## **E-Claims Application**

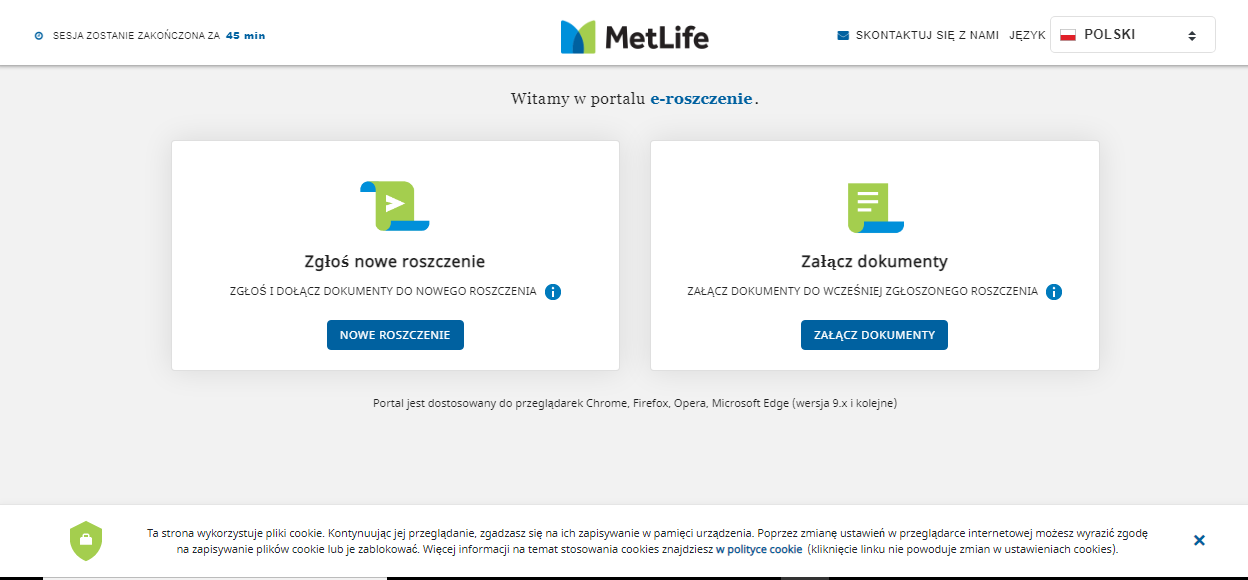
# **Abstract**

This document contains the Functional and Architectural design of the e-Claims, which is developed by CTS development team. The architecture satisﬁes the requirements in the Software Requirements Document (SRD).

# **Functional Flow**

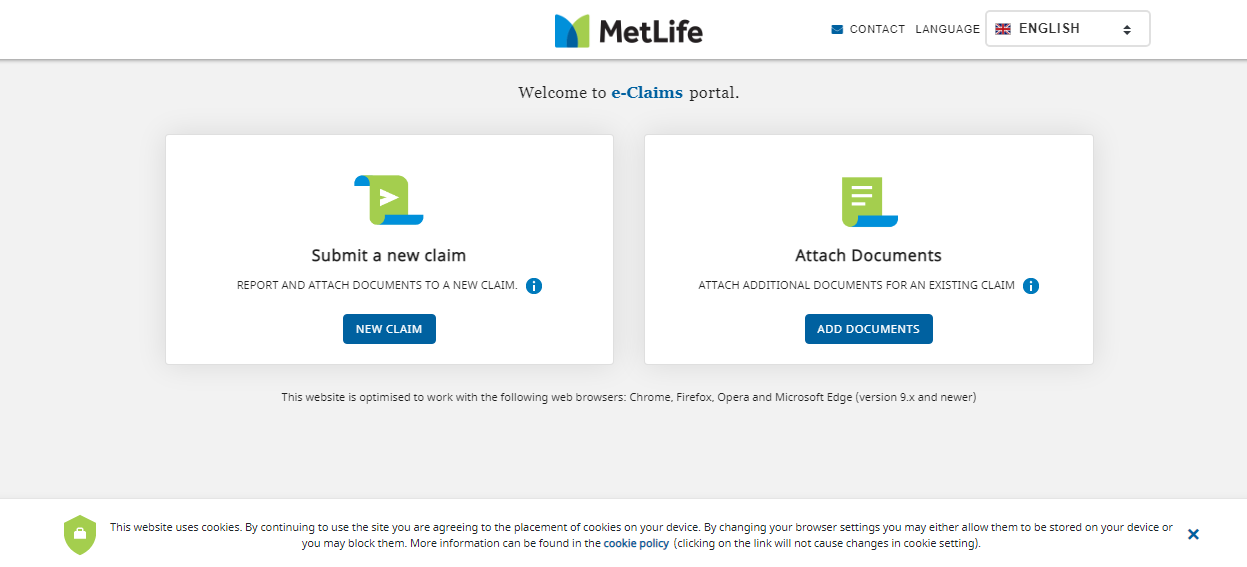
**Landing page:**

* Based on the Source of origin from the response system will be redirected to the landing screen
* O – Normal flow – landing screen - http://qa.eclaim.metlife.pl/
* C – Call center flow – Submit new claim
* B – broker or Agent flow – landing screen



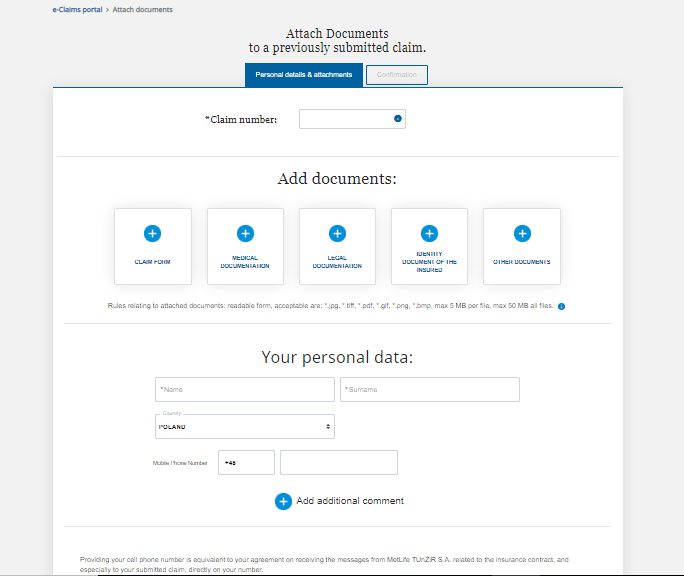
**Figure 1:** Polish Version of Landing Screen

* Top of the screen we have language translation select box. We can choose the language for the application.

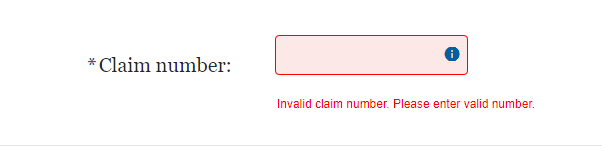


**Figure 2:** English Version of Landing Screen

* Clicking on Existing application will redirect the user to existing application where we can attach documents to already submitted claim and update some limited data for the claim.

 **Figure 3:** Existing application

* Claim number: To update the existing claim we need to enter the already submitted claim number.

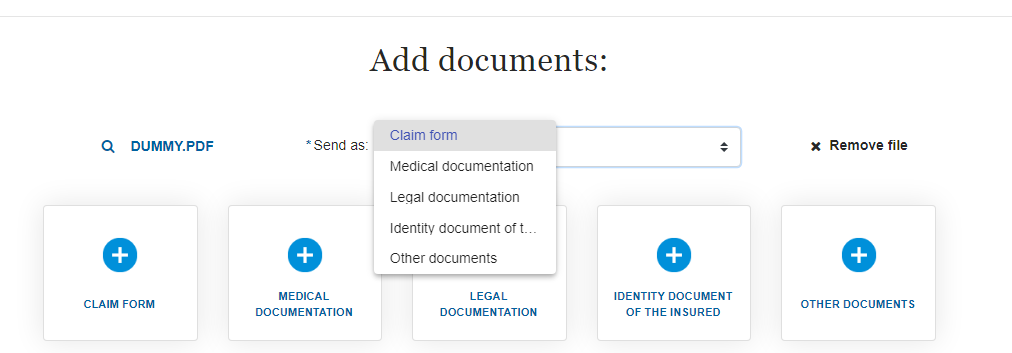


**Figure 4:** Claim Number Field

* If user missed to fill the claim number user will prompt to fill by showing the error message.

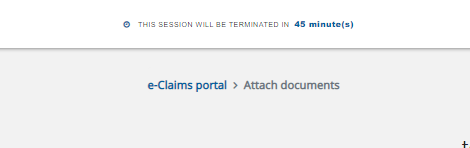
**Attach documents**:

* Clicking on the type of the document user going to submit will open the browse option to select the document from local.
* After upload the document user can change the type of the document by select the Send as select box options.



**Figure 5:** Document list in attach document section

* Clicking on breadcrumbs will redirect the user to the respected page.



**Figure 6:** Breadcrumbs

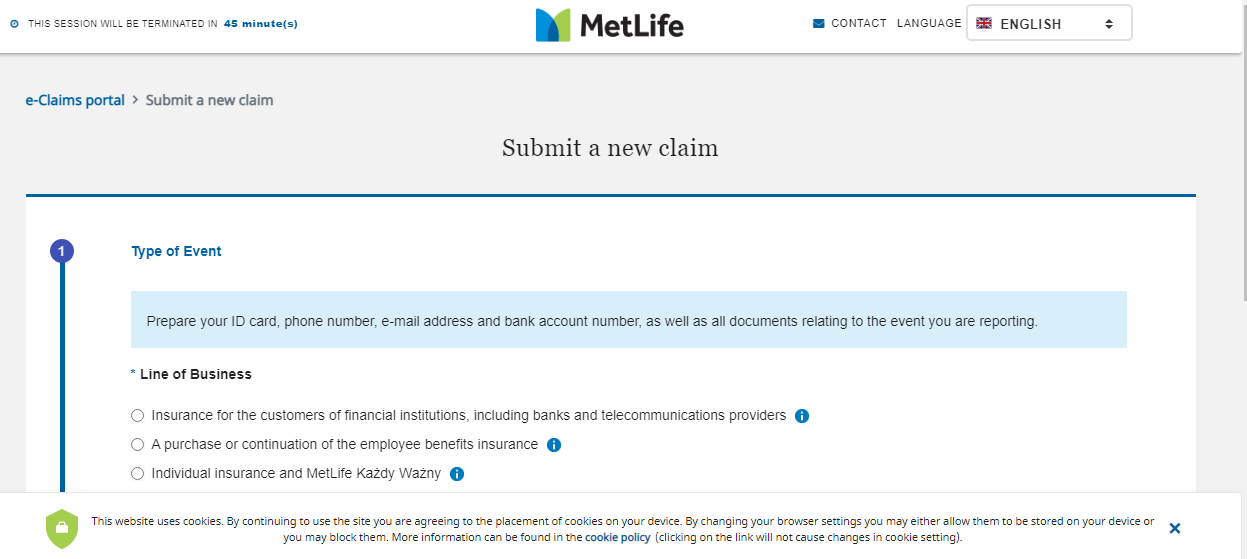
* Clicking on contact link will redirect the user to METLIFE official page.



**Figure 7:** Contact and Language translation

**New Claim:**

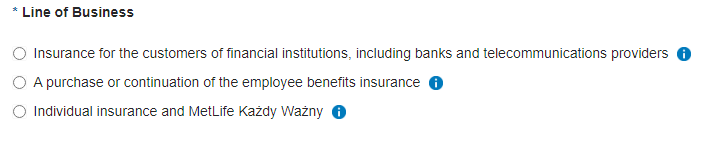
* Clicking on New Claim button will redirect he user to New Claim page where user can submit a new claim.



**Figure 8:** New Claim – Type of events

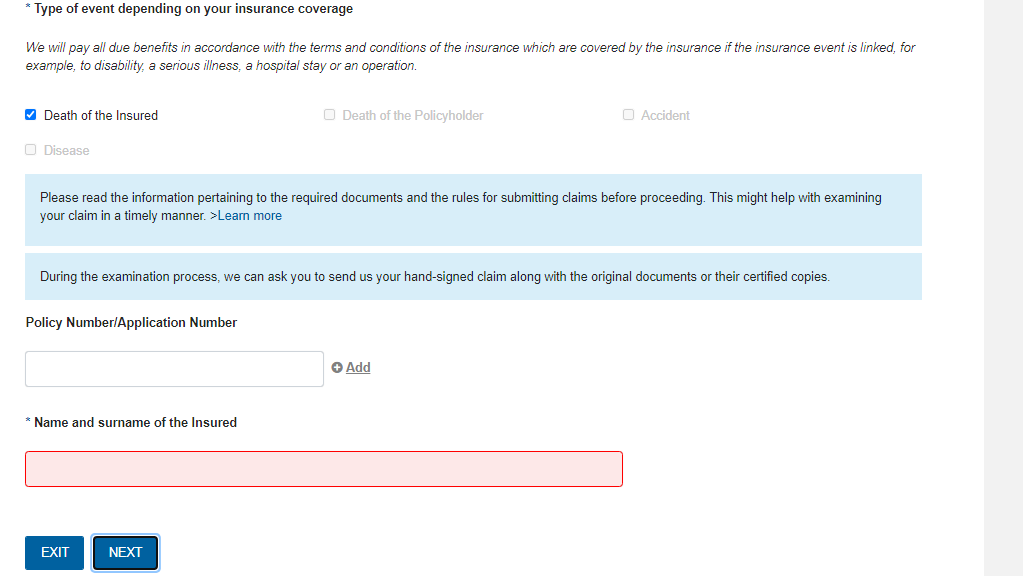
**Line of business will be shown based on lobsRendered:**

* "FI|Grp|Indv|Pen" from server response



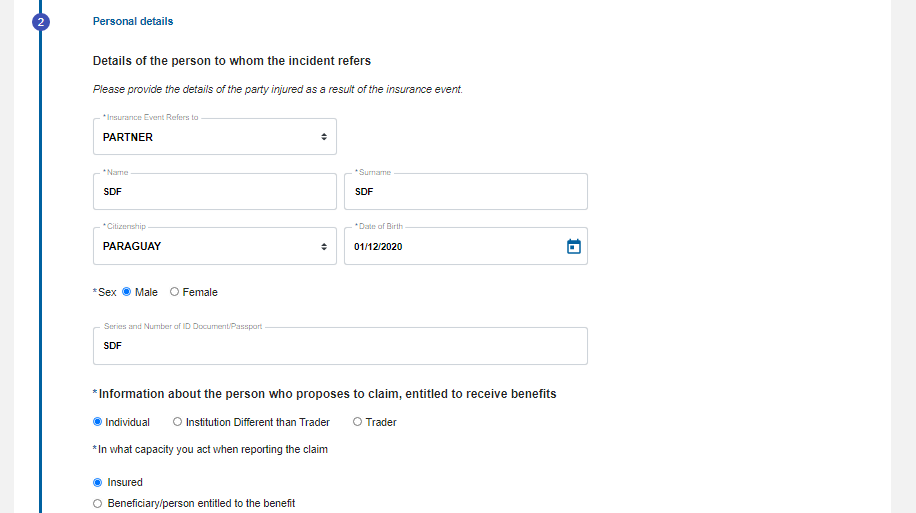
**Figure 9:** Line of business

Based on the type of event LOB selection application will show the Insurance coverage apply for that LOB.



