**Code to write jasmin and karma unit testing**

**1) facility-booking.data.mock.ts**

import { BookedFacilityDetails } from '@app/models/facility-booking/booked-facility-details.model';

import { FacilityFilterOptions } from '@app/models/facility-booking/facility-filter-optios.model';

export class FacilityBookingDataMock {

public appoinmentId: number;

constructor() { }

public createBookedFacilityDetails(id: number): BookedFacilityDetails {

return {

facilityBookedDates: ["2019-12-10","2019-12-10"],

facilityBookedTimeSlot: ["14:15","14:30","14:45","15:00"]

};

}

public createFacilitySubtypeList(facilityMainType: string): FacilityFilterOptions[] {

if (facilityMainType == "Room") {

return [

{

name: "Meeting Room",

isSelected: false

},

{

name: "Class Room",

isSelected: false

},

{

name: "Conference Room",

isSelected: false

},

{

name: "Auditorium",

isSelected: false

},

{

name: "Workshop",

isSelected: false

},

{

name: "Office Room",

isSelected: false

},

];

}

}

public createFacilityFilterOption(facilityMainType: string): FacilityFilterOptions {

return {

name: 'Meeting Room',

isSelected: false

}

}

public createFacilityBookedTimeSlotArray(): string[] {

let bookedTimeSlot: string[] = ["16:00","16:15","16:30","19:15","19:30"];

return bookedTimeSlot;

}

}

========================================================================

**2) facility-booking.store.service.ts**

import { Injectable } from '@angular/core';

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

import { catchError, map, switchMap, tap } from 'rxjs/operators';

import { Observable, of } from 'rxjs';

import { ErrorService } from '@app/core/error/error.service';

import { DateService } from '@app/core/date/date.service';

import { AvailableFacilityDetails } from '@app/models/facility-booking/available-facility-details.model';

import { BookedFacilityDetails } from '@app/models/facility-booking/booked-facility-details.model';

import { UserService } from '@app/core/authentication/user.service';

import { FacilityFilterOptions } from '@app/models/facility-booking/facility-filter-optios.model';

@Injectable({

providedIn: 'root'

})

export class FacilityBookingStoreService {

private attendeeId: number;

constructor(private http: HttpClient, private errorService: ErrorService, private dateService: DateService, private userService: UserService) {

this.userService.getAttendeeId$().subscribe(attendeeId => this.attendeeId = attendeeId)

}

public getBookedFacilityDateTime(attendeeId: number, selectedDate: Date): Observable<BookedFacilityDetails> {

const date = this.dateService.formatDate(selectedDate);

return this.http

.get<BookedFacilityDetails>(`/api/FacilityBooking/Date/${date}/Attendee/${attendeeId}/`)

.pipe(catchError((err) => this.errorService.handleError(err, 'ophalen', `attendee ${attendeeId}`)));

}

public getAvailableFacilities(): Observable<AvailableFacilityDetails> {

return this.http

.get<AvailableFacilityDetails>(`/api/FacilityBooking/Facility/Attendee/${this.attendeeId}/`)

.pipe(catchError((err) => this.errorService.handleError(err, 'ophalen', `attendee ${this.attendeeId}`)));

}

public getBookedFacilities(attendeeId: number): Observable<AvailableFacilityDetails> {

return this.http.get<AvailableFacilityDetails>(`/api/FacilityBooking/Attendee/${attendeeId}`)

.pipe(catchError((err) => this.errorService.handleError(err, 'ophalen', `attendee ${attendeeId}`)));

}

public getFacilitiesSubTypeList(facilityMainType: string): Observable<FacilityFilterOptions[]> {

return this.http.get<FacilityFilterOptions[]>(`/api/FacilityBooking/FacilitySubTypeList/FacilityMaintype/${facilityMainType}`)

.pipe(catchError((err) => this.errorService.handleError(err, 'ophalen', `Lijst met subtypen van faciliteiten`)));

}

}

========================================================================

**3) facility-booking.store.service.spec.ts**

import { TestBed } from '@angular/core/testing';

import { HttpClientTestingModule, HttpTestingController } from '@angular/common/http/testing';

import { BookedFacilityDetails } from '@app/models/facility-booking/booked-facility-details.model';

import { FacilityFilterOptions } from '@app/models/facility-booking/facility-filter-optios.model';

import { DateService } from '@app/core/date/date.service';

import { ErrorService } from '@app/core/error/error.service';

import { FacilityBookingStoreService } from '@app/store/facility-booking/facility-booking.store.service';

import { FacilityBookingDataMock } from '@mock/data/facility-booking.data.mock';

import { DateServiceMock } from '@mock/core/date.service.mock';

import { ErrorServiceMock } from '@mock/core/error.service.mock';

describe('FacilityBookingStoreService', () => {

let service: FacilityBookingStoreService;

let httpMock: HttpTestingController;

let facilityBookingDataMock: FacilityBookingDataMock;

let dateServiceMock: DateServiceMock;

let errorServiceMock: Partial<ErrorService>;

let bookedFacilityDetails: BookedFacilityDetails;

let facilityFilterOptions: FacilityFilterOptions[];

let formattedDate: string;

const attendeeId = 1;

const error: ErrorEvent = new ErrorEvent('error');

const date = new Date();

const facilityMainType = "Room";

beforeEach(() => {

dateServiceMock = new DateServiceMock();

errorServiceMock = new ErrorServiceMock();

facilityBookingDataMock = new FacilityBookingDataMock();

bookedFacilityDetails = facilityBookingDataMock.createBookedFacilityDetails(1);

facilityFilterOptions = facilityBookingDataMock.createFacilitySubtypeList(facilityMainType);

formattedDate = dateServiceMock.formatDate(date);

TestBed.configureTestingModule({

imports: [

HttpClientTestingModule

],

providers: [

FacilityBookingStoreService,

{ provide: DateService, useValue: dateServiceMock },

{ provide: ErrorService, useValue: errorServiceMock }

]

});

service = TestBed.get(FacilityBookingStoreService);

httpMock = TestBed.get(HttpTestingController);

});

it('should be created', () => {

expect(service).toBeTruthy();

});

it(`should return booked facility deatils`, () => {

service

.getBookedFacilityDateTime(attendeeId, date)

