import { AfterViewInit, ChangeDetectorRef, Component, OnInit, QueryList, ViewChild, OnDestroy, HostListener } from '@angular/core';

import { Location } from '@angular/common';

import { ActivatedRoute, Router } from '@angular/router';

import \* as moment from 'moment';

import {

CountryService, CurrencyExchangeService, CurrencyService, OrganisationService,

PolicyService, UserService, PolicyDocument, DocumentService, EngineeringService

} from '../../../commons/nri-auth';

import { NriBlockScreenService } from '../../../commons/nri-blocl-screen/nri-block-screen.service';

import { maxDate, maxLenght, minDate, required } from '../../../commons/nri-form';

import { NriGlobalStateService } from '../../../commons/nri-global-state';

import {

NriCheckboxItems,

NriInputDatepickerComponent,

NriInputMultiSelectOptions,

NriInputSelectComponent,

NriInputSelectMultipleComponent,

NriInputSelectOptions,

NriInputTextComponent,

NriRadioOptions

} from '../../../commons/nri-inputs';

import { NriModalComponent } from '../../../commons/nri-modal';

import { AbstractNameAndIdDTO } from '../../../DTO/AbstractNameAndIdDTO';

import { AddOnDTO } from '../../../DTO/AddOnDTO';

import { CountryDTO } from '../../../DTO/CountryDTO';

import { CoverageDTO } from '../../../DTO/CoverageDTO';

import { CoverageLimitDTO } from '../../../DTO/CoverageLimitDTO';

import { CurrencyDTO } from '../../../DTO/CurrencyDTO';

import { DeductibleDTO } from '../../../DTO/DeductibleDTO';

import { DeductionDTO } from '../../../DTO/DeductionDTO';

import { DeductionNameDTO } from '../../../DTO/DeductionNameDTO';

import { EntryTypeDTO } from '../../../DTO/EntryTypeDTO';

import { GetComboInfoInterface } from '../../../DTO/getComboInfoInterface';

import { InsuranceStructureDTO } from '../../../DTO/InsuranceStructureDTO';

import { LocationDTO } from '../../../DTO/LocationDTO';

import { PolicyIncomeDTO } from '../../../DTO/PolicyIncomeDTO';

import { PolicyPremiumTaxesDTO } from '../../../DTO/PolicyPremiumTaxesDTO';

import { PolicyShareDTO } from '../../../DTO/PolicyShareDTO';

import { PolicyStatusDTO } from '../../../DTO/PolicyStatusDTO';

import { StationDTO } from '../../../DTO/StationDTO';

import { SublimitDTO } from '../../../DTO/SublimitDTO';

import { TaxTypeCalculationDTO } from '../../../DTO/TaxTypeCalculationDTO';

import { UserDTO } from '../../../DTO/UserDTO';

import { UserInfoDTO } from '../../../DTO/UserInfoDTO';

import { DateUtils } from '../../../Utils/DateUtils';

import { ObjectUtils } from '../../../Utils/ObjectUtils';

import PolicyUtils from '../policyUtils';

import { ManageCommunService } from './manage-commun.service';

import { ManagePoliciesService } from './manage-policies.service';

import { NriUploadFilesComponent } from '../../../commons/nri-upload-files';

import { NriButtonComponent } from '../../../commons/nri-button';

import { DocumentDTO } from '../../../DTO/DocumentDTO';

import { DocumentTypeDTO } from '../../../DTO/DocumentTypeDTO';

import { LockResult, EntityLockService } from '../../../commons/nri-auth/services/entityLock.service';

moment.locale('en');

import { OrganisationDTO } from '../../../DTO/OrganisationDTO';

import { PolicyDTO } from '../../../DTO/PolicyDTO';

import { ContactDetailsDTO } from '../../../DTO/ContactDetailsDTO';

import { OrganisationTypeDTO } from '../../../DTO/OrganisationTypeDTO';

import { create, all } from 'mathjs';

import { UnderwritingTaskDTO } from '../../../DTO/UnderwritingTaskDTO';

import { UnderwritingProcessDTO } from '../../../DTO/UnderwritingProcessDTO';

import { DeductionFormFields, TaxFormFields, Taxes, DeductionsAndFees, IncomeFormFields, IncomeRows, ReinsuranceTableRows } from '../create-policies/create-policies.component';

const config = {

number: 'BigNumber',

precision: 20

};

const math = create(all, config);

interface InstalmentMessage {

type: 'warning' | 'error';

text: string;

}

@Component({

templateUrl: './manage-policies.html',

styleUrls: ['./manage-policies.component.scss'],

providers: [PolicyUtils]

})

export class ManagePoliciesComponent implements OnInit, AfterViewInit, OnDestroy {

@ViewChild(NriModalComponent) public initModal;

@ViewChild('documentSelect') public documentSelect: NriInputSelectComponent;

@ViewChild('finishButton') public finishButton: NriButtonComponent;

@ViewChild('lockedModal') public lockedModal: NriModalComponent;

@ViewChild('fileUpload2') public fileUpload2: NriUploadFilesComponent;

@ViewChild('uploadFiles') public uploadFiles: NriUploadFilesComponent;

@ViewChild('modalChangeBusinessType') public modalChangeBusinessType;

public localAllCurrenciesStr = [];

public localReOutTypeOrg: NriInputSelectOptions[] = [];

public localPoolInsurersOptions: NriInputMultiSelectOptions[] = [];

public localBrokerDefaultValue: string = '';

public localBrokerContactDefaultValue: string = '';

public showAutoFields: boolean = true;

public nonPoolInsurerOptions: NriInputMultiSelectOptions[] = [];

public brokerOptions: NriInputMultiSelectOptions[] = [];

public allInsured = [];

public brokerArray = [];

public nonPoolInsurersArray = [];

public poolInsurersArray = [];

public governArray = [];

public poolreArray = [];

public poolrenuclearArray = [];

public poolFACArray = [];

public entryTypeOpt: NriInputSelectOptions[] = [];

public defaultInsuredStations: NriInputMultiSelectOptions[] = [];

public defaultInsuredLocations: NriInputMultiSelectOptions[] = [];

public defaultRenewalHour: string = '0';

public defaultRenewalMinute: string = '0';

public allPolicyStatus: PolicyStatusDTO[] = [];

public localCoverageAddons: NriInputSelectOptions[] = [];

public localPolicySublimitCurrencies: NriInputMultiSelectOptions[] = [];

public localAggregateLimitCurrencies: NriInputMultiSelectOptions[] = [];

public localHotZoneMBCurrency: NriInputMultiSelectOptions[] = [];

public localHotZoneMDCurrency: NriInputMultiSelectOptions[] = [];

public localColdZoneMBCurrency: NriInputMultiSelectOptions[] = [];

public localColdZoneMDCurrency: NriInputMultiSelectOptions[] = [];

public localConventionCurrency: NriInputMultiSelectOptions[] = [];

public localNonConventionCurrency: NriInputMultiSelectOptions[] = [];

public localLiabilityCurrency: NriInputMultiSelectOptions[] = [];

public localmdSingleCurrency: NriInputMultiSelectOptions[] = [];

public localPolicyCreator: string = '';

public dark: boolean = false;

public defaultTax = 10;

public managePolicies: string;

public finishDisable: boolean = true;

public fileAdded = false;

public policy = {} as PolicyDTO;

public users: UserInfoDTO[] = [];

public userOpt: NriInputMultiSelectOptions[] = [];

public underwriterList: NriInputSelectOptions[] = [];

public underwriterList1: NriInputSelectOptions[] = [];

public underwriterList2: NriInputSelectOptions[] = [];

public insuranceEstructure: NriInputSelectOptions[] = [];

public insuredLocations: NriInputMultiSelectOptions[] = [];

public insuranceSelected: number = 0;

public showOtherFrontPanel: boolean = true;

public showBrokerFields: boolean = true;

public showBrokePanel: boolean = false;

public showReiPanel: boolean = false;

public showBrokerReiInPanel: boolean = true;

public policiesColor: string;

public titles: Array<{ check: boolean, name: string, locked: boolean, missing: string }>;

public coverage: any;

public showQuoteNumber: boolean = true;

public startDate: Date;

public endDate: Date;

public localInsuredDefault: NriInputMultiSelectOptions[] = [];

public policyBrokerDefault;

public localPoolInsurerDefault: NriInputMultiSelectOptions[] = [];

public localNonPoolInsurerDefault: NriInputMultiSelectOptions[] = [];

public localcurrency: NriInputMultiSelectOptions[] = [];

public isPolicyRenew = false;

public isPolicyEndorse = false;

public isPolicyQuote = false;

public showReOutFields: boolean = false;

public localPolicyReinsuranceOut: boolean = false;

public addOnOriginalLen: number = 0;

public localPolicyLocations: NriInputMultiSelectOptions[] = [];

public countrysOfLoc: NriInputMultiSelectOptions[] = [];

public defaultHour: string = '0';

public defaultMinute: string = '0';

public updatingStations = false;

public usersOptions: NriInputSelectOptions[] = [];

public brokerContactOptions: NriInputSelectOptions[] = [];

public recordsActive: number;

public initialSactionsVaslidation: boolean = false;

public isUpFrontNCBEnabled: boolean = false;

public isDefferedNCBEnabled: boolean = false;

public isbrokerageFEEEnabled: boolean = false;

public isSurveyFeeEnabled: boolean = false;

public isINLAFFeeEnabled: boolean = false;

public isESCFeeEnabled: boolean = false;

public isOtherIncomeEnable: boolean = false;

public debitNoteTaxList: PolicyPremiumTaxesDTO[] = [] as PolicyPremiumTaxesDTO[];

public creditNoteTaxList: PolicyPremiumTaxesDTO[] = [] as PolicyPremiumTaxesDTO[];

public policyOtherDeductionType = [];

public comboInfo: GetComboInfoInterface;

public brokerageFEEOptions: NriInputSelectOptions[] = [];

public defaultEndHour: string = '0';

public defaultEndMinute: string = '0';

public emailConfirmation: number = 1;

public isStandarSubForm: number = 1;

public active30daysCheck: number = 0;

public autoFilledRate: boolean = true;

public totalCreditNoteTax: number = 0;

public eSCFee = {} as DeductionDTO;

public iNLAFFee = {} as DeductionDTO;

public surveyFee = {} as DeductionDTO;

public cedingInsurerSurveyFee = {} as DeductionDTO;

public cedingInsurerOtherDeductuionFee = {} as DeductionDTO;

public isCedingInsurerSurveyFeeEnabled: boolean = false;

public policyCountryofLocation: string = '';

public hours: NriInputSelectOptions[] = [];

public minutes: NriInputSelectOptions[] = [];

public moreThanOnePolicyCountry = false;

public addOnLen: number = 0;

public editType = 'Edit policy';

public readonly TYPEEDIT = 'EDIT';

public readonly TYPERENEW = 'RENEW';

public readonly TYPEENDORSE = 'ENDORSE';

public readonly TYPEQUOTE = 'QUOTE';

public otherUserEditing: string;

public keepAliveID: any;

public lockId: any;

public notLocked: boolean = false;

public finishedWork: boolean = false;

public startingStep = 1;

public oldTotalReinsuranceShare: boolean = false;

public historyTasks: UnderwritingTaskDTO[] = [];

public policyStations: Array<{ locId: number, stations: number[] }> = [];

public originalPolicyNumber: String;

// #region STEP1

public endHourSelected: boolean = false;

public renewalDate: any;

public validRenewalDate: boolean = true;

public insuredCountry: string;

public allInsuredOptions: NriInputMultiSelectOptions[] = [];

public activeDatesClearCheck: number = 1;

public activeInsuredClearCheck: number = 1;

public stringStandardNonStandard: string = '';

public valid = [required];

public validEndDate = [required, minDate];

public validateRenewalDate = [required];

public validation = [maxLenght];

public isLoading: boolean;

// #endregion STEP1

// #region STEP2

public coverageOptions: NriInputSelectOptions[] = [];

// #endregion STEP2

// #region STEP3

public nriGrossShare: number = 0;

public frontInsurerTypes = [];

public frontInsurerDetails = [];

public totalReOut100 = [];

public showRemoveRelatedPolIcon: boolean = false;

public reOutDeclineSelected: boolean = false;

// #endregion STEP3

// #region STEP4

public autoreinst: NriCheckboxItems[] = [

{

value: 'Automatic reinstatement',

text: 'Automatic reinstatement'

}

];

public singleMd: NriCheckboxItems[] = [

{

value: 'Single deductible for the whole policy',

text: 'Single deductible for the whole policy'

}

];

public variousMd: NriCheckboxItems[] = [

{

value: 'Various deductibles',

text: 'Various deductibles'

}

];

public allCurrenciesOptions: NriInputMultiSelectOptions[] = [];

public allCurrenciesIdOptions: NriInputSelectOptions[] = [];

public allCurrenciesStr: string[] = [];

public localPolicyCurrency: NriInputMultiSelectOptions[] = [];

public mdlimitsopt: NriRadioOptions[] = [

{

value: 0,

text: 'Annual aggregrate limit per policy'

},

{

value: 1,

text: 'Annual aggregate limit per location'

},

{

value: 2,

text: 'No aggregate limit'

}

];

public lifetimelimit: NriCheckboxItems[] = [

{

value: 'Lifetime limit',

text: 'Lifetime limit'

}

];

public annualAggregateLimitOpt: NriCheckboxItems[] = [

{

value: 'Annual Aggregate Value',

text: 'Annual Aggregate Value'

}

];

public GroundupCover: NriCheckboxItems[] = [

{

value: 'Ground-up Cover',

text: 'Ground-up Cover'

}

];

public excessLayerOpt: NriCheckboxItems[] = [

{

value: 'Excess Layer',

text: 'Excess Layer'

}

];

public deductibleNotNilForConvention: NriCheckboxItems[] = [

{

value: 'Deductible not nil for convention cover',

text: 'Deductible not nil for convention cover'

}

];

public deductibleNotNilForNonConvention: NriCheckboxItems[] = [

{

value: 'Deductible not nil for non-convention cover',

text: 'Deductible not nil for non-convention cover'

}

];

public deductibleNotNil: NriCheckboxItems[] = [

{

value: 'Deductible not nil',

text: 'Deductible not nil'

}

];

public alllocs: NriCheckboxItems[] = [

{

value: 'All locations',

text: 'Apply to all locations'

}

];

public allcountries: NriCheckboxItems[] = [

{

value: 'All countries',

text: 'Apply to all countries'

}

];

public allstations: NriCheckboxItems[] = [

{

value: 'All stations',

text: 'Apply to all stations'

}

];

public agg: NriCheckboxItems[] = [

{

value: 'Aggregate limit',

text: 'Aggregate limit'

}

];

public liaOptions: NriCheckboxItems[] = [

{

value: 'Deductibles not nil',

text: 'Deductibles not nil'

}

];

public authoriseCapacityOpt: NriCheckboxItems[] = [

{

value: 'Authorise capacity',

text: 'Authorise capacity'

}

];

public cedingInsurerOtherDeductuionSelectOptions: NriInputSelectOptions[] = [];

// #endregion STEP4

// #region STEP5

public instalmentDate;

public defaultMinDate = new Date('01/01/1900').getTime();

public premium: string;

public creditNoteTaxType: TaxTypeCalculationDTO[] = [];

public readonly ENTER\_AMOUNT\_MESSAGE = 'Amount to be entered';

public eSCFeeSelectOptions: NriInputSelectOptions[] = [];

public readonly CEEDING\_POOL\_MESSAGE = '% of ceding pool gross share Premium after deduction of all NCB and brokerage fees';

public activeTermsClearCheck: number = 1;

public cedingIsurerOtherDeductionTypes: AbstractNameAndIdDTO[] = [];

public surveyFeeSelectOptions: NriInputSelectOptions[] = [];

public cedingInsurerSurveyFeeSelectOptions: NriInputSelectOptions[] = [];

public sizeValidation = [maxLenght];

public validInstalmentDate = [minDate, maxDate];

public negativeBrokerAmount = false;

public readonly CEEDING\_INSURER\_MESSAGE = '% of ceding insurer gross share Premium after deduction of all NCB and brokerage fees';

public readonly NRI\_AFTER\_DEDUCTION\_MESSAGE = '% of NRI gross share Premium after deduction of all NCB and brokerage fees';

public actionButtons = ['Edit', 'Remove'];

public isTaxFormShow: boolean = false;

public showDeductions: boolean = false;

public showIncome: boolean = false;

public showOtherDeductions: boolean = false;

public allDeductionTypes: any[] = [];

public taxTypeCalculations: any[] = [];

public taxType: any[] = [];

public deductionSelectOptions: NriInputSelectOptions[] = [];

public incomeSelectOptions: NriInputSelectOptions[] = [];

public commissionOrFee: NriInputSelectOptions[] = [];

public isFormEdit = {isTaxesEdit: false, isDeductionsEdit: false, isIncomeEdit: false, taxesPosition: 0, deductionsPosition: 0, incomePosition: 0};

public deleteGridRow = {name: '', index: 0};

public taxFormFields: TaxFormFields = {

country: '',

type: {} as NriInputSelectOptions,

percentage: 0,

amount: 0,

};

public deductionForm: DeductionFormFields = {

type: {} as NriInputSelectOptions,

option: {} as NriInputSelectOptions,

percentage: 0,

amount: 0,

comment: '',

commissionOrFee: {} as NriInputSelectOptions

};

public incomeForm: IncomeFormFields = {

type: {} as NriInputSelectOptions,

option: {} as NriInputSelectOptions,

percentage: 0,

amount: 0

};

public deductionsAndFees: DeductionsAndFees[] = [{typeOfDeductions: 'Total Deductions', deduction: '0.00', deductionAmount: '0.00', deductionOption: 'Total', comment: '', commissionOrFee: ''}];

public deductionTableColumns = [

{ name: 'Type of Deductions', cellClass: 'deductionType'},

{ name: 'Deduction (%)'},

{ name: 'Deduction amount'},

{ name: 'Comment'},

{ name: 'Commission Or Fee'}

];

public incomeRows: IncomeRows[] = [{ typeOfIncome: 'Total Income', income: '0.00', incomeAmount: '0.00', incomeOption: 'Total'}];

public incomeTableColumns = [

{ name: 'Type of Income', cellClass: 'incomeType' },

{ name: 'Income (%)' },

{ name: 'Income amount'}

];

public reinsuranceTableRows: ReinsuranceTableRows[] = [{ reinsurerName: 'Total Reinsurance', reinsurer: '0.00', reinsurerPremiumBeforeDeductionOfIpeAndOtherFeesToNri: '0.00', ipeAndOtherFeesToNri: '0.00', reinsurerNetPremium: '0.00' }];

public reinsuranceTableColumns = [

{ name: 'Reinsurer Name', cellClass: 'reinsuranceName' },

{ name: 'Reinsurer %' },

{ name: 'Reinsurer premium before deduction of IPE and other fees to NRI' },

{ name: 'IPE and other fees to NRI' },

{ name: 'Reinsurer Net Premium' }

];

public taxes: Taxes[] = [{country: 'Total Tax', type: '', tax: '0.00', taxAmount: '0.00'}];

public taxesTableColumns = [

{ name: 'Country', cellClass: 'taxesCountry'},

{ name: 'Tax (%)'},

{ name: 'Tax amount'}

];

public incomeOptions: NriInputSelectOptions[] = [

{

value: 'Income %',

text: 'Income %'

},

{

value: 'Amount to be entered',

text: 'Amount to be entered'

}

];

public deductionNames: NriInputSelectOptions[] = [

{

value: 'Upfront NCB',

text: 'Upfront NCB'

},

{

value: 'Deferred NCB',

text: 'Deferred NCB'

},

{

value: 'Brokerage fees',

text: 'Brokerage fees'

},

{

value: 'IPE expense',

text: 'IPE expense'

},

{

value: 'IPE income',

text: 'IPE income'

}

];

public deductionOptions: NriInputSelectOptions[] = [

{

value: 'Deduction %',

text: 'Deduction %'

},

{

value: 'Amount to be entered',

text: 'Amount to be entered'

}

];

public taxesSelectOptions: NriInputSelectOptions[] = [

{

value: '% of NRI Net Premium',

text: '% of NRI Net Premium'

},

{

value: '% of NRI Gross Premium',

text: '% of NRI Gross Premium'

},

{

value: 'Amount to be entered',

text: 'Amount to be entered'

}

];

public cedingInsurerSurveyFeeCheckboxItems: NriCheckboxItems[] = [

{

value: 'Survey fees?',

text: 'Survey fees?'

}

];

public upFrontNCBCheckboxItems: NriCheckboxItems[] = [

{

value: 'upfront NCB?',

text: 'upfront NCB?'

}

];

public defferedNCBCheckboxItems: NriCheckboxItems[] = [

{

value: 'Deferred NCB?',

text: 'Deferred NCB?'

}

];

public brokerageFEECheckboxItems: NriCheckboxItems[] = [

{

value: 'Brokerage Fees',

text: 'Brokerage Fees'

}

];

public surveyFeeCheckboxItems: NriCheckboxItems[] = [

{

value: 'Survey fees?',

text: 'Survey fees?'

}

];

public eSCFeeCheckboxItems: NriCheckboxItems[] = [

{

value: 'ESC fees?',

text: 'ESC fees?'

}

];

public iNLAFeeCheckboxItems: NriCheckboxItems[] = [

{

value: 'INLA fees?',

text: 'INLA fees?'

}

];

public otherIncomeCheckBoxOptions: NriCheckboxItems[] = [

{

value: 'Other income?',

text: 'Other income?'

}

];

public dailyRateCheckboxItem: NriCheckboxItems[] = [

{

value: 'Manual Daily Rate?',

text: 'Manual Daily Rate?'

}

];

// #endregion STEP5

// #region STEP6

// #endregion STEP6

// #region STEP7

// #endregion STEP7

// #region STEP8

public activePremiumClearCheck: number = 0;

public showMsg: boolean = false;

// #endregion STEP8

// #region STEP9

public documents: string;

public isActiveDocBtn: boolean = true;

public docName: string = '';

public serviceUrl: string = process.env.CONTEXT.rest.updateDocument;

public dataContent = [];

public versionSelected: number = 0;

public documentToDelete: DocumentDTO;

public modalMessage: string;

public uploadComponentUrl: string = process.env.CONTEXT.rest.savePolicyFile;

public accentColor: string;

public documentInfo = [] as PolicyDocument[];

public saving: boolean = false;

public fileUploaded: boolean = true;

public savingPolicy: boolean = false;

public documentTypesOpt: NriInputSelectOptions[] = [];

public docType: string = '';

public comboValues: GetComboInfoInterface;

public docDescription = '';

public userDetails: any;

public docVersion: number = 0;

public userAddedFiles: boolean = false;

public docTypeId: number;

public docColumns = [

{ name: 'Name' },

{ name: 'Type' },

{ name: 'Version' },

{ name: 'Added by' },

{ name: 'Last modified' }

];

public SortableColumns = [true, true, true, true, true];

public buttons = ['View', 'Update', 'Download'];

public fileSelected: any[] = [];

public viewFile = {

numberVersions: null,

title: null,

description: null,

typeDoc: null,

addedBy: null,

lastModification: null

};

public resultPolicy: PolicyDTO = null;

// #endregion STEP9

public debitNoteTaxType: TaxTypeCalculationDTO[] = [];

public underwritingCompletedDate: NriCheckboxItems[] = [

{

text: '',

value: ''

}

];

public underwritingTaskCheckboxItems: NriCheckboxItems[] = [

{

text: '',

value: ''

}

];

public underwritingTasks: UnderwritingTaskDTO[] = [];

public message: InstalmentMessage[];

public statusSelected1: number = 0;

public statusSelected2: number = 0;

public enquiryBoundStatus: NriInputSelectOptions[] = [

{

value: '0',

text: 'Enquiry'

},

{

value: '1',

text: 'Bound'

}

];

public enquiryInForceStatus: NriInputSelectOptions[] = [

{

value: '0',

text: 'Enquiry'

},

{

value: '1',

text: 'Policy In Force'

}

];

constructor(

public organisationService: OrganisationService,

private stateService: NriGlobalStateService,

private userService: UserService,

private policyService: PolicyService,

private router: Router,

private documentsService: DocumentService,

private entityLockService: EntityLockService,

private cdRef: ChangeDetectorRef,

private route: ActivatedRoute,

private currencyService: CurrencyService,

private managePolicyService: ManagePoliciesService,

private manageCommunService: ManageCommunService,

private engineeringService: EngineeringService,

private dateUtils: DateUtils,

private currencyExchangeService: CurrencyExchangeService,

private blockScreenService: NriBlockScreenService,

private countryService: CountryService) {

this.dark = this.stateService.getDarkTheme();

this.dark = stateService.getDarkTheme();

this.stateService

.isDarkTheme()

.subscribe((dark) => {

this.dark = dark;

});

this.policiesColor = stateService.getColor('policies');

}

public ngOnInit(): void {

this.policyService.getDeductionType().then((res: any[]) => {

this.allDeductionTypes = res;

});

this.policyService.getTaxTypeCalculation('DEBIT').then((res: any[]) => {

this.taxTypeCalculations = res;

});

this.policyService.getTaxType().then((res: any[]) => {

this.taxType = res;

});

this.managePolicyService.getLoading().subscribe((value) => {

if (value) {

this.isLoading = true;

} else {

this.isLoading = false;

this.blockScreenService.hide();

}

},

// error

(error) => { console.error(error); },

// final

() => {

this.blockScreenService.hide();

});

this.managePolicyService.sendLoading(true);

this.manageCommunService.getComboInfo().then((resp) => {

this.comboInfo = resp;

});

this.managePolicies = 'Manage Policies';

// this.policy.users = [];

this.titles = [...this.managePolicyService.titles];

this.policy = this.managePolicyService.policy;

this.getPolicy();

this.userService.getUserDetails()

.then((details) => {

this.userDetails = details;

});

this.commissionOrFeePopulate();

}

public getPolicy() {

this.route.queryParams

.subscribe((queryParams) => {

this.manageCommunService.activeStep = queryParams.StepId ? queryParams.StepId : this.manageCommunService.activeStep;

this.startingStep = queryParams.StepId ? +queryParams.StepId : 1;

});

this.route.params.subscribe(async (params) => {

this.setEditType(!!params.type ? params.type : '');

if (!this.isPolicyEndorse && !this.isPolicyRenew && !this.isPolicyQuote) {

await this.entityLockService.checkPolicyLock(+params['id'], null).then(

async (response: LockResult) => {

if (response.locked) {

await this.openLockedModal(response.lockingUserId);

} else {

this.lockId = response.lockId;

this.keepAliveID = setInterval(() => {

this.entityLockService.keepAlive(response.lockId).then((lock: any) => {

if (!lock || !lock.lockId) {

this.entityLockService.checkPolicyLock(+params['id'], this.lockId).then(

async (responseLock: LockResult) => {

if (!responseLock || responseLock.lockId !== this.lockId) {

if (!responseLock.locked) {

this.notLocked = true;

}

await this.openLockedModal(responseLock.lockingUserId);

}

});

}

});

}, 50000);

}

}

);

}

if (this.otherUserEditing) {

return;

}

this.blockScreenService.show('Loading Policy...');

this.policyService.getPolicyById(params.id).subscribe(async (res: PolicyDTO) => {

const tempHistoryList: UnderwritingTaskDTO[] = [];

if (res.underwritingProcess && res.underwritingProcess.tasks && res.underwritingProcess.tasks.length > 0) {

res.underwritingProcess.tasks.forEach((task: UnderwritingTaskDTO) => {

const temp = Object.assign({}, task);

tempHistoryList.push(temp);

});

}

this.historyTasks = tempHistoryList;

this.policy = res;

if (this.isPolicyRenew) {

await this.policyService.getAllUnderwritingTasks().then((response: UnderwritingTaskDTO[]) => {

this.policy.underwritingProcess = {} as UnderwritingProcessDTO;

this.policy.underwritingProcess.secondUnderwriter = {} as UserDTO;

this.policy.underwritingProcess.underwriter = {} as UserDTO;

this.policy.underwritingProcess.tasks = response;

});

this.historyTasks = [];

}

this.loadPolicy();

this.getAllCurrencys();

this.blockScreenService.hide();

// The line below check if the the policy is from a year below 2020

if (!this.policy.underwritingProcess) {

const titleIndex = this.titles.findIndex((title) => title.name === 'Underwriting Process & Checks');

if (titleIndex > -1) {

this.titles.splice(titleIndex, 1);

}

}

},

(error) => {

// this.openFailureModal();

}

);

});

this.getAllMinutes();

this.getAllHours();

}

public ngOnDestroy(): void {

if (this.keepAliveID) {

clearInterval(this.keepAliveID);

}

if (this.lockId) {

this.entityLockService.releaseLock(this.lockId);

}

}

public ngAfterViewInit(): void {

if (this.startingStep) {

const steps = [

'01-basic-details',

'02-insurance',

'03-structure',

'04-exposure',

'05-premium',

'06-indicators',

'07-underwriting-process',

'08-compliance',

'09-documents'

];

this.activateStepView(this.startingStep, steps[this.startingStep - 1]);

}

}

@HostListener('window:beforeunload')

public canDeactivate() {

if (this.stateService.userTimedOut) {

return true;

} else {

return this.finishedWork;

}

}

// #region STEP1

public setPlusOneDay(): void {

if (this.defaultEndHour === '23' && this.defaultEndMinute === '59') {

this.endHourSelected = true;

} else {

this.endHourSelected = false;

}

}

public setEntryType(event) {

this.policy = PolicyUtils.setEntryType(this.policy, event);

