pl-schematics

Portable Schematics

Created by @l.piciollo

Presentazione

pl-schematic nasce da una esperienza pluriennale nell'analisi e progettazione di applicazioni web tenendo conto di massimizzare le prestazioni applicative anche in diversi contesti architetturali, si propone come supporto per industrializzare o standardizzare il processo di creazione di pacchetti Angular > 2, indipendentemente dal cliente interessato.

Si occupa di mettere in piedi tutta una serie di funzionalità core come gestione rete, cache, intercettori, eccezione e molto altro, modificando e aggiungendo classi e configurazioni in totale autonomia nel pacchetto ospitante.

È stata realizzata per operare in contesti dove è richiesto uno startup del progetto, ma con le dovute accortezze è possibile usufruirne anche in progetti già esistenti, corredandoli di nuove funzionalità pronte all'uso.

Come si presenta

pl-schematics è un template SDK di supporto e va installato prima del suo utilizzo.

Questo trova collocamento nella parte **node_module** del progetto ospitante, ma non per questo verrà poi distribuito al momento della build del progetto Angular, evitando appesantimenti vari con file inutili.

Una volta installata, è possibile beneficiarne lanciando la sua esecuzione da riga di comando, come se stessimo creando un nuovo componente o un servizio.

La sua esecuzione guiderà l'utilizzatore nella sua configurazione attraverso delle domande.

Configurazione di avvio

L'installazione dello schematics, implica che l'utente risponda alle seguenti domande.

- Nome della tua compagnia?
  - Qui è possibile inserire il nome della compagnia che sta realizzando il progetto. Di default se non specificato nulla verrà posto di default: ***Accenture***
- Nome del package core?
  - È necessario inserire un package di riferimento, questo per distinguere classi nuove da quelle gia presenti. Di default viene creato il seguente package : ***com/accenture/normalize***
- Inserire il prefisso per le classi core..
  - Opzionale, viene chiesto di inserire una sigla o altro da anteporre al nome delle classi create, questo per evitare omonimia con le gia presenti
- Scegliere per quale browser il sistema deve essere compatibile..
  - Viene mostrata una lista di browser, dove è possibile indicare quale browser è abilitato. Default ***ALL***
- Configurazione del supporto alla login..
  - Viene mostrato un elenco dove è possibile scegliere il tipo di login da configurare, è possibile skippare questa configurazione Default ***NONE***
- Abilitare la configurazione bootstrap?
  - qui è possibile scegliere se configurare il progetto ad utilizzare bootstrap Default ***Y***
- Abilitare la configurazione per lo scanner sonar?
  - Qui è possibile dare la preferenza alla configurazione automatica per lo scanne sonar aziendale o locale che sia. Default ***Y***

Risultato della configurazione di avvio

Al termine della sua esecuzione, si potranno identificare nuove classi, servizi, folder, configurazioni e modifiche ad alcune classi gia presenti. È possibile inoltre risalire alla documentazione della struttura, ispezionabile con il browser. Li è possibile prendere nota di come funzionano le classi core e ed altro.

Come detto, oltre la parte core, vengono create altre folder, alcune delle quali sono vuote ma servono per standardizzare il processo di organizzazione del progetto.

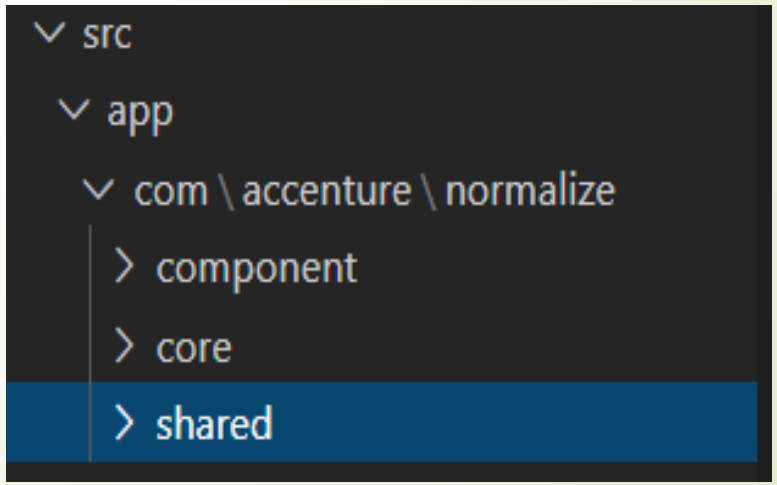
Si avranno quindi folder per incorporare pagine, componenti, servizi, moduli e altro. Seguendo questo schema, si garantisce organizzazione e decoro applicativo.