.subscribe(result => expect(result).toEqual(bookedFacilityDetails));

const mockReq = httpMock.expectOne(`/api/FacilityBooking/Date/${formattedDate}/Attendee/${attendeeId}/`);

expect(mockReq.cancelled).toBeFalsy();

expect(mockReq.request.method).toEqual('GET');

expect(mockReq.request.responseType).toEqual('json');

mockReq.flush(bookedFacilityDetails);

httpMock.verify();

});

it(`should handle the error of getBookedFacilityDateTime`, () => {

service.getBookedFacilityDateTime(attendeeId, date)

.subscribe({

error(actualError) {

expect(actualError).not.toBeNull();

expect(actualError).not.toBeUndefined();

expect(errorServiceMock.handleError).toHaveBeenCalled();

}

});

const mockReq = httpMock.expectOne(`/api/FacilityBooking/Date/${formattedDate}/Attendee/${attendeeId}/`);

expect(mockReq.cancelled).toBeFalsy();

expect(mockReq.request.method).toEqual('GET');

expect(mockReq.request.responseType).toEqual('json');

mockReq.error(error);

httpMock.verify();

});

it(`should return facility sub type list`, () => {

service

.getFacilitiesSubTypeList(facilityMainType)

.subscribe(result => expect(result).toEqual(facilityFilterOptions));

const mockReq = httpMock.expectOne(`/api/FacilityBooking/FacilitySubTypeList/FacilityMaintype/${facilityMainType}`);

expect(mockReq.cancelled).toBeFalsy();

expect(mockReq.request.method).toEqual('GET');

expect(mockReq.request.responseType).toEqual('json');

mockReq.flush(facilityFilterOptions);

httpMock.verify();

});

it(`should handle the error of getFacilitiesSubTypeList`, () => {

service.getFacilitiesSubTypeList(facilityMainType)

.subscribe({

error(actualError) {

expect(actualError).not.toBeNull();

expect(actualError).not.toBeUndefined();

expect(errorServiceMock.handleError).toHaveBeenCalled();

}

});

const mockReq = httpMock.expectOne(`/api/FacilityBooking/FacilitySubTypeList/FacilityMaintype/${facilityMainType}`);

expect(mockReq.cancelled).toBeFalsy();

expect(mockReq.request.method).toEqual('GET');

expect(mockReq.request.responseType).toEqual('json');

mockReq.error(error);

httpMock.verify();

});

});

**4) facility-booking.store.service.mock.ts**

import { BookedFacilityDetails } from '@app/models/facility-booking/booked-facility-details.model';

import { FacilityFilterOptions } from '@app/models/facility-booking/facility-filter-optios.model';

import { of } from 'rxjs';

import createSpy = jasmine.createSpy;

export class FacilityBookingStoreServiceMock {

public bookedFacilityReturnValue: BookedFacilityDetails;

public facilityFilterOptionsReturnValue: FacilityFilterOptions[];

public getBookedFacilityDateTime;

public getFacilitiesSubTypeList;

constructor() {

this.getBookedFacilityDateTime = createSpy('getBookedFacilityDateTime').and.callFake(() => of(this.bookedFacilityReturnValue));

this.getFacilitiesSubTypeList = createSpy('getFacilitiesSubTypeList').and.callFake(() => of(this.facilityFilterOptionsReturnValue));

}

}

========================================================================

**5) facility-booking.store.ts**

import { Injectable } from '@angular/core';

import { Observable } from 'rxjs';

import { AvailableFacilityDetails } from '@app/models/facility-booking/available-facility-details.model';

import { FacilityBookingStoreService } from '@app/store/facility-booking/facility-booking.store.service';

import { BookedFacilityDetails } from '@app/models/facility-booking/booked-facility-details.model';

import { UserService } from '@app/core/authentication/user.service';

import { FacilityFilterOptions } from '@app/models/facility-booking/facility-filter-optios.model';

@Injectable({

providedIn: 'root'

})

export class FacilityBookingStore {

private attendeeId: number;

constructor(private facilityBookingStoreService: FacilityBookingStoreService, private userService: UserService) {

this.userService.getAttendeeId$().subscribe(attendeeId => this.attendeeId = attendeeId);

}

public getBookedFacilityDateTime(attendeeId: number, date: Date): Observable<BookedFacilityDetails> {

return this.facilityBookingStoreService.getBookedFacilityDateTime(attendeeId, date);

}

public getAvailableFacilities(): Observable<AvailableFacilityDetails> {

return this.facilityBookingStoreService.getAvailableFacilities();

}

public getBookedFacilities(): Observable<AvailableFacilityDetails> {

return this.facilityBookingStoreService.getBookedFacilities(this.attendeeId);

}

public getFacilitiesSubTypeList(facilityMainType: string): Observable<FacilityFilterOptions[]> {

return this.facilityBookingStoreService.getFacilitiesSubTypeList(facilityMainType);

}

}

========================================================================

**6) facility-booking.store.spec.ts**

import { TestBed } from '@angular/core/testing';

import { FacilityBookingStore } from '@app/store/facility-booking/facility-booking.store';

import { FacilityBookingStoreService } from '@app/store/facility-booking/facility-booking.store.service';

import { FacilityBookingStoreServiceMock } from '@mock/store/facility-booking.store.service.mock';

describe('AppointmentStore', () => {

let service: FacilityBookingStore;

let facilityBookingStoreServiceMock: FacilityBookingStoreServiceMock;

beforeEach(() => {

facilityBookingStoreServiceMock = new FacilityBookingStoreServiceMock();

TestBed.configureTestingModule({

providers: [

FacilityBookingStore,

{ provide: FacilityBookingStoreService, useValue: facilityBookingStoreServiceMock},

]

});

service = TestBed.get(FacilityBookingStore);

});

it('should be created', () => {

expect(service).toBeTruthy();

});

});

========================================================================

**7) facility-booking.store.mock.ts**

import createSpy = jasmine.createSpy;

import { of } from 'rxjs';

import { BookedFacilityDetails } from '@app/models/facility-booking/booked-facility-details.model';

import { FacilityFilterOptions } from '@app/models/facility-booking/facility-filter-optios.model';

export class FacilityBookingStoreMock {

public getBookedFacilityDateTimeReturnValue: BookedFacilityDetails;

public getFacilitiesSubTypeListReturnValue: FacilityFilterOptions[];

public getBookedFacilityDateTime;

public getFacilitiesSubTypeList

constructor() {

this.getBookedFacilityDateTime = createSpy('getBookedFacilityDateTime')

