Python Documentation

version

February 02, 2021

Contents

Vivvo Templates Documentation

Introduction

This guide is to inform the front end developer of the stucture of the Vivvo template system, so that the UI can be configured to meet the customers requirements.

System Structure

Architecture

CitizenOne is composed of a number of services that are orchestrated to provide trust as a service. This manual deals with the service that delivers the citizen facing web pages.

The front end component is made up of two parts.

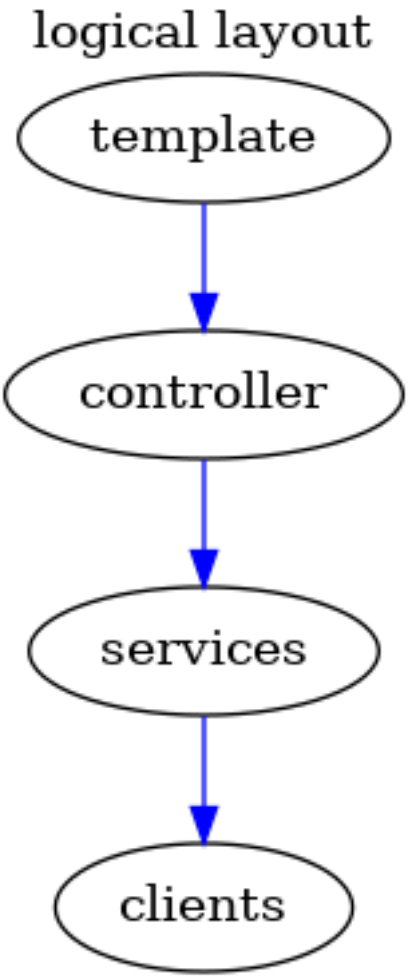
1. The templates
2. The controllers

The controllers broker the logic to generate and deliver the html/javascript/css files to the client depending on the interaction in context. For example, during a login request the controller will assemble the templates required to capture the user credentials. The controllers are also responsible to talk to the service layer, the service layer will invoke the right client to call out to other microservices when required.

Sidebar Title

Optional Sidebar Subtitle

Subsequent indented lines comprise the body of the sidebar, and are interpreted as body elements.

