UTS-Angular2 Web Application

September, 05, 2017

Mikyung Lee

Development Environment

- Prerequisites for development
  - NPM (https://www.npmjs.com/)
    - After installing, type 'node -v' and 'npm -v' in command line to confirm that it is installed successfully.
    - Need to install a number of additional packages during development, for example to load typescript into npm, type 'npm install typescript'.
    - Try to type 'npm list' and 'npm outdated' within a angular2 project directory.
    - Node.js installed include
  - ATOM (https://atom.io/)
    - Add atom-typescript plug-in in order to use useful features such as auto-complete.
  - Tomcat (https://tomcat.apache.org/)
    - Web application server
  - Bootstrap
    - Version 4 beta
  - Angular CLI (https://github.com/angular/angular-cli) – Optional
    - It facilitates generating new Angular project
    - A Command-Line Interface for Angular
    - Type 'npm install –g @angular/cli'

Angular 2

- Overview
  - A web framework for web applications.
- History
  - Angular official blog (http://angularjs.blogspot.com/)
- Resources
  - Angular.io (https://angular.io/)
  - Angular 2 Thoughtram (https://blog.thoughtram.io/exploring-angular-2/)
  - ng-conf youtube vidoes (https://www.youtube.com/user/ngconfvideos)

Modules

- Overview
  - A container for a group of related components, services and so on.
  - There are two kinds of modules based on their roles in application – RootModule vs. FeatureModule. Every Angular app has at least one root module, which you will bootstrap to launch the application.
  - There are two kinds of modules based on the time when it is loaded – eagerly loaded module vs. lazy loaded module
  - In UTS, each module is resident in each folder. Each module has its own definition in the index.ts. Each module has its own router. All the modules are implemented to load as lazy module because it is good for performance.
  - UTS is composed of a total of 10 modules, one root module of AppModule and 9 feature modules. The core module includes the most significant features, such as services, store and models. All the modules import core module. Shared module will have configuration and pipe relevant features, which most of the modules will need.

Router

- Overview
  - To map a URL to a component in the single-page application.
- Strategies
  - HashLocationStrategy
    - Hash sign is a separator between the base URL and the client-side locations.
  - PathLocationStrategy
    - The default location strategy, history-based strategy.
  - In UTS, HashLocationStrategy is adopted as most of the web applications.

UTS-Angular2 Web application Overview

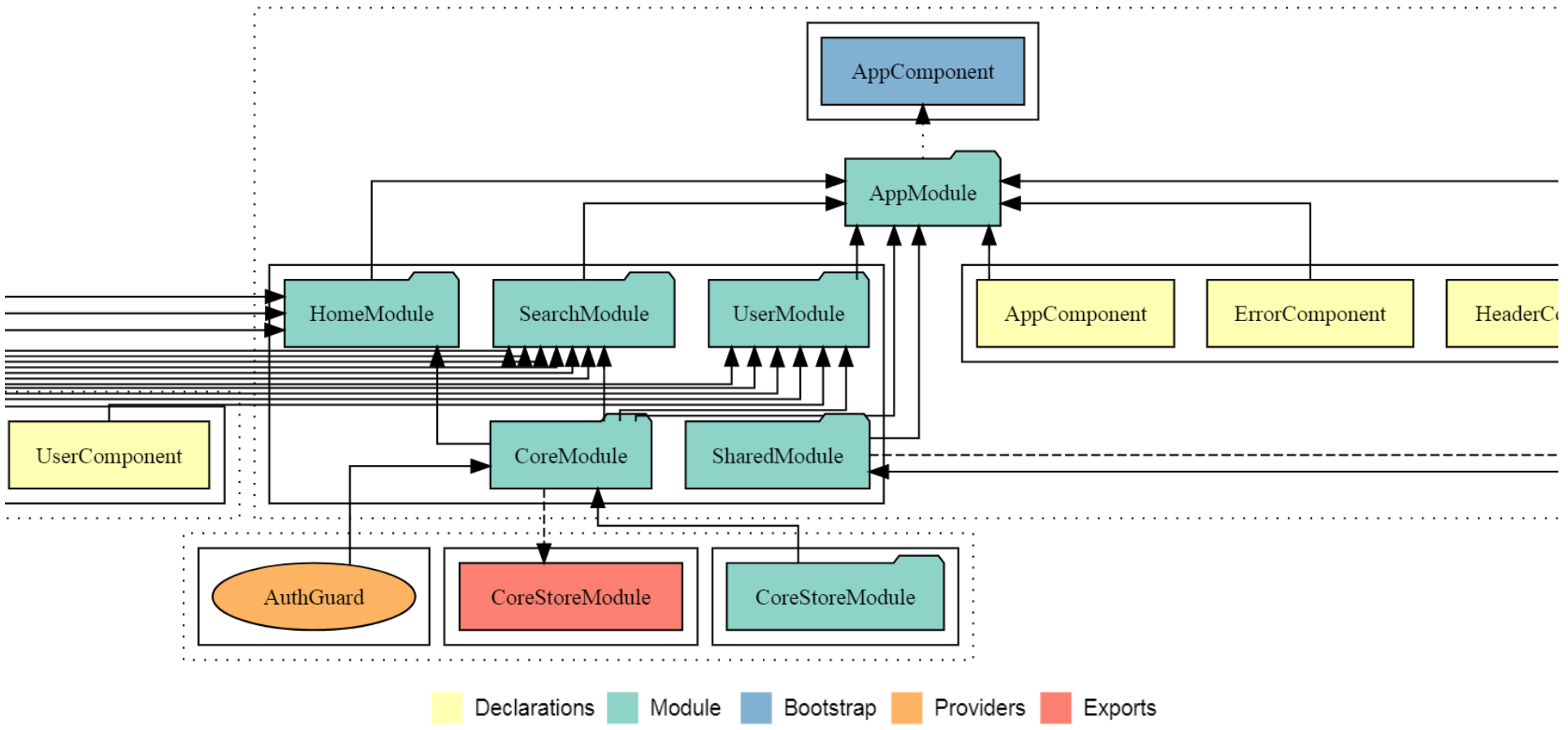


Figure 1. UTS-Angular2 web application overview (Simplified version)