.and.callFake(() => of(this.getBookedFacilityDateTimeReturnValue));

this.getFacilitiesSubTypeList = createSpy('getFacilitiesSubTypeList').and.callFake(() => of(this.getFacilitiesSubTypeListReturnValue));

}

}

========================================================================

**8) facility-booking.service.ts**

import { Injectable, OnDestroy } from '@angular/core';

import { FacilityBookingStore } from '../store/facility-booking/facility-booking.store';

import { Observable, BehaviorSubject } from 'rxjs';

import { BookedFacilityDetails } from '@app/models/facility-booking/booked-facility-details.model';

import { AvailableFacilityDetails } from '@app/models/facility-booking/available-facility-details.model';

import { FacilityFilterOptions } from '@app/models/facility-booking/facility-filter-optios.model';

@Injectable()

export class FacilityBookingService implements OnDestroy {

public selectedFacilityFilter = new BehaviorSubject<string>(null);

public selectedFacilitySubtype = new BehaviorSubject<string>(null);

public selectedWhenText = new BehaviorSubject<string>(null);

constructor(private facilityBookingStore: FacilityBookingStore) {}

public getBookedFacilityDateTime(attendeeId: number, date: Date): Observable<BookedFacilityDetails> {

return this.facilityBookingStore.getBookedFacilityDateTime(attendeeId, date);

}

public getAvailableFacilities(): Observable<AvailableFacilityDetails> {

return this.facilityBookingStore.getAvailableFacilities();

}

public getBookedFacilities(): Observable<AvailableFacilityDetails> {

return this.facilityBookingStore.getBookedFacilities();

}

public getFacilitiesSubTypeList(facilityMainType: string): Observable<FacilityFilterOptions[]> {

return this.facilityBookingStore.getFacilitiesSubTypeList(facilityMainType);

}

public setSelectedFacilityFilter(filterType: string) {

this.selectedFacilityFilter.next(filterType);

}

public getSelectedFacilityFilter() {

return this.selectedFacilityFilter.value;

}

public setSelectedFacilitySubtype(facilitySubType: string) {

this.selectedFacilitySubtype.next(facilitySubType);

}

public getSelectedFacilitySubtype() {

return this.selectedFacilitySubtype.value;

}

public setWhenText(when: string) {

this.selectedWhenText.next(when);

}

public getWhenText() {

return this.selectedWhenText.value;

}

ngOnDestroy() {

this.selectedFacilityFilter.complete();

this.selectedFacilitySubtype.complete();

this.selectedWhenText.complete();

}

}

========================================================================

**9) facility-booking.service.spec.ts**

import { TestBed } from '@angular/core/testing';

import { FacilityBookingService } from '@app/facility-booking/facility-booking.service';

import { FacilityBookingStore } from '@app/store/facility-booking/facility-booking.store';

import { BookedFacilityDetails } from '@app/models/facility-booking/booked-facility-details.model';

import { FacilityFilterOptions } from '@app/models/facility-booking/facility-filter-optios.model';

import { FacilityBookingStoreMock } from '@mock/store/facility-booking.store.mock';

import { FacilityBookingDataMock } from '@mock/data/facility-booking.data.mock';

describe('FacilityBookingService', () => {

let service: FacilityBookingService;

let bookedFacilityDetails: BookedFacilityDetails;

let facilityFilterOptions: FacilityFilterOptions[];

let facilityBookingStoreMock: FacilityBookingStoreMock;

let facilityBookingDataMock: FacilityBookingDataMock;

const attendeeId = 1;

const date = new Date();

const facilityMainType = "Room";

const facilitySubType = "Meeting Room";

const whenText = "Today 12:45";

beforeEach(() => {

facilityBookingStoreMock = new FacilityBookingStoreMock();

facilityBookingDataMock = new FacilityBookingDataMock();

bookedFacilityDetails = facilityBookingDataMock.createBookedFacilityDetails(1);

facilityFilterOptions = facilityBookingDataMock.createFacilitySubtypeList(facilityMainType);

TestBed.configureTestingModule({

providers: [

FacilityBookingService,

{ provide: FacilityBookingStore, useValue: facilityBookingStoreMock },

]

});

service = TestBed.get(FacilityBookingService);

});

it('should be created', () => {

expect(service).toBeTruthy();

});

it('should get booked facility detail model observable', () => {

facilityBookingStoreMock.getBookedFacilityDateTimeReturnValue = bookedFacilityDetails;

service.getBookedFacilityDateTime(attendeeId, date)

.subscribe(result => {

expect(result).toEqual(bookedFacilityDetails);

expect(facilityBookingStoreMock.getBookedFacilityDateTime).toHaveBeenCalledWith(attendeeId, date);

});

});

it('should get facility subtype list', () => {

facilityBookingStoreMock.getFacilitiesSubTypeListReturnValue = facilityFilterOptions;

service.getFacilitiesSubTypeList(facilityMainType)

.subscribe(result => {

expect(result).toEqual(facilityFilterOptions);

expect(facilityBookingStoreMock.getFacilitiesSubTypeList).toHaveBeenCalledWith(facilityMainType);

});

});

it('should set and get selected facility filter', () => {

service.setSelectedFacilityFilter(facilityMainType);

service.getSelectedFacilityFilter();

expect(service.getSelectedFacilityFilter()).toEqual(facilityMainType);

});

it('should set and get selected facility subtype', () => {

service.setSelectedFacilitySubtype(facilitySubType);

service.getSelectedFacilitySubtype();

expect(service.getSelectedFacilitySubtype()).toEqual(facilitySubType);

});

it('should set and get when text', () => {

service.setWhenText(whenText);

service.getWhenText();

expect(service.getWhenText()).toEqual(whenText);

});

});

========================================================================

**10) facility-booking.service.mock.ts**

import createSpy = jasmine.createSpy;

import { of, BehaviorSubject } from 'rxjs';

import { BookedFacilityDetails } from '@app/models/facility-booking/booked-facility-details.model';

import { FacilityFilterOptions } from '@app/models/facility-booking/facility-filter-optios.model';

export class FacilityBookingServiceMock {

public getBookedFacilityDateTimeReturnValue: BookedFacilityDetails;

public getFacilitiesSubTypeListReturnValue: FacilityFilterOptions[];

public getSelectedFacilityFilterReturnValue = "Room";

public getSelectedFacilitySubtypeReturnValue = "Meeting Room";

public getWhenTextReturnValue = "Today 12:45";

public getBookedFacilityDateTime;

public getFacilitiesSubTypeList;

public setSelectedFacilityFilter;

public getSelectedFacilityFilter;

public setSelectedFacilitySubtype;

public getSelectedFacilitySubtype;

public setWhenText;

public getWhenText;

constructor() {

this.getBookedFacilityDateTime = createSpy('getBookedFacilityDateTime')