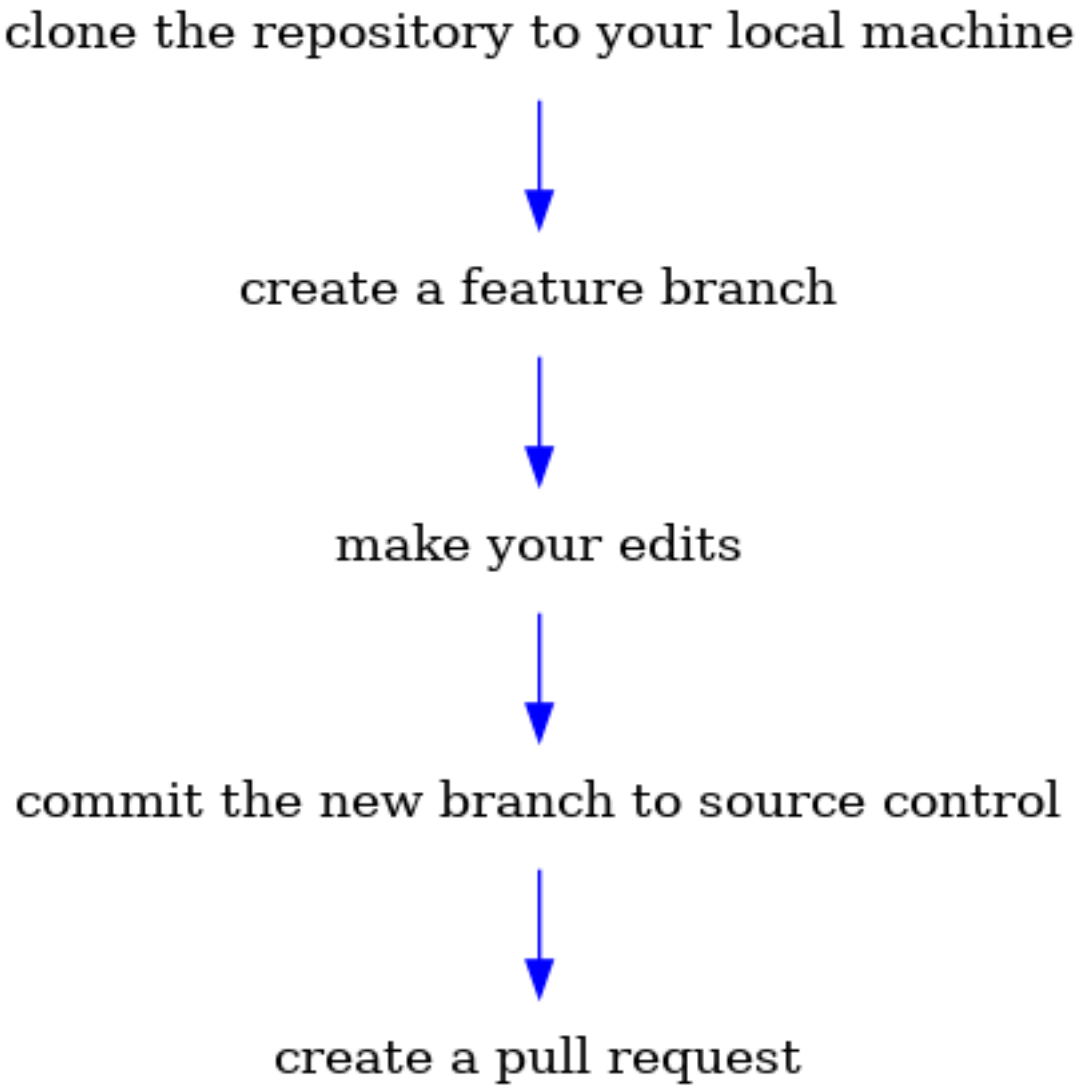


Page Controller Relationships

Page Name	Controllers	Purpose
create-account.html	profileController	account registration, high velocity federated creation
dashboard.html	dashBoardController	service card management
index.html	loginContoller	account login
profile.html	profileController	manage profile
reset-password-confirmation.html	profileController	account reset activities
reset-password.html	profileController	account reset activities
service-add.html	serviceCardController	service card management
service-card-back.html	serviceCardController	service card interactions
service-linker.html	serviceCardController	service card enablement
service-onboard.html	serviceCardController	service enrollment

Using the Tempate System

Vivvo uses github source control. Our developers work with your UI/UX team to replace placeholder information with system tagging. The workflow to achieve this is as follows:



Using the github tools:

```
1 git clone https://github.com/Vivvo/{{your template registry}}
2 cd {{your template registry}}
3 git checkout -b "template changes"
4 # make your edits
5 git add **
6 git commit -m "edits to html and style"
7 git push
```

Tagging reffers to adding logic that will add the right context to the page. In your templates, your placeholder information may say “Jane Doe”, and the Vivvo developers will replace this with {{ profile.FirstName profile.LastName }} for example.

Your template before tagging will look like this:

```
1  <div class=" row col-lg-4 ml-1">
2                  <h1>Welcome to your Account Login Tom Sawyer </h1>
3                  <h3>Your portal to personalized, simple and secure service.</h3>
4              </div>
5              <div class="card col-lg-4 mt-5">
```

```
 6                 <div class="card-body contact-box">
 7                     <div>
 8                         <div class="ml-4 mb-4">
 9                             <h2>Questions Contact Us: </h2>
10                             <p>
11                             </p>
12                         </div>
13                     </div>
```

And after the tagging is added

```
 1  <div class=" row col-lg-4 ml-1">
 2                 <h1>Welcome to your Account Login {{ .me.FirstName .me.LastName }} </h1>
 3                 <h3>Your portal to personalized, simple and secure service.</h3>
 4             </div>
 5             <div class="card col-lg-4 mt-5">
 6                 <div class="card-body contact-box">
 7                     <div>
 8                         <div class="ml-4 mb-4">
 9                             <h2>Questions Contact Us: </h2>
10                             <p>
11                             </p>
12                         </div>
13                     </div>
```

The .me part of the tag is a data object that is created from the controller. Data objects can be a single object, an object with embedded objects or a list. The .me for example is an object instantiated in the controller and attached to a view argument called “me”. The me object represents the user in context, and attributes of the user can be displayed by accessors such as:

- .me.FirstName
- .me.Address[0].Street
- .me.Phone[0].number

As examples.

Data objects that are available to views are found below. Not all objects are available to all pages, but rather exposed to the relevant page. (e.g. profile objects are exposed to the profile pages, and service card objects are exposed to service card pages. Objects the templates can be found in Appendix A

Vivvo makes use of default template functions defined in the https://revel.github.io/manual/templates.html webpage, and we create some custom ones for rendering special blocks. Examples are found in Appendix B

