Reaktivne forme

Reaktivne forme

- ReactiveFormsModule se uvozi direktno u standalone komponentu
- Model forme se definiše u klasi komponente korišćenjem klasa FormGroup i FormControl
- Forma se povezuje sa DOM-om putem direktiva (formGroup, formControlName)
- Validacija se definiše u modelu forme (npr. Validators.required)
- Promene vrednosti i stanja forme se prate reaktivno (valueChanges, statusChanges)

Importovanje modula **ReactiveFormsModule**

```
import { Component} from '@angular/core';
import { ReactiveFormsModule, FormGroup, FormControl } from '@angular/forms';



@Component({
  selector: 'app-root',
  imports: [ReactiveFormsModule],
  templateUrl: './app.html',
  styleUrl: './app.css'
})
export class App {
  title = 'Reaktivne forme';
}
```

Klasa komponente

```
export class App {
  title = 'Reaktivne forme';

  form1 = new FormGroup({
    ime: new FormControl(''),
    prezime: new FormControl('')
  });


  onSubmit() {
    console.log(this.form1.value);
  }
}
```

Šablon glavne komponente

```
<div class="container">
  <div class="jumbotron"><h5>Reaktivne forme</h5></div>


  <div class="row">
    <div class="col-6">
      <form>
        <div class="form-group">
          <label for="ime">Ime</label>
          <input id="ime" class="form-control">
        </div>
        <div class="form-group">
          <label for="prezime">Prezime</label>
          <input id="prezime" class="form-control">
        </div>
        <button class="btn btn-primary" type="submit">Pošalji</button>
      </form>
    </div>
  </div>
</div>
```

Povezivanje šablona sa modelom

- **FormGroup** objekat se kreira u komponenti i definiše strukturu reaktivne forme
- Direktiva **formGroup** povezuje HTML form, HTML <form> element sa objektom FormGroup iz komponente
- HTML ne čuva stanje forme (vrednosti, validnost, touched/dirty)
- Vrednosti i stanja pojedinačnih polja čuvaju se u **FormControl** instancama
- HTML elementi emituju događaje (input, blur, submit), koje Angular mapira na odgovarajuće kontrole
- Direktiva **formControlName** povezuje HTML kontrolu sa konkretnim FormControl objektom iz FormGroup

Direktive formControlName i formGroup

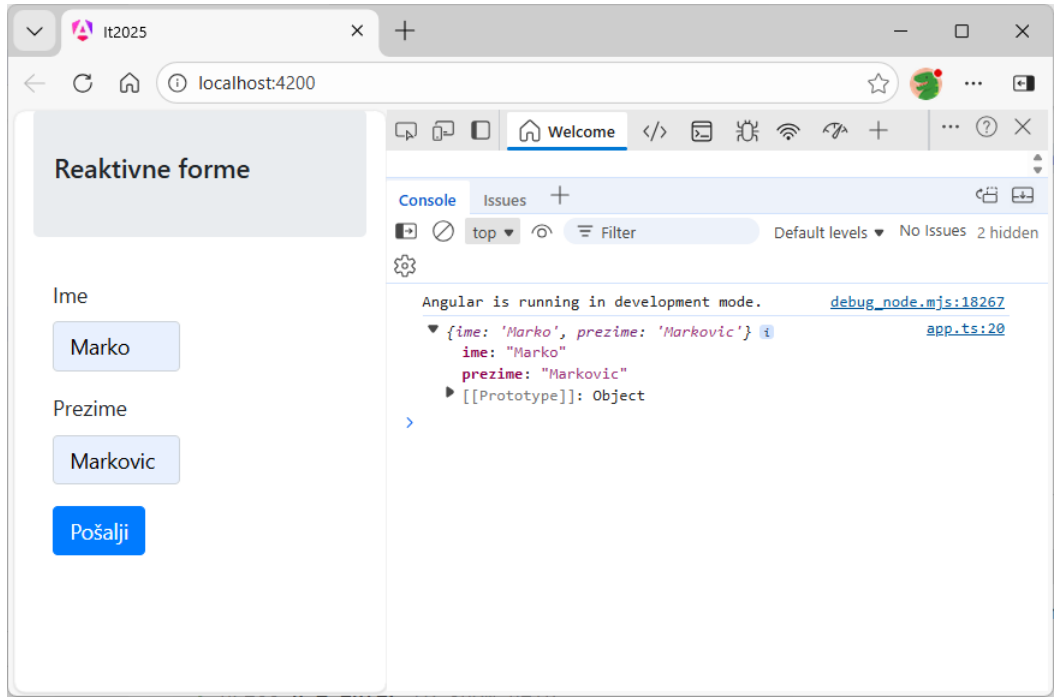
- Direktiva **formControlName** uspostavlja dvosmernu vezu između HTML kontrole i FormControl instance
- Promene u HTML inputu ažuriraju vrednost FormControl objekta
- Promene vrednosti u FormControl objektu automatski se reflektuju u HTML-u
- Direktiva **formGroup** registruje <form> kao prikaz FormGroup objekta
- Direktiva **formGroup** definiše kojem FormGroup-u pripadaju formControlName kontrole
- [**formGroup**] koristi property binding jer prima FormGroup objekat
- **formControlName** prima ključ (string) pod kojim je FormControl registrovan u FormGroup objektu