.and.callFake(() => of(this.getBookedFacilityDateTimeReturnValue));

this.getFacilitiesSubTypeList = createSpy('getFacilitiesSubTypeList')

.and.callFake(() => of(this.getFacilitiesSubTypeListReturnValue));

this.setSelectedFacilityFilter = createSpy('setSelectedFacilityFilter');

this.getSelectedFacilityFilter = createSpy('getSelectedFacilityFilter')

.and.callFake(() => this.getSelectedFacilityFilterReturnValue);

this.setSelectedFacilitySubtype = createSpy('setSelectedFacilitySubtype');

this.getSelectedFacilitySubtype = createSpy('getSelectedFacilitySubtype')

.and.callFake(() => this.getSelectedFacilitySubtypeReturnValue);

this.setWhenText = createSpy('setWhenText');

this.getWhenText = createSpy('getWhenText')

.and.callFake(() => this.getWhenTextReturnValue);

}

}

========================================================================

**11) create-booking.service.ts**

import { Injectable } from '@angular/core';

import { Observable } from 'rxjs';

import { FacilityBookingService } from '@app/facility-booking/facility-booking.service';

import { BookedFacilityDetails } from '@app/models/facility-booking/booked-facility-details.model';

@Injectable({

providedIn: 'root'

})

export class CreateBookingService {

constructor(private facilityBookingService: FacilityBookingService) { }

public getBookedFacilityDateTime(attendeeId: number, date: Date): Observable<BookedFacilityDetails> {

return this.facilityBookingService.getBookedFacilityDateTime(attendeeId,date);

}

}

========================================================================

**12) create-booking.service.spec.ts**

import { TestBed } from '@angular/core/testing';

import { BookedFacilityDetails } from '@app/models/facility-booking/booked-facility-details.model';

import { CreateBookingService } from '@app/facility-booking/create-booking/create-booking.service';

import { FacilityBookingService } from '@app/facility-booking/facility-booking.service';

import { FacilityBookingServiceMock } from '@mock/facility-booking/facility-booking.service.mock';

import { FacilityBookingDataMock } from '@mock/data/facility-booking.data.mock';

describe('CreateBookingService', () => {

let service: CreateBookingService;

let bookedFacilityDetails: BookedFacilityDetails;

let facilityBookingServiceMock: FacilityBookingServiceMock;

let facilityBookingDataMock: FacilityBookingDataMock;

const attendeeId = 1;

const date = new Date();

beforeEach(() => {

facilityBookingServiceMock = new FacilityBookingServiceMock();

facilityBookingDataMock = new FacilityBookingDataMock();

bookedFacilityDetails = facilityBookingDataMock.createBookedFacilityDetails(1);

TestBed.configureTestingModule({

providers: [

CreateBookingService,

{ provide: FacilityBookingService, useValue: facilityBookingServiceMock },

]

});

service = TestBed.get(CreateBookingService);

});

it('should be created', () => {

expect(service).toBeTruthy();

});

it('should get booked facility details observable', () => {

facilityBookingServiceMock.getBookedFacilityDateTimeReturnValue = bookedFacilityDetails;

service.getBookedFacilityDateTime(attendeeId, date)

.subscribe(result => {

expect(result).toEqual(bookedFacilityDetails);

expect(facilityBookingServiceMock.getBookedFacilityDateTime).toHaveBeenCalledWith(attendeeId,date);

});

});

});

========================================================================

**13) create-booking.service.mock.ts**

import createSpy = jasmine.createSpy;

import { of } from 'rxjs';

import { BookedFacilityDetails } from '@app/models/facility-booking/booked-facility-details.model';

export class CreateBookingServiceMock {

public getBookedFacilityDateTimeReturnValue: BookedFacilityDetails;

public getBookedFacilityDateTime;

constructor() {

this.getBookedFacilityDateTime = createSpy('getBookedFacilityDateTime')

.and.callFake(() => of(this.getBookedFacilityDateTimeReturnValue));

}

}

========================================================================

**14) create-booking.component.ts**

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

import { map } from 'rxjs/operators';

import { isNullOrUndefined } from 'util';

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

import { TranslateService } from '@ngx-translate/core';

import { BookedFacilityDetails } from '@app/models/facility-booking/booked-facility-details.model';

import { DateListService } from '@app/core/date-list/date-list.service';

import { CreateBookingService } from '@app/facility-booking/create-booking/create-booking.service';

import { FacilityNavigationService } from '@app/facility-booking/shared/facility-navigation/facility-navigation.service';

import { FacilityBookingService } from '@app/facility-booking/facility-booking.service';

@Component({

selector: 'myx-create-booking',

templateUrl: './create-booking.component.html',

styleUrls: ['./create-booking.component.scss']

})

export class CreateBookingComponent {

public isVisibleCalender = false;

public isVisibleDuration = false;

//public isVisibleBooking = false;

public displaySelectedDate = false;

public showWhenPlaceHolder = true;

public displayToday = false;

public showCalendar = false;

public showNowWhenSection = false;

public setSelectedDateClass = false;

public showFacilityFilter = false;

public showTabPanel = 1;

public isopenSection = 1;

public attendeeId = 0;

public startDate = new Date();

public selectedDate: Date;

public initialDate: Date;

public selectedDay = "";

public selectedTimeSlot = "";

public whenText = "";

public todayText = "";

public nowText = "";

public firstAvailableTimeSlot = "";

public selectedFilterBy = "";

public selectedFilterFilterParameter = "";

public timeSlots: any[] = [];

public tempTimeSlots: any[] = [];

public facilityBookedTimeSlots: string[] = [];

public facilityBookedDates: string[] = [];

public bookedFacilityDetails: BookedFacilityDetails;

constructor(private dateListService: DateListService,

private createBookingService: CreateBookingService,

private translateService: TranslateService,

private facilityNavigationService: FacilityNavigationService,

private route: ActivatedRoute,

private facilityBookingService: FacilityBookingService

) {

}

ngOnInit() {

this.getFilterSearchParameter();

this.setSelectLabel();

this.loadWhenText();