Appendix A

```
type ConsentModel struct {
    HasConsent  bool     `json:"hasConsent"`
    Scopes      []string `json:"scopes"`
    ExpiresOn   string   `json:"expiresOn"`
    CreatedDate string   `json:"createdDate"`
}

type GrantedConsentModel struct {
    ServiceCardId     string   `json:"serviceCardId"`
    PolicyId          string   `json:"policyId"`
    consentExpiryDate string   `json:"consentExpiryDate"`
    Scopes            []string `json:"scopes"`
}

type IdentityServiceCard struct {
    IdentityServiceCardId int64        `json:"identityServiceCardId"`
    IdentityId            string       `json:"identityId"`
    ServiceCardId         string       `json:"serviceCardId"`
    SortOrder             int64        `json:"sortOrder"`
    FatServiceCard        *ServiceCard `json:"serviceCard"`
    LightServiceCard      *ServiceCard `json:"lightServiceCard"`
    NotificationCount     int          `json:"notificationCount"`
}

type MaintenanceMessage struct {
    MaintenanceId int    `json:"maintenanceId"`
    Title         string `json:"title"`
    Details       string `json:"details"`
    StartDate     string `json:"startDate"`
    EndDate       string `json:"endDate"`
}

type Me struct {
    Identity           Identity `json:"identity"`
    Business           string   `json:"business,omitempty"`
    HasBusinessConnect bool     `json:"hasBusinessConnect,omitempty"`
    RedirectUrls       []Link   `json:"links,omitempty"`
}

type Identity struct {
    IdentityId               string                  `json:"identityId"`
    FirstName                string                  `json:"firstName"`
    LastName                 string                  `json:"lastName"`
    MiddleName               string                  `json:"middleName"`
    UserName                 string                  `json:"userName"`
    Salutation               string                  `json:"salutation,omitempty"`
    DateOfBirth              string                  `json:"dateOfBirth,omitempty"`
    SupportCode              string                  `json:"supportCode,omitempty"`
    AssuranceLevel           AssuranceLevel          `json:"assuranceLevel"`
    Emails                   []Email                 `json:"emails"`
    Phones                   []Phone                 `json:"phones"`
    Addresses                []Address               `json:"addresses"`
    SecondaryAuthentications []SecondaryAuthentication `json:"secondaryAuthentication"`
    ExternalIdentifiers      []ExternalIdentifiers   `json:"externalIdentifiers"`
}

type ExternalIdentifiers struct {
    Name     string `json:"name"`
```

```
    Value    string `json:"value"`
    Provider string `json:"provider"`
}

type SecondaryAuthentication struct {
    SecondaryAuthenticationId     int    `json:"secondaryAuthenticationId"`
    SecondaryAuthenticationType   string `json:"secondaryAuthenticationType"`
    SecondaryAuthenticationStatus string `json:"secondaryAuthenticationStatus"`
    IdentityId                    string `json:"identityId"`
    PhoneId                       int    `json:"phoneId"`
    Url                           string `json:"url"`
    IsGlobal                      bool   `json:"isGlobal"`
}

type AssuranceLevel struct {
    Level         int           `json:"level"`
    ClaimProvider ClaimProvider `json:"claimProvider"`
    IdentityId    string        `json:"identityId"`
}

type ClaimProvider struct {
    ClaimProviderId      int      `json:"claimProviderId"`
    Name                 string   `json:"name"`
    Description          string   `json:"description"`
    OnboardingIdentifier string   `json:"onBoardingIdentifier"`
    BaseUrl              string   `json:"baseUrl"`
    OnboardingParams     []string `json:"onboardingParams"`
    Loa                  int      `json:"loa"`
    OnboardingPath       string   `json:"onboardingPath"`
    LogoUrl              string   `json:"logoUrl"`
    Prerequisites        []string `json:"prerequisites"`
}

type Link struct {
    Name string `json:"name"`
    URL  string `json:"url"`
}

type Notification struct {
    NotificationId   int64  `json:"notificationId"`
    ApplicationName  string `json:"applicationName"`
    OrganizationName string `json:"organizationName"`
    Subject          string `json:"subject"`
    Body             string `json:"body"`
    NotificationDate string `json:"notificationDate"`
    Status           string `json:"status"`
    FolderType       string `json:"folderType"`
    ReferenceNumber  string `json:"referenceNumber"`
    Recipient        string `json:"recipient"`
}

type PolicyCallback struct {
    PolicyCallbackId    int64  `json:"policyCallbackId,omitempty"`
    PolicyId            string `json:"policyId,omitempty"`
    RevisionNum         int64  `json:"revisionNum,omitempty"`
    DataBundleType      string `json:"dataBundleType,omitempty"`
    Url                 string `json:"url,omitempty"`
    ConsentScopeGranted bool   `json:"consentScopeGenerated,omitempty"`
}