Šablon komponente

```
<div class="col-6">
  <form [formGroup]="form1" (ngSubmit)="onSubmit()">
    <div class="form-group">
      <label for="ime">Ime</label>
      <input id="ime" class="form-control" formControlName="ime">
    </div>
    <div class="form-group">
      <label for="prezime">Prezime</label>
      <input id="prezime" class="form-control" formControlName="prezime">
    </div>
    <div class="form-group">
      <button type="submit" class="btn btn-primary">Pošalji</button>
    </div>
  </form>
</div>
```

Rezultat slanja forme

- Klikom na dugme Pošalji aktivira se (ngSubmit) događaj
- Metoda onSubmit() dobija vrednosti forme iz form1.value
- Podaci se prikazuju u konzoli kao JavaScript objekat



Validacija reaktivne forme

- Validacija se definiše u modelu forme pomoću validatora
- **Validators.required** je validaciona funkcija koja proverava da li FormControl ima vrednost
- **Validators.minLength(n) / Validators.maxLength(n)** – ograničenje dužine unosa
- **Validators.pattern()** – validacija prema regularnom izrazu
- FormControl može imati više validatora, prosleđenih kao niz
- Svaki **FormControl** ima stanja: valid, invalid, touched, dirty
- FormGroup agregira validaciono stanje svih kontrola
- Forma je validna samo ako su sve kontrole validne (form1.valid)
- Validacija se izvršava automatski pri promeni vrednosti

Validacija reaktivne forme

```
import { Component } from '@angular/core';
import { FormControl, FormGroup, ReactiveFormsModule, Validators } from '@angular/forms';

@Component({
  selector: 'app-forme1-component',
  imports: [ReactiveFormsModule],
  templateUrl: './forme1-component.html',
  styleUrl: './forme1-component.css',
})
export class Forme1Component {
title = 'Reaktivne forme';

form1 = new FormGroup({
  ime: new FormControl('', Validators.required),
  prezime: new FormControl('', Validators.required)
});

onSubmit() {
  console.log(this.form1.value);
}
}
```

Prikaz validacionih grešaka – polje ime

```
<div class="form-group">
  <label for="ime">Ime</label>
  <input id="ime" class="form-control" formControlName="ime">


  @if (form1.get('ime')?.invalid && form1.get('ime')?.touched) {
    <div class="alert alert-danger">
      Unesi ime
    </div>
  }
</div>
```

