Forms in Angular

523419

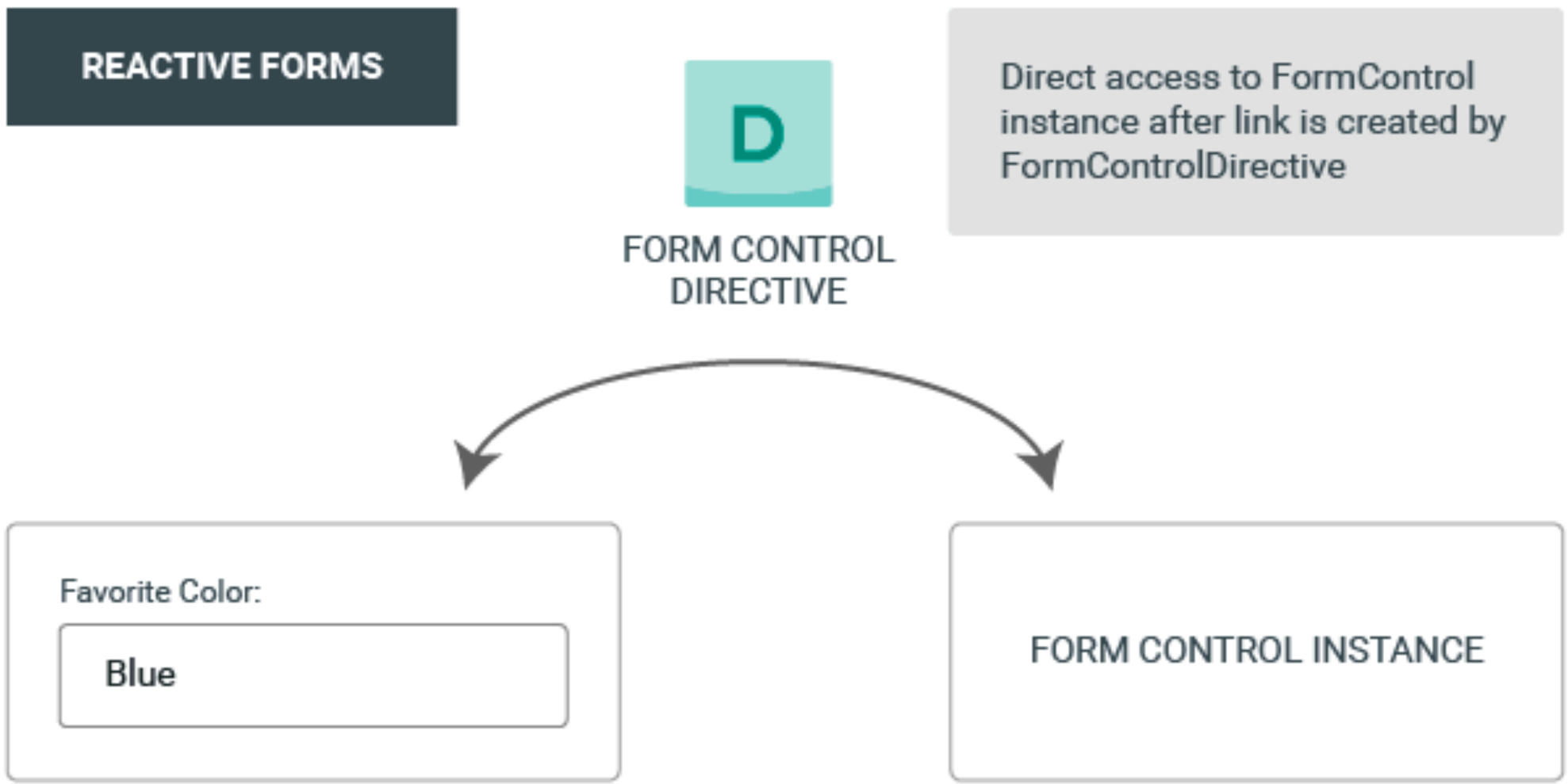
(Advanced Web Application Development)

