Korištenje Constant Servisa

U svrhu izbjegavanja hardkodiranja preporuka je koristiti kodiranje kao na primjeru **PckConstantsService**

```
export class PckConstantsService {
public readonly productStatus = {
      offerStatus: {
         OFFER_NEW: "OFFER_NEW",
         OFFER_IN_PROGRESS: "OFFER_IN_PROGRESS",
         OFFER_CANCELED: "OFFER_CANCELED",
         OFFER_RESOLVED: "OFFER_RESOLVED",
         OFFER_BOOKED: "OFFER_BOOKED",
         OFFER_INVALID: "OFFER_INVALID",
         OFFER_REBOOKING: "OFFER_REBOOKING",
         OFFER_CONFIRMED: "OFFER_CONFIRMED",
         OFFER_VERIFIED: "OFFER_VERIFIED"
      },
 ...
export enum CapacityTypes {
 CAPACITY = 0x01,
 RELEASE = 0x02,
 MINSTAY = 0x03,
}
```

.ts

```
    switch (context) {
        case BcStatusContext.Product:
          return offerStatusLookup.filter(x =>
            showOnlyCancelled ?
              x.mnemonic === this._c.productStatus.offerStatus.OFFER_CANCELED : exclude ?
                x.mnemonic !== this._c.productStatus.offerStatus.OFFER_CANCELED : true);
```

.html

```
      <div class="row">
        <div class="col-lg-12 mb-3">
          <capacity-product-group #productGroup [CapacityTypeId]="CapacityTypes.CAPACITY" [CapacityStateParams]="state"
[NumberOfShownLists]="3" [IsQuickPreview]="false"></capacity-product-group>
        </div>
      </div>
```

takeUntil()

```
export class SpecialOffersSearchComponent implements OnInit {
 ...
 destroy$: Subject<boolean> = new Subject<boolean>();
 ...
 private LoadLookups(productGroupType: string) {
   this.specialOfferService
    .getLookups(productGroupType, null)
    .pipe(
     takeUntil(this.destroy$))  // <- unsubscribe
    .subscribe(l => {
     this.lookups = l;
     this._promotions = l.promotions;
    });
 }
...
```

ili

```
export class AppDocumentComponent extends AppBaseComponent implements OnInit {
...
    this.router.queryParams
      .pipe(
        takeUntil(this.destroyed$))  // <- unsubscribe
      .subscribe(params => {
        if (params) {
          const type = this.router.snapshot.params["type"];
          const qs = Object
            .keys(params)
            .map(function(key) {
              return `${key}=${params[key]}`;
            })
            .join('&');

          this.documentUrl = this.sanitizer.bypassSecurityTrustResourceUrl();
        }
        // else 404, auto handled!
      });
```

take(x)

```
export class EmployeeAdditionalInfoComponent extends AppBaseComponent {
 ...
  constructor (public dataService: EmployeeDataStoreService) {
    super();
 ...
    this.dataService.onDataLoadComplete
      .pipe(take(1)) // <- unsubscribe
      .subscribe(data => this.onDataLoadComplete(data));
  }

  private onDataLoadComplete(data: EmployeeEditModel) {
    if (!data) {
      return null;
    }

    this.dataService.lookup<IdNameMnemonicModel[]>("employmentTypes")
      .pipe(take(1)) // <- unsubscribe
      .subscribe(employmentTypes => {
        const employmentType = employmentTypes.find(et => et.id === data.basicInfo.employmentTypeId);

        if (employmentType) {
          this.employmentTypeMnemonic = employmentType.mnemonic;
        }
      });
  }
 ...
```

forkJoin()

```
const businessLookup = this.businessLookupService.Get<Array<IdNameMnemonicModel>>([AppBusinessLookupNames.businessYears]);
const supplierLookup = this.supplierLookupService.Get<Array<IdNameMnemonicModel>>([SupLookupNames.legalCapacities]);

forkJoin(businessLookup, supplierLookup)
    .pipe(takeUntil(this.destroyed$))  // <- unsubscribe
    .subscribe(lookups => {
      this.businessYears = lookups[0][AppBusinessLookupNames.businessYears];
      this.legalCapacities = lookups[1][SupLookupNames.legalCapacities];

      if (this.businessYears && this.businessYears.length > 0) {
        this.businessYearId = this.businessYears[0].id;
      }

      if (this.legalCapacities && this.legalCapacities.length > 0) {
        this.legalCapacityId = this.legalCapacities[0].id;
      }
});
```

