin:

- Pages, Layouts, Formats and Constraints

Anton van der Tuijn:

- Theme, Skill section, Contact section, Language section, Project section

Appendices

A Convention used to write our eBNF

=	definition
;	termination
|	alternation
(...)	grouping
" ... "	terminal string
(...)*	zero or more
(...)+	one or more
(...)?	zero or one
<...>	concept