}

public setPolicyAdditionalInfo(event) {

if (event !== undefined) {

this.policy.additionalInformation = event;

}

}

// FIXME: unused function

public setStartDate(

date: Date,

policyEndDate: NriInputDatepickerComponent,

policyRenewalDate: NriInputDatepickerComponent): void {

if (date) {

policyEndDate.errorMsg = '';

const startDate = new Date(date);

this.startDate = startDate;

const fullDate = new Date(startDate.getFullYear(), startDate.getMonth(), startDate.getDate(),

+this.defaultHour, +this.defaultMinute, 0, 0);

this.policy.startDate = fullDate;

if (!this.manageCommunService.isPolicyEndorse && !this.manageCommunService.isPolicyRenew && !this.policy.endorsementNumber) {

this.policy.yearOfAccount = fullDate.getFullYear();

}

policyEndDate.defaultSimpleDate = fullDate;

policyEndDate.updateDatepickerSimpleMinDate(fullDate);

if (this.policy.endDate && this.policy.startDate > this.policy.endDate) {

policyEndDate.clearSimple();

this.policy.endDate = null;

}

if (this.policy.renewalDate && this.policy.startDate > this.policy.renewalDate) {

policyRenewalDate.clearSimple();

this.policy.renewalDate = null;

}

if (this.policy.instalmentDate && !this.isLoading) {

this.policy.instalmentDate.forEach((instDate, index) => {

if (!!instDate) {

if (this.startDate.getTime() > this.dateUtils.convertToStartofDay(instDate)) {

this.policy.instalmentDate[index] = null;

}

}

});

}

if ((!this.policy.currencyID || (this.policy.currencyID.code && this.policy.currencyID.code !== 'GBP'))

&& !this.policy.rOEDate && !this.policy.rOE) {

this.setPolicyROEDate(this.policy.startDate);

}

} else {

this.policy.startDate = undefined;

if (!this.manageCommunService.isPolicyEndorse && !this.manageCommunService.isPolicyRenew && !this.policy.endorsementNumber) {

this.policy.yearOfAccount = undefined;

}

}

}

public setRenewalDate(date: Date): void {

if (date) {

const renewalDate = new Date(date);

const fullDate = new Date(renewalDate.getFullYear(), renewalDate.getMonth(), renewalDate.getDate(),

+this.defaultRenewalHour, +this.defaultRenewalMinute, 0, 0);

this.policy.renewalDate = fullDate.getTime();

this.renewalDate = fullDate.getTime();

this.renewalDateIsValid();

} else {

this.policy.renewalDate = undefined;

this.renewalDate = undefined;

}

}

/\*\*

\* Function to validate the renewal date of the policy

\*/

public renewalDateIsValid(): void {

// Validate if renewaldate has value and check if that value is latest than enddate

!this.policy.renewalDate || (this.policy.renewalDate >= this.policy.endDate) ? this.validRenewalDate = true : this.validRenewalDate = false;

}

public setEndDate(

date: Date,

policyRenewalDate: NriInputDatepickerComponent): void {

if (date) {

policyRenewalDate.errorMsg = '';

const endDate = new Date(date);

const fullDate = new Date(endDate.getFullYear(), endDate.getMonth(), endDate.getDate(),

+this.defaultEndHour, +this.defaultEndMinute, 0, 0);

const renewal = new Date(endDate.getFullYear(), endDate.getMonth(), endDate.getDate(),

+this.defaultEndHour, +this.defaultEndMinute, 0, 0);

this.policy.endDate = fullDate;

policyRenewalDate.updateDatepickerSimpleMinDate(renewal);

this.setRenewalDate(renewal);

if (this.policy.instalmentDate && !this.isLoading) {

this.policy.instalmentDate.forEach((instDate, index) => {

if (instDate) {

const instalmentDate = new Date(instDate);

const endDateVerification = new Date(this.policy.endDate);

this.instalmentDateVerification(endDateVerification, instalmentDate, index);

}

});

}

this.renewalDateIsValid();

} else {

this.policy.endDate = null;

this.endDate = null;

policyRenewalDate.clearSimple();

this.policy.renewalDate = null;

this.renewalDate = null;

}

}

public setDecimals(num: number, decimalPoints?: number) {

if (decimalPoints || decimalPoints === 0) {

return PolicyUtils.setDecimals(num, decimalPoints);

} else {

return PolicyUtils.setDecimals(num);

}

}

public daydiff(first, second) {

this.policy = PolicyUtils.daydiff(this.policy, first, second);

}

public selectPolicyInsured(event) {

if (event != null) {

if (event.length === 0) {

this.policy.locationId = undefined;

this.policy.insured = undefined;

this.policy.policyLocations = [];

this.policy.countrysOfLoc = [];

this.policy.stationsOfLoc = [];

this.insuredLocations = [];

this.defaultInsuredLocations = [];

this.defaultInsuredStations = [];

} else {

if (!this.policy.insured || this.policy.insured.id !== +event) {

this.policy.locationId = [] as LocationDTO[];

this.insuredLocations = [];

this.policy.insured = {} as OrganisationDTO;

this.policy.insured.id = +event;

this.policy.policyLocations = [];

this.policy.stationsOfLoc = [];

this.policy.countrysOfLoc = [];

this.defaultInsuredLocations = [];

this.defaultInsuredStations = [];

if (this.policy.insured.id) {

this.organisationService.getOrganization(this.policy.insured.id).then((response: OrganisationDTO) => {

if (response) {

const insured = response;

this.policy.insured = insured;

this.insuredCountry = insured.countryId.name;

insured.location.forEach((loc) => {

if (loc.enabled) {

this.insuredLocations.push({ id: loc.id.toString(), name: loc.name } as NriInputMultiSelectOptions);

}

});

this.updateSublimitsLocationsAndCountries();

if (!this.policy.isLoading) {

this.policy = PolicyUtils.defaultStand(this.policy);

}

setTimeout(() => {

this.setDefaultInsured();

}, 10);

}

});

}

}

}

}

this.defaultStand();

}

public updateSublimitsLocationsAndCountries() {

this.policy = PolicyUtils.updateSublimitsLocationsAndCountries(this.policy);

}

public defaultStand() {

switch (this.policy.standardNonStandard) {

case true:

this.stringStandardNonStandard = 'Standard';

break;

case false:

this.stringStandardNonStandard = 'Non-Standard';

break;

default:

this.stringStandardNonStandard = '';

}

}

public getOrganisationlist() {

this.organisationService.getOrganizationLists(true).then((response) => {

this.allInsured = response['INSURED'];

this.brokerArray = response['BROKER'];

this.nonPoolInsurersArray = response['NON\_POOL\_INSURER'];

this.poolInsurersArray = response['POOL\_INSURER'];

this.governArray = response['GOV'];

this.poolreArray = response['POOL\_RE'];

this.poolrenuclearArray = response['POOL\_RE\_NUCLEAR'];

this.poolFACArray = response['FAC'];

const tempBrokerOptions = [] as NriInputMultiSelectOptions[];

const tempNonPoolOptions = [] as NriInputMultiSelectOptions[];

const tempPoolOptions = [] as NriInputMultiSelectOptions[];

const tempPoolAliasOptions = [] as NriInputMultiSelectOptions[];

const tempGovernmentOptions = [] as NriInputMultiSelectOptions[];

const tempGovernmentaliasOptions = [] as NriInputMultiSelectOptions[];

const tempPoolReOptions = [] as NriInputMultiSelectOptions[];

const tempPoolReAliasOptions = [] as NriInputMultiSelectOptions[];

const tempPoolReNuclearOptions = [] as NriInputMultiSelectOptions[];

const tempPoolReNuclearAliasOptions = [] as NriInputMultiSelectOptions[];

const tempPoolFacOptions = [] as NriInputMultiSelectOptions[];

const tempPoolFacOptAlias = [] as NriInputMultiSelectOptions[];

tempBrokerOptions.push({ id: 'N/A', name: 'N/A' });

this.brokerArray.forEach((element) => {

tempBrokerOptions.push({ id: element.id, name: element.fullName });

});

this.brokerOptions = tempBrokerOptions;

this.nonPoolInsurersArray.forEach((element) => {

tempNonPoolOptions.push({ id: element.id, name: element.fullName });

});

this.nonPoolInsurerOptions = tempNonPoolOptions;

this.poolInsurersArray.forEach((element) => {

tempPoolOptions.push({ id: element.id, name: element.fullName });

tempPoolAliasOptions.push({ id: element.id, name: element.nrialias });

});

this.policy.poolInsurersOptions = tempPoolOptions;

this.policy.poolInsurersAliasOptions = tempPoolAliasOptions;

this.governArray.forEach((element) => {

tempGovernmentOptions.push({ id: element.id, name: element.fullName });

tempGovernmentaliasOptions.push({ id: element.id, name: element.nrialias });

});

this.policy.governArrayOpt = tempGovernmentOptions;

this.policy.governAliasArrayOpt = tempGovernmentaliasOptions;

this.poolreArray.forEach((element) => {

tempPoolReOptions.push({ id: element.id, name: element.fullName });

tempPoolReAliasOptions.push({ id: element.id, name: element.nrialias });

});

this.policy.poolreArrayOpt = tempPoolReOptions;

this.policy.poolreAliasArrayOpt = tempPoolReAliasOptions;

this.poolrenuclearArray.forEach((element) => {

tempPoolReNuclearOptions.push({ id: element.id, name: element.fullName });

tempPoolReNuclearAliasOptions.push({ id: element.id, name: element.nrialias });

});

this.policy.poolrenuclearArrayOpt = tempPoolReNuclearOptions;

this.policy.poolrenuclearAliasArrayOpt = tempPoolReNuclearAliasOptions;

this.poolFACArray.forEach((element) => {

tempPoolFacOptions.push({ id: element.id, name: element.fullName });

tempPoolFacOptAlias.push({ id: element.id, name: element.nrialias });

});

this.policy.poolFACopt = tempPoolFacOptions;

this.policy.poolFACoptAlias = tempPoolFacOptAlias;

});

}

public removeLocation(index) {

if (this.policy.locationId[index].id && this.policy.locationId[index].name) {

this.insuredLocations.push({ id: this.policy.locationId[index].id.toString(), name: this.policy.locationId[index].name });

}

this.policy = PolicyUtils.removeLocation(this.policy, index);

}

public getAllCoverages() {

this.managePolicyService.getAllCoverages().then((result) => {

this.coverageOptions = result.options;

this.policy.coverage = result.coverages;

});

}

public selectPolicyCoverage(event) {

const updateCapacity = event && (!this.policy.coverageID || this.policy.coverageID.coverageID !== +event);

this.policy = PolicyUtils.selectPolicyCoverage(this.policy, event);

this.getCoverageLimits(updateCapacity);

if (this.policy.coverageID.coverageName === 'Material Damage') {

this.policy.deductibleID[0] = {} as DeductibleDTO;

this.policy.deductibleID[1] = {} as DeductibleDTO;

this.policy.deductibleID[2] = {} as DeductibleDTO;

this.policy.deductibleID[3] = {} as DeductibleDTO;

}

this.setBooleansByCoverage();

}

public setBooleansByCoverage() {

if (this.policy.coverageID && this.policy.coverageID.coverageName) {

this.policy = PolicyUtils.setBooleansByCoverage(this.policy);

}

}

public addAddOn() {

this.policy.addOnID.push({} as AddOnDTO);

this.policy.addOnLen++;

// this.showRemoveAddIcon = true;

}

public selectInsuranceAddOns(event, index) {

this.policy = PolicyUtils.selectInsuranceAddOns(this.policy, event, index);

}

public removeAddOn(index, addon: AddOnDTO) {

if (!!this.policy.addOnID) {

this.policy.addOnID.forEach((element) => {

if (element === addon && element.id) {

this.policy.addonOptions.push({ value: element.id.toString(), text: element.name });

this.policy.addonOptions.sort((a, b) => a.text.localeCompare(b.text));

}

});

}

if (addon.name === 'Including BI') {

this.policy.deductibleID.splice(4, 1);

}

this.policy = PolicyUtils.removeAddOn(this.policy, index);

this.policy.addOnLen--;

}

public selectPolicyLocation(event, insuredStations: QueryList<NriInputSelectMultipleComponent>) {

if (event != null && !this.policy.isLoading && insuredStations) {

const enabledLocations: LocationDTO[] = [];

if (this.policy.locationId.length > 0) {

this.policy.locationId.forEach((loc) => {

if (loc.enabled) {

enabledLocations.push(loc);

}

});

}

const prevPolicyLocations = enabledLocations.length;

if (prevPolicyLocations !== event.length) {

this.policy = PolicyUtils.selectPolicyLocation(this.policy, event);

if (prevPolicyLocations === event.length - 1) {

this.updatingStations = true;

setTimeout(() => {

const newStation = insuredStations.toArray()[insuredStations.toArray().length - 1];

if (newStation) {

if (this.policy.stationIDList[this.policy.stationIDList.length - 1] !== [] &&

(newStation.selectedOptions.length === 0 || newStation.selectedOptions[0] === undefined)

) {

newStation.setAllOptions();

}

}

this.updatingStations = false;

}, 150);

} else if (event.length === this.policy.insured.location.length && prevPolicyLocations !== event.length) {

this.updateInsuredStations(insuredStations);

}

const tempDefaultLocations = [];

this.policy.locationId.forEach((loc) => {

if (loc.enabled) {

tempDefaultLocations.push(loc);

}

});

if (this.defaultInsuredLocations.length !== tempDefaultLocations.length) {

this.defaultInsuredLocations = tempDefaultLocations.map( (loc) => {

return { id: loc.id.toString(), name: loc.name };

});

}

}

if (this.policyStations && this.policyStations.length > 0) {

// add a new location to the location-station list

if (this.policy.locationId.length > this.policyStations.length) {

this.policy.locationId.forEach((locationID) => {

let locExists = false;

this.policyStations.forEach((loc) => {

locExists = locationID.id === loc.locId ? true : false;

});

if (!locExists) {

this.policyStations.push({locId: locationID.id, stations: []});

}

});

}

// remove a location from the location-station list

if (this.policy.locationId.length < this.policyStations.length) {

this.policyStations.forEach((loc) => {

let locExists = false;

this.policy.locationId.forEach((locationID) => {

locExists = locationID.id === loc.locId ? true : false;

});

if (!locExists) {

this.policyStations.splice(this.policyStations.findIndex((ps) => ps.locId === loc.locId), 1);

}

});

}

// add location do locations list

this.policy.policyLocations = [];

if (this.policy.locationId.length > 0) {

this.policy.locationId.forEach((loc) => {

this.policy.policyLocations.push({ id: loc.id.toString(), name: loc.name });

});

}

}

}

}

public updateInsuredStations(insuredStations: QueryList<NriInputSelectMultipleComponent>) {

// timeout makes sure the list of components is updated correctly

this.updatingStations = true;

setTimeout(() => {

if (insuredStations.toArray().length > 0) {

insuredStations.toArray().forEach((element) => {

element.setAllOptions();

});

}

this.updatingStations = false;

}, 150);

}

public selectPolicyStation(insuredStations: NriInputSelectMultipleComponent, loc: LocationDTO) {

if (!this.policy.isLoading && !this.updatingStations) {

const selectedOptions = [];

selectedOptions.push(insuredStations.selectedOptions);

this.pushPolicyStations(selectedOptions, loc.id);

}

}

public pushPolicyStations(selectedOptions, locationId) {

this.policyStations[this.policyStations.findIndex((ps) => ps.locId === locationId)].stations = [];

selectedOptions[0].forEach((stationId) => {

this.policyStations[this.policyStations.findIndex((ps) => ps.locId === locationId)].stations.push(+stationId);

});

const stationsList = [];

this.policyStations.forEach((locStations) => {

if (locStations.stations.length > 0) {

stationsList.push(locStations.stations);

}

});

this.policy = PolicyUtils.selectPolicyStation(this.policy, stationsList);

}

// #endregion STEP2

// #region STEP3

public setNRIGrossShareTotalPoolShare(event) {

if (!isNaN(event)) {

this.policy.totalOverallPoolShare = +event;

this.calcNRIshare();

this.policy.nRIGSAfterDeductionPrimaryCeding = PolicyUtils.calcNRIGSAfterDeductionPrimaryCeding(this.policy);

this.policy.nriPremiumBeforeIpeAndOtherFees = PolicyUtils.calcNRIPremiumBeforeIPEAndFees(this.policy);

this.recalculateMultiShares(this.policy.nriGrossShare);

}

}

public recalculateMultiShares(event) {

this.policy.policyShareID.forEach((share, index) => {

if (share.insurerRole === 'reout') {

let result = 0;

if (share.reinsurerPct && event) {

result = this.setDecimals((((share.reinsurerPct) \* (+event)) / 100), 4);

}

share.share100Percent = result;

}

});

}

public calcNRIshare() {

if (this.policy.totalOverallPoolShare === undefined) {

this.policy.totalOverallPoolShare = 0;

}

if (this.policy.totalPoolInsuranceShare === undefined) {

this.policy.totalPoolInsuranceShare = 0;

}

this.policy.nriGrossShare = Number(math.divide(

math.BigNumber(math.multiply(math.BigNumber(

+this.policy.totalOverallPoolShare), math.BigNumber(+this.policy.totalPoolInsuranceShare))), 100));

this.policy.nriGrossShare = this.setDecimals(this.policy.nriGrossShare, 4);

if (+this.policy.nriGrossShare !== +this.nriGrossShare) {

this.nriGrossShare = this.policy.nriGrossShare ? +this.policy.nriGrossShare : 0;

this.calcNriNetMaximun();

}

if (this.policy.grossPremium100Pct && this.policy.grossPremium100Pct > 0) {

this.setNRIGrossPremium();

}

return this.nriGrossShare;

}

public setCedantShare1(event) {

if (!isNaN(event)) {

this.policy.cedantShare1 = +event;

this.calcTotalCedants();

this.updateTotalPooolInsShare();

}

}

public setCedantShare2(event) {

if (!isNaN(event)) {

this.policy.cedantShare2 = +event;

this.calcTotalCedants();

this.updateTotalPooolInsShare();

}

}

public calcTotalCedants() {

this.policy = PolicyUtils.calcTotalCedants(this.policy);

return this.policy.totalCedantShare;

}

public updateTotalPooolInsShare() {

if (!this.policy.isLoading && this.oldTotalReinsuranceShare) {

this.policy.totalPoolInsuranceShare = this.policy.totalCedantShare;

}

}

public setTotalPoolInsuranceShare(event) {

if (!isNaN(event)) {

this.policy.totalPoolInsuranceShare = +event;

this.oldTotalReinsuranceShare = this.policy.totalPoolInsuranceShare === this.policy.totalCedantShare ? true : false;

}

}

public setCedingPoolReference(event) {

if (event !== undefined && event !== '') {

this.policy.cedingPoolReference = event;

if (this.policy.cedingPoolReference.length > 50) {

this.policy.cedingPoolReference = this.policy.cedingPoolReference.substring(0, 50);

}

}

}

public setBroker(event) {

if (!!event) {

if (event.length === 0 || event[0] === 'N/A') {

this.policy.broker = undefined;

this.policy.brokerContact = undefined;

this.brokerContactOptions = [];

} else if (!this.policy.broker || +event[0] !== this.policy.broker.id) {

this.policy.broker = {} as OrganisationDTO;

this.policy.broker.id = +event[0];

this.policyBrokerDefault = [this.brokerOptions.find((element) => element.id === event[0])] as NriInputMultiSelectOptions[];

this.policy.brokerContact = {} as ContactDetailsDTO;

this.policy.broker.contacts = [] as ContactDetailsDTO[];

this.brokerContactOptions = [];

if (this.policy.broker.id) {

this.organisationService.getOrganization(this.policy.broker.id).then((response: OrganisationDTO) => {

if (response) {

const broker = response;

this.policy.broker = broker;

if (!!broker.contacts) {

broker.contacts.forEach((contact) => {

this.brokerContactOptions.push({ value: contact.contactDetailID.toString(), text: contact.fullName });

});

}

}

});

}

}

}

}

public setBrokerContact(event) {

this.policy = PolicyUtils.setBrokerContact(this.policy, event);

const contact = this.policy.broker.contacts.find((element) => element.contactDetailID === +event);

this.policy.brokerContact = contact;

}

public selectReinsurance(event) {

this.policy = PolicyUtils.selectReinsurance(this.policy, event);

}

public setNonPoolInsurer(event) {

this.policy = PolicyUtils.setNonPoolInsurer(this.policy, event);

const tempNonPoolOptions = this.nonPoolInsurerOptions.find((element) => element.id === event[0]);

this.localNonPoolInsurerDefault = tempNonPoolOptions !== undefined ? [tempNonPoolOptions] as NriInputMultiSelectOptions[] : [];

}

public setPoolInsurer(event) {

this.policy = PolicyUtils.setPoolInsurer(this.policy, event);

const tempPoolOptions = this.policy.poolInsurersOptions.find((element) => element.id === event[0]);

this.localPoolInsurerDefault = tempPoolOptions !== undefined ? [tempPoolOptions] as NriInputMultiSelectOptions[] : [];

}

public setShare2(event, index) {

if (!!event) {

this.policy.policyShareID[index].share2 = +event;

}

}

public updateShare2() {

if (!this.policy.isLoading && this.policy.policyShareID && this.policy.nriGrossShare) {

this.policy.policyShareID.forEach((reout) => {

if (!reout.share2 || reout.share2 === null || reout.share2 === 0 || reout.oldShare2) {

if (reout.insurerRole === 'reout') {

reout.share2 = this.policy.nriGrossShare;

}

}

});

}

}

public selectNriLead(event) {

this.policy = PolicyUtils.selectNriLead(this.policy, event);

}

public getFrontingOrganizationTypes() {

if (this.frontInsurerTypes.length === 0) {

this.organisationService.getFrontingOrganizationTypes()

.then((response) => {

this.frontInsurerDetails = response as OrganisationTypeDTO[];

(response as OrganisationTypeDTO[]).forEach((element) => {

this.frontInsurerTypes.push(

{

value: element.organisationTypeCode.toString(),

text: element.organisationType

});

});

});

}

}

public setFrontInsurerType(event, index) {

if (!this.policy.isLoading) {

if (!!event) {

this.policy = PolicyUtils.setFrontInsurerType(this.policy, this.frontInsurerDetails, event, index);

}

}

}

public removeInsurer(index) {

this.policy = PolicyUtils.removeInsurer(this.policy, index);

if (this.policy.reShare1Array) {

this.policy.reShare1Array.splice(index, 1);

}

if (this.totalReOut100) {

this.totalReOut100.splice(index, 1);

}

if (this.policy.totalReOut100Arr) {

this.policy.totalReOut100Arr.splice(index, 1);

}

this.calcTotalReOut();

this.calcTotalReOut100();

this.updateReinsuranceTableRows();

}

public setFrontInsurerName(event, index) {

if (!this.policy.isLoading) {

if (!!event) {

this.policy = PolicyUtils.setFrontInsurerName(this.policy, event, index);

}

}

}

public calcFrontInsurerShare(index) {

this.policy = PolicyUtils.calcFrontInsurerShare(this.policy, index);

}

public addFrontInsurer() {

this.policy = PolicyUtils.addFrontInsurer(this.policy);

}

public setReOut(event) {

if (event.length === 0 || event.length === 2) {

this.showReOutFields = false;

this.totalReOut100 = [];

this.policy.totalReOut100Arr = [];

this.policy.reShare1Array = [];

this.showAutoFields = false;

if (!!this.policy.policyShareID) {

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

if (this.policy.policyShareID[i].insurerRole === 'reout') {

this.policy.policyShareID.splice(i, 1);

}

}

}

this.updateReinsuranceTableRows();

} else {

this.showReOutFields = true;

}

if (!this.policy.nriGrossShare) {

this.policy.nriGrossShare = 0;

}

if (!this.policy.totalReinsurerPctOf100) {

this.policy.totalReinsurerPctOf100 = 0;

}

this.policy = PolicyUtils.setReOut(this.policy, event);

this.localPolicyReinsuranceOut = this.policy.reinsuranceOut;

this.calcTotalReOut();

this.calcNRIshare();

this.calcTotalReOut100();

}

public selectReOutTypeOrg(event, index, polShare: PolicyShareDTO, viewChild?) {

if (!this.policy.isLoading) {

this.policy = PolicyUtils.selectReOutTypeOrg(this.policy, event, index);

if (typeof viewChild.isArray !== 'undefined' ) {

viewChild.toArray()[index].selectedOptions = [];

} else {

viewChild.selectedOptions = [];

}

}

}

public selectReOutNameOrg(event, index: number, polShare: PolicyShareDTO) {

if (!this.policy.isLoading && event) {

this.policy = PolicyUtils.selectReOutNameOrg(this.policy, event, index);

this.calcTotalReOut100();

this.updateReinsuranceTableRows();

}

}

public setShare1(event, index) {

if (event) {

this.policy.policyShareID[index].reinsurerPct = +event;

let totalDeductionPct = 0;

if (this.deductionsAndFees.length > 0) {

this.deductionsAndFees.forEach((deduction) => {

totalDeductionPct = +deduction.deduction;

});

}

const reinsurerPremiumBeforeDeductions = this.policy.nriGrossPremiumShare ? ((this.policy.nriGrossPremiumShare

\* (100 - totalDeductionPct) / 100) \* +event / 100) : 0;

const reinsurerPremiumAfterDeductions = this.policy.nriGrossPremiumShare ?