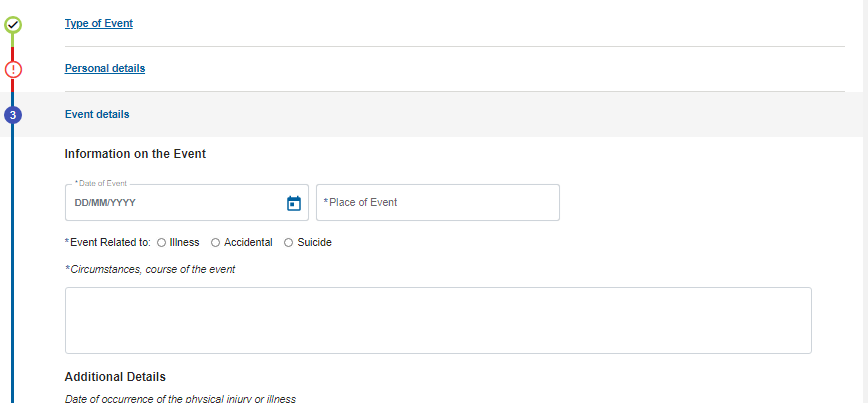
**Figure 10:** New Claim – Type of events – Insurance coverage

* After filing the type of event form and click on next will take you to the next stepper personal Details. If any error in the form it would not allow user to move next step.



**Figure 11:** New Claim – Personal Details

* In personal details screen if any errors in the form it would not stop the user to move next step. Instead of that it will show the error icon in the stepper in red color.



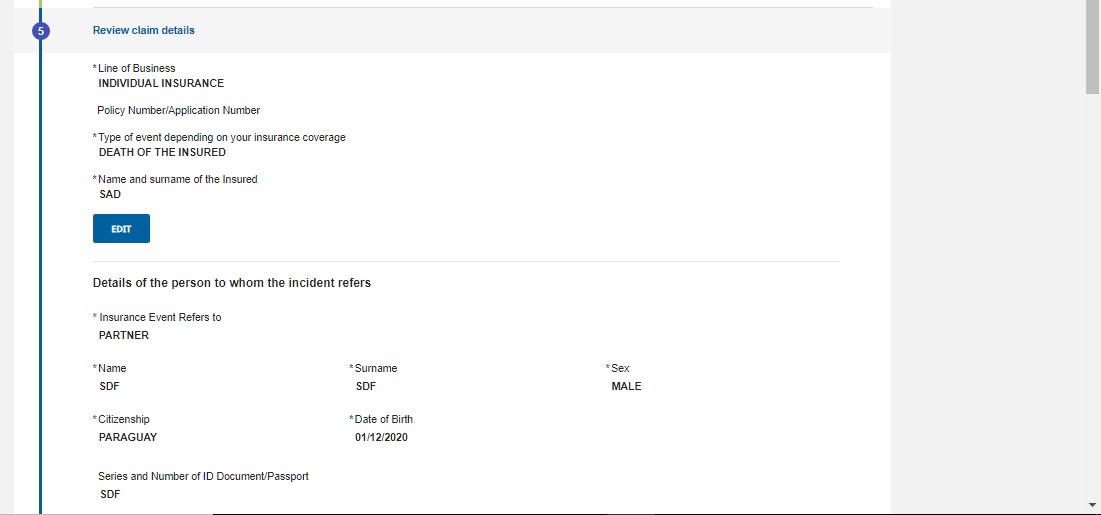
**Figure 12:** New Claim – Event Details

* Based on the Flow type, LOB and Insurance coverage selection attachments section will be shown with the mandatory documents and applicable documents to be uploaded as below.



**Figure 13:** New Claim – Attachments

* By clicking next button from attachment section user will be navigated to the review section, where user can check the data he/she filled.

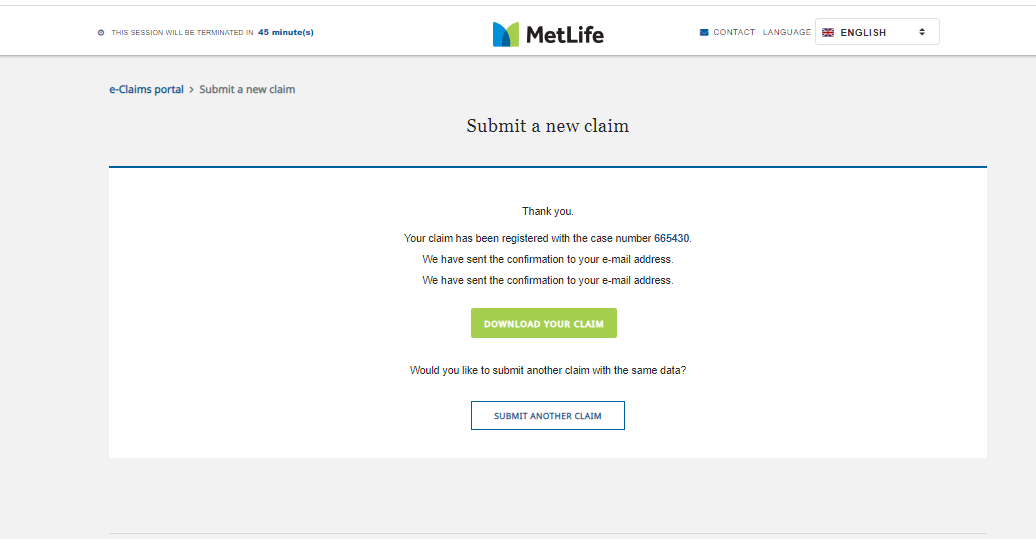


**Figure 13:** New Claim – Review Section

* If the user wants to change the data, he can click the edit button. It will take him to the respected section and user can fill the updated data and submit again.

**Confirmation Screen:**

* Once all the checklist and mandatory fields filled without any errors. The form will be submitted and user will be redirected to Confirmation screen.
* Here user can download the claim details.
* Also if the user wants to submit the claim which is more or less same to the previous claim, he can click on the “Submit Another Claim” button. It will redirect the user to the last claim page which he submitted before with data. User can modify the data and submit it again.



**Figure 14:** New Claim – Confirmation Screen

# **Technical Flow**

**Technology used in UI development:**

This section describes the technologies that are used in the development of the UI-library. These technologies are included in the UI-library’s source code. The distributed UI-library package includes the libraries, documentation application and their dependencies.

* Angular 8
* Bootstrap 4
* Angular Material
* Node Js

## **Angular 8**

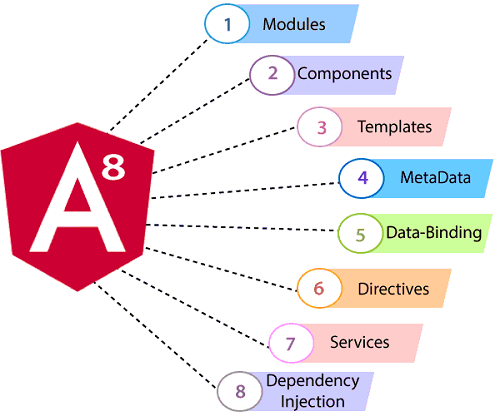
Angular 8 is a platform and a framework which is used to build client applications in HTML and TypeScript.

Angular 8 is written in TypeScript. It implements core and optional functionality as a set of TypeScript libraries that you can import into your apps.

The basic building blocks of an Angular application are NgModules, which provide a compilation context for components. NgModules collect related code into functional sets; an Angular app is defined by a set of NgModules. An app always has at least a root module that enables bootstrapping, and typically has many more feature modules.

Angular 8 Framework which allows to create complex, customizable, modern, responsive and user friendly web applications.

**Key parts of Angular 8 Architecture:**



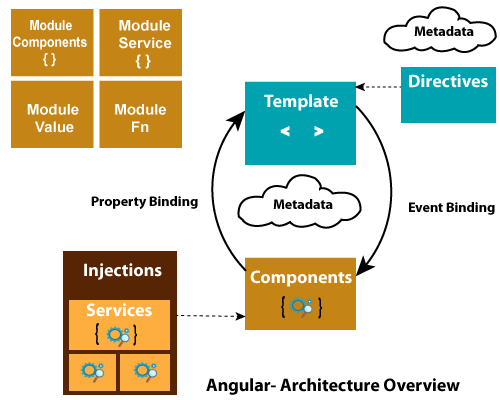
**Figure 14:** Angular – Key parts

In Angular 8, Components and services both are simply classes with decorators that mark their types and provide metadata which guide Angular to do things.

Every Angular application always has at least one component known as root component that connects a page hierarchy with page DOM. Each component defines a class that contains application data and logic, and is associated with an HTML template that defines a view to be displayed in a target environment.

**Metadata of Component class**

* The metadata for a component class associates it with a template that defines a view. A template combines ordinary HTML with Angular directives and binding markup that allow Angular to modify the HTML before rendering it for display.
* The metadata for a service class provides the information Angular needs to make it available to components through dependency injection (DI).



**Figure 16:** Angular Architecture

**Bootstrap and Angular Material:**

Bootstrap is a free and open-source front-end library for creating websites and web applications. It contains HTML- and CSS-based design templates for typography, forms, buttons, navigation and other interface components, as well as optional JavaScript extensions. It aims to ease the development of dynamic websites and web applications. There are several more alternatives for the Bootstrap UI but the main known competitor is the Foundation framework.

Bootstrap contains boiler-plated code that can be used by a developer to create out-of-the-box user interface elements. In this way, Bootstrap improves rapid development.

With Angular 8 we will use Material UI. Angular Material is a set of modern UI components designed by the Angular team and based on Google’s Material design specification. Google describes Material design as a set of guidelines, icons, and components that combine to create a unified user experience across platforms.

**Components:**

Components are the most basic building block of an UI in an Angular application. An Angular application is a tree of Angular components. A component controls a patch of screen called a view. CSyllabus modules will contain multiple components. If component is reusable, for example list of faculties, it will go to SharedModules, and will be reused throughout the whole application. Components we plan to have are AppComponent, FooterComponent, HeaderComponent, CoreComponent which has 2 children components ExplorerComponent and ComparatorComponent and those two each have SearchComponent and ResultComponent.

**Modules:**

Angular applications are modular and Angular has its own modularity system called NgModules. Every Angular application has at least one NgModule class, the root module, conventionally named AppModule. While the root module may be the only module in a small application, most applications have many more feature modules, each a cohesive block of code dedicated to an application domain, a workflow, or a closely related set of capabilities. CSyllabus will have multiple modules divided by features. For now we are going to have AppModule, CoreModule, ExistingClaimModule, AddNewClaim and additionally SharedModule.

**Services:**

As said, components should focus on presenting data and not fetch or save data directly. For those tasks we use services. Services are a great way to share information among classes. CSyllabus web application will have services which get data from API connected to database. Services used in application are CountriesService, CitiesService, UniversitiesService, FacultiesService, ProgramsService and CoursesService.

**CSS:**

Cascading Style Sheets is a style sheet language that defines the presentation of a markup language document such as an HTML document. CSS is commonly used in describing styles and layout of a web page. CSS rulesets can define for example the color, paddings and margins of an element with style properties. The elements that specific rulesets are applied are defined with selectors like classes or id:s. In larger projects using the plain old CSS can become increasingly cluttered and hard to maintain. CSS preprocessor languages like SASS and LESS have been designed to achieve better maintainability and add new language features like mixins and nesting of rules. Latest technologies have introduced new ways to achieve maintainability and separation of components. For example, with Vue.js library you can define isolated CSS rules for each component. The documented UI-library uses LESS as a CSS preprocessor.

**Typescript:**

Typescript is a superset language to ECMAScript 6 that compiles to Javascript which then can be executed in browsers. Typescript introduces types to Javascript which can enhance the workflow in larger projects and provide IDE features such as advanced autocompletion and refactoring in Javascript. Typescript is a viable choice to neglect timeconsuming issues that arise from dynamic languages. Types can prevent the application to end up in faulty behavior. Features and architecture can be described, documented and shared to other developers better with types. Typescript’s tooling can save a lot of time by preventing common mistakes caused by human error. For example the Typescript compiler can be very helpful when you define your types, classes and interfaces. Developers can get error reports before running the code and tools like intellisense present the APIs for you conveniently. When using Typescript you can interactively inspect library interfaces from your editor and take benefit from auto-completion.

**Node JS and NPM:**

NodeJS is a Javascript runtime environment used widely in web applications. NodeJS also includes its package-manager, npm (node package manager) which is the largest ecosystem of open source libraries and packages. NodeJS is designed for scalable network applications. It’s asynchronous, event-driven and has non-blocking I/O model. In our application, NodeJS is used to build the application and manage its packages and dependencies. NodeJS is also used as a platform of the development server. NodeJS also supports ES6 features.

**Source Mapping:**

When the source code is built, Typescript is converted to Javascript and LESS is converted to CSS. The source code is minified and other procedures are executed like concatenating the source code. The result might be impossible to read and to track down bugs. It is preferable to keep the client-side code readable and debuggable while developing the software. Source maps help with this as they map the code back to readable and debuggable form. Developer tools like Google Chrome dev tools will automatically use the generated source maps to map the code. The application uses gulp to make source maps and as result, the source code readable again and it is possible to debug the original Typescript source code and inspect the LESS stylesheets.

**Agile & Scrum:**

The development of the documentation library was done in parallel with the development of the UI-library. The most crucial part of any team work is communication. Our team had the traditional daily scrum meetings and retrospectives.

Scrum meetings emphasize the importance of continuous communication. In a daily scrum meeting each team member explains what they have done yesterday, what they are going to do today and if they have any problems. The meeting is supposed to be fast and brief, lasting around 15 minutes. This time is not dedicated to solve problems. After a scrum daily meeting every member of the team gains good understanding of what work has been done and what work remains. When a team member announces their commitment to a certain task the commitment is made for the team and the team will know the next day if the task is completed. This creates a certain sense of commitment for each member of the team.

In the scrum retrospective the team discusses the recently concluded sprint. Sprint stands for a period of time in which the team has agreed to complete certain set of tasks. In scrum retrospective team discusses on what went well during the latest spring cycle, what went wrong and what could be done differently to improve the efficiency and quality of the next sprint. In sprint retrospectives the team can provide feedback and discuss different ways to improve efficiency and quality of their work.

Development & Deployment During development of the documentation application and the UI-library there were certain best practices the team members followed. In addition to scrum daily meetings and retrospectives the team members also did code reviews to new features. Before new changes were added to the source code, other members reviewed the new code added for a new feature. This guaranteed to keep the established code quality and improved the common understanding of the source code when members could question and reason about the source code. Some small oversights and mistakes could already be caught in the code review process. When code is pushed to the repository the continuous integration system could spot regression bugs and errors when the developer was implementing a new feature. This continuous integration process guaranteed that these errors could be spotted and addressed early.

**Perspective and Insights:**

The team should be willing to deeply understand what the real customer needs from their product. It should be clear to understand who the customer is in the first place. It is a common misconception for a developer to solely implement features for their manager or product owner. But understanding the needs of the actual users of the product is the most crucial and challenging skill to acquire. In the perspective of the documentation application the customer is only indirectly the user of the business product, the application that uses the library. The primary customer of the documentation application are the developers and product owners that are using it on their product or are interested in doing so. It is crucial to truly understand who is the real customer and what their needs are.

