Model-driven Form Exercise

By: Mosh Hamedani

Implementing a custom validator

In this section, you learned how to create a custom validator. We define a method that takes a **Control** object:

```
static cannotContainSpace(control: Control){
     if (valid)
        return null;

     return { cannotContainSpace: true };
}
```

Then, we reference this method when creating a **Control** using **FormBuidler**:

```
this.form = fb.group({
     username: ['', UsernameValidators.cannotContainSpace]
});
```

If we need multiple validators, we compose them:

```
this.form = fb.group({
     username: ['', Validators.compose([
         Validators.required,
         UsernameValidators.cannotContainSpace
     ])
});
```

Model-driven Form Exercise

By: Mosh Hamedani

Comparing fields

The validator method we've implemented here takes a **Control** object, which represents a single field in the form. To compare two fields, you need to implement a validation method that takes a **ControlGroup**. Then, we can access any **Controls** in that group:

```
static myCustomValidator(group: ControlGroup){
     var control1 = group.find('control1');
     var control2 = group.find('control2');
     …
}
```

We can then pass this validator, when creating a group using **FormBuilder**:

```
this.form = fb.group({
…
}, { validator: myCustomValidator });
```

So, as the second argument to the **group()** method, we pass an object that contains "**validator**" property. We reference our custom validator method here. This method, like other validator methods can be in the component, or in a separate class (as a static method) for re-usability.

Note: If the form is not functioning properly when doing this exercise, make sure to have a look at the console in developer tools. You might have an error and be totally unaware of it!