ANGULAR



We have used Angular for developing UI. Angular is designed to make updating as easy as possible, so that we can take advantage of the latest developments with a minimum of effort.

Angular is a development platform, built on [TypeScript](https://www.typescriptlang.org/). As a platform, Angular includes:

* A component-based framework for building scalable web applications
* A collection of well-integrated libraries that cover a wide variety of features, including routing, forms management, client-server communication, and more
* A suite of developer tools to help you develop, build, test, and update your code

ARCHITECTURE –

The architecture of an Angular application relies on certain fundamental concepts. The basic building blocks of the Angular framework are Angular components that are organized into NgModules. NgModules collect related code into functional sets; an Angular application is defined by a set of NgModules. An application always has at least a root module that enables bootstrapping, and typically has many more feature modules.

* Components define views, which are sets of screen elements that Angular can choose among and modify according to your program logic and data.
* Components use services, which provide specific functionality not directly related to views. Service providers can be injected into components as dependencies, making your code modular, reusable, and efficient.

Modules, components, and services are classes that use decorators. These decorators mark their type and provide metadata that tells Angular how to use them.

* The metadata for a component class associates it with a template that defines a view. A template combines ordinary HTML with Angular directives and binding markup that allow Angular to modify the HTML before rendering it for display.
* The metadata for a service class provides the information Angular needs to make it available to components through dependency injection (DI).

An application's components typically define many views, arranged hierarchically. Angular provides the [Router](https://angular.io/api/router/Router) service to help you define navigation paths among views. The router provides sophisticated in-browser navigational capabilities.



MODULE

Angular *NgModules* differ from and complement JavaScript (ES2015) modules. An NgModule declares a compilation context for a set of components that is dedicated to an application domain, a workflow, or a closely related set of capabilities. An NgModule can associate its components with related code, such as services, to form functional units.

Every Angular application has a *root module*, conventionally named AppModule, which provides the bootstrap mechanism that launches the application. An application typically contains many functional modules.

Like JavaScript modules, NgModules can import functionality from other NgModules, and allow their own functionality to be exported and used by other NgModules. For example, to use the router service in your app, you import the [Router](https://angular.io/api/router/Router) NgModule.

Organizing your code into distinct functional modules helps in managing development of complex applications, and in designing for reusability. In addition, this technique lets you take advantage of *lazy-loading*—that is, loading modules on demand—to minimize the amount of code that needs to be loaded at startup.

COMPONENTS

Every Angular application has at least one component, the *root component* that connects a component hierarchy with the page document object model (DOM). Each component defines a class that contains application data and logic and is associated with an HTML *template* that defines a view to be displayed in a target environment.

The @[Component](https://angular.io/api/core/Component)() decorator identifies the class immediately below it as a component, and provides the template and related component-specific metadata.

**TEMPLATES, DIRECTIVES & DATA BINDING**

A template combines HTML with Angular markup that can modify HTML elements before they are displayed. Template *directives* provide program logic, and *binding markup* connects your application data and the DOM. There are two types of data binding:

* *Event binding* lets your application respond to user input in the target environment by updating your application data.
* *Property binding* lets you interpolate values that are computed from your application data into the HTML.

**SERVICES & DEPENDENCY INJECTION**

* For data or logic that isn't associated with a specific view, and that you want to share across components, you create a *service* class. A service class definition is immediately preceded by the @[Injectable](https://angular.io/api/core/Injectable)() decorator. The decorator provides the metadata that allows other providers to be injected as dependencies into your class.
* *Dependency injection* (DI) lets you keep your component classes lean and efficient. They don't fetch data from the server, validate user input, or log directly to the console; they delegate such tasks to services.

