Wizard Development Guide

Hunt Mortgage Portal Wizard Development Guide
Written by Jeremiah Smith

INTRODUCTION

This document is designed to be a developers guide to adding a new wizard to the Hunt Mortgage Portal system. It will cover the structure of the wizard layout which will include a description of the components, models, and service in the wizard layout. Additional it will include a section on the process of adding a new wizard to the layout.

OBJECTIVES

1. Cover Wizard Layout Structure
2. Learn the process of adding a new wizard

WIZARD LAYOUT STRUCTURE

FILE STRUCTURE

The file structure for the layout is designed to be in a hierarchical structure, with sections on the same level requiring the same level of access to sub sections. With in the wizard layout the top most folders are the service, models and wizard-container.

Service Directory

The service directory is meant to hold all the services for the wizard layout. It is separated into subdirectories, that make the project easier to navigate when trying to find the location of a service. These will include general and one each for the different classes of wizards (eg… freddie, fannie). The general directory is designed to hold any common service to wizard layout and the wizard class subdirectories are meant the hold the services that are for wizards in that class.

Models Directory

The models directory is meant to hold all the models for the wizard layout. It is separated into subdirectories which include general and one for each class of wizard (eg… freddie, fannie). The general directory is meant to hold all the models that are common in the wizard layout and the wizard class subdirectories are meant to hold the models that are designed for that class of wizard.

Wizard-Container Directory

The wizard-container directory hold all the components for the wizard layout. It is designed to be a wrapper for the components required to make the wizard layout work. This directory is structured as a hierarchy.

COMPONENTS

The wizard layout is sectioned off into subcomponents, which is designed to help with modularity. The overall layout is structured in the wizard-container component. This component contains all the sub components that make up the wizard layout.

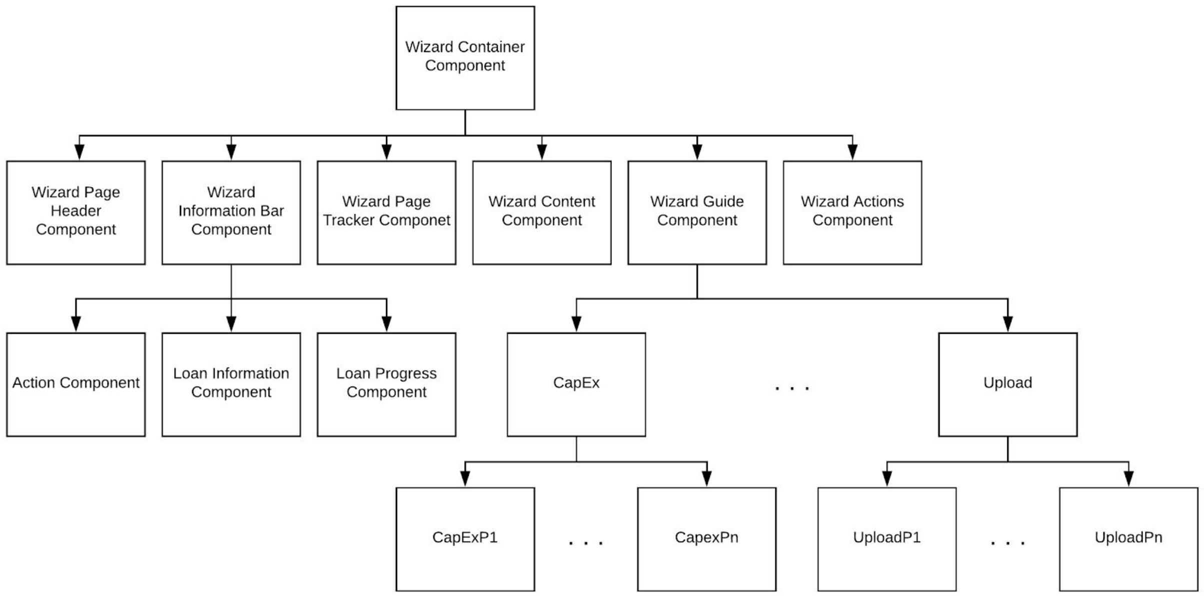

Image 1: Component Tree

Wizard Container Component

This component is the overall container component for the wizard layout. The primary responsibility of the component to to gather and distribute the need information to the subcomponents. Additionally it handles the layout of the components.

Wizard Actions Component

This component handles the actions for the wizards. These include changing pages, for wizards with more than one page, handling the submission process, and the review and sign process. Currently the upload wizard has the buttons in the wizard and this component will not show for the upload wizard.