```

```go
type Policy struct {
    PolicyId          string         `json:"policyId"`
    Name              string         `json:"name"`
    Description       string         `json:"description,omitempty"`
    PublicKey         string         `json:"publicKey,omitempty"`
    DefaultExpiryDays int            `json:"defaultExpiryDays,omitempty"`
    ActiveRevision    PolicyRevision `json:"activeRevision,omitempty"`
}

type PolicyRevision struct {
    PolicyId    string           `json:"policyId,omitempty"`
    RevisionNum int              `json:"revisionNum,omitempty"`
    PolicyText  string           `json:"policyText,omitempty"`
    IsActive    bool             `json:"isActive,omitempty"`
    CreatedDate string           `json:"createdDate,omitempty"`
    Callbacks   []PolicyCallback `json:"callbacks,omitempty"`
    Scopes      []PolicyScope    `json:"scopes,omitempty"`
    Rules       []PolicyRule     `json:"rules,omitempty"`
}

type PolicyRule struct {
    PolicyRuleId   int64    `json:"policyRuleId,omitempty"`
    PolicyId       string   `json:"policyId,omitempty"`
    RevisionNum    int64    `json:"revisionNum,omitempty"`
    PolicyRuleType string   `json:"policyRuleType,omitempty"`
    RuleId         int64    `json:"ruleId,omitempty"`
    Args           []string `json:"args,omitempty"`
}

type PolicyScope struct {
    PolicyScopeId  int64    `json:"policyScopeId,omitempty"`
    PolicyId       string   `json:"policyId,omitempty"`
    RevisionNum    int64    `json:"revisionNum,omitempty"`
    Value          string   `json:"value,omitempty"`
    Description    string   `json:"description,omitempty"`
    Tag            string   `json:"tag,omitempty"`
    Reason         string   `json:"reason,omitempty"`
    IsRequired     bool     `json:"isRequired,omitempty"`
    Consented      bool     `json:"consented,omitempty"`
    IncludedFields []string `json:"includedFields,omitempty"`
}

type Address struct {
    AddressId    int    `json:"addressId,omitempty"`
    IdentityId   string `json:"identityId"`
    AddressType  string `json:"addressType"`
    PersonaType  string `json:"personaType"`
    AddressLine1 string `json:"addressLine1"`
    AddressLine2 string `json:"addressLine2"`
    City         string `json:"city"`
    ProvinceCode string `json:"provinceCode"`
    PostalCode   string `json:"postalCode"`
    CountryCode  string `json:"countryCode"`
    IsPrimary    bool   `json:"isPrimary"`
}

type Phone struct {
    PhoneId         int    `json:"phoneId"`
    IdentityId      string `json:"identityId"`
    PhoneType       string `json:"phoneType"`
```

```
179     CountryCallCode string `json:"countryCallCode"`
180     CountryCode     string `json:"countryCode"`
181     PhoneNumber     string `json:"phoneNumber"`
182     Extension       string `json:"extension"`
183     IsPrimary       bool   `json:"isPrimary"`
184     IsMfa           bool   `json:"isMfa"`
185     Verified        bool   `json:"verified"`
186 }
187
188 type Email struct {
189     EmailId      int    `json:"emailId"`
190     EmailAddress string `json:"emailAddress"`
191     IsPrimary    bool   `json:"isPrimary"`
192     IsVerified   bool   `json:"isVerified"`
193     IdentityId   string `json:"identityId"`
194 }
195
196 type Register struct {
197     FirstName         string `json:"firstName"`
198     MiddleName        string `json:"middleName"`
199     LastName          string `json:"lastName"`
200     Email             string `json:"email"`
201     UserName          string `json:"userName"`
202     Password          string `json:"password"`
203     SupportCode       string `json:"supportCode"`
204     GotoUrl           string `json:"goToUrl"`
205     TermsOfUse        string `json:"termsOfUse"`
206     CreatedDate       string `json:"createdDate"`
207     RecaptchaResponse string `json:"recaptchaResponse"`
208     ProviderId        string `json:"providerId"`
209 }
210
211 type ServiceCardApplication struct {
212     ApplicationId int                     `json:"applicationId"`
213     Name          string                  `json:"name"`
214     Organization  ServiceCardOrganization `json:"organization"`
215     PolicyId      string                  `json:"policyId"`
216 }
217
218 type ServiceCardCallback struct {
219     Files   []gosdkModels.ServiceCardFileDto  `json:"files"`
220     Labels  []gosdkModels.ServiceCardLabelDto `json:"labels"`
221     Actions []gosdkModels.ServiceCardTaskDto  `json:"actions"`
222 }
223
224 type ServiceCardContact struct {
225     FirstName       string                    `json:"firstName,omitempty"`
226     LastName        string                    `json:"lastName,omitempty"`
227     Prefix          string                    `json:"prefix,omitempty"`
228     Suffix          string                    `json:"suffix,omitempty"`
229     Title           string                    `json:"title,omitempty"`
230     EntityId        string                    `json:"entityId,omitempty"`
231     Phone           ServiceCardContactPhone   `json:"phone,omitempty"`
232     Phone2          ServiceCardContactPhone   `json:"phone2,omitempty"`
233     Fax             ServiceCardContactPhone   `json:"fax,omitempty"`
234     Email           string                    `json:"email,omitempty"`
235     Url             string                    `json:"url,omitempty"`
236     MailingAddress  ServiceCardAddress        `json:"mailingAddress,omitempty"`
237     PhysicalAddress ServiceCardAddress        `json:"physicalAddress,omitempty"`
238     PhotoUrl        string                    `json:"photoUrl,omitempty"`
```

```
239     Type            string                      `json:"type,omitempty"`
240     OrgUnit         ServiceCardOrganizationUnit `json:"orgUnit,omitempty"`
241 }
242
243 type ServiceCardContactList struct {
244     CtaTitle     string `json:"ctaTitle,omitempty"`
245     CtaUrl       string `json:"ctaUrl,omitempty"`
246     ContactTitle string `json:"contactTitle,omitempty"`
247     Contacts     []ServiceCardContact
248 }
249
250 type ServiceCardContactPhone struct {
251     Prefix    string `json:"prefix,omitempty"`
252     Number    string `json:"number,omitempty"`
253     Extension string `json:"extension,omitempty"`
254     Note      string `json:"note,omitempty"`
255 }
256
257 type ServiceCardAddress struct {
258     Line       string `json:"line,omitempty"`
259     City       string `json:"city,omitempty"`
260     Province   string `json:"province,omitempty"`
261     Country    string `json:"country,omitempty"`
262     PostalCode string `json:"postalCode,omitempty"`
263 }
264
265 type ServiceCardOrganizationUnit struct {
266     Name            string                   `json:"name,omitempty"`
267     EntityId        string                   `json:"entityId,omitempty"`
268     Phone           ServiceCardContactPhone `json:"phone,omitempty"`
269     Phone2          ServiceCardContactPhone `json:"phone2,omitempty"`
270     Fax             ServiceCardContactPhone `json:"fax,omitempty"`
271     Email           string                   `json:"email,omitempty"`
272     Url             string                   `json:"url,omitempty"`
273     MailingAddress  ServiceCardAddress      `json:"mailingAddress,omitempty"`
274     PhysicalAddress ServiceCardAddress      `json:"physicalAddress,omitempty"`
275     PhotoUrl        string                   `json:"photoUrl,omitempty"`
276     Type            string                   `json:"type,omitempty"`
277 }
278
279 type ServiceCardAdditionalDetails struct {
280     ActionTitle string `json:"actionTitle,omitempty"`
281     Description string `json:"description,omitempty"`
282     Url         string `json:"url,omitempty"`
283 }
284
285 type ServiceCard struct {
286
287     //The following are required for both ServiceCardDto and LightServiceCardDto
288     IsPilot           bool   `json:"isPilot"`
289     AutomationEnabled bool   `json:"automationEnabled"`
290     ServiceCardId     string `json:"serviceCardId"`
291     Title             string `json:"title"`
292     ShortDescription  string `json:"shortDescription"`
293     AdditionalDetail  string `json:"additionalDetail"`
294     ServiceCardType   string `json:"serviceCardType"`
295
296     //The following are not required and are a part of ServiceCardDto
297     Saved                   bool                     `json:"saved,omitempty"`
298     ActivePolicy            Policy                   `json:"activePolicy,omitempty"`
```

```
299      BasicContentCallbackUrls []string                  `json:"basicContentCallbackUrls
300      Applications             []ServiceCardApplication   `json:"applications,omitempty"`
301      Tasks                    []gosdk.ServiceCardTaskDto `json:"tasks,omitempty"`
302  }
```