Alberatura

L'alberatura del progetto si presenta secondo la struttura sotto riporta.

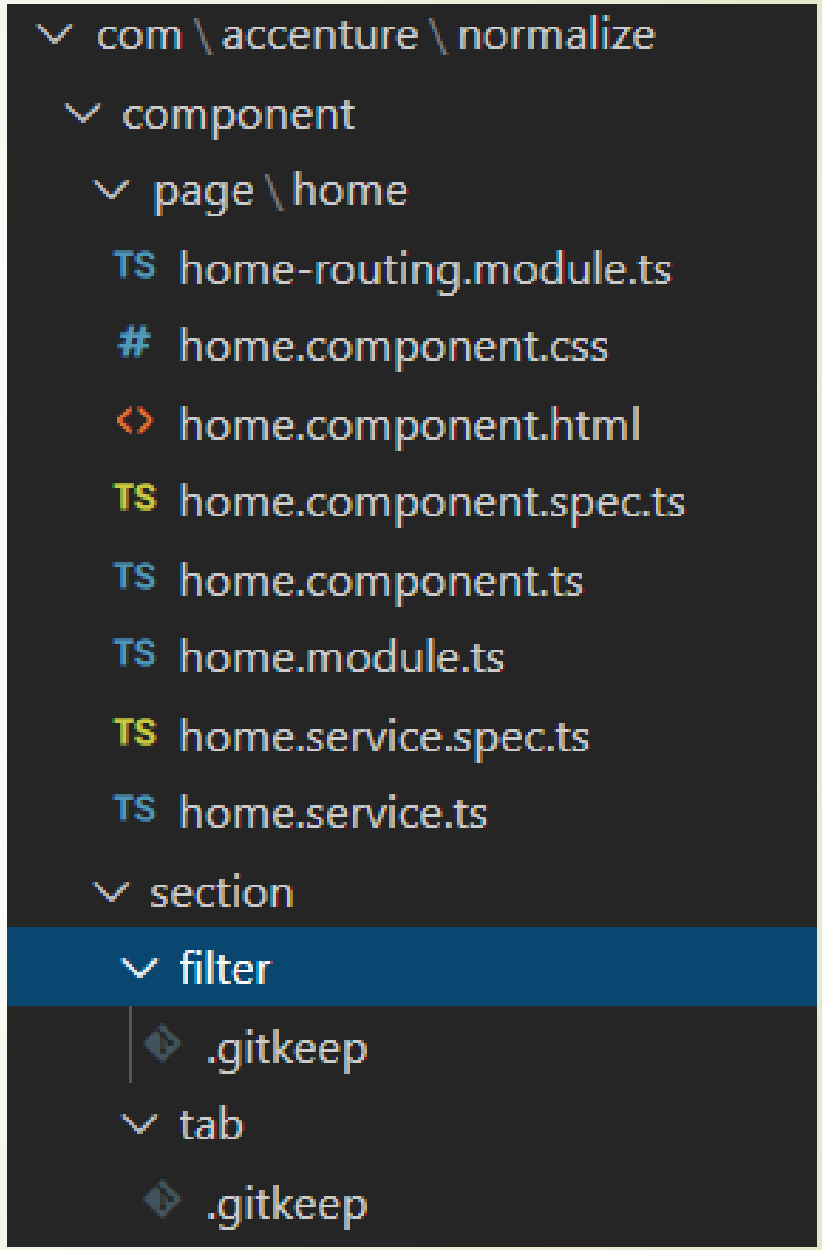
Si tenga conto che la configurazione ha seguito i parametri impostati di default. Al termine della configurazione, viene modificato il file package.json e vengono specializzate due funzionalità per la build in prod e in dev del portale

- **npm run build-dev**
- **npm run build-prod**



Component

- Page
  - Vengono riportati tutti I component che hanno la funzione di pagina, ad esempio è stato creato Il modulo home, come contenitore peeventuali loghi o cruscotti o altro.
    - Ogni pagina deve avere il suo modulo, il suo routing e il suo service.
      - I service devono essere create nel componente e non nel modulo, quindi inseire il provider all'interno del component, questo garantisce la sua distruzione al momento dello scaricamento del modulo.
- Section
  - Qui possono essere riportati sezioni riutilizzabili in piu pagine, come ad esempio filtri di ricerca o tab. Queste strutture devono essere create in modo generico e non specific, quindi verranno inseriti solo I macro component. Anche in questo caso si consiglia l'utilizzo di un modulo proprio per ogni macro componente da realizzare
    - La composizione dei macro component prevede l'implementazione di micro component ma che verranno create nella pasrte shared dell'alberatura.