Wizard Guide Component

The wizard guide component is a miniature checklist within the wizard. The guide provides all the checklist items for the current loan, highlight the current checklist item. This list is sorted by group and uses the Angular Material Expansion Panel.

Wizard Information Bar

The wizard information bar component displays all the current loan checklist information. This component is composed of three subcomponents guided action component, loan information component, and loan progress component.

Guided Action Component

The guided action component handles the actions while the user is being guided or not being guided. If the user is being guided then the component displays the Skip this Item and Exit Guide buttons, if not then the component display the my checklist mutton.

Loan Information Component

The loan information component displays the current loan information. This information includes the loan name, the checklist section name, and the checklist description.

Loan Progress Component

The loan progress component displays the overall progress that the user has mad on their checklist. The information displayed is the outstanding items and a progress bar indicating how many checklist items have been completed.

Wizard Page Header Component

The wizard page header component displays the application header. The header includes the LeapOnline logo, help button and the user menu.

Wizard Page Tracker Component

The wizard page tracker component displays a button for each page of the currently loaded wizard, with the current page highlighted

Wizard Content Component

The wizard content component is the component responsible for displaying each wizard in the application. This component will have sub components for all the wizards.

SERVICES

The services for the wizards will be in the service directory in a sub directory for the service.

Wizard Service

This is the primary service for the wizard layout. Here you will find all of the communication methods that the wizard layout need to function. These methods include information from the Leaponline API as well as internal method needed for the wizard to function.

Wizard Specific Services

These services are specific to each wizard and contain all the required method for a wizard to function. The service will be named after the wizard. These include getting and sending data to the api as well as any internal service methods.

MODELS

Because of the nature of of a growing application and the possible need for future models for new wizards this section will primarily focus on the models needed for the wizard layout and not the ones for the individual wizards. These models include wizard information, wizard navigation, and wizard page.

Wizard Information

This model is designed to hold all the information required for the wizard.

```
export interface WizardInformation {
    WizardName: string;
    WizardRoute: string;
    WizardType: string;
    NumberOfPages: number;
    NextChecklistDetailId: number;
    PreviousChecklistDetailId: number;
    Pages: WizardPage[];
}
```

WizardName - The name that describes the wizard

WizardRoute - Tells the front end which file to use to save and which version of the report to use

WizardType -

NumberOfPages - This is the number of pages total in the wizard

NextChecklistDetailId - This is used in guided mode to tell what wizard to navigate to next

PreviousChecklistDetailId - This used in guided mode to tell what wizard to navigate to when going back

Pages - This is the array of wizard pages associated with the wizard

Wizard Navigation

This model is designed to map the information from the call to the api

```
export interface WizardNavigation {
   ChecklistDetailId: number;
   WizardRoute: string;
   PriorChecklistDetailId: number;
   NextChecklistDetailId: number;
}
```

ChecklistDetailId - The Checklist Detail Id assiociated with the current wizard

WizardRoute - Tells the front end which file to use to save and which version of the report to use.

NumberOfPages - This is the number of pages total in the wizard.

NextChecklistDetailId - This is used in

guided mode to tell what wizard to navigate to next.

PriorChecklistDetailId - This used in guided mode to tell what wizard to navigate to when going back.

Wizard Page

This model is designed to hold all the information required for the wizard page.

```
export interface WizardPage {
   PageTitle: string;
   PageNumber: number;
   NextButtonEvent: string;
   PreviousButtonEvent: string;
}
```

PageTitle - The title of the page

PageNumbe - This is the number of the page

NextButtonEvent - This is the event that is emitted when the nxt button is click, this is also the text on the button

PreviousButtonEvent - This is the event that is emitted when the previous button is clicked, this is also the text for the button.

Guide Mode Status

This model is designed to hold the information returned from the Guided API Check

```
export interface GuideModeStatus{
  Guided: boolean;
}
```

Guided - is the user in guided mode.

Wizard Interface

This is the primary interface that insures the required functions and members are implemented within the base wizard component

```
export interface WizardInterface {

   wizardInformation: WizardInformation;

   initWizardInformation();
   handleAction(actionToPerform:
string);

}
```

wizardInformation- base information for the wizard

initWizardInformation - initialization function for the wizard

handleAction - function used to handle all the actions for the wizard

ROUTE STRUCTURE

The route structure is how the wizard layout tells what wizard to display and what information display in that wizard. This is accomplished with the data provided in the route.

