Services Lab

Let's say that once the user is logged in we want to display their name on each page.

Obviously, we want to have the user log in to one component. Then we could pass the user's identity around using property binding, but that's a whole lot of properties. We'd have to do it in every component. Too much work spread around too many components. Gosh, if only we could somehow share properties and methods between components!

This is where services shine! If we were to inject an AuthService into a Login Component and set its *user* property and then inject that same AuthService into other components, its *user* property could be seen in all of them. Let's work on making that happen.

1. To prepare, create a new type in the shared folder called User.

```
export type User = {
  id?: number,
  username?: string,
  password?: string,
  givenName?: string,
  familyName?: string,
  isAdmin?: boolean
}
```

2. Create a new Angular component called Login. In the class, add a user property and string properties for message, username and password.
3. In the template, create:
   - inputs for email and password,
   - a button to log them in. The button's click event should run a method called login().
   - a <div> to hold a success/failure message for their attempted login.
4. The login method will pretend to log them in and create a fake user object. This object should hold placeholder information for id, username, password, givenName, and familyName. Also have it set this.message to "Successfully logged in".
5. Now let's use the login component. Add a route to it in app.router.ts. Then add a router link to it in the menu at the top of App component.
6. Run and test. Make sure that you can get to the component and that 'logging in' (wink wink) will show a success message and set the user object's properties.

While you're still running, notice that you can navigate to the Receive Product component or the Orders Ready to Ship component. Our goal is to make the user data from Login appear in those other components.

Creating the service

7. In the Angular CLI, create a new service called Auth Service. Something like this might do the trick:

```
ng generate service auth/auth
```

8. Give that service a Boolean property called loggedIn. Initialize it to false.

9. Also give it an optional property called user.

Note that this is a super-simple service so far. It has no methods (yet).

Using the service

10. Inject the service into your Login Component.
11. In LoginComponent's ngOnInit, set its private user property to the injected service's user property. Something like this will do:

```
this.user = this._authSvc.user;
```

12. In LoginComponent's Login() method, pretend to authenticate. Set the service's loggedIn property to true:

```
this._authSvc.loggedIn = true;
this._authSvc.user = {
  // Whatever values you had in there before
};
```

Cool. It looks like the user is being logged in in the LoginComponent. Let's make sure in another component.

13. Inject the AuthService into your App Component as a **public** property.

14. Go into the template and use interpolation to display the user's given name and family name. Put this inside the navbar at the top:

```
<ul *ngIf="authSvc.loggedIn" class="navbar-nav d-flex justify-content-end">
  <a class="nav-link" [routerLink]="'account'">
    {{ authSvc.user?.givenName }} {{ authSvc.user?.familyName }}
  </a>
</ul>
```

15. Run and test. Login and then navigate to any component. Do you see your name appear? Cool! You've just shared data through a service!

Refactoring the service

We've just demonstrated that the purpose of a service is to share -- just properties so far but we can share methods as well.

16. Give the Auth Service a method called *authenticate*. Move the logic for logging in from the login component to this method. (Hint: this._authSvc.loggedIn becomes this.loggedIn and this._authSvc.user becomes this.user). These will still be hardcoded placeholder values.
17. Back in the LoginComponent, call the authenticate() method from your AuthService.
18. Run and test. Can you still log in? If so, we've moved the work of logging in from a component into the AuthService where it really belongs.

Making an HTTP call in the Service

Last step to make is more realistic. Let's use the login endpoint from our API server.

19. Edit the authenticate() method in the AuthService
20. Make sure it is receiving a username and password as parameters.

21. Make an HTTP GET call to `/login/${username}/${password}`†. The API will return an entire user object for a good username/password combination.
    Hints:
    - You can edit the database.json file in the server folder to get good username/password combinations. You can even add your own user records if you like.
    - You'll need to inject an HttpClient into the AuthService's constructor.
    - The authenticate() method should return an Observable so it can be activated in other components. ie. They subscribe when they're ready like on a button click.
    - If you still can't work it out without some more help, here's a way to do it:

```
authenticate(username?: string, password?: string): Observable<User> {
  return this._http.get<Array<User>>(`/api/login/${username}/${password}`)
    .pipe(
      map(users => users[0]),
      tap(user => this.user = user),
      tap((users) => this.loggedIn = true)
    )
}
```

22. You'll change your LoginComponent's login() method from this:

```
login() {
  this._authSvc.authenticate(this.username, this.password)
  this.message = "Successfully logged in";
}
```

to this:

```
login() {
  this._authSvc
    .authenticate(this.username, this.password)
    .subscribe({
      next: user => this.message =
        `Welcome, ${user.givenName}. You're successfully logged in`,
    });
}
```

23. Run and test. It should work for good credentials and should set the givenName and familyName from the database instead of your hardcoded values.

† The way this authentication is set up is not secure. You'll never actually make a GET call to really authenticate. In the real world it'll be a POST with the credentials in the body. But we're trying to keep this simple for you.