ROUTING

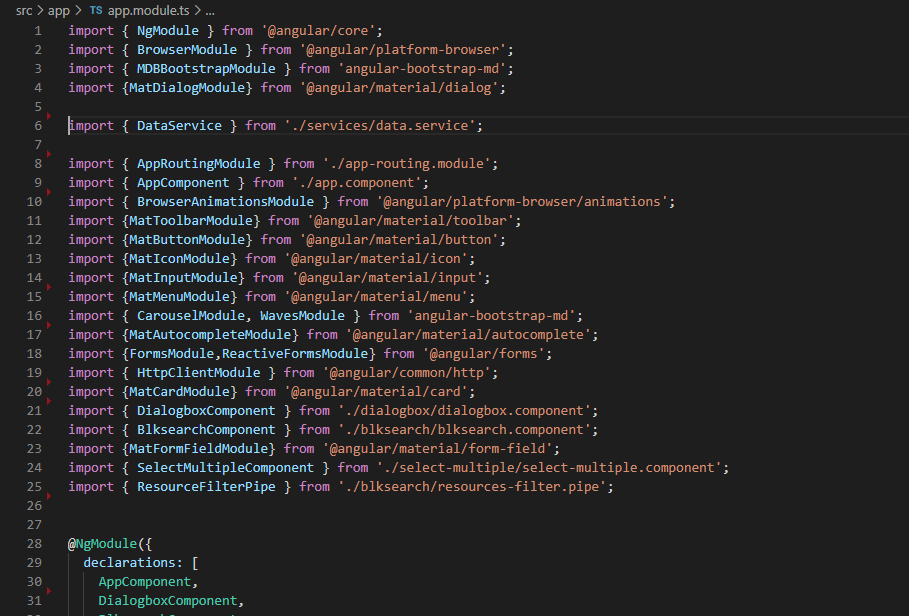
The Angular [Router](https://angular.io/api/router/Router) NgModule provides a service that lets you define a navigation path among the different application states and view hierarchies in your application. It is modeled on the familiar browser navigation conventions:

* Enter a URL in the address bar and the browser navigates to a corresponding page.
* Click links on the page and the browser navigates to a new page.
* Click the browser's back and forward buttons and the browser navigates backward and forward through the history of pages you've seen.

The router maps URL-like paths to views instead of pages. When a user performs an action, such as clicking a link, that would load a new page in the browser, the router intercepts the browser's behavior, and shows or hides view hierarchies.

APPROACH –

The app.module.ts is the main Module which contains all the declarations and imports used for developing our application.