Prikaz validacionih grešaka – polje prezime

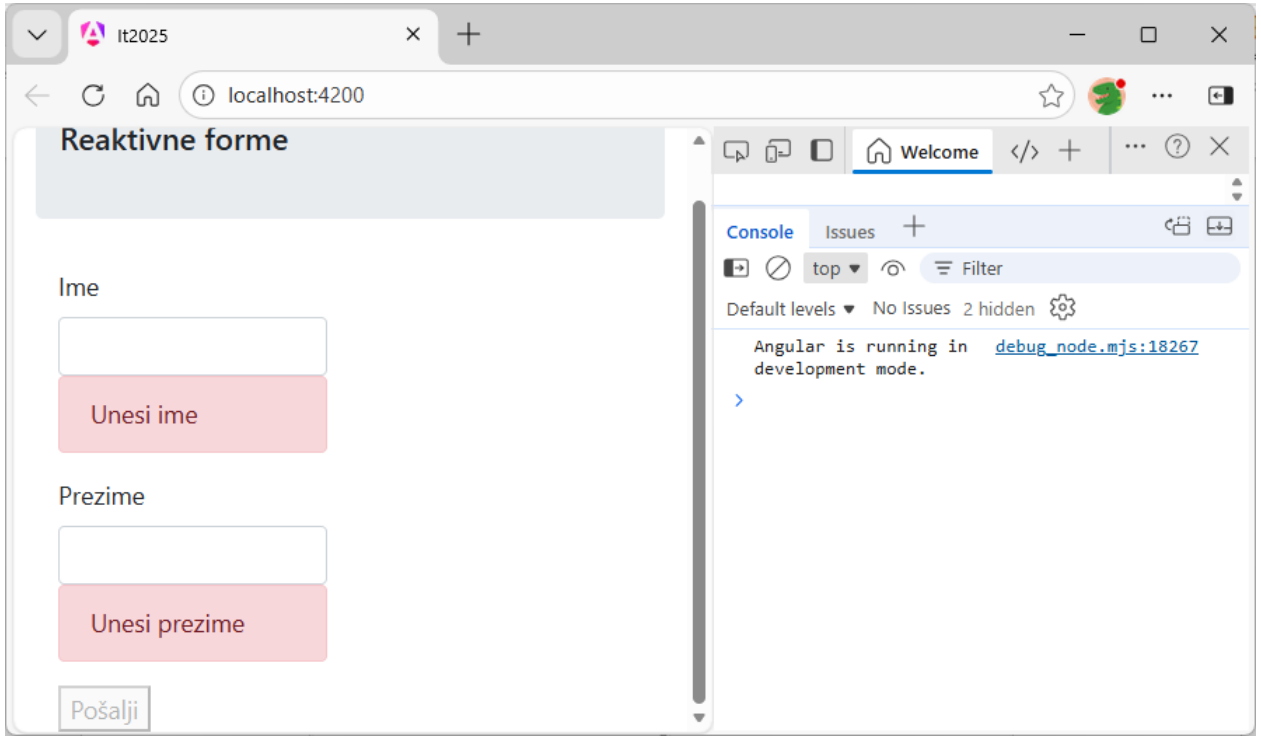
```
<div class="form-group">
  <label for="prezime">Prezime</label>
  <input id="prezime" class="form-control" formControlName="prezime">


  @if (form1.get('prezime')?.invalid && form1.get('prezime')?.touched) {
    <div class="alert alert-danger">
      Unesi prezime
    </div>
  }
</div>
```

Zabrana sabmitovanje ne validnih podataka

```
<div class="form-group">
    <button type="submit" [disabled]="form1.invalid">Pošalji</button>
</div>
```

Submit dugme je onemogućeno dok FormGroup ima nevalidne kontrole

Validacione greške



Getteri za pristup kontrolama forme

- Getter metode omogućavaju kraći i čitljiviji pristup kontrolama forme
- **this.form1.controls.ime** vraća konkretnu FormControl instancu
- Getter se koristi direktno u šablonu (ime.invalid, ime.touched)
- Time se izbegava ponavljanje izraza form1.get('ime') u HTML-u
- Getteri ne menjaju stanje forme, već samo olakšavaju čitanje

Getteri za pristup kontrolama forme

```
get ime() { return this.form1.controls.ime; }
// get ime() { return this.form1.get('ime'); }

get prezime() { return this.form1.controls.prezime; }
// get prezime() { return this.form1.get('prezime'); }
```