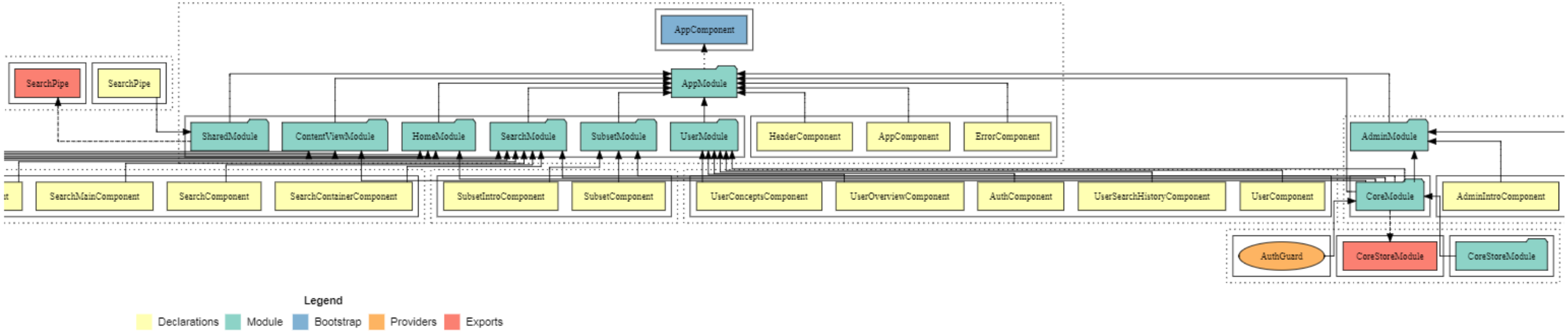


Figure 2. UTS-Angular2 web application overview

Modules / AppModule (Root Module)

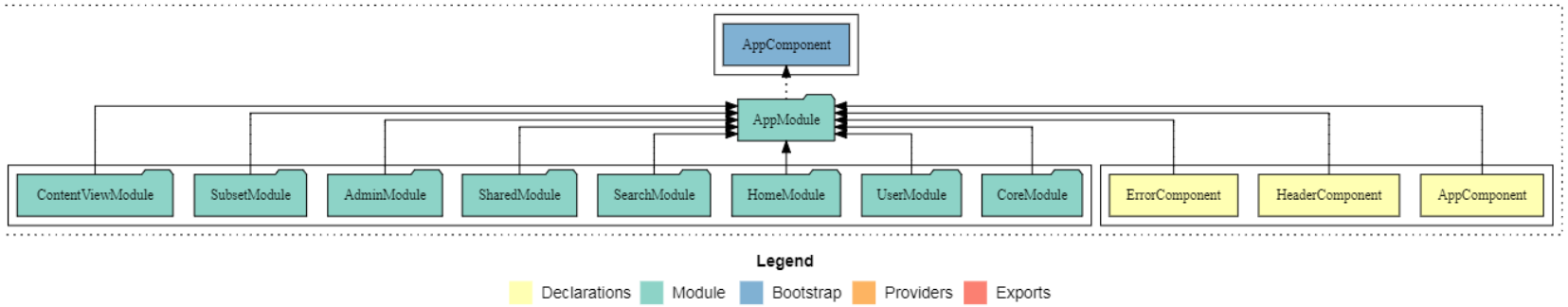


Figure 3. The composition of root module (AppModule). A total of nine feature modules are embedded into root module, such as HomeModule, UserModule, CoreModule, CoreStoreModule (not shown here) and so on.

Modules / CoreModule

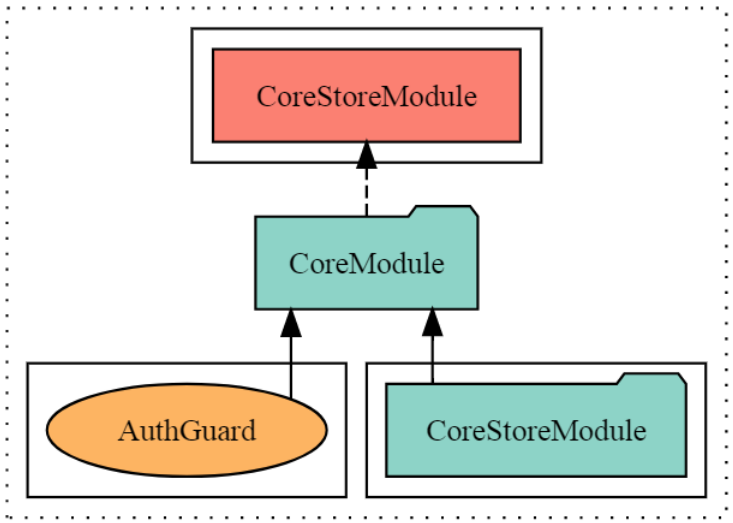


Figure 4. The composition of CoreModule. This module includes most of the significant features, such as services, models, store, authentication guard and so on.

Modules / UserModule

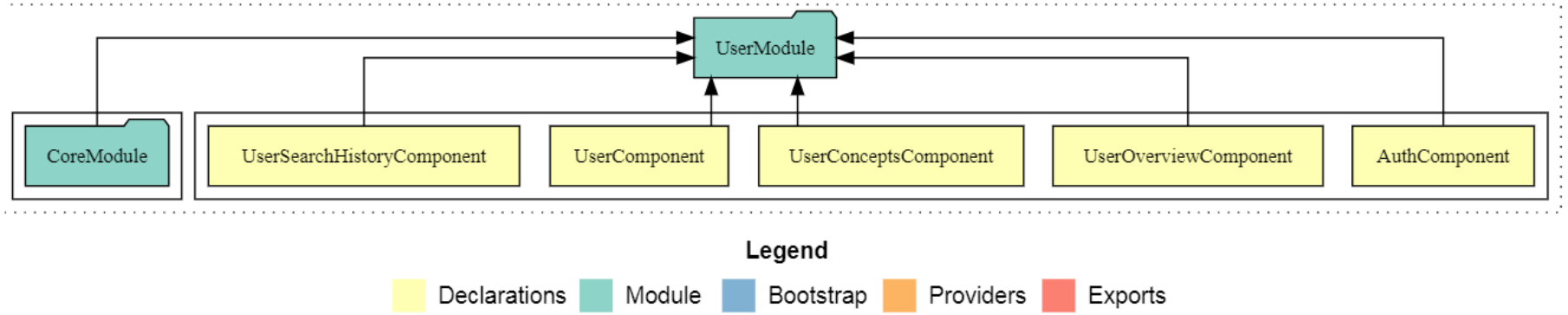


Figure 5. The composition of UserModule. This module is responsible for user relevant functions, such as authentication, user profile, user preferences and so on. It imports CoreModule as indicated.