Appendix B

```
	revel.TemplateFuncs["generic"] = func(obj map[string]string) map[string]interface{}
		ret := make(map[string]interface{})
		for key, val := range obj {
			ret[key] = val
		}
		return ret
	}

	revel.TemplateFuncs["json"] = func(f interface{}) string {
		s, _ := json.MarshalIndent(f, "", "    ")
		return string(s)
	}

	revel.TemplateFuncs["doesFileDtoHaveFiles"] = func(f []gosdkModels.ServiceCardFileD
		if len(f) < 0 {
			return true
		}
		return false
	}

	revel.TemplateFuncs["translate"] = func(viewArgs map[string]interface{}, key string
		lang, ok := viewArgs[revel.CurrentLocaleViewArg].(string)
		if !ok {
			revel.AppLog.Warnf("unable to determine language, %s", viewArgs)
			lang = "en"
		}

		return language.Service.GetByKeyAndLang(lang, key)
	}

	revel.TemplateFuncs["map"] = func(obj interface{}) map[string]interface{} {
		return structs.Map(obj)
	}

	revel.TemplateFuncs["url_for"] = func(url string) string {
		return utils.UrlFor(url)
	}

	revel.TemplateFuncs["changeLanguage"] = func(url, currentLocale, targetLocale strin
		return strings.Replace(url, fmt.Sprintf("/%s/", currentLocale), fmt.Sprintf
	}

	revel.TemplateFuncs["tmpl"] = func(parentContext map[string]interface{}, args ...in
		var templateName = ""

		var viewArgs interface{}
		switch len(args) {
		case 0:
			revel.AppLog.Errorf("tmpl: No arguments passed to template call")
		case 1:
			templateName = args[0].(string)
		default:
			templateName = args[0].(string)
			viewArgs = args[1]
			if len(args) > 3 {
				revel.AppLog.Errorf("tmpl: Received more parameters then ne
			}
		}
```

```
59
60              viewArgsMap, ok := viewArgs.(map[string]interface{})
61              if !ok {
62                      fmt.Printf("viewArgs is type %T, %+v", viewArgs, viewArgs)
63                      revel.AppLog.Error("tmpl: failed to convert to map", "name", templa
64              }
65              for _, key := range []string{"_csrftoken", "currentLocale", "errors", "flas
66                      if val, ok := parentContext[key]; ok {
67                              viewArgsMap[key] = val
68                      }
69              }
70
71              var buf bytes.Buffer
72              tmpl, err := revel.MainTemplateLoader.Template(templateName)
73              if err == nil {
74                      err = tmpl.Render(&buf, viewArgsMap)
75                      if err != nil {
76                              revel.AppLog.Error("tmpl: Failed to render", "name", templa
77                      }
78              } else {
79                      revel.AppLog.Error("i18ntemplate: Failed to render i18ntemplate ",
80              }
81              return template.HTML(buf.String())
82      }
83
84      revel.TemplateFuncs["html"] = func(t string) template.HTML {
85              return template.HTML(t)
86      }
87
88      revel.TemplateFuncs["formatPhoneNumber"] = func(t string) string {
89              return "(" + string(t[0:3]) + ") " + string(t[3:6]) + "-" + string(t[6:10])
90      }
91
92      revel.TemplateFuncs["extractParams"] = func(p []gosdkModels.ServiceCardTaskDto, csr
93              revel.AppLog.Debugf("adding csfr_token to form submission %v", csrf_token)
94              revel.AppLog.Debugf("Vivvo-Test %+v", p[0].Params)
95              if len(p[0].Params) > 0 {
96                      p[0].Params["csrf_token"] = csrf_token
97                      return p[0].Params
98              }
99              return map[string]interface{}{
100                     "csrf_token": csrf_token,
101             }
102     }
103
104     revel.TemplateFuncs["formatTime"] = func(t string) string {
105             if len(t) > 0 {
106                     t = t[:strings.IndexByte(t, '.')] + "Z"
107             }
108
109             formatedTime, err := time.Parse(time.RFC3339, t)
110             if err != nil {
111                     fmt.Println(err.Error())
112                     return t
113             }
114             return formatedTime.Format("02-Jan-06 15:04:05")
115     }
116
117     revel.TemplateFuncs["sortAndReverseMap"] = func(m map[string]string) *slicemultimap
118             s := slicemultimap.New()
```

```
119             r := slicemultimap.New()
120
121             for key, value := range m {
122                     s.Put(value, key)
123             }
124             var keys []string
125             for _, key := range s.Keys() {
126                     keys = append(keys, key.(string))
127             }
128             sort.Strings(keys)
129             for _, k := range keys {
130                     value, ok := s.Get(k)
131                     if ok {
132                             r.Put(k, value)
133                     }
134             }
135             return r
136     }
```

