CREATING NEW APPLICATION:

This is a manual for creating a new application to Dashboard system.

Components of Application:

Main component: {Name}ApplicationComponent

- Component that will be loaded in the dashboard widget. It needs to be able to appear only in the given space and must be able to adjust to different sizes of widget
- Must extend class ApplicationBaseComponent, which defines two important Input() attributes:

`@Input() state;` current state of the dashboard (normal or edit)
`@Input() widget;` widget object: account, position, size

Popup component: {Name}PopupComponent

- Component represents popup window. Must implement bootsrap popup and be able to be triggered as one. It needs a dependency injection of NgbActiveModal (https://ng-bootstrap.github.io/#/components/modal/api)
- Must extend class PopupBaseComponent, which defines one Input() attributes:

`@Input() widget;` widget object: account, position, size

Others:
Application can implement other helping components, but these components can be used only inside popup or main component.

Mappings of the components:

All applications need to have mapped string name to component (for both Main and Popup components), because of dynamic loading of applications. If they are not mapped, they will not appear on the dashboard - ErrorApplication component will appear on their place. The mappings are added to MAPPINGS variable:
`MAPPINGS['{name}-application'] = {Name}ApplicationComponent;`
`MAPPINGS['{name}-popup'] = {Name}PopupComponent;`

These mappings could be defined anywhere in the code, but are recommended right after component definition.

Accounts:

Account of application is in the hands of application to manage. Access to these accounts are provided with ApplicationManagerService. Apps can create or retrieve account and edit Widget which stores account ID. Following are the definitions of objects Account and Widget:

```
export interface AccountInterface {
  id?: number; // identification
  owner?: Number; // Account owner
```

```
  type: String; // Defined by app
  name: String; // Type, but in readable form for user
  token: String; // Access / Refresh token
  info?: String; // Some information for user
}

export interface WidgetInterface {
  id?: Number; // Identification
  dashboard: Number; // Dashboard na ktorom sa nachádza
  app: Number; // Application
  account?: Number; // Account
  position_x: number; // Dashboard position X
  position_y: number; // Dashboard position Y
  size_x: Number; // Width
  size_y: Number; // Height
}
```

Interface of ApplicationManagerService:

```
getAccount(widget: WidgetInterface): Observable<AccountInterf
ace> // returns account from the widget
saveAccount(account: AccountInterface): Observable<AccountInt
erface> // saves new account instance
updateWidget(widget: WidgetInterface): Observable<WidgetInter
face> // updates widget attributes
```

Custom Backend:

App can have own backend Module in the form of Django application which is added to Django backend:

1. Create new Django app
2. Implement it and create file urls.py with the definition of URLs used for API. This definitions will be automatically added to project URLs.

Good practice:

- Documentation comments according to standard of Compodoc https://compodoc.github.io/compodoc/
- Use of Services for logic of application
- UI components from ng-bootstrap (https://ng-bootstrap.github.io/#/home) or MDB UI framework (https://mdbootstrap.com/), already provided by default module.
- Error handling with Alert component provided by application:

Typescript:

```
public alerts: AlertInterface[];
```

Template:

```
<alert [alerts]="alerts"></alert>
```

Can have own definition of custom Alert with interface AlertInterface, example:

```
const customAlertMessage: AlertInterface = {
 id: 1, // used for identification
 type: 'success', // success, danger, warning
 message: 'My custom alert message'
};
```

Templates for client components and their mappings:

Following are templates for components of the application. Just replace ‘Name’ with the name of your app, and you’re ready to go.

Main component:

```
import {Component} from '@angular/core';
import {MAPPINGS} from '../../components/main.components/appl
ication.component';
import {ApplicationBaseComponent} from '../../components/main
.components/application-base.component';

@Component({
  selector: 'name-application',
  templateUrl: './name-application.component.html',
})

export class NameApplicationComponent extends ApplicationBase
Component {
```

```
    constructor(private appManagerService: ApplicationManagerS
ervice) {
        super();
    }
}
MAPPINGS['name-application'] = NameApplicationComponent;
```

Popup component:

```
import {Component} from '@angular/core';
import {MAPPINGS} from '../../components/main.components/appl
ication.component';
import {NgbActiveModal} from '@ng-bootstrap/ng-bootstrap';
import {PopupBaseComponent} from '../../components/main.compo
nents/popup-base.component';

@Component({
  selector: 'name-popup',
  templateUrl: './name-popup.component.html',
})
export class NamePopupComponent extends PopupBaseComponent {
  constructor(public activeModal: NgbActiveModal
              private appManagerService: ApplicationManagerSe
rvice) {
    super();
  }
}
```

```
MAPPINGS['name-popup'] = NamePopupComponent;
```

Adding new app to project:

1. Add application components to project to `/src/app/applications/{name}-application/` folder
2. Import all components to module. Add Main and Popup component also to Entry components
3. Add new application to database:

> *name: {name}*
>
> *description: something brief about the app (functions, etc.)*
>
> *allows_small_sizes: True or False, does the app supports smaller widget sizes*
>
> *has_backend: True or False, if the backend exists*

Note: {name} of the app must correspond to mappings of components in MAPPINGS variable

If there is also backend module present set has_backend variable to True. If the app uses any different or none backend, set it to False.