Modules / SearchModule

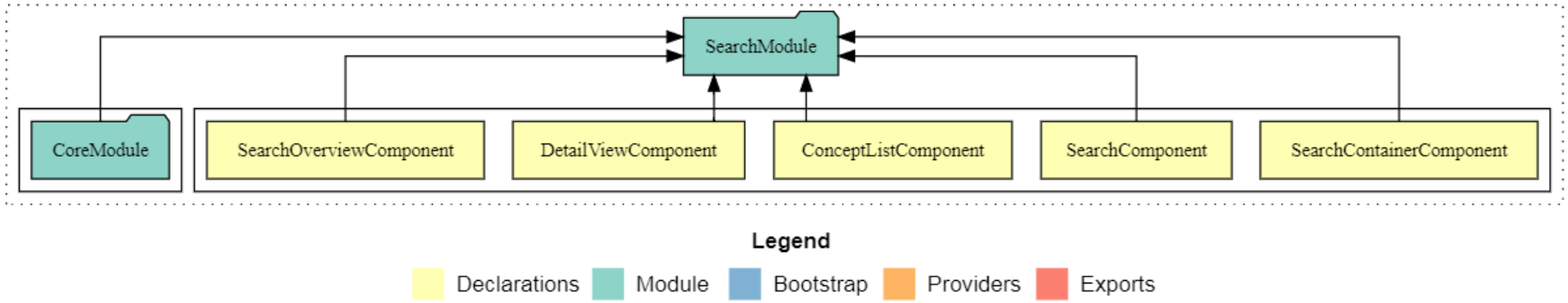


Figure 6. The composition of SearchModule. This module is in charge of search relevant feature, such as term, ID searches.

app.module.ts

```ts
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormControl, FormsModule } from '@angular/forms';
import { HttpModule } from '@angular/http';
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
import { RouterModule } from "@angular/router";
import { routes } from './app.routes';
import { AppComponent } from './app.component';
import { HeaderComponent } from './header.component';
import { ErrorComponent } from './error.component';
import { CoreModule } from './core/index';
import { UserModule } from './user/index';
import { HomeModule } from './home/index';
import { SearchModule } from './search/index';
import { SharedModule } from './shared/index';
import { AdminModule } from './admin/index';
import { SubsetModule } from './subset/index';
import { ContentViewModule } from './contentview/index';

@NgModule({
  imports: [
    BrowserModule,
    HttpModule,
    FormsModule,
    CoreModule,
    UserModule,
    HomeModule,
    SearchModule,
    SharedModule,
    AdminModule,
    SubsetModule,
    ContentViewModule,
    RouterModule.forRoot(routes, {useHash: true})
  ],
  declarations: [
    AppComponent,
    HeaderComponent,
    ErrorComponent
  ],
  bootstrap: [AppComponent],
  providers: [ ]
})
export class AppModule {
}

platformBrowserDynamic().bootstrapModule(AppModule);
```

app.routes.ts

```ts
import { Routes, RouterModule, Router, ActivatedRoute } from "@angular/router";
import { AppComponent } from './app.component';
import { ErrorComponent } from './error.component';
import { AuthGuard } from './core/guards/auth.guard';

export const routes: Routes = [
  {
    path: '',
    loadChildren: './home/index#HomeModule'
  },
  {
    path: 'user',
    loadChildren: './user/index#UserModule',
    canActivate: [ AuthGuard ]
  },
  {
    path: 'search',
    loadChildren: './search/index#SearchModule'
  },
  {
    path: 'admin',
    loadChildren: './admin/index#AdminModule'
  },
  {
    path: 'subset',
    loadChildren: './subset/index#SubsetModule'
  },
  {
    path: 'contentview',
    loadChildren: './contentview/index#ContentViewModule'
  },
  {path: '**', component: ErrorComponent},
];
```

Not yet implemented

Modules

- @NgModule
  - Decorator to define module
- Imports
  - Imports necessary modules
- Declarations
  - Declares components module will use
- Providers
  - Register services module will use
- BootstrapModule
  - The module that will be launched first
- Bootstrap
  - The component that will be launched first
- loadChildren
  - The way to load module as lazy module
- CanActivate
  - The way to restrict access based on proper authentication

search/search.routes.ts

```ts
import { SearchContainerComponent } from './search.container.component';
import { DetailViewComponent } from './components/definition.component';
import { SearchOverviewComponent } from './components/search.overview.component';

export const SearchRoutes = [
  {
    path: 'search',
    component: SearchContainerComponent,
    children: [
      { path: '', redirectTo: 'overview', pathMatch: 'full' },
      { path: 'overview', component: SearchOverviewComponent },
      { path: ':ui', component: DetailViewComponent }
    ]
  },
];
```

search/components/definition.compoment.ts

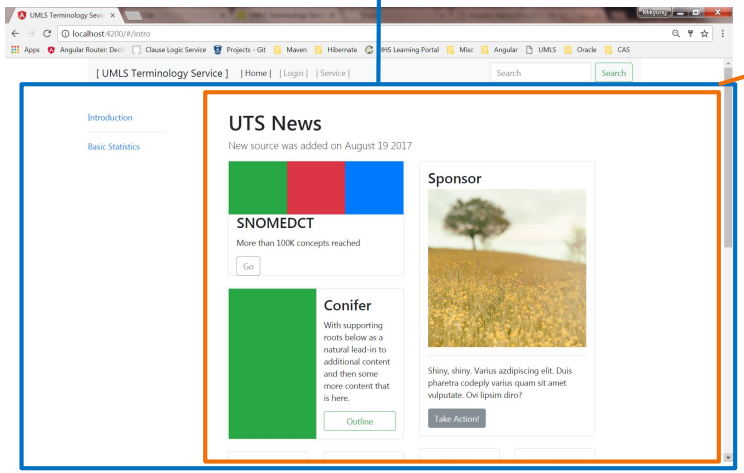
```ts
export class DetailViewComponent implements OnInit {
  routeSubscription$: Subscription;
  cui: string;
  definitions: Array<Definition> = [];

  constructor (private searchService: SearchService,
               private route: ActivatedRoute
  ) {
  }

  ngOnInit() {
    this.routeSubscription$ = this.route.params.subscribe(
      (params: any) => {
        this.cui = params['ui'];
      }
    );
  }
}
```

search/index.ts

```ts
import { CommonModule } from '@angular/common';
import { NgModule } from '@angular/core';
import { FormControl, FormsModule } from '@angular/forms';
import { HttpModule } from '@angular/http';
import { RouterModule } from '@angular/router';
import { SearchRoutes as routes } from './search.routes';
import { CoreModule } from '../core/index';
import { ConceptListComponent } from './components/concept.list.component';
import { DetailViewComponent } from './components/definition.component';
import { SearchComponent } from './components/search.component';
import { SearchContainerComponent } from './search.container.component';
import { SearchOverviewComponent } from './components/search.overview.component';

@NgModule({
  declarations: [
    SearchContainerComponent,
    SearchComponent,
    ConceptListComponent,
    DetailViewComponent,
    SearchOverviewComponent
  ],
  exports: [

  ],
  providers: [

  ],
  imports: [
    CommonModule,
    FormsModule,
    RouterModule.forChild(routes),
    CoreModule
  ]
})
export class SearchModule {}
```