ili

```
  public getPgAttrLookups(): Observable<LookupsPgAttLookupModel> {
    return forkJoin(
      this.pcLookupSrvc.Get([PcLookupNames.productGroupSubtypes]),
      this.commonLookupSrvc.Get([AppCommonLookupNames.languages])
    ).pipe(
      map((t: Array<any>) => {
        const tmp: LookupsPgAttLookupModel = new LookupsPgAttLookupModel();
        t.forEach(x => {
          AppUtils.mergeObjectValues(x, tmp);
        });
        return tmp;
      }));
  }

export class LookupsPgAttResolver implements Resolve<boolean> {
  constructor (
    private srvc: LookupsService,
    private storage: LookupsPgAttrStorageService,
    private router: Router,
  ) { }

  resolve(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): Observable<boolean> {
    return this.srvc.getPgAttrLookups().pipe(
      take(1), // <- unsubscribe
      map(data => {
        this.storage.lookup = data;
        return true;
      }),
      catchError(err => {
        this.router.navigate(['../']);
        return of(null);
      })) as Observable<boolean>;
  }
}
```

filter()

```
export class SpecialOffersPricingSubTypesPipe implements PipeTransform {
   transform(input: Array<{ id: number, name: string, mnemonic: string, serviceTypeId: number }>,
      serviceTypeId: number
   ): Array<{ id: number, name: string, mnemonic: string, serviceTypeId: number }> {
      // avoid filter on null object or equality parameter
      if (!input || !serviceTypeId) { return input; }
      return input.filter(x => x.serviceTypeId === serviceTypeId);
   }
}
```

ili

```
private setChildren(permissions: Array<PermissionModel>, permission: PermissionModel) {
      const children = permissions.filter((p: PermissionModel) => p.parentId === permission.menuId);

      // avoid forEach on null or empty object
      if (children && children.length > 0) {
         children.forEach((p: PermissionModel) => {
            ...
         });
      }
   }
```

sort()

```
const itinerary = this.PackageModel.packageVersion.itineraryItems
                              .filter(it => this._itemListTemplate.findIndex(il => il.itineraryItemId === it.id) > -1)
                              .sort((a, b) => a.sequence >= b.sequence ? a.sequence === b.sequence ? 0 : 1 : -1);
```

Object.assign()

```
{} <- single object    [] <- array
 public selectGeneralPaymentTermInvoicings = () => {
      this.editInvoicePaymentModel.invoicing = Object.assign({}, ...[this.editGeneralPaymentTermInvoicings]
         .map(egpi => {
            const gpti = this.generalPaymentTermInvoicings.find(s => s.id === this.selectedGeneralPaymentTermInvoicingsId);
            return new Cb2bInvoiceRenderingModel(
                  ...
            );
         }));
 }
```

ili

```
public edit(x: BcPaymentWorkflowModel): void {
      // avoid null object
      if (x) {
         if (!this.editingPaymentWorkflow) {
            this.editingPaymentWorkflow = new BcPaymentWorkflowModel();
         }
         Object.assign(this.editingPaymentWorkflow, ...[x]);

         if (!this.editingPaymentWorkflow.guid) {
            this.editingPaymentWorkflow.guid = this.editingPaymentWorkflow.id ? this.guidService.Guid() : null;
         }
         this.popupVisible = true;
      }
}
```

mergeMap(), groupBy() i reduce()

```
  private _initRenderingModel = (model: Array<PlGridDataRenderingModel>) => {
    ...
    // gets unique keys
    model = model.map((m, i) => {
      m.key = i.toString();
      return m;
    });

    from(Object
      .keys(model)
      .map(key => ({ key, value: model[key] }))
    )
    .pipe(
        groupBy(gb => (AppUtils.getStringHashCode(gb.value.productGroupName) +
          gb.value.productGroupId +
          gb.value.partnerId +
          AppUtils.getStringHashCode(gb.value.partnerName)), gv => gv),
        // return each item in group as array
        mergeMap(g => g.pipe(toArray()))
      )
      .subscribe(s => {
        ...
        // returns only unique objects
        const arrDoubleProducts = s.map(m => m.value.productId)
          .reduce((arrProd, prodId) => {
            if (prodId in arrProd) {
              arrProd[prodId] = true; // found duplicate
            } else {
              arrProd[prodId] = false; // non duplicate
            }
            return arrProd;
          }, {});
        ...
    }, error => {
      console.error(error);
    })
    .unsubscribe(); // <- unsubscribe

    // at the end unsubscribe if not needed
```

