COMSM0104: Web Technologies 2019
Final Assignment Report

Tao Xu
si19010@bristol.ac.uk

Yinan Yang
ff19085@bristol.ac.uk

June 8, 2020

Abstract

Our team consists of Tao Xu (si19010) and Yinan Yang (ff19085). Due to the impact of Covid19, we collaboarated remotely via GitHub to co-develop this project.

Our website is an online CV maker, featuring a simple and convenient interface for editing, online storage (at our server) and extensibility (the CV templates are easy to make).

The frontend of our website was based on the Vue framework, taking advantage of Vue's MVVM, the Model-View-View Model, which helped us in keeping code modular and implementing reactive user interfaces.

The backend of our website was powered by Node.js with Express.js and SQLite.

Keywords: *Vue, Node-Js, SQLite.*

Contents

1 Introduction

The website we created, named ″Simple Resume Maker″, was an online CV maker, where users could employ the CV templates provided by us to make tailored resumes of their own.

Basic editing features like text editing, adding/removing pages, inserting/deleting sections, italicising/boldifing text, increasing/decreasing font size, and uploading avatar have been implemented in the frontend. Users could also save and load their progress and generate their CVs in pdf format, these features are powered by the backend.

In short, the idea behind our CV-maker was that the contents of a CV was just the plain html in the container enclosing the CV pages, and how the CV was displayed were only affected by the CSS code applied, which were present in the ″<head>″ section of the document. Therefore, when saving and loading the progress, the ″<head>″ and the ″<div>″ containing the CV contents were uploaded to and downloaded from our server, respectively. Applying different templates was actually just replacing one CSS file that was being used with another, which also made our templates quite easy to make since they were just CSS files. When a user requests to download a pdf version of their CV, the backend loads their data from the database and creates a temporary html file that contains all the information needed to regenerate the CV pages that the user sees. This temporary html is then loaded by the ″puppeteer″ module at our server, which is a headless Chrome browser. So the temporary html page should appear exactly the same as the related section of the page that the user sees (if the user is using a Chrome browser or other browsers that have the same behaviour as chrome). The ″puppeteer″ module then convert the page into pdf, which will be related by our server to the user.

Logging in is required for users to access any information regarding CV contents stored at the backend, including avatars and text. After logging in, each user is assigned an encryped JSON web token (JWT) containing their user Id. This JWT is stored as a Cookie so that it will be present in every request to the server from the user thereafter. Our server identifies users by verifying the JWTs and reading the user Id information from the JWTs. Therefore, as long as a user do not leak the JWT, no one else could access their data, which protects the users′ privacy to some extent.

Although our website does not provides extensive functionalities regarding text editing compared to mainstream text editors, our aim was to give users experience of editing documents online. Moreover, what would otherwise be a cumbersome formatting process was made easier with employing different CSS code. This addresses the initial point we made in designing the product, which was to make things easier.

In building this site, we used the VUE framework, which is a progressive framework for building user interfaces.

> Unlike other monolithic frameworks, Vue is designed from the ground up to be incrementally adoptable. The core library is focused on the view layer only, and is easy to pick up and integrate with other libraries or existing projects. On the other hand, Vue is also perfectly capable of powering sophisticated Single-Page Applications when used in combination with modern tooling and supporting libraries.
> —— Official development documentation from Vue

1.1 Set up the environment

```
npm install
```

1.2 Compile the frontend

```
npm run build
```

1.3 Run the server

- localhost https (recommended):
  ```
  node server.js
  ```
- cloud https:
  ```
  node server.js -cloud
  ```
- http only:
  ```
  node server.js -http
  ```
- query the database:
  ```
  node server.js -sql
  ```

1.4 NOTE

For the server-side node module "puppeteer" to function, you may need to install some dependencies if you don't have: Puppeteer dependencies. Otherwise, the pdf generator may not work, although it won't crash the server.

2 Self Evaluation

2.1 Estimation of marks

- A+ for HTML
- A for CSS
- A for JS
- A for PNG
- A for SVG
- A for Server
- A for Database
- A for Dynamic pages

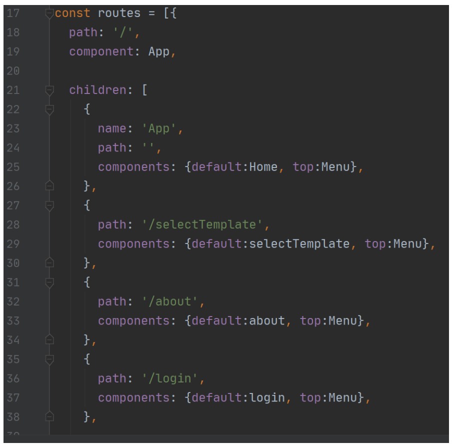
2.2 Client Side

2.2.1 HTML

- We have been quite proficient in generating HTML pages via the Vue framework.

  In terms of front-end technology, we utilised the Vue architecture, with Vue-CLI aiding our development. We chose Vue because we wanted to develop a less web-heavy application, which is the trend in some part of the world. Had we used React, it might have been the right choice at some point, but the frontend pages as a whole would have been weighty and heavily dependent on communications to and from the backend, which would have been a departure from our original intent.