(((this.policy.nriGrossPremiumShare

\* (100 - totalDeductionPct) / 100) \* +event / 100) +

((this.policy.nriGrossPremiumShare

\* (100 - totalDeductionPct) / 100) \* +event / 100) \*

((this.policy.ipeIncomePct ? this.policy.ipeIncomePct : 0) \* -1) / 100) : 0;

const ipeAndOtherFeesToNri = this.policy.ipeIncomePct ? reinsurerPremiumBeforeDeductions \* this.policy.ipeIncomePct / 100 : 0;

this.policy.policyShareID[index].premiumReinsurerBeforeIpe = reinsurerPremiumBeforeDeductions;

this.policy.policyShareID[index].iPEAndOtherFeesToNRI = ipeAndOtherFeesToNri;

this.policy.policyShareID[index].premiumReinsurerAfterIpe = reinsurerPremiumAfterDeductions;

this.updateReinsuranceTableRows();

this.calcNriNetSharePct();

this.recalculateIncome();

}

}

public pushShare1(event, index) {

this.policy = PolicyUtils.pushShare1(this.policy, event, index);

this.calcTotalReOut100();

}

public multiShares(val1, val2, index) {

let result = 0;

if (!!val1 && !!val2) {

result = this.setDecimals((((+val1) \* (+val2)) / 100), 4);

}

this.totalReOut100[index] = result;

this.policy.policyShareID[index].share100Percent = result;

this.calcTotalReOut100();

return result;

}

public calcTotalReOut100() {

if (this.policy.policyShareID && this.policy.policyShareID.length > 0) {

let totalReOut: number = 0;

this.policy.policyShareID.forEach((reout) => {

if (reout.insurerRole === 'reout') {

totalReOut += reout.share100Percent;

}

this.policy.totalReinsurerPctOf100 = this.setDecimals(totalReOut, 4);

});

}

}

public calcTotalReOut(): number {

this.policy = PolicyUtils.calcTotalReOut(this.policy);

this.calcCedingInsurerIPEAndAllDeductions();

return this.policy.totalReinsurancePct;

}

public selectReOutStatus(event, index) {

this.policy = PolicyUtils.selectReOutStatus(this.policy, event, index);

if (!!event && event === 'Declined') {

this.reOutDeclineSelected = true;

} else {

this.reOutDeclineSelected = false;

}

this.calcTotalReOut100();

}

public addReInsurer() {

this.getReoutOrganizationTypes();

this.policy = PolicyUtils.addReInsurer(this.policy);

this.showAutoFields = true;

this.updateReinsuranceTableRows();

}

public selectPolicyRelated(event, index) {

this.policy = PolicyUtils.selectPolicyRelated(this.policy, event, index);

}

public removeRelatedPol(index) {

this.policy = PolicyUtils.removeRelatedPol(this.policy, index);

}

public addRelatedPolicy() {

this.policy.relatedPolicyID.push({} as PolicyDTO);

this.showRemoveRelatedPolIcon = true;

}

public setDeductibleComment(event, index) {

if (event !== undefined) {

this.policy.deductibleID[index].deductibleComment = event;

if (this.policy.deductibleID[index].deductibleComment.length > 255) {

this.policy.deductibleID[index].deductibleComment = this.policy.deductibleID[index].deductibleComment.substring(0, 255);

}

}

}

public addDeductible() {

this.policy = PolicyUtils.addDeductible(this.policy);

}

public removeDeductible(index: number) {

this.policy = PolicyUtils.removeDeductible(this.policy, index);

}

public setIndemnityPeriod(limit: SublimitDTO, event) {

if (event !== '') {

if (+limit.indemnityPeriod !== +event) {

limit.indemnityPeriod = +event;

this.calcDailyRate(limit);

}

} else {

limit.indemnityPeriod = 0;

limit.bIDailyRate = 0;

}

}

public enableDailyRateInput(event, limit: SublimitDTO) {

if (event && event.length > 0) {

limit.biDailyRateManual = true;

} else {

limit.biDailyRateManual = false;

this.calcDailyRate(limit);

}

}

public setbIDailyRate(limit: SublimitDTO, event) {

if (!isNaN(event)) {

limit.bIDailyRate = +event;

}

}

public setAutoReinstatement(event) {

this.policy = PolicyUtils.setAutoReinstatement(this.policy, event);

}

public onSelectDeductibleNotNil(event) {

this.policy.deductibleID = [];

if (event && event.length > 0) {

this.policy.deductibleID.push({ isNotNil: true } as DeductibleDTO);

this.policy.isNotNilChecked = true;

if (this.policy.currencyID && this.policy.currencyID.id) {

this.policy.deductibleID[0].currencyID = { id: this.policy.currencyID.id } as CurrencyDTO;

this.policy.deductibleID[0].localCurrency = [{ id: this.policy.currencyID.code, name: this.policy.currencyID.name }];

this.policy.deductibleID[0].rate = 1;

}

} else {

this.policy.isNotNilChecked = false;

}

}

public setMDDeductibleValueCurrency(event, obj, type?) {

if (event !== undefined && event.length > 0 && event[0] !== undefined) {

const currency = {} as CurrencyDTO;

const roe = {

from: null,

to: this.policy.currencyID && this.policy.currencyID.code ? this.policy.currencyID.code : ''

};

let option = {} as NriInputSelectOptions;

let optionCode = {} as NriInputMultiSelectOptions;

this.getAllCurrencys();

if (this.policy.allCurrenciesIdOptions && this.policy.allCurrenciesOptions && this.policy.allCurrenciesStr) {

option = this.policy.allCurrenciesIdOptions[this.policy.allCurrenciesStr.indexOf(event[0])];

optionCode = this.policy.allCurrenciesOptions[this.policy.allCurrenciesStr.indexOf(event[0])];

currency.id = +option.value;

currency.code = optionCode.id;

roe.from = currency.code;

this.currencyExchangeService.getExchangeRate(roe).then((response) => {

if (!!response) {

obj.currency = currency.id;

obj.code = currency.code;

obj.rate = +response;

this.pushMdDeductibles();

this.getMinDeductible(this.policy);

}

});

}

if ((type === 'localHotZoneMBCurrency') && optionCode.id !== this.localHotZoneMBCurrency[0].id) {

this.localHotZoneMBCurrency = [{id: optionCode.id, name: optionCode.name}];

}

if ((type === 'localHotZoneMDCurrency') && optionCode.id !== this.localHotZoneMDCurrency[0].id) {

this.localHotZoneMDCurrency = [{id: optionCode.id, name: optionCode.name}];

}

if ((type === 'localColdZoneMBCurrency') && optionCode.id !== this.localColdZoneMBCurrency[0].id) {

this.localColdZoneMBCurrency = [{id: optionCode.id, name: optionCode.name}];

}

if ((type === 'localColdZoneMDCurrency') && optionCode.id !== this.localColdZoneMDCurrency[0].id) {

this.localColdZoneMDCurrency = [{id: optionCode.id, name: optionCode.name}];

}

}

}

public showvariousMdField(event) {

if (this.policy.currencyID) {

const option = { id: this.policy.currencyID.code.toString(), name: this.policy.currencyID.name } as NriInputMultiSelectOptions;

this.localHotZoneMBCurrency = [];

this.localHotZoneMDCurrency = [];

this.localColdZoneMBCurrency = [];

this.localColdZoneMDCurrency = [];

this.localHotZoneMBCurrency.push(option);

this.localHotZoneMDCurrency.push(option);

this.localColdZoneMBCurrency.push(option);

this.localColdZoneMDCurrency.push(option);

}

this.policy = PolicyUtils.showvariousMdField(this.policy, event);

}

public setDeductibleValueCurrency(event) {

if (event !== undefined && event.length >= 1) {

const currency = {} as CurrencyDTO;

const roe = {

from: null,

to: null

};

let option = {} as NriInputSelectOptions;

let optionCode = {} as NriInputMultiSelectOptions;

this.getAllCurrencys();

if (this.policy.allCurrenciesIdOptions && this.policy.allCurrenciesStr) {

option = this.policy.allCurrenciesIdOptions[this.policy.allCurrenciesStr.indexOf(event[0])];

}

if (this.policy.allCurrenciesOptions && this.policy.allCurrenciesStr) {

optionCode = this.policy.allCurrenciesOptions[this.policy.allCurrenciesStr.indexOf(event[0])];

}

if (option) {

currency.id = +option.value;

}

if (optionCode) {

currency.code = optionCode.id;

}

if (this.policy.deductibleID[0].currencyID === undefined) {

this.policy.deductibleID[0].currencyID = {} as CurrencyDTO;

}

if (this.policy.currencyID && this.policy.currencyID.code) {

roe.to = this.policy.currencyID.code;

}

roe.from = currency.code;

this.currencyExchangeService.getExchangeRate(roe).then((response) => {

if (!!response) {

this.policy.deductibleID[0].currencyID.id = currency.id;

this.policy.deductibleID[0].currencyID.code = currency.code;

this.policy.deductibleID[0].rate = +response;

this.getMinDeductible(this.policy);

}

});

}

}

public setDeductibleAmount(event) {

this.policy = PolicyUtils.setDeductibleAmount(this.policy, event);

this.getMinDeductible(this.policy);

}

public showsingleMdField(event) {

if (this.policy.currencyID) {

const option = { id: this.policy.currencyID.code.toString(), name: this.policy.currencyID.name } as NriInputMultiSelectOptions;

this.localmdSingleCurrency = [];

this.localmdSingleCurrency.push(option);

}

this.policy = PolicyUtils.showsingleMdField(this.policy, event);

}

public gepMdLimit() {

return PolicyUtils.gepMdLimit(this.policy);

}

public setMdlimit(event) {

this.policy = PolicyUtils.setMdlimit(this.policy, event);

}

public setLifeTimeLimit(event) {

this.policy = PolicyUtils.setLifeTimeLimit(this.policy, event);

}

public manuallySetPolicyROE(rate: string) {

if (!this.autoFilledRate && this.policy && +rate !== this.policy.rOE) {

this.setPolicyROE(+rate);

this.policy.rOEDate = undefined;

}

this.autoFilledRate = false;

}

public getAllActiveCurrencies() {

if (this.allCurrenciesOptions.length === 0) {

this.currencyService.getAllActiveCurrencys()

.then((response) => {

const tempCurrencyOptions = [] as NriInputMultiSelectOptions[];

const parser = new DOMParser();

(response as CurrencyDTO[]).forEach((element) => {

const symbol = parser.parseFromString(element.symbol, 'text/html').body.textContent;

tempCurrencyOptions.push(

{

id: element.code,

name: element.name // + ' (' + symbol + ')'

});

this.allCurrenciesIdOptions.push(

{

value: element.id.toString(),

text: element.name // + ' (' + symbol + ')'

});

this.allCurrenciesStr.push(element.code);

});

this.allCurrenciesOptions = tempCurrencyOptions;

});

}

}

public verifiedVariable(variable: any) {

return variable ? variable : '';

}

public selectPolicyCurrency(event, rOEDateSelector: NriInputDatepickerComponent) {

let oldCurrencyCode = null;

if (event !== undefined) {

if (this.policy.currencyID && this.policy.currencyID.code && event[0] && this.policy.currencyID.code === event[0]) {

return;

}

if (event.length === 0) {

this.policy.currencyID = undefined;

this.policy.hotzoneMB.rate = undefined;

this.policy.hotzoneotherMB.rate = undefined;

this.policy.coldzoneMB.rate = undefined;

this.policy.coldzoneotherMB.rate = undefined;

this.policy.currencyID = undefined;

this.policy.hotzoneMB.currency = undefined;

this.policy.hotzoneotherMB.currency = undefined;

this.policy.coldzoneMB.currency = undefined;

this.policy.coldzoneotherMB.currency = undefined;

this.policy.hotzoneMB.code = undefined;

this.policy.hotzoneotherMB.code = undefined;

this.policy.coldzoneMB.code = undefined;

this.policy.coldzoneotherMB.code = undefined;

this.policy.deductibleID.forEach((deduct: DeductibleDTO) => {

if (deduct.currencyID) {

deduct.currencyID = undefined;

}

});

} else {

this.autoFilledRate = true;

this.clearLocalDeductibleFields();

oldCurrencyCode = this.policy.currencyID && this.policy.currencyID.code ? this.policy.currencyID.code : null;

this.policy = PolicyUtils.selectPolicyCurrency(this.policy, event);

this.setDefaultCurrency();

const roe = {

from: null,

to: 'GBP'

};

roe.from = this.policy.currencyID.code;

if (this.policy.deductibleID) {

this.policy.deductibleID.forEach((deduct: DeductibleDTO) => {

if (deduct.currencyID) {

deduct.currencyID = {

id: this.policy.currencyID.id,

code: this.policy.currencyID.code,

name: this.policy.currencyID.name,

status: this.policy.currencyID.status,

symbol: this.policy.currencyID.symbol

} as CurrencyDTO;

deduct.rate = 1;

const option = [{ id: this.policy.currencyID.code, name: this.policy.currencyID.name }];

deduct.localCurrency = option;

}

});

}

if (this.policy.additionalLimitID) {

this.policy.additionalLimitID.forEach((addLimit) => {

if (addLimit.deductibleID) {

addLimit.deductibleID.currencyID = {

id: this.policy.currencyID.id,

code: this.policy.currencyID.code,

name: this.policy.currencyID.name,

status: this.policy.currencyID.status,

symbol: this.policy.currencyID.symbol

} as CurrencyDTO;

addLimit.deductibleID.rate = 1;

}

addLimit.currencyID = { id: this.policy.currencyID.id } as CurrencyDTO;

addLimit.rate = 1;

addLimit.defaultCurrency = [{ id: this.policy.currencyID.code, name: this.policy.currencyID.name }];

addLimit.localLimitDeductCurrency = [{ id: this.policy.currencyID.code, name: this.policy.currencyID.name }];

addLimit.localAggregateLimitCurrencies = [{ id: this.policy.currencyID.code, name: this.policy.currencyID.name }];

if (addLimit.aggregateLimitCurrencyID) {

addLimit.aggregateLimitCurrencyID = {

id: this.policy.currencyID.id,

code: this.policy.currencyID.code,

name: this.policy.currencyID.name,

status: this.policy.currencyID.status,

symbol: this.policy.currencyID.symbol

} as CurrencyDTO;

}

});

}

if (this.policy.sublimitID) {

this.policy.sublimitID.forEach((sublimit) => {

if (sublimit.deductibleID) {

sublimit.deductibleID.currencyID = {

id: this.policy.currencyID.id,

code: this.policy.currencyID.code,

name: this.policy.currencyID.name,

status: this.policy.currencyID.status,

symbol: this.policy.currencyID.symbol

} as CurrencyDTO;

sublimit.deductibleID.rate = 1;

}

sublimit.currencyID = { id: this.policy.currencyID.id } as CurrencyDTO;

sublimit.defaultCurrency = [{ id: this.policy.currencyID.code, name: this.policy.currencyID.name }];

sublimit.localLimitDeductCurrency = [{ id: this.policy.currencyID.code, name: this.policy.currencyID.name }];

sublimit.localAggregateLimitCurrencies = [{ id: this.policy.currencyID.code, name: this.policy.currencyID.name }];

if (sublimit.aggregateLimitCurrencyID) {

sublimit.aggregateLimitCurrencyID = {

id: this.policy.currencyID.id,

code: this.policy.currencyID.code,

name: this.policy.currencyID.name,

status: this.policy.currencyID.status,

symbol: this.policy.currencyID.symbol

} as CurrencyDTO;

}

});

}

this.policy.hotzoneMB.rate = 1;

this.policy.hotzoneotherMB.rate = 1;

this.policy.coldzoneMB.rate = 1;

this.policy.coldzoneotherMB.rate = 1;

this.policy.hotzoneMB.currency = this.policy.currencyID.id;

this.policy.hotzoneotherMB.currency = this.policy.currencyID.id;

this.policy.coldzoneMB.currency = this.policy.currencyID.id;

this.policy.coldzoneotherMB.currency = this.policy.currencyID.id;

if (this.policy.hotzoneMB.amount || this.policy.hotzoneotherMB.amount || this.policy.coldzoneMB.amount ||

this.policy.coldzoneotherMB.amount) {

this.pushMdDeductibles();

}

const option = { id: this.policy.currencyID.code.toString(), name: this.policy.currencyID.name } as NriInputMultiSelectOptions;

this.localNonConventionCurrency.push(option);

this.localConventionCurrency.push(option);

this.localLiabilityCurrency.push(option);

this.localmdSingleCurrency.push(option);

this.localHotZoneMBCurrency.push(option);

this.localHotZoneMDCurrency.push(option);

this.localColdZoneMBCurrency.push(option);

this.localColdZoneMDCurrency.push(option);

if (roe.from === roe.to) {

this.autoFilledRate = true;

this.setPolicyROE(1);

rOEDateSelector.clearSimple();

this.policy.rOEDate = undefined;

} else {

if (this.policy.startDate && ((!this.policy.rOEDate && !this.policy.rOE) || oldCurrencyCode === 'GBP')) {

this.definePolicyROEDate();

} else {

this.setPolicyROEDate(this.policy.rOEDate);

}

}

}

}

}

public clearLocalDeductibleFields() {

this.localNonConventionCurrency = [];

this.localConventionCurrency = [];

this.localLiabilityCurrency = [];

this.localmdSingleCurrency = [];

this.localHotZoneMBCurrency = [];

this.localHotZoneMDCurrency = [];

this.localColdZoneMBCurrency = [];

this.localColdZoneMDCurrency = [];

}

public pushMdDeductibles() {

this.policy = PolicyUtils.pushMdDeductibles(this.policy);

}

public addSubLimit() {

const subLim = {} as SublimitDTO;

subLim.showBIfields = false;

if (this.policy.currencyID && this.policy.currencyID.id) {

subLim.currencyID = this.policy.currencyID;

}

this.policy.sublimitID.push(subLim);

}

public addAdditionalLimit() {

const subLim = {} as SublimitDTO;

subLim.rate = 1;

subLim.showBIfields = false;

if (this.policy.currencyID && this.policy.currencyID.id) {

subLim.currencyID = this.policy.currencyID;

}

this.policy.additionalLimitID.push(subLim);

}

public setLimitPerOcurrenceFor100(event) {

if (event !== '') {

this.policy.limitPerOcurrenceFor100 = +event;

this.calcLimitPerOcurr(this.policy.limitPerOcurrenceFor100, this.policy.rOE);

this.calcOverallPolicyLimit();

}

}

public selectSublimitCurrency(event, index) {

if (event !== undefined) {

if (event === [] && index !== undefined) {

this.policy.sublimitID[index].currencyID = undefined;

this.policy.sublimitID[index].currencyID.id = undefined;

} else {

const currency = {} as CurrencyDTO;

let option = {} as NriInputSelectOptions;

let optionCode = {} as NriInputMultiSelectOptions;

this.getAllActiveCurrencies();

option = this.allCurrenciesIdOptions[this.allCurrenciesStr.indexOf(event[0])];

optionCode = this.policy.allCurrenciesOptions[this.policy.allCurrenciesStr.indexOf(event[0])];

if (option) {

currency.id = +option.value;

}

this.policy.sublimitID[index].currencyID = {} as CurrencyDTO;

this.policy.sublimitID[index].currencyID.id = currency.id;

if (this.policy.sublimitID[index].defaultCurrency && this.policy.sublimitID[index].defaultCurrency.length > 0 && option) {

this.policy.sublimitID[index].defaultCurrency.push({id: optionCode.id , name: option.text});

this.policy.sublimitID[index].defaultCurrency.splice(0, 1);

}

}

}

}

public setSublimitValue(limit: SublimitDTO, event) {

if (event !== '') {

if (+limit.sublimitValue !== +event) {

limit.sublimitValue = +event;

this.calcDailyRate(limit);

}

} else {

limit.sublimitValue = 0;

limit.bIDailyRate = 0;

}

}

public setSublimitName(event, index) {

this.policy = PolicyUtils.setSublimitName(this.policy, event, index);

}

public calcDailyRate(limit: SublimitDTO) {

if (limit && limit.sublimitValue && limit.indemnityPeriod && limit.showBIfields && !limit.biDailyRateManual) {

if (+limit.sublimitValue === 0) {

limit.bIDailyRate = 0;

} else if (+limit.indemnityPeriod === 0) {

limit.bIDailyRate = 0;

} else {

limit.bIDailyRate = +(+limit.sublimitValue / +limit.indemnityPeriod).toFixed(2);

}

}

}

public removeLimit(index) {

this.policy = PolicyUtils.removeLimit(this.policy, index);

}

public selectAddLimitCurrency(event, index) {

if (event !== undefined) {

if (event === [] && index !== undefined) {

this.policy.additionalLimitID[index].currencyID = undefined;

this.policy.additionalLimitID[index].currencyID.id = undefined;

} else {

const currency = {} as CurrencyDTO;

const roe = { from: null, to: null, date: null };

let option = {} as NriInputSelectOptions;

this.getAllActiveCurrencies();

option = this.allCurrenciesIdOptions[this.allCurrenciesStr.indexOf(event[0])];

let optionCode = {} as NriInputMultiSelectOptions;

optionCode = this.allCurrenciesOptions[this.allCurrenciesStr.indexOf(event[0])];

if (option) {

currency.id = +option.value;

}

if (this.policy.currencyID && this.policy.currencyID.code) {

roe.to = this.policy.currencyID.code;

}

roe.from = optionCode ? optionCode.id : null;

if (this.policy.rOEDate || this.policy.startDate) {

roe.date = moment(new Date((this.policy.rOEDate ? this.policy.rOEDate : this.policy.startDate))).format('YYYY-MM-DD');

}

if (roe.from && roe.to && roe.date) {

roe.from = this.policy.currencyID.code;

roe.to = optionCode ? optionCode.id : null;

this.currencyExchangeService.historicRate(roe).then((response) => {

if (!!response) {

this.policy.additionalLimitID[index].rate = +response;

this.calcOverallPolicyLimit();

}

});

} else if (roe.from && roe.to) {

this.currencyExchangeService.getExchangeRate(roe).then((response) => {

if (!!response) {

this.policy.additionalLimitID[index].rate = +response;

this.calcOverallPolicyLimit();

}

});

}

this.policy.additionalLimitID[index].currencyID = {} as CurrencyDTO;

this.policy.additionalLimitID[index].currencyID.id = currency.id;

}

}

}

public setAddLimitName(event, index) {

this.policy = PolicyUtils.setAddLimitName(this.policy, event, index);

}

public setAddLimitValue(limit: SublimitDTO, event) {

if (event !== '') {

if (+limit.sublimitValue !== +event) {

limit.sublimitValue = +event;

this.calcDailyRate(limit);

this.calcOverallPolicyLimit();

}

} else {

limit.sublimitValue = 0;

limit.bIDailyRate = 0;

this.policy.overallPolicyLimit = (this.policy.limitPerOcurrenceFor100 ? +this.policy.limitPerOcurrenceFor100 : 0);

}

}

public removeAdditionalLimit(index) {

this.policy = PolicyUtils.removeAdditionalLimit(this.policy, index);

this.calcOverallPolicyLimit();

}

public setAnnualAggregateValue(event) {

if (event !== '') {

if (+this.policy.annualAggregateValue !== +event) {

this.policy.annualAggregateValue = +event;

}

}

}

public setAggregateLimitValue(limit, event) {

if (event !== '') {

if (+limit.aggregateLimitValue !== +event) {

limit.aggregateLimitValue = +event;

}

}

}

public setAnnualAggregateLimit(event) {

if (event && event.length > 0) {

this.policy.annualAggregateLimit = true;

this.policy.annualAggregateValue = this.policy.limitPerOcurrenceFor100;

} else {

this.policy.annualAggregateLimit = false;

this.policy.annualAggregateValue = undefined;

}

}

public setExcessLayer(event) {

this.policy = PolicyUtils.setExcessLayer(this.policy, event);

}

public setGroundUpCover(event) {

this.policy = PolicyUtils.setGroundUpCover(this.policy, event);

}

public setDeductibleNotNilForConvention(event) {

this.policy = PolicyUtils.setDeductibleNotNilForConvention(this.policy, event);

}

public setDeductibleNotNilForNonConvention(event) {

this.policy = PolicyUtils.setDeductibleNotNilForNonConvention(this.policy, event);

}

public isSublimitIncludingBI(sublimit: SublimitDTO) {

return sublimit && sublimit.addOnID && sublimit.addOnID.name && sublimit.addOnID.name.toLowerCase().includes('including bi');

}

public selectDeductibleCurrency(event, index) {

if (event !== undefined) {

if (event === [] && index !== undefined) {

this.policy.deductibleID[index].currencyID = undefined;

this.policy.deductibleID[index].currencyID.id = undefined;

} else {

const currency = {} as CurrencyDTO;

const roe = { from: null, to: null };

let option = {} as NriInputSelectOptions;

let optionCode = {} as NriInputMultiSelectOptions;

this.getAllActiveCurrencies();

if (this.policy.allCurrenciesIdOptions && this.policy.allCurrenciesOptions && this.policy.allCurrenciesStr) {

option = this.allCurrenciesIdOptions[this.allCurrenciesStr.indexOf(event[0])];

optionCode = this.allCurrenciesOptions[this.allCurrenciesStr.indexOf(event[0])];

}

if (option) {

currency.id = +option.value;

if (this.policy.deductibleID && this.policy.deductibleID[index] &&

this.policy.deductibleID[index].currencyID && this.policy.deductibleID[index].currencyID.id === currency.id) {

return;

}

if (this.policy.currencyID && this.policy.currencyID.code) {

roe.to = this.policy.currencyID.code;

}

roe.from = optionCode ? optionCode.id : null;

if (roe.from && roe.to) {

this.currencyExchangeService.getExchangeRate(roe).then((response) => {

if (!!response) {

this.policy.deductibleID[index].rate = +response;

this.getMinDeductible(this.policy);

this.calcOverallPolicyLimit();

}

});

}

this.policy.deductibleID[index].currencyID = {} as CurrencyDTO;

this.policy.deductibleID[index].currencyID.id = currency.id;

}

}

}

}

public setAllLocations(event, index) {

this.policy = PolicyUtils.setAllLocations(this.policy, event, index);

}

public setAllCountries(event, index) {

this.policy = PolicyUtils.setAllCountries(this.policy, event, index);

}

public setAllStations(event, index) {

this.policy = PolicyUtils.setAllStations(this.policy, event, index);

}

public setSublimitLocation(event, index) {

this.policy = PolicyUtils.setSublimitLocation(this.policy, event, index);

setTimeout(() => {

this.policy.sublimitID.forEach((sublimit) => {

this.setSublimitLocDefaults(sublimit);

});

}, 100);

}

public setSublimitCountry(event, index) {

this.policy = PolicyUtils.setSublimitCountry(this.policy, event, index);

setTimeout(() => {

this.policy.sublimitID.forEach((sublimit) => {

this.setSublimitCountryDefaults(sublimit);

});

}, 100);

}

public setSublimitStation(selectedCountries, index) {

this.policy = PolicyUtils.setSublimitStation(this.policy, selectedCountries, index);

setTimeout(() => {

this.policy.sublimitID.forEach((sublimit) => {

this.setSublimitStationDefaults(sublimit);

});

}, 100);

}

public setAllLocationsToAddLimits(event, index) {

this.policy = PolicyUtils.setAllLocationsToAddLimits(this.policy, event, index);

}

public setAllCountriesToAddLimits(event, index) {

this.policy = PolicyUtils.setAllCountriesToAddLimits(this.policy, event, index);

}

public setAllStationsToAddLimits(event, index) {

this.policy = PolicyUtils.setAllStationsToAddLimits(this.policy, event, index);

}

public setSublimitLocationToAddLimit(selectedLocations, index) {

this.policy = PolicyUtils.setSublimitLocationToAddLimit(this.policy, selectedLocations, index);

setTimeout(() => {

this.policy.additionalLimitID.forEach((additionalLimit) => {

this.setAddlimitLocationDefault(additionalLimit);

});

}, 100);

}

public setSublimitCountryToAddLimit(selectedCountries, index) {

this.policy = PolicyUtils.setSublimitCountryToAddLimit(this.policy, selectedCountries, index);

setTimeout(() => {

this.policy.additionalLimitID.forEach((additionalLimit) => {

this.setAddlimitCountryDefault(additionalLimit);

});

}, 100);

}

public setSublimitStationToAddLimit(selectedCountries, index) {

this.policy = PolicyUtils.setSublimitStationToAddLimit(this.policy, selectedCountries, index);

setTimeout(() => {

this.policy.additionalLimitID.forEach((additionalLimit) => {

this.setAddlimitStationDefault(additionalLimit);

});

}, 100);

}

public setAggLimit(event, index) {

this.policy = PolicyUtils.setAggLimit(this.policy, event, index);

}

public setAggregateLimitCurrency(event, index) {

if (event && event.length > 0 && event[0] !== undefined) {

this.policy = PolicyUtils.setAggregateLimitCurrency(this.policy, event, index);

}

}

public setAggregateLimitCurrencyToAddLimit(event, index) {

if (event !== undefined && event.length > 0 && event[0] !== undefined) {

this.policy = PolicyUtils.setAggregateLimitCurrencyToAddLimit(this.policy, event, index);

}

}

public setAggLimitToAddLimit(event, index) {

if (event.length !== 0) {

this.policy.additionalLimitID[index].aggregateLimit = true;

if (this.policy.currencyID && this.policy.currencyID.id) {

this.policy.additionalLimitID[index].aggregateLimitCurrencyID = { id: this.policy.currencyID.id } as CurrencyDTO;

}

if (this.policy.limitPerOcurrenceFor100) {

this.policy.additionalLimitID[index].aggregateLimitValue = this.policy.limitPerOcurrenceFor100;

}

} else {

this.policy.additionalLimitID[index].aggregateLimit = false;

this.policy.additionalLimitID[index].aggregateLimitValue = null;

this.policy.additionalLimitID[index].aggregateLimitCurrencyID = null;

this.policy.additionalLimitID[index].localAggregateLimitCurrencies = [];

if (this.policy.currencyID && this.policy.currencyID.code && this.policy.currencyID.name) {

this.policy.additionalLimitID[index].localAggregateLimitCurrencies =

[{ id: this.policy.currencyID.code, name: this.policy.currencyID.name }];

}

}

}

public showSublimitDeduct(index, event) {

this.policy = PolicyUtils.showSublimitDeduct(this.policy, index, event);

}

public showSublimitDeductToAddLimit(index, event) {

this.policy = PolicyUtils.showSublimitDeductToAddLimit(this.policy, index, event);

}

public setLimitDeductibleAmount(event, sublimit) {

if (event !== '') {

if (!sublimit.deductibleID) {

sublimit.deductibleID = {} as DeductibleDTO;

}

if (+sublimit.deductibleID.amount !== +event) {

sublimit.deductibleID.amount = +event;

this.calcWaitingPeriod(sublimit, true);

}

}

}

public setWaitingPeriod(event, index) {

if (this.policy.sublimitID && this.policy.sublimitID[index]) {

if (this.policy.sublimitID[index]) {

if (+event !== this.policy.sublimitID[index].waitingPeriod) {

this.policy.sublimitID[index].waitingPeriod = +event;

if (this.policy.sublimitID[index].bIDailyRate && this.policy.sublimitID[index].deductibleID) {

this.policy.sublimitID[index].deductibleID.amount = +((+event) \* (+this.policy.sublimitID[index].bIDailyRate)).toFixed(0);

}

}

}

}

}

public setWaitingPeriodToAddLimit(event, index) {

if (this.policy.additionalLimitID && this.policy.additionalLimitID[index]) {

if (this.policy.additionalLimitID[index]) {

if (+event !== this.policy.additionalLimitID[index].waitingPeriod) {

this.policy.additionalLimitID[index].waitingPeriod = +event;

if (!this.policy.additionalLimitID[index].sublimitValueChanged &&

this.policy.additionalLimitID[index].bIDailyRate && this.policy.additionalLimitID[index].deductibleID) {

this.policy.additionalLimitID[index].deductibleID.amount = +((+event) \*

(+this.policy.additionalLimitID[index].bIDailyRate)).toFixed(0);

}

}

}

this.policy.additionalLimitID[index].sublimitValueChanged = false;

}

}

public calcWaitingPeriod(sublimit, sublimitValueChanged) {

let result = 0;

if (sublimit && sublimit.deductibleID && sublimit.deductibleID.amount && sublimit.bIDailyRate) {

result = (+sublimit.deductibleID.amount / +sublimit.bIDailyRate);

}

if (+result !== +sublimit.waitingPeriod) {

sublimit.waitingPeriod = +result.toFixed(2);

sublimit.sublimitValueChanged = !!sublimitValueChanged;

}

}

public setLimitDeductibleValueCurrency(event, sublimit) {

if (event) {

if (!sublimit) {

sublimit = {} as SublimitDTO;

}

if (!sublimit.deductibleID) {

sublimit.deductibleID = {} as DeductibleDTO;

}

const currency = {} as CurrencyDTO;

const roe = {

from: null,

to: null,

date: null,

};

let option = {} as NriInputSelectOptions;

let optionCode = {} as NriInputMultiSelectOptions;

this.getAllActiveCurrencies();

if (this.allCurrenciesIdOptions && this.allCurrenciesStr) {

option = this.allCurrenciesIdOptions[this.allCurrenciesStr.indexOf(event[0])];

}

if (this.allCurrenciesOptions && this.allCurrenciesStr) {

optionCode = this.allCurrenciesOptions[this.allCurrenciesStr.indexOf(event[0])];