Template Structure

Assets

- icons - we used font awesome ([learn more here](https://fontawesome.com/how-to-use/on-the-web/referencing-icons/basic-use)), free icon library, but we did't find all needed icons there as such we have added missing icons to this folder as .svg
- images - all images used in a project are contained here (logo etc.)
- Sass - all SCSS styles are here

Pages

This folder contains main pages: * *index* login * *create-account* registration * *reset-password* forgot password * *reset-password-confirmation* * *dashboard* (my services) * *service-add* (search and link services) * *service-linker* (steps for linking the service) * *service-onboard* (service onboarding) * *service-card-back* (service card page for the linked service) * *profile*

Includes

Includes are small html components that we're importing to the pages. For example *header* is one of them, it makes the code easier to maintain. Include is imported to the page, and when we want to make some changes, we can just modify an include and the change will be visible on any page where the include is imported

Each page has a js function that loads all needed includes to the page:

```
1 $(function(){
2     $("#dashboard-navigation").load("../includes/dashboard-navigation.html");
3     $(".serviceCard").load("../includes/service-card.html");
4   });
```

and <*div*> element inside *html*, id will match includes name:

```
1  <div id="dashboard-navigation"></div>
```

SCSS

Each page and include has its SCSS, so it makes it easy to maintain. SCSS files name match the HTML name: *invoices.html = invoices.scss*. You'll also see a few files that are not matching any HTML file: * *index.scss* - this file is used to compile correct SCSS files, if you create a new SCSS you need to import, the stylesheet here, as`@import "invoices";` (use the name, .scss extension is not needed) * *variables.scss* sass allows to use variables: ex *$body-color: #20354E;* - defines body text color, if we change it

it will update the colour in each place that was used

- *$body-color* - body text color
- *$heading-color* - heading color
- *$theme-primary* - used for primary color (background in a primary button, border in a secondary button)
- *$theme-red* - error messages background
- *$theme-green* - success messages
- *$theme-primary-dark* - hover state for buttons
- *$theme-primary-grey* - disabled elements
- *$theme-background* - background color
- *$theme-light-grey* - used as a background for some elements (ex. Create Account > Terms of Use)
- *$theme-light-grey* - used for borders