Wizard Route

\newwizard\{loanid}\{checklistdetailid}\{WizardRoute}

Ex.

\newwizard\8529\125\CreditAuth

Note: newwizard will be changed

COMMUNICATION STRUCTURE

The primary communication between the components of the wizard layout is handled through the wizard service via behavior subjects.

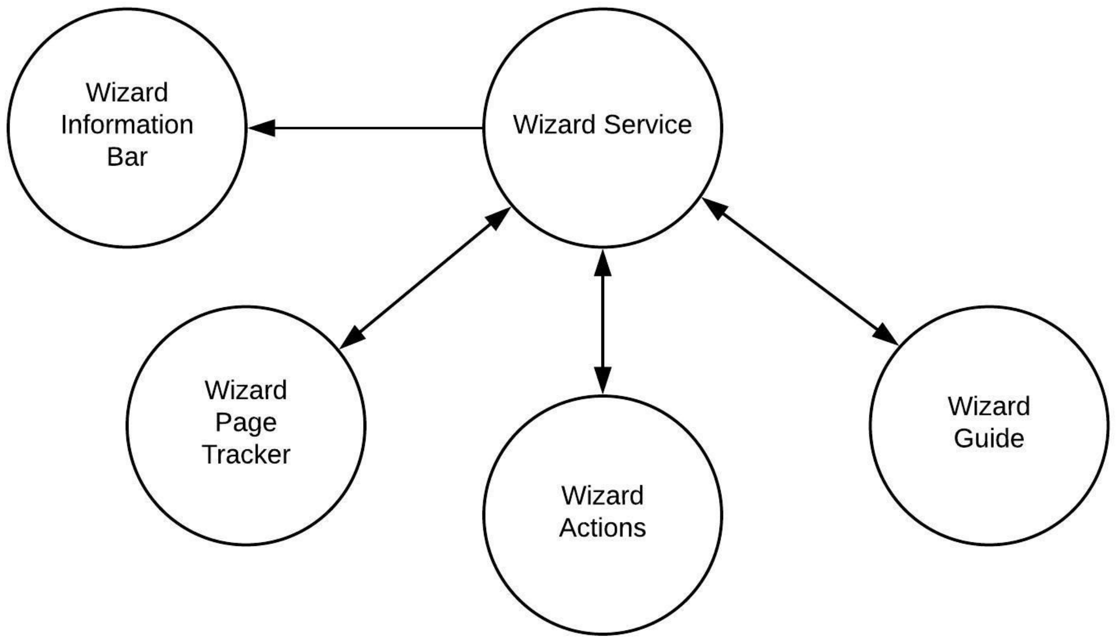

Image 2: Communication Graph

NEW WIZARD ADDITION PROCESS

Steps to add a new wizard

Step 1: Add Component for the wizard into the wizard content component

Part 1: Generating the main wizard component

The first part in adding a new wizard to the portal is to create the base wizard component to the wizard content. This can be done by running the following command from the root directory of the project.

```
ng generate component wizard-layout/wizard-content/wizards/{new-wizard-name}
```

This command will add the base wizard components {new-wizard-name}.html | ts | scss | spec.ts is the folder {new-wizard-name} under the wizards folder in wizard-content directory.

Part 2: Generating the wizard page components

The next part of the process is to create the page/s component for the wizard. This will be inline with the number of pages in the wizard. This can be done by running the following command from the root directory of the project for each page in the wizard.

```
ng generate component
wizard-layout/wizard-content/wizards/{new-wizard-name}/{new-wizard-name-pagen}
Where n is the page number
```

This command will add the base wizard components {new-wizard-name-pagen}.html | ts | scss | spec.ts is the folder {new-wizard-name-pagen} under the {new-wizard-name} folder.

Part 3: Add Implement the WizardInterface to component class

After this the next part is to make sure that the base wizard component *implements* the WizardInterface.

```
export class CreditAuthComponent implements OnInit, WizardInterface {
       ....
}
```

This will insure that the required functions and members are implemented.