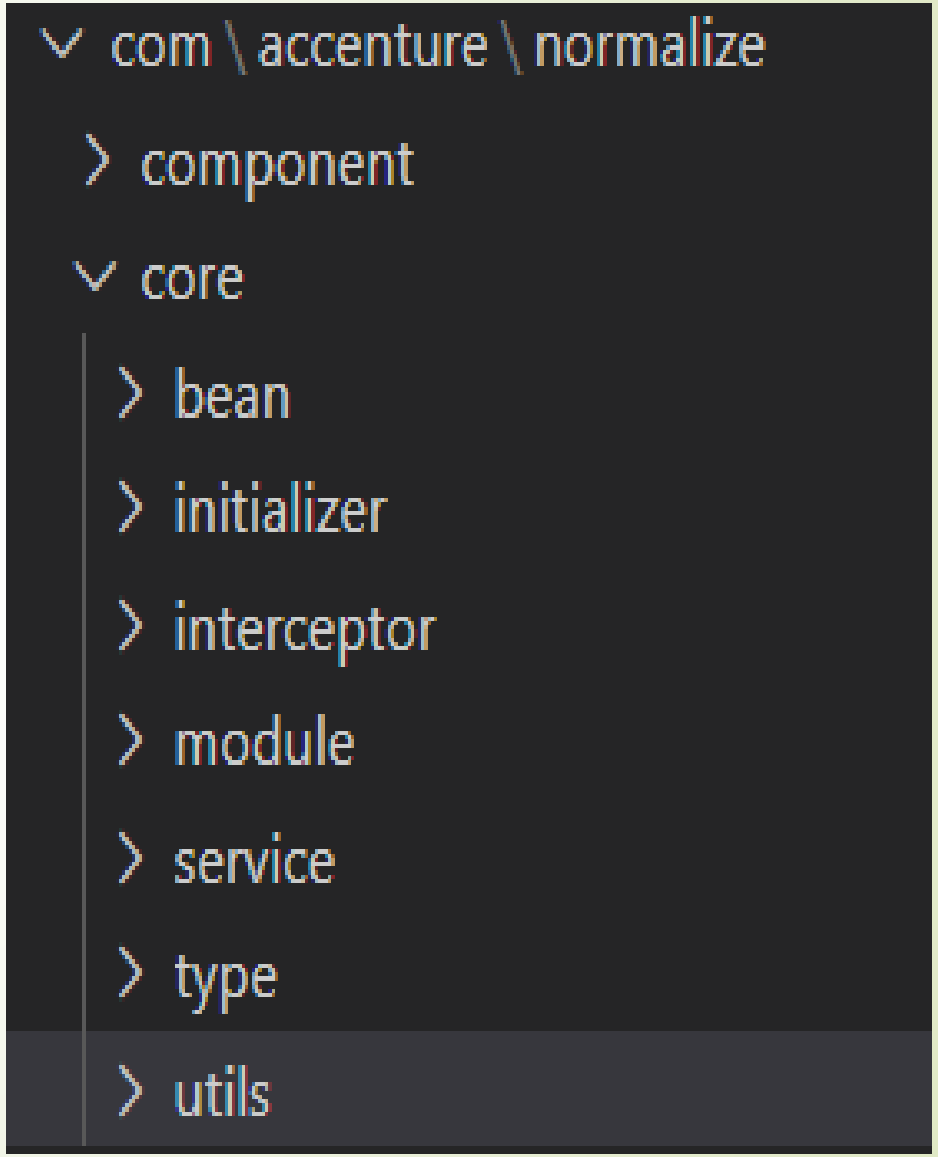


Core

In questa sezione è raccolto il motore vero e proprio di tutta l'applicazione. Vengono creati e configurati metodi accessori funzionalità piu complesse che servono a gestire le rotte di navigazione, controlli su permessi di rotta, chiamate alla rete BE, centralizzazione degli errori, interrutore di chiamata ai servizi ajax, moduli di inizializzazione per reperire informazioni sul browser e tanto altro.