- controls.ime je direktan i tipizovan pristup sa autocomplete podrškom
- get('ime') koristi string ključ i nema pomoć okruženja (nema type-safety)
- get() je fleksibilniji i radi sa ugnježdenim kontrolama
- Izbor pristupa zavisi od složenosti forme

Validacija polja ime upotrebom getera

```
<div class="form-group">
    <label for="ime">Ime</label>
    <input id="ime" class="form-control" formControlName="ime">


    @if (ime.invalid && ime.touched) {
    <div class="alert alert-danger">Unesi ime</div>
    }
</div>
```

Validacija polja prezime upotrebom getera

```
<div class="form-group">
  <label for="prezime">Prezime</label>
  <input id="prezime" class="form-control" formControlName="prezime">


  @if (prezime.invalid && prezime.touched) {
    <div class="alert alert-danger">
      Unesi prezime
    </div>
  }
</div>
```

Pristup validacionim greškama (errors)

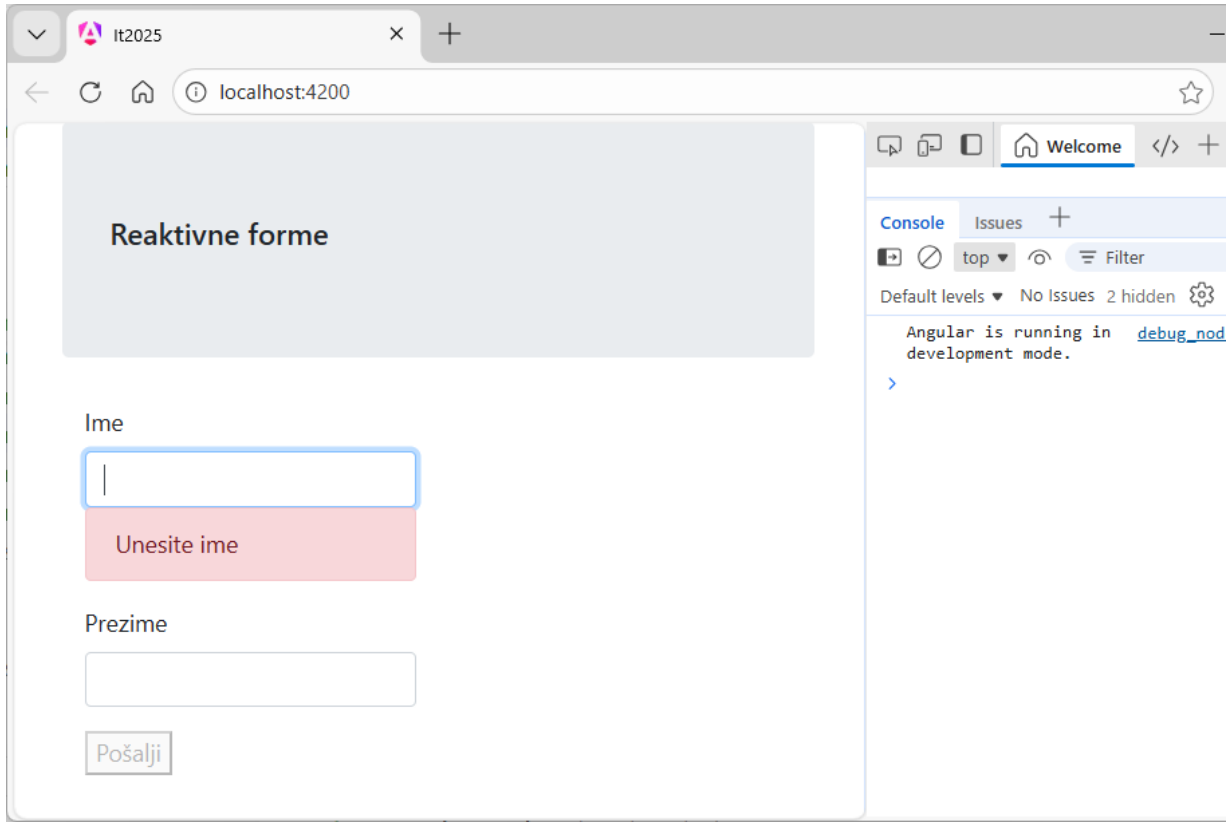
- **errors** je objekat koji sadrži validacione greške za FormControl
- Svaki ključ u errors objektu odgovara **nazivu validatora**
- **ime.errors?.['required']** proverava da li je aktivna greška validatora required
- Operator ?. sprečava grešku ako errors trenutno ne postoji
- Alternativno, može se koristiti metoda **hasError**('required')
- hasError() vraća true ako je određena validaciona greška aktivna
- Ovo je ispravan i preporučen način provere pojedinačnih validacionih grešaka

