**ANGULAR**

import { CloseScrollStrategy } from '@angular/cdk/overlay';

import { HttpClient } from '@angular/common/http';

import { Component, OnInit } from '@angular/core';

import { FormBuilder, FormControl, FormGroup, Validators } from '@angular/forms';

import { MatDialog } from '@angular/material/dialog';

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

import { DormantAccountService } from '@modules/accounts/services/dormant-account.service';

import { AuthService } from '@modules/auth/services';

import { map } from 'rxjs/operators';

import { Answer, DispositionOption, Quiz, QuizConfig, Script } from '../models';

import { DispositionsService } from '../services/dispositions.service';

import { ScriptService } from '../services/script.service';

@Component({

selector: 'sb-script-question',

templateUrl: './script.component.html',

styleUrls: ['./script.component.scss']

})

export class ScriptComponent implements OnInit {

date = new FormControl(new Date())

is\_callback: any

arrayList: any

hasTextBoxValue: any

callbackdatevalue: any

showcallbackboxes: any

questionCount: any

isPresent: any

serializedDate = new FormControl((new Date()).toISOString())

dispositionForm: any

dispositionReasonForm!: FormGroup

textboxForm!: FormGroup

contactId: any

is\_reachable: any

userCurrentCampaign: any

contactDetails: any = []

hasAnswer!: Answer[]

disposition!: DispositionOption[]

selectedDisposition!: any

dispositions: any = []

dispositionId!: string

nextQuiz: any

currentquiz: any

quizes: any

quiz!: Quiz

mode = 'quiz'

quizName!: string

config: QuizConfig = {

allowBack: true,

allowReview: true,

autoMove: false, // if true, it will move to next question automatically when answered.

duration: 0, // indicates the time in which quiz needs to be completed. 0 means unlimited.

pageSize: 1,

requiredAll: false, // indicates if you must answer all the questions before submitting.

richText: false,

shuffleQuestions: false,

shuffleOptions: false,

showClock: false,

showPager: true,

theme: 'none'

};

pager = {

index: 0,

size: 1,

count: 1

};

constructor(private \_http: HttpClient, protected scriptService: ScriptService,

protected dormantAccountService: DormantAccountService,

protected authService: AuthService,

protected router: Router, protected rout: ActivatedRoute,

private fb: FormBuilder,

protected dispositionService: DispositionsService,

public dialog: MatDialog) {

}

ngOnInit() {

this.getAll()

this.loadQuiz(this.quizName);

this.loadContact()

this.createForm()

this.createForm2()

this.createForm3()

this.getUserCurrentCampaign()

}

openDialog() {

const dialogRef = this.dialog.open(DialogContentComponent);

dialogRef.afterClosed().subscribe(result => {

console.log(`Dialog result: ${result}`);

});

}

createForm() {

this.dispositionForm = this.fb.group({

dispositioncat: [''],

});

}

createForm2() {

this.dispositionReasonForm = this.fb.group({

disposition: [''],

callbackdate: [''],

callbacktime: ['']

});

}

createForm3() {

this.textboxForm = this.fb.group({

text\_box\_answer: ['']

});

}

getUserCurrentCampaign() {

this.authService.getUserProfile('id').subscribe(res => {

this.userCurrentCampaign = res.data.campaign\_id

console.log('Campaign\_Id', this.userCurrentCampaign)

})

}

loadQuiz(quizName: any) {

this.scriptService.getCampaignQuestionList(quizName).subscribe(res => {

this.quiz = res.data

this.pager.count = this.quiz.question.length

this.quizName = res.data.id

});

this.mode = 'quiz';

}

loadContact() {

this.contactId = this.rout.snapshot.paramMap.get('id');

this.dormantAccountService.getContact(this.contactId).subscribe(res => {

this.contactDetails = Array(res.data)

})

}

changeDisposition(id: any) {

this.dispositionService.getdispostions(id).subscribe((res) => {

this.disposition = res.data.disposition

this.is\_callback = res.data.disposition[1].id

if (this.dispositionForm.value.dispositioncat === 'Reachable') {

this.loadQuiz(this.quizName)

this.questionCount = 0

}

if (this.dispositionForm.value.dispositioncat === 'Not Reachable') {

this.loadQuiz('')

this.questionCount = 1

}

console.log('disposi', this.disposition)

});

}

changeDispositionReason(id: string) {

if (this.dispositionReasonForm.value.disposition === this.is\_callback) {

this.showcallbackboxes = 1

this.questionCount = 1

}

else {

this.showcallbackboxes = 0

this.questionCount = 1

}

}

get filteredQuestions() {

return (this.quiz.question) ?

this.quiz.question.slice(this.pager.index, this.pager.index + this.pager.size) : [];

}

}

onSelect(question: Script, option: Answer, index: number) {

const dialogRef = this.dialog.open(DialogContentComponent)

dialogRef.afterClosed().subscribe(result => {

if (result === true) {

const data = {

contact\_id: this.contactId,

question\_id: question.id,

answer\_id: option.id,

text\_box\_answer: this.textboxForm.value.text\_box\_answer

}

this.scriptService.postAnswer(data).subscribe(

res => {

this.nextQuiz = res.data.priority

this.currentquiz = (this.nextQuiz) - 1

this.hasAnswer = res.data.answer

this.isPresent = this.hasAnswer.some(function (el: any) { return el.has\_text\_box === 1 })