app.component.ts

```
import { Component } from '@angular/core';

@Component({
  selector: 'app',
  template: `
  <div class="container">
  <app-header></app-header>
      <div class="m-t-1">
        <router-outlet></router-outlet>
      </div>
  </div>
  `
})
export class AppComponent {

}
```



home/home.component.ts

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

@Component({
  selector: 'app-home',
  template: `
  <div class="container">
    <div class="row">
    <div class="col-md-2">
      <p><a routerLink="/intro" routerLinkActive="active">
      Introduction</a></p>
      <hr>
      <p><a routerLink="/welcome" routerLinkActive="active">
      Basic Statistics</a></p>
    </div>
    <div class="col-md-10">
      <router-outlet></router-outlet>
    </div>
  </div>
  `
})

export class HomeComponent implements OnInit {
  constructor() { }
  ngOnInit() {
  }
}
```

Router

- RouterModule.forRoot(routes, {useHash: true})
  - The way to define router for root module
  - HashLocationStrategy is adopted by indicating useHash as true
- RouterModule.forChild(routes)
  - The way to define router for feature module
- { path: 'overview', component: SearchOverviewComponent }
  - The way to link URL and component
- { path: ':ui', component: DetailViewComponent }
  - The way to pass parameter to route
- This.routeSubscription$ = this.route.params.subscribe (……
  - The way to extract parameter from activated route
  - FYI, most of the Observable instance's name is recommended to end with $.
- <router-outlet></router-outlet>
  - The area where the router will render the component.

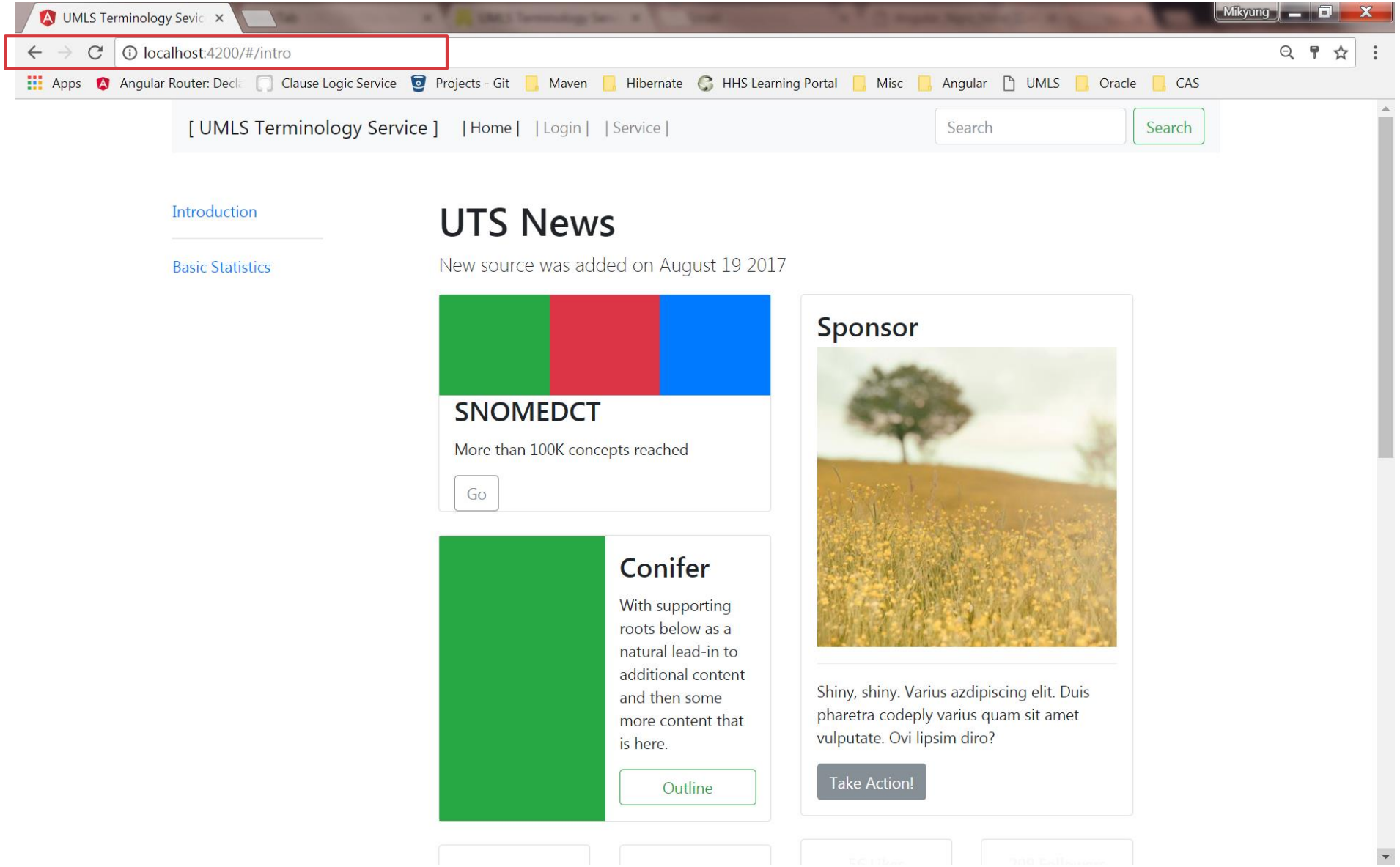


Figure 7. One example of routes. As mentioned, url before hash sign is a base URL (server) while url after hash sign is client-side locations. This is from HomeModule, of which sub path is null.

