export class RegisterComponent implements OnInit, OnDestroy {

titles$: Observable<Title[]>;

ngOnInit() {

this.titles$ = this.userService.getTitles().pipe(

tap((titles) => {

if (Object.keys(titles).length === 0) {

this.userService.loadTitles();

}

}),

map((titles) => {

return titles.sort(sortTitles); (1)

})

);

}

(1)

D:\应用程序案例\spartacus\projects\storefrontlib\src\shared\utils\forms\title-utils.ts

export const titleScores = {

mr: 1,

mrs: 2,

miss: 3,

ms: 4,

dr: 5,

rev: 6,

};

export function sortTitles(title1: Title, title2: Title) {

if (!titleScores[title1.code] || !titleScores[title2.code]) {

return 1;

} else {

return titleScores[title1.code] - titleScores[title2.code];

}

}

(2)

\*\*\* Facade

D:\应用程序案例\spartacus\projects\core\src\user\facade\user.service.ts

@Injectable({ providedIn: 'root' })

export class UserService {

constructor(

protected store: Store<StateWithUser | StateWithProcess<void>>,

protected authService: AuthService

) {}

/\*\*

\* Returns titles

\*/

getTitles(): Observable<Title[]> {

return this.store.pipe(select(UsersSelectors.getAllTitles));

}

/\*\*

\* Retrieves titles

\*/

loadTitles(): void {

this.store.dispatch(new UserActions.LoadTitles());

}

}

\*\* Store

1 - state

import { Title, User } from '../../model/misc.model';

export const USER\_FEATURE = 'user';

export interface StateWithUser {

[USER\_FEATURE]: UserState;

}

export interface UserState {

account: UserDetailsState;

titles: TitlesState;

}

export interface TitleEntities {

[key: string]: Title;

}

export interface TitlesState {

entities: TitleEntities;

}

export interface UserDetailsState {

details: User;

}

2 - action

export const LOAD\_TITLES = '[User] Load Tiltes';

export const LOAD\_TITLES\_FAIL = '[User] Load Titles Fail';

export const LOAD\_TITLES\_SUCCESS = '[User] Load Titles Success';

export class LoadTitles implements Action {

readonly type = LOAD\_TITLES;

constructor() {}

}

export class LoadTitlesFail implements Action {

readonly type = LOAD\_TITLES\_FAIL;

constructor(public payload: any) {}

}

export class LoadTitlesSuccess implements Action {

readonly type = LOAD\_TITLES\_SUCCESS;

constructor(public payload: Title[]) {}

}

export type TitlesAction = LoadTitles | LoadTitlesFail | LoadTitlesSuccess;

3 - reducer

export const initialState: TitlesState = {

entities: {},

};

export function reducer(

state = initialState,

action: UserActions.TitlesAction | UserActions.ClearUserMiscsData

): TitlesState {

switch (action.type) {

case UserActions.LOAD\_TITLES\_SUCCESS: {

const titles = action.payload;

const entities = titles.reduce(

(titleEntities: { [code: string]: Title }, name: Title) => {

return {

...titleEntities,

[name.code]: name,

};

},

{

...state.entities,

}

);

return {

...state,

entities,

};

}

case UserActions.CLEAR\_USER\_MISCS\_DATA: {

return initialState;

}

}

return state;

}

4 - selector

import { createSelector, MemoizedSelector } from '@ngrx/store';

import { Title } from '../../../model/misc.model';

import {

StateWithUser,

TitleEntities,

TitlesState,

UserState,

} from '../user-state';

import { getUserState } from './feature.selector';

export const getTitlesState: MemoizedSelector<

StateWithUser,

TitlesState

> = createSelector(getUserState, (state: UserState) => state.titles);

export const getTitlesEntites: MemoizedSelector<

StateWithUser,

TitleEntities

> = createSelector(getTitlesState, (state: TitlesState) => state.entities);

export const getAllTitles: MemoizedSelector<

StateWithUser,

Title[]

> = createSelector(getTitlesEntites, (entites) =>

Object.keys(entites).map((code) => entites[code])

);

export const titleSelectorFactory = (

code: string

): MemoizedSelector<StateWithUser, Title> =>

createSelector(getTitlesEntites, (entities) =>

Object.keys(entities).length !== 0 ? entities[code] : null

);