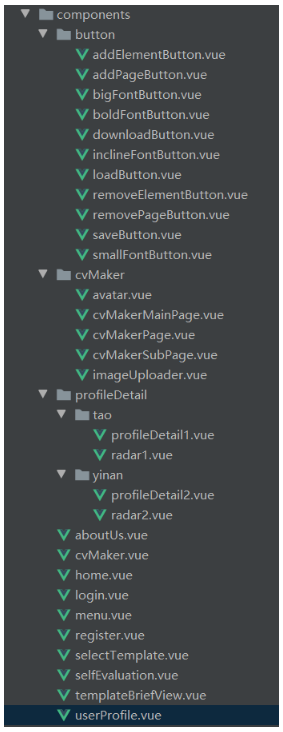
The Vue-router facilitates routing at the frontend, which reduces the burden of the server. The Vue compiler compiles and compresses a project into a single page website, where all the HTML contents are injected to that page at run-time, which reduces the overall size of a website. If you visit our website, you will see that while it appears like there are multiple pages, there is actually only one page with dynamically changing contents, which is achieved with the power of the Vue-router:



```
const routes = [{
  path: '/',
  component: App,

  children: [
    {
      name: 'App',
      path: '',
      components: {default:Home, top:Menu},
    },
    {
      path: '/selectTemplate',
      components: {default:selectTemplate, top:Menu},
    },
    {
      path: '/about',
      components: {default:about, top:Menu},
    },
    {
      path: '/login',
      components: {default:login, top:Menu},
    },
```

Figure 1: routes

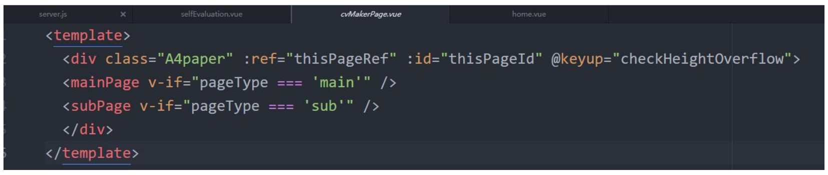
We completed a total of more than 30 Vue components with multiple levels of parent-child relationships, and they worked quite perfectly. Thanks to Vue's component-based design model, our front-end application is not as messy as what it would have been a decade ago. Moreover, this design pattern will make maintenance in the future quite effortless.



```
components
  button
    addElementButton.vue
    addPageButton.vue
    bigFontButton.vue
    boldFontButton.vue
    downloadButton.vue
    inclineFontButton.vue
    loadButton.vue
    removeElementButton.vue
    removePageButton.vue
    saveButton.vue
    smallFontButton.vue
  cvMaker
    avatar.vue
    cvMakerMainPage.vue
    cvMakerPage.vue
    cvMakerSubPage.vue
    imageUploader.vue
  profileDetail
    tao
      profileDetail1.vue
      radar1.vue
    yinan
      profileDetail2.vue
      radar2.vue
  aboutUs.vue
  cvMaker.vue
  home.vue
  login.vue
  menu.vue
  register.vue
  selectTemplate.vue
  selfEvaluation.vue
  templateBriefView.vue
  userProfile.vue
```

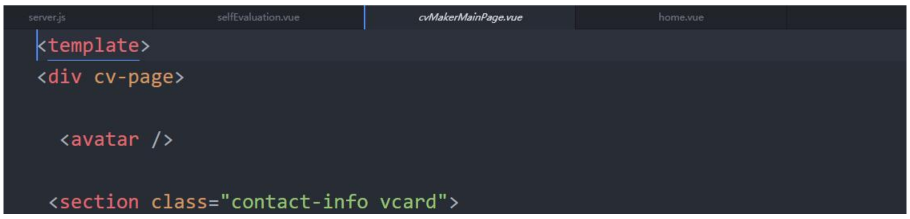
Figure 2: components

Below is an example of multi-level hierarchy, where the ”cvMakerPage” has two children components: ”mainPage” and ”subPage”, which will never be visible at the same time, while the ”mainPage” has a child ”avatar”. Furthermore, the ”avatar” has a child ”imageUploader”:



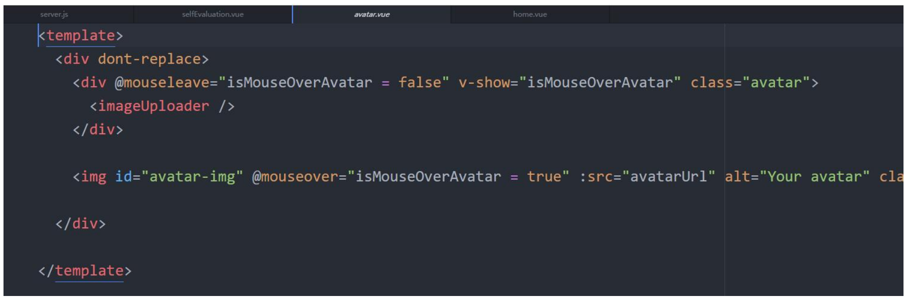
```
<template>
  <div class="A4paper" :ref="thisPageRef" :id="thisPageId" @keyup="checkHeightOverflow">
  <mainPage v-if="pageType === 'main'" />
  <subPage v-if="pageType === 'sub'" />
  </div>
</template>
```

Figure 3: cvMakerPage



```
<template>
<div cv-page>

  <avatar />

 <section class="contact-info vcard">
```