//bind the temparary time slot array from 00.00 to 23.45

var quarterHours = ["00", "15", "30", "45"];

for (var i = 0; i < 24; i++) {

for (var j = 0; j < 4; j++) {

this.tempTimeSlots.push(i.toString() + ":" + quarterHours[j]);

}

}

}

public getFilterSearchParameter() {

this.selectedFilterBy = this.facilityBookingService.getSelectedFacilityFilter();

this.selectedFilterFilterParameter = this.facilityBookingService.getSelectedFacilitySubtype();

}

public loadWhenText() {

this.whenText = this.facilityBookingService.getWhenText();

if (isNullOrUndefined(this.whenText)) {

this.showWhenPlaceHolder = true;

}

else {

this.showWhenPlaceHolder = false;

}

}

public setSelectLabel() {

this.nowText = this.translateService.instant('BOOKING.CREATE\_BOOKING.NOW\_TEXT');

this.todayText = this.translateService.instant('BOOKING.CREATE\_BOOKING.TODAY\_TEXT');

}

public showCalender() {

this.isVisibleCalender = true;

this.displaySelectedDate = false;

this.whenText = "";

this.selectedDate = null;

this.selectedDay = "";

this.selectedTimeSlot = "";

this.timeSlots = new Array();

this.bindTimeSlot(new Date());

}

public hideCalender() {

this.isVisibleCalender = false;

this.facilityBookedTimeSlots = new Array();

this.attendeeId = 0;

if (this.whenText == "" && this.selectedTimeSlot == "") {

this.showWhenPlaceHolder = true;

this.showNowWhenSection = false;

}

}

public showDuration() {

this.isVisibleDuration = true;

this.facilityNavigationService.goToDuration(this.route);

}

public hideDuration() {

this.isVisibleDuration = false;

}

public closeNewBooking() {

//this.isVisibleBooking = false;

this.showWhenPlaceHolder = true;

this.showNowWhenSection = false;

//this.hideBookingForm.emit(false);

}

public setSelectedDate(selectedDate: Date) {

this.setSelectedDateClass = false;

this.timeSlots = new Array();

this.facilityBookedTimeSlots = new Array();

this.selectedTimeSlot = "";

this.displaySelectedDate = true;

this.displayToday = true;

this.dateListService.setSelectedDate(selectedDate);

this.initialDate = selectedDate;

this.selectedDate = selectedDate;

if (selectedDate.getDay() == new Date().getDay()) {

this.selectedDay = this.todayText;

}

else {

this.selectedDay = "";

}

this.bindTimeSlot(selectedDate);

if (this.attendeeId != 0) {

this.getBookedFacilityDateTime(this.attendeeId, selectedDate);

}

}

public bindTimeSlot(date: Date) {

const currentDate = new Date();

let calculatedMinutes = this.getCalculatedMinutes(currentDate.getMinutes());

var currentTimeSlot = currentDate.getHours() + ":" + (calculatedMinutes !== 0 ? calculatedMinutes : "00");

const indexOfTimeSlot = this.tempTimeSlots.indexOf(currentTimeSlot);

for (var i = indexOfTimeSlot; i < this.tempTimeSlots.length; i++) {

this.timeSlots.push(this.tempTimeSlots[i]);

}

if (date > new Date()) {

for (var i = 0; i < indexOfTimeSlot; i++) {

this.timeSlots.push(this.tempTimeSlots[i]);

}

}

this.firstAvailableTimeSlot = this.timeSlots[0];

}

getCalculatedMinutes(actualMinutes: number) {

if (actualMinutes < 15) {

return 0;

}

else if (actualMinutes < 30) {

return 15;

}

else if (actualMinutes < 45) {

return 30;

}

else if (actualMinutes < 60) {

return 45;

}

}

public timeSlotClicked(selectedTimeSlot: any) {

var todaysDate = new Date();

this.displayToday = false;

this.showWhenPlaceHolder = false;

if (this.firstAvailableTimeSlot == selectedTimeSlot) {

this.selectedTimeSlot = selectedTimeSlot + " " + this.nowText;

this.showNowWhenSection = true;

}

else {

this.selectedTimeSlot = selectedTimeSlot;

this.showNowWhenSection = false;

}

if (this.selectedDate == null) {

this.selectedDate = todaysDate;

this.selectedDay = this.todayText;

}

if (this.selectedDay == "" && this.selectedDate != todaysDate) {

var month = this.selectedDate.getMonth() + 1;

var fullDate = this.selectedDate.getDate() + "/" + month + "/" + this.selectedDate.getFullYear();

this.whenText = fullDate + " " + selectedTimeSlot;

}

else {

this.whenText = this.selectedDay + " " + selectedTimeSlot;

}

this.facilityBookingService.setWhenText(this.whenText);

this.hideCalender();

}

public timeSlotClass(timeSlot: any) {

if (this.facilityBookedTimeSlots != null) {

if (this.facilityBookedTimeSlots.indexOf(timeSlot) > -1) {

return "is-disable-list-item";

}

else {

return "";

}

}

else {

return "";

}

}

attendeeEmitter(data: any) {

this.setSelectedDateClass = true;

this.attendeeId = data;

this.getBookedFacilityDateTime(this.attendeeId, this.startDate);

}

public getBookedFacilityDateTime(attendeeId: number, date: Date) {

this.createBookingService

.getBookedFacilityDateTime(attendeeId, date)

.pipe(map(response => this.bookedFacilityDetails = response))

.subscribe((data) => {

if (data != null && data.facilityBookedTimeSlot != null) {

this.facilityBookedTimeSlots = data.facilityBookedTimeSlot;

}

});

}

public filterFacility() {

this.facilityBookingService.setSelectedFacilityFilter("");

this.facilityBookingService.setSelectedFacilitySubtype("");

this.facilityNavigationService.goToFacilityFilter(this.route);

}

public clearWhenSection() {

this.whenText = " ";

this.nowText = " ";

this.showWhenPlaceHolder = true;

this.showNowWhenSection = false;

}

public clearFilterParam() {

this.selectedFilterFilterParameter = null;