Questo aspetto architetturale non va manomesso o modificato, non vi è alcuna motivazione. in quanto tutto il sistema è autonomo e attivo in background, l'intervento manuale non è previsto e potrebbe portare complicazioni di prestazioni o errori accidentali

l'unico intervento che si potrebbe aver bisogno e di gestire la centralizzazione degli errori, la login e gli errori di rete che il core mette a disposizione. a tale proposito è previsto che il core demandi in modo asincrono eventi nella ***global.service.ts****. Questa classe è stata adibita all'intercettazione di questi eventi. Qui è possibile gestirne in autonomia tutte le situazioni per cui si necessita l'intervento.*



Descrizione Core

- Bean
  - Contesto in cui vengono specificati e racchiusi i classi bean che fungono da trasporto dei dati tra la parte core e lib di supporto
- Initializer
  - Sono presenti funzioni di avvio applicativo, intercettano l'avvio del portale evitando caricamenti dei moduli applicativi sino a quando non siano siddisfatti tutti i controlli
    - Avviene la fasi autenticazione.. Non viene avviato il portale fin quando non si è autenticati
    - Avviene la fase di identificazione dell'ambiente ospitante il portale
      - Viene riconosciuto il browser e vengono aggiunte funzionalità supplementari
        - Gestione di array, di stringhe, di json
        - Funzione di override per il meccanismo di download.. In base a IE o chrome il funzionamento varia , il core si occupa di questo automatismo, lasciando al programmatore finale solo la semplice chiamata di download, ad esempio.

Descrizione Core

- Interceptor
  - Sono presenti tutti gli intercettori di servizio, vengono specializzati quindi gli intercettori di rete e gli intercettori per la centralizzazione e la gestione degli errori.
    - l'intercettore di rete, si occupa di controllare se l''utente è autenticato e si occupa di reperire in autonomia il token da passare al BE. In caso di anomalie o errori riscontrati durante la fase di chiamata, viene emesso un evento di broadcast rintracciabile nella **global.service.ts**
    - La centralizzazione degli errori ha lo stesso comportamento.. Si occupa di acquisire l'errore riscontrato e di inoltrarlo alla **global.service.ts**
- Module
  - È presente il modulo di avvio delle funzionalità del core. Qui sono specificati tutti i parametri di inizializzazione del core, coma la durata della cache rest, il tipo di browser da bloccare, se o meno disabilitare la console.log, utile in contesto produttivo dove i log non sono previsti. È presente l'inizializzazione delle variabili per la login, è presente la configurazione per la durata alle chiamate rest prima del timeout. Sono presenti anche gli starter per le loading bar sia di navigazione portale che di chiamata alla rete.

Descrizione Core

- Service
  - È presente una classe di servizio, adibito all'invocazione di funzionalità di rete.. Come GET, POST ed altro. In questo contesto è possibile risalire a diverse tipologie di chiamata, ad esempio asincrone, ma che vengono interrotte automaticamente al cambio di rotta o chiamate background che girano in autonomia, non verranno e mai interrotte se non da un timeout del BE o di rete. Questi servizi sono utilizzati per upload e download file
    - Questo servizio funge da interfaccia per la **pl-core.utils.library** che è una libreria di supporto dove vengono racchiuse molte funzionalità. Questa lib mette a disposizione una svariata quantità di funzionalità e servizi utili non solo alla parte core dell'applicativo, ma possono essere utilizzate anche in contesto shared, dove la realizzazione delle funzionalità spetta al programmatore. La lib di supporto espone la propria documentazione sempre all'interno di quella schematics.
      - cosa espone la lib, tra le più importanti
        - Funzionalità per la rete.. Con gestione di progress bar e interruzione di chiamata
        - Virtualizzazione di codice complesso per evitare appesantimenti applicativi
        - Specializzazione di funzionalità per la grafica, screenshot ad esempio
        - Funaionalita per identificare il resize del browser
        - Messa in opera del blocco browser
        - Aggiunta funzionalità per le stringhe, gli array, oggetti JSON