- we used the bootstrap (free and open-source CSS framework - [read more](https://getbootstrap.com/docs/4.0/getting-started/introduction/)), but we overwrite bootstrap styles, and the changes are in a *global.scss*
- practical custom styles (that are NOT bootstrap overwrites) are defined in *helpers.html* (e.g. layout stylings for pages, forms…)
- long and/or complex stylings of components should have their own stylesheet such as tables *tables.scss* or modals…

SCSS files are organized from top to the bottom, the best practice, to make any changes, is to find the right spot in a document, for example

service-card.scss structure : .service-card; .card-title; .card-notification; .card-body;

We followed a few rules in SCSS, they make the code easier to read and to maintain. They're best practices, but they're not breaking the code.

```
 1 .card-title {
 2      display: flex; (flexbox)
 3      align-items: baseline; (flexbox)
 4      justify-content: space-between; (flexbox)
 5      width: 12em; (width)
 6      height: 12em; (height)
 7      margin-bottom: 0.5em; (margin)
 8      padding: 0; (padding)
 9      background: transparent; (other)
10      border: none; (other)
11  }
```

Not all declarations will always be present, but the main rule is to keep flexbox declarations at the top (flex, width, height, margin, padding), and another styling lower in a block.

Webpack

he code needs to be watched by Webpack, a new page must be added to the webpack.config.js, there is no need to add includes to the Webpack, when you open Webpack you'll see that each page is added there. You can copy the piece of code below, replace test with the name of the new HTML page and paste it to the webpack.config.js (currently line 102)

```
new HtmlWebPackPlugin({
   template: './pages/test.html',
   filename: './test.html',
 }),
```

webpack will be watching created file.

Git-ignore

In the project root there is a hidden file called **.git-ignore**, in this file are a list of files that will not be committed back to the source code repo when done.

```
dist
node_modules
.idea
yarn.lock
```

Tools to Manage Templates

installing on a debian system:

```
 curl -sS https://dl.yarnpkg.com/debian/pubkey.gpg | sudo apt-key add -
 echo "deb https://dl.yarnpkg.com/debian/ stable main" | sudo tee /etc/apt/sources.list.d/
 sudo apt update && sudo apt install yarn
```

Style Sheets

Introduction

We styled the project using SCSS. SCSS is the premier CSS extension language and it's completely compatible with all versions of CSS. SCSS provides the ability to write maintainable, reusable, and extensible code, ultimately meaning that code is faster to develop and better structured than plain CSS. The main advantage of using SCSS is the ability to use programming functionality such as variables, functions, inheritance, and operators. The benefits of these functionalities are that the code is displayed more logically, making it easier to manage. SCSS is not read by the browser, it must be compiled to CSS. There are various automated options to do that and we chose Webpack. Webpack processes SASS code and transforms it into CSS. We can give it a large number of files (as you can see in the SASS folder) and it generates a single file. To preview the project the files need to be compiled. Use yarn (package manager) to compile and preview the project.

Steps to run the Templates

1. open the terminal and install yarn (make sure you're in the projects directory): *yarn install*
2. run yarn: *yarn start*
3. if the process was successful you'll see *Compiled successfully.* in a terminal
4. now you can preview the project in the browser: * login page: http://localhost:9000/ * dashboard (my services): http://localhost:9000/dashboard.html * my profile: http://localhost:9000/profile.html
5. **you can make changes to the project and you'll see your updates in a browser after refresh**

    - if you can't see your changes, check if *yarn* is still running
    - you can restart the process by clicking *ctrl+c* (stops the process) and next *yarn start*
    - sometimes the browser needs a hard refresh: hold *shift* + click *refresh* (in a browser)
    - stop *yarn* when you're done work (*ctr* + *c*)

Create Account

Login

Description

Below is a template for the entrance page/login page with the NFLD treatment. This file can be found at the project root (pages/index.html). This template is the default page rendered when the user requests the fully qualified domain name without any sub context.

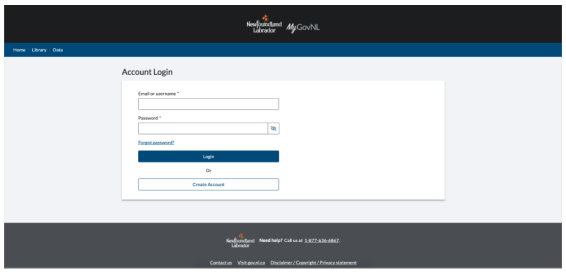
Controller Logic

General Information

The root controller has a fair bit of logic behind it other than rendering the login dialog below. Other logic it performs:

1. looks for query arguments such as *client_id* and *goto* parameters for single sign-flow.
2. Creates a tenant identifier so the correct templates are rendered.
3. Sets session identifiers
4. Looks for maintenance messages to be included in the ribbon at the top of the page.
5. Inspects for an active user and redirects to the dashboard page.
6. Looks for a cookie to that defines the prefered language, if none is found it looks are the header request for a declared language.