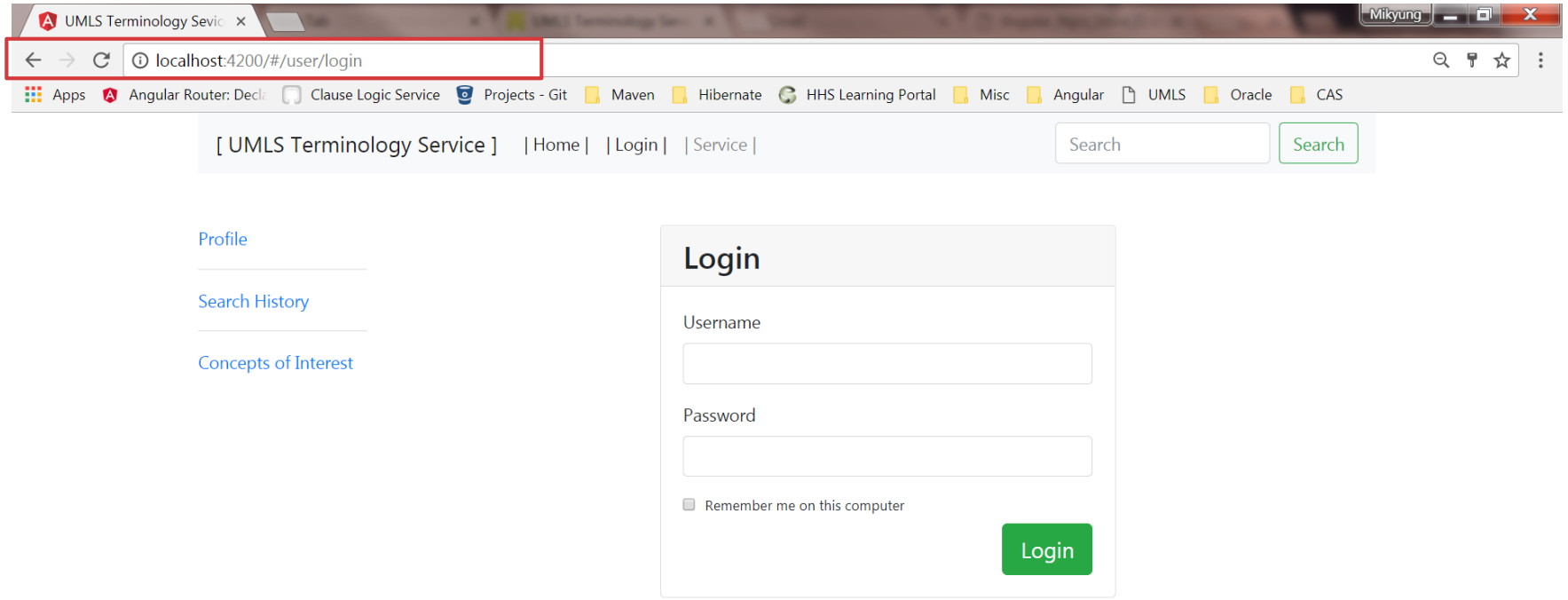


Figure 8. One example of routes. This is from UserModule, of which sub path is ‘user‘.

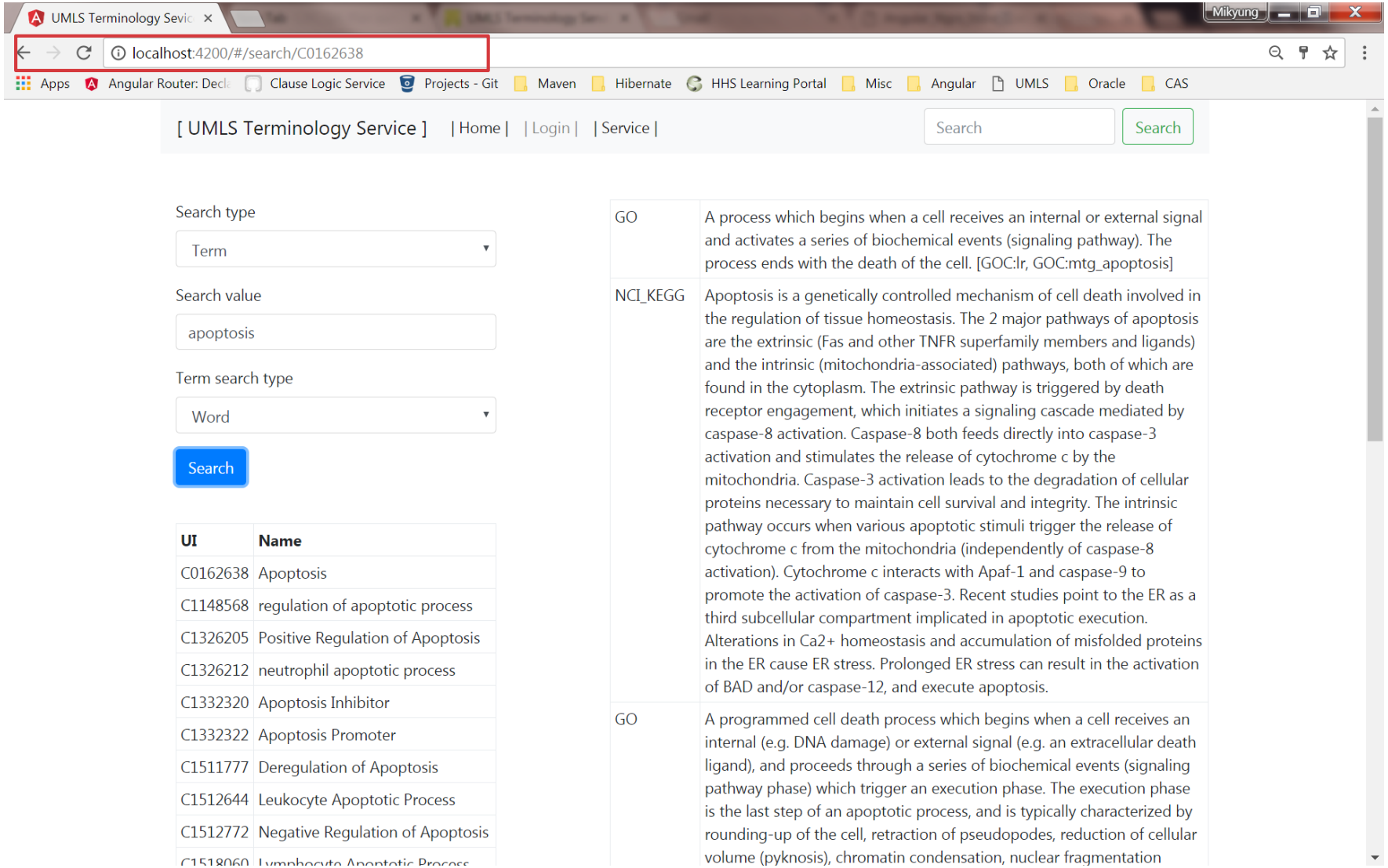


Figure 9. One example of routes. This is from SearchModule, of which sub path is 'search'. UI parameter is passed into route.

Components

- Overview
  - The main building block of an Angular application.
- Composition
  - A view – User Interface
  - A class – Logic behind the view
- Two types of components in the perspective of design pattern
  - There are two important component types, container (smart) component and presentational (dumb, child) component. The container component is more focused on how application works while the presentational component are more concerned with how looks.
  - In UTS, the search.container.component corresponds to a container component, which connects to store and retrieve application states and pass