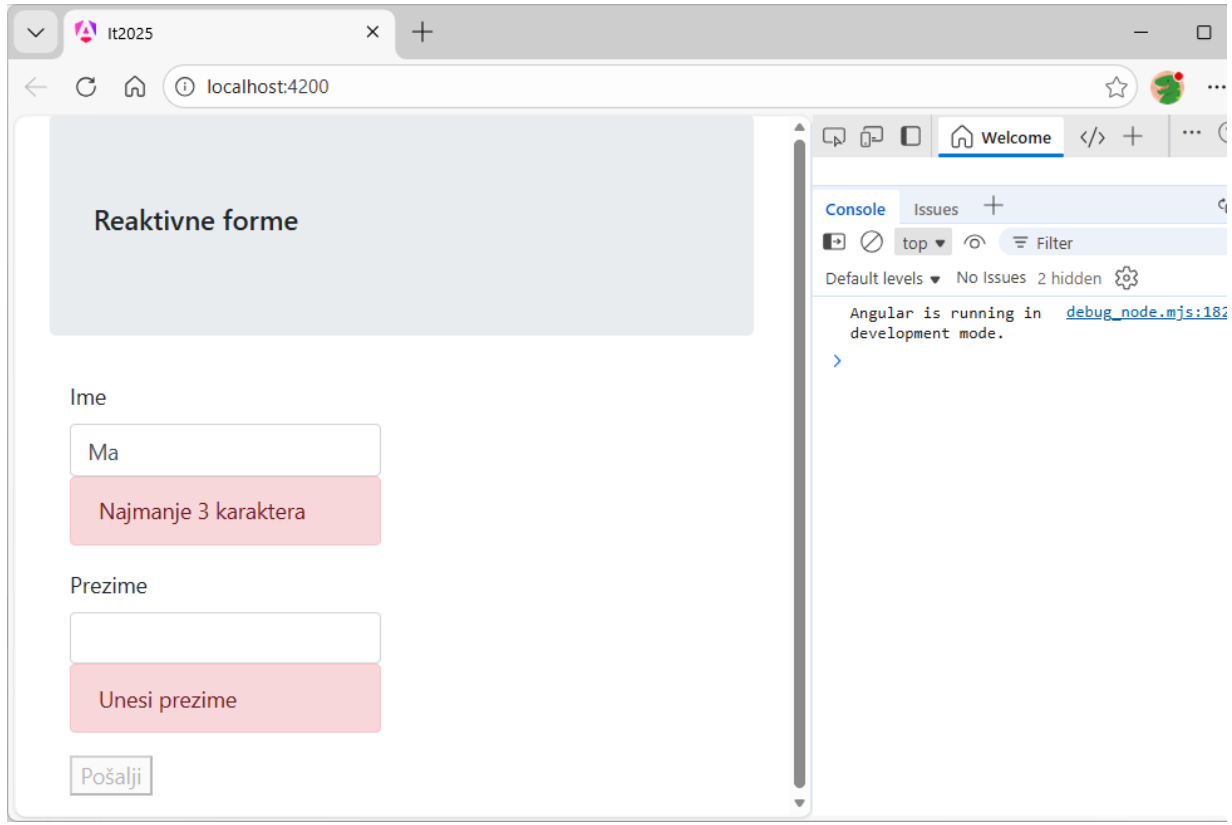
Definisanje više validatora za polje ime

```
form1 = new FormGroup({
  ime: new FormControl('', [Validators.required, Validators.minLength(3)]),
  prezime: new FormControl('', Validators.required)
});
```

```
@if (ime.invalid && ime.dirty) {
<div class="alert alert-danger">
    @if (ime.errors?.['required']) {
    <div>Unesite ime</div>
    }
    @if (ime.errors?.['minlength']) {
    <div>Najmanje 3 karaktera</div>
    }
</div>
```