}

}

**15) create-booking.component.spec.ts**

import **{ ComponentFixture, TestBed } from '@angular/core/testing';**

import { NO\_ERRORS\_SCHEMA } from '@angular/core';

import { TranslateService, TranslateModule } from '@ngx-translate/core';

import { CreateBookingComponent } from '@app/facility-booking/create-booking/create-booking.component';

import { BookedFacilityDetails } from '@app/models/facility-booking/booked-facility-details.model';

import { DateListService } from '@app/core/date-list/date-list.service';

import { CreateBookingService } from '@app/facility-booking/create-booking/create-booking.service';

//import { TranslateService, TranslateModule } from '@ngx-translate/core';

import { FacilityNavigationService } from '../shared/facility-navigation/facility-navigation.service';

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

import { FacilityBookingService } from '../facility-booking.service';

import { DateServiceMock } from '@mock/core/date.service.mock';

import { CreateBookingServiceMock } from '@mock/facility-booking/create-booking.service.mock';

import { FacilityBookingDataMock } from '@mock/data/facility-booking.data.mock';

import { TranslateServiceMock } from '@mock/angular/translate.service.mock';

import { FacilityNavigationServiceMock } from '@mock/facility-booking/facility-navigation.service.mock';

import { ActivatedRouteMock } from '@mock/angular/activated-route.mock';

import { FacilityBookingServiceMock } from '@mock/facility-booking/facility-booking.service.mock';

import { DateListServiceMock } from '@mock/core/date-list.service.mock';

import { CoachModule } from '@app/coach/coach.module';

describe('FacilityBookingFilterComponent', () => {

let component: CreateBookingComponent;

let fixture: ComponentFixture<CreateBookingComponent>;

let bookedFacilityDetails: BookedFacilityDetails;

let facilityBookedTimeSlot: string[];

let activatedRouteMock: ActivatedRouteMock;

let dateListServiceMock: DateListServiceMock;

let createBookingServiceMock: CreateBookingServiceMock;

let facilityBookingServiceMock: FacilityBookingServiceMock;

let translateServiceMock: TranslateServiceMock;

let facilityBookingDataMock: FacilityBookingDataMock;

let facilityNavigationServiceMock: FacilityNavigationServiceMock;

let date: Date = new Date();

const selectedFilterBy = "Room";

const selectedFilterFilterParameter = "Meeting Room";

const todayText = "BOOKING.CREATE\_BOOKING.TODAY\_TEXT";

const nowText = "BOOKING.CREATE\_BOOKING.NOW\_TEXT";

const whenText = "BOOKING.CREATE\_BOOKING.TODAY\_TEXT 16:30";

const timeSlot = "16:30";

const attendeeId = 1;

beforeEach(() => {

dateListServiceMock = new DateListServiceMock();

createBookingServiceMock = new CreateBookingServiceMock();

translateServiceMock = new TranslateServiceMock();

facilityNavigationServiceMock = new FacilityNavigationServiceMock();

facilityBookingServiceMock = new FacilityBookingServiceMock();

activatedRouteMock = new ActivatedRouteMock();

facilityBookingDataMock = new FacilityBookingDataMock();

bookedFacilityDetails = facilityBookingDataMock.createBookedFacilityDetails(attendeeId);

facilityBookedTimeSlot = facilityBookingDataMock.createFacilityBookedTimeSlotArray();

TestBed.configureTestingModule({

declarations: [

CreateBookingComponent

],

imports: [TranslateModule.forRoot()],

providers: [

{ provide: DateListService, useValue: dateListServiceMock },

{ provide: CreateBookingService, useValue: createBookingServiceMock },

{ provide: FacilityNavigationService, useValue: facilityNavigationServiceMock },

{ provide: FacilityBookingService, useValue: facilityBookingServiceMock },

{ provide: ActivatedRoute, useValue: activatedRouteMock },

],

schemas: [NO\_ERRORS\_SCHEMA]

});

fixture = TestBed.createComponent(CreateBookingComponent);

component = fixture.componentInstance;

});

it('should create', () => {

fixture.detectChanges();

expect(component).toBeTruthy();

});

it(`should get the filter search parameter`, () => {

translateServiceMock.getReturnValue = 'Now';

fixture.detectChanges();

expect(facilityBookingServiceMock.getSelectedFacilityFilter).toHaveBeenCalled();

expect(component.selectedFilterBy).toEqual(selectedFilterBy);

expect(component.selectedFilterFilterParameter).toEqual(selectedFilterFilterParameter);

});

it(`should load when text`, () => {

const whenText = "Today 12:45";

fixture.detectChanges();

expect(facilityBookingServiceMock.getWhenText).toHaveBeenCalled();

expect(component.whenText).toEqual(whenText);

expect(component.showWhenPlaceHolder).toEqual(false);

});

it(`should show the calendar`, () => {

fixture.detectChanges();

component.showCalender();

expect(component.isVisibleCalender).toEqual(true);

expect(component.displaySelectedDate).toEqual(false);

expect(component.whenText).toEqual("");

expect(component.selectedDate).toEqual(null);

expect(component.selectedDay).toEqual("");

expect(component.selectedTimeSlot).toEqual("");

});

it(`should get the time slot array`, () => {

spyOn(component, 'bindTimeSlot');

component.showCalender();

expect(component.bindTimeSlot).toHaveBeenCalled();

});

it(`should hide the calendar`, () => {

fixture.detectChanges();

component.hideCalender();

expect(component.isVisibleCalender).toEqual(false);

expect(component.facilityBookedTimeSlots).toEqual(new Array());

});

it(`should show the duration`, () => {

fixture.detectChanges();

component.showDuration();

expect(facilityNavigationServiceMock.goToDuration).toHaveBeenCalled();

expect(component.isVisibleDuration).toEqual(true);

});

it(`should hide the duration`, () => {

fixture.detectChanges();

component.hideDuration();

expect(component.isVisibleDuration).toEqual(false);

});

it(`should close new booking form`, () => {

fixture.detectChanges();

component.closeNewBooking();

expect(component.showWhenPlaceHolder).toEqual(true);

expect(component.showNowWhenSection).toEqual(false);

});

it(`should set the selected date`, () => {

fixture.detectChanges();

const date = new Date();

component.setSelectedDate(date);

expect(component.setSelectedDateClass).toEqual(false);

expect(component.selectedTimeSlot).toEqual("");

expect(component.displaySelectedDate).toEqual(true);

expect(component.displayToday).toEqual(true);

expect(dateListServiceMock.setSelectedDate).toHaveBeenCalledWith(date);

expect(component.initialDate).toEqual(date);

expect(component.selectedDate).toEqual(date);

expect(component.selectedDay).toEqual(todayText);