Dr. Nuntawut Kaoungku
Assistant Professor of Computer Engineering

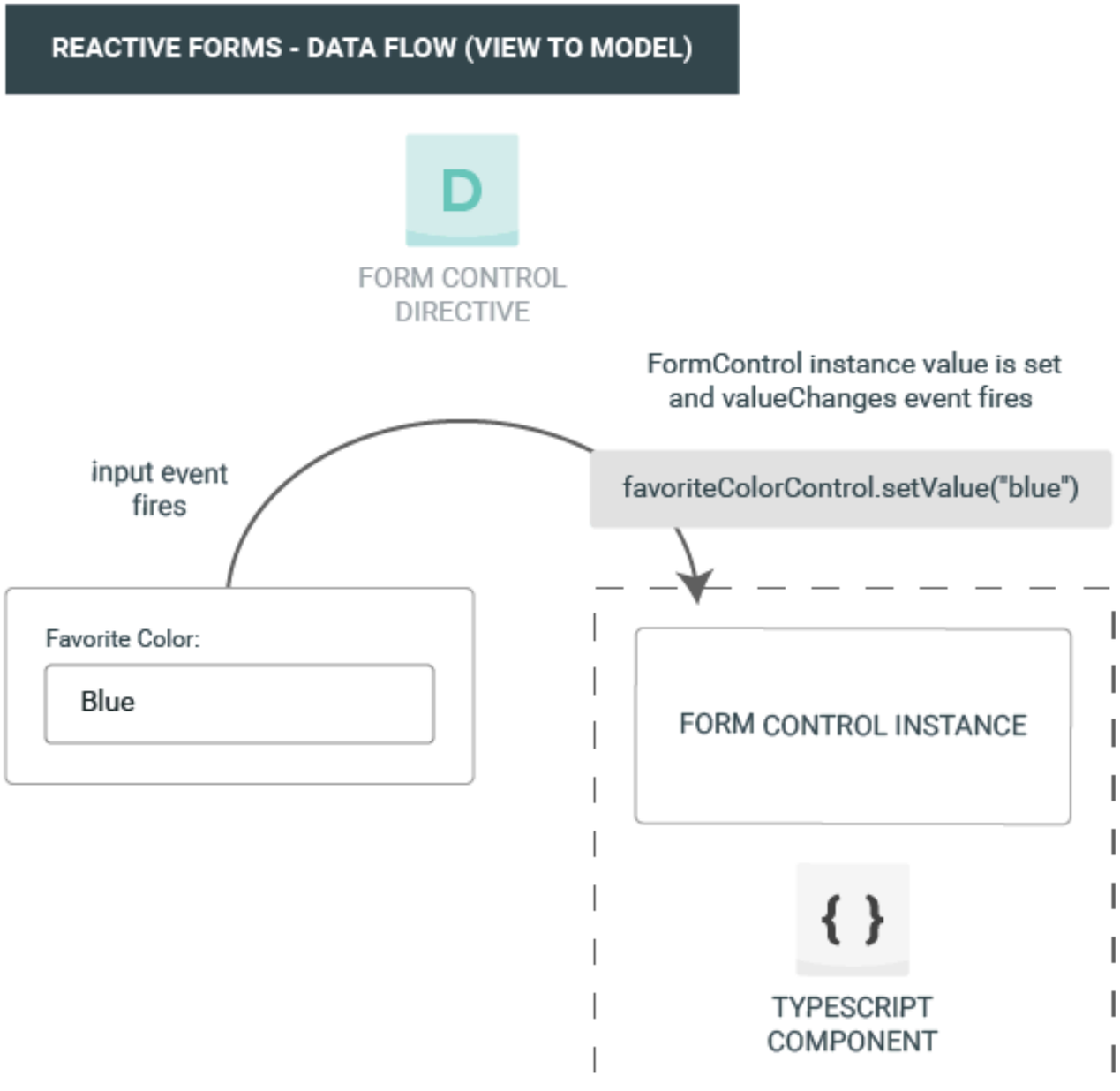
Introduction Forms in Angular

- Reactive forms are more robust: they're more scalable, reusable, and testable.
    - If forms are a key part of your application, or you're already using reactive patterns for building your application, use reactive forms.
- Template-driven forms are useful for adding a simple form to an app, such as an email list signup form.
    - They're easy to add to an app, but they don't scale as well as reactive forms. If you have very basic form requirements and logic that can be managed solely in the template, use template-driven forms.