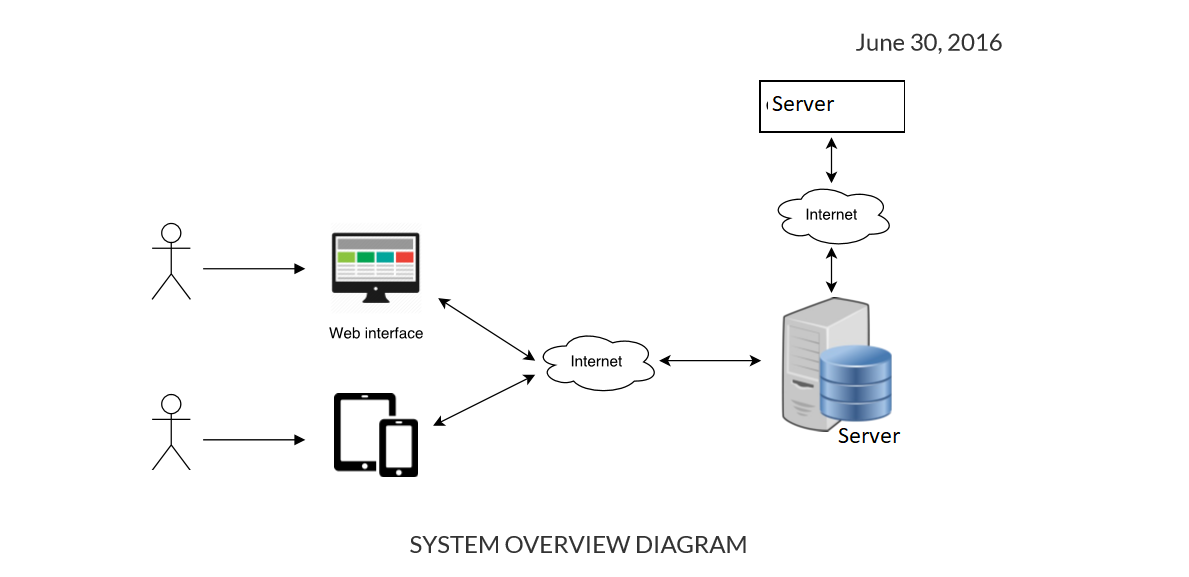
**Minifying & Concatenating**

For production there are various tasks to make the resulting package as lightweight as possible. These tasks include source code minification which removes all the unnecessary characters from the source code. This means that everything that doesn’t change the logic or functionality of the program and only exists for human readability, eg. white space characters, new line characters and comments will be removed. It will take care by angular

**Application Module Overview**

Application is sub divided into multiple modules, we will discuss in brief.

**System Overview**



**Figure 17:** System Overview

**core directory is to put the core elements for the application**

**Header Component**

* Allows user to select Language for the application.
* Click on logo will redirect to home page.

**Footer Component**

* Close Toaster
* app-footer

**Page Not Found Component (View)**

* Page not found

**Side Nav list Component**

* In the mobile view menu will be in side bar component.

**Toaster**

* Toaster is comes in the footer component. We can close the toaster if we don’t want to be in the page.

**Constants**

* All the constants we are using in the application (value we don’t want to change) will be in constants file.

**Interceptor**

* Interceptor in core is used to change the configuration of the http service call. We can add or remove the parameters, loaders and response etc. Token is added via interceptor

**Update Claim Validation Model**

* updateClaimValidationModel for set model for personal details component.ts in existing claim.
* Utils -> validators.ts for validation related change

**Utils Validator**

* validators.ts for validation related change.

**Auth Service**

* Auth service is the entry point for the application

**Getter Setter Service**

* setterservice.ts is to set and get general data.

**Preload Strategy Service**

* Preload strategy is for download all the dependencies once the initial page is ready.

**Add new Claim Common Service**

* Add-new-claim-common service is for common functionality for reuse in add-new claim.

**Html Structure Diagram** :

**Landing page for normal flow:**

HEADER Component

Router-outlet where the routing is happening

FOOTER Component

**Figure 17:** Landing Page component Structure

**Existing Application:**

Header

Existing application component

Angular material tab

Personal details component

Documents attachment component

App-confirmation component

Footer

**Figure 17:** Existing application Component Structure

**New Claim Application:**

Header

New application component

Personal Details component

Type of event component

Personal Details component

Event Details component

Attach Document Component

Review Component

App-confirmation component

Footer

**Figure 18:** New Claim Component Structure

**Personal Details Component:**

Insurance Refers Sec Component

Benefit Sec Component

Correspondance Address Component

Residence Address Component

Form of Disbursement Component

**Figure 19:** Personal details Component Structure

**Event details:**

Additional Details

Event Information

Health Care Event

Health care family Doctor

**Figure 20:** Event Details Component Structure

**Review:**

Type of Event Review component

Personal Review Component

Benefit sec review

Form of disbursement review

Personal Review Html

Event Review

Event Info

Additional Sec

Health care doctor

HealthCare Review

**Figure 21:** Review Section Component Structure

**Http interaction flow:**

server

Request request

Token, Additional params, loader

Response response Response

Remove loader symbol

**Figure 22:** Http interaction flow

**Configurations:**

**Date Config:**

* Date parser in app.module.ts

CustomDatePickerAdapter

{

    provide: DateAdapter,

    useClass: MomentDateAdapter,

    deps: [MAT\_DATE\_LOCALE, MAT\_MOMENT\_DATE\_ADAPTER\_OPTIONS]

  },

    {

      provide: DateAdapter,

      useClass: CustomDatePickerAdapter

    },

    { provide: MAT\_DATE\_FORMATS, useValue: MAT\_MOMENT\_DATE\_FORMATS },

* App.component.ts: If we are going to add new languages we need to it inside the dateTranslate object. Based on the language selection it takes the date localizer.

dateLanguageChange(language) {

    let dateTranslate = {

      'pl\_en': 'en',

      'pl\_pl': 'pl',

      'ro\_ro': 'ro',

      'ro\_en': 'en'

    }

    this.dateAdapter.setLocale(dateTranslate[language]);

  }

  switchLang(language: string) {

langChangeCalls(language) {

**Translation Configurations:**

Translation factory is initialized from initTranslation Class. Need to add the languages whatever we are going to use in the application.

providers: [ TranslatePipe,SettingService, {

    'provide': APP\_INITIALIZER,

    'useFactory': initTranslation,

    'deps': [TranslateService],

    'multi': true

  }

export function initTranslation(translate: TranslateService, storage: GetterSetterService) {

  return () => {

    //  let defaultLang = storage.getSession('userData');

    // let defaultLang = JSON.parse(sessionStorage['userData']).defaultLanguage;

    translate.addLangs(['pl', 'pl\_en', 'ro', 'ro\_en']);

    //translate.use(defaultLang);

    return Promise.resolve();

  };

}

* Translation Json path and prefix and suffix can be given in the below format

export function httpTranslateLoader(http: HttpClient) {

 //return new TranslateHttpLoader(http,'URL');

//const prefix = "http://127.0.0.1:8887/i18n/";

//const suffix = ".json"

return new TranslateHttpLoader(http, './assets/i18n/','.json');

//  return new TranslateHttpLoader(http);

}

**Http Interceptor Configurations:**

* To us the interceptor we need to inject it in the module. For eg.( HttpConfigInterceptor)

  { provide: HTTP\_INTERCEPTORS, useClass: HttpConfigInterceptor, multi: true

**Session timeout:**

For session timeout configuration we are taking the time from the env.js.

const sessionTimeoutInMinutes = 45;

* To find the timeout we are using “@ng-idle/core” module.
* We can modify the timeout warning and timeout functionalities in App.component.ts file

import { Idle, DEFAULT\_INTERRUPTSOURCES } from '@ng-idle/core';

import { Keepalive } from '@ng-idle/keepalive';

    let sessionOut = ((window['\_\_env']['sessionTimeoutInMinutes']) + 1) \* 60;

    idle.setTimeout(sessionOut);

* Timeout warning

idle.onTimeoutWarning.subscribe((countdown, doCountdown) => {

* Timeout

idle.onTimeout.subscribe(() => {

      this.headerRef.timeLeft = 0;

**Title Configurations:**

* title config – app component.ts
* To add the new language title it needs to be added in “titleTranslate” object. Based on the translate dropdown selection in the header. Function call will be triggered and it sets the title dynamically.

titleLanguageChange(language) {

    let titleTranslate = {

      'pl\_en': 'eClaims',

      'pl\_pl': 'e-roszczenie',

      'ro\_ro': 'e-Claim',

      'ro\_en': 'e-Claim'

    }

    this.titleService.setTitle(titleTranslate[language]);

  }

langChangeCalls(language) {

    this.languageJsonChange(language);

    this.dateLanguageChange(language);

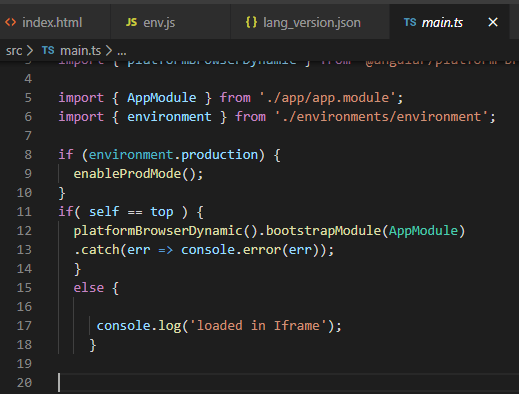
    sessionStorage.setItem("defaultLanguage", language);

    this.lang = language;

    this.titleLanguageChange(language);

  }

**Main.ts:**



* security changes we did. like password encoding..secured informations like pesel fields should be password, application should not load on iframe.
* Self object and parent object should be same otherwise application won’t get bootstrapped.

**Login:**

Initial call of the application to perform the business process is token generation

it will call auth service. Token generation call

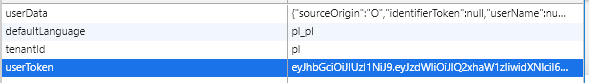
this.authService.tokenGeneration().subscribe((token) => {

* Response of token service will call the “user info” service to get the user details and redirect the user to the respected page.

this.authService.loginTo().subscribe

**Session Storage:**

Session storage for saving the data in session to make use of reload functionality. Default language is used to set the default language.



**Figure 23:** Session Storage Keys

**Tooltip Config:**

<div class="fdField d-inline-block">

                 <img id="labelfinLobText\_Popover" src="assets/img/infoIcon.png" class="popoverSubmit"

                 [matTooltip]="'finLobHint' | translate"

                    #tooltip="matTooltip"

                    (click)="tooltip.show()"/>

                </div>

**URL details to Auth token:**

|  |  |
| --- | --- |
| **URL** | http://URL:port/emea/api/v1/user/token/emea/api/v1/user/token |
| **Method** | POST |
| **Request** | {"userName":"eclaims","password":"eclaims"} |
| **Response** | {"userName":"eClaims","token":"eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJlQ2x0"} |

**URL details to User Info:**

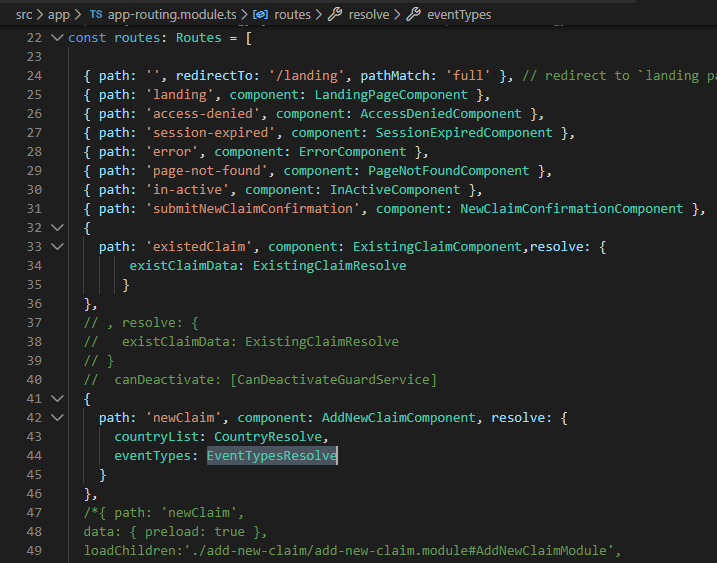
|  |  |
| --- | --- |
| **URL** | http://10.65.153.19:9080/emea/api/v1/ user/infoea/api/v1/user/token |
| **Method** | POST |
| **Request** | {"identifierToken":null,"sourceOrigin":null,"countryCode":"pl","encrytpedKeys":null} |
| **Response** | {"sourceOrigin":"O","identifierToken":null,"userName":null,"authToken":null,"isAdminUser":null,"submittedBy":"Customer","defaultLanguage":"pl","displayLanguages":"pl","landingPage":"landing","minFilesize":"5242880","maxFilesize":"52428800","clientId":null,"mobileDevice":false,"tablet":false} |

Based on the Source of origin from the response

* O - normal flow
* C – Call center flow - https://qa.eclaim.metlife.pl/?sourcekey=2de4f0048fbe84b96a9ae77801b5c9db5cbc0b01056e7f539f9c61c57820e9f2d97a66b1701d9b29a80596a211ca34604d5c5a9a4d19cc1cf1340344b85201a5451660dc98afc23947841120f6d28a131ccd91c419c3efe9642d615c7d69b2a7
* B – broker or Agent flow - https://qa.eclaim.metlife.pl/?countryCode=pl&sourcekey=2de4f0048fbe84b9611333e2745bba485cbc0b01056e7f539f9c61c57820e9f2d97a66b1701d9b29a80596a211ca3460a802a6cc08db563a843654eaa3d066c8beb3fc4b30f10aab933c8424a4e206cd
* minFilesize – refers to the single file maximum size
* maxFilesize – refers to multiple files maximum size should not greater than maxFilesize

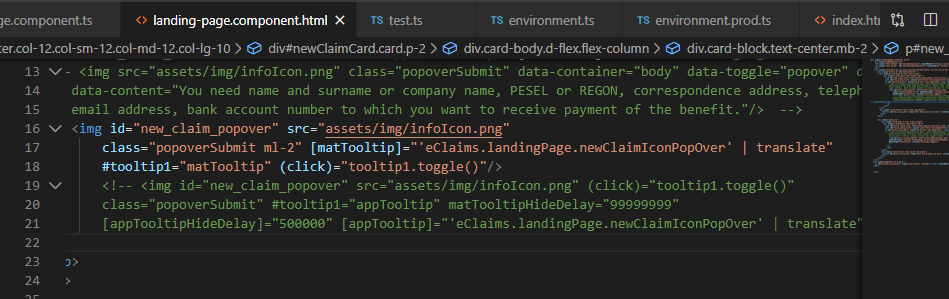
**Routing:**

Based on the response of <http://10.65.153.19:9080/emea/api/v1/rule/screen/info> application will show the steppers.



