1. **Początek – przypomnienie z wykładu.** Ściągnij projekt przykładowy z wykładu nr 5 (formularz osobowy)
    - Zainstaluj moduły dla node'a
    - Uruchom projekt
    - Zapoznaj się z projektem przykładowym:
        - Sprawdź jak powiązany widok formularza z klasą komponentu.
        - Do czego służą atrybuty [formGroup] , formControlName, [ngClass]?
        - Sprawdź jak podłączono walidację pod poszczególne kontrolki interfejsu, oraz jak jest ta walidacja reprezentowane po stronie UI.
        - Sprawdź jak się inicjalizuje kontrolki przy pomocy obiektu klasy FormBuilder

Do samodzielnego wykonania. Jako ćwiczenie do interfejsu pod polem 'email' dodaj pole telefon – pod te pole będzie podpięty walidator sprawdzający poprawność wpisanego wyrażenia (np. format dla telefonu +99(99)9999-9999, lub inne http://html5pattern.com/Phones). Jeśli numer nie jest poprawny to pole powinno mieć czerwoną ramkę, a pod nim powinna się pojawić odpowiednia informacja o błędzie walidacji.

2. **Grupa formularza.** Naszym zadaniem będzie dorobić logikę, która będzie sprawdzała czy hasła podane w formularza się zgadzają – w tym celu należy stworzyć grupę kontrolek w kodzie html interfejsu formularza – div z atrybutem 'formGroupName'

```
<div formGroupName="pswdGroup" >

          <div class="form-group" [ngClass]="{
          'has-error':(userForm.get('password').touched ||
userForm.get('password').dirty)
            &&!userForm.get('password').valid
        }">
            <label class="col-md-3 control-label">Password:</label>
            <div class="col-md-8">
              <input class="form-control" type="password"
formControlName="password">
            </div>
          </div>
          <div class="form-group">
            <label class="col-md-3 control-label">Confirm password:</label>
            <div class="col-md-8">
              <input class="form-control" type="password"
formControlName="cPassword">
            </div>
          </div>
        </div>
```

Kiedy tworzymy grupę kontrolek musimy to także uwzględnić po stronie klasy komponentu w linijkach kodu, które inicjalizują formularz – zatem pola 'password' oraz 'cPassword' zostaną przypisane do nowej grupy

```
this.userForm = this.formBuilder.group({
      firstName: ['',[Validators.required, Validators.minLength(3)]],
      lastName:  ['',[Validators.required, Validators.maxLength(50)]],
      email:  ['',[Validators.required, Validators.email]],
      username:  ['',[Validators.required, Validators.minLength(3)]],
      pswdGroup: this.formBuilder.group({
        password:  ['',[Validators.required,
Validators.pattern('^(?=.*\d)(?=.*[a-z])(?=.*[A-Z])(?!.*\s).*$')]],
        cPassword:  ['',[Validators.required]]
      })
    });
```

Sprawdź działanie formularza – niestety walidacja przestała działać – problem polega na tym, że teraz logika odpowiadająca za dodanie klasy css która zmienia ramkę kontrolki na czerwone nie potrafi znaleźć kontrolki ‘password’

```
          <div class="form-group" [ngClass]="{
          'has-error':(userForm.get('password').touched ||
userForm.get('password').dirty)
            &&!userForm.get('password').valid
        }">
```

Kontrolka ‘password’ jest teraz zagnieżdżona i należy to teraz uwzględnić w powyższym warunku logicznym

```
          <div class="form-group" [ngClass]="{
          'has-error':(userForm.get('pswdGroup.password').touched ||
userForm.get('pswdGroup.password').dirty)
            &&!userForm.get('pswdGroup.password').valid
        }">
```

Teraz potrzebujemy zaaplikować już napisany walidator do grupy kontrolek do wypełniania haseł – jest to funkcja ‘passwordMatcher’

```
function passwordMatcher(c: AbstractControl):{[key:string]:boolean}|null{
  let passwordControl = c.get('password');
  let confirmPasswordControl = c.get('cPassword');
  if(passwordControl.pristine || confirmPasswordControl.pristine)
    return null;
  if(passwordControl.value === confirmPasswordControl.value)
    return null;
  return {'matchPasswords':true};
}
```

Aby walidator był zaaplikowany podczas inicjalizacji formularza należy go zadeklarować przy definiowaniu grupy kontrolek

```
      pswdGroup: this.formBuilder.group({
        password:  ['',[Validators.required,
Validators.pattern('^(?=.*\d)(?=.*[a-z])(?=.*[A-Z])(?!.*\s).*$')]],
        cPassword:  ['',[Validators.required]]
      },{
        validator: passwordMatcher
      })
```

Na koniec zmodyfikujmy UI tak aby nasza walidacja była reprezentowana w formularzu w postaci czerwonych ramek dla kontrolek

```
        <div formGroupName="pswdGroup"
             [ngClass]="{
               'has-error':userForm.get('pswdGroup').errors
             }"  >
```

oraz informacji pod kontrolką do ponownego wpisania hasła

```
<div class="form-group">
            <label class="col-md-3 control-label">Confirm password:</label>
            <div class="col-md-8">
              <input class="form-control" type="password"
formControlName="cPassword">
              <span
*ngIf="userForm.get('pswdGroup').errors&&userForm.get('pswdGroup').errors.matc
hPasswords">
                Passwords don't match
              </span>
            </div>
          </div>
```