}

currency.id = option ? +option.value : undefined;

currency.code = optionCode ? optionCode.id : undefined;

if (sublimit.deductibleID && sublimit.deductibleID.currencyID && sublimit.deductibleID.currencyID.id === currency.id) {

return;

}

if (sublimit.deductibleID.currencyID === undefined) {

sublimit.deductibleID.currencyID = {} as CurrencyDTO;

}

if (this.policy.currencyID && this.policy.currencyID.code) {

roe.to = this.policy.currencyID.code;

}

sublimit.deductibleID.currencyID.id = currency.id;

sublimit.deductibleID.currencyID.code = currency.code;

roe.from = currency.code;

if (this.policy.rOEDate || this.policy.startDate) {

roe.date = moment(new Date((this.policy.rOEDate ? this.policy.rOEDate : this.policy.startDate))).format('YYYY-MM-DD');

}

if (roe.from && roe.to && roe.date) {

roe.from = this.policy.currencyID.code;

roe.to = currency.code;

this.currencyExchangeService.historicRate(roe).then((response) => {

if (!!response) {

sublimit.deductibleID.rate = +response;

this.getMinDeductible(this.policy);

}

});

} else if (roe.from && roe.to) {

this.currencyExchangeService.getExchangeRate(roe).then((response) => {

if (!!response) {

sublimit.deductibleID.rate = +response;

this.getMinDeductible(this.policy);

}

});

}

}

}

public setAuthoriseCapacity() {

if (this.policy.authoriseCapacity) {

this.policy.authoriseCapacity = false;

} else {

this.policy.authoriseCapacity = true;

}

}

// #endregion STEP4

// #region STEP5

/\*#Income fields\*/

public getIncomeTypes(incomeType?: string) {

this.incomeSelectOptions = [];

this.deductionNames.forEach((element) => {

if (element.text.match(/^(Survey income|IPE income)$/)) {

if (!this.incomeRows.find((income) => income.typeOfIncome === element.text) || (incomeType && incomeType === element.text)) {

this.incomeSelectOptions.push(element);

}

}

});

}

public setIncomeType(event) {

if (event) {

const options = this.incomeSelectOptions.find((element) => element.value === event);

this.incomeForm.type = options;

}

}

public setIncomeOption(event) {

if (event) {

const options = this.incomeOptions.find((element) => element.value === event);

this.incomeForm.option = options;

}

}

public setIncomePercentage(event) {

this.incomeForm.percentage = event ? event : 0;

let amount = 0;

if (this.incomeForm.type && this.incomeForm.type.text.toUpperCase() === 'IPE INCOME') {

amount = this.policy.totalReinsurerPremiumBeforeDeductionOfIPEAndOtherFeesToNRI ?

this.policy.totalReinsurerPremiumBeforeDeductionOfIPEAndOtherFeesToNRI \* (event / 100) : 0;

this.policy.ipeIncomeAmount = this.removeCommas(this.roundDecimals(amount, 2));

this.policy.ipeIncomePct = +event;

} else {

amount = this.policy.nriNetPremiumBeforeIpe ? this.policy.nriNetPremiumBeforeIpe \* (event / 100) : 0;

}

this.incomeForm.amount = this.roundDecimals(amount, 2);

this.recalculateReinsurerNetPremium();

}

public setIncomeAmount(event) {

this.incomeForm.amount = event ? event : 0;

let percentage = 0;

if (this.incomeForm.type && this.incomeForm.type.text.toUpperCase() === 'IPE INCOME') {

percentage = this.policy.totalReinsurerPremiumBeforeDeductionOfIPEAndOtherFeesToNRI ?

event / this.policy.totalReinsurerPremiumBeforeDeductionOfIPEAndOtherFeesToNRI \* 100 : 0;

this.policy.ipeIncomePct = this.removeCommas(this.roundDecimals(percentage, 2));

this.policy.ipeIncomeAmount = +event;

} else {

percentage = this.policy.nriNetPremiumBeforeIpe ? event / this.policy.nriNetPremiumBeforeIpe \* 100 : 0;

}

this.incomeForm.percentage = this.roundDecimals(percentage, 2);

this.recalculateReinsurerNetPremium();

}

public addIncome() {

const lastItem = this.incomeRows.length - 1;

if (this.isFormEdit && this.isFormEdit.isIncomeEdit) {

const index = this.isFormEdit.incomePosition;

this.incomeRows[index].typeOfIncome = this.incomeForm.type.text;

this.incomeRows[index].income = this.addCommasToNumber(this.incomeForm.percentage);

this.incomeRows[index].incomeAmount = this.addCommasToNumber(+this.incomeForm.amount);

this.incomeRows[index].incomeOption = this.incomeForm.option ? this.incomeForm.option.text : null;

} else {

const incomeToAdd = {

typeOfIncome: this.incomeForm.type.text,

income: this.addCommasToNumber(this.incomeForm.percentage),

incomeAmount: this.addCommasToNumber(+this.incomeForm.amount),

incomeOption: this.incomeForm.option.text ? this.incomeForm.option.text : null

} as IncomeRows;

this.incomeRows.splice(lastItem, 0, incomeToAdd);

}

this.showIncome = false;

this.isFormEdit.isIncomeEdit = false;

this.clearIncomeForm();

this.calculateTotalIncome();

}

public editIncome(index) {

let tempIncome = this.incomeRows[index].typeOfIncome;

this.getIncomeTypes(tempIncome);

this.isFormEdit.isIncomeEdit = true;

this.isFormEdit.incomePosition = index;

this.showIncome = true;

let options = this.incomeSelectOptions.find((element) => element.text === tempIncome);

this.incomeForm.type = options;

tempIncome = this.incomeRows[index].incomeOption;

options = this.incomeOptions.find((element) => element.text === tempIncome);

this.incomeForm.option = options ? options : {} as NriInputSelectOptions;

this.incomeForm.percentage = this.removeCommas(this.incomeRows[index].income);

this.incomeForm.amount = this.removeCommas(this.incomeRows[index].incomeAmount);

}

public removeIncome(index) {

this.incomeRows.splice(index, 1);

this.showIncome = false;

this.calculateTotalIncome();

this.recalculateIncome();

}

public clearIncomeForm() {

this.incomeForm.type = {} as NriInputSelectOptions;

this.incomeForm.option = {} as NriInputSelectOptions;

this.incomeForm.percentage = 0;

this.incomeForm.amount = 0;

}

public calculateTotalIncome() {

let totalIncomePct: number = 0;

let totalIncomeAmount: number = 0;

const lastItem = this.incomeRows.length - 1;

this.incomeRows.forEach((element, ind) => {

if (ind !== lastItem) {

totalIncomePct += Number(element.income);

totalIncomeAmount += this.removeCommas(element.incomeAmount);

}

});

this.incomeRows[lastItem].income = this.addCommasToNumber(totalIncomePct);

this.incomeRows[lastItem].incomeAmount = this.addCommasToNumber(totalIncomeAmount);

this.calcNriNetPremium();

}

public recalculateIncome() {

this.policy.ipeIncomePct = 0;

this.policy.ipeIncomeAmount = 0;

if (this.incomeRows.length > 1) {

this.incomeRows.forEach((element) => {

if (element.incomeOption.toUpperCase() === 'AMOUNT TO BE ENTERED') {

if (element.typeOfIncome && element.typeOfIncome.toUpperCase() === 'IPE INCOME') {

const percentage = this.policy.totalReinsurerPremiumBeforeDeductionOfIPEAndOtherFeesToNRI ?

this.removeCommas(element.incomeAmount) / this.policy.totalReinsurerPremiumBeforeDeductionOfIPEAndOtherFeesToNRI \* 100 : 0;

element.income = this.roundDecimals(percentage, 2);

this.policy.ipeIncomePct = this.removeCommas(element.income);

this.policy.ipeIncomeAmount = this.removeCommas(element.incomeAmount);

} else {

const percentage = this.policy.nriNetPremiumBeforeIpe ? this.removeCommas(element.incomeAmount) / this.policy.nriNetPremiumBeforeIpe \* 100 : 0;

element.income = this.roundDecimals(percentage, 2);

}

} else if (element.incomeOption.toUpperCase() === 'INCOME %') {

if (element.typeOfIncome && element.typeOfIncome.toUpperCase() === 'IPE INCOME') {

const amount = this.policy.totalReinsurerPremiumBeforeDeductionOfIPEAndOtherFeesToNRI ?

this.policy.totalReinsurerPremiumBeforeDeductionOfIPEAndOtherFeesToNRI \* (this.removeCommas(element.income) / 100) : 0;

element.incomeAmount = this.addCommasToNumber(amount);

this.policy.ipeIncomeAmount = this.removeCommas(element.incomeAmount);

this.policy.ipeIncomePct = this.removeCommas(element.income);

} else {

const amount = this.policy.nriNetPremiumBeforeIpe ? this.policy.nriNetPremiumBeforeIpe \* (this.removeCommas(element.income) / 100) : 0;

element.incomeAmount = this.addCommasToNumber(amount);

}

} else {

this.calculateTotalIncome();

}

});

}

this.recalculateReinsurerNetPremium();

}

public recalculateReinsurerNetPremium() {

if (this.policy.policyShareID.length > 0) {

let totalDeductionPct = 0;

if (this.deductionsAndFees.length > 0) {

this.deductionsAndFees.forEach((deduction) => {

totalDeductionPct = +deduction.deduction;

});

}

this.policy.policyShareID.forEach((element, ind) => {

const reinsurerPremiumAfterDeductions = this.policy.nriGrossPremiumShare ?

(((this.policy.nriGrossPremiumShare

\* (100 - totalDeductionPct) / 100) \* +element.reinsurerPct / 100) +

((this.policy.nriGrossPremiumShare

\* (100 - totalDeductionPct) / 100) \* +element.reinsurerPct / 100) \*

((this.policy.ipeIncomePct ? this.policy.ipeIncomePct : 0) \* -1) / 100) : 0;

const ipeAndOtherFeesToNri = this.policy.ipeIncomePct ? this.policy.policyShareID[ind].premiumReinsurerBeforeIpe \* this.policy.ipeIncomePct / 100 : 0;

this.policy.policyShareID[ind].premiumReinsurerAfterIpe = reinsurerPremiumAfterDeductions;

this.policy.policyShareID[ind].iPEAndOtherFeesToNRI = ipeAndOtherFeesToNri;

});

this.updateReinsuranceTableRows();

}

}

public onIncomeAction(event) {

const buttonClicked = event[0];

const ind = event[1].$$index;

switch (buttonClicked) {

case 'Edit':

this.editIncome(ind);

break;

case 'Remove':

this.deleteGridRow = {name: 'income', index: ind};

this.openPremiumModal();

break;

default:

break;

}

}

public showIncomeForm() {

this.showIncome = true;

this.getIncomeTypes();

}

public onIncomeFormHide() {

this.showIncome = false;

this.clearIncomeForm();

this.isFormEdit.isIncomeEdit = false;

}

/\*#Deductions fields\*/

public getDeductionTypes(deductionType?: string) {

this.deductionSelectOptions = [];

this.deductionNames.forEach((element) => {

if (element.text.match(/^(Upfront NCB|Deferred NCB|Brokerage fees|IPE expense|Other deductions)$/)) {

if (!this.deductionsAndFees.find((deduction) => deduction.typeOfDeductions === element.text) || element.text === 'Other deductions'

|| (deductionType && deductionType === element.text)) {

this.deductionSelectOptions.push(element);

}

}

});

}

public setDeductionAndFeesType(event) {

event === '4' ? this.showOtherDeductions = true : this.showOtherDeductions = false;

if (event) {

const options = this.deductionSelectOptions.find((element) => element.value === event);

this.deductionForm.type = options;

}

}

public setDeductionAndFeesOption(event) {

if (event) {

const options = this.deductionOptions.find((element) => element.value === event);

this.deductionForm.option = options;

}

}

public setDeductionAndFeesPercentage(event) {

this.deductionForm.percentage = event ? event : 0;

const amount = this.policy.nriGrossPremiumShare ? this.policy.nriGrossPremiumShare \* (event / 100) \* -1 : 0;

this.deductionForm.amount = this.roundDecimals(amount, 2);

}

public setDeductionAndFeesAmount(event) {

if (this.policy.nriGrossPremiumShare && this.policy.nriGrossPremiumShare > 0) {

this.deductionForm.amount = event ? (Math.abs(event) \* -1) : 0;

} else {

this.deductionForm.amount = event ? Math.abs(event) : 0;

}

const percentage = this.policy.nriGrossPremiumShare ? Math.abs(event / this.policy.nriGrossPremiumShare) \* 100 : 0;

this.deductionForm.percentage = this.roundDecimals(percentage, 2);

}

public setDeductionCommissionOrFee(event) {

if (event) {

const options = this.commissionOrFee.find((element) => element.value === event);

this.deductionForm.commissionOrFee = options;

}

}

public setDeductionComment(event) {

this.deductionForm.comment = event ? event : '';

}

public onDeductionsAction(event) {

const buttonClicked = event[0];

const ind = event[1].$$index;

switch (buttonClicked) {

case 'Edit':

this.editDeductionsAndFees(ind);

break;

case 'Remove':

this.deleteGridRow = {name: 'deductions', index: ind};

this.openPremiumModal();

break;

default:

break;

}

}

public addDeductionsAndFees() {

const lastItem = this.deductionsAndFees.length - 1;

if (this.isFormEdit && this.isFormEdit.isDeductionsEdit) {

const index = this.isFormEdit.deductionsPosition;

this.deductionsAndFees[index].typeOfDeductions = this.deductionForm.type.text;

this.deductionsAndFees[index].deduction = this.addCommasToNumber(this.deductionForm.percentage);

this.deductionsAndFees[index].deductionAmount = this.addCommasToNumber(this.deductionForm.amount);

this.deductionsAndFees[index].deductionOption = this.deductionForm.option ? this.deductionForm.option.text : null;

this.deductionsAndFees[index].comment = this.deductionForm.comment;

this.deductionsAndFees[index].commissionOrFee = this.deductionForm.commissionOrFee ? this.deductionForm.commissionOrFee.text : null;

} else {

const deductionToAdd = {

typeOfDeductions: this.deductionForm.type.text,

deduction: this.addCommasToNumber(this.deductionForm.percentage),

deductionAmount: this.addCommasToNumber(this.deductionForm.amount),

deductionOption: this.deductionForm.option.text ? this.deductionForm.option.text : null,

comment: this.deductionForm.comment ? this.deductionForm.comment : '',

commissionOrFee: this.deductionForm.commissionOrFee.text ? this.deductionForm.commissionOrFee.text : null

} as DeductionsAndFees;

this.deductionsAndFees.splice(lastItem, 0, deductionToAdd);

}

this.showDeductions = false;

this.isFormEdit.isDeductionsEdit = false;

this.clearDeductionsForm();

this.calculateTotalDeductions();

this.recalculateReinsurer();

}

public editDeductionsAndFees(index) {

let tempDeduction = this.deductionsAndFees[index].typeOfDeductions;

this.getDeductionTypes(tempDeduction);

this.isFormEdit.isDeductionsEdit = true;

this.isFormEdit.deductionsPosition = index;

this.showDeductions = true;

let options = this.deductionSelectOptions.find((element) => element.text === tempDeduction);

this.deductionForm.type = options;

tempDeduction = this.deductionsAndFees[index].deductionOption;

options = this.deductionOptions.find((element) => element.text === tempDeduction);

this.deductionForm.option = options ? options : {} as NriInputSelectOptions;

this.deductionForm.percentage = this.removeCommas(this.deductionsAndFees[index].deduction);

this.deductionForm.amount = this.removeCommas(this.deductionsAndFees[index].deductionAmount);

this.deductionForm.comment = this.deductionsAndFees[index].comment;

tempDeduction = this.deductionsAndFees[index].commissionOrFee;

options = this.commissionOrFee.find((element) => element.text === tempDeduction);

this.deductionForm.commissionOrFee = options ? options : {} as NriInputSelectOptions;

}

public removeDeductionsAndFees(index) {

this.deductionsAndFees.splice(index, 1);

this.showDeductions = false;

this.calculateTotalDeductions();

}

public clearDeductionsForm() {

this.deductionForm.type = {} as NriInputSelectOptions;

this.deductionForm.option = {} as NriInputSelectOptions;

this.deductionForm.percentage = 0;

this.deductionForm.amount = 0;

this.deductionForm.comment = '';

this.deductionForm.commissionOrFee = {} as NriInputSelectOptions;

}

public calculateTotalDeductions() {

let totalDeductionsPct: number = 0;

let totalDeductionsAmount: number = 0;

const lastItem = this.deductionsAndFees.length - 1;

this.deductionsAndFees.forEach((element, ind) => {

if (ind !== lastItem) {

totalDeductionsPct += Number(element.deduction);

totalDeductionsAmount += this.removeCommas(element.deductionAmount);

}

});

this.deductionsAndFees[lastItem].deduction = this.addCommasToNumber(totalDeductionsPct);

this.deductionsAndFees[lastItem].deductionAmount = this.addCommasToNumber(totalDeductionsAmount);

this.calcIPEAndOtherFeesToNRI();

this.calcNRIPremiumBeforeIPEAndOtherFees();

}

public recalculateDeductions() {

this.deductionsAndFees.forEach((element) => {

if (this.policy.nriGrossPremiumShare && this.policy.nriGrossPremiumShare > 0) {

element.deductionAmount = this.addCommasToNumber(Math.abs(this.removeCommas(element.deductionAmount)) \* -1);

} else {

element.deductionAmount = this.addCommasToNumber(Math.abs(this.removeCommas(element.deductionAmount)));

}

if (element.deductionOption.toUpperCase() === 'AMOUNT TO BE ENTERED') {

const amount = this.removeCommas(element.deductionAmount);

const percentage = this.policy.nriGrossPremiumShare ? Math.abs(amount / this.policy.nriGrossPremiumShare) \* 100 : 0;

element.deduction = this.roundDecimals(percentage, 2);

} else if (element.deductionOption.toUpperCase() === 'DEDUCTION %') {

const amount = this.policy.nriGrossPremiumShare ? this.policy.nriGrossPremiumShare \* (this.removeCommas(element.deduction) / 100) \* -1 : 0;

element.deductionAmount = this.addCommasToNumber(amount);

} else {

this.calculateTotalDeductions();

}

});

}

public showDeductionsForm() {

this.showDeductions = true;

this.getDeductionTypes();

}

public onDeductionsFormHide() {

this.showDeductions = false;

this.clearDeductionsForm();

this.isFormEdit.isDeductionsEdit = false;

}

/\*#Reinsurer fields\*/

public calculateTotalReinsurer() {

let totalReinsurerPct: number = 0;

let totalReinsurerAmount: number = 0;

let totalIpeAndOtherFees: number = 0;

let totalReinsurerNetPremium: number = 0;

const lastItem = this.reinsuranceTableRows.length - 1;

this.reinsuranceTableRows.forEach((element, ind) => {

if (ind !== lastItem) {

totalReinsurerPct += Number(element.reinsurer);

totalReinsurerAmount += this.removeCommas(element.reinsurerPremiumBeforeDeductionOfIpeAndOtherFeesToNri);

totalIpeAndOtherFees += this.removeCommas(element.ipeAndOtherFeesToNri);

totalReinsurerNetPremium += this.removeCommas(element.reinsurerNetPremium);

}

});

this.reinsuranceTableRows[lastItem].reinsurer = this.addCommasToNumber(totalReinsurerPct);

this.reinsuranceTableRows[lastItem].reinsurerPremiumBeforeDeductionOfIpeAndOtherFeesToNri = this.addCommasToNumber(totalReinsurerAmount);

this.reinsuranceTableRows[lastItem].ipeAndOtherFeesToNri = this.addCommasToNumber(totalIpeAndOtherFees);

this.reinsuranceTableRows[lastItem].reinsurerNetPremium = this.addCommasToNumber(totalReinsurerNetPremium);

this.policy.totalReinsurerPremiumBeforeDeductionOfIPEAndOtherFeesToNRI = +totalReinsurerAmount;

this.policy.totalReinsurancePct = +(totalReinsurerPct.toFixed(2));

this.calcNRIPremiumBeforeIPEAndOtherFees();

}

public recalculateReinsurer() {

this.policy.policyShareID.forEach((element, index) => {

if (element.insurerRole === 'reout') {

this.setShare1(element.reinsurerPct, index);

}

});

}

/\*#Taxes fields\*/

public setTaxesCountry(event) {

this.taxFormFields.country = event ? event : '';

}

public setTaxesType(event) {

if (event) {

const options = this.taxesSelectOptions.find((element) => element.value === event);

this.taxFormFields.type = options;

}

this.taxFormFields.percentage = 0;

this.taxFormFields.amount = 0;

}

public setTaxesPercentage($event) {

let amount: number = 0;

this.taxFormFields.percentage = $event ? $event : 0;

if (this.taxFormFields.type.text === '% of NRI Net Premium') {

amount = this.policy.nRInetPremium ? this.policy.nRInetPremium \* (this.taxFormFields.percentage / 100) : 0;

} else {

amount = this.policy.nriGrossPremiumShare ? this.policy.nriGrossPremiumShare \* (this.taxFormFields.percentage / 100) : 0;

}

this.taxFormFields.amount = this.roundDecimals(amount, 2);

}

public setTaxesAmount($event) {

this.taxFormFields.amount = $event ? $event : 0;

const percentage = this.policy.nriGrossPremiumShare ? $event / this.policy.nriGrossPremiumShare \* 100 : 0;

this.taxFormFields.percentage = this.roundDecimals(percentage, 2);

}

public onTaxesAction(event) {

const buttonClicked = event[0];

const ind = event[1].$$index;

switch (buttonClicked) {

case 'Edit':

this.onEditTaxes(ind);

break;

case 'Remove':

this.deleteGridRow = {name: 'taxes', index: ind};

this.openPremiumModal();

break;

default:

break;

}

}

public addTax() {

const lastItem = this.taxes.length - 1;

if (this.isFormEdit && this.isFormEdit.isTaxesEdit) {

const index = this.isFormEdit.taxesPosition;

this.taxes[index].country = this.taxFormFields.country;

this.taxes[index].type = this.taxFormFields.type.text;

this.taxes[index].tax = this.addCommasToNumber(this.taxFormFields.percentage);

this.taxes[index].taxAmount = this.addCommasToNumber(+this.taxFormFields.amount);

} else {

const taxToAdd = {

country: this.taxFormFields.country, type: this.taxFormFields.type.text, tax: this.addCommasToNumber(this.taxFormFields.percentage),

taxAmount: this.addCommasToNumber(+this.taxFormFields.amount)

} as Taxes;

this.taxes.splice(lastItem, 0, taxToAdd);

}

this.isTaxFormShow = false;

this.isFormEdit.isTaxesEdit = false;

this.clearTaxesForm();

this.calculateTotalTaxes();

}

public onEditTaxes(index) {

const tempText = this.taxes[index].type;

this.isFormEdit.isTaxesEdit = true;

this.isFormEdit.taxesPosition = index;

this.isTaxFormShow = true;

this.taxFormFields.country = this.taxes[index].country;

const options = this.taxesSelectOptions.find((element) => element.text === tempText);

this.taxFormFields.type = options;

this.taxFormFields.percentage = this.removeCommas(this.taxes[index].tax);

this.taxFormFields.amount = this.removeCommas(this.taxes[index].taxAmount);

}

public removeTaxes(index) {

this.taxes.splice(index, 1);

this.isTaxFormShow = false;

this.calculateTotalTaxes();

}

public clearTaxesForm() {

this.taxFormFields.country = '';

this.taxFormFields.type = {} as NriInputSelectOptions;

this.taxFormFields.percentage = 0;

this.taxFormFields.amount = 0;

}

public calculateTotalTaxes() {

let totalPct: number = 0;

let totalAmount: number = 0;

const lastItem = this.taxes.length - 1;

this.taxes.forEach((element, ind) => {

if (ind !== lastItem) {

totalPct += Number(element.tax);

totalAmount += this.removeCommas(element.taxAmount);

}

});

this.taxes[lastItem].tax = this.addCommasToNumber(totalPct);

this.taxes[lastItem].taxAmount = this.addCommasToNumber(totalAmount);

this.calcNRIPremiumDebt();

}

public recalculateTaxes() {

this.taxes.forEach((element, index) => {

if (element.type.toUpperCase() === 'AMOUNT TO BE ENTERED') {

const amount = this.removeCommas(element.taxAmount);

const percentage = this.policy.nriGrossPremiumShare ? Math.abs(amount / this.policy.nriGrossPremiumShare) \* 100 : 0;

element.tax = this.roundDecimals(percentage, 2);

} else if (element.type === '% of NRI Net Premium') {

const amount = this.policy.nRInetPremium ? this.policy.nRInetPremium \* (this.removeCommas(element.tax) / 100) : 0;

element.taxAmount = this.addCommasToNumber(amount);

} else if (element.type === '% of NRI Gross Premium') {

const amount = this.policy.nriGrossPremiumShare ? this.policy.nriGrossPremiumShare \* (this.removeCommas(element.tax) / 100) : 0;

element.taxAmount = this.addCommasToNumber(amount);

}

});

this.calculateTotalTaxes();

}

public showTaxForm() {

this.isTaxFormShow = true;

}

public onTaxesFormHide() {

this.isTaxFormShow = false;

this.clearTaxesForm();

this.isFormEdit.isTaxesEdit = false;

}

public onDeleteRow() {

switch (this.deleteGridRow.name) {

case 'income':

this.removeIncome(this.deleteGridRow.index);

break;

case 'deductions':

this.removeDeductionsAndFees(this.deleteGridRow.index);

break;

case 'taxes':

this.removeTaxes(this.deleteGridRow.index);

break;

}

}

public addDeductionComment(comment: string, index: number) {

if (this.policy && this.policy.deductionID && this.policy.deductionID[index]) {

this.policy.deductionID[index].deductionComment = comment;

}

}

public addCommisionOrFee(value: string, index: number) {

if (this.policy && this.policy.deductionID && this.policy.deductionID[index]) {

this.policy.deductionID[index].commissionOrFee = value;

}

}

public setActualPrice(actualPriceInputText: NriInputTextComponent): number {

if (actualPriceInputText) {

this.policy.actualPrice = PolicyUtils.calcActualPrice(this.policy);

}

return this.policy.actualPrice;

}

public setActualPriceNoValidation(): number {

this.policy.actualPrice = PolicyUtils.calcActualPrice(this.policy);

return this.policy.actualPrice;

}

public setTableNameToProperty(name: string[]) {

let stringToReturn: string = '';

name.forEach((element) => {

element = element.toLowerCase();

element = element.charAt(0).toUpperCase() + element.slice(1);

stringToReturn = stringToReturn.concat(element);

});

return stringToReturn;

}

public addCommasToNumber(numberValue: number) {

const options = { minimumFractionDigits: 2, maximumFractionDigits: 2 };

const numberWithCommas = numberValue.toLocaleString('en', options) !== '0' ? numberValue.toLocaleString('en', options) : '0.00';

return numberWithCommas;

}

public removeCommas(numberValue: string) {

numberValue = numberValue ? numberValue.toString().replace(/,/g , '') : numberValue;

return +numberValue;

}

public formatNumber(valueToStore, fieldToFormat) {

const valueFormatted = parseFloat(valueToStore).toFixed(2);

switch (fieldToFormat) {

case 'Gross Premium':

this.policy.grossPremium100Pct = Number(valueFormatted);

break;

case 'Deduction Amount':

this.deductionForm.amount = Number(valueFormatted);

break;

case 'Deduction Pct':

this.deductionForm.percentage = Number(valueFormatted);

break;

case 'Tax Pct':

this.taxFormFields.percentage = Number(valueFormatted);

break;

case 'Tax Amount':

this.taxFormFields.amount = Number(valueFormatted);

break;

default:

break;

}

}

public commissionOrFeePopulate() {

this.commissionOrFee.push(

{

value: 'Commission',

text: 'Commission'

},

{

value: 'Fee',

text: 'Fee'

}

);

}

public calcNriNetPremium() {

const lastItem = this.incomeRows.length - 1;

const amount = this.removeCommas(this.incomeRows[lastItem].incomeAmount);

this.policy.nRInetPremium = this.policy.nriNetPremiumBeforeIpe + amount;

if (this.taxes.length > 1) {

this.recalculateTaxes();

} else {

this.calcNRIPremiumDebt();

}

}

public calcNriNetSharePct() {

const lastItem = this.reinsuranceTableRows.length - 1;

this.policy.nriNetSharePct = 100 - (this.reinsuranceTableRows ? this.removeCommas(this.reinsuranceTableRows[lastItem].reinsurer) : 0);

}

public roundDecimals(valueToRound, decimalPlaces) {

let isNegative = false;

if (decimalPlaces === undefined) {

decimalPlaces = 0;

}

if (valueToRound < 0) {

valueToRound = valueToRound \* -1;

isNegative = true;

}

const multiplicator = Math.pow(10, decimalPlaces);

valueToRound = parseFloat((valueToRound \* multiplicator).toFixed(11));

valueToRound = (Math.round(valueToRound) / multiplicator).toFixed(2);

if (isNegative) {

valueToRound = (valueToRound \* -1).toFixed(2);

}

return valueToRound;

}

public setPremiumBeforeDeductions(event) {

if (+event !== this.policy.grossPremium100Pct) {

this.policy = PolicyUtils.setPremiumBeforeDeductions(this.policy, event);

this.setNRIGrossPremium();

}

}

public setNRIGrossPremium() {

if (this.policy && this.policy.grossPremium100Pct) {

this.policy.nriGrossPremiumShare = this.policy.grossPremium100Pct \* (this.nriGrossShare / 100);

if (this.deductionsAndFees.length > 1) {

this.recalculateDeductions();