them to presentational components (app-search, concept-list) as well as updates application states by receiving actions (event) from presentational component. Updating application states is less likely to achieved by container component, it is more achieved by taking it over to service level as shown in Figure 10.

Development Design Concept I

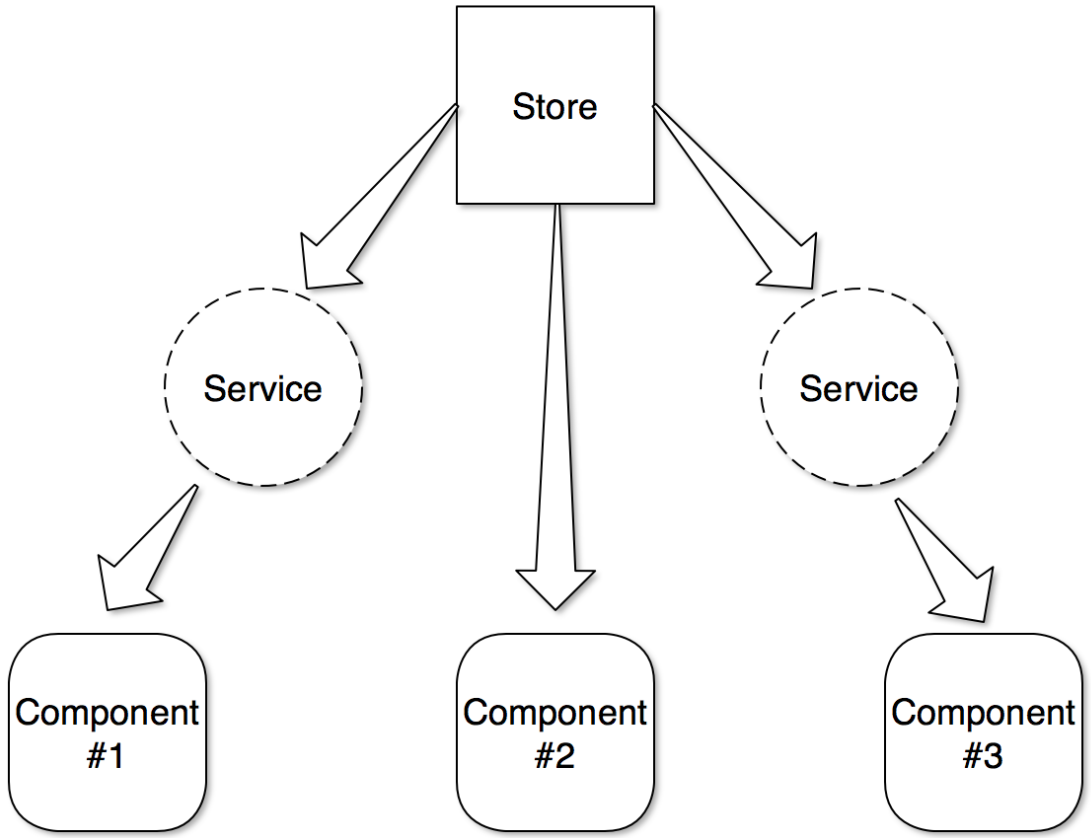


Figure 10. Store is a collection of Observable for application states. In the most of the cases, the service access to store and pass states into component.

Development Design Concept II

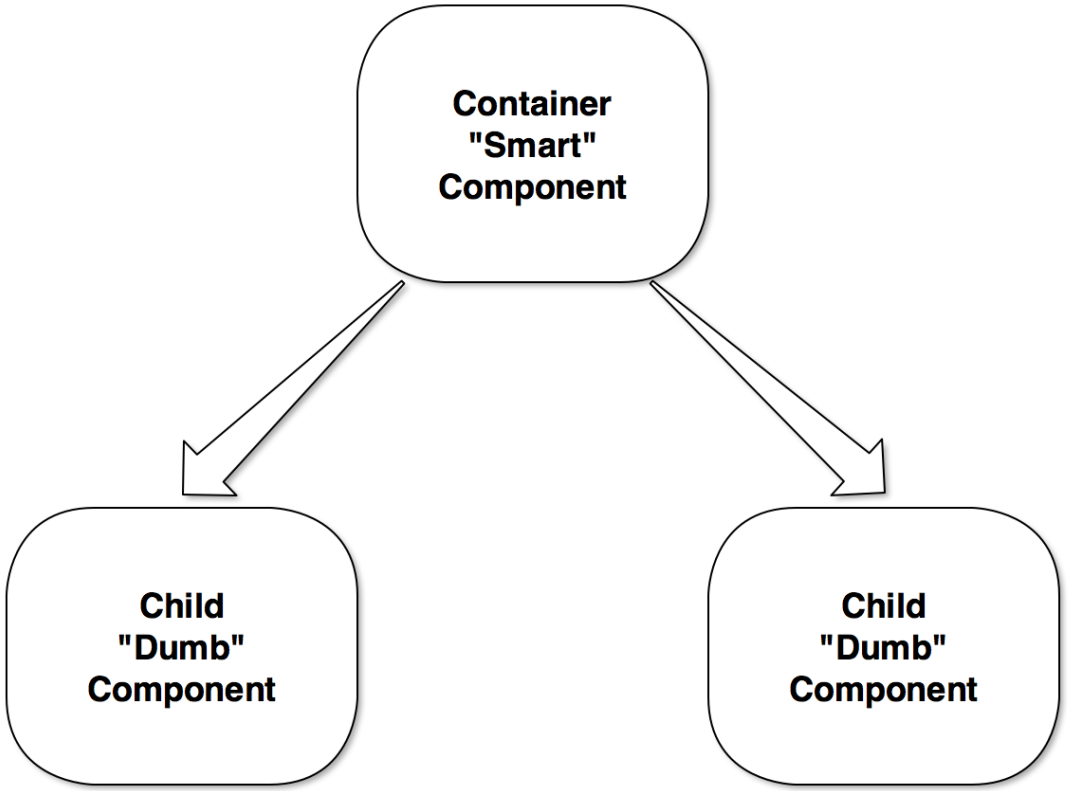


Figure 11. Depending on what is the role of component, there are two types of components, container component and child component.

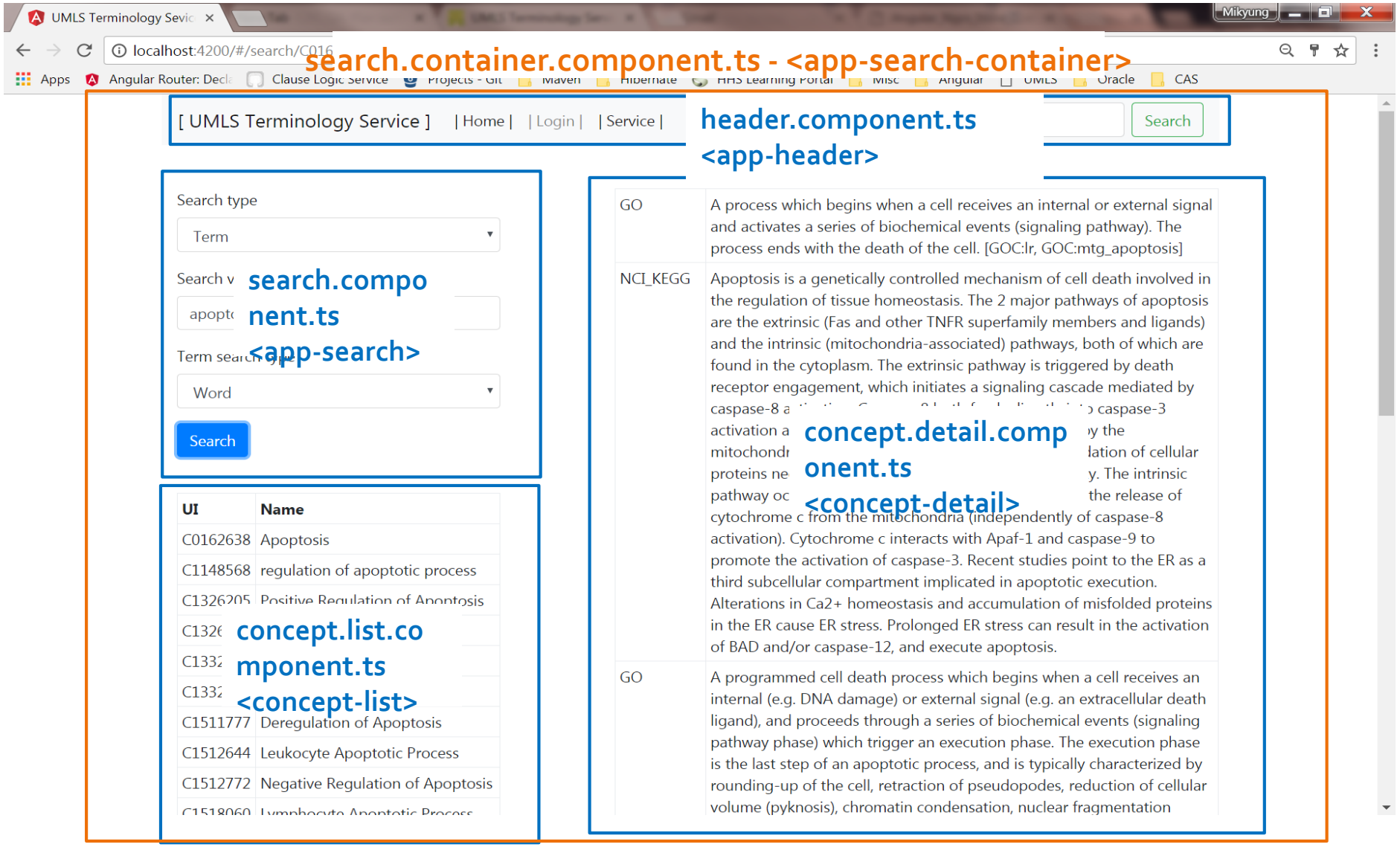


Figure 12. Example of container (smart) component and presentational (dumb) component.

search/search.container.component.ts