5 - effect

import { makeErrorSerializable } from '../../../util/serialization-utils';

import { UserConnector } from '../../connectors/user/user.connector';

import { UserActions } from '../actions/index';

@Injectable()

export class TitlesEffects {

@Effect()

loadTitles$: Observable<UserActions.TitlesAction> = this.actions$.pipe(

ofType(UserActions.LOAD\_TITLES),

switchMap(() => {

return this.userAccountConnector.getTitles().pipe(

map((titles) => {

return new UserActions.LoadTitlesSuccess(titles);

}),

catchError((error) =>

of(new UserActions.LoadTitlesFail(makeErrorSerializable(error)))

)

);

})

);

constructor(

private actions$: Actions,

private userAccountConnector: UserConnector

) {}

}

\*\*\* Connector

1 -

D:\应用程序案例\spartacus\projects\core\src\user\connectors\user\user.connector.ts

@Injectable({

providedIn: 'root',

})

export class UserConnector {

constructor(protected adapter: UserAdapter) {}

getTitles(): Observable<Title[]> {

return this.adapter.loadTitles();

}

}

2 - D:\应用程序案例\spartacus\projects\core\src\user\connectors\user\user.adapter.ts

export abstract class UserAdapter {

abstract loadTitles(): Observable<Title[]>;

}

\*\*\*\* occ

UserOccModule

@NgModule({

imports: [CommonModule, HttpClientModule],

providers: [

provideDefaultConfig(defaultOccUserConfig),

{ provide: UserAdapter, useClass: OccUserAdapter },

{

provide: PRODUCT\_INTERESTS\_NORMALIZER,

useExisting: OccUserInterestsNormalizer,

multi: true,

},

{

provide: ORDER\_RETURN\_REQUEST\_NORMALIZER,

useExisting: OccReturnRequestNormalizer,

multi: true,

},

],

})

export class UserOccModule {}

import { Title, User, UserSignUp } from '../../../model/misc.model';

import {

TITLE\_NORMALIZER,

USER\_NORMALIZER,

USER\_SERIALIZER,

USER\_SIGN\_UP\_SERIALIZER,

} from '../../../user/connectors/user/converters';

import { UserAdapter } from '../../../user/connectors/user/user.adapter';

import { ConverterService } from '../../../util/converter.service';

import { Occ } from '../../occ-models';

import { OccEndpointsService } from '../../services/occ-endpoints.service';

import { InterceptorUtil, USE\_CLIENT\_TOKEN,} from '../../utils/interceptor-util';

@Injectable()

export class OccUserAdapter implements UserAdapter {

constructor(

protected http: HttpClient,

protected occEndpoints: OccEndpointsService,

protected converter: ConverterService

) {}

loadTitles(): Observable<Title[]> {

const url = this.occEndpoints.getUrl('titles');

return this.http.get<Occ.TitleList>(url).pipe(

map((titleList) => titleList.titles),

this.converter.pipeableMany(TITLE\_NORMALIZER)

);

}

}

\*\*\* OccModule

@NgModule({

imports: [

UserOccModule,

OccConfigLoaderModule.forRoot(),

],

})

export class OccModule {

static forRoot(): ModuleWithProviders<OccModule> {

return {

ngModule: OccModule,

providers: [

{

provide: HTTP\_INTERCEPTORS,

useExisting: WithCredentialsInterceptor,

multi: true,

},

{ provide: OccConfig, useExisting: Config },

provideDefaultConfig(defaultOccConfig),

provideConfigValidator(occConfigValidator),

],

};

}

}

\*\*\*\*OccConfigLoaderModule

import {

ConfigInitializer,

CONFIG\_INITIALIZER,

} from '../../config/config-initializer/config-initializer';

import { SiteContextConfig } from '../../site-context/config/site-context-config';

/\*\*

\* Initializes the Spartacus config asynchronously basing on the external config

\*/

export function initConfig(

configLoader: OccConfigLoaderService,

config: SiteContextConfig

): ConfigInitializer {

/\*\*

\* Load config for `context` from backend only when there is no static config for `context.baseSite`

\*/

if (!config.context || !config.context[BASE\_SITE\_CONTEXT\_ID]) {

return {

scopes: ['context', 'i18n.fallbackLang'],

configFactory: () => configLoader.loadConfig(),

};

}

return null;