Setup in reactive forms



Reactive Forms - Data Flow (View to Model)



Reactive Forms - Data Flow (View to Model)

- The steps below outline the data flow from view to model.

    1. The user types a value into the input element, in this case the favorite color Blue.

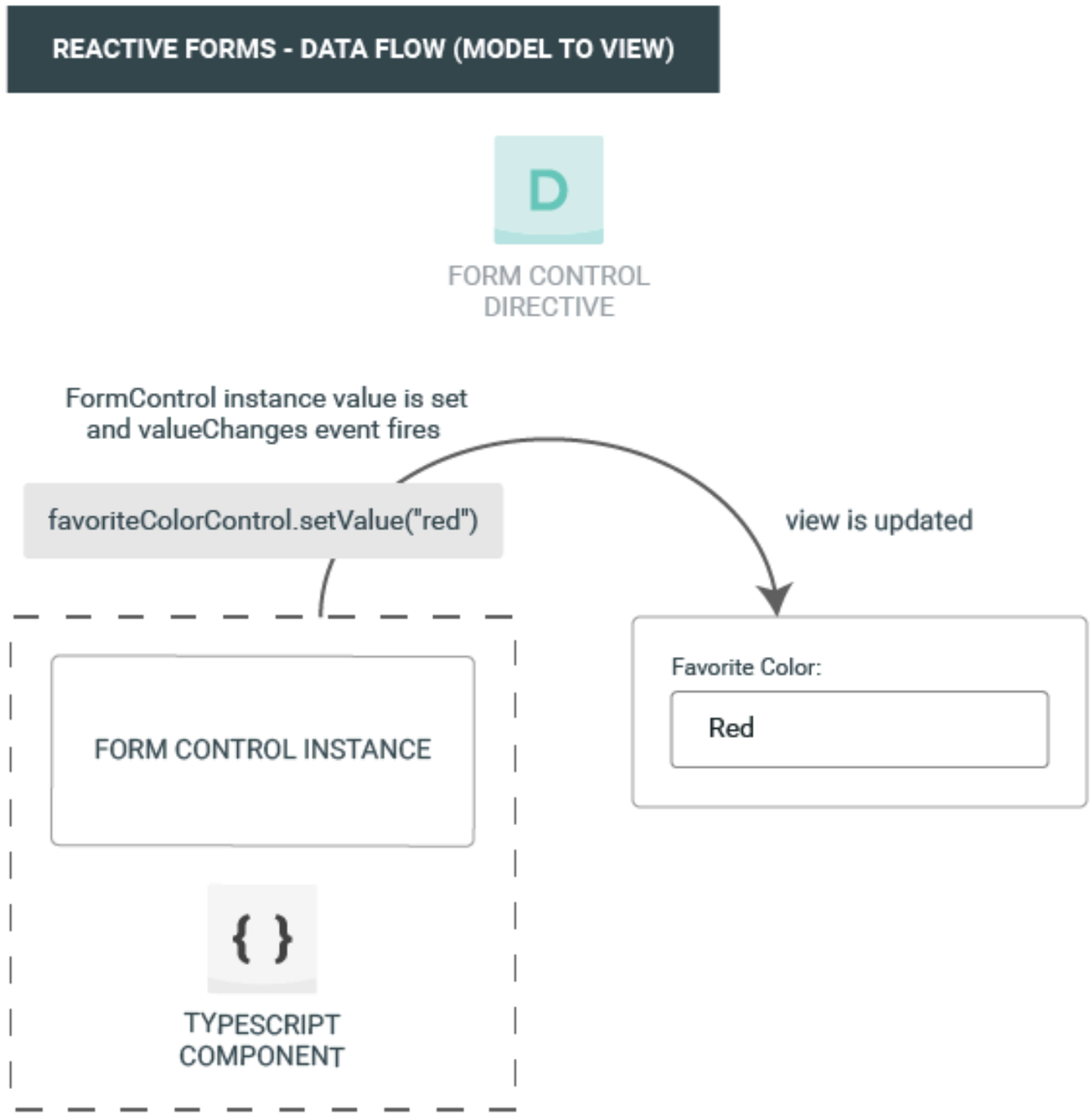
    2. The form input element emits an "input" event with the latest value.