Prikaz validacionih grešaka





FormBuilder

- FormBuilder je pomoćni API za konstrukciju FormGroup/FormControl instanci, umesto direktnog new
- Metoda **group**() kreira instancu FormGroup
- Metoda **control**() kreira instancu FormControl
- FormBuilder se ubrizgava putem dependency injection-a u komponentu
- Najčešće se koristi umesto ručnog pozivanja new FormGroup() i new FormControl()

```
import { FormGroup, FormBuilder, Validators } from '@angular/forms';
```

FormBuilder

```
export class Forme3Component {
title = 'FormBuilder';
  form1: FormGroup;


  constructor(private fb: FormBuilder) {
    this.form1 = this.fb.group({
      ime: ['', [Validators.required, Validators.maxLength(10)]],
      prezime: ['', Validators.required]
    });
  }


  get ime() { return this.form1.controls['ime']; }
  get prezime() { return this.form1.controls['prezime']; }

  onSubmit() {
    console.log(this.form1.value);
  }
}
```

Šablon komponente

```
<div class="form-group">
  <label for="ime">Ime</label>
  <input id="ime" class="form-control" formControlName="ime">


  @if (ime.invalid && ime.touched) {
    <div class="alert alert-danger">
      @if (ime.hasError('required')) { <div>Unesite ime</div> }
      @if (ime.hasError('maxlength')) { <div>Najviše 10 karaktera</div> }
    </div>
  }
</div>
...
<div class="form-group">
<button type="submit" class="btn btn-primary"
[disabled]="form1.invalid">Pošalji</button>
</div>
```

id atribut nije obavezan za reaktivne forme, ali se koristi zbog povezivanja sa <label> i pristupačnosti.

Reaktivna forma primer – model klasa

```
export class Student {
  constructor(
    public ime: string,
    public prezime: string,
    public pol: string,
    public smer: string
  ) {}
}
```

Komponenta StudentComponent

```
export class StudentComponent {

  title = 'Reaktivne forme vežba';

  smerovi = ['Informatika', 'Marketing', 'Menadžment'];

  student: Student = new Student('', '', 'muski', 'Informatika');


  studentForm = new FormGroup({
    ime: new FormControl('', Validators.required),
    prezime: new FormControl('', Validators.required),
    pol: new FormControl('muski', Validators.required),
    smer: new FormControl('Informatika', Validators.required)
  });
...
}
```

Geteri za pristup kontrolama

```
get ime() { return this.studentForm.controls.ime; }
  get prezime() { return this.studentForm.controls.prezime; }
  get pol() { return this.studentForm.controls.pol; }
  get smer() { return this.studentForm.controls.smer; }
```

Metoda onSubmit()

```
onSubmit() {
  if (this.studentForm.invalid) return;

  const student = new Student(
    this.ime.value!,
    this.prezime.value!,
    this.pol.value!,
    this.smer.value!
  );


  console.log(student);
}
```

- ! je TypeScript non-null assertion operator
- Njime se govori kompajleru: ova vrednost ovde nije null.