Login Dialogue



The login form on this page accepts two parameters that are sent to the login processor at /{lang}/login

```
{
    "usernameOrEmail": string,
    "password": string
}
```

This controller will either response with an error object, or direct the user to the dashboard page.

** Maintenance Messaging**

Also, when this page is rendered, there is a system call to the maintenance service looking for any upcoming alerts with title and collapsed details, If there is a response object the following object is available to the template, and the CitizenOne tagging will expose the html component that displays the maintenance ribbon.

```
{
    "maintenanceId": string,
    "title": string,
    "details": string,
    "startDate": string,
    "endDate": string
}
```

And makes available to the HTML view the following objects:

```
type MaintenanceMessage struct {
    MaintenanceId int    `json:"maintenanceId"`
```

```
    Title         string `json:"title"`
    Details       string `json:"details"`
    StartDate     string `json:"startDate"`
    EndDate       string `json:"endDate"`
}
```

The if the maintenance object is not empty, it will trigger the logic in the view to display the maintence banner: *(following code used as an example only)*

```
<div class="maintenance {{ if maintenanceMessage }} hidden {{ end }}">
    <div class="maintenanceTitle">{{ maintenanceMessage.Title }} </div>
    <div class="maintenanceMessage">{maintenanceMessage.Details</div>
</div>
```

Note

Todo: Create images for the collapsable maintence ribbon

Error Messaging

There is an error treatment to inform the user that the incorrect information was provided. This template is a re-usable template framed to paint errors in a globally common way. This error component is displayed by default just over the username input. This template can be styled to meet your standards. The default template appears like the following:

Note

Todo: Create images for the error states

and the HTML code that renders the error state (is also the success state) is as follows:

```
{{if .flash.success}}
    <div class="alert alert-success">
        {{.flash.success}}
    </div>
    {{end}}

    {{if or .errors .flash.error .e}}
    <div class="alert alert-danger mt-1">
        <h3 class="text-danger">Error, there was an issue processing that request</h3>
        <p>More details may be found below</p>
        {{ .flash.error }}
        <ul style="margin-top:10px;">

            {{range .errors}}
                <li>{{.}}</li>
            {{end}}
            {{range .e}}
                <li>{{.}}</li>
            {{end}}
        </ul>
    </div>
```

Language Toggle

If there is a requirement for multiple languages, this is satisifed by various ways:

1. The platform inpects for a set language cookie and sets the language accordingly

2. The platform inspects the URL route for the language parameter and sets accordingly
3. The platform looks at the header request and respects the language paremeter, and sets accordingly

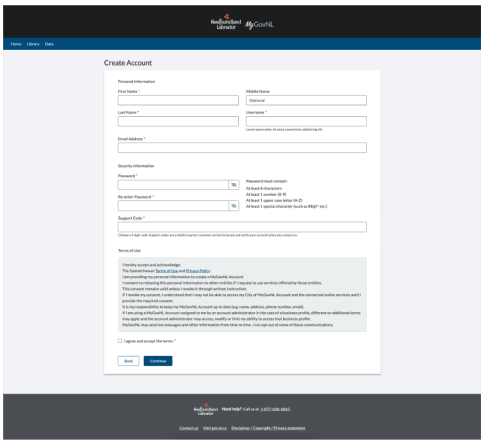
This toggle control can be a button, image, text link anything that can take a dynamic contruction of injecting a HTML view argument of {{ lang }

For example:

```
<div>
<a href="/{{ .lang }} /login"> {{ lang .lang }} </a>
</div>
```

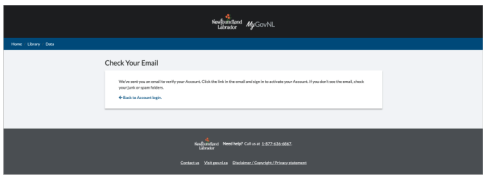
> ***Note***
>
> Todo: Create images for the language toggle

Create Account



When the user fills the form and clicks on “Continue” a confirmation email is being sent, and the user is being informed with “Check your email” screen

Check Your Email



Email being sent:

Hi Bea,

You recently created a MyGovNL Account.

Click this link to sign in and activate your MyGovNL Account.

If the above link is not clickable, copy and paste this link into your web browser’s address bar:

https://gov.nl.ca/#/activate/5hpfhp6qqk9875vlt36m8abc41c2f8du6pc67c74m5j6kth9

Regards,

Government of Newfoundland and Labrador
This message including attachments was sent to bea@vivvo.com for a specific recipient. If you are not the intended recipient, any redistribution or copying of this message is prohibited. If you have received this email in error, please let us know immediately, and delete this email.