} else {

this.calcNRIPremiumBeforeIPEAndOtherFees();

this.calcIPEAndOtherFeesToNRI();

}

this.calcNRIPremiumDebt();

this.calcNriNetSharePct();

}

}

public calcNRIGSPremiumAfterDeductionOfNCBAndFees() {

if (this.negativeBrokerAmount && this.policy.brokerageAmount > 0) {

this.policy.brokerageAmount = -(this.policy.brokerageAmount);

}

const result = PolicyUtils.calcNRIGSPremiumAfterDeductionOfNCBAndFees(this.policy);

return isNaN(result) ? 0 : result;

}

public setNriPremiumBeforeIpeAndOtherFees() {

this.policy.nriPremiumBeforeIpeAndOtherFees = PolicyUtils.calcNRIPremiumBeforeIPEAndFees(this.policy);

}

public calcIPEOutAmount() {

this.policy.ipeExpenseAmount = PolicyUtils.calcIPEOutAmount(this.policy);

return this.policy.ipeExpenseAmount.toFixed(2);

}

public calcTotalFeesForPrimaryCeedingInsurer() {

if (this.policy === undefined) {

return 0;

}

let result = 0;

if (this.policy.deductionID !== undefined && this.policy.deductionID.length > 0) {

this.policy.deductionID.forEach((deduction) => {

if (deduction.deductionCedingTypeID && deduction.deductionCedingTypeID.deductionCedingType.toUpperCase() ===

'PRIMARY CEDING INSURER') {

result = result + (+deduction.amount);

}

});

}

result = result + (isNaN(this.policy.ipeExpenseAmount) ? 0 : (+this.policy.ipeExpenseAmount));

this.policy.totalFeesForPrimaryCeedingInsurer = isNaN(result) ? 0 : result;

this.calcIPEAndOtherFeesLessIncomeToNRI();

return this.policy.totalFeesForPrimaryCeedingInsurer;

}

public calcCedingInsurerGSPremiumAfterAllNCBAndFees(): number {

return Number(PolicyUtils.calcCedingInsurerGSPremiumAfterAllNCBAndFees(this.policy).toFixed(2));

}

public calcIPEAndOtherFeesLessIncomeToNRI(): number {

this.policy.IPEAndOtherExpensesToNRI = (this.policy.totalFeesForPrimaryCeedingInsurer

\* (this.policy.totalOverallPoolShare !== undefined ? ((+this.policy.totalOverallPoolShare) / 100) : 0));

return this.policy.IPEAndOtherExpensesToNRI;

}

public calcNRIPremiumBeforeIPEAndOtherFees() {

let lastItem = this.deductionsAndFees.length - 1;

const totalDeductions = this.removeCommas(this.deductionsAndFees[lastItem].deductionAmount);

lastItem = this.reinsuranceTableRows.length - 1;

const totalReinsurerBeforeIpe = this.removeCommas(this.reinsuranceTableRows[lastItem].reinsurerPremiumBeforeDeductionOfIpeAndOtherFeesToNri);

this.policy.nriNetPremiumBeforeIpe = this.policy.nriGrossPremiumShare + totalDeductions - totalReinsurerBeforeIpe;

this.calcNriNetPremium();

}

public calcCedingInsurerIPEAndAllDeductions(): number {

this.policy.iPEAndOtherFeesToNRI = PolicyUtils.calcCedingInsurerIPEAndAllDeductions(this.policy);

return this.policy.iPEAndOtherFeesToNRI;

}

public calcNRIPremiumDebt() {

const taxAmount = this.removeCommas(this.taxes[this.taxes.length - 1].taxAmount);

const ipeAndOtherFees = this.policy.iPEAndOtherFeesToNRI ? this.policy.iPEAndOtherFeesToNRI : 0;

let deferredNCB = 0;

this.deductionsAndFees.forEach((deduction) => {

if (deduction.typeOfDeductions.toUpperCase() === 'DEFERRED NCB') {

deferredNCB = this.removeCommas(deduction.deductionAmount);

}

});

const nriGPremiumShare = this.policy.nriGrossPremiumShare ? this.policy.nriGrossPremiumShare : 0;

this.policy.nRIPremiumDebt = nriGPremiumShare + ipeAndOtherFees + taxAmount - deferredNCB;

}

public calcIPEOutAmountForFees() {

return PolicyUtils.calcIPEOutAmountForFees(this.policy);

}

public setInstalmentArray(numberOfInstalments) {

if (+numberOfInstalments === 0) {

this.policy.instalmentDate = undefined;

this.policy.numberOfInstalments = +numberOfInstalments;

this.instalmentDate = undefined;

} else {

this.policy.numberOfInstalments = +numberOfInstalments;

if (!this.instalmentDate) {

this.instalmentDate = [];

}

if (this.policy.numberOfInstalments && this.policy.instalmentDate) {

if (this.policy.numberOfInstalments > this.policy.instalmentDate.length) {

const addDates = this.policy.numberOfInstalments - this.policy.instalmentDate.length;

let i = 0;

do {

this.policy.instalmentDate.push(null);

this.instalmentDate.push(null);

i++;

} while (i < addDates);

}

if (this.policy.numberOfInstalments < this.policy.instalmentDate.length) {

const removeDates = this.policy.instalmentDate.length - this.policy.numberOfInstalments;

let i = 0;

do {

this.policy.instalmentDate.pop();

this.instalmentDate.pop();

i++;

} while (i < removeDates);

}

} else {

this.instalmentDate = [];

this.instalmentDate = Array(this.policy.numberOfInstalments).fill(null);

this.policy.instalmentDate = [];

this.policy.instalmentDate = Array(this.policy.numberOfInstalments).fill(null);

}

}

}

public calcPremiumAfterDeductionOfNCBAndFees() {

this.policy.grossSharePremiumAfterDeductionOfNCBAndFees = PolicyUtils.calcNRIGSPremiumAfterDeductionOfNCBAndFees(this.policy);

}

public calcIpeInAmount() {

this.policy.ipeIncomeAmount = PolicyUtils.calcIPEAmount(this.policy);

return this.policy.ipeIncomeAmount.toFixed(2);

}

public getSingleCheckValue(value, i?) {

return this.manageCommunService.getSingleCheckValue(value);

}

public checkerValidation(value) {

return this.checker(value);

}

public checker(value) {

if (value !== null) {

return [0];

}

return [];

}

public completedValidation(task: UnderwritingTaskDTO, index) {

if (this.historyTasks.length > 0 && this.historyTasks[index].completedDate) {

return [0];

}

if (this.policy.underwritingProcess && ((this.policy.underwritingProcess.underwriter && this.policy.underwritingProcess.underwriter.id === this.userDetails.id)

|| (this.policy.underwritingProcess.secondUnderwriter && this.policy.underwritingProcess.secondUnderwriter.id === this.userDetails.id))) {

return [];

}

return [0];

}

public reviewValidation(task: UnderwritingTaskDTO, index) {

if (this.historyTasks.length > 0 && this.historyTasks[index].reviewedDate) {

return [0];

}

if (!task.completedDate) {

return [0];

}

if (task.completedBy && task.completedBy.id === this.userDetails.id) {

return [0];

}

if (this.policy.underwritingProcess && (!this.policy.underwritingProcess.underwriter &&

(this.policy.underwritingProcess.secondUnderwriter && this.policy.underwritingProcess.secondUnderwriter.id !== this.userDetails.id)) ||

((!this.policy.underwritingProcess.secondUnderwriter) && (this.policy.underwritingProcess.underwriter &&

this.policy.underwritingProcess.underwriter.id !== this.userDetails.id ))) {

return [0];

}

if (this.policy.underwritingProcess && (this.policy.underwritingProcess.underwriter &&

this.policy.underwritingProcess.underwriter.id !== this.userDetails.id) && (this.policy.underwritingProcess.secondUnderwriter &&

this.policy.underwritingProcess.secondUnderwriter.id !== this.userDetails.id)) {

return [0];

}

return [];

}

public calcIPEAndOtherFeesToNRI(): void {

const lastItem = this.deductionsAndFees.length - 1;

let totalIpe = 0;

if (this.deductionsAndFees.length > 1) {

this.deductionsAndFees.forEach((deduction, ind) => {

if (ind !== lastItem) {

totalIpe += Math.abs(this.removeCommas(deduction.deductionAmount));

}

});

}

if (this.policy.nriGrossPremiumShare && this.policy.nriGrossPremiumShare > 0) {

this.policy.iPEAndOtherFeesToNRI = totalIpe \* -1;

} else {

this.policy.iPEAndOtherFeesToNRI = Math.abs(totalIpe);

}

if (this.reinsuranceTableRows.length > 1) {

this.recalculateReinsurer();

}

this.calcNRIPremiumDebt();

}

public setSpecifyForm(policy: PolicyDTO, event) {

return PolicyUtils.setSpecifyForm(this.policy, event);

}

public isOtherDeductionPercentageEnable(i) {

if (this.policy && this.policy.deductionID && this.policy.deductionID[i] && this.policyOtherDeductionType[i] !== undefined) {

if (this.policy.deductionID[i].deductionTypeID && this.policy.deductionID[i].deductionTypeID.name

&& this.policy.deductionID[i].deductionTypeID.name.toUpperCase() === this.NRI\_AFTER\_DEDUCTION\_MESSAGE.toUpperCase()) {

return true;

}

}

return false;

}

public calcDeductPercentage(\_obj: DeductionDTO) {

\_obj.percentage = 0;

if (\_obj.amount === undefined) {

\_obj.amount = 0;

}

const premiumAfterNcbAndFees = this.calcCedingInsurerGSPremiumAfterAllNCBAndFees();

if (premiumAfterNcbAndFees !== 0 && \_obj.amount !== 0) {

const result = ((\_obj.amount) / (+premiumAfterNcbAndFees)) \* 100;

\_obj.percentage = result ? Number((Math.abs(result)).toFixed(2)) : 0;

} else {

\_obj.percentage = 0;

}

return \_obj.percentage;

}

public updateFinanceDeductionAmountGBP(deduction: DeductionDTO, policy: PolicyDTO, deductionIndex: number) {

if (this.isOtherDeductionPercentageEnable(deductionIndex)) {

if (policy.rOE && policy.rOE !== 0) {

deduction.financeDeductionAmountGBP = -Math.abs(

(((deduction.percentage ? deduction.percentage : 0) \*

(policy.grossPremium100Pct ? policy.grossPremium100Pct : 0)) / 100)

/ policy.rOE);

} else {

deduction.financeDeductionAmountGBP = 0;

}

}

}

public getNRICedingType(surveyFeeTypeDrop: NriInputSelectComponent,

cedingInsurerSurveyFeePercentageInput: NriInputSelectComponent,

cedingInsurerSurveyFeeAmountInput: NriInputSelectComponent) {

if (this.cedingInsurerSurveyFeeSelectOptions.length === 0) {

this.policyService.getNRICedingType()

.then((response) => {

(response as AbstractNameAndIdDTO[]).forEach((element) => {

this.cedingInsurerSurveyFeeSelectOptions.push(

{

value: element.id.toString(),

text: element.name

} as NriInputSelectOptions);

});

});

}

if (!!this.cedingInsurerSurveyFee) {

if (!!surveyFeeTypeDrop && !!this.cedingInsurerSurveyFee.deductionTypeID && !!this.cedingInsurerSurveyFee.deductionTypeID.name) {

surveyFeeTypeDrop.btnText = this.cedingInsurerSurveyFee.deductionTypeID.name;

}

if (!!cedingInsurerSurveyFeePercentageInput && this.cedingInsurerSurveyFee.percentage) {

cedingInsurerSurveyFeePercentageInput.value = this.cedingInsurerSurveyFee.percentage.toString();

}

if (!!cedingInsurerSurveyFeeAmountInput && this.cedingInsurerSurveyFee.amount) {

cedingInsurerSurveyFeeAmountInput.value = this.cedingInsurerSurveyFee.amount.toString();

}

}

}

public isOtherDeductionNriCedingInsurer(deduction: DeductionDTO) {

return PolicyUtils.isOtherDeductionNriCedingInsurer(deduction);

}

public setIpeInPercentageForFees(event) {

if (!isNaN(event)) {

this.policy.ipeIncomePct = +event;

}

}

public onSelectOtherIncomeEnable(event: any) {

this.isOtherIncomeEnable = !this.isOtherIncomeEnable;

if (this.isOtherIncomeEnable) {

this.policy.policyIncomeID = [];

} else {

this.policy.policyIncomeID = undefined;

}

}

public removeOtherIncome(i: number) {

this.policy.policyIncomeID.splice(i, 1);

}

public addOtherIncome() {

this.policy.policyIncomeID.push({} as PolicyIncomeDTO);

}

public setInstalmentDate(event: Date, index) {

let validStartDate: boolean = true;

if (!this.message) {

this.message = [];

}

if (event) {

if (this.policy.startDate) {

validStartDate = (this.dateUtils.convertToStartofDay(event) >= this.startDate.getTime());

}

if (validStartDate) {

this.policy.instalmentDate[index] = event.getTime();

}

const endDate: Date = new Date(this.policy.endDate);

this.instalmentDateVerification(endDate, event, index);

} else {

this.policy.instalmentDate[index] = null;

this.message[index] = null;

}

}

public convertToDate(date: any) {

return date ? new Date(date) : null;

}

// #endregion STEP5

// #region STEP6

// #endregion STEP6

// #region STEP7

public changeValue(event, actualPriceInputText: NriInputTextComponent) {

if (actualPriceInputText) {

this.policy.actualPrice = event;

this.cdRef.detectChanges();

}

}

public calcApTp(val1, val2) {

let result = 0;

if (!val1) {

val1 = 0;

}

if (!val2) {

val2 = 0;

}

if (+val1 !== 0 && +val2 !== 0) {

result = (((+val1) / (+val2)) \* 100);

}

this.policy.aPTP = result;

}

// #endregion STEP7

// #region STEP8

public activeBindingCheck(): number {

return PolicyUtils.activeBindingCheck(this.policy);

}

public setActiveInsuredClearCheck() {

this.activeInsuredClearCheck = PolicyUtils.activeInsuredClearCheck(this.policy);

}

public setActiveDatesClearCheck() {

this.activeDatesClearCheck = PolicyUtils.activeDatesClearCheck(this.policy);

}

public setActivePremiumClearCheck() {

this.activePremiumClearCheck = PolicyUtils.activePremiumClearCheck(this.policy);

}

public setActiveTermsClearCheck() {

this.activeTermsClearCheck = PolicyUtils.activeTermsClearCheck(this.policy);

}

public activeAmountCessionCheck(policy: PolicyDTO): number {

return PolicyUtils.activeAmountCessionCheck(this.policy);

}

public activeEmailConfirmCheck(policy: PolicyDTO): number {

return PolicyUtils.activeEmailConfirmCheck(this.policy);

}

public activeWithWhomCheck(policy: PolicyDTO): number {

return PolicyUtils.activeWithWhomCheck(this.policy);

}

public goAccelus() {

window.open('https://app.accelus.com');

}

public record(event) {

this.recordsActive = event;

if (event === 0) {

this.policy.sanctionsValidation = true;

} else {

this.policy.sanctionsValidation = false;

}

}

// #endregion STEP8

// #region STEP9

public setEditType(type: string) {

switch (type) {

case this.TYPEQUOTE:

this.isPolicyQuote = true;

this.isPolicyRenew = false;

this.isPolicyEndorse = false;

this.editType = 'New Quote';

break;

case this.TYPERENEW:

this.isPolicyRenew = true;

this.isPolicyQuote = false;

this.isPolicyEndorse = false;

this.editType = 'Renew policy';

break;

case this.TYPEENDORSE:

this.isPolicyEndorse = true;

this.isPolicyQuote = false;

this.isPolicyRenew = false;

this.editType = 'Endorse policy';

break;

}

}

public onAction(event) {

const docid = event[1].id;

const docname = event[1].name;

this.serviceUrl = process.env.CONTEXT.rest.updateDocument;

let numberOfversions: number = 1;

if (event[0] === 'Download') {

this.documentsService.downloadDoc(docid).then((response: any) => {

const file = window.URL.createObjectURL(new Blob([response.file], { type: response.file.type }));

const a = document.createElement('a');

a.href = file;

a.download = docname;

a.click();

});

}

if (event[0] === 'Update') {

this.serviceUrl = this.serviceUrl + '/' + docid;

this.openUpdateDocumentModal();

}

if (event[0] === 'View') {

this.dataContent = [];

this.fileSelected = [];

this.documentsService.getMetaData(docid).then((res: any) => {

this.blockScreenService.show('Opening Document...');

res.content.forEach((element) => {

this.dataContent.push(element);

this.documentsService.previewAllVersions(docid, element.majorVersion, element.minorVersion).then((response: any) => {

if (!!response && !!response.file) {

const files = new Blob([response.file], { type: response.file.type });

this.fileSelected.push({

data: files, version: +element.majorVersion, name: element.name,

type: element.type, docId: docid, majorVersion: element.majorVersion, minorVersion: element.minorVersion

});

this.fileSelected = this.fileSelected.sort(

// tslint:disable-next-line:only-arrow-functions

function(a, b) { return (a.version > b.version) ? 1 : ((b.version > a.version) ? -1 : 0); }).concat([]);

this.blockScreenService.hide();

} else {

this.blockScreenService.hide();

}

numberOfversions = this.dataContent.length - 1;

this.viewFile = {

numberVersions: numberOfversions,

title: this.dataContent[this.dataContent.length - 1].name,

description: res.description,

typeDoc: this.dataContent[this.dataContent.length - 1].type,

addedBy: res.author,

lastModification: this.dataContent[this.dataContent.length - 1].dateModified

};

this.versionSelected = this.fileSelected.length - 1;

const viewerComp = document.getElementById('viewerComp');

const link = viewerComp.getElementsByClassName('href-style')[0] as HTMLElement;

link.click();

this.blockScreenService.hide();

});

});

});

}

if (event[0] === 'Delete') {

const index = this.policy.documents.findIndex((doc) => {

return doc.id === docid;

});

this.documentToDelete = this.policy.documents[index];

this.modalMessage = 'You will delete ' +

(this.documentToDelete.version ? 'the version ' + this.documentToDelete.version + ' of the file ' : ' the document ') +

this.documentToDelete.name;

this.openWarningModal();

}

this.setActive30daysCheck();

this.standarSubForm();

this.emailConfirm();

}

public pushDoc(documentSelect: NriInputSelectComponent) {

let document = {} as DocumentDTO;

this.documentTypesOpt.forEach((element) => {

if (+element.value === this.docTypeId) {

this.docType = element.text;

}

});

if (this.fileAdded && !!this.docTypeId && this.docName !== '') {

document.name = this.docName;

document.documentType = {} as DocumentTypeDTO;

document.documentType.id = this.docTypeId;

document.documentType.documentType = this.docType;

document.userID = {} as UserDTO;

document.userID.id = this.userDetails.id;

document.userID.name = this.userDetails.givenName;

document.description = this.docDescription;

document.version = this.docVersion;

document.modificationTime = new Date();

document.policy = {

id: this.policy.id

} as PolicyDTO;

this.policy.documents.push(document);

this.policy.documents = this.policy.documents.concat([]);

this.userAddedFiles = true;

}

document = {} as DocumentDTO;

this.docName = '';

this.docType = '';

this.docTypeId = undefined;

this.docDescription = '';

this.fileAdded = false;

this.hideDocumentMessage();

documentSelect.clearField();

this.setActive30daysCheck();

this.standarSubForm();

this.emailConfirm();

this.activeDocBtn();

}

public setDocName(event) {

setTimeout(() => {

if (!!event) {

this.docName = event[0].name;

this.activeDocBtn();

}

}, 500);

}

public getFiles(event) {

setTimeout(() => {

if (!!event) {

this.uploadFiles.setfiles(event, this.uploadComponentUrl);

}

}, 500);

}

public setDocType(event) {

if (!!event) {

this.docTypeId = +event;

}

}

public activeDocBtn() {

if (this.fileAdded && !!this.docTypeId && this.docName !== '') {

this.isActiveDocBtn = false;

} else {

this.isActiveDocBtn = true;

}

}

public fileSubmited() {

if (this.saving) {

this.savePolicy();

this.saving = false;

}

}

public fileSubmitionError() {

this.fileUploaded = false;

this.openWarningUploadModal();

this.saving = false;

this.savingPolicy = false;

}

public getDocumentTypeByDomain() {

this.documentTypesOpt = this.manageCommunService.documentTypesOpt;

}

public enableSave(): boolean {

if (!!this.policy &&

((this.isPolicyEndorse || this.isPolicyRenew || this.isPolicyQuote) ||

(!!this.policy.policyNumber && this.policy.policyNumber.length > 4)

)) {

return this.validateEnquiry();

} else if (!this.policy) {

return false;

} else {

return true;

}

}

// Returns true if it is an enquiry

public validateEnquiry(): boolean {

if (this.titles[0].check === true && this.titles[2].check === true) {

return true;

}

return false;

}

public async openLockedModal(lockingUserId) {

await this.userService.getUserDetailsById(lockingUserId).then((userDetails: UserInfoDTO) => {

this.otherUserEditing = userDetails.fullName;

this.lockedModal.open();

if (this.keepAliveID) {

clearInterval(this.keepAliveID);

this.keepAliveID = null;

}

});

}

public async onSave() {

if (this.policy.underwritingProcess && this.isPolicyRenew && Object.keys(this.policy.underwritingProcess.underwriter).length === 0) {

this.policy.underwritingProcess.underwriter = this.userDetails;

}

if (this.policy.underwritingProcess && this.policy.underwritingProcess.tasks && !this.policy.underwritingProcess.tasks[0].completedDate) {

this.policy.underwritingProcess.tasks[0].completedDate = new Date();

this.policy.underwritingProcess.tasks[0].completedBy = this.userDetails;

}

if (this.policy.underwritingProcess && this.policy.underwritingProcess.tasks && !this.policy.underwritingProcess.tasks[1].completedDate && this.policy.sanctionsValidation) {

this.policy.underwritingProcess.tasks[1].completedDate = new Date();

this.policy.underwritingProcess.tasks[1].completedBy = this.userDetails;

}

if (!this.isPolicyEndorse && !this.isPolicyRenew && !this.isPolicyQuote) {

await this.entityLockService.checkPolicyLock(this.policy.id, this.lockId).then(async (lock: LockResult) => {

if (!lock || lock.lockId !== this.lockId) {

if (!lock.locked) {

this.notLocked = true;

}

await this.openLockedModal(lock.lockingUserId);

}

});

}

if (this.otherUserEditing) {

return;

}

this.disableFinishButtons();

if (this.userAddedFiles) {

this.saving = true;

this.uploadPolicyDocuments(this.policy.policyNumber, this.policy.documents);

this.userAddedFiles = false;

} else {

if (this.isPolicyRenew) {

if (this.policy.coverageID.coverageID != null && this.policy.startDate > new Date()) {

this.openEnquiryBoundModal();

} else if (this.policy.coverageID.coverageID != null && this.policy.startDate <= new Date()) {

this.openEnquiryPolicyInForceModal();

}

} else {

this.savePolicy();

}

}

}

public updateGrid(event: DocumentDTO) {

const newDoc = {} as DocumentDTO;

newDoc.id = event.id;

newDoc.name = event.name;

newDoc.userID = event.userID;

newDoc.documentType = event.documentType;

newDoc.version = event.version;

newDoc.modificationTime = new Date();

this.policy.documents.forEach((doc, index, object) => {

if (doc.id === event.id) {

this.policy.documents[index] = newDoc;

this.policy.documents = this.policy.documents.concat([]);

}

});

this.docsToRow(this.policy.documents);

}

public savePolicy() {

this.clearUploadFiles();

this.policy.policyPremiumTaxesID = this.policy.policyPremiumTaxesID.concat(this.creditNoteTaxList);

this.policy.policyPremiumTaxesID = this.policy.policyPremiumTaxesID.concat(this.debitNoteTaxList);

this.policy.deductionID.forEach((x) => {

switch (x.deductionNameID.name.toUpperCase()) {

case 'ESC':

x = this.eSCFee;

break;

case 'INLA':

x = this.iNLAFFee;

break;

case 'SURVEY FEES':

x = this.surveyFee;

break;

case 'OTHER DEDUCTIONS':

x = this.cedingInsurerOtherDeductuionFee;

break;

}

});

if (this.policy.deductibleID) {

this.policy.deductibleID.forEach((deduct: DeductibleDTO) => {

if (!deduct.currencyID && !!this.policy.currencyID) {

deduct.currencyID = { id: this.policy.currencyID.id } as CurrencyDTO;

if (this.policy.rOE) {

deduct.rate = this.policy.rOE;

}

}

});

}

if (this.policy.relatedPolicyID) {

this.policy.relatedPolicyID.forEach((related, value) => {

const relatedPlcId = {} as PolicyDTO;

relatedPlcId.id = related.id;

this.policy.relatedPolicyID[value] = relatedPlcId;

});

}

this.updatePolicyInfoForSave();

const policyCall = PolicyUtils.cleanPolicyDto(ObjectUtils.copyObjectWithCorrectDates(this.policy));

const thisScope = this;

if (this.isPolicyEndorse) {

this.policyService.createPolicy(policyCall, 'ENDORSEMENT')

.subscribe((res: PolicyDTO) => {

thisScope.policy.policyNumber = res.policyNumber;

this.resultPolicy = res;

thisScope.openSuccessModal();

this.enableFinishButtons();

},

(err) => {

thisScope.openFailureModal();

this.enableFinishButtons();

});

} else if (this.isPolicyRenew) {

this.blockScreenService.show('Renewing Policy...');

thisScope.policyService.createPolicy(policyCall, 'RENEWAL').subscribe(

(res: PolicyDTO) => {

this.blockScreenService.hide();

this.resultPolicy = res;

thisScope.policy.policyNumber = res.policyNumber;

thisScope.openSuccessModal();

if (!thisScope.fileAdded || thisScope.fileUploaded) {

if (res.policyStatusID.policyStatusName === 'Documentation to issue') {

thisScope.downloadEndorsmentDocDocument('RENEWAL', thisScope.policy.id);

}

}

switch (res.policyStatusID.policyStatusName) {

case 'Documentation to issue':

thisScope.downloadEndorsmentDocDocument('SCHEDULE', thisScope.policy.id);

break;

case 'Cover note to issue':

thisScope.downloadEndorsmentDocDocument('COVERNOTE', thisScope.policy.id);

break;

}

this.enableFinishButtons();

},

(err) => {

this.blockScreenService.hide();

thisScope.openFailureModal();

this.enableFinishButtons();

}

);

} else if (this.isPolicyQuote) {

thisScope.policyService.createPolicy(policyCall, 'QUOTE').subscribe(

(res: PolicyDTO) => {

this.resultPolicy = res;

thisScope.policy.policyNumber = res.policyNumber;

thisScope.openSuccessModal();

this.enableFinishButtons();

},

(err) => {

thisScope.openFailureModal();

this.enableFinishButtons();

}

);

} else {

this.blockScreenService.show('Updating Policy...');

thisScope.policyService.updatePolicy(policyCall).subscribe(

(res: PolicyDTO) => {

this.resultPolicy = res;

if (res.id === undefined) {

this.blockScreenService.hide();

thisScope.openFailureModal();

} else {

if (thisScope.policy.policyStatusID &&

thisScope.policy.policyStatusID.policyStatusCode === 'WAITING\_FOR\_FIRM\_ORDER' &&

res.policyStatusID && res.policyStatusID.policyStatusCode !== 'WAITING\_FOR\_FIRM\_ORDER') {

this.blockScreenService.hide();

this.downloadStandardExcelDocument(this.policy.id);

this.downloadCreditDebitNoteDocDocument(this.policy.id);

}

thisScope.policy.policyNumber = res.policyNumber;

thisScope.policy.policyStatusID = res.policyStatusID;

thisScope.policy.policyVersion = res.policyVersion;

thisScope.policy.rAGcolourID = res.rAGcolourID;

thisScope.policy.quoteNumber = res.quoteNumber;

res.documents.forEach((doc) => {

doc.policy = { id: res.id } as PolicyDTO;

});

this.fileAdded = false;

this.docsToRow(res.documents);

this.blockScreenService.hide();

thisScope.openSuccessModal();

}

this.enableFinishButtons();

},

(error) => {

this.blockScreenService.hide();

thisScope.openFailureModal();

this.enableFinishButtons();

}

);

}

}

public updatePolicyInfoForSave() {

this.policy.deductionID = [];

this.policy.upfrontNcbAmount = 0;

this.policy.upfrontNcbPct = 0;

this.policy.defferedNcbAmount = 0;

this.policy.defferedNcbPct = 0;

this.policy.brokerageAmount = 0;

this.policy.brokeragePct = 0;

this.policy.brokerType = null;

this.policy.ipeExpenseAmount = 0;

this.policy.ipeExpensePct = 0;

this.deductionsAndFees.forEach((deduction, index) => {

switch (deduction.typeOfDeductions.toUpperCase()) {

case 'OTHER DEDUCTIONS':

const deductionsToAdd = {} as DeductionDTO;

deductionsToAdd.deductionNameID = {id: 4, name: deduction.typeOfDeductions} as DeductionNameDTO;

deductionsToAdd.amount = this.removeCommas(deduction.deductionAmount);

deductionsToAdd.percentage = this.removeCommas(deduction.deduction);

deductionsToAdd.deductionComment = deduction.comment;

deductionsToAdd.commissionOrFee = deduction.commissionOrFee;

if (deduction.deductionOption.toUpperCase() === 'DEDUCTION %') {

deductionsToAdd.deductionTypeID = this.comboInfo.otherDeductionType[0];