    3. The control value accessor listening for events on the form input element immediately relays the new value to the FormControl instance.

    4. The FormControl instance emits the new value through the valueChanges observable.

    5. Any subscribers to the valueChanges observable receive the new value.

Reactive Forms - Data Flow (Model to View)



Reactive Forms - Data Flow (Model to View)

- The steps below outline the data flow from model to view.

    1. The user calls the favoriteColorControl.setValue() method, which updates the FormControl value.

    2. The FormControl instance emits the new value through the valueChanges observable.

    3. Any subscribers to the valueChanges observable receive the new value.

    4. The control value accessor on the form input element updates the element with the new value.

Reactive Form Class

Class	Description
AbstractControl	The abstract base class for the concrete form control classes FormControl, FormGroup, and FormArray. It provides their common behaviors and properties.
FormControl	Manages the value and validity status of an individual form control. It corresponds to an HTML form control such as <input> or <select>.
FormGroup	Manages the value and validity state of a group of AbstractControl instances. The group's properties include its child controls. The top-level form in your component is FormGroup.
FormArray	Manages the value and validity state of a numerically indexed array of AbstractControl instances.
FormBuilder	An injectable service that provides factory methods for creating control instances.

Registering the reactive forms module

app.module.ts

src > app > app.module.ts > ...

```ts
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { ReactiveFormsModule } from '@angular/forms';

import { AppComponent } from './app.component';
import { SignupComponent } from './components/signup/signup.component';

@NgModule({
  declarations: [
    AppComponent,
    SignupComponent
  ],
  imports: [
    BrowserModule,
    ReactiveFormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }

```

Importing a new form control

TS signup.component.ts ●

src > app > components > signup > TS signup.component.ts > ...

```ts
import { Component, OnInit } from '@angular/core';
import { FormControl } from '@angular/forms';

@Component({
  selector: 'app-signup',
  templateUrl: './signup.component.html',
  styleUrls: ['./signup.component.css']
})
export class SignupComponent implements OnInit {

  name = new FormControl();

  constructor() { }

  ngOnInit(): void {

  }

}
```