});

it(`should get facility booked date and time`, () => {

createBookingServiceMock.getBookedFacilityDateTimeReturnValue = bookedFacilityDetails;

component.attendeeId = attendeeId;

spyOn(component, 'getBookedFacilityDateTime');

component.setSelectedDate(date);

expect(component.getBookedFacilityDateTime).toHaveBeenCalledWith(attendeeId, date);

});

it(`should get timeSlot array`, () => {

fixture.detectChanges();

component.bindTimeSlot(date);

expect(component.timeSlots.length).toBeGreaterThan(0);

expect(component.timeSlots).not.toBeNull();

expect(component.firstAvailableTimeSlot).not.toEqual("");

});

it(`should return calculated minutes`, () => {

spyOn(component, 'getCalculatedMinutes');

fixture.detectChanges();

component.bindTimeSlot(date);

expect(component.getCalculatedMinutes).toHaveBeenCalledWith(date.getMinutes());

});

it(`should call timeSlotClicked when click on time slot`, () => {

fixture.detectChanges();

const date = new Date();

component.firstAvailableTimeSlot = "16:30";

component.selectedDate = null;

component.timeSlotClicked(timeSlot);

expect(component.displayToday).toEqual(false);

expect(component.showWhenPlaceHolder).toEqual(false);

expect(component.selectedTimeSlot).toEqual(timeSlot + " " + nowText);

expect(component.selectedDate).toEqual(date);

expect(component.selectedDay).toEqual(todayText);

expect(component.whenText).toEqual(todayText + " " + timeSlot);

expect(facilityBookingServiceMock.setWhenText).toHaveBeenCalledWith(whenText);

});

it(`should call timeSlotClass and return css class`, () => {

const cssClass = "is-disable-list-item";

fixture.detectChanges();

component.facilityBookedTimeSlots = facilityBookedTimeSlot;

const returnValue = component.timeSlotClass(timeSlot);

expect(cssClass).toEqual(returnValue);

});

it(`should call attendeeEmitter`, () => {

fixture.detectChanges();

component.attendeeEmitter(attendeeId);

expect(component.setSelectedDateClass).toEqual(true);

expect(component.attendeeId).toEqual(attendeeId);

});

it(`should call getBookedFacilityDateTime & return bookedFacilityDetails`, () => {

fixture.detectChanges();

createBookingServiceMock.getBookedFacilityDateTimeReturnValue = bookedFacilityDetails;

component.getBookedFacilityDateTime(attendeeId, date);

expect(createBookingServiceMock.getBookedFacilityDateTime).toHaveBeenCalledWith(attendeeId, date);

expect(component.facilityBookedTimeSlots).toEqual(bookedFacilityDetails.facilityBookedTimeSlot);

});

it(`should navigate to facility filter`, () => {

fixture.detectChanges();

component.filterFacility();

expect(facilityBookingServiceMock.setSelectedFacilityFilter).toHaveBeenCalledWith("");

expect(facilityBookingServiceMock.setSelectedFacilitySubtype).toHaveBeenCalledWith("");

expect(facilityNavigationServiceMock.goToFacilityFilter).toHaveBeenCalledWith(activatedRouteMock);

});

});

=======================================================================

**16) facility-booking-filter.service.ts**

import { Injectable, OnDestroy } from '@angular/core';

import { Observable } from 'rxjs';

import { FacilityBookingService } from '@app/facility-booking/facility-booking.service';

import { FacilityFilterOptions } from '@app/models/facility-booking/facility-filter-optios.model';

@Injectable()

export class FacilityBookingFilterService {

constructor(private facilityBookingService: FacilityBookingService) { }

public getFacilitiesSubTypeList(facilityMainType: string): Observable<FacilityFilterOptions[]> {

return this.facilityBookingService.getFacilitiesSubTypeList(facilityMainType);

}

}

========================================================================

**17) facility-booking-filter.service.spec.ts**

**import { TestBed } from '@angular/core/testing';**

import { FacilityFilterOptions } from '@app/models/facility-booking/facility-filter-optios.model';

import { FacilityBookingService } from '@app/facility-booking/facility-booking.service';

import { FacilityBookingFilterService } from '@app/facility-booking/shared/facility-filter/Facility-booking-filter.service';

import { FacilityBookingServiceMock } from '@mock/facility-booking/facility-booking.service.mock';

import { FacilityBookingDataMock } from '@mock/data/facility-booking.data.mock';

describe('FacilityBookingFilterService', () => {

let service: FacilityBookingFilterService;

let facilityBookingServiceMock: FacilityBookingServiceMock;

let facilityBookingDataMock: FacilityBookingDataMock;

let facilityFilterOptions: FacilityFilterOptions[];

const facilityMainType = "Room";

beforeEach(() => {

facilityBookingServiceMock = new FacilityBookingServiceMock();

facilityBookingDataMock = new FacilityBookingDataMock();

facilityFilterOptions = facilityBookingDataMock.createFacilitySubtypeList(facilityMainType);

TestBed.configureTestingModule({

providers: [

FacilityBookingFilterService,

{ provide: FacilityBookingService, useValue: facilityBookingServiceMock },

]

});

service = TestBed.get(FacilityBookingFilterService);

});

it('should be created', () => {

expect(service).toBeTruthy();

});

it('should get facility subtype list', () => {

facilityBookingServiceMock.getFacilitiesSubTypeListReturnValue = facilityFilterOptions;

service.getFacilitiesSubTypeList(facilityMainType)

.subscribe(result => {

expect(result).toEqual(facilityFilterOptions);

expect(facilityBookingServiceMock.getFacilitiesSubTypeList).toHaveBeenCalledWith(facilityMainType);

});

});

});

========================================================================

**18) facility-booking-filter.service.mock.ts**

**import createSpy = jasmine.createSpy;**

import { of, BehaviorSubject } from 'rxjs';

import { FacilityFilterOptions } from '@app/models/facility-booking/facility-filter-optios.model';

export class FacilityBookingFilterServiceMock {

public getFacilitiesSubTypeListReturnValue: FacilityFilterOptions[];

public getFacilitiesSubTypeList

constructor() {

this.getFacilitiesSubTypeList = createSpy('getFacilitiesSubTypeList')