} else {

deductionsToAdd.deductionTypeID = this.comboInfo.otherDeductionType[1];

}

if (deduction.deductionID) {

deductionsToAdd.id = deduction.deductionID;

}

this.policy.deductionID.push(deductionsToAdd);

break;

case 'UPFRONT NCB':

this.policy.upfrontNcbAmount = this.removeCommas(deduction.deductionAmount);

this.policy.upfrontNcbPct = this.removeCommas(deduction.deduction);

break;

case 'DEFERRED NCB':

this.policy.defferedNcbAmount = this.removeCommas(deduction.deductionAmount);

this.policy.defferedNcbPct = this.removeCommas(deduction.deduction);

break;

case 'BROKERAGE FEES':

this.policy.brokerageAmount = this.removeCommas(deduction.deductionAmount);

this.policy.brokeragePct = this.removeCommas(deduction.deduction);

if (deduction.deductionOption.toUpperCase() === 'DEDUCTION %') {

this.policy.brokerType = this.comboInfo.brokerageType[0].id.toString();

} else {

this.policy.brokerType = this.comboInfo.brokerageType[1].id.toString();

}

break;

case 'IPE EXPENSE':

this.policy.ipeExpenseAmount = this.removeCommas(deduction.deductionAmount);

this.policy.ipeExpensePct = this.removeCommas(deduction.deduction);

break;

}

});

this.policy.ipeIncomeAmount = 0;

this.policy.ipeIncomePct = 0;

this.incomeRows.forEach((income) => {

switch (income.typeOfIncome.toUpperCase()) {

case 'SURVEY INCOME':

const deductionsToAdd = {} as DeductionDTO;

deductionsToAdd.deductionNameID = {id: 1, name: income.typeOfIncome} as DeductionNameDTO;

deductionsToAdd.amount = this.removeCommas(income.incomeAmount);

deductionsToAdd.percentage = this.removeCommas(income.income);

if (income.incomeOption.toUpperCase() === 'INCOME %') {

deductionsToAdd.deductionTypeID = this.allDeductionTypes.find((deductionType) => deductionType.id === 1);

} else {

deductionsToAdd.deductionTypeID = this.allDeductionTypes.find((deductionType) => deductionType.id === 2);

}

if (income.incomeID) {

deductionsToAdd.id = income.incomeID;

}

this.policy.deductionID.push(deductionsToAdd);

break;

case 'IPE INCOME':

this.policy.ipeIncomeAmount = this.removeCommas(income.incomeAmount);

this.policy.ipeIncomePct = this.removeCommas(income.income);

break;

}

});

this.policy.policyPremiumTaxesID = [];

this.taxes.forEach((tax, index) => {

const premiumTaxes = {} as PolicyPremiumTaxesDTO;

if (index < this.taxes.length - 1) {

if (+tax.taxAmount !== 0 || +tax.tax !== 0) {

premiumTaxes.taxAmount = this.removeCommas(tax.taxAmount);

premiumTaxes.taxPercentage = this.removeCommas(tax.tax);

const option = this.policy.countryOptions[this.policy.countryOptionsStr.indexOf(tax.country)];

premiumTaxes.countryID = {id: +option.value} as CountryDTO;

premiumTaxes.taxTypeID = this.taxType.find((taxType) => taxType.id === 1);

if (tax.type === '% of NRI Net Premium') {

premiumTaxes.taxTypeCalculation = this.taxTypeCalculations.find((taxTypeCalc) => taxTypeCalc.id === 8);

} else if (tax.type === '% of NRI Gross Premium') {

premiumTaxes.taxTypeCalculation = this.taxTypeCalculations.find((taxTypeCalc) => taxTypeCalc.id === 9);

} else {

premiumTaxes.taxTypeCalculation = this.taxTypeCalculations.find((taxTypeCalc) => taxTypeCalc.id === 7);

}

if (tax.taxId) {

premiumTaxes.id = tax.taxId;

}

this.policy.policyPremiumTaxesID.push(premiumTaxes);

}

}

});

}

public downloadCreditDebitNoteDocDocument(policyId: number) {

this.policyService.downloadDocDocument(policyId).subscribe(

(res) => {

if (res === {}) {

throw Error;

}

const byteCharacters = atob(res['report']);

const byteNumbers = new Array(byteCharacters.length);

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

byteNumbers[i] = byteCharacters.charCodeAt(i);

}

const byteArray = new Uint8Array(byteNumbers);

const docType = 'application/vnd.openxmlformats-officedocument.wordprocessingml.document';

const data = new Blob([byteArray], { type: docType });

const report = window.URL.createObjectURL(data);

const a = document.createElement('a');

a.href = report;

a.download = [res['title']] + '.';

a.click();

return res;

},

(error) => {

throw Error;

}

);

}

public downloadStandardExcelDocument(policyId: number) {

this.policyService.downloadStandardExcelDocument(policyId).subscribe(

(res) => {

if (res === {}) {

throw Error;

}

const byteCharacters = atob(res['report']);

const byteNumbers = new Array(byteCharacters.length);

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

byteNumbers[i] = byteCharacters.charCodeAt(i);

}

const byteArray = new Uint8Array(byteNumbers);

const docType = 'application/vnd.ms-excel';

const data = new Blob([byteArray], { type: docType });

const report = window.URL.createObjectURL(data);

const a = document.createElement('a');

a.href = report;

a.download = [res['title']] + '.';

a.click();

return res;

},

(error) => {

throw Error;

}

);

}

public downloadEndorsmentDocDocument(type, policyId) {

this.policyService.downloadEndorsmentDocDocument(type, policyId).subscribe(

(res) => {

if (res === {}) {

throw Error;

}

const byteCharacters = atob(res['report']);

const byteNumbers = new Array(byteCharacters.length);

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

byteNumbers[i] = byteCharacters.charCodeAt(i);

}

const byteArray = new Uint8Array(byteNumbers);

const docType = 'application/vnd.openxmlformats-officedocument.wordprocessingml.document';

const data = new Blob([byteArray], { type: docType });

const report = window.URL.createObjectURL(data);

const a = document.createElement('a');

a.href = report;

a.download = [res['title']] + '.';

a.click();

return res;

},

(error) => {

throw Error;

}

);

}

public uploadPolicyDocuments(policyNumber, documents) {

const documentInfo = [] as PolicyDocument[];

this.policy.documents.forEach((doc) => {

if (!doc.id) {

documentInfo.unshift({

policyNumber: this.policy.id.toString(),

name: doc.name,

type: doc.documentType.id.toString()

});

}

});

this.uploadFiles.updateAndUploadFiles(documentInfo);

this.setActive30daysCheck();

this.standarSubForm();

this.emailConfirm();

}

public continueDeleteFile() {

this.modalMessage = '';

this.documentsService.deleteFile(this.documentToDelete.id).then((res: DocumentDTO) => {

if (!res || res.version === null || res.version === undefined) {

this.modalMessage = 'There was an error saving the file';

this.openFailureModalCustom();

} else {

const index = this.policy.documents.indexOf(this.documentToDelete);

if (res.version < 0) {

this.policy.documents.splice(index, 1);

} else {

this.policy.documents[index] = res;

}

this.docsToRow(this.policy.documents);

this.modalMessage = (this.documentToDelete.version ? 'The version ' + this.documentToDelete.version + ' of the file ' :

'The document ') +

this.documentToDelete.name + ' was deleted';

this.openSuccessModalCustom();

this.documentToDelete = null;

}

}).catch((e) => {

this.modalMessage = 'There was an error deleting the file';

this.openFailureModalCustom();

});

this.setActive30daysCheck();

this.standarSubForm();

this.emailConfirm();

}

public onBack() {

this.router.navigate(['/policies', 'view', this.policy.id]);

}

public onLockBack() {

this.finishedWork = true;

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

}

// #endregion STEP9

/\* create policy function \*/

public enableFinish(): boolean {

if (this.titles[0].check === true && this.titles[1].check === true

&& this.titles[2].check === true && this.titles[3].check === true && this.titles[4].check === true && this.fileAdded) {

this.finishDisable = false;

} else {

this.finishDisable = true;

}

return this.finishDisable;

}

public setEnquiry() {

if (this.policy.endDate > this.policy.renewalDate) {

this.modalMessage = 'Renewal date cannot be earlier than expiry date.';

this.openFailureModalCustom();

return;

}

let instalmentDateNumberVal = true;

let currencyVal = true;

if (this.policy.numberOfInstalments === undefined) {

this.policy.numberOfInstalments = 0;

}

if (this.policy.numberOfInstalments > 0) {

this.policy.instalmentDate.forEach((instDate) => {

if (!instDate) {

instalmentDateNumberVal = false;

}

});

}

if (!this.policy.currencyID) {

currencyVal = false;

}

this.validBasicDetails();

this.validStructure();

if (this.validateEnquiry() && instalmentDateNumberVal && currencyVal) {

if (!this.policy.authoriseCapacity && this.policy.coverageID.coverageID) {

this.showCapacityMessage();

} else {

this.openConfirmationModal();

}

} else {

if (instalmentDateNumberVal && currencyVal) {

this.modalMessage = 'Not all the mandatory information is completed. Please fill all mandatory fields.';

} else if (!instalmentDateNumberVal) {

this.modalMessage = 'Please ensure the number of premium instalment dates is the same as the number of instalments.';

} else if (!currencyVal) {

this.modalMessage = 'Please select a valid currency.';

}

this.openFailureModalCustom();

}

}

public insuranceStructureRadioSelect(event) {

this.policy.insuranceEstructure.forEach((element) => {

if (element.value === event.toString()) {

if (!!this.policy.insuranceStructureID) {

this.policy.insuranceStructureID = {

insuranceStructureID: Number(element.value),

insuranceStructureName: element.text

} as InsuranceStructureDTO;

if (element.text === 'Direct insurance') {

this.insuranceSelected = 0;

this.policy.insuranceSelected = 0;

this.showOtherFrontPanel = true;

this.showBrokerFields = true;

this.showBrokePanel = true;

this.showReiPanel = false;

} else { // Reinsurance Inwards

this.showOtherFrontPanel = false;

this.showBrokerFields = false;

this.insuranceSelected = 1;

this.policy.insuranceSelected = 1;

this.showReiPanel = true;

this.showBrokerReiInPanel = true;

}

}

}

});

this.closeChangeBusinessType();

this.setActualPriceNoValidation();

this.validBasicDetails();

this.validStructure();

}

public setDefaultInsured() {

const insuredObject = { id: this.policy.insured.id.toString(), name: this.policy.insured.fullName };

this.policy.allInsuredOptions = [insuredObject];

this.localInsuredDefault = [insuredObject] as NriInputMultiSelectOptions[];

}

public setDefaultCurrency() {

const currencyObj = { id: this.policy.currencyID.code, name: this.policy.currencyID.name };

this.localcurrency = [currencyObj] as NriInputMultiSelectOptions[];

}

public insuranceStructureSubmit() {

this.initModal.hide();

}

public goPolicies() {

this.managePolicyService.cleanPolicyDto(this.managePolicyService.policy);

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

}

public loadPolicy() {

this.policy.isLoading = true;

this.originalPolicyNumber = this.policy.policyNumber;

if (this.coverage) {

this.policy.coverage = this.coverage;

}

if (!this.policy.enquiry && !this.policy.policyNumber.endsWith('0')) {

this.showQuoteNumber = false;

}

if (this.policy.startDate) {

this.startDate = new Date(this.policy.startDate);

}

if (this.policy.endDate) {

this.endDate = new Date(this.policy.endDate);

}

if (!!this.policy.insured && !!this.policy.insured.id) {

this.setDefaultInsured();

}

if (!!this.policy.broker && !!this.policy.broker.id) {

this.policyBrokerDefault = [] as NriInputMultiSelectOptions[];

this.policyBrokerDefault.push({ id: this.policy.broker.id.toString(), name: this.policy.broker.fullName });

}

if (!!this.policy.nonPoolInsurer && !!this.policy.nonPoolInsurer.id) {

this.policy.showBrokerFields = true;

this.localNonPoolInsurerDefault = [] as NriInputMultiSelectOptions[];

this.localNonPoolInsurerDefault.push({

id: this.policy.nonPoolInsurer.id.toString(),

name: this.policy.nonPoolInsurer.fullName

});

}

if (!!this.policy.poolInsurer && !!this.policy.poolInsurer.id) {

this.localPoolInsurerDefault = [] as NriInputMultiSelectOptions[];

this.localPoolInsurerDefault.push({ id: this.policy.poolInsurer.id.toString(), name: this.policy.poolInsurer.fullName });

}

if (this.policy.policyShareID) {

if (this.policy.policyShareID.length > 1) {

this.policy.policyShareID.sort((a, b) => (

(a.organisationID ? a.organisationID.fullName : '').localeCompare(b.organisationID ? b.organisationID.fullName : '')

));

}

this.updateReinsuranceTableRows();

}

if (this.policy.policyPremiumTaxesID && this.policy.policyPremiumTaxesID.length > 0) {

this.policy.policyPremiumTaxesID.forEach((tax) => {

let taxTypeCalc = '% of NRI Net Premium';

if (tax.taxTypeCalculation && tax.taxTypeCalculation.name === '% of NRI Net Premium') {

taxTypeCalc = '% of NRI Net Premium';

} else if (tax.taxTypeCalculation && tax.taxTypeCalculation.name === '% of NRI Gross Premium') {

taxTypeCalc = '% of NRI Gross Premium';

} else if (tax.taxTypeCalculation && tax.taxTypeCalculation.name === 'Amount to be entered') {

taxTypeCalc = 'Amount to be entered';

}

const taxToTable = {

country: tax.countryID && tax.countryID.name ? tax.countryID.name : '',

type: taxTypeCalc,

tax: tax.taxPercentage ? this.addCommasToNumber(tax.taxPercentage) : '0.00',

taxAmount: tax.taxAmount ? this.addCommasToNumber(tax.taxAmount) : '0.00',

taxId: tax.id

} as Taxes;

this.taxes.splice(this.taxes.length - 1, 0, taxToTable);

});

}

if (this.policy.upfrontNcbPct || this.policy.upfrontNcbAmount) {

const deductionToAdd = {

typeOfDeductions: 'Upfront NCB',

deduction: this.policy.upfrontNcbPct ? this.addCommasToNumber(this.policy.upfrontNcbPct) : '0.00',

deductionOption: 'Deduction %', // FIXME: Add a type

deductionAmount: this.policy.upfrontNcbAmount ? this.addCommasToNumber(this.policy.upfrontNcbAmount) : '0.00',

comment: '',

commissionOrFee: ''

};

this.deductionsAndFees.splice(this.deductionsAndFees.length - 1, 0, deductionToAdd);

}

if (this.policy.defferedNcbPct || this.policy.defferedNcbAmount) {

const deductionToAdd = {

typeOfDeductions: 'Deferred NCB',

deduction: this.policy.defferedNcbPct ? this.addCommasToNumber(this.policy.defferedNcbPct) : '0.00',

deductionOption: 'Deduction %', // FIXME: Add a type

deductionAmount: this.policy.defferedNcbAmount ? this.addCommasToNumber(this.policy.defferedNcbAmount) : '0.00',

comment: '',

commissionOrFee: ''

};

this.deductionsAndFees.splice(this.deductionsAndFees.length - 1, 0, deductionToAdd);

}

if (this.policy.brokeragePct || this.policy.brokerageAmount) {

const deductionToAdd = {

typeOfDeductions: 'Brokerage fees',

deduction: this.policy.brokeragePct ? this.addCommasToNumber(this.policy.brokeragePct) : '0.00',

deductionOption: 'Deduction %', // FIXME: Add a type

deductionAmount: this.policy.brokerageAmount ? this.addCommasToNumber(this.policy.brokerageAmount) : '0.00',

comment: '',

commissionOrFee: ''

};

this.deductionsAndFees.splice(this.deductionsAndFees.length - 1, 0, deductionToAdd);

}

if (this.policy.ipeExpensePct || this.policy.ipeExpenseAmount) {

const deductionToAdd = {

typeOfDeductions: 'IPE expense',

deduction: this.policy.ipeExpensePct ? this.addCommasToNumber(this.policy.ipeExpensePct) : '0.00',

deductionOption: 'Deduction %', // FIXME: Add a type

deductionAmount: this.policy.ipeExpenseAmount ? this.addCommasToNumber(this.policy.ipeExpenseAmount) : '0.00',

comment: '',

commissionOrFee: ''

};

this.deductionsAndFees.splice(this.deductionsAndFees.length - 1, 0, deductionToAdd);

}

if (this.policy.deductionID && this.policy.deductionID.length > 0) {

this.policy.deductionID.forEach((deductionsSaved) => {

if (deductionsSaved.deductionNameID && deductionsSaved.deductionNameID.name) {

if (deductionsSaved.deductionNameID.name.toUpperCase() === 'OTHER DEDUCTIONS') {

let otherDeducCalc = 'Deduction %';

if (deductionsSaved.deductionTypeID && deductionsSaved.deductionTypeID.name === 'Amount to be entered') {

otherDeducCalc = 'Amount to be entered';

}

const deductionToAdd = {

typeOfDeductions: deductionsSaved.deductionNameID.name,

deduction: deductionsSaved.percentage ? this.addCommasToNumber(deductionsSaved.percentage) : '0.00',

deductionOption: otherDeducCalc,

deductionAmount: deductionsSaved.amount ? this.addCommasToNumber(deductionsSaved.amount) : '0.00',

comment: deductionsSaved.deductionComment ? deductionsSaved.deductionComment : '',

commissionOrFee: deductionsSaved.commissionOrFee ? deductionsSaved.commissionOrFee : '',

deductionID: deductionsSaved.id

};

this.deductionsAndFees.splice(this.deductionsAndFees.length - 1, 0, deductionToAdd);

} else if (deductionsSaved.deductionNameID.name.toUpperCase() === 'SURVEY INCOME') {

const incomeRow = {

typeOfIncome: deductionsSaved.deductionNameID.name,

income: deductionsSaved.percentage ? this.addCommasToNumber(deductionsSaved.percentage) : '0.00',

incomeOption: 'Income %', // FIXME: Add a type

incomeAmount: deductionsSaved.amount ? this.addCommasToNumber(deductionsSaved.amount) : '0.00',

incomeID: deductionsSaved.id

};

this.incomeRows.splice(this.incomeRows.length - 1, 0, incomeRow);

}

}

});

}

if (this.policy.ipeIncomePct || this.policy.ipeIncomeAmount) {

const incomeRow = {

typeOfIncome: 'IPE income',

income: this.policy.ipeIncomePct ? this.addCommasToNumber(this.policy.ipeIncomePct) : '0.00',

incomeOption: 'Income %', // FIXME: Add a type

incomeAmount: this.policy.ipeIncomeAmount ? this.addCommasToNumber(this.policy.ipeIncomeAmount) : '0.00'

};

this.incomeRows.splice(this.incomeRows.length - 1, 0, incomeRow);

}

if (this.isPolicyRenew) {

this.setRenewValues();

this.renewalDateIsValid();

}

if (this.isPolicyRenew || this.isPolicyQuote || this.isPolicyEndorse) {

this.policy.documents = [];

}

if (this.isPolicyEndorse || this.isPolicyRenew) {

this.policy.entryTypeID = undefined;

this.policy.sanctionsValidation = false;

}

if (this.policy.coverageID && this.policy.coverageID.coverageName) {

this.policy = PolicyUtils.setBooleansByCoverage(this.policy);

}

this.getComplementaryPolicyInformation();

this.docsToRow(this.policy.documents);

if (this.policy.coverageID && this.policy.coverageID.addOnID) {

this.addOnOriginalLen = this.policy.coverageID.addOnID.length;

}

if (!!this.policy.currencyID) {

this.setDefaultCurrency();

}

if (!this.policy.reinsuranceOut) {

this.policy.reinsuranceOut = false;

} else {

this.showReOutFields = this.policy.reinsuranceOut;

this.localPolicyReinsuranceOut = this.policy.reinsuranceOut;

}

if (this.policy.stationId === undefined) {

this.policy.stationId = [];

}

if (!!this.policy.locationId) {

this.policy.policyLocations = [];

this.policy.countrysOfLoc = [];

this.policy.stationsOfLoc = [];

this.policy.stationList = [];

let tempStationList = [];

this.policy.locationId.forEach((element, index) => {

this.policy.policyLocations.push({ id: element.id + '', name: element.name });

this.localPolicyLocations.push({ id: element.id.toString(), name: element.name });

let contains = false;

for (const country of this.countrysOfLoc) {

if (country.name === element.country.name) {

contains = true;

break;

}

}

if (!contains && !!element.country) {

this.countrysOfLoc.push({ id: element.country.id.toString(), name: element.country.name });

this.policy.countrysOfLoc.push({ id: element.country.id.toString(), name: element.country.name });

}

if (element.station) {

element.station.forEach((station: StationDTO, statIndex) => {

this.policy.stationsOfLoc.push({

id: station.id.toString(),

name: station.stationName ? station.stationName : (element.name + ' station ' + (statIndex + 1))

} as

NriInputMultiSelectOptions);

tempStationList.push({

id: station.id.toString(),

name: station.stationName ? station.stationName : (element.name + ' station ' + (statIndex + 1))

} as NriInputMultiSelectOptions);

});

this.policy.stationList[index] = tempStationList;

}

if (element.types && element.types.length > 0) {

element.locationTypes = '';

element.types.forEach((type) => {

element.locationTypes = (element.locationTypes ? element.locationTypes : '') + type.name + ', ';

});

element.locationTypes = element.locationTypes.slice(0, -2);

}

tempStationList = [];

});

}

// after finding the stations, go through the dropdowns and force update

// if (!!this.policy.stationId) {

// this.updatingStations = true;

// if (this.insuredStations.toArray().length > 0) {

// this.insuredStations.toArray().forEach((element, index) => {

// element.updateSelectedOptions(this.policy.stationIDList[index]);

// });

// }

// this.updatingStations = false;

// }

if (!!this.policy.sublimitID) {

this.policy.sublimitID.forEach((sublimit) => {

if (!!sublimit.bIDailyRate) {

sublimit.bIDailyRate = +(sublimit.bIDailyRate.toFixed(2));

}

if (!!sublimit.waitingPeriod) {

sublimit.waitingPeriod = +(sublimit.waitingPeriod.toFixed(2));

}

if (!!sublimit.currencyID) {

sublimit.defaultCurrency = [{ id: sublimit.currencyID.code, name: sublimit.currencyID.name }];

}

if (!!sublimit.aggregateLimitCurrencyID) {

sublimit.localAggregateLimitCurrencies = [];

sublimit.localAggregateLimitCurrencies.push(

{ id: sublimit.aggregateLimitCurrencyID.code, name: sublimit.aggregateLimitCurrencyID.name }

);

}

if (!!sublimit.locationID) {

this.setSublimitLocDefaults(sublimit);

}

if (!!sublimit.countryID) {

this.setSublimitCountryDefaults(sublimit);

}

if (!!sublimit.stationID) {

this.setSublimitStationDefaults(sublimit);

}

if (!!sublimit.deductibleID) {

const currencyForDeduct = this.policy.currencyID;

sublimit.showSublimitsDeduct = true;

sublimit.localLimitDeductCurrency = [];

if (sublimit.deductibleID.currencyID) {

sublimit.localLimitDeductCurrency.push(

{ id: sublimit.deductibleID.currencyID.code, name: sublimit.deductibleID.currencyID.name }

);

} else if (this.policy.currencyID && this.policy.currencyID.code && this.policy.currencyID.name) {

sublimit.deductibleID.currencyID = currencyForDeduct;

sublimit.localLimitDeductCurrency.push(

{ id: sublimit.deductibleID.currencyID.code, name: sublimit.deductibleID.currencyID.name }

);

}

if (sublimit.deductibleID.currencyID && this.policy.currencyID) {

const roe = {

from: sublimit.deductibleID.currencyID.code,

to: this.policy.currencyID.code,

date: moment(new Date(this.policy.rOEDate ? this.policy.rOEDate : this.policy.startDate)).format('YYYY-MM-DD'),

};

if (roe.to && roe.from && roe.date) {

roe.from = this.policy.currencyID.code;

roe.to = sublimit.deductibleID.currencyID.code;

this.currencyExchangeService.historicRate(roe).then((response) => {

if (!!response) {

sublimit.deductibleID.rate = +response;

this.getMinDeductible(this.policy);

}

});

} else if (roe.to && roe.from) {

this.currencyExchangeService.getExchangeRate(roe).then((response) => {

if (!!response) {

sublimit.deductibleID.rate = +response;

this.getMinDeductible(this.policy);

}

});

}

}

}

if (sublimit.addOnID && sublimit.addOnID.name.toLowerCase().includes('including bi')) {

sublimit.showBIfields = true;

}

});

}

if (!!this.policy.additionalLimitID) {

this.policy.additionalLimitID.forEach((addlimit) => {

if (!!addlimit.bIDailyRate) {

addlimit.bIDailyRate = +(addlimit.bIDailyRate.toFixed(2));

}

if (!addlimit.rate) {

let roe;

roe = {

from: null,

to: null,

date: null

};

if (this.policy.currencyID && this.policy.currencyID.code) {

roe.to = this.policy.currencyID.code;

}

roe.from = addlimit.currencyID ? addlimit.currencyID.code : (this.policy.currencyID ? this.policy.currencyID.code : null);

if (this.policy.rOEDate || this.policy.startDate) {

roe.date = moment(new Date((this.policy.rOEDate ? this.policy.rOEDate : this.policy.startDate))).format('YYYY-MM-DD');

}

if (roe.from && roe.to && roe.date && roe.date < moment(new Date()).format('YYYY-MM-DD')) {

roe.from = this.policy.currencyID.code;

roe.to = addlimit.currencyID ? addlimit.currencyID.code : (this.policy.currencyID ? this.policy.currencyID.code : null);

this.currencyExchangeService.historicRate(roe).then((response) => {

if (!!response) {

addlimit.rate = +response;

this.calcOverallPolicyLimit();

}

});

} else if (roe.from && roe.to) {

roe.date = null;

this.currencyExchangeService.getExchangeRate(roe).then((response) => {

if (!!response) {

addlimit.rate = +response;

this.calcOverallPolicyLimit();

}

});

}

}

if (!!addlimit.waitingPeriod) {

addlimit.waitingPeriod = +(addlimit.waitingPeriod.toFixed(2));

}

if (!!addlimit.currencyID) {

addlimit.defaultCurrency = [{ id: addlimit.currencyID.code, name: addlimit.currencyID.name }];

}

if (!!addlimit.aggregateLimitCurrencyID) {

addlimit.localAggregateLimitCurrencies = [];

addlimit.localAggregateLimitCurrencies.push(

{ id: addlimit.aggregateLimitCurrencyID.code, name: addlimit.aggregateLimitCurrencyID.name }

);

}

if (!!addlimit.locationID) {

this.setAddlimitLocationDefault(addlimit);

}

if (!!addlimit.countryID) {

this.setAddlimitCountryDefault(addlimit);

}

if (!!addlimit.stationID) {

this.setAddlimitStationDefault(addlimit);

}

if (!!addlimit.deductibleID) {

const currencyForDeduct = this.policy.currencyID;

addlimit.showSublimitsDeduct = true;

addlimit.localLimitDeductCurrency = [];

if (addlimit.deductibleID.currencyID) {

addlimit.localLimitDeductCurrency.push(

{ id: addlimit.deductibleID.currencyID.code, name: addlimit.deductibleID.currencyID.name }

);

} else if (this.policy.currencyID && this.policy.currencyID.code && this.policy.currencyID.name) {

addlimit.deductibleID.currencyID = currencyForDeduct;

addlimit.localLimitDeductCurrency.push(

{ id: addlimit.deductibleID.currencyID.code, name: addlimit.deductibleID.currencyID.name }

);

}

if (addlimit.deductibleID.currencyID && this.policy.currencyID) {

const roe = {

from: addlimit.deductibleID.currencyID.code,

to: this.policy.currencyID.code,

date: moment(new Date(this.policy.rOEDate ? this.policy.rOEDate : this.policy.startDate)).format('YYYY-MM-DD'),

};

if (roe.to && roe.from && roe.date) {

roe.from = this.policy.currencyID.code;

roe.to = addlimit.deductibleID.currencyID.code;

this.currencyExchangeService.historicRate(roe).then((response) => {

if (!!response) {

addlimit.deductibleID.rate = +response;

this.getMinDeductible(this.policy);

}

});

} else if (roe.to && roe.from) {

this.currencyExchangeService.getExchangeRate(roe).then((response) => {

if (!!response) {

addlimit.deductibleID.rate = +response;

this.getMinDeductible(this.policy);

}

});

}

}

}

if (addlimit.addOnID && addlimit.addOnID.name.toLowerCase().includes('including bi')) {

addlimit.showBIfields = true;

}

});

}

if (!!this.policy.users) {

this.policy.users.forEach((user) => {

this.usersOptions.push({ value: user.id.toString(), text: user.name });

});

}

if (this.policy.numberOfInstalments) {

if (this.policy.instalmentDate.length > 0) {

this.instalmentDate = [];

this.policy.instalmentDate.forEach((date) => {

this.instalmentDate.push(date);

});

} else {

this.instalmentDate = [];