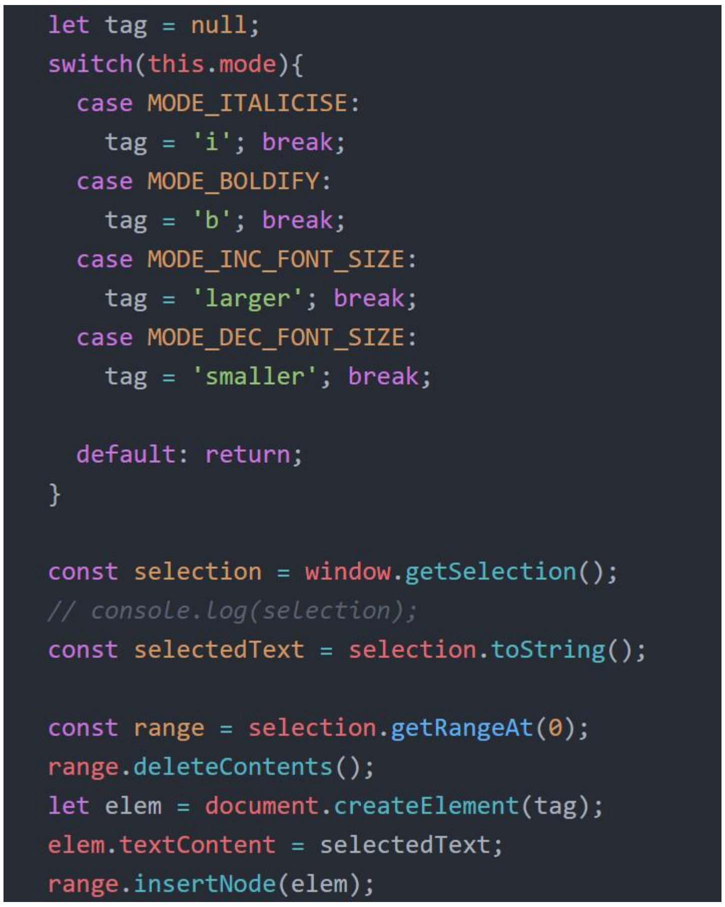
Figure 4: mainPage



```
<template>
  <div dont-replace>
    <div @mouseleave="isMouseOverAvatar = false" v-show="isMouseOverAvatar" class="avatar">
      <imageUploader />
    </div>

    <img id="avatar-img" @mouseover="isMouseOverAvatar = true" :src="avatarUrl" alt="Your avatar" cla

  </div>

</template>
```

Figure 5: avatar

- The main functionality of our website is online CV making, which was implemented by editing the html. Without decent understanding about the HTML, we wouldn't have made it functioning. For example, the italicising, boldifing and font size increasing/decreasing functionalities were implemented by inserting ”i”, ”b”, ”larger”, ”smaller” tags around the text that users select, respectively:

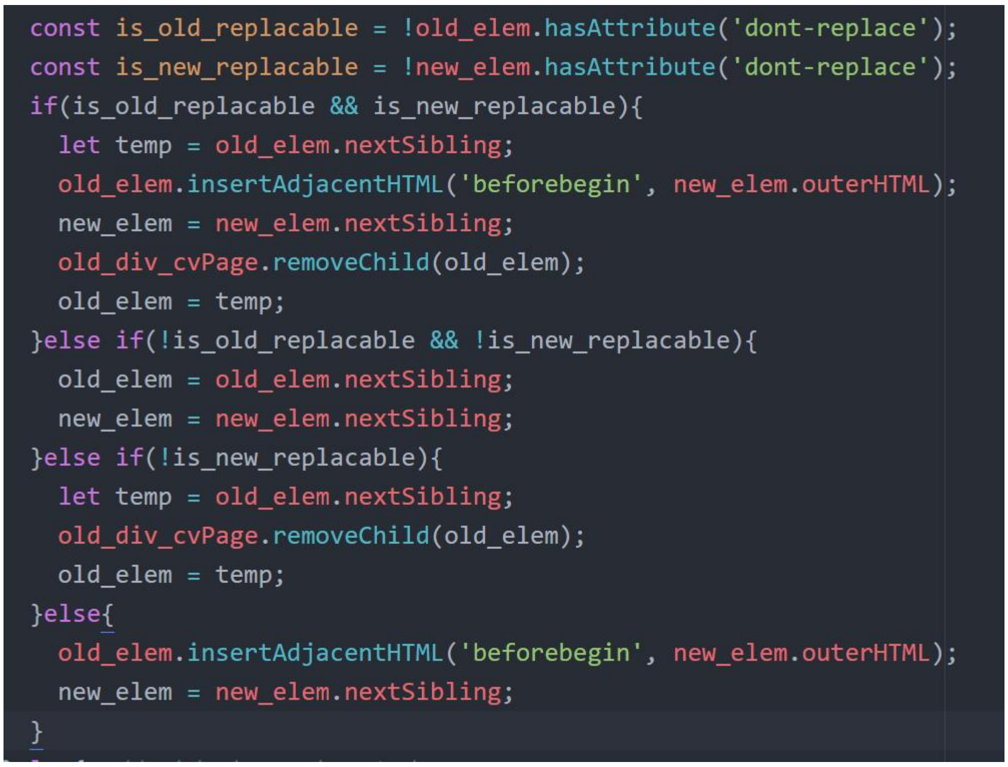


```
let tag = null;
switch(this.mode){
  case MODE_ITALICISE:
    tag = 'i'; break;
  case MODE_BOLDIFY:
    tag = 'b'; break;
  case MODE_INC_FONT_SIZE:
    tag = 'larger'; break;
  case MODE_DEC_FONT_SIZE:
    tag = 'smaller'; break;

  default: return;
}

const selection = window.getSelection();
// console.log(selection);
const selectedText = selection.toString();

const range = selection.getRangeAt(0);
range.deleteContents();
let elem = document.createElement(tag);
elem.textContent = selectedText;
range.insertNode(elem);
```

Figure 6: Inserting tags

We explored many HTML features such as contenteditable, which powered the text-editing functionality of the CV pages, and custom attributes, which we used to distinguish something that should be treated differently from their siblings.

