Authenticating a User

Paul D. Sheriff

BUSINESS SOLUTIONS ARCHITECT, FAIRWAY TECHNOLOGIES, INC.

www.fairwaytech.com psheriff@fairwaytech.com

Goals

- Create user and authorization classes
- Create login mocks
- Create security service
- Create login page

User Security Classes

Security Architecture Using Mocks

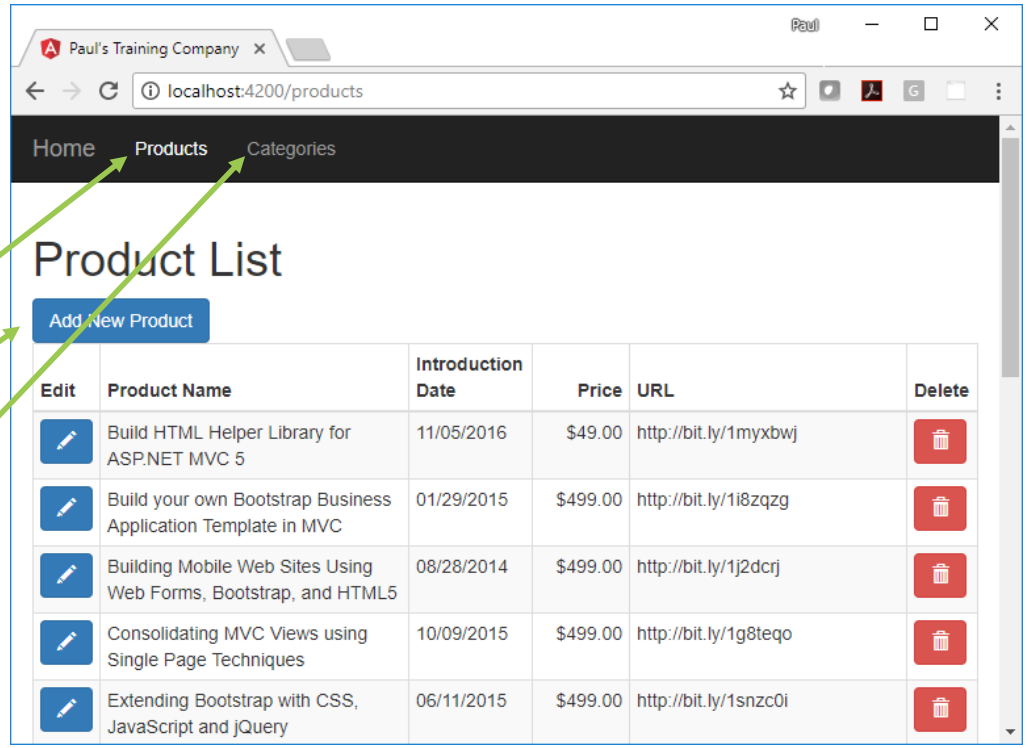
Login Page

User Object
username
password

Security Service

Array of Mock Logins

Authorization Object
userName
isAuthenticated
bearerToken
canAccessProducts
canAddProduct
canSaveProduct
canAccessCategories
canAddCategory



User Class

```
export class AppUser  {
  userName: string = "";
  password: string = "";
}
```

User Authorization Class

```
export class AppUserAuth {
  userName:          string = "";
  bearerToken:       string = "";
  isAuthenticated: boolean = false;

  canAccessProducts:   boolean = false;
  canAddProduct:       boolean = false;
  canSaveProduct:      boolean = false;
  canAccessCategories: boolean = false;
  canAddCategory:      boolean = false;
}
```

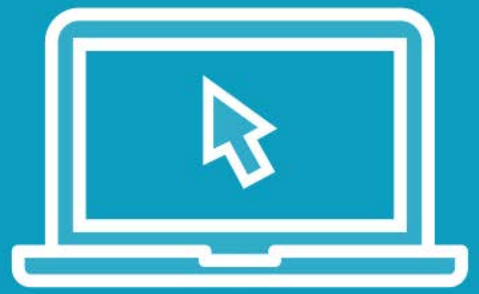
Login Mocks

```
export const LOGIN_MOCKS: AppUserAuth[] = [
{
  userName:          "PSheriff",
  bearerToken:       "abi393kdkd9393ikd",
  isAuthenticated: true,

  canAccessProducts:   true,
  canAddProduct:       true,
  canSaveProduct:      true,
  canAccessCategories: true,
  canAddCategory:      false
},

// MORE MOCKS HERE
```