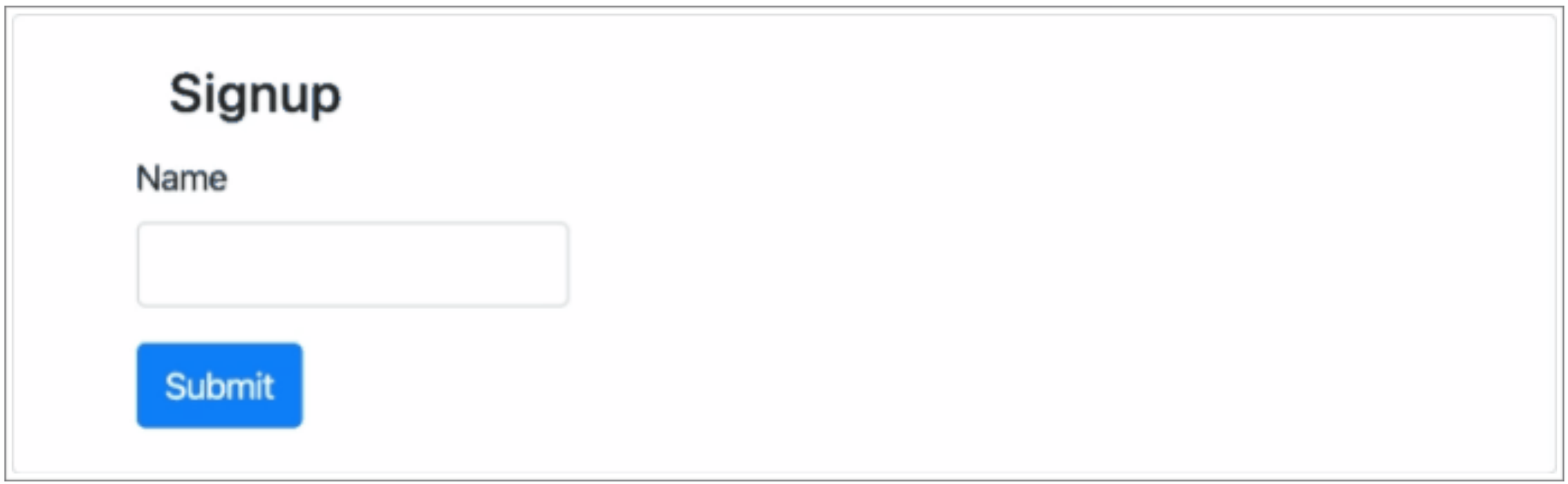
Registering the control in the template

signup.component.html

src > app > components > signup > signup.component.html > div.container

```
1   <div class="container" style="margin-top:60px">
2     <div class="card">
3       <div class="card-body" style="margin-left:50px">
4         <h4 class="card-title">Signup</h4>
5         <div class="row">
6           <form>
7             <div class="form-group">
8               <label for="">Name</label>
9               <input type="text" class="form-control" [formControl]="name">
10              <small id="helpId" class="form-text text-muted">{{name.value}}</small>
11            </div>
12            <button type="button" class="btn btn-primary">Submit</button>
13          </form>
14        </div>
15      </div>
16    </div>
17
```

Displaying the component



Replacing a form control value

signup.component.ts

```
onUpdate(){
  this.name.setValue("Nuntawut");
}
```

signup.component.html

```
<form>
  <div class="form-group">
    <label for="">Name</label>
    <input type="text" class="form-control" [formControl]="name">
    <small id="helpId" class="form-text text-muted">{{name.value}}</small>
  </div>
  <button type="button" (click)="onUpdate()" class="btn btn-primary">Submit</button>
</form>
```

Grouping form controls

TS signup.component.ts

src > app > components > signup > TS signup.component.ts > ...

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

@Component({
  selector: 'app-signup',
  templateUrl: './signup.component.html',
  styleUrls: ['./signup.component.css']
})
export class SignupComponent implements OnInit {

  profileForm = new FormGroup({
    firstName: new FormControl(''),
    lastName: new FormControl(''),
  });