As the above Fig.5 avatar shows, the first div has a custom attribute ”dont-replace”, because we do not want it to be replaced during loading progress. And this information is captured in the ”loadSavedData” method of the cvMaker.vue component:



```
const is_old_replacable = !old_elem.hasAttribute('dont-replace');
const is_new_replacable = !new_elem.hasAttribute('dont-replace');
if(is_old_replacable && is_new_replacable){
  let temp = old_elem.nextSibling;
  old_elem.insertAdjacentHTML('beforebegin', new_elem.outerHTML);
  new_elem = new_elem.nextSibling;
  old_div_cvPage.removeChild(old_elem);
  old_elem = temp;
}else if(!is_old_replacable && !is_new_replacable){
  old_elem = old_elem.nextSibling;
  new_elem = new_elem.nextSibling;
}else if(!is_new_replacable){
  let temp = old_elem.nextSibling;
  old_div_cvPage.removeChild(old_elem);
  old_elem = temp;
}else{
  old_elem.insertAdjacentHTML('beforebegin', new_elem.outerHTML);
  new_elem = new_elem.nextSibling;
}
```

Figure 7: custom attribute

- I think the idea behind our project, i.e. employing HTML and CSS and their manipulation to make text editors with built-in templates is quite innovative and creative, and we came up with it by ourselves, which I think might worth some meagre credit. That is why I am so arrogantly claiming an A+.

2.2.2 CSS

- We used the Vue framework to deliver pages, so it’s quite difficult to tell whether we had “style” tag in HTML pages because we did not have HTML pages during development. The styles resided in the “style” section for each Vue file and they were injected into the $< head >$ part of the page as $< style >$ tags at runtime. It’s just how the framework worked. However, we made sure we didn’t have style attributes in the template section of Vue files.

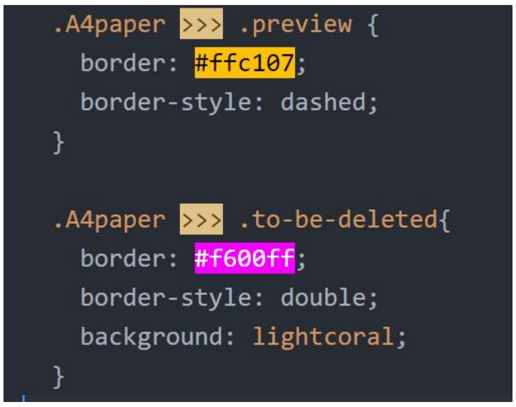
- Each Vue file could have any number of ”style” tags to contain CSS code, which already satisfies the purpose of not having internal or inline CSS code in traditional website, that is to keep different things at different places so that everything is more modular and therefore easier to maintain. The CSS code in Vue files are already separate from HTML as long as we do not put them in the ”template” section, and we did not. Despite that, we still placed most CSS code in separate files under the folder $src/view/index/assets$ for easier management.

- We successfully used basic CSS, Vue specific CSS and the classes provided by the Bootstrap framwork to make our frontend pages satisfy our poor and abnormal aesthetics.

Vue has two options for CSS code positioned in the "style" section: global or scoped. Although they will both be injected to the "head" section of the page when the pages are rendered, scoped CSS will have a "data-?" attribute attached to them, where ? is a seemly random value but is resolved at compile time. All the tags that related to a scoped CSS will be added the same attribute as the CSS. That is how the Vue identifies them.

Scoped CSS with combined or descendant selectors were used for precise location in specific pages while global class like .background (one with a furry glass effect to give the whole screen more colour) is accessible from all the pages.

Below is an example of using Vue specific CSS (the deep selector >>>):

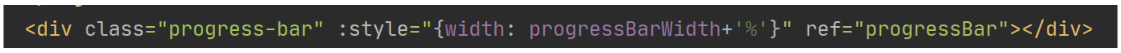


```
.A4paper >>> .preview {
  border: #ffc107;
  border-style: dashed;
}

.A4paper >>> .to-be-deleted{
  border: #f600ff;
  border-style: double;
  background: lightcoral;
}
```

Figure 8: Vue CSS

,which means every item that has class "preview" inside a container with class "A4paper" will have a yellow dashed border even those that are rendered by child components. Similarly, every item that has class "to-be-deleted" inside a "A4paper" container will have a red-purplish double border and a lightcoral background. In insertion/deletion mode of the CVmaker page, these two classes are added to the item that the user's mouse cursor is currently pointing at, and removed from that item when the cursor moves away.

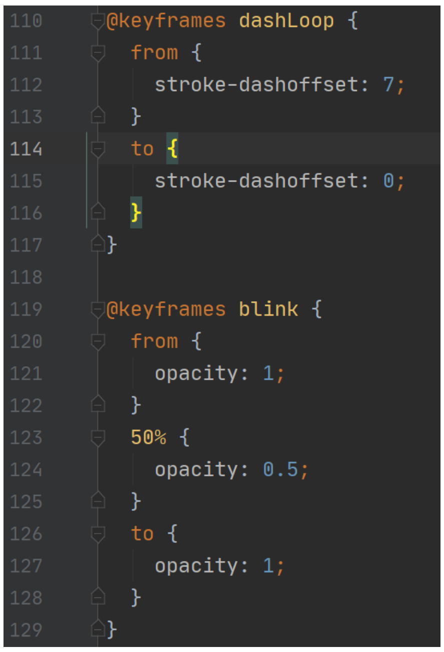
- The following is an example of how to adjust the progress bar according to the download progress in the downloadButton. We dynamically adjust the width of the bar to match the expected download time. By the way, a timer was used to control how often the progress bar refreshed to avoid blocking the execution since javascript is single-threaded.



```
<div class="progress-bar" :style="{width: progressBarWidth+'%'}" ref="progressBar"></div>
```