Descrizione Core

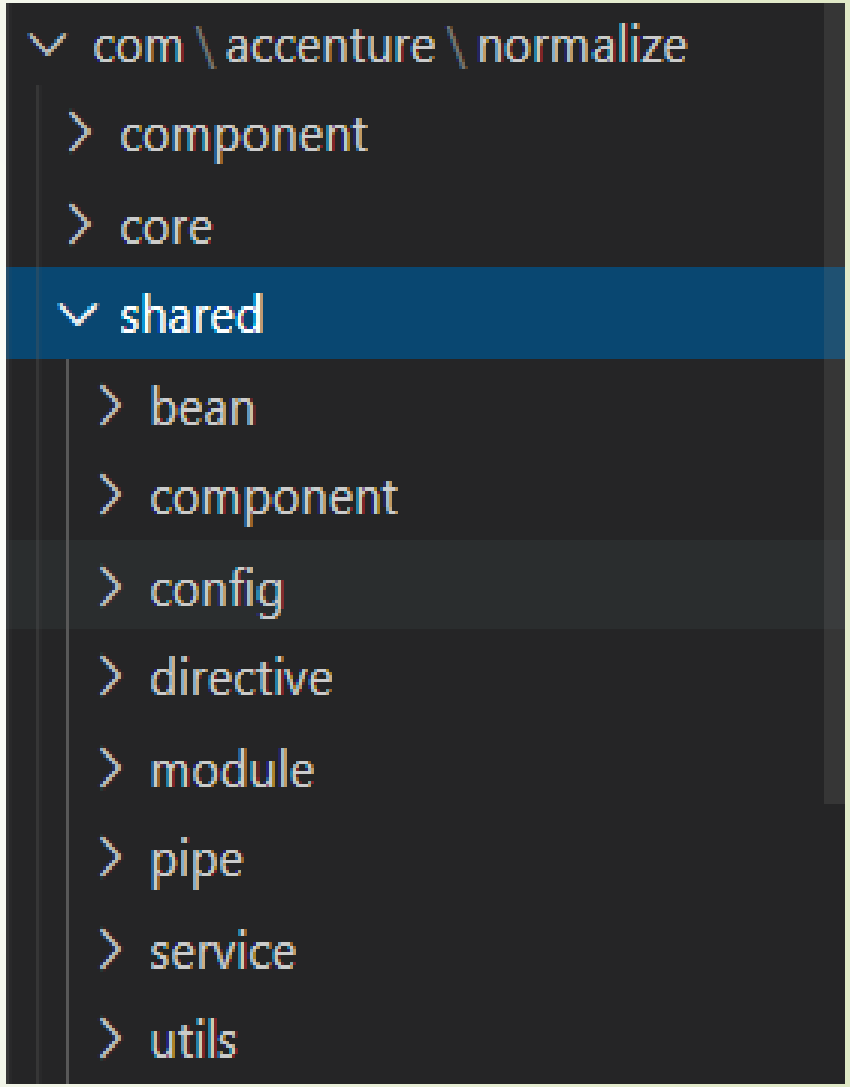
- type
  - Sono presenti file contenenti delle tipologiche. Queste a sua volta contengono la descrizione degli eventi che il core lancia quando previsto .
- Utils
  - Classi di utlità, vengono specializzate qui tutte le classi di utilita per l'esecuzione del core, come ad esempio le configurazioni delle loading bar.

Shared

Qui viene presetata un gerarchia di folder campione. Vengono suddivise le diverse tipologie di raccolta file. *Gli sviluppi dei nuovi componenti devono avvenire in questo contesto distribuito.*

- *Bean*
  - *Classi di trasporto per il be*
- *Component*
  - *Vengono specificati i componenti micro, utilizzabili nelle pagine, come menu, footer, header, e altro*
- *Config*
  - *Raccoglie le classi che fungono da configurazione, ad esempio classi di costanti o configurazioni per tabelle*
- *Diretive e pipe*
  - *Vengono utlizzate per la creazione di direttive o pipe utili a tutto il contesto applicativo*
- *Service*
  - *Vengono qui scritti tutti i serviziglobali che servono al contesto applicativo e non ai singoli moduli, in quanto i singoli moduli devono avere il proprio servizio*
- *Utils*
  - *Classi di utilità*
- *Module*
  - *Contiene i moduli condivisi con tutta l'applicazione*

Il template crea in autonomia diverse classi gia impostate come la global.service.ts, utils.ts e sharedmodule.ts



Librerie di supporto

La creazione del template, prevede che vengono create funzionalità di supporto aggiuntive

- **Bootstrap 4**
  - Viene innescato il processo di configurazione per l'utilizzo immediato di tutto il contesto bootstrap
- **Sonar**
  - Viene configurato l'ambiente per poter eseguire i comandi di scansione del tool sonar.
    - A tale proposito viene modificato il package.json inserendo un nuovo comando lanciabile tramite.
      - **npm run sonar**
- **pl-core-utils-library**
  - Libreria di supporto per le utilita e funzionlalita aggiuntive
- **msal**
  - Libreria azure per login

installazione

L'installazione del template prevede i seguenti passi:

1. Creazione di un nuovo progetto
    1. **ng new myprojectName**

2. Installazione del template da npm
    1. **npm i pl-schematics@latest**

3. Esecuzione del template:
    1. **schematics pl-schematics:pl-schematics –force**

N.B
I passi descritti sono specifici per adattare un nuovo progetto al template pattern. Per adattare un progetto gia esistente, occorre prestare attenzione a clonare i seguenti file

- **app.component.html**
- **environment.ts e environment.prod.ts**
- **sonar-project.properties**
- **app-routing.module.ts**

pl-schematics

Esempio BE

Created by @l.piciollo

Esempi di chiamata al BE

- Vengono riportate alcune situazioni di chiamata al BE, per mostrare la semplicità d'uso

App.component.html

```
<!--  @author l.piciollo
      Si specializza un pulsante per chiamare il BE mock
-->
<button (click)="callMock()">Esempio di chiamata</button>
```

App.component.ts

```
/**
 * @author l.piciollo
 * funzione richiamata dal componente web,
 * per inescare il processo .di chiamata al BE
 * come mostrato sotto, occorre passare dal servizio
 * global.service.ts. in questo caso sono passati
 * due parametri e viene effettuata la sottoscrizione
 * alla richiesta
 */
callMock() {
  this.global.callMock(1, 2).subscribe(sb => {
    console.log(sb); //viene stampato il comntenuto AJAX
  }, error => {
    console.log(error)  //In caso di errore, viene stampato
  }, () => {
    console.log("Completed..") // al termine della richiesta, si ha il complete
  })
```

Esempi di chiamata semplice al BE

global.service.ts

```
/**
* @author l.piciollo
* esempio di chiamata ajax centralizzata nella global.service.ts
* si occupa di invocare il be tramite la url.
* questa è modificata con il .format, una funzione della pl_core_utils
* che si occupa di replicare le string che hanno  {0} , {1} pattern
* la funzione si interfaccia con la parte core, e riceve il json di risposta
* impostato per default.
*/
callMock(p1:number,p2:number): Observable<any> {
  return new Observable<any>(obs => {
    this.httpService.GET(environment.http.api.exampleApeNoCache.format(p1,p2) , {}).subscribe(sb => {
      obs.next(sb);
      obs.complete()
    }, error => {
      obs.error(error);
    }, () => { console.log("Completed..")})
  })
}
```

Le chiamate al be, avvengono di base in soli tre step

1) Configurazione del componente html
2) Specializzazione del ts appartenente al componente html

Per la terza fase occorre capire se la funzionalità è utilizzata da tutti i moduli o meno, in caso di utilizzo su di un singolo modulo la funzione di chiamata va riportata sul servizio del modulo in altro caso va posta nel global.service.ts

Ad esempio

- Risalire a una tipologica
  - I dati reperiti dal be possono servire a tutti i moduli, per la decodifica o codifica di determinati stati di un item o di un task
- Reperire i profili di un utente
  - La funzionalita, non occorre metterla a livello globale, ma va specializzata nel servizio del modulo della login, ad esempio o in un modulo che gestisce le profilazioni

Le chiamate alla rete possono essere messe in cache o meno.. Il funzionamento è automatico non occorre configurare nulla… è il sistema che capisce in base alla tipologia di url se tentare di mettere in cache o meno

Tutte le url di rete devono essere poste nell'environment e non nel codice sparse cosi alla meglio.. Le chiamate come detto possono essere messe in cache, occorre stabilire quali mettere e quali no..

Generalmente si mettono in cache chiamate GET a servizi di tipologihe, che praticamente variano di rado durante tutta la sessione applicativa

Per indicare che una chiamata deve essere messa in cache occorre annotare la url con **@cachable@.**