}

if (this.policy.numberOfInstalments > this.policy.instalmentDate.length) {

const diffSize = this.policy.numberOfInstalments - this.policy.instalmentDate.length;

for (let i = diffSize; i > 0; i--) {

this.policy.instalmentDate.push(null);

this.instalmentDate.push(null);

}

}

}

if (this.policy.insuranceStructureID) {

switch (this.policy.insuranceStructureID.insuranceStructureName) {

case 'Direct insurance':

this.policy.insuranceSelected = 0;

this.showBrokerFields = true;

break;

case 'Reinsurance inwards':

this.policy.insuranceSelected = 1;

this.showBrokerFields = false;

break;

}

}

if (!this.policy.groundUpCover) {

this.policy.groundUpCover = false;

}

if (!this.policy.excessLayer) {

this.policy.excessLayer = false;

}

if (!this.policy.authoriseCapacity) {

this.policy.authoriseCapacity = false;

}

if (!this.policy.overallPolicyLimit) {

this.policy.overallPolicyLimit = 0;

}

if (!this.policy.overallPolicyLimitGBP) {

this.policy.overallPolicyLimitGBP = 0;

}

if (!this.policy.nRINetMaximumExposure) {

this.policy.nRINetMaximumExposure = 0;

}

if (!this.policy.nRINetMaximumExposureGBP) {

this.policy.nRINetMaximumExposureGBP = 0;

}

if (this.policy.broker) {

this.policy.showBrokerFields = true;

if (!!this.policy.broker.contacts) {

this.policy.broker.contacts.forEach((contact) => {

this.brokerContactOptions.push({ value: contact.contactDetailID.toString(), text: contact.fullName });

});

}

}

if (this.policy.sanctionsValidation === true) {

this.recordsActive = 0;

this.initialSactionsVaslidation = true;

} else {

this.recordsActive = 1;

}

if (!this.policy.hotzoneMB) {

this.policy.hotzoneMB = { id: null, amount: null, currency: null, code: '', rate: null };

}

if (this.policy.policyIncomeID.length > 0) {

this.isOtherIncomeEnable = true;

} else {

this.isOtherIncomeEnable = false;

}

if (!this.policy.hotzoneotherMB) {

this.policy.hotzoneotherMB = { id: null, amount: null, currency: null, code: '', rate: null };

}

if (!this.policy.coldzoneMB) {

this.policy.coldzoneMB = { id: null, amount: null, currency: null, code: '', rate: null };

}

if (!this.policy.coldzoneotherMB) {

this.policy.coldzoneotherMB = { id: null, amount: null, currency: null, code: '', rate: null };

}

if (this.policy.deductionID) {

this.policy.deductionID.forEach((deduction) => {

this.policyOtherDeductionType.push('');

deduction.amount = Number(deduction.amount.toFixed(2));

});

}

this.reducePolicyDecimals();

this.getPolicyFees();

this.manageCommunService.getComboInfo().then((response) => {

this.comboInfo = response;

this.comboInfo.deductionName.forEach((deduction) => {

const deductionType = {

value: deduction.id.toString(),

text: deduction.name

};

this.deductionNames.push(deductionType);

});

});

const brokerage = this.brokerageFEEOptions.find((x) => x.value === this.policy.brokerType);

switch (brokerage && brokerage.text) {

case 'NRI gross share Premium after deduction of upfront NCB':

this.policy.showFee = true;

this.policy.showAmount = false;

break;

case 'Amount to be entered':

this.policy.showFee = false;

this.policy.showAmount = true;

break;

case 'Sliding scale':

this.policy.showFee = false;

this.policy.showAmount = false;

break;

}

if (this.policy.underwritingProcess && this.policy.underwritingProcess.tasks && this.policy.underwritingProcess.tasks.length > 0) {

this.policy.underwritingProcess.tasks.forEach((task: UnderwritingTaskDTO) => {

task.completedDate = task.completedDate ? new Date(task.completedDate) : null;

task.reviewedDate = task.reviewedDate ? new Date(task.reviewedDate) : null;

});

}

this.directStepClick();

setTimeout(() => {

this.policy.isLoading = false;

}, 2000); // prevents the Authorise Capacity from being recalculated on loading

this.calcNRIshare();

this.setActive30daysCheck();

this.standarSubForm();

this.emailConfirm();

this.calcNriNetMaximun();

this.getMinDeductible(this.policy);

this.getAllActiveCurrencies();

this.calculateTotalDeductions();

this.calculateTotalReinsurer();

this.calculateTotalIncome();

this.recalculateTaxes();

this.managePolicyService.sendLoading(false);

}

public setAddlimitStationDefault(addlimit: SublimitDTO) {

if (addlimit.localLimitStations === undefined) {

addlimit.localLimitStations = [];

}

if (addlimit.stationID !== undefined) {

addlimit.stationID.forEach((station: StationDTO) => {

addlimit.localLimitStations.push({ id: station.id + '', name: station.stationName });

if (addlimit.localLimitStations.length > 1) {

addlimit.localLimitStations.splice(0, 1);

}

});

}

}

public setAddlimitCountryDefault(addlimit: SublimitDTO) {

if (addlimit.localLimitCountries === undefined) {

addlimit.localLimitCountries = [];

}

if (addlimit.countryID !== undefined) {

addlimit.countryID.forEach((country) => {

addlimit.localLimitCountries.push({ id: country.id + '', name: country.name });

if (addlimit.localLimitCountries.length > 1) {

addlimit.localLimitCountries.splice(0, 1);

}

});

}

}

public setAddlimitLocationDefault(addlimit: SublimitDTO) {

if (addlimit.localLimitLocations === undefined) {

addlimit.localLimitLocations = [];

}

if (addlimit.locationID !== undefined) {

addlimit.locationID.forEach((loc) => {

addlimit.localLimitLocations.push({ id: loc.id + '', name: loc.name });

if (addlimit.localLimitLocations.length > 1) {

addlimit.localLimitLocations.splice(0, 1);

}

});

}

}

public setSublimitStationDefaults(sublimit: SublimitDTO) {

if (sublimit.localLimitStations === undefined) {

sublimit.localLimitStations = [];

}

if (sublimit.stationID !== undefined) {

sublimit.stationID.forEach((station: StationDTO) => {

sublimit.localLimitStations.push({ id: station.id + '', name: station.stationName });

if (sublimit.localLimitStations.length > 1) {

sublimit.localLimitStations.splice(0, 1);

}

});

}

}

public setSublimitCountryDefaults(sublimit: SublimitDTO) {

if (sublimit.localLimitCountries === undefined) {

sublimit.localLimitCountries = [];

}

if (sublimit.countryID !== undefined) {

sublimit.countryID.forEach((country) => {

sublimit.localLimitCountries.push({ id: country.id + '', name: country.name });

if (sublimit.localLimitCountries.length > 1) {

sublimit.localLimitCountries.splice(0, 1);

}

});

}

}

public setSublimitLocDefaults(sublimit: SublimitDTO) {

if (sublimit.localLimitLocations === undefined) {

sublimit.localLimitLocations = [];

}

if (sublimit.locationID !== undefined) {

sublimit.locationID.forEach((loc) => {

sublimit.localLimitLocations.push({ id: loc.id + '', name: loc.name });

if (sublimit.localLimitLocations.length > 1) {

sublimit.localLimitLocations.splice(0, 1);

}

});

}

}

public getComplementaryPolicyInformation() {

if (!this.policy.locationId) {

this.policy.locationId = [];

}

if (!this.policy.users) {

this.policy.users = [];

}

if (!this.policy.policyShareID) {

this.policy.policyShareID = [];

}

if (!this.policy.relatedPolicyID) {

this.policy.relatedPolicyID = [];

}

if (!this.policy.sublimitID) {

this.policy.sublimitID = [];

}

if (!this.policy.additionalLimitID) {

this.policy.additionalLimitID = [];

}

if (!this.policy.coverageID) {

this.policy.coverageID = {} as CoverageDTO;

this.policy.coverageID.addOnID = [];

}

if (!this.policy.insuranceStructureID) {

this.policy.insuranceStructureID = {} as InsuranceStructureDTO;

}

if (!this.policy.deductionID) {

this.policy.deductionID = [];

}

if (!this.policy.deductibleID) {

this.policy.deductibleID = [];

}

if (this.policy.stationIDList === undefined) {

this.policy.stationIDList = [];

}

if (!this.policy.policyPremiumTaxesID) {

this.policy.policyPremiumTaxesID = [] as PolicyPremiumTaxesDTO[];

this.policy.policyPremiumTaxesID.push({} as PolicyPremiumTaxesDTO);

this.policy.policyPremiumTaxesID[0].taxPercentage = this.defaultTax;

}

if (!!this.policy.insured && this.policy.insured.location) {

this.insuredLocations = [];

this.policy.insured.location.forEach((loc: LocationDTO) => {

if (loc.id && loc.enabled) {

this.insuredLocations.push({ id: loc.id.toString(), name: loc.name });

}

});

}

this.policyStations = [];

if (!!this.policy.locationId) {

let tempInsuredStationIDList = [];

this.policy.locationId.forEach((location, locationIndex) => {

this.policyStations.push({ locId: location.id, stations: [] });

if (location.enabled) {

this.defaultInsuredLocations.push({ id: location.id.toString(), name: location.name });

if (!!location.country) {

if (this.policyCountryofLocation.length === 0) {

this.policyCountryofLocation = this.policyCountryofLocation.concat(location.country.name);

} else {

if (!this.policyCountryofLocation.includes(location.country.name)) {

this.moreThanOnePolicyCountry = true;

this.policyCountryofLocation = this.policyCountryofLocation.concat(', ' + location.country.name);

}

}

}

}

// find which stations of the insured are covered

if (!!location.station && !!this.policy.stationId) {

location.station.forEach((station, index) => {

this.policy.stationId.forEach((policyStation) => {

if (station.id === policyStation.id) {

this.defaultInsuredStations.push({

id: station.id.toString(),

name: station.stationName ? station.stationName : location.name + ' station ' + (index + 1)

});

tempInsuredStationIDList.push({

id: station.id.toString(),

name: station.stationName ? station.stationName : location.name + ' station ' + (index + 1)

});

this.policyStations[this.policyStations.findIndex((ps) => ps.locId === location.id)].stations.push(

station.id

);

}

});

});

this.policy.stationIDList[locationIndex] = tempInsuredStationIDList;

} else {

// stationIDList must have the same length as stationList otherwise mappings won't work

this.policy.stationIDList[locationIndex] = [];

}

tempInsuredStationIDList = [];

});

}

if (!!this.policy.addOnID) {

this.policy.addonOptions = [];

this.policy.sublimitsOptions = [{ value: null, text: 'Other' }];

this.policy.addOnID.forEach((element: AddOnDTO) => {

this.policy.addonOptions.push({

value: element.id.toString(),

text: element.name

} as NriInputSelectOptions);

this.policy.sublimitsOptions.push({ value: element.id.toString(), text: element.name });

});

this.addOnLen = this.policy.addOnID.length;

this.policy.sublimitsOptions.sort((a, b) => a.text.localeCompare(b.text));

}

if (!!this.policy.coverageID) {

this.policy.coverageID.addOnID.forEach((element: AddOnDTO) => {

if (this.policy.addOnID.filter((addOn: AddOnDTO) => addOn.id === element.id).length === 0) {

this.localCoverageAddons.push({

value: element.id.toString(),

text: element.name

} as NriInputSelectOptions);

}

this.policy.addonOptions.push({ value: element.id.toString(), text: element.name });

});

this.localCoverageAddons.sort((a, b) => a.text.localeCompare(b.text));

}

if (!!this.policy.deductibleID && this.policy.deductibleID.length > 0) {

// code to shift BI to end of list

let foundBI = false;

let countDeductible = 1;

this.policy.deductibleID.forEach((deduct) => {

if (countDeductible === 1 && deduct.deductibleTypeID && deduct.deductibleTypeID.name === 'BI deductibles amount') {

foundBI = true;

}

countDeductible++;

});

if (foundBI) {

this.policy.deductibleID.push(this.policy.deductibleID.splice(0, 1)[0]);

}

this.policy.deductibleID.forEach((deduct, index) => {

if (!!deduct.currencyID) {

if (!deduct.localCurrency) {

deduct.localCurrency = [];

}

deduct.localCurrency.push({ id: deduct.currencyID.code, name: deduct.currencyID.name });

if (!!deduct.currencyID && !!deduct.currencyID.code) {

const roe = {

from: deduct.currencyID.code,

to: this.policy.currencyID ? this.policy.currencyID.code ? this.policy.currencyID.code : null : null

};

if (roe.from && roe.to) {

this.currencyExchangeService.getExchangeRate(roe).then((response) => {

if (!!response) {

deduct.rate = +response;

this.getMinDeductible(this.policy);

}

});

} else {

deduct.rate = 1;

}

}

}

if (!!deduct.deductibleTypeID && deduct.deductibleTypeID.name === 'Deductibles for non-convencion cover') {

this.policy.nonConvCoverFields = true;

if (!!deduct.currencyID) {

this.localNonConventionCurrency.push({ id: deduct.currencyID.code, name: deduct.currencyID.name });

}

}

if (!!deduct.deductibleTypeID && deduct.deductibleTypeID.name === 'Deductible for convention cover') {

this.policy.convCoverFields = true;

if (!!deduct.currencyID) {

this.localConventionCurrency.push({ id: deduct.currencyID.code, name: deduct.currencyID.name });

}

}

if (!!deduct.deductibleTypeID && deduct.deductibleTypeID.name === 'Liability deductibles') {

this.policy.showLiabilityDeduct = true;

if (!!deduct.currencyID) {

this.localLiabilityCurrency.push({ id: deduct.currencyID.code, name: deduct.currencyID.name });

}

}

if (!!deduct.deductibleTypeID && deduct.deductibleTypeID.name === 'MD Single deductibles') {

this.policy.singleMdField = true;

this.policy.showMDsinglecheck = true;

this.policy.variousMdField = false;

this.policy.showMDvariouscheck = false;

if (!!deduct.currencyID) {

this.localmdSingleCurrency.push({ id: deduct.currencyID.code, name: deduct.currencyID.name });

}

}

if (!!deduct.deductibleTypeID && deduct.deductibleTypeID.name === 'Hot zone MB') {

this.policy.variousMdField = true;

this.policy.showMDvariouscheck = true;

this.policy.singleMdField = false;

this.policy.showMDsinglecheck = false;

this.policy.hotzoneMB = {

id: deduct.id, amount: deduct.amount,

currency: deduct.currencyID ? deduct.currencyID.id : null,

code: deduct.currencyID ? deduct.currencyID.code : null, rate: deduct.rate

};

if (!!deduct.currencyID) {

this.localHotZoneMBCurrency.push({ id: deduct.currencyID.code, name: deduct.currencyID.name });

}

}

if (!!deduct.deductibleTypeID && deduct.deductibleTypeID.name === 'Hot zone other MD') {

this.policy.variousMdField = true;

this.policy.showMDvariouscheck = true;

this.policy.singleMdField = false;

this.policy.showMDsinglecheck = false;

this.policy.hotzoneotherMB = {

id: deduct.id, amount: deduct.amount,

currency: deduct.currencyID ? deduct.currencyID.id : null,

code: deduct.currencyID ? deduct.currencyID.code : null, rate: deduct.rate

};

if (!!deduct.currencyID) {

this.localHotZoneMDCurrency.push({ id: deduct.currencyID.code, name: deduct.currencyID.name });

}

}

if (!!deduct.deductibleTypeID && deduct.deductibleTypeID.name === 'Cold zone MB') {

this.policy.variousMdField = true;

this.policy.showMDvariouscheck = true;

this.policy.singleMdField = false;

this.policy.showMDsinglecheck = false;

this.policy.coldzoneMB = {

id: deduct.id, amount: deduct.amount,

currency: deduct.currencyID ? deduct.currencyID.id : null,

code: deduct.currencyID ? deduct.currencyID.code : null, rate: deduct.rate

};

if (!!deduct.currencyID) {

this.localColdZoneMBCurrency.push({ id: deduct.currencyID.code, name: deduct.currencyID.name });

}

}

if (!!deduct.deductibleTypeID && deduct.deductibleTypeID.name === 'Cold zone other MD') {

this.policy.variousMdField = true;

this.policy.showMDvariouscheck = true;

this.policy.singleMdField = false;

this.policy.showMDsinglecheck = false;

this.policy.coldzoneotherMB = {

id: deduct.id, amount: deduct.amount,

currency: deduct.currencyID ? deduct.currencyID.id : null,

code: deduct.currencyID ? deduct.currencyID.code : null, rate: deduct.rate

};

if (!!deduct.currencyID) {

this.localColdZoneMDCurrency.push({ id: deduct.currencyID.code, name: deduct.currencyID.name });

}

}

if (!deduct.deductibleTypeID) {

deduct.isOtherDeductible = true;

}

if (deduct.isNotNil === true) {

this.policy.isNotNilChecked = true;

}

if (deduct.isNotNilForConvention === true) {

this.policy.isNotNilForConventionChecked = true;

}

if (deduct.isNotNilForNonConvention === true) {

this.policy.isNotNilForNonConventionChecked = true;

}

if (!deduct.deductibleTypeID) {

deduct.isOtherDeductible = true;

}

if (index >= 4) {

this.policy.variousMdField = true;

this.policy.showMDvariouscheck = true;

deduct.isOtherDeductible = true;

}

});

}

if (!!this.policy.deductionID) {

this.policy.deductionID.forEach((element) => {

if (!!element.deductionNameID && !!element.deductionTypeID) {

if (element.deductionTypeID.name === 'NRI gross share Premium after deduction of upfront NCB') {

element.showDeductionPercentageField = true;

} else if (element.deductionTypeID.name === 'Amount to be entered') {

element.deductAmountField = false;

}

}

});

}

if (!!this.policy.startDate) {

this.defaultHour= (new Date(this.policy.startDate).getHours()<10)

? "0"+new Date(this.policy.startDate).getHours().toString()

: new Date(this.policy.startDate).getHours().toString();

this.defaultMinute= (new Date(this.policy.startDate).getMinutes()<10)

? "0"+new Date(this.policy.startDate).getMinutes().toString()

: new Date(this.policy.startDate).getMinutes().toString();

}

if (!!this.policy.endDate) {

this.defaultEndHour = (new Date(this.policy.endDate).getHours()<10)

? "0"+new Date(this.policy.endDate).getHours().toString()

: new Date(this.policy.endDate).getHours().toString();

this.defaultEndMinute = (new Date(this.policy.endDate).getMinutes()<10)

? "0"+new Date(this.policy.endDate).getMinutes().toString()

: new Date(this.policy.endDate).getMinutes().toString();

}

if (!!this.policy.renewalDate) {

this.defaultRenewalHour =( new Date(this.policy.renewalDate).getHours()<10)

? "0"+new Date(this.policy.renewalDate).getHours().toString()

: new Date(this.policy.renewalDate).getHours().toString();

this.defaultRenewalMinute = (new Date(this.policy.renewalDate).getMinutes()<10)

? "0"+new Date(this.policy.renewalDate).getMinutes().toString()

: new Date(this.policy.renewalDate).getMinutes().toString()

}

if (this.policy.totalCedantShare && this.policy.totalPoolInsuranceShare) {

this.oldTotalReinsuranceShare = this.policy.totalPoolInsuranceShare === this.policy.totalCedantShare ? true : false;

}

this.getAllPolicies();

this.getEntryTypesByName();

this.getUsersByGroup();

this.getAllCountries();

if (!!this.manageCommunService.comboInfo) {

this.allPolicyStatus = this.manageCommunService.comboInfo.allPolicyStatus as PolicyStatusDTO[];

this.policy.insuranceEstructure = [];

(this.manageCommunService.comboInfo.insuranceStructures as InsuranceStructureDTO[]).forEach((element) => {

this.policy.insuranceEstructure.push({

value: element.insuranceStructureID.toString(),

text: element.insuranceStructureName

});

});

} else {

this.getAllPolicyStatus();

this.getInsuranceStructures();

}

this.getOrganisationsByType();

this.getCoverageLimits(false);

}

public getEntryTypesByName() {

let entryType: string = '';

if (this.isPolicyEndorse) {

entryType = 'Endorsement';

} else if (this.isPolicyRenew) {

entryType = 'Renewal';

} else if (this.policy && this.policy.entryTypeID && this.policy.entryTypeID.entryTypeName) {

entryType = this.policy.entryTypeID.entryTypeName;

} else {

entryType = 'New business';

}

this.policyService.getEntryTypesByName({ entryTypeName: entryType })

.then((response) => {

(response as EntryTypeDTO[]).forEach((element) => {

this.entryTypeOpt.push({

value: String(element.entryTypeId),

text: element.entrySubType

} as NriInputSelectOptions);

});

});

}

public getOrganisationsByType() {

this.policy.poolInsurersOptions = [];

this.policy.governArrayOpt = [];

this.policy.poolFACopt = [];

this.policy.poolreArrayOpt = [];

this.policy.poolrenuclearArrayOpt = [];

this.brokerContactOptions = [];

const tempBrokerOptions = [] as NriInputMultiSelectOptions[];

const tempNonPoolOptions = [] as NriInputMultiSelectOptions[];

const tempPoolOptions = [] as NriInputMultiSelectOptions[];

const tempPoolAliasOptions = [] as NriInputMultiSelectOptions[];

const tempGovernmentOptions = [] as NriInputMultiSelectOptions[];

const tempGovernmentaliasOptions = [] as NriInputMultiSelectOptions[];

const tempPoolReOptions = [] as NriInputMultiSelectOptions[];

const tempPoolReAliasOptions = [] as NriInputMultiSelectOptions[];

const tempPoolReNuclearOptions = [] as NriInputMultiSelectOptions[];

const tempPoolReNuclearAliasOptions = [] as NriInputMultiSelectOptions[];

const tempPoolFacOptions = [] as NriInputMultiSelectOptions[];

const tempPoolFacOptAlias = [] as NriInputMultiSelectOptions[];

tempBrokerOptions.push({ id: 'N/A', name: 'N/A' });

this.organisationService.getOrganizationLists()

.then((response) => {

this.allInsured = response['INSURED'];

this.brokerArray = response['BROKER'];

this.nonPoolInsurersArray = response['NON\_POOL\_INSURER'];

this.poolInsurersArray = response['POOL\_INSURER'];

this.governArray = response['GOV'];

this.poolreArray = response['POOL\_RE'];

this.poolrenuclearArray = response['POOL\_RE\_NUCLEAR'];

this.poolFACArray = response['FAC'];

if (!this.brokerArray) {

this.brokerArray = [];

}

this.brokerArray.forEach((element) => {

tempBrokerOptions.push({ id: element.id.toString(), name: element.fullName });

this.brokerOptions = tempBrokerOptions;

});

if (!this.nonPoolInsurersArray) {

this.nonPoolInsurersArray = [];

}

this.nonPoolInsurersArray.forEach((element) => {

tempNonPoolOptions.push({ id: element.id.toString(), name: element.fullName });

this.nonPoolInsurerOptions = tempNonPoolOptions;

this.localNonPoolInsurerDefault = this.localNonPoolInsurerDefault;

});

if (!this.poolInsurersArray) {

this.poolInsurersArray = [];

}

this.poolInsurersArray.forEach((element) => {

tempPoolOptions.push({ id: element.id.toString(), name: element.fullName });

tempPoolAliasOptions.push({ id: element.id.toString(), name: element.nrialias });

this.policy.poolInsurersOptions = tempPoolOptions;

});

this.policy.poolInsurersAliasOptions = tempPoolAliasOptions;

if (!this.governArray) {

this.governArray = [];

}

this.governArray.forEach((element) => {

tempGovernmentOptions.push({ id: element.id.toString(), name: element.fullName });

tempGovernmentaliasOptions.push({ id: element.id.toString(), name: element.nrialias });

this.policy.governArrayOpt = tempGovernmentOptions;

});

this.policy.governAliasArrayOpt = tempGovernmentaliasOptions;

if (!this.poolFACArray) {

this.poolFACArray = [];

}

this.poolFACArray.forEach((element) => {

tempPoolFacOptions.push({ id: element.id.toString(), name: element.fullName });

tempPoolFacOptAlias.push({ id: element.id.toString(), name: element.nrialias });

this.policy.poolFACopt = tempPoolFacOptions;

});

this.policy.poolFACoptAlias = tempPoolFacOptAlias;

if (!this.poolreArray) {

this.poolreArray = [];

}

this.poolreArray.forEach((element) => {

tempPoolReOptions.push({ id: element.id.toString(), name: element.fullName });

tempPoolReAliasOptions.push({ id: element.id.toString(), name: element.nrialias });

this.policy.poolreArrayOpt = tempPoolReOptions;

});

this.policy.poolreAliasArrayOpt = tempPoolReAliasOptions;

if (!this.poolrenuclearArray) {

this.poolrenuclearArray = [];

}

this.poolrenuclearArray.forEach((element) => {

tempPoolReNuclearOptions.push({ id: element.id.toString(), name: element.fullName });

tempPoolReNuclearAliasOptions.push({ id: element.id.toString(), name: element.nrialias });

this.policy.poolrenuclearArrayOpt = tempPoolReNuclearOptions;

});

this.policy.poolrenuclearAliasArrayOpt = tempPoolReNuclearAliasOptions;

if (!this.policy.allInsured) {

this.policy.allInsured = [];

}

if (!!this.policy.policyShareID) {

this.policy.policyShareID.forEach((element) => {

if (element.insurerRole === 'reout') {

this.policy.showReOutFields = true;

this.showAutoFields = true;

if (!!element && !!element.organisationID && !!element.organisationID.organisationType

&& !!element.organisationID.organisationType.organisationType) {

switch (element.organisationID.organisationType.organisationType) {

case 'Pool Insurer':

element.reOutOptions = this.policy.poolInsurersAliasOptions;

break;

case 'Government':

element.reOutOptions = this.policy.governAliasArrayOpt;

break;

case 'Pool Re':

element.reOutOptions = this.policy.poolreAliasArrayOpt;

break;

case 'Pool Re Nuclear':

element.reOutOptions = this.policy.poolrenuclearAliasArrayOpt;

break;

case 'FAC':

element.reOutOptions = this.policy.poolFACoptAlias;

break;

}

}

}

});

}

this.localBrokerDefaultValue = !!this.policy.broker && !!this.policy.broker.id ? this.policy.broker.id.toString() : '';

this.localBrokerContactDefaultValue = !!this.policy.brokerContact && !!this.policy.brokerContact.contactDetailID ?

this.policy.brokerContact.contactDetailID.toString() : '';

this.localPoolInsurersOptions = this.policy.poolInsurersOptions;

this.localReOutTypeOrg = this.policy.ReOutTypeOrg;

});

}

public getUsersByGroup() {

this.userService.getUsersByGroup(['Engineer', 'Underwriter'])

.then((response) => {

this.users = response as UserInfoDTO[];

const tempUserOpt = [] as NriInputMultiSelectOptions[];

const tempUserList = [] as NriInputSelectOptions[];

(response as any[]).forEach((element) => {

tempUserOpt.push({

id: element.id.toString(),

name: element.fullName

});

tempUserList.push({

text: element.fullName,

value: element.id

});

});

this.userOpt = tempUserOpt;

this.underwriterList = tempUserList;

this.underwriterListChosen();

});

}

public getAllCountries() {

this.policy.countryOptions = [];

this.policy.countryOptionsStr = [];

this.countryService.getAllCountries()

.then((response) => {

(response as CountryDTO[]).forEach((element) => {

this.policy.countryOptions.push(

{

value: String(element.id),

text: element.name

});

this.policy.countryOptionsStr.push(element.name);

});

});

}

public getAllPolicyStatus() {

this.policyService.getAllPolicyStatus()

.subscribe(

((res: PolicyStatusDTO[]) => {

this.allPolicyStatus = res;

})

);

}

public getInsuranceStructures() {

this.policy.insuranceEstructure = [];

this.policyService.getInsuranceStructures()

.then((response) => {

(response as InsuranceStructureDTO[]).forEach((element) => {

this.policy.insuranceEstructure.push(

{

value: String(element.insuranceStructureID),

text: element.insuranceStructureName

});

});

});

}

public getAllPolicies() {

this.policy.policiesOptions = [];

this.policy.policiesOptionsStr = [];

this.policyService.getAllPoliciesForSelection()

.then((response) => {

this.policy.policies = response as PolicyDTO[];

(response as PolicyDTO[]).forEach((element) => {

this.policy.policiesOptions.push(

{

value: String(element.id),

text: element.policyNumber

});

this.policy.policiesOptionsStr.push(element.policyNumber);

});

});

}

public getCoverageLimits(updateCapacity: boolean) {

if (this.policy.coverageID && this.policy.coverageID.coverageID && this.policy.yearOfAccount) {

this.policyService.getCoverageLimits(this.policy.coverageID.coverageID, this.policy.yearOfAccount)

.then((response: CoverageLimitDTO) => {

this.policy.standardLimit = response.standardLimit;

this.policy.authoriseCapacityLimit = response.authoriseLimit;

if (updateCapacity) {

this.policy = PolicyUtils.checkCapacity(this.policy);

}

});

}

}

public setRenewValues() {

// Period

const period = PolicyUtils.differenceBetweenDates(new Date(this.policy.startDate), new Date(this.policy.endDate));

// Start Date

const newInitDate = this.policy.renewalDate !== null && this.policy.renewalDate !== undefined ? new Date(this.policy.renewalDate) : null;

this.policy = PolicyUtils.setStartDate(this.policy, newInitDate, this.defaultHour, this.defaultMinute);