* Event types and country list should be loaded before the page load. So we resolve it while routing

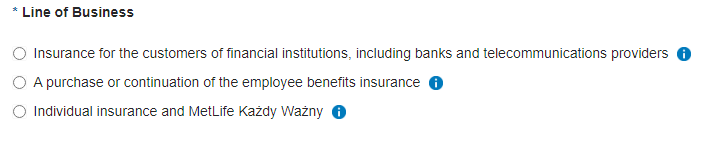
**Landing Component:**



* MatTooltip is used in landing page
* From landing page there are two major routes existing claim and new claim

**Type of Event:**

Line of business will be shown based on lobsRendered: "FI|Grp|Indv|Pen"



**Figure 24:** Line of Business

* Insurance refers to field will be based on the selection of LOB

eventType: "PrimaryPolicyHolderEvent|SpouseEvent|PartnerEvent|OtherNomineeEvent|ChildPrimaryPolicyHolderEvent"

* landingPage is where the user will be navigated.
* Window.scroll(0,0) or Window.scrollTo(0,0) used for scrolling the view to top of the screen.
* Enable disable claimtype based on secondary claim type selection.

disableCheckbox(index) {

    this.eventSecondaryValue = this.eventTypeClaimBusinessSec1

    const control = this.typeOfEventForm.controls.newclaimCode as FormArray;

   if(this.eventSecondaryValue && this.eventSecondaryValue.length){

    for (let i = 0; i < this.eventSecondaryValue.length; i++) {

* Type of event secondary claims type service call and disclaimer call.

  this.getEventTypeSec(secondaryClaimEvent, index);

      this.getDisclaimerSection(secondaryClaimEvent);

* Event type model

export class EventTypeModel {

  claimTypeSection: string = null;

  policyDetailsSection: string = null;

  policyNumber: string = null;

  employerDetails: string = null;

}

* Service call for getting secondary claim type

  getEventTypeSec(claim: string, index) {

    const i = index

    // const flagtest2="assets/mocks/TypeOfEventSecondary.json";

    // this.cService.getData(flagtest2)

    this.typeOfEventSecondary();

* ruleFileName: "Eclaims\_ComponentRenderBySourceOfOrigin.xls\_FIELD\_RENDER\_DETAILS"
* Based on the below service call response key we are loading the stepper.
* For broker flow
* breadCrumbsRendered: "TypeOfEvent|PersonalDetails"
* For Normal flow
* breadCrumbsRendered: "TypeOfEvent|PersonalDetails|EventDetails|Attachments|Review"
* Attach document file upload based on browse button
* browseButton: {renderFlag: true, mandatoryFlag: false, subQuestionFlag: false, fieldmaxlength: null,…}

**Adding input field common in components:**

<input-field [fdGroupName]="typeOfEventForm" [fdArrayName]="'policyNumber'" [fdControlName]="(i+'')" [fdName]="'policyNumber'" [fdId]="('policyNumber'+i)" [errMessage]="'errors.policyNumber'" [fdClassName]="'form-control w-100'" [fdTooltip]=" (this.product+'policyNumberTooltip'|translate) :null"></input-field>

**Input field component :**

<input matInput [type]="fdControlType" [formControlName]="fdControlName" [class]="fdClassName" [id]="fdId [name]="fdName" [attr.disabled]="fdDisabled" (focus)="onInputFocus($event.target)" (blur)="onInputBlur($event.target)" (change)="changeEvent($event) [restrict]="noRestrict?null:(fdRestrict?fdRestrict:(patternTypes[fdGroupName.get(fdControlName)?.restrict?.pattern]))" [textMask]="masking?masking:{mask: false} [maxlength]="fdMaxlength? fdMaxlength:fdGroupName.get(fdControlName)?.restrict?.maxlength" [matTooltip]="fdTooltip"/>

* textMask – add masking type
* restrict – restrict the character to type the values. It will take the value from the control
* fdGroupName - form group name
* fdArrayName - form array name
* fdControlName ­– form control name
* fdClassName – we can add different style for the input if we want dynamically
* errMessage – error message binding based on the field error
* fdTooltip – tooltip added dynamically

**Error Message Common component:**

<error-messages [control]="typeOfEventForm?.get('newclaim')" [errMessageStr]="'errors.newclaim'"></error-messages>

**Radio button:**

<radio-field class="col-12 col-md-4 col-sm-6 my-3" [fdGroupName]="benefitForm" [fdControlName]="'sexTrader'" [fdLabel]="[('eClaims.newClaim.personalDtails.sexMaleLabel' | translate),('eClaims.newClaim.personalDtails.sexFemaleLabel' | translate)]" [fdPLabel]="'eClaims.newClaim.personalDtails.sexLabel'| translate" [fdName]="'sexTrader'" [fdId]="['malesexTrader','femalesexTrader']" [fdValue]="['M','F']" [fdPClassName]="'form-check form-check-inline'" [fdClassName]="'mr-2'" [errMessage]="'errors.radio'"></radio-field>

* fdPLabel – parent label of the radio button
* fdPClassName – Parent of the label group style class. We can change the structure of the group using this dynamic class.
* fdValue – values for the radio buttons. It should be an array.
* fdLabel – labels for radio buttons. It should be an array of labels.

**Select common Component field:**

<select-field \*ngIf="showTaxResidencyCountryIndv" class="col-12 col-md-4 col-sm-6"

[fdGroupName]="benefitForm" [fdControlName]="'taxResidencyCountryIndv'"

[fdLabel]="'eClaims.newClaim.personalDtails.taxResidenceCountry'|translate"

[fdName]="'taxResidenceCountry'" [fdId]="'taxResidenceCountry'"

[fdClassName]="'form-group w-100 custom-select mb-0'"

(change)="changeTaxResidencyCountryIndv($event)">

<option \*ngFor="let cityOption of citizenshipOptions"

[value]="cityOption.countryCode">{{cityOption.countryName}}</option>

<option [value]="'other'">{{'eClaims.country.other' | translate}}

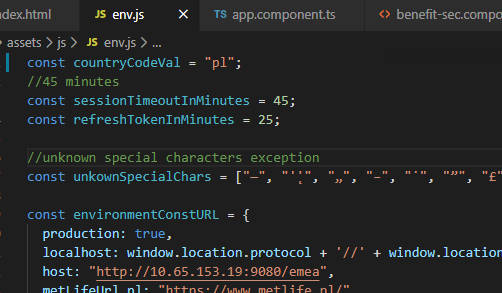
</option>

</select-field>

* Using ng-content we are replacing the options in the common component and other parameters are taken from the attributes.

**Env.js**

* Server ip add, remove and alter, service URL and method type, session out time setting.



* unkownSpecialChars in env.js for accepting special characters that is not in our keyboard
* unkownSpecialChars is used in src\app\shared\directives\patternRestrict.directive.ts

et unkownSpecialChars = (environment\_constants && environment\_constants.unkownSpecialChars) ?

      environment\_constants.unkownSpecialChars :[] ;

      for(let i=0;i<unkownSpecialChars.length;i++){

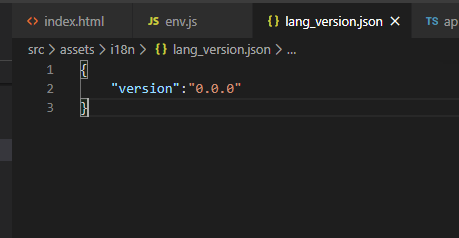
        if( $event.target.value.indexOf(unkownSpecialChars[i])>-1){

          $event.target.value=($event.target.value).replaceAll(unkownSpecialChars[i],'');

         }

      }

* For country based setup and configuration we added env.js. Everytime we don’t need to build separately for different countries and different servers like QA,UAT etc.. instead of that we can change the country name and host URL in env.js. it will load while the index page is loading.



* Language version needs to be change if u want to reflect the language json changes in the server immediately.

**Style.scss**

* Common style changes will be in style.css. We can override the default styles of the elements or css framework like bootstrap here.

**Angular.json**

* Configurations - > production - > file replacement -
* For local environment environment.ts file.
* For production environment.prod.ts file will take the configuration from env.js

<base href="./">

In index.html path where should all the folder configuration should take place

Index.html to make google translation disable

translate = "no"

* ng serve
* ng build --prod to production build
* index.html - > app-root(app-component)

mobile sidebar menu changes in

app-sidenav-list

**Images:**

SRC->assets->img to alter ,add or remove images and need to provide the path in the html img tag.

**Language Setup:**

SRC-> assets->I18 to add language json or alter the language text

As of now Polish, Polish English, Romania, Romania English language translation files are there.

If you want to change any text for the respected labels. We can take the label from the property files and search the same word in the json and replace it. If still you are not able to search the word, search for the English word (because roman and polish words may have some Unicode characters) which you want to replace in the pl\_en.json or ro\_en.json. and take the respected label and update it.

**Existing-claim:**

Any data that need to be resolved and needs to be used in existing claim page before routing are done in Existing-claim- resolve.ts

**Dialog box:**

* Common dialog boxes are created in dialog box directory.

**Directives:**

**Length restrict:**

* Using this directive we are restricting the user from input the characters more than the given length.

**Pattern restrict:**

* Using this directive we are restricting the user from input the characters which are not permitted.

**Services:**

* Create-form-services: Form related common services will be in this service. For eg. setFormValidation, markFormGroupTouched, clearFieldValidation, generateForm.

**Dialog services:**

* Dialog related manipulation will be in this service.

**File-preview service:**

* Logics for Uploaded file preview in the application will be in this service.

**http-common-service:**

Common Get, Post, Put methods will be in common services. We can add or alter the parameter in this services also. Generalized things for the server call can be done here. For eg. We can add or remove loader symbol here.

# **Add new Claim:**

* Add-new-claim.ts has 2 main components showing based on the flag “renderClaimSections”.
* Used mat-vertical-stepper to show the new claim sections in stepper mode.

[hasError]="typeOfEventSecError"

* To show the stepper error if something is not valid
* If any other new stepper needs to be added that will added here by Using “mat-step”.

<mat-step></mat-step>

* Below are the sub components in add-new-claim.html.

<app-type-of-events #typeOfEventRefComp></app-type-of-events>

<app-personal-details #personalDetailsRefComp></app-personal-details>

<app-event-details #eventDetailsComp></app-event-details>

 <attachments #attachmentsComp></attachments>

 <app-review-details (stepperEdit)='gotoStep($event)'></app-review-details>

  <new-claim-confirmation></new-claim-confirmation>

* Next button click have the validation of the respected form.

  <button mat-button class="btn-primary"

                  (click)="validateEventDetails('attachmentFormDiv')">{{'eClaims.newClaim.personalDtails.nextButtonLabel'| translate}}</button>

**Add-new-claim:**

  path: 'newClaim', component: AddNewClaimComponent, resolve: {

      countryList: CountryResolve,

      eventTypes: EventTypesResolve

    }

* Before loading the new claim page we need to load the event types which is the first tab user is going to view. Event types is taken from the service call and it is resolved in the routing part. Also country list is resolved before the page loading.
* Event types resolved from routing logic has been moved to add-new-claim.component.ts onInit() function
* this.newClaimService.triggerShowClaimCompApi()
* showing a new claim and its confirmation screen is based on the flag “showNewClaim”

gotoDownloadConfirmation(saveClaimsResponse) {

    this.showNewClaim = false;

    window.scroll(0,0);

    //this.newClaimService.setNewClaimResponse(saveClaimsResponse);

    this.newClaimService.updateSaveClaimResponseComponents(saveClaimsResponse);

    //this.router.navigate(['/submitNewClaimConfirmation'], saveClaimsResponse,);

  }

Once the claim is submitted successfully new claim section will be hide and Confirmation screen will be shown using “this.showNewClaim = false”

“updateSaveClaimResponseComponents” will save the response from save claim

* We are using the material stepper to show the improvement of form filling and showing the error icon and success icon based on the form validation.

stepperReset(){

  this.personalDetailsSecError = false;

  this.eventDetailsSecError = false;

  this.attachmentSecError = false;

  this.reviewSecError = false;

  if(this.stepper && this.stepper.reset){

    this.stepper.reset();

  }

}

* Some fields will be show and hide based on the other field values. We are storing the

 typeOfEventLobOldValue = "";

  primaryClaimEventOldValue = "";

  eventRelatedOldValue = '';

  insuranceReferOldValue = '';

  capcityOldValue = '';

  policeConductedInvestigationOldVal = '';

  employerPreAccidentRepOldVal = '';

  sportsClubPlayerOldValue = '';

* calledOnce is to check whether the oninit and initial service calls for the sections are called already. If it is called for each section calledOnce value will be true. Otherwise it will be false.
* Also if we changed the LOB or claimtype in typeofevent section we make the called once for other section as false.

if (event.previouslySelectedIndex == 0) {

      if ((this.typeOfEventLobOldValue != this.typeOfEventRefComp.typeOfEventForm.value.newclaim)

        || this.comparePrimaryClaimArr(this.primaryClaimEventOldValue, this.typeOfEventRefComp.typeOfEventForm.value.claimTypeCode)) {

        //  this.stepperReset();

          for (let i = 0; i < this.formDetails.length; i++) {

          this.formDetails[i].calledOnce = false;

        }

      }

    }

* If new value is differ from old value for the code value below will re initialize the attachment section.

 this.getSectionOldValue(event.selectedIndex);

    if (event.previouslySelectedIndex == 2 || event.previouslySelectedIndex == 1) {

      if ((this.eventDetailsComp && this.eventDetailsComp.eventDetailForm.value &&

        this.eventDetailsComp.eventDetailForm.value.eventInformationSection &&

        this.eventDetailsComp.eventDetailForm.value.eventInformationSection.eventRelatedTo != undefined &&

         this.eventRelatedOldValue != this.eventDetailsComp.eventDetailForm.value.eventInformationSection.eventRelatedTo) ||

        (this.personalDetailsRefComp.insuranceRefComp &&

          this.personalDetailsRefComp.insuranceRefComp.insureForm.value.insuranceEventRefersTo !== null &&

          this.insuranceReferOldValue !== this.personalDetailsRefComp.insuranceRefComp.insureForm.value.insuranceEventRefersTo) ||

        (this.personalDetailsRefComp.personalForm.value.entitledForBenefitsSection &&

          this.personalDetailsRefComp.personalForm.value.entitledForBenefitsSection.capacity != undefined && this.capcityOldValue !==

          this.personalDetailsRefComp.personalForm.value.entitledForBenefitsSection.capacity) ||

          this.questionarieStatus()) {

        if (this[formDetailClone.formRef][formDetailClone.initFunc] && formDetailClone.formRef == "attachmentsComp") {

          //  event.selectedIndex == 3

          // && event.previouslySelectedIndex == 0

          this[formDetailClone.formRef][formDetailClone.initFunc]();

        }

      }

* Based on the response atrribute breadCrumbsRendered we are removing the section details from the formDetails array.

        for (let i = this.formDetails.length - 1; i >= 0; i--) {

          if (response.breadCrumbsRendered.indexOf(this.formDetails[i].nameInResponse) == -1) {

            this.formDetails.splice(i, 1)

          }

        }

callAttachmentsFunc(generatedClaimNo){

  if (generatedClaimNo) {

    let attachmentReference: any = null;

    if (this.renderClaimSections.tab2AttachmentsSection.renderFlag) {

      attachmentReference = this.personalDetailsRefComp.attachmentsComp;

    } else {

      attachmentReference = this.attachmentsComp;

    }

    this.attachmentFiles(this.typeOfEventRefComp, attachmentReference, generatedClaimNo);

  }

}

  //generateAllForms(): create form group based on the validation response JSON

  //getSectionDetailsIfMandatory : return the sections which are mandatory

  //setSectionRenderValues() : change validation if type of event selection changed

formDetailsOrigin = [

    {

      "form": "typeOfEventForm",

      "initFunc": "formInit",

      "formRef": "typeOfEventRefComp",

      "submit": "formSubmit",

      "error": "typeOfEventSecError",

      "nameInResponse": "TypeOfEvent",

      "calledOnce": false

    },

    {

      "form": "personalForm",

      "initFunc": "formInit",

      "formRef": "personalDetailsRefComp",

      "submit": "formSubmit",

      "error": "personalDetailsSecError",

      "nameInResponse": "PersonalDetails",

      "calledOnce": false

    },

Below code will call the sub components forms submit function

  callSubCompFormSubmit() {

    let newClaimSubCompSubmit = true;

    let submitFormValid = true;

  for (let i = 0; i < this.formDetails.length; i++) {

      if (this[this.formDetails[i].formRef] && this[this.formDetails[i].formRef][this.formDetails[i].submit]) {

        let newClaimSubCompSubmit = this[this.formDetails[i].formRef][this.formDetails[i].submit]();

It will return the form status whether it is valid or not.

Next step buttons are in add-new-claim.html page and Below code will validate the form. If the form is valid then we will be moved to next section.

  validateTypeOfEvent(id) {

    this.typeOfEventRefComp.submitEventType();

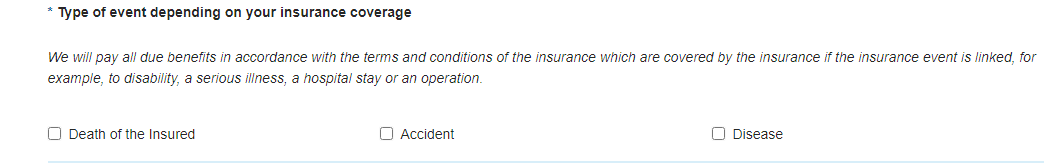
    if (this.typeOfEventRefComp.typeOfEventForm.valid) {

      this.nextStep(id);

    }

eventTypes() once we get event types values

Based on the \*Line of Business we will get the claim type code which will be translated from language json files.