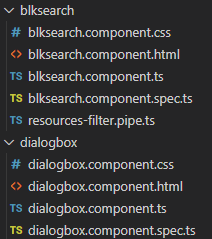


Code Snippet of app.module.ts

We have used 2 Components to build our Application. They are

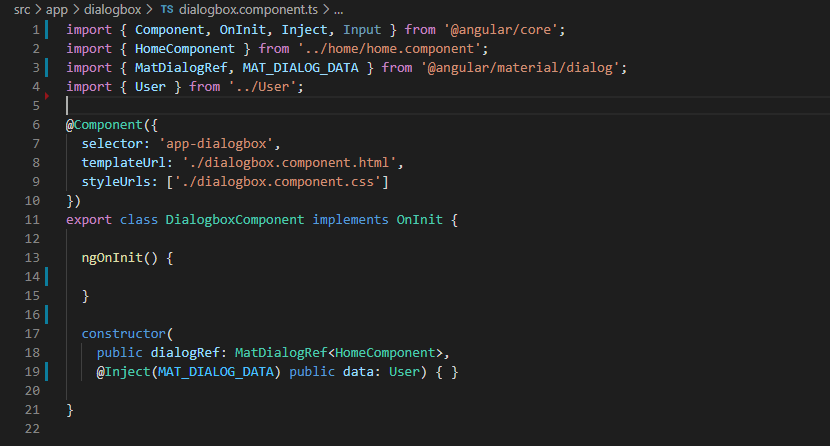
1)blksearch

2)dialogbox

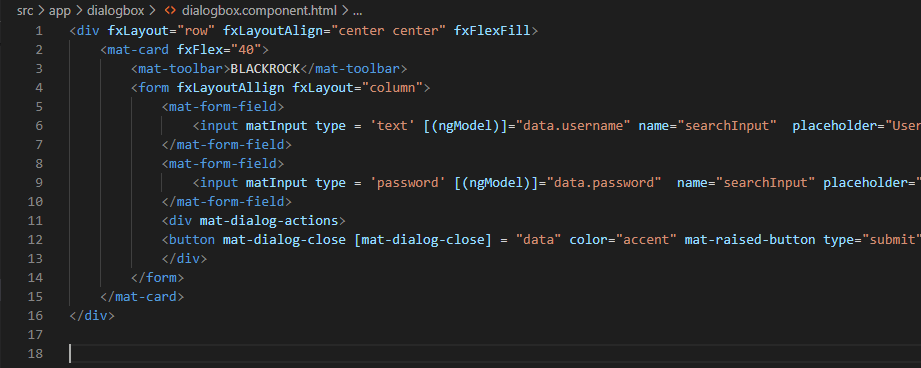


The role of these components -

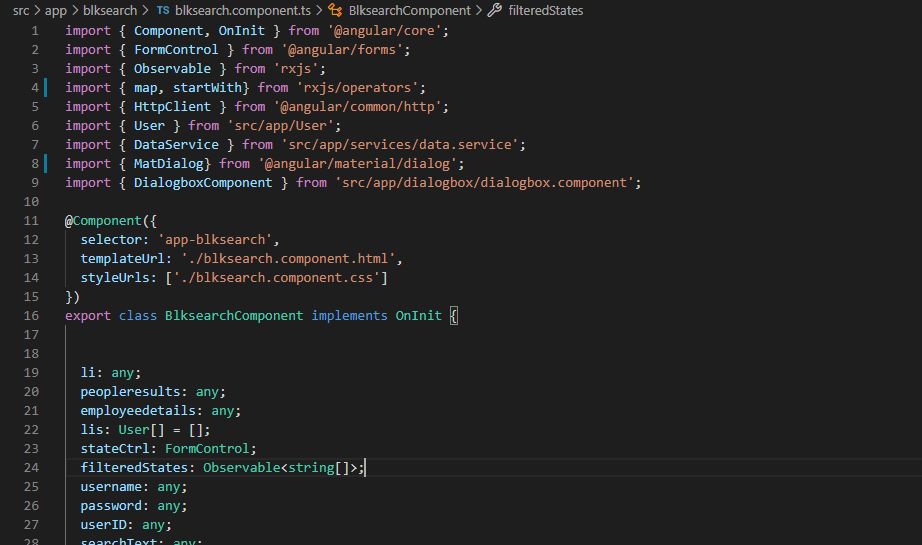
* blksearch: This is the main component which provides the core functionality to our application. Various functions like filtering, sorting, opening of dialog box etc are performed in this component. This component on default shows the relevant search results of the user and on searching shows the search results i.e Boost Results, Normal Results & Employee details based on the query.
* dialogbox: This is a supportive component used for collecting login information of the user.