Metode resetuj() i setDefault()

```
resetuj() {
  this.student = new Student('', '', 'muski',
'Informatika');
  this.studentForm.reset(this.student);
}
// setDefault {
//   this.studentForm.patchValue({
//     ime: 'Marko',
//     prezime: 'Markovic',
//     pol: 'muski',
//     smer: 'Informatika'
//   });
// }
setDefault() {
  this.studentForm.setValue({
    ime: 'Marko',
    prezime: 'Markovic',
    pol: 'muski',
    smer: 'Informatika'
  });
}
```

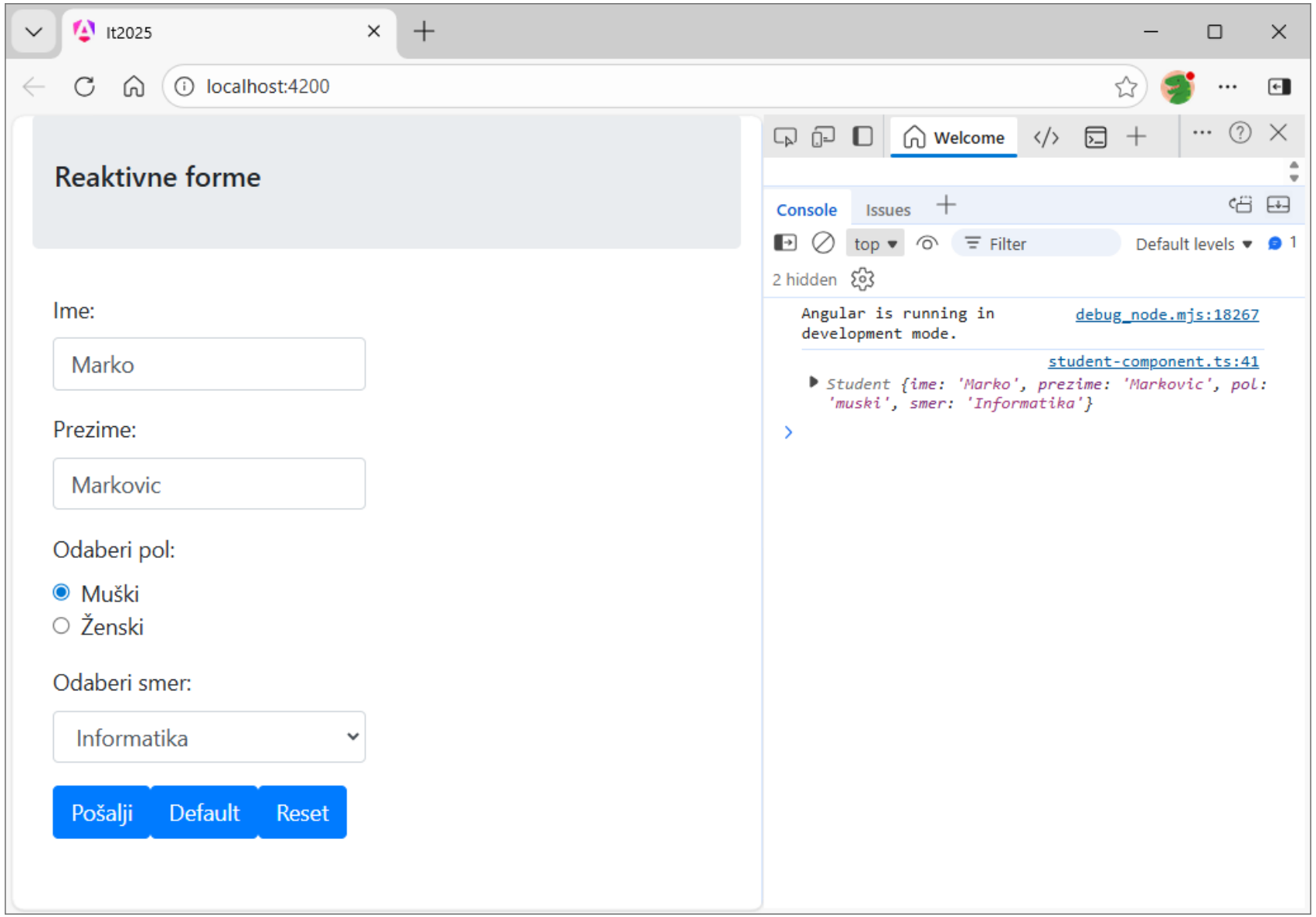
- **reset**() – vraća formu u početno stanje (vrednosti + statusi)
- **patchValue**() – dozvoljava parcijalno popunjavanje
- **setValue**() – zahteva vrednosti za sva polja forme