{partner: "metlife", product: "Financial", claimType: "E240|E250|E270",…}

* 1. claimType: "E240|E250|E270"
  2. eventType: "PrimaryPolicyHolderEvent|SpouseEvent|PartnerEvent|OtherNomineeEvent|ChildPrimaryPolicyHolderEvent"
  3. ksession: null
  4. partner: "metlife"
  5. product: "Financial"
  6. ruleFileName: "Eclaims\_ProductClaimType.xls\_CLAIM\_DETAILS"
  7. sheetName: null

also another service call will get you the rules for claimtype, policy number, etc..

1. {ruleFileName: "Eclaims\_EventTypeComponentRender.xls\_CLAIMTYPE\_FIELD\_RENDER", sheetName: null,…}
   1. claimTypeSection: {renderFlag: true, mandatoryFlag: true, subQuestionFlag: false, fieldmaxlength: null,…}
   2. employerDetails: {renderFlag: true, mandatoryFlag: true, subQuestionFlag: false, fieldmaxlength: "40",…}
   3. ksession: null
   4. lob: "Financial"
   5. partner: "metlife"
   6. policyDetailsSection: {renderFlag: true, mandatoryFlag: false, subQuestionFlag: false, fieldmaxlength: null,…}
   7. policyNumber: {renderFlag: true, mandatoryFlag: false, subQuestionFlag: false, fieldmaxlength: "26",…}
      1. allowedDataType: ","
      2. fieldmaxlength: "26"
      3. fieldminlength: "0"
      4. mandatoryFlag: false
      5. renderFlag: true
      6. subQuestionFlag: false
   8. ruleFileName: "Eclaims\_EventTypeComponentRender.xls\_CLAIMTYPE\_FIELD\_RENDER"
   9. sheetName: null

**Personal-details.ts:**

We have separate pattern directive date field. So we added those fields in pattern exception.

  patternException = ['dobField', 'dateOfBirthIndv', 'bankaccountnumber'];

Below is the format for form manipulation

  formDetails = [

    {

      "form": "insurance",

      "initFunc": "formInit",

      "formRef": "insuranceRefComp",

      "submit": "formSubmit",

      "renderJson": null,

      "ruleFileIncludes": "InsuranceRefersTo"

    }

**Insurance section:**

copyFromEklient()

* get the values from server using client id and set it.
* Onchange insuranceEventRefersTO we will reset the attachment. We will give alert for that and empty the files upload.

attachmentReset()

 setInsuranceRefersVal(){

    this.renderClaimSections = this.newClaimService.getParamValue('renderClaimSections');

    let insuranceValue =  this.insureForm.get('insuranceEventRefersTo').value;

    let lobValue = this.newClaimService.getClaimData().newclaim;

    let claimTypeValue = this.newClaimService.getPrimaryClaimType();

    if ((lobValue == 'IndividualLife' || lobValue == 'IndividualLifeM')

    && (claimTypeValue.indexOf('E410') != -1)) {

    this.eventRefersToOptions = ['Insured'];

  } else if ((lobValue == 'IndividualLife' || lobValue == 'IndividualLifeM')

    && (claimTypeValue.indexOf('E420') != -1)) {

    this.eventRefersToOptions = ['PrimaryOwnerEvent'];

  }

  else {

    this.eventRefersToOptions = this.newClaimService.getParamValue('eventTypeList') ? (this.newClaimService.getParamValue('eventTypeList').split('|')) : [];

  }

  //IndividualLife IndividualLifeM E420 E410

blurPseiField(event) {

   if(this.peselOldValue!= this.insureForm.controls.cnpfield.value){

    let tenantId = sessionStorage.getItem('tenantId');

on focus out on pesel Id will trigger a api service call which will return the DOB and gender if the ID is valid. Else error message will be shown.

 this.peselOldValue = this.insureForm.controls.cnpfield.value;

old pesel value is stored in a variable because if the user focus out from the field if the field has value which is not changed then api service call will return the same value as before. To avoid the un necessary service call we used the old value.

If the DOB and gender fields are filled from the service response then 2 star symbols will be shown in the label.

  this.dateFromCNP = "\*";

                this.genderFromCNP = "\*"

explicitly change the gender field remove the 2nd star rom the lable. It is same for DOB field also.

changeGender(event) {

    this.genderFromCNP = '';

Other than Poland country selected as nationality would not get the pesel number field.

 changeNationality(event) {

based on the below conditions event refers to dropdown options are added.

  setInsuranceRefersVal(){

    if ((lobValue == 'IndividualLife' || lobValue == 'IndividualLifeM')

    && (claimTypeValue.indexOf('E410') != -1)) {

    this.eventRefersToOptions = ['Insured'];

  } else if ((lobValue == 'IndividualLife' || lobValue == 'IndividualLifeM')

    && (claimTypeValue.indexOf('E420') != -1)) {

    this.eventRefersToOptions = ['PrimaryOwnerEvent'];

  }

else {

    this.eventRefersToOptions = this.newClaimService.getParamValue('eventTypeList') ? (this.newClaimService.getParamValue('eventTypeList').split('|')) : [];

  }

Event type list is set from type-of-event.ts using below code

this.newClaimService.setParamValue('eventTypeList', data1.eventType);

“eventRefersTo” option list will be based on the lob selection

Once user select the “eventRefersTo” field and then he changed the lob. It will impact the option list also. So the user selected option might not be there in the list. If it is there then we are setting the same value. If it is not there then we are setting the null value.

setTimeout(() => {

  if(this.eventRefersToOptions.indexOf(insuranceValue)!=-1){

    this.insureForm.get('insuranceEventRefersTo').setValue(insuranceValue);

  }else{

    this.insureForm.get('insuranceEventRefersTo').setValue(null);

  }

Insurance refers to change onEventReferChange()

attachmentReset(){

  try{

    let fileInAttachment = this.dataService.getOption('uploadNewClaimList');

    if (fileInAttachment && fileInAttachment.fileUpload && fileInAttachment.fileUpload.length > 0) {

      this.dialogService.openDialog(AlertDialogComponent, { 'heading': this.translate.instant('eClaims.existingClaim.confirmationLabel'), 'body': this.translate.instant('reAttachDocument'), 'primaryButton': 'OK', 'fromTypeofevent': true });

    }

   } catch(e){

console.log("error",e);

   }

}

**Personal-details.model.ts:**

* All sub components of personal details sections form models will be under personal-details.model.ts

export class CorrespondenceAddressModel {

    copyDataFromEc:string = null;

    streetName: string = null;

    houseNumber: string = null;

    flatNumber: string = null;

    country: string = null;

    otherCountry:string = null;

    postalCode: string = null;

    town: string = null;

    postBox:string = null;

    mobileNumber: string = null;

    isdCode: string = null;

    email: string = null;

    preferredModeOfCommunication: string = null;

    addressAcceptanceCheck: string = null;

    county: string = null;

    zipCode: string = null;

    city: string = null;

    block: string = null;

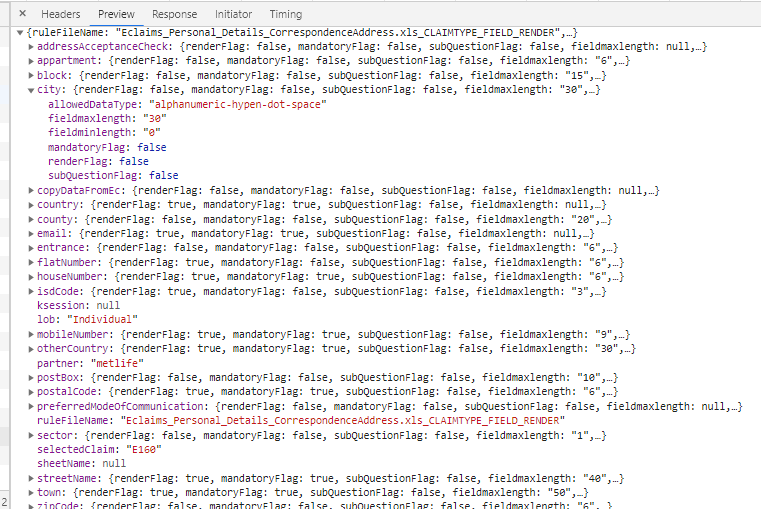
    entrance: string = null;

    appartment: string = null;

    sector: string = null;

}

* If new field is to be added in the html, the model name should be added here.
* We are getting the response form the server like below:



* In createFormService we are constructing the custom formControl using generateForm function.
* We are adding some extra attributes to the default object to make use of common components.
* We will get the value from server response for the field name as renderFlag.
* Based on the “renderFlag” we are showing the field to the user.
* In this we are adding the “isVisible” property for taking the flag to show and hide the field for common input component.
* Server response for fields and usage:
* renderFlag – show and hide the field
* allowedDataType – restrict the user to type invalid characters. Allowed characters will be given from response for eg. alphanumeric-hypen-dot-space. We need to take the equivalent of this regex from the constant file.
* fieldMaxlength – restrict the user to from input character more than max length
* fieldMinlength – check the input fields for minimum character for the field.

  if (data[property] )

            {

                controls[property].restrict={}

                if( data[property].hasOwnProperty('renderFlag') ){

    controls[property].isVisible = data[property].renderFlag;

}

if (data[property].hasOwnProperty('allowedDataType') && data[property].allowedDataType) {

    controls[property].restrict.pattern = data[property].allowedDataType;

}

if (data[property].hasOwnProperty('fieldmaxlength') && data[property].fieldmaxlength) {

    controls[property].restrict.maxlength  = data[property].fieldmaxlength;

}

if (data[property].hasOwnProperty('fieldminlength') && data[property].fieldminlength) {

    controls[property].restrict.minlength  = data[property].fieldminlength;

}

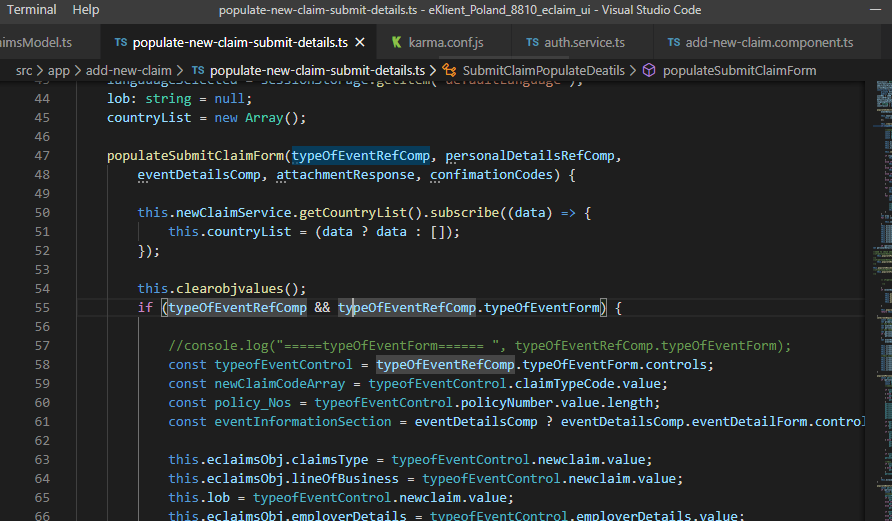
            }

            controls[property].fieldName = property;

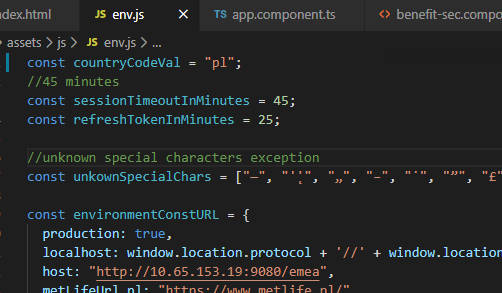
        });

        return this.fb.group(controls);

**Claim Submission:**



* Final save object screening will be happened here.



1. unkownSpecialChars in env.js for accepting special characters that is not in our keyboard

unkownSpecialChars is used in src\app\shared\directives\patternRestrict.directive.ts

et unkownSpecialChars = (environment\_constants && environment\_constants.unkownSpecialChars) ?

      environment\_constants.unkownSpecialChars :[] ;

      for(let i=0;i<unkownSpecialChars.length;i++){

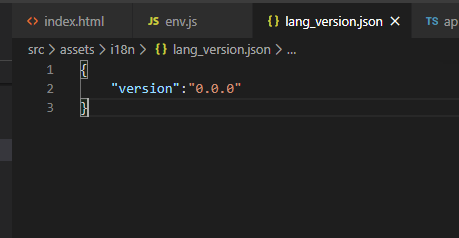
        if( $event.target.value.indexOf(unkownSpecialChars[i])>-1){

          $event.target.value=($event.target.value).replaceAll(unkownSpecialChars[i],'');

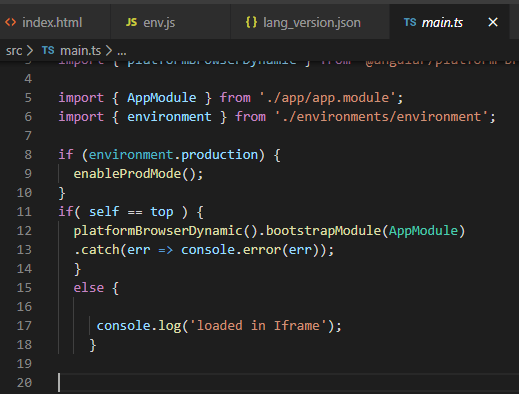
         }

      }

2. for country based setup and configuration env.js is created. Everytime we don’t need to build separately for different countries and different servers like QA,UAT etc.. instead of that we can change the country name and host URL in env.js. it will load while the index page is loading.

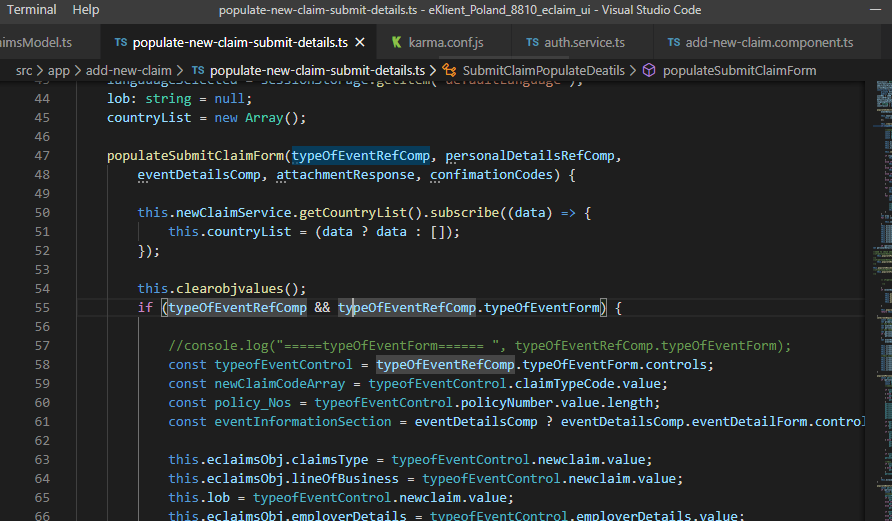


3. Language version needs to be change if u want to reflect the language json changes in the server immediately.

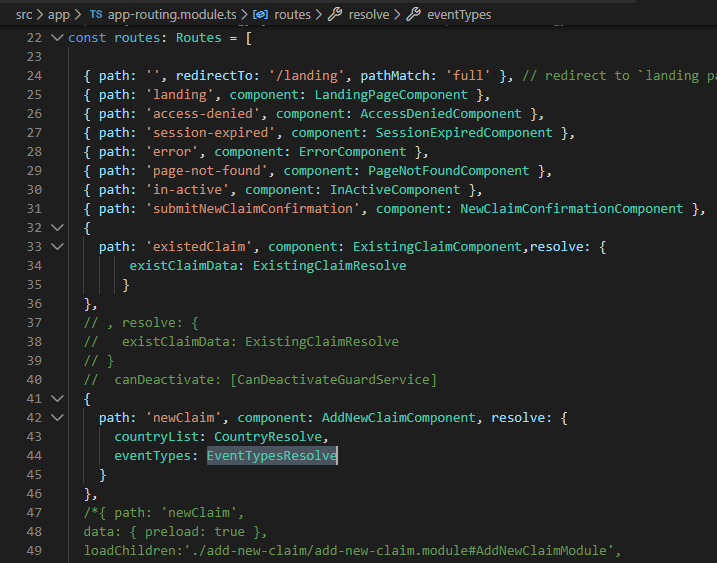


4. security changes we did. like password encoding..secured informations like pesel fields should be password, application should not load on iframe.