Lub zapisane trochę krócej

```
<div class="form-group">
            <label class="col-md-3 control-label">Confirm password:</label>
            <div class="col-md-8">
              <input class="form-control" type="password"
formControlName="cPassword">
              <span *ngIf="userForm.get('pswdGroup').errors?.matchPasswords">
                Passwords don't match
              </span>
            </div>
          </div>
```

Notacja ze znakiem zapytania sprawdza, czy właściwość error jest nullem – jeśli jest to zwraca false a dalsza część linijku kodu nie jest już brana pod uwagę. Właściwość ‘matchPasswords’ jest kluczem który jest zwracany przez funkcję ‘passwordMatcher’ (walidator podpięty do grupy)

Kolejna rzecz która może nam się przydać to jak wydelegować informacje o błędzie do klasy komponentu, albo do jakiegoś słownika – w tym celu dodajmy słownik w klasie komponentu którego kluczem będzie nazwa błędu a wartością informacja o tym błędzie, oraz pola które będą podpięte do UI

```
export class UserFormComponent implements OnInit {

  passwordMessages={
    matchPasswords:"Passwords don't match",
    pattern: "Password is invalid"
  };

  matchPasswordsMsg;
  passwordPatternMsg;

  userForm: FormGroup;
  userExists:boolean=false;
  constructor(private formBuilder: FormBuilder) { }
```

po stronie UI

```
          <div class="form-group" [ngClass]="{
          'has-error':(userForm.get('pswdGroup.password').touched ||
userForm.get('pswdGroup.password').dirty)
            &&!userForm.get('pswdGroup.password').valid
        }">
            <label class="col-md-3 control-label">Password:</label>
            <div class="col-md-8">
              <input class="form-control" type="password"
formControlName="password">
            </div>
            <span *ngIf="userForm.get('pswdGroup.password').errors?.pattern">
                {{passwordPatternMsg}}
              </span>
          </div>
          <div class="form-group">
            <label class="col-md-3 control-label">Confirm password:</label>
            <div class="col-md-8">
              <input class="form-control" type="password"
formControlName="cPassword">
              <span *ngIf="userForm.get('pswdGroup').errors?.matchPasswords">
                {{matchPasswordsMsg}}
              </span>
            </div>
          </div>
```

W związku z tym trzeba teraz napisać logikę która będzie uzupełniać pola w klasie ‘passwordPatternMsg’ oraz ’matchPasswordsMsg’ – zmodyfikujmy metodę ngOnInit

```
  ngOnInit() {

    this.userForm = this.formBuilder.group({
      firstName: ['',[Validators.required, Validators.minLength(3)]],
      lastName:  ['',[Validators.required, Validators.maxLength(50)]],
      email:  ['',[Validators.required, Validators.email]],
      username:  ['',[Validators.required, Validators.minLength(3)]],
      pswdGroup: this.formBuilder.group({
        password:  ['',[Validators.required,
Validators.pattern('^(?=.*\d)(?=.*[a-z])(?=.*[A-Z])(?!.*\s).*$')]],
        cPassword:  ['',[Validators.required]]
      },{
        validator: passwordMatcher
      })
    });


this.userForm.get('username').valueChanges.debounceTime(1000).subscribe(value=
>{
      if(value==='admin')
        this.userExists=true;
      else this.userExists=false;
    });

    let passwordGroup = this.userForm.get("pswdGroup");
    let password = this.userForm.get("pswdGroup.password");
    password.valueChanges.debounceTime(500).subscribe(value=>{
      this.passwordPatternMsg='';
        if((password.touched||password.dirty)&&password.getError('pattern')){
          this.passwordPatternMsg = this.passwordMessages['pattern'];
        }
    });

    let confirmPassword = this.userForm.get("pswdGroup.cPassword");
    confirmPassword.valueChanges.debounceTime(500).subscribe(value=>{
      this.matchPasswordsMsg='';

if((confirmPassword.touched||confirmPassword.dirty)&&passwordGroup.getError('m
atchPasswords')){
          this.matchPasswordsMsg = this.passwordMessages['matchPasswords'];
        }
    });
  }
```

Do samodzielnego wykonania. Do formularza dodaj radio buton, w którym użytkownik, będzie mógł wybrać jedną z dwóch form weryfikacji danych: poprzez adres email, lub sms. Jeśli wybierze wysłanie tokena weryfikującego smsem to do pola z numerem telefonu ma zostać dodana reguła walidacji, że pole ma być wymagane – jeśli email ta walidacja ma zostać zdeaktywowana. Do formularza dodaj także pole do weryfikacji tokena – jeśli token jest nie prawidłowy (np. inny niż abc123) to UI powinien poinformować o tym użytkownika.