}

/\*\*

\* Re-provides the external config chunk given before Angular bootstrap

\*/

@NgModule()

export class OccConfigLoaderModule {

static forRoot(): ModuleWithProviders<OccConfigLoaderModule> {

return {

ngModule: OccConfigLoaderModule,

providers: [

{

provide: CONFIG\_INITIALIZER,

useFactory: initConfig,

deps: [OccConfigLoaderService, SiteContextConfig],

multi: true,

},

],

};

}

}

OccConfigLoaderService

import { Config } from '../../config/config.module';

import { deepMerge } from '../../config/utils/deep-merge';

import { I18nConfig } from '../../i18n/config/i18n-config';

import { SiteContextConfig } from '../../site-context/config/site-context-config';

import { SERVER\_REQUEST\_URL } from '../../ssr/ssr.providers';

import { OccLoadedConfig } from './occ-loaded-config';

import { OccLoadedConfigConverter } from './occ-loaded-config-converter';

import { OccSitesConfigLoader } from './occ-sites-config-loader';

export const EXTERNAL\_CONFIG\_TRANSFER\_ID: StateKey<string> = makeStateKey<

string

>('cx-external-config');

@Injectable({ providedIn: 'root' })

export class OccConfigLoaderService {

constructor(

@Inject(PLATFORM\_ID) protected platform: any,

@Inject(DOCUMENT) protected document: any,

@Inject(Config) protected config: any,

protected sitesConfigLoader: OccSitesConfigLoader,

protected converter: OccLoadedConfigConverter,

@Optional() protected transferState: TransferState,

@Optional()

@Inject(SERVER\_REQUEST\_URL)

protected serverRequestUrl?: string

) {}

private get currentUrl(): string {

if (isPlatformBrowser(this.platform)) {

return this.document.location.href;

}

if (this.serverRequestUrl) {

return this.serverRequestUrl;

}

if (isDevMode()) {

console.error(

`Please provide token 'SERVER\_REQUEST\_URL' with the requested URL for SSR`

);

}

}

/\*\*

\* Initializes the Spartacus config asynchronously basing on the external config

\*/

loadConfig(): Promise<I18nConfig | SiteContextConfig> {

return this.get()

.pipe(

tap((externalConfig) => this.transfer(externalConfig)),

map((externalConfig) =>

deepMerge({}, ...this.getConfigChunks(externalConfig))

)

)

.toPromise();

}

/\*\*

\* Returns the external config

\*/

protected get(): Observable<OccLoadedConfig> {

const rehydratedExternalConfig = this.rehydrate();

return rehydratedExternalConfig

? of(rehydratedExternalConfig)

: this.load();

}

/\*\*

\* Loads the external config from backend

\*/

protected load(): Observable<OccLoadedConfig> {

return this.sitesConfigLoader

.load()

.pipe(

map((baseSites) =>

this.converter.fromOccBaseSites(baseSites, this.currentUrl)

)

);

}

/\*\*

\* Tries to rehydrate external config in the browser from SSR

\*/

protected rehydrate(): OccLoadedConfig {

if (this.transferState && isPlatformBrowser(this.platform)) {

return this.transferState.get(EXTERNAL\_CONFIG\_TRANSFER\_ID, undefined);

}

}

/\*\*

\* Transfers the given external config in SSR to the browser

\*

\* @param externalConfig

\*/

protected transfer(externalConfig: OccLoadedConfig) {

if (

this.transferState &&

isPlatformServer(this.platform) &&

externalConfig

) {

this.transferState.set(EXTERNAL\_CONFIG\_TRANSFER\_ID, externalConfig);

}

}

protected getConfigChunks(

externalConfig: OccLoadedConfig

): (I18nConfig | SiteContextConfig)[] {

const chunks: any[] = [this.converter.toSiteContextConfig(externalConfig)];

if (this.shouldReturnI18nChunk()) {

chunks.push(this.converter.toI18nConfig(externalConfig));

}

return chunks;

}

private shouldReturnI18nChunk(): boolean {

const fallbackLangExists =

typeof (

this.config &&

this.config.i18n &&

this.config.i18n.fallbackLang

) !== 'undefined';

if (fallbackLangExists && isDevMode()) {

console.warn(

`There is an already provided static config for 'i18n.fallbackLang', so the value from OCC loaded config is ignored.`

);

}

return !fallbackLangExists;

}

}