AppUtils

```
filter = filter && AppUtils.IsArray(filter) ? filter : new Array<any>();
```

```
@Input() set Lookup(x: Array<any>) {
    this.staticIncludedDataSource = new DataSource({
      store: AppUtils.deepCopyArray(x),
      pageSize: 50,
      filter: (z) => {
        return this.excluded.findIndex(y => y === z[this.Key]) === -1;
      }
    });
    this.staticExcludedDataSource = new DataSource({
      store: AppUtils.deepCopyArray(x),
      pageSize: 50,
      filter: (z) => {
        return this.included.findIndex(y => y === z[this.Key]) === -1;
      }
    });
}
```

```
private onBeforeSend(operation: string, ajaxSettings: DxAjaxSettingsModel) {
    // deep copy object to another
    const tmp = AppUtils.deepCopyObject(this.filter);
...
```

```
options.onBeforeSend = (operation: string, ajaxSettings: DxAjaxSettingsModel): void => {
        const f = AppUtils.deepCopyObject(this.filter);

        ajaxSettings.data = AppUtils.Clone({}, f, ajaxSettings.data);
};
```

moment

korištenje ngx-moment:

- **amTimeAgo**: {{myDate | amTimeAgo}} => a few seconds ago
- **amCalendar**: {{myDate | amCalendar}} => Today at 14:00
- **amDateFormat**: {{myDate | amDateFormat:'LL'}} => January 24, 2016
- **amParse**: {{'24/01/2014' | amParse:'DD/MM/YYYY' | amDateFormat:'LL'}} => January 24, 2016
- **amLocal**: {{myDate | amFromUtc | amLocal | amDateFormat: 'DD.MM.YYYY HH:mm'}} => 24.01.2016 12:34 (koristi se npr. za capacity-history komponentu gdje se za sve vrijednosti upisa dodaje time zone i daylight time na strani clienta, u ovom slučaju zapisano vrijeme na strani servisa je 24.01.2016 11:34)
- **amLocale**: {{'2016-01-24 14:23:45' | amLocale:'en' | amDateFormat:'MMMM Do YYYY, h:mm:ss a'}} => January 24th 2016, 2:23:45 pm
- **amFromUnix**: {{ (1456263980 | amFromUnix) | amDateFormat:'hh:mmA'}} => 01:46PM
- **amDuration**: {{ 365 | amDuration:'seconds' }} => 6 minutes
- **amDifference**: {{nextDay | amDifference: today :'days' : true}} days => 1 day
- **amAdd** i **amSubtract**:
    - {{'2017-03-17T16:55:00.000+01:00' | amAdd: 2 : 'hours' | amDateFormat: 'YYYY-MM-DD HH:mm'}} => 2017-03-17 18:55
    - {{'2017-03-17T16:55:00.000+01:00' | amSubtract: 5 : 'years' | amDateFormat: 'YYYY-MM-DD HH:mm'}} => 2012-03-17 16:55
- **amFromUtc**: {{ '2016-12-31T23:00:00.000-01:00' | amFromUtc | amDateFormat: 'YYYY-MM-DD' }} => 2017-01-01 (parsira vrijednost u UTC i priprema moment object u kombinaciji sa amLocal pipe-om)
- **amUtc**: {{ '2016-12-31T23:00:00.000-01:00' | amUtc | amDateFormat: 'YYYY-MM-DD' }} => 2017-01-01

za typescript definirane su globalne extendirane metode nad Date tipom podataka a implementacija se nalazi u **app_common\extensions\date.extensions.ts**, scope je na nivou cijelog AppModula te nije potreban import, definirane su osnovne metode:

- **toLocalDate**, new Date().toLocalDate(), u slučaju da DevExpress dx-date-box komponenta nakon validacije izmijeni Date tip podatka u modelu, toLocalDate() će ispraviti Date tip podatka i prema servisu se šalje ispravan request. Implementacija za Date/DateTime samo za lokalno vrijeme
- **toUTCDate**, model.validTo.toUTCDate(), koristi se uglavnom za konverziju na strani klijenta, lokalnih DateTime vrijednosti u UTC DateTime ovisno o tome na kojoj vremenskoj zoni i daylight vremenu je postavljen preglednik, čime se osigurava da backend servis dohvati uvijek UTC vrijednosti. Implementacija DateTime za lokalno vrijeme u UTC.

- **toLocalFromUTCDate**, model.validTo.toLocalFromUTCDate(), vrši konverziju UTC time vrijednosti u lokalno vrijeme u zavisnosti od postavki na strani klijenta za vremensku zonu i daylight time offset, u ovom slučaju treba paziti da se nakon konverzije ne koriste dodatno gore navedeni pipe-ovi. Implementacija DateTime UTC u lokalno vrijeme.
- **addZone,** ukoliko je potrebno dodati time offset koristi se kao parametar [string | number] u formatu string: [+/-]HH:mm or [+/-]HHmm ili number: [+/-]mm
- **clone**, osigurava sigurno kopiranje Date objekta u novu instancu.