Figure 9: change style

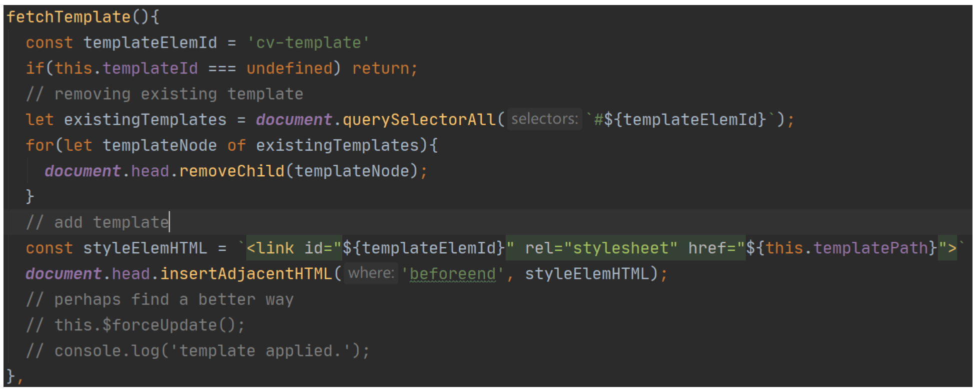
- SVG-based animation
  We completed some svg-based animation in CSS, which will be explained further in detail in the SVG section.



```css
@keyframes dashLoop {
  from {
    stroke-dashoffset: 7;
  }
  to {
    stroke-dashoffset: 0;
  }
}

@keyframes blink {
  from {
    opacity: 1;
  }
  50% {
    opacity: 0.5;
  }
  to {
    opacity: 1;
  }
}
```

Figure 10: css animation

- Dynamically retrieving CSS from the server
  The main feature of our website, i.e. making CVs with different templates is facilitated by dynamically fetching and replacing stylesheets at run-time. In CVMaker, the CSS that is responsible for the CV pages are fetched from the server according to the user's choice. Below is the method we wrote to remove existing template(s) and fetch the template that the user choose from the server. To be honest, this method does not fetch the template, it just adds a link to the template to the "head" of the page.



```javascript
fetchTemplate(){
  const templateElemId = 'cv-template'
  if(this.templateId === undefined) return;
  // removing existing template
  let existingTemplates = document.querySelectorAll( selectors: `#${templateElemId}`);
  for(let templateNode of existingTemplates){
    document.head.removeChild(templateNode);
  }
  // add template
  const styleElemHTML = `<link id="${templateElemId}" rel="stylesheet" href="${this.templatePath}">`
  document.head.insertAdjacentHTML( where: 'beforeend', styleElemHTML);
  // perhaps find a better way
  // this.$forceUpdate();
  // console.log('template applied.');
},
```

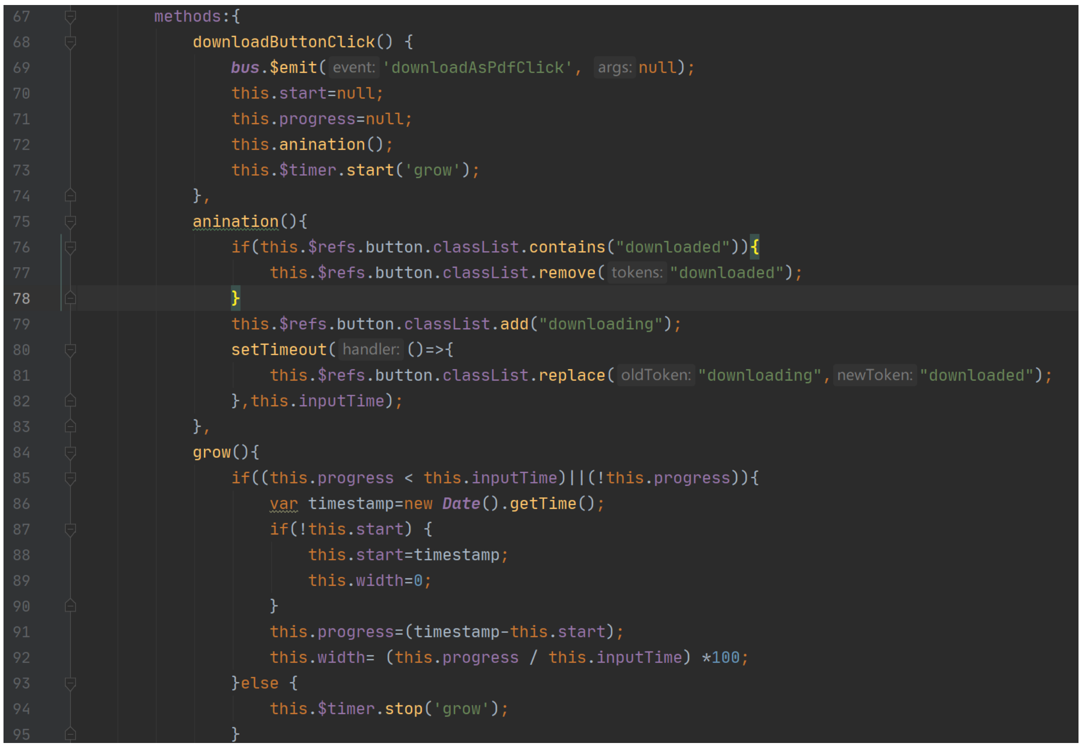
Figure 11: fetch template

- Again, I think using CSS files as templates is quite a good innovation since they are stylesheets themselves but we use them for other purposes rather than just for the sake of beautifying web pages. Furthermore, this also makes our templates quite easy to make: creating a new template is just a matter of creating a new CSS file, which makes our application quite extensible.