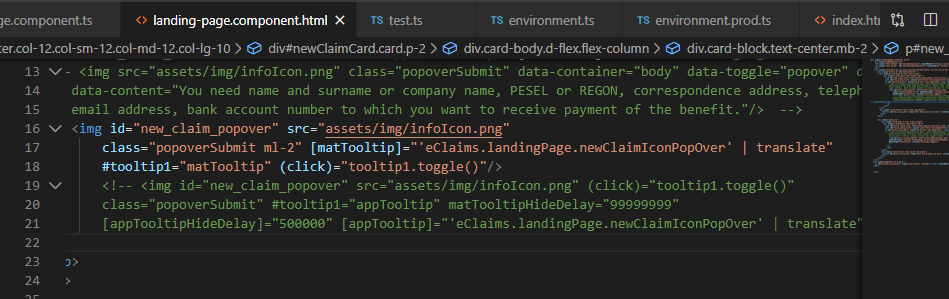
Self object and parent object should be same otherwise application won’t get bootstrapped.



Final save object screening will be happened here.



Event types and country list should be loaded before the page load. So we resolve it while routing



MatTooltip is used in landing page

From landing page there are two major routes existing claim and new claim

In Existing claim there are 2 tabs attachment tab and confirmation tab.

In Attachment tab there are two sub sections personal details section and add document section

Confirmation tab will be disabled until and unless attachment section is completed and submit

@ViewChild(PersonalDetailsComponent, { static: false }) personalDetailComponent: PersonalDetailsComponent;

  @ViewChild(DocumentAttachmentComponent, { static: false }) attachDocument: DocumentAttachmentComponent;

ViewChild is used to get the child values.

File selection checks the file size

If we uploaded the document,

  allowedType = ['jpeg', 'jpg', 'tiff', 'pdf', 'gif', 'png', 'bmp', 'tif'];

virus scan service will be triggered. Once it is success file size is compared with the given file size. It should be minimum of given minFileSize and files

1. maxFilesize: "52428800"
2. minFilesize: "5242880"

Preview file:

Using a Object tag in a popup we are showing the preview.

remove file:

using call by reference calling the confirm-dialog-component and deleting the file.

common dialog serice

 const dialogRef = this.dialog.open(dialogComponent, {

        width: '300px',

        data: data,

        disableClose: true

    });

Email id has the web api service call for validating it.

While sending the file to the service. Document type +”\_”+filename should be send.

fileNameAlter(event, index) {

    let claimDocType = event.value.replace("\_existing", "");

    this.fileUploadModel[index].name = claimDocType + "\_" + this.fileUploadModel[index].data.name;

    this.dataService.setOption('uploadedFile', this.fileUploadModel);

  }

Country dropdown we will be getting the data from web service call. If server is down or getting any error country service call, we will be putting local json call and get the details

If the browser is not supporting to view the tiff file then we are allowing the user to download the file.

  path: 'newClaim', component: AddNewClaimComponent, resolve: {

      countryList: CountryResolve,

      eventTypes: EventTypesResolve

    }

Before loading the new claim page we need to load the event types which is the first tab user is going to view. Event types is taken from the service call and it is resolved in the routing part. Also country list is resolved before the page loading.

Event types resolved from routing logic has been moved to add-new-claim.component.ts onInit() function

this.newClaimService.triggerShowClaimCompApi()

showing a new claim and its confirmation screen is based on the flag “showNewClaim”

We are using the material stepper to show the improvement of form filling and showing the error icon and success icon based on the form validation.

stepperReset(){

  this.personalDetailsSecError = false;

  this.eventDetailsSecError = false;

  this.attachmentSecError = false;

  this.reviewSecError = false;

  if(this.stepper && this.stepper.reset){

    this.stepper.reset();

  }

}

Some fields will be show and hide based on the other field values. We are storing the

 typeOfEventLobOldValue = "";

  primaryClaimEventOldValue = "";

  eventRelatedOldValue = '';

  insuranceReferOldValue = '';

  capcityOldValue = '';

  policeConductedInvestigationOldVal = '';

  employerPreAccidentRepOldVal = '';

  sportsClubPlayerOldValue = '';

calledOnce is to check whether the oninit and initial service calls for the sections are called already. If it is called for each section calledOnce value will be true. Otherwise it will be false.

Also if we changed the LOB or claimtype in typeofevent section we make the called once for other section as false.

if (event.previouslySelectedIndex == 0) {

      if ((this.typeOfEventLobOldValue != this.typeOfEventRefComp.typeOfEventForm.value.newclaim)

        || this.comparePrimaryClaimArr(this.primaryClaimEventOldValue, this.typeOfEventRefComp.typeOfEventForm.value.claimTypeCode)) {

        //  this.stepperReset();

          for (let i = 0; i < this.formDetails.length; i++) {

          this.formDetails[i].calledOnce = false;

        }

      }

    }

If new value is differ from old value for the code value below will re initialize the attachment section.

 this.getSectionOldValue(event.selectedIndex);

    if (event.previouslySelectedIndex == 2 || event.previouslySelectedIndex == 1) {

      if ((this.eventDetailsComp && this.eventDetailsComp.eventDetailForm.value &&

        this.eventDetailsComp.eventDetailForm.value.eventInformationSection &&

        this.eventDetailsComp.eventDetailForm.value.eventInformationSection.eventRelatedTo != undefined &&

         this.eventRelatedOldValue != this.eventDetailsComp.eventDetailForm.value.eventInformationSection.eventRelatedTo) ||

        (this.personalDetailsRefComp.insuranceRefComp &&

          this.personalDetailsRefComp.insuranceRefComp.insureForm.value.insuranceEventRefersTo !== null &&

          this.insuranceReferOldValue !== this.personalDetailsRefComp.insuranceRefComp.insureForm.value.insuranceEventRefersTo) ||

        (this.personalDetailsRefComp.personalForm.value.entitledForBenefitsSection &&

          this.personalDetailsRefComp.personalForm.value.entitledForBenefitsSection.capacity != undefined && this.capcityOldValue !==

          this.personalDetailsRefComp.personalForm.value.entitledForBenefitsSection.capacity) ||

          this.questionarieStatus()) {

        if (this[formDetailClone.formRef][formDetailClone.initFunc] && formDetailClone.formRef == "attachmentsComp") {

          //  event.selectedIndex == 3

          // && event.previouslySelectedIndex == 0

          this[formDetailClone.formRef][formDetailClone.initFunc]();

        }

      }

Based on the response atrribute breadCrumbsRendered we are removing the section details from the formDetails array.

        for (let i = this.formDetails.length - 1; i >= 0; i--) {

          if (response.breadCrumbsRendered.indexOf(this.formDetails[i].nameInResponse) == -1) {

            this.formDetails.splice(i, 1)

          }

        }

callAttachmentsFunc(generatedClaimNo){

  if (generatedClaimNo) {

    let attachmentReference: any = null;

    if (this.renderClaimSections.tab2AttachmentsSection.renderFlag) {

      attachmentReference = this.personalDetailsRefComp.attachmentsComp;

    } else {

      attachmentReference = this.attachmentsComp;

    }

    this.attachmentFiles(this.typeOfEventRefComp, attachmentReference, generatedClaimNo);

  }

}

Event type model :

export class EventTypeModel {

  claimTypeSection: string = null;

  policyDetailsSection: string = null;

  policyNumber: string = null;

  employerDetails: string = null;

}

Service call for getting secondary claim type

  getEventTypeSec(claim: string, index) {

    const i = index

    // const flagtest2="assets/mocks/TypeOfEventSecondary.json";

    // this.cService.getData(flagtest2)

    this.typeOfEventSecondary();

Enable disable claimtype based on secondary claim type

disableCheckbox(index) {

    this.eventSecondaryValue = this.eventTypeClaimBusinessSec1

    const control = this.typeOfEventForm.controls.newclaimCode as FormArray;

   if(this.eventSecondaryValue && this.eventSecondaryValue.length){

    for (let i = 0; i < this.eventSecondaryValue.length; i++) {

Type of event secondary claims type service call and disclaimer call.

  this.getEventTypeSec(secondaryClaimEvent, index);

      this.getDisclaimerSection(secondaryClaimEvent);

broker

https://qa.eclaim.metlife.pl/?countryCode=pl&sourcekey=2de4f0048fbe84b9611333e2745bba485cbc0b01056e7f539f9c61c57820e9f2d97a66b1701d9b29a80596a211ca3460a802a6cc08db563a843654eaa3d066c8beb3fc4b30f10aab933c8424a4e206cd

callcenter

https://qa.eclaim.metlife.pl/?sourcekey=2de4f0048fbe84b96a9ae77801b5c9db5cbc0b01056e7f539f9c61c57820e9f2d97a66b1701d9b29a80596a211ca34604d5c5a9a4d19cc1cf1340344b85201a5451660dc98afc23947841120f6d28a131ccd91c419c3efe9642d615c7d69b2a7

ruleFileName: "Eclaims\_ComponentRenderBySourceOfOrigin.xls\_FIELD\_RENDER\_DETAILS"

Based on the below service call response key we are loading the stepper.

For broker flow

breadCrumbsRendered: "TypeOfEvent|PersonalDetails"

For Normal flow

breadCrumbsRendered: "TypeOfEvent|PersonalDetails|EventDetails|Attachments|Review"

Attach document file upload based on browse button

browseButton: {renderFlag: true, mandatoryFlag: false, subQuestionFlag: false, fieldmaxlength: null,…}

  //generateAllForms(): create form group based on the validation response JSON

  //getSectionDetailsIfMandatory : return the sections which are mandatory

  //setSectionRenderValues() : change validation if type of event selection changed

Insurance section

copyFromEklient()

get the values from server using client id and set it.

Onchange insuranceEventRefersTO we will reset the attachment. We will give alert for that and empty the files upload.

attachmentReset()

Benefit section:

fdRadioLabelGrp is an array label, translation key, form control and its value.

Iterate the fdRadioLabelGrp and bind control, value,label with translation pattern and labelname.

    <div [class]="fdPClassName" \*ngFor="let controlValue of fdRadioLabelGrp;let i = index">

      <!-- <pre> {{i}} {{fdLabelControlName && fdLabelControlName.length>0 && (fdGroupName.get(fdLabelControlName[i])?.isVisible)}} </pre>

        -->

      <div \*ngIf="(controlValue &&

          (fdGroupName.get(controlValue.controlName)?.isVisible!=undefined)) ?

        (fdGroupName.get(controlValue.controlName)?.isVisible) : true"

        [class.error\_label]="(fdGroupName?.get(fdControlName)?.errors && fdGroupName?.get(fdControlName)?.touched)">

        <input type="radio" [formControlName]="fdControlName" class="form-check-input" [id]="controlValue.controlName"

          [name]="fdControlName" [attr.disabled]="fdDisabled" [value]="controlValue.value"

          (click)="fdEventValue(controlValue.value)" (change)="changeEvent($event)">

        <label [for]="controlValue.controlName" class="mr-3"

          \*ngIf="controlValue.labelName && controlValue.translatePattern">

          <span [innerHtml]="(controlValue.translatePattern + controlValue.labelName) | translate">

          </span>

          <p \*ngIf="controlValue.tooltip" [id]="controlValue.controlName+i"><i

              [innerHtml]="(controlValue.translatePattern + controlValue.tooltip)|translate"></i>

          </p>

        </label>

        <radio-group [fdGroupName]="benefitForm" [fdControlName]="'benefitiaryGrp'" [fdClassName]="'mr-0'"

            [fdName]="'benefitiaryGrp'" [fdPClassName]="'form-check form-check-inline col-12 col-sm-auto col-md-auto'"

            (changeValue)="onBenefiaryChange($event)" [fdRadioLabelGrp]="benefitiaryFormRadioGrp"

            [errMessage]="'errors.radio'">

        </radio-group>

      this.benefitiaryFormRadioGrp = [{

        labelName: 'individual',

        translatePattern: 'eClaims.newClaim.personalDtails.',

        controlName: 'individual',

        value: 'individual'

      }

        ,

      {

        labelName: (data ? 'institutionDifferentThanTrader' : 'institution'),

        translatePattern: 'eClaims.newClaim.personalDtails.',

        controlName: 'institutionDifferentThanTrader',

        value: 'institution'

      },

      {

        labelName: 'trader',

        translatePattern: 'eClaims.newClaim.personalDtails.',

        controlName: 'trader',

        value: 'trader'

      }];

On benefiary change based on the selection (Individual, Institution and trader) and country of the application, the capacity will be changed

      <radio-group [fdGroupName]="benefitForm" [fdControlName]="'capacity'" [fdClassName]="'mr-0'"

            [fdName]="'capacity'" [fdPClassName]="'form-check my-2'" [fdRadioLabelGrp]="capacityFormRadioGrp"

            [errMessage]="'errors.radio'" (changeValue)="onCapacityChange($event)">

        </radio-group>

onBenefiaryChange(event) {

if (this.benefitForm.get('benefitiaryGrp').value === 'individual') {

      //['insured','beneficiary','statutory','plenipotentiary']

      this.newClaimService.setIndividual(true);

      this.capacityFormRadioGrp = this[('capacityFormRadioGrp\_' + this.userData.defaultLanguage + '\_indv')];

      this.setRequiredFields(this.benefitForm.controls, this.beneficiaryIndvFields, 'enable');

      this.setRequiredFields(this.benefitForm.controls, this.beneficiaryInstituteFields, 'disable');

      this.setRequiredFields(this.benefitForm.controls, this.beneficiaryTraderFields, 'disable');

For eg. Poland country individual selection below labels will be shown in the radio group options

capacityFormRadioGrp\_pl\_indv = [

    {

      labelName: 'insured',

      translatePattern: 'eClaims.newClaim.personalDtails.',

      controlName: 'insured',

      value: 'insured',

      tooltip: ''

    }, {

      labelName: 'owner',

      translatePattern: 'eClaims.newClaim.personalDtails.',

      controlName: 'owner',

      value: 'owner',

      tooltip: ''

    },

    {

      labelName: 'otherPerson',

      translatePattern: 'eClaims.newClaim.personalDtails.',

      controlName: 'otherPerson',

      value: 'otherPerson',

      tooltip: ''

    },

    {

      labelName: 'beneficiaryEntitledTo',

      translatePattern: 'eClaims.newClaim.personalDtails.',

      controlName: 'benefitiary',

      value: 'benefitiary',

      tooltip: 'InheritenceLabel'

    },

    {

      labelName: 'statutoryOfBeneficiary',

      translatePattern: 'eClaims.newClaim.personalDtails.',

      controlName: 'statutoryRepBen',

      value: 'statutoryRepBen',

      tooltip: 'StatutoryTooltip'