// End date

const newEndDate = new Date(this.policy.startDate);

newEndDate.setDate(newEndDate.getDate() + period);

this.policy = PolicyUtils.setEndDate(this.policy, newEndDate, this.defaultEndHour, this.defaultEndMinute);

// Renewal date

let newRenewalDate = null;

if (this.policy.endDate !== null && this.policy.endDate !== undefined) {

newRenewalDate = new Date(this.policy.endDate);

newRenewalDate.setDate(newRenewalDate.getDate() + 1);

}

this.policy = PolicyUtils.setRenewalDate(this.policy, newRenewalDate, this.defaultEndHour, this.defaultEndMinute);

// Reset re-out organisation acceptance

if (!!this.policy.policyShareID) {

this.policy.policyShareID.forEach((policyShare: PolicyShareDTO) => {

if (policyShare.insurerRole.toLowerCase() === 'reout') {

policyShare.rEOUTStatus = undefined;

}

});

}

// NRIIMPLEM-2613 - set policy to enquiry unless user changes

this.policy.policyStatusID.policyStatusCode = 'ENQUIRY';

this.policy.policyStatusID.policyStatusID = 1;

}

public docsToRow(\_array) {

this.policy.documents = ObjectUtils.copyObject(\_array);

this.setActive30daysCheck();

this.standarSubForm();

this.emailConfirm();

}

public emailConfirm() {

this.emailConfirmation = PolicyUtils.emailConfirm(this.policy);

}

public standarSubForm() {

this.isStandarSubForm = PolicyUtils.standarSubForm(this.policy);

}

public setActive30daysCheck() {

this.active30daysCheck = PolicyUtils.active30daysCheck(this.policy);

}

public definePolicyROEDate() {

this.setPolicyROE(0);

const today = new Date();

const startTodayDate = new Date(today.getFullYear(), today.getMonth(), today.getDate(),

+this.defaultHour, +this.defaultMinute, 0, 0);

const policyStartDate = new Date(this.policy.startDate);

if (startTodayDate <= policyStartDate) {

this.setPolicyROEDate(today);

} else {

this.setPolicyROEDate(policyStartDate);

}

}

public setPolicyROEDate(date: Date): void {

if (date) {

const roeDate = new Date(date);

const fullROEDate = new Date(roeDate.getFullYear(), roeDate.getMonth(), roeDate.getDate(),

+this.defaultHour, +this.defaultMinute, 0, 0);

const today = new Date();

const startTodayDate = new Date(today.getFullYear(), today.getMonth(), today.getDate(),

+this.defaultHour, +this.defaultMinute, 0, 0);

if (this.policy.rOEDate === fullROEDate) { return; }

this.policy.rOEDate = fullROEDate;

if (this.policy.currencyID && this.policy.currencyID.code) {

let roe;

this.setPolicyROE(0);

if (startTodayDate <= fullROEDate) {

roe = {

from: this.policy.currencyID.code,

to: 'GBP'

};

this.currencyExchangeService.getExchangeRate(roe).then((response) => {

if (!!response) {

this.autoFilledRate = true;

const truncatedResponse = this.setDecimals(+response, 6);

this.setPolicyROE(truncatedResponse);

}

});

} else {

roe = {

from: 'GBP',

to: this.policy.currencyID.code,

date: moment(new Date(roeDate)).format('YYYY-MM-DD')

};

this.currencyExchangeService.historicRate(roe).then((response) => {

if (!!response) {

this.autoFilledRate = true;

const truncatedResponse = this.setDecimals(+response, 6);

this.setPolicyROE(truncatedResponse);

}

});

}

}

} else {

this.policy.rOEDate = undefined;

}

}

public setPolicyROE(rate: number) {

this.policy.rOE = rate;

this.calcOverallPolicyLimit();

this.calcLimitPerOcurr(this.policy.limitPerOcurrenceFor100, this.policy.rOE);

this.getMinDeductible(this.policy);

}

public calcOverallPolicyLimit() {

let result = 0;

let overallPolicyLimitCalc = 0;

let overallPolicyGBPLimitCalc = 0;

if (this.policy.additionalLimitID) {

this.policy.additionalLimitID.forEach((addLimit) => {

if (addLimit.sublimitValue && addLimit.rate) {

result += (+addLimit.sublimitValue / +addLimit.rate);

}

});

}

overallPolicyLimitCalc = (this.policy.limitPerOcurrenceFor100 ? +this.policy.limitPerOcurrenceFor100 : 0);

if (result !== 0) {

overallPolicyLimitCalc += result;

}

if (overallPolicyLimitCalc !== this.policy.overallPolicyLimit) {

this.policy.overallPolicyLimit = +overallPolicyLimitCalc;

}

if (this.policy.rOE) {

overallPolicyGBPLimitCalc = (this.policy.overallPolicyLimit / this.policy.rOE);

}

if (+this.policy.overallPolicyLimitGBP !== +overallPolicyGBPLimitCalc) {

this.policy.overallPolicyLimitGBP = +overallPolicyGBPLimitCalc;

}

this.calcNriNetMaximun();

}

public calcNriNetMaximun() {

this.policy = PolicyUtils.calcNriNetMaximun(this.policy);

this.policy.nRINetMaximumExposure = this.setDecimals( this.policy.nRINetMaximumExposure , 0) ;

this.calcNetMax(this.policy.nRINetMaximumExposure, this.policy.rOE);

}

public calcLimitPerOcurr(val1, val2) {

this.policy = PolicyUtils.calcLimitPerOcurr(this.policy, val1, val2);

this.calcNriNetMaximun();

this.policy.limitPerOcurrenceFor100GBP = this.setDecimals( this.policy.limitPerOcurrenceFor100GBP, 0) ;

return this.policy.limitPerOcurrenceFor100GBP ? this.policy.limitPerOcurrenceFor100GBP.toFixed(0) : 0;

}

public calcNetMax(val1, val2) {

this.policy = PolicyUtils.calcNetMax(this.policy, val1, val2);

}

public getMinDeductible(policy: PolicyDTO): number {

this.policy = PolicyUtils.getMinDeductible(this.policy);

return this.policy.minimumDeductiblesForThePolicy ? +this.policy.minimumDeductiblesForThePolicy.toFixed(0) : 0;

}

public reducePolicyDecimals() {

if (this.policy.brokerageAmount) {

this.policy.brokerageAmount = Number(this.policy.brokerageAmount.toFixed(2));

}

if (this.policy.cedantShare1) {

this.policy.cedantShare1 = Number(this.policy.cedantShare1.toFixed(4));

}

if (this.policy.cedantShare2) {

this.policy.cedantShare2 = Number(this.policy.cedantShare2.toFixed(4));

}

if (this.policy.iPEAndOtherFeesToNRI) {

this.policy.iPEAndOtherFeesToNRI = Number(this.policy.iPEAndOtherFeesToNRI.toFixed(2));

}

if (this.policy.iPEincomeGBP) {

this.policy.iPEincomeGBP = Number(this.policy.iPEincomeGBP.toFixed(2));

}

if (this.policy.ipeExpenseAmount) {

this.policy.ipeExpenseAmount = Number(this.policy.ipeExpenseAmount.toFixed(2));

}

if (this.policy.limitPerOcurrenceFor100GBP) {

this.policy.limitPerOcurrenceFor100GBP = Number(this.policy.limitPerOcurrenceFor100GBP.toFixed(2));

}

if (this.policy.nRIGSAfterDeductionPrimaryCeding) {

this.policy.nRIGSAfterDeductionPrimaryCeding = Number(this.policy.nRIGSAfterDeductionPrimaryCeding.toFixed(2));

}

if (this.policy.nRIGSPremiumAfterUpfrontDeferredNCB) {

this.policy.nRIGSPremiumAfterUpfrontDeferredNCB = Number(this.policy.nRIGSPremiumAfterUpfrontDeferredNCB.toFixed(2));

}

if (this.policy.nRIGSPremiumAfterUpfrontNCB) {

this.policy.nRIGSPremiumAfterUpfrontNCB = Number(this.policy.nRIGSPremiumAfterUpfrontNCB.toFixed(2));

}

if (this.policy.nRIGShareAfterIPEAndOtherFeesToNRI) {

this.policy.nRIGShareAfterIPEAndOtherFeesToNRI = Number(this.policy.nRIGShareAfterIPEAndOtherFeesToNRI.toFixed(2));

}

if (this.policy.nRIGrossSharePremiumAfterNCBBrokerageFees) {

this.policy.nRIGrossSharePremiumAfterNCBBrokerageFees = Number(this.policy.nRIGrossSharePremiumAfterNCBBrokerageFees.toFixed(2));

}

if (this.policy.totalOverallPoolShare) {

this.policy.totalOverallPoolShare = Number(this.policy.totalOverallPoolShare.toFixed(4));

}

if (this.policy.nRIPremiumDebt) {

this.policy.nRIPremiumDebt = Number(this.policy.nRIPremiumDebt.toFixed(2));

}

if (this.policy.nriNetPremiumBeforeIpe) {

this.policy.nriNetPremiumBeforeIpe = Number(this.policy.nriNetPremiumBeforeIpe.toFixed(2));

}

if (this.policy.nRInetPremium) {

this.policy.nRInetPremium = Number(this.policy.nRInetPremium.toFixed(2));

}

if (this.policy.overallPolicyLimitGBP) {

this.policy.overallPolicyLimitGBP = Number(this.policy.overallPolicyLimitGBP.toFixed(2));

}

if (this.policy.totalReinsurerPctOf100) {

this.policy.totalReinsurerPctOf100 = Number(this.policy.totalReinsurerPctOf100.toFixed(4));

}

if (this.policy.grossPremium100Pct) {

this.policy.grossPremium100Pct = Number(this.policy.grossPremium100Pct.toFixed(2));

}

}

public directStepClick(event?, withRouting?: boolean) {

this.stepValidations();

if (event === 6 && this.policy !== undefined) {

for (let share of this.policy.policyShareID) {

if (share.insurerRole.toLowerCase() === 'reout') {

share = this.calcPremiumAfterDeductIPE(share);

share = this.calcIPEotherFees(share);

share = this.calcPremiumBeforeDeductIPE(share);

this.calcTotalReinsurerNetPremium();

this.calcTotalIPEAndOtherFeesToNRI();

this.calcTotalReinsurerPremiumBeforeDeductionOfIPEAndOtherFeesToNRI();

}

}

}

if (withRouting) {

if (!this.policy.underwritingProcess && +event >= 7) {

this.activateStep(event, this.route.routeConfig.children[event].path, true);

} else {

this.activateStep(event, this.route.routeConfig.children[event - 1].path, true);

}

}

}

public validBasicDetails() {

this.titles = PolicyUtils.validBasicDetails(this.policy, this.titles);

}

public validStructure() {

this.titles = PolicyUtils.validStructure(this.policy, this.titles);

}

public calcPremiumBeforeDeductIPE(insured: PolicyShareDTO): PolicyShareDTO {

insured = PolicyUtils.calcPremiumBeforeDeductIPE(this.policy, insured);

return insured;

}

public calcPremiumAfterDeductIPE(insured: PolicyShareDTO): PolicyShareDTO {

insured = PolicyUtils.calcPremiumAfterDeductIPE(this.policy, insured);

return insured;

}

public calcIPEotherFees(insur): PolicyShareDTO {

let result = 0;

if (this.calcPremiumBeforeDeductIPE(insur).premiumReinsurerBeforeIpe && this.policy.ipeIncomePct) {

result = ((+this.calcPremiumBeforeDeductIPE(insur).premiumReinsurerBeforeIpe) \* ((+this.policy.ipeIncomePct) / 100));

}

insur.iPEAndOtherFeesToNRI = result ? result.toFixed(2) : 0;

return insur;

}

public calcTotalReinsurerPremiumBeforeDeductionOfIPEAndOtherFeesToNRI(): number {

let total = 0;

if (this.policy && this.policy.policyShareID) {

this.policy.policyShareID.forEach((insur) => {

if (insur.insurerRole === 'reout') {

total += this.calcPremiumBeforeDeductIPE(insur).premiumReinsurerBeforeIpe

? Number(this.calcPremiumBeforeDeductIPE(insur).premiumReinsurerBeforeIpe)

: 0;

this.policy.totalReinsurerPremiumBeforeDeductionOfIPEAndOtherFeesToNRI = total;

}

});

}

return total;

}

public calcTotalReinsurerNetPremium() {

let total = 0;

if (this.policy && this.policy.policyShareID) {

this.policy.policyShareID.forEach((insur) => {

if (insur.insurerRole === 'reout') {

total += this.calcPremiumAfterDeductIPE(insur).premiumReinsurerAfterIpe

? Number(this.calcPremiumAfterDeductIPE(insur).premiumReinsurerAfterIpe)

: 0;

this.policy.totalReinsurerNetPremium = total;

}

});

}

}

public calcTotalIPEAndOtherFeesToNRI() {

let total = 0;

if (this.policy && this.policy.policyShareID) {

this.policy.policyShareID.forEach((insur) => {

if (insur.insurerRole === 'reout') {

total += this.calcIPEotherFees(insur).iPEAndOtherFeesToNRI ? Number(this.calcIPEotherFees(insur).iPEAndOtherFeesToNRI) : 0;

this.policy.totalIpeAndOtherFeesToNri = total;

}

});

}

}

public getPolicyFees() {

this.policy.deductionID.forEach((x) => {

switch (x.deductionNameID.name.toUpperCase()) {

case 'ESC':

this.eSCFee = x;

this.isESCFeeEnabled = true;

this.eSCFee.amount = x.amount;

this.eSCFee.percentage = x.percentage;

this.eSCFee.deductionTypeID = x.deductionTypeID;

break;

case 'INLA':

this.iNLAFFee = x;

this.isINLAFFeeEnabled = true;

this.iNLAFFee.amount = x.amount;

this.iNLAFFee.percentage = x.percentage;

this.iNLAFFee.deductionTypeID = x.deductionTypeID;

break;

case 'SURVEY FEES':

if (x.deductionCedingTypeID && x.deductionCedingTypeID.deductionCedingType === 'Primary ceding insurer') {

this.surveyFee = x;

this.isSurveyFeeEnabled = true;

this.surveyFee.amount = x.amount;

this.surveyFee.percentage = x.percentage;

this.surveyFee.deductionTypeID = x.deductionTypeID;

} else {

this.cedingInsurerSurveyFee = x;

this.isCedingInsurerSurveyFeeEnabled = true;

this.cedingInsurerSurveyFee.amount = x.amount;

this.cedingInsurerSurveyFee.percentage = x.percentage;

this.cedingInsurerSurveyFee.deductionTypeID = x.deductionTypeID;

}

break;

case 'NRI as Ceding insurer survey fee'.toUpperCase():

this.cedingInsurerSurveyFee = x;

this.isCedingInsurerSurveyFeeEnabled = true;

this.cedingInsurerSurveyFee.amount = x.amount;

this.cedingInsurerSurveyFee.percentage = x.percentage;

this.cedingInsurerSurveyFee.deductionTypeID = x.deductionTypeID;

break;

}

});

}

public selectProcessOwner(event) {

if (!!event) {

if (!this.policy.underwritingProcess) {

this.policy.underwritingProcess = {} as UnderwritingProcessDTO;

}

this.policy.underwritingProcess['underwriter'] = {

id: event,

fullName : this.underwriterList.find((x) => x.value.toUpperCase() === event.toUpperCase()).text

};

this.underwriterListChosen();

}

}

public selectProcessReviewer(event) {

if (!!event) {

if (!this.policy.underwritingProcess) {

this.policy.underwritingProcess = {} as UnderwritingProcessDTO;

}

this.policy.underwritingProcess['secondUnderwriter'] = {

id: event,

fullName : this.underwriterList.find((x) => x.value.toUpperCase() === event.toUpperCase()).text

};

this.underwriterListChosen();

}

}

public checkUnderwritingDate(event, index) {

if (this.policy.underwritingProcess) {

if (!!event && event.length > 0) {

this.policy.underwritingProcess.tasks[index].completedDate = new Date();

this.policy.underwritingProcess.tasks[index].completedBy = this.userDetails;

} else if (event.length === 0) {

this.policy.underwritingProcess.tasks[index].completedDate = null;

this.policy.underwritingProcess.tasks[index].completedBy = null;

}

} else {

if (!!event && event.length > 0) {

this.policy.underwritingProcess['tasks'][index].completedDate = new Date();

this.policy.underwritingProcess['tasks'][index].completedBy = this.userDetails;

} else if (event.length === 0) {

this.policy.underwritingProcess['tasks'][index].completedDate = null;

this.policy.underwritingProcess['tasks'][index].completedBy = null;

}

}

}

public checkUnderwritingReviewDate(event, index) {

if (this.policy.underwritingProcess) {

if (!!event && event.length > 0) {

const date = new Date().setHours(0, 0, 0, 0);

this.policy.underwritingProcess.tasks[index].reviewedDate = new Date(date);

this.policy.underwritingProcess.tasks[index].reviewedBy = this.userDetails;

} else if (event.length === 0) {

this.policy.underwritingProcess.tasks[index].reviewedDate = null;

this.policy.underwritingProcess.tasks[index].reviewedBy = null;

}

} else {

if (!!event && event.length > 0) {

const date = new Date().setHours(0, 0, 0, 0);

this.policy.underwritingProcess['tasks'][index].reviewedDate = new Date(date);

this.policy.underwritingProcess['tasks'][index].reviewedBy = this.userDetails;

} else if (event.length === 0) {

this.policy.underwritingProcess['tasks'][index].reviewedDate = null;

this.policy.underwritingProcess['tasks'][index].reviewedBy = null;

}

}

}

public setUnderwritingNotes(event) {

if (!!event) {

if (this.policy.underwritingProcess) {

this.policy.underwritingProcess.notes = event.toString();

} else {

this.policy.underwritingProcess['notes'] = event.toString();

}

}

}

public activateStep(stepId: number, stepName: string, directStepClick?: boolean, direction?: 'forward' | 'backward', policyView?: boolean) {

if (stepId === 2) {

if (this.policy.endDate > this.policy.renewalDate) {

this.modalMessage = 'Renewal date cannot be earlier than expiry date.';

this.openFailureModalCustom();

return;

}

}

if (directStepClick) {

this.stepValidations();

}

this.manageCommunService.activateStep(stepId, stepName, this.route);

}

public activateStepView(stepId: number, stepName: string) {

if (!this.policy.underwritingProcess && stepId > 7) {

this.manageCommunService.activateStep((stepId - 1), stepName, this.route);

} else {

this.manageCommunService.activateStep(stepId, stepName, this.route);

}

}

public getAllMinutes() {

for (let index = 0; index < 24; index++) {

this.hours.push((index<=9)

? { value: 0+index.toString(), text: 0+index.toString() }

: { value: index.toString(), text: index.toString() }); }

}

public getAllHours() {

for (let index = 0; index < 60; index++) {

this.minutes.push( (index<=9)

? { value: 0+index.toString(), text: 0+index.toString() }

: { value: index.toString(), text: index.toString() } ); }

}

public onOk() {

this.finishedWork = true;

this.router.navigate(['policies']);

}

public tickAndSaveCapacityExceeded() {

this.policy.authoriseCapacity = true;

this.onSave();

}

public onExitPolicy() {

this.finishedWork = true;

this.router.navigate(['/policies', 'view', this.policy.id]);

}

public onStay() {

this.closeSucessModal();

}

public updateReinsuranceTableRows() {

if (this.policy.policyShareID) {

this.reinsuranceTableRows = [];

this.policy.policyShareID.forEach((share: PolicyShareDTO, index) => {

if (share.organisationID && share.organisationID.id && share.organisationID.nRIalias) {

share.defaultInsurerName = [] as NriInputMultiSelectOptions[];

share.defaultInsurerName.push({ id: share.organisationID.id.toString(), name: share.organisationID.nRIalias });

}

if (share.insurerRole === 'reout') {

const reinsuranceToTable = {

reinsurerName: share.organisationID ? (share.organisationID.nRIalias ? share.organisationID.nRIalias : 'N/A') : 'N/A',

reinsurer: share.reinsurerPct ? share.reinsurerPct : '0.00',

reinsurerPremiumBeforeDeductionOfIpeAndOtherFeesToNri: share.premiumReinsurerBeforeIpe ? this.addCommasToNumber(share.premiumReinsurerBeforeIpe) : '0.00',

ipeAndOtherFeesToNri: share.iPEAndOtherFeesToNRI ? this.addCommasToNumber(share.iPEAndOtherFeesToNRI) : '0.00',

reinsurerNetPremium: share.premiumReinsurerAfterIpe ? this.addCommasToNumber(share.premiumReinsurerAfterIpe) : '0.00'

} as ReinsuranceTableRows;

this.reinsuranceTableRows.push(reinsuranceToTable);

}

});

this.reinsuranceTableRows.sort((a, b) => a.reinsurerName.localeCompare(b.reinsurerName));

const totalReinRow = { reinsurerName: 'Total Reinsurance',

reinsurer: '0.00',

reinsurerPremiumBeforeDeductionOfIpeAndOtherFeesToNri: '0.00',

ipeAndOtherFeesToNri: '0.00', reinsurerNetPremium: '0.00'

} as ReinsuranceTableRows;

this.reinsuranceTableRows.push(totalReinRow);

this.calculateTotalReinsurer();

}

}

public policyStatusRadioSelect1(event) {

if (event !== undefined) {

if (event.toString() === '1') {

this.policy.policyStatusID.policyStatusCode = 'BOUND';

this.policy.policyStatusID.policyStatusID = 20;

this.statusSelected1 = 1;

} else if (event.toString() === '0') {

this.policy.policyStatusID.policyStatusCode = 'ENQUIRY';

this.policy.policyStatusID.policyStatusID = 1;

this.statusSelected1 = 0;

}

}

}

public policyStatusRadioSelect2(event) {

if (event !== undefined) {

if (event.toString() === '0') {

this.policy.policyStatusID.policyStatusCode = 'ENQUIRY';

this.policy.policyStatusID.policyStatusID = 1;

this.statusSelected2 = 0;

} else if (event.toString() === '1') {

this.policy.policyStatusID.policyStatusCode = 'POLICY\_IN\_FORCE';

this.policy.policyStatusID.policyStatusID = 7;

this.statusSelected2 = 1;

}

}

}

private stepValidations() {

this.validBasicDetails();

this.validStructure();

}

private underwriterListChosen() {

if (this.policy.underwritingProcess && this.policy.underwritingProcess.secondUnderwriter) {

this.underwriterList1 = this.underwriterListSpliced(this.policy.underwritingProcess.secondUnderwriter);

} else {

this.underwriterList1 = this.underwriterList;

}

if (this.policy.underwritingProcess && this.policy.underwritingProcess.underwriter) {

this.underwriterList2 = this.underwriterListSpliced(this.policy.underwritingProcess.underwriter);

} else {

this.underwriterList2 = this.underwriterList;

}

}

private underwriterListSpliced(underwriter: UserDTO) {

const list = [...this.underwriterList];

const chosenItem: NriInputSelectOptions = {

text: underwriter.fullName,

value: underwriter.id

};

const index = list.findIndex((option: NriInputSelectOptions) => {

return option.value === chosenItem.value;

});

list.splice(index, 1);

return list;

}

/\*\*

\* @description This function check if the instalmentDate is 90 days bigger than endDate

\* @private

\*/

private instalmentDateVerification(endDate: Date, instalmentDate: Date, instalmentDateIndex: number) {

if (!this.message) {

this.message = [];

}

if (this.isValidDate(endDate) && this.isValidDate(instalmentDate)) {

const ninetyDaysAfterEndDate: Date = new Date(endDate.setDate(endDate.getDate() + 90));

if (instalmentDate.getTime() > ninetyDaysAfterEndDate.getTime()) {

this.message[instalmentDateIndex] = {

type: 'warning',

text: 'Date is 90 days above expiry date'

};

} else {

this.message[instalmentDateIndex] = null;

}

} else {

this.message[instalmentDateIndex] = null;

}

}

private isValidDate(date: Date): boolean {

if (!date) {

return false;

}

const \_date = new Date(date);

if (Object.prototype.toString.call(\_date) === '[object Date]') {

// it is a date

if (isNaN(\_date.getTime())) {

return false;

} else {

return true;

}

} else {

return false;

}

}

private closeSucessModal() {

this.actionOnModal('successModal', 'close');

}

private showCapacityMessage() {

const modal = document.getElementById('warningCapacityModal');

const link = modal.getElementsByClassName('href-style')[0] as HTMLElement;

link.click();

}

// START MODALS - nri-modal methods

private openEnquiryBoundModal() {

const modal = document.getElementById('enquiryBoundModal');

const link = modal.getElementsByClassName('href-style')[0] as HTMLElement;

link.click();

}

private hideEnquiryBoundModal() {

const modal = document.getElementById('enquiryBoundModal');

const link = modal.getElementsByClassName('close')[0] as HTMLElement;

link.click();

}

private openEnquiryPolicyInForceModal() {

const modal = document.getElementById('enquiryPolicyInForceModal');

const link = modal.getElementsByClassName('href-style')[0] as HTMLElement;

link.click();

}

private hideEnquiryPolicyInForceModal() {

const modal = document.getElementById('enquiryPolicyInForceModal');

const link = modal.getElementsByClassName('close')[0] as HTMLElement;

link.click();

}

private openConfirmationModal() {

this.actionOnModal('confirmationModal', 'href-style');

}

private clearUploadFiles() {

this.fileUpload2.clearUploadFiles();

}

private disableFinishButtons() {

this.savingPolicy = true;

}

private openFailureModalCustom() {

this.actionOnModal('failureModalCustom', 'href-style');

}

private openWarningUploadModal() {

this.actionOnModal('warningUploadModal', 'href-style');

}

private openUpdateDocumentModal() {

this.actionOnModal('updateDocumentModal', 'href-style');

}

private actionOnModal(modalId: string, action: string) {

const modal = document.getElementById(modalId);

const button = modal.getElementsByClassName(action)[0] as HTMLElement;

button.click();

}

private openWarningModal() {

const modal = document.getElementById('warningModal');

const link = modal.getElementsByClassName('href-style')[0] as HTMLElement;

link.click();

}

private closeConfirmationModal() {

this.actionOnModal('confirmationModal', 'close');

}

private closeWarningModal() {

const modal = document.getElementById('warningModal');

const button = modal.getElementsByClassName('close')[0] as HTMLElement;

button.click();

}

private hideDocumentMessage() {

if (document.getElementById('fileReadyNotification') !== null) {

document.getElementById('fileReadyNotification').style.display = 'none';

}

}

private openSuccessModalCustom() {

this.actionOnModal('successModalCustom', 'href-style');

}

private closeWarningUploadModal() {

this.actionOnModal('warningUploadModal', 'close');

}

private openFailureModal() {

this.actionOnModal('failureModal', 'href-style');

}

private openPremiumModal() {

const modal = document.getElementById('premiumModal');

const link = modal.getElementsByClassName('href-style')[0] as HTMLElement;

link.click();

}

private closePremiumModal() {

const modal = document.getElementById('premiumModal');

const button = modal.getElementsByClassName('close')[0] as HTMLElement;

this.deleteGridRow = {name: '', index: 0};

button.click();

}

private closeSuccessModalCustom() {

this.actionOnModal('successModalCustom', 'close');

}

private closeFailureModalCustom() {

this.actionOnModal('failureModalCustom', 'close');

}

private hideCapacityMessage() {

this.actionOnModal('warningCapacityModal', 'close');

}

private closeUpdateDocumentModal() {

this.actionOnModal('updateDocumentModal', 'close');

}

private openSuccessModal() {

const modal = document.getElementById('successModal');

const link = modal.getElementsByClassName('href-style')[0] as HTMLElement;

link.click();

}

private openSuccessModalEnquiry() {

const modal = document.getElementById('successModalEnquiry');

const link = modal.getElementsByClassName('href-style')[0] as HTMLElement;

link.click();

}

private closeFailureModal() {

this.actionOnModal('failureModal', 'close');

}

private closeModal() {

this.initModal.hide();

}

private openChangeBusinessType() {

this.modalChangeBusinessType.open();

}

private closeChangeBusinessType() {

this.modalChangeBusinessType.hide();

}

private getAllCurrencys() {

if (this.localAllCurrenciesStr.length === 0) {

const tempCurrencyOptions = [] as NriInputMultiSelectOptions[];

this.policy.allCurrenciesOptions = [];

this.policy.allCurrenciesIdOptions = [];

this.policy.allCurrenciesStr = [];

this.currencyService.getAllActiveCurrencys()

.then((response) => {

(response as CurrencyDTO[]).forEach((element) => {

tempCurrencyOptions.push(

{

id: element.code,

name: element.name

});

this.policy.allCurrenciesIdOptions.push(

{

value: String(element.id),

text: element.name

});

this.policy.allCurrenciesStr.push(element.code);

this.localAllCurrenciesStr = this.policy.allCurrenciesStr;

this.policy.allCurrenciesOptions = tempCurrencyOptions;

});

if (!!this.policy.currencyID) {

this.setDefaultCurrency();

}

});

}

}

// #region START MODALS - nri-modal methods

// #region private STEP3

private getReoutOrganizationTypes() {

this.policy.ReOutTypeOrg = [];

this.organisationService.getReoutOrganizationTypes()

.then((response) => {

this.policy.reOutOrgDetails = response as any[];

(response as OrganisationTypeDTO[]).forEach((element) => {

this.policy.ReOutTypeOrg.push(

{

value: String(element.organisationTypeID),

text: String(element.organisationType)

});

});

});

}

// #endregion private STEP3

// #region START MODALS - nri-modal methods

private enableFinishButtons() {

this.savingPolicy = false;

}

}