2.2.3 JS

Because of the vue template, it becomes straightforward to embed js in the page. Each page component exists separately as an element, and we write js methods internally or externally that can change the arithmetic. Of course, this is not technically a javascript file anymore, but primarily at the developmental level, they are a kind of Stuff.

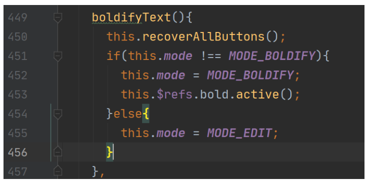
We wrote a lot of front-end and back-end js logic to ensure that the complete project documentation worked adequately. Similar to the button or switch screen functions are used to implement the vue methods. The example below is the js method after the download button is pressed.



```
        methods:{
            downloadButtonClick() {
                bus.$emit( event: 'downloadAsPdfClick',  args: null);
                this.start=null;
                this.progress=null;
                this.anination();
                this.$timer.start('grow');
            },
            anination(){
                if(this.$refs.button.classList.contains("downloaded")){
                    this.$refs.button.classList.remove( tokens: "downloaded");
                }
                this.$refs.button.classList.add("downloading");
                setTimeout( handler: ()=>{
                    this.$refs.button.classList.replace( oldToken: "downloading", newToken: "downloaded");
                },this.inputTime);
            },
            grow(){
                if((this.progress < this.inputTime)||(!this.progress)){
                    var timestamp=new Date().getTime();
                    if(!this.start) {
                        this.start=timestamp;
                        this.width=0;
                    }
                    this.progress=(timestamp-this.start);
                    this.width= (this.progress / this.inputTime) *100;
                }else {
                    this.$timer.stop('grow');
                }
```

Figure 12: resume template

We also made full use of the strengths of Vue component communication in the mutual communication of components. Below is an example of a parent component calling a child component method. In the CVMaker page, when we press the bold button in the tools bar and complete the click, the button in an active state.



```
    boldifyText(){
      this.recoverAllButtons();
      if(this.mode !== MODE_BOLDIFY){
        this.mode = MODE_BOLDIFY;
        this.$refs.bold.active();
      }else{
        this.mode = MODE_EDIT;
      }
    },
```

Figure 13: component communication

2.2.4 PNG

We used GMIP for mapping, including the default resume header and png diagrams for 404 pages. The source file of the 404 page is saved in $src/view/index/img$. We used masks, filters and transparent alaph channels, among other techniques.

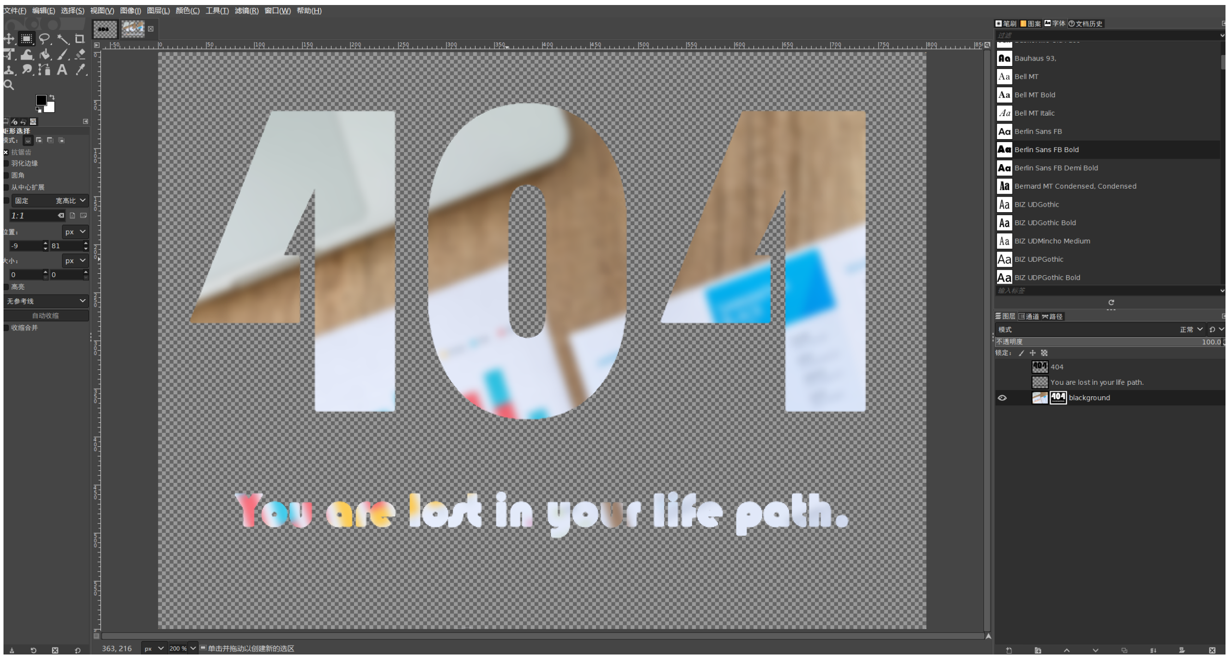


Figure 14: 404.png

2.2.5 SVG

In the use of SVG, we have used a variety of ways to construct SVG images to take full advantage of his benefits. We even created SVG animation on the home page. We will describe this in detail below.

- Basic SVG images
  We used tools like Inkscape to draw simple SVG portraits, and since the team has the skills to operate adobe Kit experience, so we are light on SVG production.