    },

    {

      labelName: 'plenipotentiaryOfBeneficiary',

      translatePattern: 'eClaims.newClaim.personalDtails.',

      controlName: 'plenipotentiaryBenf',

      value: 'plenipotentiaryBenf',

      tooltip: 'PlenipotentiaryTooltip'

    }

  ];

If nationality is selected as “other” then the text field will be enabled to enter the nationality which is not in the list. If selected the country in the option list, “fill in nationality” field will be disabled.

  changeNationalityTrader(event) {

    if (this.benefitForm.get('nationalityTrader').value == 'other') {

      this.setRequiredFields(this.benefitForm.controls, ['fillInTheNationalityTrader'], 'enable');

    } else {

      this.setRequiredFields(this.benefitForm.controls, ['fillInTheNationalityTrader'], 'disable');

    }

  }

Nationality for institution will be an array we can add as many as value in “addCitizenshipInstituteCount” in constants.

  addNationalityBuInst() {

    if ((this.benefitForm.get('nationality\_bu\_Inst').controls.length == 0

      || this.benefitForm.get('nationality\_bu\_Inst').valid) && (this.benefitForm.get('nationality\_bu\_Inst').controls.length < this.addCitizenshipInstituteCount)) {

      this.benefitForm.get('nationality\_bu\_Inst').push(this.nationality\_bu\_InstGrp());

    }

    this.createForm.markFormGroupTouched(this.benefitForm.get('nationality\_bu\_Inst'));

  }

In the same way individual and trader we can add the nationality

addNationalityBuTrader

addCitizenship() {

for the selection of other country than logged in country “document validity” is mandatory

 if (this.benefitForm.controls.nationalityIndv['controls'][i].controls.documentValidupto

        && this.benefitForm.get('idDocumentsValidUptoIndv')['isVisible']

        && (this.documentValidUptoShow.indexOf(this.benefitForm.controls.nationalityIndv['controls'][i].controls.citizenship.value) === -1)

        && (this.defaultLanguage != this.benefitForm.controls.nationalityIndv['controls'][i].controls.citizenship.value)) {

        this.benefitForm.controls.nationalityIndv['controls'][i].controls.documentValidupto.setValidators([Validators.required]);

        this.benefitForm.controls.nationalityIndv['controls'][i].controls.documentValidupto.updateValueAndValidity();

On selection of nationality as other will mandate the fill in nationality field where we need to fill the country name.

Any of the nationality is selected as “United States” then “tax residency country” is mandatory and it should be visible.

onNationalityIndvChange(event) {

this.showTaxResidencyCountryIndv = false;

    for (let i = 0; i < this.benefitForm.controls.nationalityIndv.value.length; i++) {

      // //console.log("value",this.benefitForm.controls.nationalityIndv.value[i].citizenship)

      if (this.benefitForm.controls.nationalityIndv.value[i].citizenship == "US") {

        this.showTaxResidencyCountryIndv = true;

        this.benefitForm.controls.taxResidencyCountryIndv.setValue(this.defaultLanguage);

        break;

      }

    }

eventTypes() once we get event types values

formDetailsOrigin = [

    {

      "form": "typeOfEventForm",

      "initFunc": "formInit",

      "formRef": "typeOfEventRefComp",

      "submit": "formSubmit",

      "error": "typeOfEventSecError",

      "nameInResponse": "TypeOfEvent",

      "calledOnce": false

    },

    {

      "form": "personalForm",

      "initFunc": "formInit",

      "formRef": "personalDetailsRefComp",

      "submit": "formSubmit",

      "error": "personalDetailsSecError",

      "nameInResponse": "PersonalDetails",

      "calledOnce": false

    },

Selected beneftiary is an individual then enable the validations for the particular below fields and disable for the other fields which are hidden but has the validations.

    if (this.benefitForm.get('benefitiaryGrp').value === 'individual') {

//beneficiaryIndvFields

  beneficiaryIndvFields = ['nameIndv', 'surNameIndv', 'countryOfBirthIndv',

    'cnpfieldIndv', 'dateOfBirthIndv', 'sexIndv', 'seriesAndNumberIndv',

    'fillInCountryBirth', 'fillInCountryTax', 'maidenName'];

 this.setRequiredFields(this.benefitForm.controls, this.beneficiaryIndvFields, 'enable');

      this.setRequiredFields(this.benefitForm.controls, this.beneficiaryInstituteFields, 'disable');

      this.setRequiredFields(this.benefitForm.controls, this.beneficiaryTraderFields, 'disable');

  beneficiaryInstituteFields = ['nationalityInst',

    'institutionNameInst', 'nIPInst', 'regionInst', 'kRSInst', 'nameInst',

    'surNameInst', 'sexInst', 'fillInTheNationalityInst', 'fillInTheNationalityReg'];

  beneficiaryTraderFields = ['nationalityTrader',

    'institutionNameTrader', 'nIPTrader', 'regionTrader', 'kRSTrader',

    'nameTrader', 'surNameTrader', 'sexTrader', 'fillInTheNationalityTrader',

    'fillInTheNationalityTraderReg'];

  setRequiredFields(parentControl, fieldsArr, validation) {

    for (let i = 0; i < fieldsArr.length; i++) {

      if (parentControl[fieldsArr[i]]) {

        //parentControl[fieldsArr[i]].setValidators(validation);

        //parentControl[fieldsArr[i]].updateValueAndValidity();

        parentControl[fieldsArr[i]][validation]();

      }

    }

  }

“inValidPeselNumber” is checked from web api service call and other is form validation. If both are valid then return “true” otherwise return “false”.

 formValid() {

     //console.log("benefitForm errorFormGroup", this.createForm.errorFormGroup(this.benefitForm));

    if (!this.inValidPeselNumber && this.benefitForm.valid) {

      return true;

    }

    return false;

  }

Adding citizenship shown when “otherNationlityBtn” render flag is true

                        <button mat-button (click)="addCitizenship()" \*ngIf="benefitRenderSecRules?.otherNationlityBtn?.renderFlag &&

                        (benefitForm.get('nationalityIndv').controls.length<this.addCitizenshipIndvCount)"

                            [innerHtml]="'eClaims.newClaim.personalDtails.fillInTheCitizenShip1'|translate"></button>

  addNationality() {

    if (this.benefitForm.get('nationalityIndv').controls.length < this.addCitizenshipIndvCount) {

      this.benefitForm.get('nationalityIndv').push(this.citizenshipGrp());

    }

  }

this.addCitizenshipIndvCount

add count will be restrict based on the above value which is defined in the constant file.

copy data from insurance section and use it in benefit section

            <div class="col-12 p-2 my-3 greyBg"

                \*ngIf="insuranceRefersToSectionFlag && (benefitRenderSecRules?.copyData?.renderFlag) && (benefitForm.get('capacity').value ==='insured')">

                <span class="mr-2">{{'eClaims.newClaim.personalDtails.copyDataFromInfo'|translate}}</span>

                <button mat-button class="btn-primary" type="button" (click)="copyDetailsFromInsuranceSec()">

                    {{'eClaims.newClaim.personalDtails.copy'|translate}} </button>

            </div>

copyDetailsFromInsuranceSec() {

    this.benefitForm.get('nameIndv').setValue(this.personalFormControl.value.insuranceRefersToSection.nameField);

    this.benefitForm.get('surNameIndv').setValue(this.personalFormControl.value.insuranceRefersToSection.surnameField);

Correspondance section:

  public postalMasking = {

    guide: false,

    showMask: true,

    mask: [/\d/, /\d/, '-',

      /\d/, /\d/, /\d/]

  };

            <input-field [fdGroupName]="corresspondenceAddressForm" [fdControlName]="'postalCode'"

                [fdPlaceHolder]="'postalCode'" [fdLabel]="'eClaims.newClaim.personalDtails.postalCodeLabel'|translate"

                [fdName]="'postalCode'" [fdId]="'postalCode'" class="col-12 col-md-4 col-sm-6"

                [fdClassName]="'form-control w-100'"

                \*ngIf="(postBoxEnableCountry.indexOf(defaultLanguage.toUpperCase())!=-1) || corresspondenceAddressForm.get('country').value !== 'RO'"

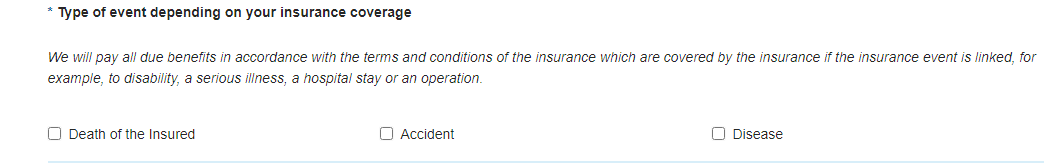
                (BlurValueEvent)="blurPostalCodeField($event)"

                [masking]="((postBoxEnableCountry.indexOf(defaultLanguage.toUpperCase())!=-1) && corresspondenceAddressForm.get('country').value === 'PL') ? postalMasking:null"

                [noRestrict]="defaultLanguage.toUpperCase()=='PL' ? true:false">

            </input-field>

Based on the \*Line of Business we will get the claim type code which will be translated from language json files.



{partner: "metlife", product: "Financial", claimType: "E240|E250|E270",…}

* 1. claimType: "E240|E250|E270"
  2. eventType: "PrimaryPolicyHolderEvent|SpouseEvent|PartnerEvent|OtherNomineeEvent|ChildPrimaryPolicyHolderEvent"
  3. ksession: null
  4. partner: "metlife"
  5. product: "Financial"
  6. ruleFileName: "Eclaims\_ProductClaimType.xls\_CLAIM\_DETAILS"
  7. sheetName: null

also another service call will get you the rules for claimtype, policy number, etc..

1. {ruleFileName: "Eclaims\_EventTypeComponentRender.xls\_CLAIMTYPE\_FIELD\_RENDER", sheetName: null,…}
   1. claimTypeSection: {renderFlag: true, mandatoryFlag: true, subQuestionFlag: false, fieldmaxlength: null,…}
   2. employerDetails: {renderFlag: true, mandatoryFlag: true, subQuestionFlag: false, fieldmaxlength: "40",…}
   3. ksession: null
   4. lob: "Financial"
   5. partner: "metlife"
   6. policyDetailsSection: {renderFlag: true, mandatoryFlag: false, subQuestionFlag: false, fieldmaxlength: null,…}
   7. policyNumber: {renderFlag: true, mandatoryFlag: false, subQuestionFlag: false, fieldmaxlength: "26",…}
      1. allowedDataType: ","
      2. fieldmaxlength: "26"
      3. fieldminlength: "0"
      4. mandatoryFlag: false
      5. renderFlag: true
      6. subQuestionFlag: false
   8. ruleFileName: "Eclaims\_EventTypeComponentRender.xls\_CLAIMTYPE\_FIELD\_RENDER"
   9. sheetName: null

Insurance section:

 setInsuranceRefersVal(){

    this.renderClaimSections = this.newClaimService.getParamValue('renderClaimSections');

    let insuranceValue =  this.insureForm.get('insuranceEventRefersTo').value;

    let lobValue = this.newClaimService.getClaimData().newclaim;

    let claimTypeValue = this.newClaimService.getPrimaryClaimType();

    if ((lobValue == 'IndividualLife' || lobValue == 'IndividualLifeM')

    && (claimTypeValue.indexOf('E410') != -1)) {

    this.eventRefersToOptions = ['Insured'];

  } else if ((lobValue == 'IndividualLife' || lobValue == 'IndividualLifeM')

    && (claimTypeValue.indexOf('E420') != -1)) {

    this.eventRefersToOptions = ['PrimaryOwnerEvent'];

  }

  else {

    this.eventRefersToOptions = this.newClaimService.getParamValue('eventTypeList') ? (this.newClaimService.getParamValue('eventTypeList').split('|')) : [];

  }

  //IndividualLife IndividualLifeM E420 E410

Insurance refers to change onEventReferChange()

attachmentReset(){

  try{

    let fileInAttachment = this.dataService.getOption('uploadNewClaimList');

    if (fileInAttachment && fileInAttachment.fileUpload && fileInAttachment.fileUpload.length > 0) {

      this.dialogService.openDialog(AlertDialogComponent, { 'heading': this.translate.instant('eClaims.existingClaim.confirmationLabel'), 'body': this.translate.instant('reAttachDocument'), 'primaryButton': 'OK', 'fromTypeofevent': true });

    }

   } catch(e){

console.log("error",e);

   }

}

Correspondance section:

Postal code masking format.

  public postalMasking = {

    guide: false,

    showMask: true,

    mask: [/\d/, /\d/, '-',

      /\d/, /\d/, /\d/]

  };

Set form validation and empty the values if the field is not rendered.

    setFormValidationLogic(models, data, patternExceptions?: any) {

        Object.keys(data).forEach(property => {

            if (models.get(property)) {

                let validations = this.getValidators(data[property], property, patternExceptions);

                /\* if(form.get(key)){

                          form.get(key).setValidators();

                      }

              \*/

                //console.log('property',property,validations,data[property],models.get(property));

                if (data[property]

                    && data[property].hasOwnProperty('renderFlag')

                    && data[property].renderFlag != true) {

                    if (this.isArray(models.get(property).controls)) {

                    } else {

                        models.get(property).setValue(null);

                    }

                }

Setting initial maxlength value from the service response. If other than default country is selected we will change the maxlength value of the field. If again user change the country to default country then we will take the value from this variable.

    this.mobilMaxLength = {

      length: this.corresspondenceAddressForm.controls.mobileNumber['restrict'].maxlength,

      required: (this.corresspondenceAddressForm && this.corresspondenceAddressForm.get('mobileNumber').validator({} as AbstractControl)) ? this.corresspondenceAddressForm.get('mobileNumber').validator({} as AbstractControl).required : ''

    }

    this.postalcodeMaxlenthinit = {

      length: this.corresspondenceAddressForm.controls.postalCode['restrict'].maxlength,

      required: this.corresspondenceAddressForm.get('postalCode').validator({} as AbstractControl).required

    };

Setting mobile maxlength value.

setMobileNumberLength(){

get the data from eklient using a service call and paste the respected values in correspondence section.

copyFromEklient

Based on the country selection, if selected country is Romania and the logged in country also Romania

Then we need to enable below fields

 ['county', 'zipCode', 'city', 'block', 'entrance', 'appartment', 'sector'];

Else if the selected country is other than Romania and logged in country is Romania

setAddressFields(event) {

    let countryValueRO = ['county', 'zipCode', 'city', 'block', 'entrance', 'appartment', 'sector'];

    let countryValueROField = ['flatNumber', 'postalCode', 'town', 'postBox'];

If the Logged in country is Poland then these fields are not required.