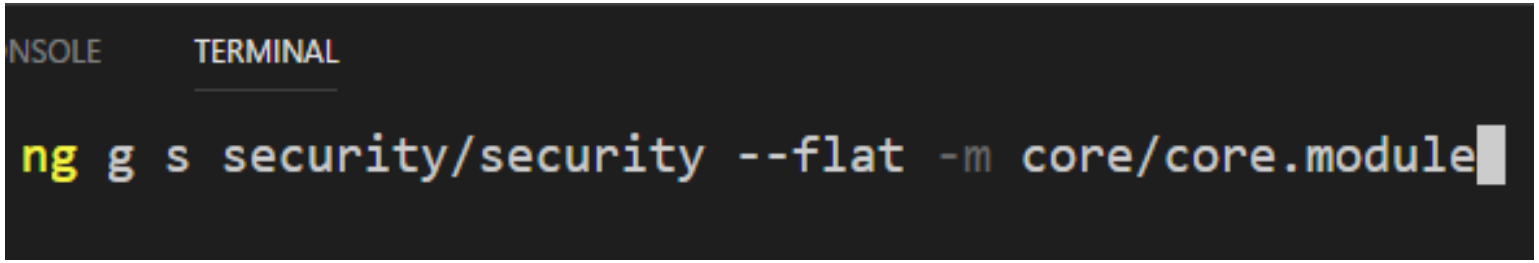
Demo



Build user classes

Security Service

Generate new service using Angular CLI



Instance of the AppUserAuth class

Verify user is valid

Set *securityObject* property

Call ResetSecurityObject method

Initialize security object properties to false or empty string

```
@Injectable()
export class SecurityService {
  securityObject: AppUserAuth = new AppUserAuth();

  constructor() { }

  login(entity: AppUser): Observable<AppUserAuth> {…
  }

  logout(): void {…
  }

  resetSecurityObject(): void {…
  }
}
```

Reset security object

Copy properties using Object.assign()

Don't recreate object

Invalidates bindings

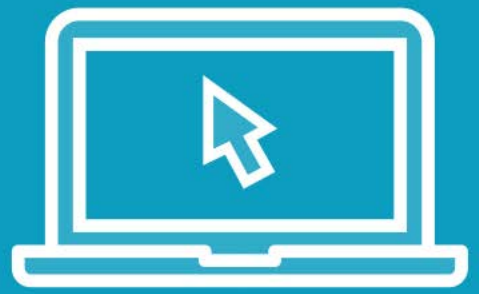
Store bearer token into local storage

```
login(entity: AppUser): Observable<AppUserAuth> {
  // Initialize security object
  this.resetSecurityObject();

  Object.assign(this.securityObject,
    LOGIN_MOCKS.find(user => user.userName.toLowerCase() ===
                             entity.userName.toLowerCase()));
  if (this.securityObject.userName !== "") {
    // Store token into local storage
    localStorage.setItem("bearerToken",
      this.securityObject.bearerToken);
  }

  return of<AppUserAuth>(this.securityObject);
}
```

Demo



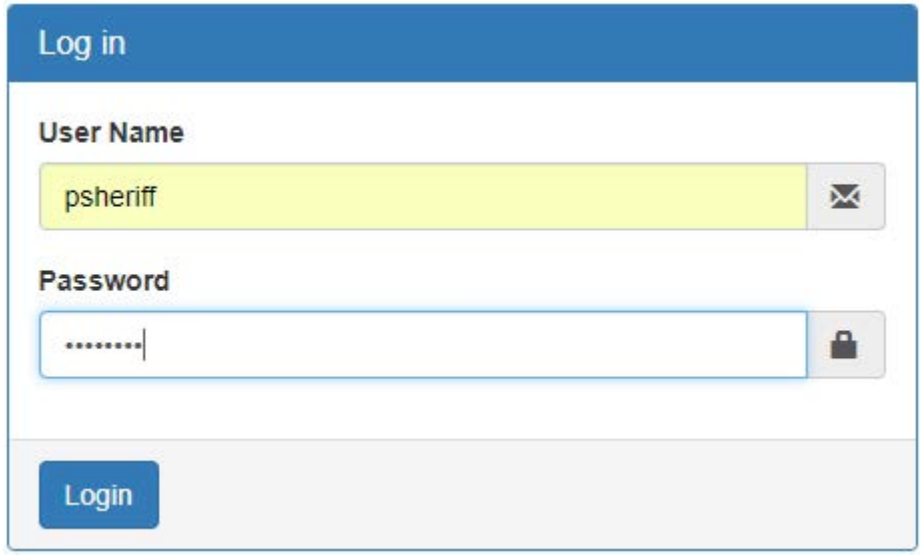
Build security service

Login Page

Create login page and component using Angular CLI

```
ng g c security/login --flat
```

Create the following page



Bind Inputs to User Object

```
<input id="userName" [(ngModel)]="user.userName" />

<input id="password" [(ngModel)]="user.password"
       type="password" />
```

Add Two Properties to Login Component

```
// Bind to inputs
user: AppUser = new AppUser();
```

```
// To display user authorization object
securityObject: AppUserAuth = null;
```

Add json Pipe to View Security Object

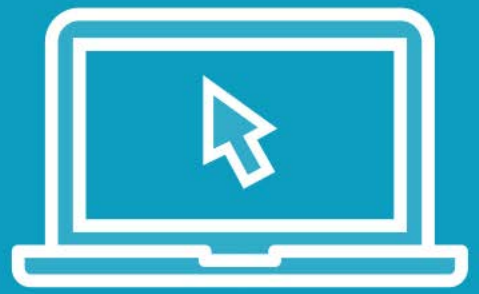
```
<div class="row">
  <div class="col-xs-12">
    <label>{{securityObject | json}}</label>
  </div>
</div>
```

Add Login Route

Open app-routing.module.ts

```
{
  path: 'login',
  component: LoginComponent
},
```

Demo



Create login page

Try out authentication

Summary

- Created user auth classes
- Created security service
- Created login page
- Viewed the authorization object

Coming up in the next module...

Use directives to secure UI elements

Add route guards to secure navigation