È possibile anche inserire i pattern format, come si vede **{0} {1},** sono poi utilizzati da .**format** all'interno della chiamata al be per inserire i parametri da passare

```
api: {
  /**
   *  @author l.piciollo
   *  si riporta un esempio di una api riconosciuta come storable, grazie al tag @cacable@ presente nella URL.
   *  si nota come i parametri sono passati con {0} e {1}.. il sistema è equipagiato da una funzionalita che specializza
   *  le stringhe ad avere il format function.. quindi .. è possibile formattare la url richimandola in questo modo:
   *  E.S.
   *    let url = environment.exampleApi.format("P1","P2")
   *    quindi avviene una formattazione per posizione dei paraetri..
   *
   *  exampleApi: `@cachable@/example/cacable/api?param1={0}&param2={1}`
   */
  exampleApi: `@cachable@/example/cache/api?param1={0}&param2={1}`,
  exampleApeNoCache: `example/no/cache/api`
}
```

Esempi di chiamata al BE con stato avanzamento

global.service.ts

```
/**
 * @author l.piciollo
 * esempio di chiamata ajax centralizzata nella global.service.ts
 * si occupa di invocare il be tramite la url.
 * questa è modificata con il .format, una funzione della pl_core_utils
 * che si occupa di replicare le string che hanno  {0} , {1} pattern
 * la funzione si interfaccia con la parte core, e riceve il json di risposta
 * impostato per default. in aggiunta vi è una callback che riceve in ingresso un oggetto
 * che indica in formato json, lo stato avanzamento della chiamata
 */
callMock(p1:number,p2:number , callBack?: (id: any) => void): Observable<any> {
  return new Observable<any>(obs => {
    this.httpService.GET(environment.http.api.exampleApeNoCache.format(p1,p2) , {},null,callBack).subscribe(sb => {
      obs.next(sb);
      obs.complete()
    }, error => {
      obs.error(error);
    }, () => { console.log("Completed..")})
  })
}
```

A differenza della precedente chiamata, è stato inserito anche il parametro di callBack. Questa funzione viene invocata al momento della creazione della richiesta ajax. Ogni richiesta ha un id univoco, quindi è possibile risalire allo stato in qualsiasi momento, basta passare l'id tokenAjax

Esempi di chiamata al BE con stato avanzamento

Sotto si riporta come effettuare una chiamata alla global.service.ts passando una callback per reperire il tokenAjax creato dal servizio

```
/**
 * @author l.piciollo
 * funzione richiamata dal componente web,
 * per inescare il processo .di chiamata al BE
 * come mostrato sotto, occorre passare dal servizio
 * global.service.ts. in questo caso sono passati
 * due parametri e viene effettuata la sottoscrizione
 * alla richiesta. qui viene richiesto di reperire l'oggetto
 * che indica a runtime lo stato di proressione
 */
callMock() {
  this.global.callMock(1, 2, (ticket) => {
      console.log(ticket)
  }).subscribe(sb => {
    console.log(sb); //viene stampato il comntenuto AJAX
  }, error => {
    console.log(error)  //In caso di errore, viene stampato
  }, () => {
    console.log("Completed..") // al termine della richiesta, si ha il complete
  })
```

Esempi di chiamata al BE con stato avanzamento

Rimanendo in ascolto sul tokenAjax, sotto un esempmio, è possibile a questo punto prendere i dati della progressione dal servizio

```
c29244fd-028c-871b-242a-6517e1b804ba
```

Richiamando la funzione sotto presente in global.service.ts, si riceve il subject staccato dal servizio httpService della pl-core-utils

```
/**
 * @author l.piciollo
 * passando in ingresso l'ajaxToken, è possibile ricevere il
 * subject per sottoscriversi e ricevere in tuntime i valori di stato della chiamata
 * @param idAjax
 */
getProgression(idAjax:string):Subject<any> {
  try {
    return this.httpService.TAILAJXCALL(idAjax);
   }
  catch (e) {
    throw new ErrorBean(e.message)
  }
}
```

Esempi di chiamata al BE con stato avanzamento

Il chiamante puo sottoscriversi al subject seguendo la modalità sotto riportata ed evidenziata

```
/**
 * @author l.piciollo
 * funzione richiamata dal componente web,
 * per inescare il processo .di chiamata al BE
 * come mostrato sotto, occorre passare dal servizio
 * global.service.ts. in questo caso sono passati
 * due parametri e viene effettuata la sottoscrizione
 * alla richiesta. qui viene richiesto di reperire l'oggetto
 * che indica a runtime lo stato di proressione
 */
callMock() {
  this.global.callMock(1, 2, (ticket) => {
    this.global.getProgression(ticket).subscribe(sb => {
      console.log(sb);
    })
  }).subscribe(sb => {
    console.log(sb); //viene stampato il comntenuto AJAX
  }, error => {
    console.log(error)  //In caso di errore, viene stampato
  }, () => {
    console.log("Completed..") // al termine della richiesta, si ha il complete
  })

}
```

Esempi di chiamata al BE con stato avanzamento

Si è scelto di stampare a video il risultato tramite la console.log, e il log riporta

Questo oggetto si avra ad ogni cambiamento di rete, in quanto ci si è sottoscritti.. E fino alla fine del servizio sarà possibile ricevere informazioni.