Form of Disbursement:

Any one of the checkbox should be selected for form of disbursement section.

    if (this.disbursementRenderSecRules.interestedinreinvestment.renderFlag) {

      disbursementBoxchecked.push(this.disbursementForm.value.interestedinreinvestment);

    }

    if (this.disbursementRenderSecRules.viabanktransfer.renderFlag) {

      disbursementBoxchecked.push(this.disbursementForm.value.viabanktransfer);

    }

    if (this.disbursementRenderSecRules.viafinancialinst.renderFlag) {

      disbursementBoxchecked.push(this.disbursementForm.value.viafinancialinst);

    }

    if (this.disbursementRenderSecRules.viapost.renderFlag) {

      disbursementBoxchecked.push(this.disbursementForm.value.viapost);

    }

    if (this.disbursementRenderSecRules.viapost.renderFlag) {

      disbursementBoxchecked.push(this.disbursementForm.value.viafinancialinst);

    }

    this.benefitChecboxError = true;

    for (let i = 0; i < disbursementBoxchecked.length; i++) {

      if (disbursementBoxchecked[i] && disbursementBoxchecked[i] != '') {

        this.benefitChecboxError = false;

      }

    }

If the logged in country is Romania then the bank account number should be a password field.

 if (this.defaultLanguage.toUpperCase() === 'RO') {

      this.bankAccControlType = 'password';

    } else {

      this.bankAccControlType = 'text';

    }

If via bank transfer checkbox is selected then the below fields get enabled and validation happened based on the rules service response. If it is unchecked then we will disable the validation for those fields.

bankTransferFields: any = ['countryofbank', 'bankaccountnumber', 'codeswift', 'bankname'

    , 'bankaddress', 'name', 'surname', 'businessunit', 'county', 'zipCode', 'city', 'streetname',

    'streetnumber', 'flatnumber', 'block', 'entrance', 'appartment', 'sector', 'country',

    'otherNationalityField', 'postalcode', 'town', 'postbox', 'disbursementDisclaimerChkbox'];

Same logic is applied for postValue, viafinancialChange

On blur bank account number field will request the service for validating the bank number. If the bank account number is valid then service will return with the valid bank name. we need to populate it in the bank name field.

 blurBankAccountNumberField(event) {

    let bankaccountnumberUnmask = this.cleanup(this.disbursementForm.controls.bankaccountnumber.value);

    if (this.oldBankAccVal != bankaccountnumberUnmask) {

      this.showwbankaccountnumberInValid = false;

      this.disbursementForm.controls.bankaccountnumber['invalidFlag'] = this.showwbankaccountnumberInValid;

Event details section:

In this section there will be 4 sections eventInfo, Additional Details, HealthcareCenterInfo, HealthCareFamilyDoctor.

In call center flow there will be “Additional comments fields” that is added using addControl

<div class="text-center" style="background-color: white;"

            \*ngIf="renderClaimSections.additionalCommentsSection.renderFlag">

            <div class="my-auto pt-3 additionalComments">

                <img (click)="addtionalCommentsShow=!addtionalCommentsShow" \*ngIf="!addtionalCommentsShow"

                    src="assets/img/utility-plus.png" class="card-img-top mx-auto pointerCursor"

                    alt="...">{{ 'eClaims.existingClaim.personalData.addadditionalComment' | translate }}</div>

            <div class="form-group py-3 d-block d-sm-flex justify-content-center" \*ngIf="addtionalCommentsShow">

                <textarea formControlName="eventDetailsAdditionalComments" matInput class="form-control" rows="10"

                    maxlength="300" style="border: 1px solid #d9d9d6;text-align:left">

    </textarea>

  callCenterFlowControls() {

    if (this.renderClaimSections.additionalCommentsSection.renderFlag) {

      if (this.eventDetailForm.get('eventDetailsAdditionalComments')) {

      } else {

        this.eventDetailForm.addControl('eventDetailsAdditionalComments', new FormControl('', Validators.nullValidator));

      }

    }

  }

Both heathcare center and healthcare doctor sections are form in the array of components.

            <healthcare-event-sec

                \*ngFor="let healthInfo of eventDetailForm.get('informationHealthCareEventSection')['controls'];let indx = index"

                [healthCareInfoForm]="healthInfo" [indx]="indx" [healthCareEventSecRules]="healthCareEventSecRules">

            </healthcare-event-sec>

 <healthcare-familydoc-sec [healthCareDoctorEventSecRules]="healthCareDoctorEventSecRules"

                \*ngFor="let doctrInfo of eventDetailForm.get('informationHealthCareFamilyDocSection')['controls'];let indx = index"

                [healthCareDocotorInfoForm]="doctrInfo" [indx]="indx"></healthcare-familydoc-sec>

Date fields have the mask pattern, it should auto fill the “/” slash so we are using the mask directive and disabling the pattern validation.

  //pattern exception for date fields

  patternException = ['healthCareCenterPeriodFrom',

    'healthCareCenterPeriodTo',

    'dateOfEvent',

    'dateOfBirth',

    'dateOffirstSymptomsDisease',

    'dateWhenPhysicalInjuriesOccurred',

    'dateOfRecognitionDisease',

    'sickLeavePeriod',

    'sickLeavePeriodFromDate',

    'sickLeavePeriodToDate',

    'dateCompletionOfMedicalTreatment',

    'employmentPeriod',

    'employmentPeriodFromDate',

    'employmentPeriodToDate'];

Event info section:

Event Related to radio button field values will be getting through service call.

  this.eventRelatedToLableDetail = [];

      let eventDetailInfoSec = this.eventSecReqConfig.rulesRequestParam('EventDetailEventInformationSectionRender');

      this.commonService[environment.eventDetailServiceConfig.method](this.baseUrl, eventDetailInfoSec, this.headers)

        //this.commonService.getData(flagtest1)

        .subscribe((dataValue) => {

          //console.log("-------> ", dataValue);

          ///this.clearradioButtons();

          this.eventRelatedToValueDetail = dataValue.eventRelatedToValues.split('|')

          for (let i = 0; i < this.eventRelatedToValueDetail.length; i++) {

            this.eventRelatedToLableDetail.push(this.eventRelatedToValueDetail[i])

          }

          let eventRelatedToValue = this.eventInformationForm.get('eventRelatedTo').value ;

          if(eventRelatedToValue && eventRelatedToValue!='' && this.eventRelatedToLableDetail.indexOf(eventRelatedToValue)!=-1){

            this.getMessage(this.eventInformationForm.get('eventRelatedTo').value);

          }

        });

On change “event related to” field will change the “additional details” section fields with the help of service call.

Change in event related field will reset the attachment section. All the attachments uploaded will be removed and alert popup will be triggered.

attachmentReset(){

    try{

      let fileInAttachment = this.dataService.getOption('uploadNewClaimList');

      if (fileInAttachment && fileInAttachment.fileUpload && fileInAttachment.fileUpload.length > 0) {

        this.dialogService.openDialog(AlertDialogComponent, { 'heading': this.translate.instant('eClaims.existingClaim.confirmationLabel'), 'body': this.translate.instant('reAttachDocument'), 'primaryButton': 'OK', 'fromTypeofevent': true });

      }

     } catch(e){

  console.log("error",e);

     }

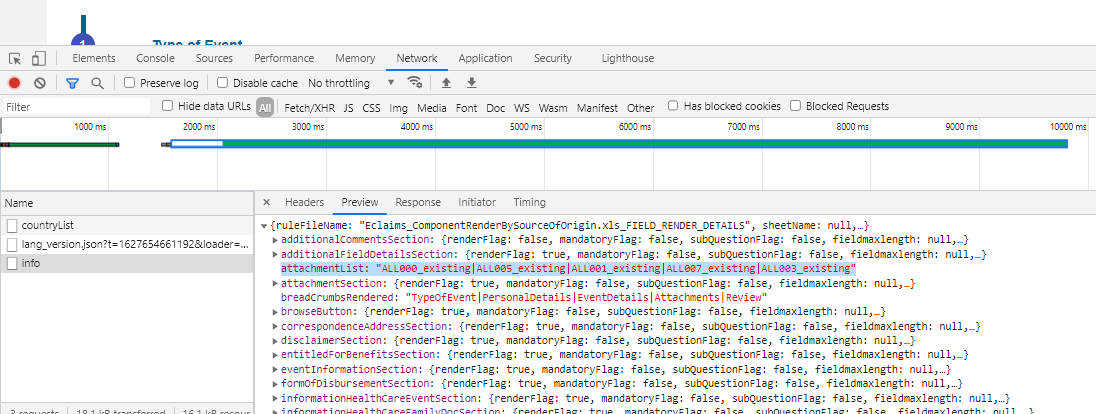
  }

Attachment Section:

Attachment List is based on eventType,claimType,Insurance refers to and eventRelatedTo fields.

attachmentList.split('|');

if the logged in user is a “Borrower” then attachment list will be taken from the “componentsRenderBySourceOfOrigin” service call.



Below are the allowed file format for uploading the files.

  allowedType = ['jpeg', 'jpg', 'tiff', 'pdf', 'gif', 'png', 'bmp', 'tif'];

On each file selection files will be checked for Virus check. If the file is virus free, then file size is checked for maximum size and minimum size.

     this.httpCommonService[environment.virusFileCheck.method](virusFileCheck, reqPayload, "").subscribe(data => {

           if (data.statusCode == 200) {

        this.fileSizeObject.filesSizeCalculation = this.fileSizeObject.filesSizeCalculation + file.size;

        if (file.size < this.userInfo.minFilesize && this.fileSizeObject.filesSizeCalculation < this.userInfo.maxFilesize) {

          const reader = new FileReader();

file name is altered by filename\_filetype.

  this.fileUploadModel[index].name = removeExistingClaim + '\_' + this.fileUploadModel[index].data.name;

Review section:

Using the common label component we are showing the values in the review screen.

            <label-field [fdGroupName]="benefitSectionFormGroup" [fdControlName]="'seriesAndNumberIndv'"

                [fdLabel]="'eClaims.newClaim.personalDtails.seriesIdDocumentLabel'| translate" [fdClassName]="'mr-1'">

            </label-field>

            <label class="mb-2 mt-1 ml-2 font-weight-bold upperCase">{{beneficiarySection?.seriesAndNumberIndv}}</label>

Submit :

While submitting the data first service call will be the claim number generation. Once claim number is generated and getting the response then attachments are attached with the generated claim number.

onSubmit() {

    // console.log(this.newClaimService.getCountryList());\*

    let isSubmitValid = this.callSubCompFormSubmit();

    if (isSubmitValid) {

      const baseUrl = environment.host + environment.generateClaimServiceConfig.url;

      var hiddenhcCount\_val = document.getElementById("hiddenhcCount");

      //const val = hiddenhcCount\_val.getAttribute("hcCount");

    if(!this.generatedClaimNo || this.generatedClaimNo ==''){

      this.httpService[environment.generateClaimServiceConfig.method](baseUrl).

      subscribe((generatedClaimNo) => {

        ////console.log("======generatedClaimNo===== >", generatedClaimNo);

        this.generatedClaimNo = generatedClaimNo;

        this.callAttachmentsFunc(generatedClaimNo);

      },(err) => {console.log(err)

        this.dialogService.openDialog(ErrorDialogComponent, { 'heading': 'Error'});

//        this.router.navigate(['/errorPage']);

      })

    }else if(this.generatedClaimNo && this.generatedClaimNo!=''){

      this.callAttachmentsFunc(this.generatedClaimNo);

    }

    } else {

      this.dialogService.openDialog(AlertDialogComponent, { 'heading': '', 'body': this.translate.instant('mandatoryErrorMsg'), 'primaryButton': 'OK' });

    }

On successfully attached the documents with the claim number, Save claim data will be initiated.

    attachmentsFormData.append('claimDetails', JSON.stringify(claimDetails));

    this.httpService[environment.attachmentServiceConfig.method](baseUrl, attachmentsFormData, headers).

      subscribe((attachmentResponse) => {

        //console.log("======attachmentResponse===== >", attachmentResponse);

        this.saveClaim(attachmentResponse, generatedClaimNo, attachmentsComp);

      },(err) => {console.log(err)

Using “populateSubmitClaimForm” function we are generating the data object which is going to be send in save service call.

import { SubmitClaimPopulateDeatils } from './populate-new-claim-submit-details';

creating instance

 populateDetailsObj: SubmitClaimPopulateDeatils =

    new SubmitClaimPopulateDeatils(this.translate, this.\_formBuilder, this.httpService, this.newClaimService,

      this.dataService);

saveClaim(attachmentResponse: any, claimNo: string, attachmentsComp?: any) {

    var headers = new HttpHeaders();

    ///let screenRequestObj = new ScreenRenderReqModel();

    const baseUrl = environment.host + environment.saveServiceConfig.url;

    this.claimAttachmentModel = [];

    let confimationCodes = this.newClaimService.getnewClaimConfirmationCodes();

    this.eclaimsObj = this.populateDetailsObj.populateSubmitClaimForm(this.typeOfEventRefComp,

      this.personalDetailsRefComp, this.eventDetailsComp,

      attachmentResponse, confimationCodes);

    this.newClaimService.setCNPFieldValue(this.eclaimsObj.personProposesClaimInfoVO.cnpFieldIndv);

    let documentTypesArray = attachmentsComp.documentTypes;

    let selectedFilesList = attachmentsComp.fileUploadModel;

Confirmation:

On Save claim success page will be redirected to confirmation screen.

On click “Download claim” will ask for a confirmation about sensitive data. Once confirmed PDF will be downloaded from the service call.

downloadConfirmFormByClaimNo() {

    //if (((code == 'conf13') || (code == 'conf15') || (code == 'conf16') || (code == 'conf14')) && this.showDownloadBtn) {

    let confirmDownload = confirm(this.translate.instant('eClaims.existingClaim.confirmation.CacheWarning'));

    if (confirmDownload == true) {

      this.getDownloadPDF()

    }

  getDownloadPDF() {

    let fileID = this.newClaimResponse.claimAttachmentVO[this.newClaimResponse.claimAttachmentVO.length - 1].claimnumber;//this.dataService.getOption('claimnumber');

    let docId = this.newClaimResponse.claimAttachmentVO[this.newClaimResponse.claimAttachmentVO.length - 1].fileValue;

   // console.log(docId);

    let url = environment.host + environment.confirmationDownloadServiceConfig.url + docId;

    this.commonService.downloadPDF(url).subscribe((data) => {

      this.downloadFile(data);

    });

  }

downloadFile(data) {

    window.scroll(0,0);

    let blob = new Blob([data], { type: "application/pdf" });

    if (window.navigator.msSaveOrOpenBlob) {

      //IE11 & Edge

      window.navigator.msSaveOrOpenBlob(blob,'Claim\_Form.pdf');

    } else {

      let url = window.URL.createObjectURL(blob);

      window.open(url);

    }

  }

Submit Another claim:

On clicking “Submit another claim” will open a popup and ask for the “PESEL” number which user entered while filling the data. It validates and if it is correct then screen will be redirected to new claim page and all the data which is entered will be retained based on the “Confirmation code”.

  gotoReportNextClaim(code) {

    if (code == 'conf21') {

      //clear form values and navigate to landing page

      this.redirectTo();

    } else if (code == "conf20") {

      if (this.isIndividual && this.pesel !== '') {

        this.dialogService.openDialog(AlertInputDialogComponent, { 'heading': '', 'body': this.translate.instant('backAlert'), 'cnfmCode': code, 'primaryButton': this.translate.instant('eClaims.newClaim.personalDtails.nextButtonLabel'), 'tertiaryButton': this.translate.instant('Cancel'), 'nextUrl': 'newClaim' });

      } else {

        // navigate to landing with clearing all data

        this.redirectTo();

      }

    } else if (code == "conf19") {

      if (this.isIndividual && this.pesel !== '') {

        this.dialogService.openDialog(AlertInputDialogComponent, { 'heading': '', 'body': this.translate.instant('backAlert'), 'cnfmCode': code, 'primaryButton': this.translate.instant('eClaims.newClaim.personalDtails.nextButtonLabel'), 'tertiaryButton': this.translate.instant('Cancel'), 'nextUrl': 'newClaim' });

        //this.router.navigate(['/newClaim']);

      } else {

        // navigate to landing with clearing all data

        this.redirectTo();

      }

    } else {

      this.redirectTo();

    }

**CONCLUSION:**

In the future, the main challenges are to unify the documentation application with other guidelines used in the organization. Also, Angular 8 has recently been released and in the future the UI-library will be updated to use that. In web-development new attractive technologies are released and updated each year and updating the dependencies for the documentation application and the UI library is one part of maintaining the software. Switching to new libraries needs to be carefully studied first. In the future the documentation application could be extended to have live editing features that would enable the user to edit the demonstration examples directly in the browser.