Layout Service

The layout service provides the means to display and swap in outer-frames of an app.

This service is given two things:

- Layout Anchor - that tells where to inject outer frame layouts into the DOM
- Layout Choices - an array list of available outer frames, organized by name

Code Location

This service, its directive and model, are currently located in the common-lib library of the Backups UI workspace project.

That library will eventually get moved to its own monorepo, and independently built, for app consumption.

Usage

Here's steps to use the layout service.

App.Config Updates

In the app.config.ts component of your app, you need to do the following:

Import the layout service and choices array type:

```
import { LayoutChoices } from './layout-control/model/ILayoutChoice';
import { LayoutService } from './layout-control/services/layout.service';
```

Inject the layout service into your app.config.ts, like this:

```
const _layoutsvc = inject(LayoutService);
```

If your app uses an initialization function, provided by provideAppInitializer, update your `initializeApp` function to create a list of layouts and send them to the layout service for use.

Here's a snippet that will do so, and can be added to your `initializeApp` function:

```
// This method is called when the app first starts up.
// It will load any initial config, and setup anything required.
export function initializeApp(): Promise<boolean>
//export function initializeApp(): () => Promise<boolean>
{
  const _appconfig = inject(AppconfigService);
  const _ctsvc = inject(ColorThemingService);
  const _layoutsvc = inject(LayoutService);
  const env = inject(ENVIRONMENT_INJTOKEN);

  console.log("AppModule-" + "_" + ":initializeApp - triggered.");

  // To ensure that dependencies are stood up and ready, we must wrap our logic in a lambda, instead of naked execution...
  return (async (): Promise<boolean> =>
  {
    console.log("AppModule-" + "_" + ":initializeApp - inside startup promise...");

    ... Do other startup things...

    // Define layouts...
    {
      console.log("AppModule-" + "_" + ":initializeApp - Defining layout choices...");

      // Create the layout choice listing...
      let lc:LayoutChoices =
      [
        {
          name: 'main',
          component: AppLayoutComponent,
          requireLogin: false
        },
        {
          name: 'error',
          component: Error404PageComponent,
          requireLogin: false
        },
        {
```

```
          name: 'test',
          component: TestPageComponent,
          requireLogin: false
        },
        {
          name: 'login',
          component: LoginPageComponent,
          requireLogin: false
        }
      ];

      // Pass it to the layout service...
      if(_layoutsvc.DefineLayoutChoices(lc) === false)
      {
        console.error("AppModule-" + "_" + ":initializeApp - Failed to set layout choices.");

        return false;
      }
    }

    // Report success...
    console.log("AppModule-" + "_" + ":initializeApp - returned.");
    return true;
  })(); // Notice the () at the end - this immediately invokes the async lambda
}
```

App.Component Updates

For the layout service to swap outer frames of your app, it must have an anchor point for insertion.

For an app with multiple outer frames (for login, app-layout, testing, error, etc), the ideal spot for the outer frame anchor is in App.Component.

Here are changes to incorporate in the pieces of App.Component:

app.component.html:

```
<div class="fill-parent">
  <!-- Below is the a dynamic insertion point for anchoring components to the empty page -->
  <ng-template EmptyFrameAnchor></ng-template>
</div>
```

The above html template include the anchor directive on an ng-template element.
And, it contains a div that fills the parent width and height.

app.component.ts:

```
import { Component, Inject, ViewChild } from '@angular/core';
import { OnInit, AfterViewInit, AfterContentInit, OnDestroy } from '@angular/core';

import { mergeMap, takeUntil } from 'rxjs/operators';
import { Subject } from 'rxjs';

// Import things we need, for swapping layouts...
import { EmptyFrameAnchorDirective } from './layout-control/directives/emptyframe-anchor.directive';
import { LayoutService } from './layout-control/services/layout.service';

// Import security service.
// This module subscribes to the global login state, so that it can flip to the Login page if no one is logged in.
import { IRuntimeSecurityService } from 'lib-oga-webui-sharedkernel';
import { RUNTIMESECURITYSERVICE_INJTOKEN } from 'lib-oga-webui-sharedkernel';



@Component({
  selector: 'app-root',
  imports:
  [
    EmptyFrameAnchorDirective
  ],
  templateUrl: './app.component.html',
  styleUrl: './app.component.scss'
})
export class AppComponent implements OnInit, AfterViewInit {
  // Define an anchor reference for where layouts will be pushed...
  @ViewChild(EmptyFrameAnchorDirective, {static: true}) domanchor!: EmptyFrameAnchorDirective;

  constructor(private _layoutsvc:LayoutService,
              @Inject(RUNTIMESECURITYSERVICE_INJTOKEN) private _securityService: IRuntimeSecurityService)
  {
  }

  ngOnInit(): void
```

```
  {
    this.#_Initialize_DOMAnchor();

    // Check if the security service can notify us of login events...
    if(this._securityService.Has_Privilege_to_Open_App$ == null)
    {
      console.error("AppComponent-" + this.instanceId + ":ngOnInit - security service instance is null.");
      throw new Error("domanchor is undefined");
    }

    // Setup an observable that will take action on login event changes...
    // This observable will tell the profile service to load the appropriate component based on user login state.
    // It will remain active until this component is destroyed.
    this._securityService.Has_Privilege_to_Open_App$
      .pipe(
        takeUntil(this.destroySubject),
        mergeMap(has_privilege_to_open_app => Promise.resolve(null).then(() =>
        {
          console.log("AppComponent-" + this.instanceId + ":ngOnInit - observed has_privilege_to_open_app
change state to: " +
                    has_privilege_to_open_app);

          console.log("AppComponent-" + this.instanceId + ":ngOnInit - " +
                    "telling the layout service to swap outer frames for the user change event...");
          this.SwapBasePageforLoginEvent(has_privilege_to_open_app);

          console.log("AppComponent-" + this.instanceId + ":ngOnInit - " +
                    "SwapBasePageforLoginEvent swapped in the desired frame for the user security event.");
        })
        )
      )
      .subscribe();
  }

  ngAfterViewInit() {
    console.log("AppComponent-" + this.instanceId + ":ngAfterViewInit - triggered.");
  }

  // Private method that locates the outer frame anchor and sends it to the layout service.
  #_Initialize_DOMAnchor()
```

```
  {
    // Throw an exception of the layout dom anchor can't be found...
    if (!this.domanchor) {
      console.error("AppComponent-" + this.instanceId + ":_Initialize_DOMAnchor - Cannot locate layout DOM
anchor.");
      throw new Error("domanchor is undefined");
    }

    // Give the layout anchor point to the layout service...
    this._layoutsvc.SetLayoutAnchor(this.domanchor);
  }

  ngOnDestroy() {
    console.log("AppComponent-" + this.instanceId + ":ngOnDestroy - triggered.");

    // Unhook and close out subscriptions to the log in observable...
    this.destroySubject.next();
    this.destroySubject.complete();
  }



}
```

The above logic does several things:

- It locates the anchor point, where outer frames are swapped into the DOM.
- Sends the outer frame anchor point to the layout service, for use.
- Subscribes to user login events, to be notified when a user logs in and out.
-

Revision #1
Created 27 March 2025 01:41:05 by glwhite
Updated 24 April 2025 04:58:33 by glwhite