N.B al termine del servizio, l'oggetto viene rimosso dalla coda, quindi non piu reperibile.

Esempi di chiamata al BE con stato avanzamento

Nell'oggetto reperito, non vi sono solo le informazioni sulla progressione, ma vi è un oggetto chiamato interrupt. Questo oggetto serve per uccidere il processo, in caso si manifesti necessità. Quindi, sotto viene riportato come ad esempio viene terminata una richiesta

```
/**
 * @author l.piciollo
 * funzione richiamata dal componente web,
 * per inescare il processo .di chiamata al BE
 * come mostrato sotto, occorre passare dal servizio
 * global.service.ts. in questo caso sono passati
 * due parametri e viene effettuata la sottoscrizione
 * alla richiesta. qui viene richiesto di reperire l'oggetto
 * che indica a runtime lo stato di proressione
 */
callMock() {
  this.global.callMock(1, 2, (ticket) => {
    this.global.getProgression(ticket).subscribe(sb => {
      sb.interrupt.next(true);
    })
  }).subscribe(sb => {
    console.log(sb); //viene stampato il comntenuto AJAX
  }, error => {
    console.log(error)  //In caso di errore, viene stampato
  }, () => {
    console.log("Completed..") // al termine della richiesta, si ha il complete
  })

}
```

pl-schematics

Esempio Decoratori

Created by @l.piciollo

Esempi decoratori

pl-core-utils espone altre funzionalità sempre in ottica di ottimizzazione e centralizzazione delle funzionalità
È possibile usufruire di diversi decoratori: di classe, di metodo e di attributo.

```
PLDelay(milliseconds: number = 0)
```

Inserendo la notazione sotto.. Sopra un qualsiasi metodo, è possibile ritardare la sua esecuzione una volta invocato. L'annotazione però trasforma automaticamente il tipo di ritorno in Observable. Quindi occorre sottoscriversi alla funzione per ricevere il response dell'esecuzione.

```
PLTraceHooks(disabled: Array<string> = [])
```

Questa annotazione, posta su di una classe component, impone alle funzioni di ciclo hook del componente di loggare la loro attivazione ogni qual volta e per tutta la durata del componente. È possibile specificare quale ciclo non prendere in considerazione.

Esempi decoratori

```
PLFormatDate(format: FORMAT_DATE | string = "dd/MM/yyyy hh:mm:ss", local = "it-IT", localeId = "it")
```

Posizionando questa annotation su di un attributo di classe, si chiede al sistema di decorare a run time l'attributo con i relativi metodi accessori get e set e impone di assumere la data in formato dichiarato.
Quindi se diciamo che il campo **dataCreazione:Date** deve avere 'dd/mm/yyyy' al momento della sua valorizzazione, in automatico verrà associato questo valore.

```
PLUnsubscribe(ignore = [])
```

Questa annotazione, posta su di una classe, indica la distruzione automatica di tutti gli osservatori dichiarati.. Cio significa che il sistema compensa la dimenticanza del programmatore a killare eventuali sottoscrittori attivati, questo avviene al momento dell'onDestroy. È possibile passare il nome della variabile da non distruggere.

Esempi decoratori

```
PLPermission(enabled = true);
```

Posizionando questa annotazione su di una classe di componente, si abilita in caso di enable =true, il sistema a prendere possesso degli elementi del DOM al momento in visualizzati in pagina.

La funzionalità va configurata seguendo questa logica

lanciare document.dispatchEvent(new CustomEvent('PL:SETPERMISSION', { detail:
[PROFILO1,PROFILO2,PROFILO3,...]
}));

inserire nel DOM <input permission="READONLY" type="text>" e al decoratore passare @PLPermission(true)

l'elemento del dom viene elininato in quanto non contiene il permesso READONLY.

per NON far eliminare il componente dalla routin.
inserire nel DOM <input permission="PROFILO1" type="text>" e al decoratore passare @PLPermission(true)

il componente non viene eliminato in quanto il permesso è presente.

Created by

@l.piciollo