The members include:

```
wizardInformation: WizardInformation;
```

The functions include:

```
initWizardInformation() { ... }

handleAction(actionToPerform: string) { ... }
```

Part 4: Implement the initWizardInformation function

After adding the implements WizardInterface the process of populating these functions is next. First is the initWizardInformation function. This function is responsible for building all the information needed for the wizard to function.

```
initWizardInformation() {
   this.wizardService.getSpecificWizard(this.loanId,
this.currentChecklistDetailId).subscribe(
      response => {
        // build the wizard infromation here
        this.wizardInformation = {
          WizardName: response.WizardRoute,
          WizardRoute: response.WizardRoute,
          WizardType: 'Needed?',
          NumberOfPages: 2,
          NextChecklistDetailId: response.NextChecklistDetailId,
          PreviousChecklistDetailId: response.PriorChecklistDetailId,
          Pages: [
            {
              PageTitle: 'Overview',
              PageNumber: 1,
              PreviousButtonEvent: 'PreviousWizard',
              NextButtonEvent: 'Next',
            },
            {
              PageTitle: 'Review',
              PageNumber: 2,
              PreviousButtonEvent: 'Previous',
              NextButtonEvent: 'Submit'
            },
          ]
        };
        // set the wizard infromation in the wizard service
        this.wizardService.checkBeingGuided(this.loanId).subscribe(
          guidedCheck => {
            this.wizardService.setBeingGuided(guidedCheck.Guided);
          }
        );

        this.wizardService.setCurrentWizardInformation(this.wizardInformation);
        this.wizardService.setWizardInformationStillLoading(false);

      },
      error => {
        console.log('Error getting Wizard Navigation Information: ', error);
      }
    );
  }
```

The process of initializing the wizard happens when the wizard loads on the screen. There are a few things to take note of in the initialize function:

1. The first part is a call to get the specific wizard information from the server. This information includes: Wizard Route, Previous, and Next Checklist Detail Id.

2. Number of Pages is the total pages in the wizard and should match the size of the Pages array.

3. Pages Array is the page structure of the pages in the wizard. Things to note in this structure include:
    a. PageTitle is what shows up on the buttons in the wizard page tracker component.
    b. PreviousButtonEnvent is the name of the event that will be handled in the handleAction function.
    c. NextButtonEvent is the name of the event that will be handled in the handleAction function as well as the name that appears on the next button in the wizard action component.

Part 5: Implement the handleHaction function

Next in the process is to implement the handleAction function. This functions is responsible for processing any actions the wizard needs to perform for the next and previous events in the pages array.

```
 handleAction(actionToPerform: string) {
    switch (actionToPerform) {
      case 'Submit':
      this.wizardService.goToNextWizard(this.loanId);
      break;

      case 'Next':
      this.saveData();
```

```
        this.wizardService.goToNextPage();
        break;

        case 'Previous':
        this.wizardService.goToPreviousPage();
        break;

        case 'PreviousWizard':
        this.wizardService.goToPreviousWizard(this.loanId)
        break;

        default:
        break;
      }
    }
```

Things to note in this function include:

1. There should be a case for each different previous and next button events in the Pages array in the wizard information data structure.

2. Any specific actions that the wizard needs to take should be handled in these cases.

Step 2: Add the specific service for the wizard to the service/name-of-wizard folder.

If the wizard needs a service to handle specific wizard actions and information retrieval, most will, add that service in this step with the following command:

```
ng generate service wizard-layout/service/{new-wizard-name}
```

This will add the service files {new-wizard-name}.ts | .spec.ts to the {new-wizard-name} folder in the service directory.

Step 3: Add wizard specific models to the model/wizard-type folder.

If the wizard needs any specific models add those models in this step.

Step 4: Modify the wizard-content template and component file to add the wizard

Part 1: Edit the wizard content template file

After all the components are created, the wizard content template file will need to be edited to add the new wizard to the list. Add the wizard component tag to the switch case with the test being the Wizard Route.

```
<mat-card>
   <mat-card-content [ngSwitch]="currentWizard">
       <app-credit-auth [loanId]="loanId" *ngSwitchCase="'CreditAuth'">
       </app-credit-auth>
       <!-- Add New Wizards Here -->

       <!-- Add Upload as default wizard to load -->
       <app-upload [loanId]="loanId" *ngSwitchDefault></app-upload>

</mat-card-content>
</mat-card>
```

Part 2: Edit the wizard content component file

After editing the template file next edit the component file. This is a temporary measure until all current wizards have been migrated to the new layout.

```
// in place until all wizards are converted to new layout
 private loadWizard(wizardToLoad: string) {
   switch (wizardToLoad) {
     case 'CreditAuth':
     case 'Upload':
     // Add new wizards here
     this.currentWizard = wizardToLoad;
     break;

     default:
     this.router.navigate(['wizard', 'landing', this.loanId]);
   }

 }
```

Wizard Complete