.and.callFake(() => of(this.getFacilitiesSubTypeListReturnValue));

}

}

**19) facility-subtype-list.component.ts**

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

import { map } from 'rxjs/operators';

import { isNullOrUndefined } from 'util';

import { FacilityFilterOptions } from '@app/models/facility-booking/facility-filter-optios.model';

import { FacilityBookingFilterService } from '@app/facility-booking/shared/facility-filter/Facility-booking-filter.service';

import { FacilityBookingService } from '@app/facility-booking/facility-booking.service';

import { TranslateService } from '@ngx-translate/core';

@Component({

selector: 'myx-facility-subtype-list',

templateUrl: './facility-subtype-list.component.html',

styleUrls: ['./facility-subtype-list.component.scss']

})

export class FacilitySubtypeListComponent implements OnInit {

@Output() openFilterBy = new EventEmitter<boolean>();

public facilitySubTypeList: FacilityFilterOptions[];

public seletcedFacilitySubtype = "";

public selectedFilterBy = "";

public prevSelectedFacility: any;

constructor(private facilityBookingFilterService: FacilityBookingFilterService, private facilityBookingService: FacilityBookingService, private translateService: TranslateService) {

this.selectedFilterBy = this.facilityBookingService.getSelectedFacilityFilter();

}

ngOnInit() {

this.getFacilitySubTypeList(this.selectedFilterBy);

}

public getFacilitySubTypeList(filterBy: string) {

this.facilityBookingFilterService

.getFacilitiesSubTypeList(filterBy)

.pipe(map(response => this.facilitySubTypeList = response))

.subscribe((data) => {

if (data != null) {

this.facilitySubTypeList = data;

}

});

}

public showSelectedSubmenuList(submenu: any) {

this.facilityBookingService.setSelectedFacilitySubtype(submenu.name);

this.seletcedFacilitySubtype = submenu.name;

submenu.isSelected = true;

if (isNullOrUndefined(this.prevSelectedFacility))

this.prevSelectedFacility = submenu;

if (this.prevSelectedFacility != submenu) {

this.prevSelectedFacility.isSelected = false;

this.prevSelectedFacility = submenu;

}

}

public backToFilterBy() {

this.openFilterBy.emit(true);

}

}

========================================================================

**20) facility-subtype-list.component.spec.ts**

import { async, ComponentFixture, TestBed } from '@angular/core/testing';

import { NO\_ERRORS\_SCHEMA } from '@angular/core';

import { FacilitySubtypeListComponent } from './facility-subtype-list.component';

import { TranslateService, TranslateModule } from '@ngx-translate/core';

import { FacilityBookingFilterService } from '@app/facility-booking/shared/facility-filter/Facility-booking-filter.service';

import { FacilityBookingService } from '@app/facility-booking/facility-booking.service';

import { FacilityFilterOptions } from '@app/models/facility-booking/facility-filter-optios.model';

import { FacilityBookingFilterServiceMock } from '@mock/facility-booking/facility-booking-filter.service.mock';

import { FacilityBookingDataMock } from '@mock/data/facility-booking.data.mock';

import { TranslateServiceMock } from '@mock/angular/translate.service.mock';

import { FacilityBookingServiceMock } from '@mock/facility-booking/facility-booking.service.mock';

import { CoachModule } from '@app/coach/coach.module';

describe('FacilitySubtypeListComponent', () => {

let component: FacilitySubtypeListComponent;

let fixture: ComponentFixture<FacilitySubtypeListComponent>;

let facilityFilterOptions: FacilityFilterOptions[];

let subMenu: FacilityFilterOptions;

let facilityBookingFilterServiceMock: FacilityBookingFilterServiceMock;

let facilityBookingServiceMock: FacilityBookingServiceMock;

let translateServiceMock: TranslateServiceMock;

let facilityBookingDataMock: FacilityBookingDataMock;

const facilityMainType = "Room";

const facilitySelectedSubType = "Meeting Room";

beforeEach(() => {

facilityBookingFilterServiceMock = new FacilityBookingFilterServiceMock();

translateServiceMock = new TranslateServiceMock();

facilityBookingDataMock = new FacilityBookingDataMock();

facilityBookingServiceMock = new FacilityBookingServiceMock();

facilityFilterOptions = facilityBookingDataMock.createFacilitySubtypeList(facilityMainType);

subMenu = facilityBookingDataMock.createFacilityFilterOption(facilityMainType);

TestBed.configureTestingModule({

declarations: [

FacilitySubtypeListComponent

],

imports: [TranslateModule.forRoot()],

providers: [

{ provide: FacilityBookingFilterService, useValue: facilityBookingFilterServiceMock },

{ provide: FacilityBookingService, useValue: facilityBookingServiceMock },

],

schemas: [NO\_ERRORS\_SCHEMA]

});

fixture = TestBed.createComponent(FacilitySubtypeListComponent);

component = fixture.componentInstance;

});

fit('should create', () => {

fixture.detectChanges();

expect(component).toBeTruthy();

});

fit(`should get the facility subtype list`, () => {

spyOn(component, 'getFacilitySubTypeList');

component.ngOnInit();

expect(component.getFacilitySubTypeList).toHaveBeenCalledWith(facilityMainType);

});

fit(`should set the facility subtype list`, () => {

fixture.detectChanges();

facilityBookingFilterServiceMock.getFacilitiesSubTypeListReturnValue = facilityFilterOptions;

component.getFacilitySubTypeList(facilityMainType);

expect(facilityBookingFilterServiceMock.getFacilitiesSubTypeList).toHaveBeenCalledWith(facilityMainType);

expect(component.facilitySubTypeList).toEqual(facilityFilterOptions);

});

fit(`should set selected facility sub type`, () => {

fixture.detectChanges();

component.showSelectedSubmenuList(subMenu);

facilityBookingServiceMock.getSelectedFacilitySubtypeReturnValue = facilitySelectedSubType;

expect(facilityBookingServiceMock.setSelectedFacilitySubtype).toHaveBeenCalledWith(subMenu.name);

expect(component.seletcedFacilitySubtype).toEqual(subMenu.name);

expect(subMenu.isSelected).toEqual(true);

});

fit(`should open filter by view`, () => {

spyOn(component.openFilterBy, 'emit');

component.backToFilterBy();

expect(component.openFilterBy.emit).toHaveBeenCalledWith(true);

});

});