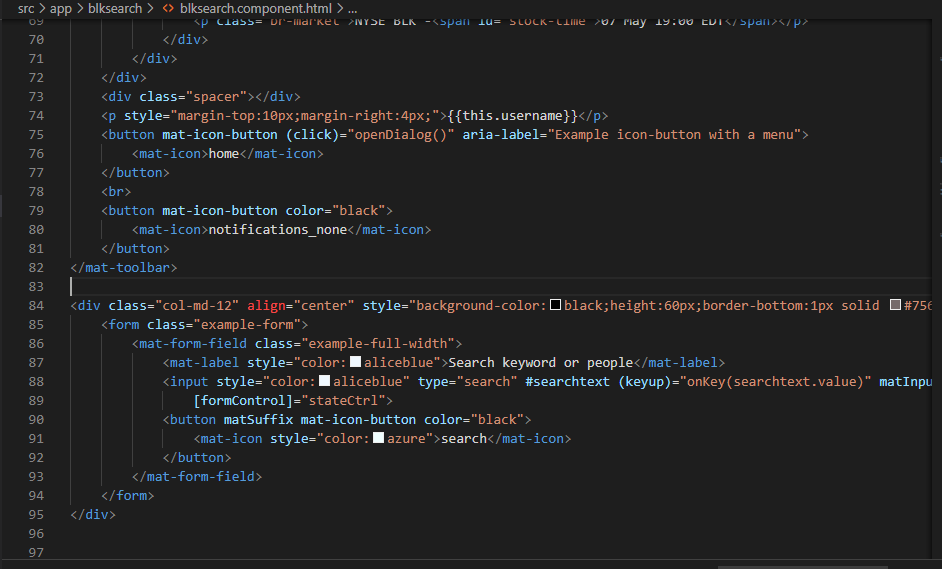
Code Snippet of dialogbox.component.ts



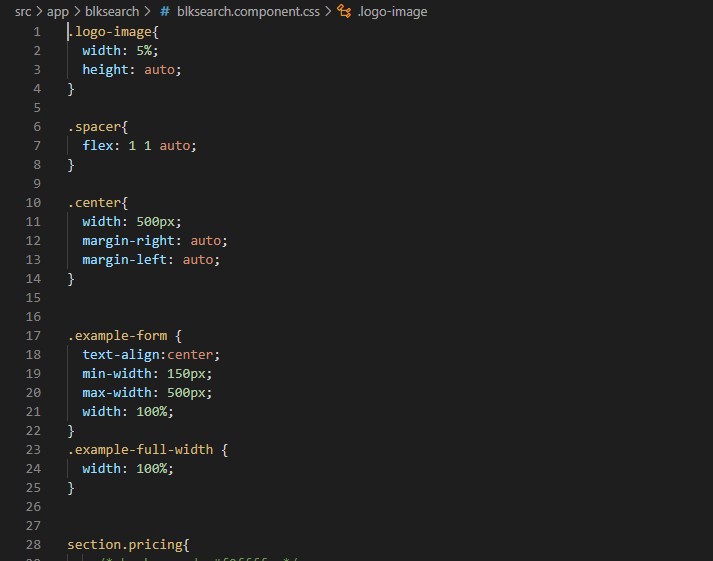
Code Snippet of dialogbox.component.html



Code Snippet of blksearch.component.ts

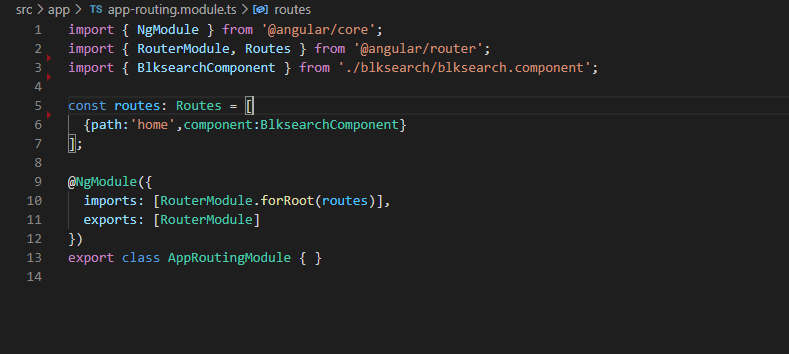


Code Snippet of blksearch.component.html



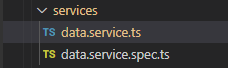
Code Snippet of blksearch.component.css

We have used Routes for easier Navigation between the components.



Code Snippet of app-routing.module.ts

We have used Services for getting the API’s –



There are 4 main API’s specified in the data.service.ts

1) getUserId(username:any) : Observable<any>

2) getSearchResults(query:string,employeeId : number) : Observable<any>

3) getRelevantSearchResults(userid:any) : Observable<any>

4)getEmployeeDetails(query:string,encodedString:string) : Observable<any>



Code snippet of data.service.ts