```
@Component({
  selector: 'app-search-container',
  template: `
  <div class="container">
    <div class="row">
    <div class="col-md-5">
      <app-search (search)="searchConcepts($event);"> </app-search>
      <concept-list
        [conceptList]="conceptsList"
        (concept)="displaySelectedConcept($event)"
      ></concept-list>
    </div>
    <div class="col-md-7">
      <router-outlet></router-outlet>
    </div>
  </div>
  `
})
```

```
searchConcepts (query: string) {
    this.store.select('termSearchReducer').subscribe((termSearch:TermSearch) => {
      this.conceptsList = termSearch.concepts;
      this.wholeconceptsList = termSearch.whole_concepts;
      this.selectedUI = termSearch.selectedUI;
    });
  }
}
```

core/services/search.service.ts

```
getConcepts( category:string, value:string, release:string, type:string ): Observable<ConceptBit[]> {
  return this.authService.getServiceTicket().switchMap ( (sticket:string) => {
    let api_url: string = '';
    api_url = BASE_API_URL + TERM_SEARCH_API_URL + value + '&ticket=' + sticket;

    return this.http.get(api_url)
    .map((responseData) => {
        this.jsonBase = <JSONBase>responseData.json();
        let conceptBase = <ConceptsBase>this.jsonBase.result;
        let conceptsList: Array<ConceptBit> = [];
        for(let i in conceptBase.results) {
          let conceptBit = <ConceptBit>conceptBase.results[i];
          conceptsList.push(conceptBit);
        }
        this.store.dispatch(this.termSearchActions.loadSearchResult(conceptsList, conceptsList[0].ui));
        this.store.dispatch(this.termSearchActions.addSearchResult(conceptsList));
        return conceptsList;
      }
    )
  });
}
```

Components

- @Component
  - Decorator to define component
- Selector
  - Identifier (element tag) of this component when adding this component into view
  - Within html file, it is added in the way <app-search> </app-search>
- Styles (styleUrls)
  - Add styles to a component
  - Can add a separate style file by using styleUrls
- templateUrl (template)
  - Define html view to a component
  - Either using inline template (template) or using separate file (templateUrl)

Components

- <app-search (search)="searchConcepts($event);"> </app-search>
  - The container(parent) component, search.container.component, includes search.component (app-search) as child (presentational) component.
  - (search) indicates that parent component receives output from child component. In the child component, there should be output named 'search' which is same name inside parentheses of parent component like @Output() search = new EventEmitter<String>(). This is a subject more relevant with component communications.

- <concept-list [conceptList]="conceptsList" (concept)="displaySelectedConcept($event)"> </concept-list>
  - The container(parent) component, search.container.component includes app-concept.list.component (concept-list) as child (presentational) components.
  - (concept) indicates that parent component receives output from child component. The child component should have output named 'concept' which is same name inside parentheses of parent component like @Output() concept = new EventEmitter<String>().
  - [conceptList] indicates that parent component send its property into child component as input. The child component should have input named 'conceptList' like @Input() conceptList: Array<Concept>. The "conceptsList" is a property of parent component.

Components

- this.store.select('termSearchReducer').subscribe(….
  - In terms of design pattern, the container (parent) component, search.container.component, receives the application states from store by selecting proper reducer. Those application states are passed to presentational (child) component.
  - FYI, a container component is not equal to a parent component. If the component (A) is located in the higher level compared to a certain component (B) in terms of components hierarchy. A is parent of B regardless of component's role.
- this.store.dispatch(this.termSearchActions.loadSearchResult….
  - The way to dispatch action to reducer.

State management

- Overview
  - To maintain consistent application state.
  - It is composed of 3 core concepts such as store, action and reducer.
  - Store - Observable collections of application states data.
  - Action – User events, eventually change the states.
  - Reducer – Function (pure) accepting previous state and action and returning new state.
- In UTS, @ngrx/store library is employed.

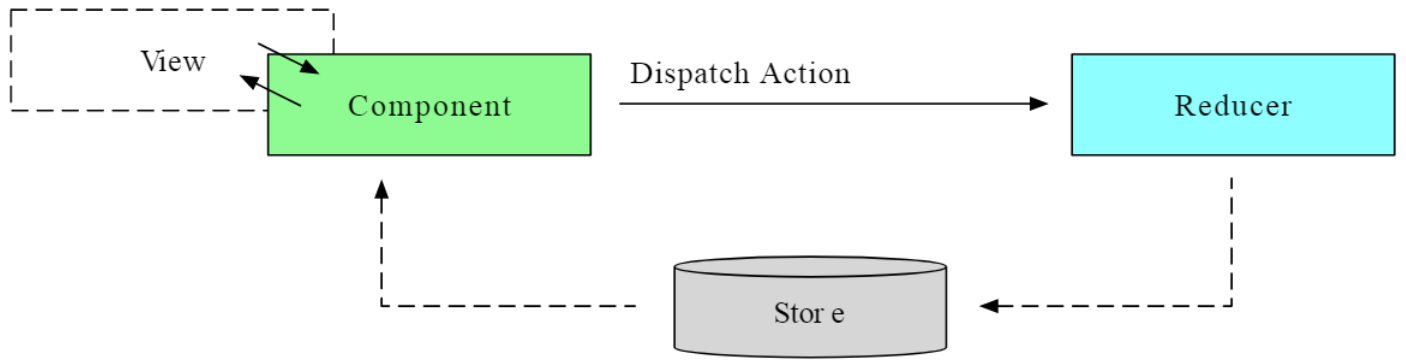


Figure 13. The way the @ngrx/store works

core/store/app-store.ts

```ts
import { Observable } from 'rxjs/Observable';
import { UserProfile } from './user-profile.reducer';
import { TermSearch } from './term-search.reducer';

export interface UTSAppState {
  user: UserProfile;
  termSearch: TermSearch;
};
```

core/store/index.ts

```ts
import { Observable } from 'rxjs/Observable';
import { NgModule } from '@angular/core';
import { StoreModule, combineReducers } from '@ngrx/store';
import { StoreDevtoolsModule } from '@ngrx/store-devtools';
import { UserProfileActions } from './user-profile.actions';
import { userReducer } from './user-profile.reducer';
import { TermSearchActions } from './term-search.actions';
import { termSearchReducer } from './term-search.reducer';
import { UTSAppState } from './app-store';

@NgModule({
  imports: [
    StoreModule.provideStore({userReducer, termSearchReducer}),
  ],
  declarations: [],
  exports: [],
  providers: [UserProfileActions, TermSearchActions]
})
export class CoreStoreModule {};
```

core/store/term-reducer.ts

```
import { Observable } from 'rxjs/Observable';
import { Action } from '@ngrx/store';
import { TermSearchActions } from './term-search.actions';
import { ConceptBit } from '../models';

export * from './term-search.actions';

export interface TermSearch {
  isSearching: boolean;
  query: string;
  concepts: ConceptBit[];
  whole_concepts: ConceptBit[];
  selectedUI: string;
};

const initialState: TermSearch = {
  isSearching: false,
  query: "",
  concepts: [],
  whole_concepts: [],
  selectedUI: ""
};

export function termSearchReducer
      (state: TermSearch = initialState, action: Action): TermSearch {
  switch (action.type) {
    case TermSearchActions.SEARCH_STARTED:
      return Object.assign( {}, state, { query:action.payload, isSearching:true });
    case TermSearchActions.LOAD_SEARCH_RESULT:
      return Object.assign( {}, state, { concepts:action.payload.conceptlist,
        isSearching:false, selectedUI: action.payload.selectedConcept });
    case TermSearchActions.ADD_SEARCH_RESULT:
      return Object.assign( {}, state, { whole_concepts:
        addConcepts(state.whole_concepts, action.payload) });
    default:
    return Object.assign({}, initialState, state);
  }
};
```

core/store/term-actions.ts

```
import { Injectable } from '@angular/core';
import { Action } from '@ngrx/store';
import { JSONBase, ConceptBit, ConceptsBase } from '../models';

export class TermSearchActions {

  static SEARCH_STARTED = '[TermSearch] SEARCH_STARTED';
  static SEARCH_QUERY = '[TermSearch] SEARCH_QUERY';
  static LOAD_SEARCH_RESULT = '[TermSearch] LOAD_SEARCH_RESULT';
  static ADD_SEARCH_RESULT = '[TermSearch] ADD_SEARCH_RESULT';

  searchStarted (id: string, token: string, hasLogin: boolean) : Action {
    return { type: TermSearchActions.SEARCH_STARTED, payload: {
    'id':id, 'access_token':token, 'is_authenticated':hasLogin } };
  }

  searchQuery(query: string) : Action {
    return { type: TermSearchActions.SEARCH_QUERY, payload: query };
  }

  loadSearchResult( concepts: ConceptBit[], firstUI: string ) : Action {
    return { type: TermSearchActions.LOAD_SEARCH_RESULT, payload: {
    conceptlist: concepts, selectedConcept: firstUI } }
  }

  addSearchResult( concepts: ConceptBit[] ) : Action {
    return { type: TermSearchActions.ADD_SEARCH_RESULT, payload: concepts }
  }
}
```

Future work

- Configure dynamic environment and constant variables.
- Bundling and deployment using WebPack – almost done.
- State management – integrate @ngrx/effects, implement proper selectors class.
- Change detection – OnPush strategy