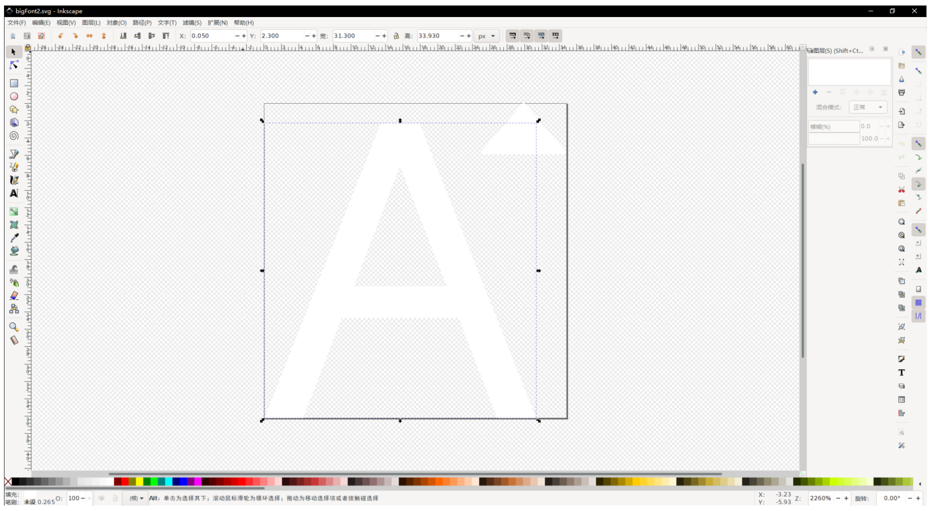


Figure 15: Inkscape

We used this basic graphical drawing to create 12 buttons, 3 of which are embedded in the page, while the remaining nine are used as Individual components are independent of the elements in the *src/components/button*. We take advantage of the object-oriented component design of the Vue components so that each module is easy to maintain and update later.

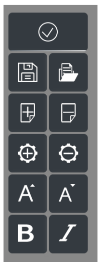
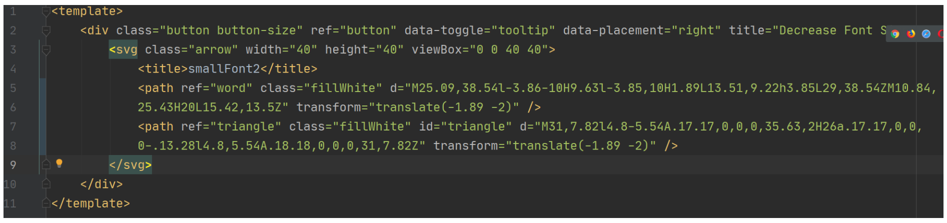


Figure 16: buttonBar



```
<template>
    <div class="button button-size" ref="button" data-toggle="tooltip" data-placement="right" title="Decrease Font S
        <svg class="arrow" width="40" height="40" viewBox="0 0 40 40">
            <title>smallFont2</title>
            <path ref="word" class="fillWhite" d="M25.09,38.54l-3.86-10H9.63l-3.85,10H1.89L13.51,9.22h3.85L29,38.54ZM10.84,
            25.43H20L15.42,13.5Z" transform="translate(-1.89 -2)" />
            <path ref="triangle" class="fillWhite" id="triangle" d="M31,7.82l4.8-5.54A.17.17,0,0,0,35.63,2H26a.17.17,0,0,
            0-.13.28l4.8,5.54A.18.18,0,0,0,31,7.82Z" transform="translate(-1.89 -2)" />
        </svg>
    </div>
</template>
```

Figure 17: smallFontButton

- SVG-based css animation
  We were not satisfied with making basic SVG graphics. We created four SVG animations with CSS animation effects. They are the start button on the home page, the continue and new buttons on the user-profile page, and the download button inside CVMaker. The most complex one is the download button, which activates the animation by changing the button's class when clicked.

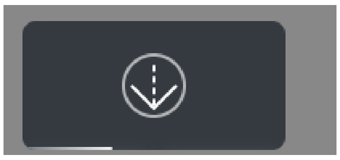
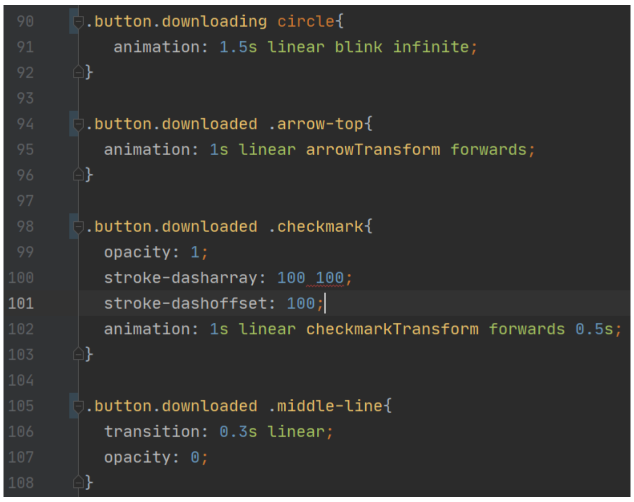


Figure 18: downloadButton animation

The download animation is divided into four parts, the first is the flashing of the outer ring, the second is the downward movement of the vertical line in the middle, and the third is the download of The middle arrow pattern becomes a checkmark when finished, and the fourth is the download progress bar at the bottom.



```
.button.downloading circle{
   animation: 1.5s linear blink infinite;
}

.button.downloaded .arrow-top{
  animation: 1s linear arrowTransform forwards;
}

.button.downloaded .checkmark{
  opacity: 1;
  stroke-dasharray: 100 100;
  stroke-dashoffset: 100;
  animation: 1s linear checkmarkTransform forwards 0.5s;
}

.button.downloaded .middle-line{
  transition: 0.3s linear;
  opacity: 0;
}
```

Figure 19: downloadButton animation

- svg animation based on vue-lottie
  Of course, doing this will not satisfy our ambition to try the coolest animations. So we introduced the vue-lottie open-source package, which is based on the lottie. Vue-lottie project vue architecture lottie can be interpreted as an SVG animation interpreter, and he supports the use of SVG in adobe After Effects exports complex animations to a JSON file and then self-rendering through the front-end of the web page to get cool effects.