Šablon komponente

```
<form [formGroup]="studentForm" (ngSubmit)="onSubmit()">


  <div class="form-group">
    <label for="ime">Ime:</label>
    <input id="ime" type="text" class="form-control" formControlName="ime">
    @if (ime.invalid && ime.touched) { <div>Unesite ime.</div> }
  </div>


  <div class="form-group">
    <label for="prezime">Prezime:</label>
    <input id="prezime" type="text" class="form-control" formControlName="prezime">
    @if (prezime.invalid && prezime.touched) { <div>Unesite prezime.</div> }
  </div>

  ....
</form>
```

```
<div class="form-group">
  <label>Odaberi pol:</label>
  <div class="form-check">
    <input type="radio" id="radio1" value="muski" class="form-check-input" formControlName="pol">
    <label for="radio1" class="form-check-label">Muški</label>
  </div>
  <div class="form-check">
    <input type="radio" id="radio2" value="zenski" class="form-check-input" formControlName="pol">
    <label for="radio2" class="form-check-label">Ženski</label>
  </div>
</div>


<div class="form-group">
  <label for="smer">Odaberi smer:</label>
  <select id="smer" class="form-control" formControlName="smer">
    @for (smer of smerovi; track smer) { <option [value]="smer">{{ smer }}</option> }
  </select>
</div>
```

```
<div class="form-group">
  <button class="btn btn-primary" [disabled]="studentForm.invalid">Pošalji</button>
  <button type="button" class="btn btn-primary" (click)="setDefault()">Default</button>
  <button type="button" class="btn btn-primary" (click)="resetuj()">Reset</button>
</div>
```


It2025
localhost:4200
Reaktivne forme
Ime:
Marko
Prezime:
Markovic
Odaberi pol:
Muški
Ženski
Odaberi smer:
Informatika
Pošalji
Default
Reset
Welcome
Console
Issues
top
Filter
Default levels
2 hidden
Angular is running in development mode.
debug_node.mjs:18267
student-component.ts:41
Student {ime: 'Marko', prezime: 'Markovic', pol: 'muski', smer: 'Informatika'}

Pitanje 1

Povezivanje postojeće instance klase FormGroup sa HTML elementom <form> u reaktivnim formama vrši se korišćenjem direktive:

a. formGroup
b. formControlName
c. bindForm

Odgovor: a

Pitanje 2

Sinhronizacija instance klase FormControl sa DOM elementom u reaktivnim formama vrši se posredstvom direktive:

a. control
b. controlBind
c. formControlName

Odgovor: c

Pitanje 3

Kod reaktivnih formi :

a. Instance klase **FormGroup** i **FormControl** se kreiraju na osnovu šablona komponente u kome se nalazi forma
b. Kreira se model forme u kome se instanciraju klase **FormGroup** i **FormControl** pa se vrši sinhronizacija tog modela sa formom
c. Forma se kreira instanciranjem klase **ReactiveForm**

Odgovor: b

Pitanje 4

Kod reaktivnih formi Angular servis koji se koristi za kreiranje instanci FormGroup i FormControl naziva se:

a. FormGenerator
b. CreateForm
c. FormBuilder

Odgovor: c

Pitanje 5

U reaktivnim formama gde se čuvaju vrednosti polja i validaciona stanja forme?

a. U HTML DOM elementima
b. U instancama FormGroup i FormControl
c. U šablonu komponente

Odgovor: b

Pitanje 6

U reaktivnim formama, šta predstavlja svojstvo errors objekta FormControl?

a. Objekat koji sadrži aktivne validacione greške za kontrolu
b. Niz poruka o greškama koje Angular automatski prikazuje
c. Boolean vrednost koja označava da li je kontrola validna

Odgovor: a

Pitanje 7

Koja od sledećih opcija predstavlja ispravnu sintaksu za povezivanje FormGroup instance sa HTML <form> elementom u reaktivnim formama?

a. <form (formGroup)="form1">
b. <form formGroup="form1">
c. <form [formGroup]="form1">

Odgovor: c