  We've made a dynamic animation on the home page to highlight our theme, which we're sure you've seen.We save the exported animation JSON file that we send to AE in the $src/view/index/assets/animation$ folder.

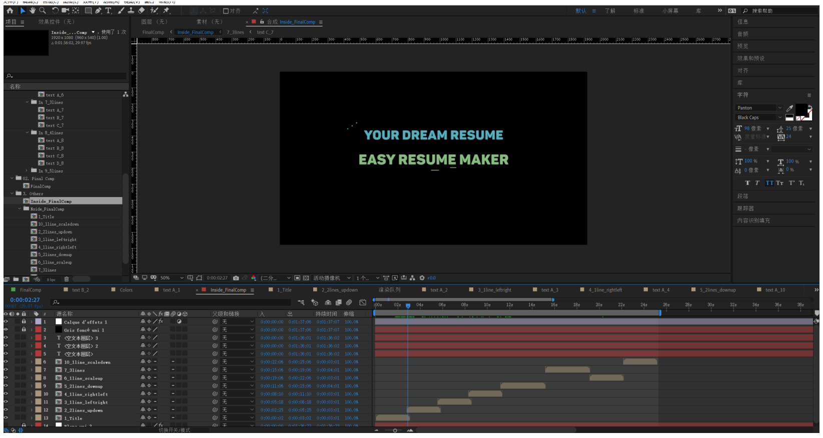


Figure 20: After Effect

Since this lottie tool is so new, we think we have made it pretty far ahead of the curve in terms of SVG usage.

2.3 Server Side

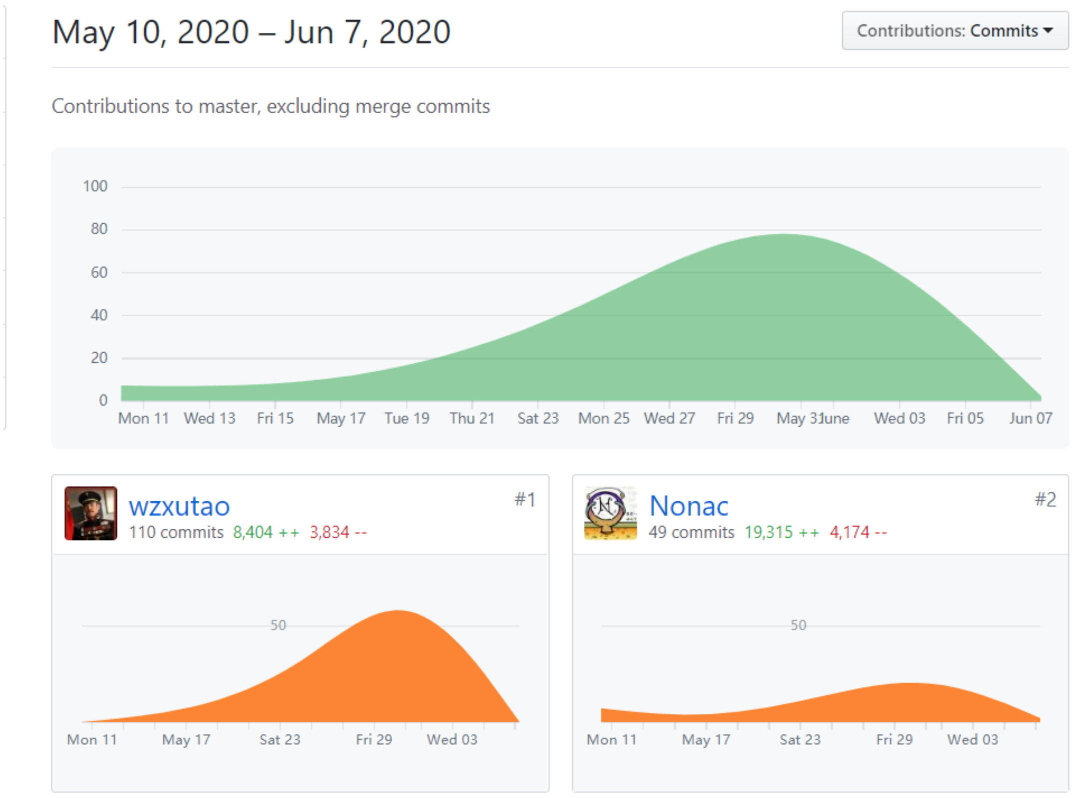
2.3.1 Server

2.3.2 Database

2.3.3 Dynamic pages

3 Working practices of the group

We used GitHub technology for remote collaboration, with Tao Xu handling the back-end technology and Yinan Yang is in charge of front-end technology. Our project address is https://github.com/Nonac/webtech.The screenshot below reflects the progress of our project.



May 10, 2020 – Jun 7, 2020

Contributions to master, excluding merge commits

wzxutao #1
110 commits 8,404 ++ 3,834 --

Nonac #2
49 commits 19,315 ++ 4,174 --

Figure 21: After Effect