  constructor() { }

  ngOnInit(): void {
  }
}
```

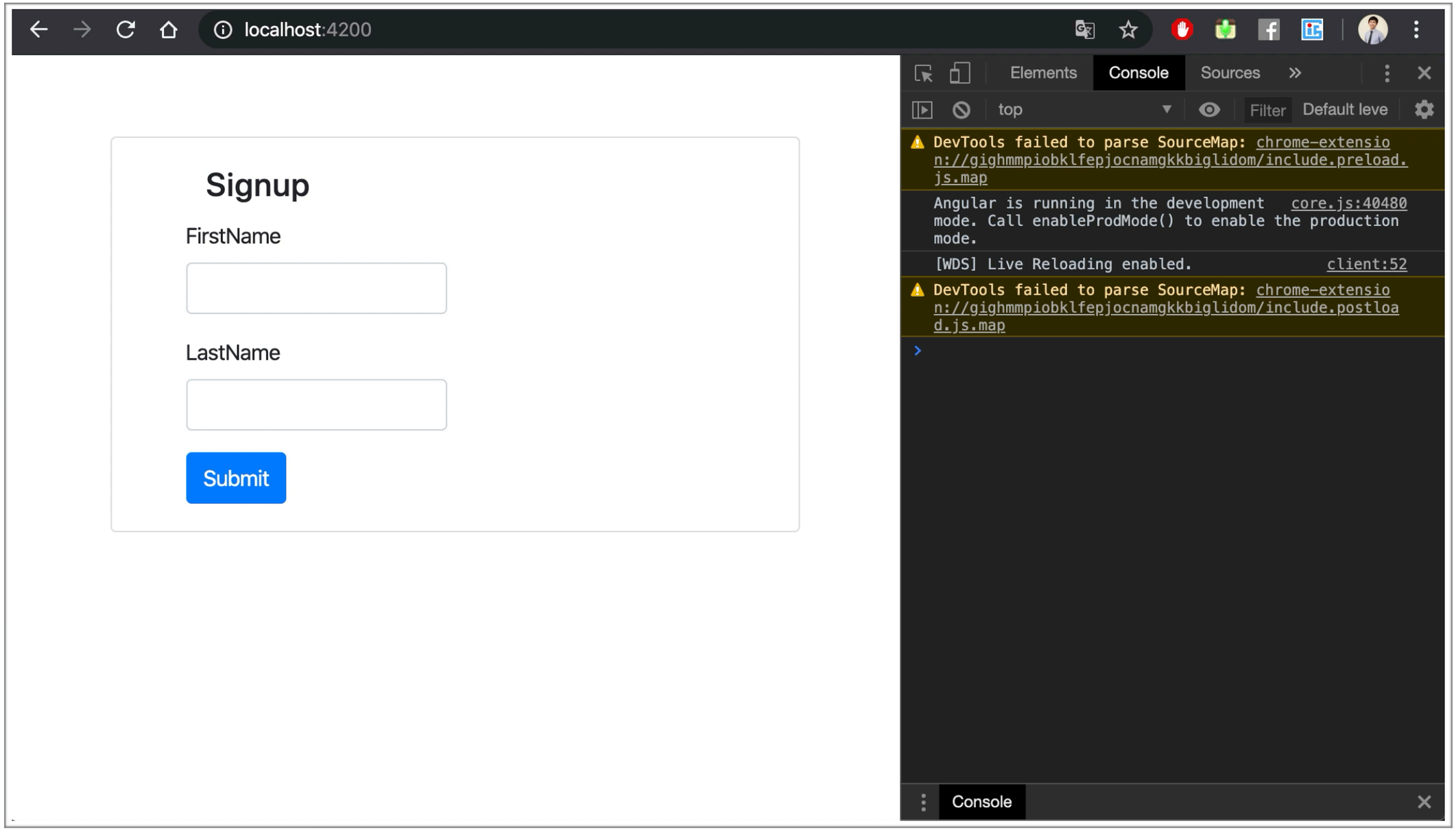
Associating the FormGroup model and view

signup.component.html

src > app > components > signup > signup.component.html > div.container

```
1    <div class="container" style="margin-top:60px">
2      <div class="card">
3        <div class="card-body" style="margin-left:50px">
4          <h4 class="card-title">Signup</h4>
5          <div class="row">
6            <form [formGroup]="profileForm" (ngSubmit)="onSubmit()">
7              <div class="form-group">
8                <label>FirstName</label>
9                <input type="text" class="form-control" formControlName='firstName'>
10             </div>
11             <div class="form-group">
12               <label>LastName</label>
13               <input type="text" class="form-control" formControlName='lastName'>
14             </div>
15             <button type="submit" class="btn btn-primary">Submit</button>
16           </form>
17       </div>
18     </div>
19   </div>
20
```

Displaying the component



Creating nested form groups

TS signup.component.ts ×

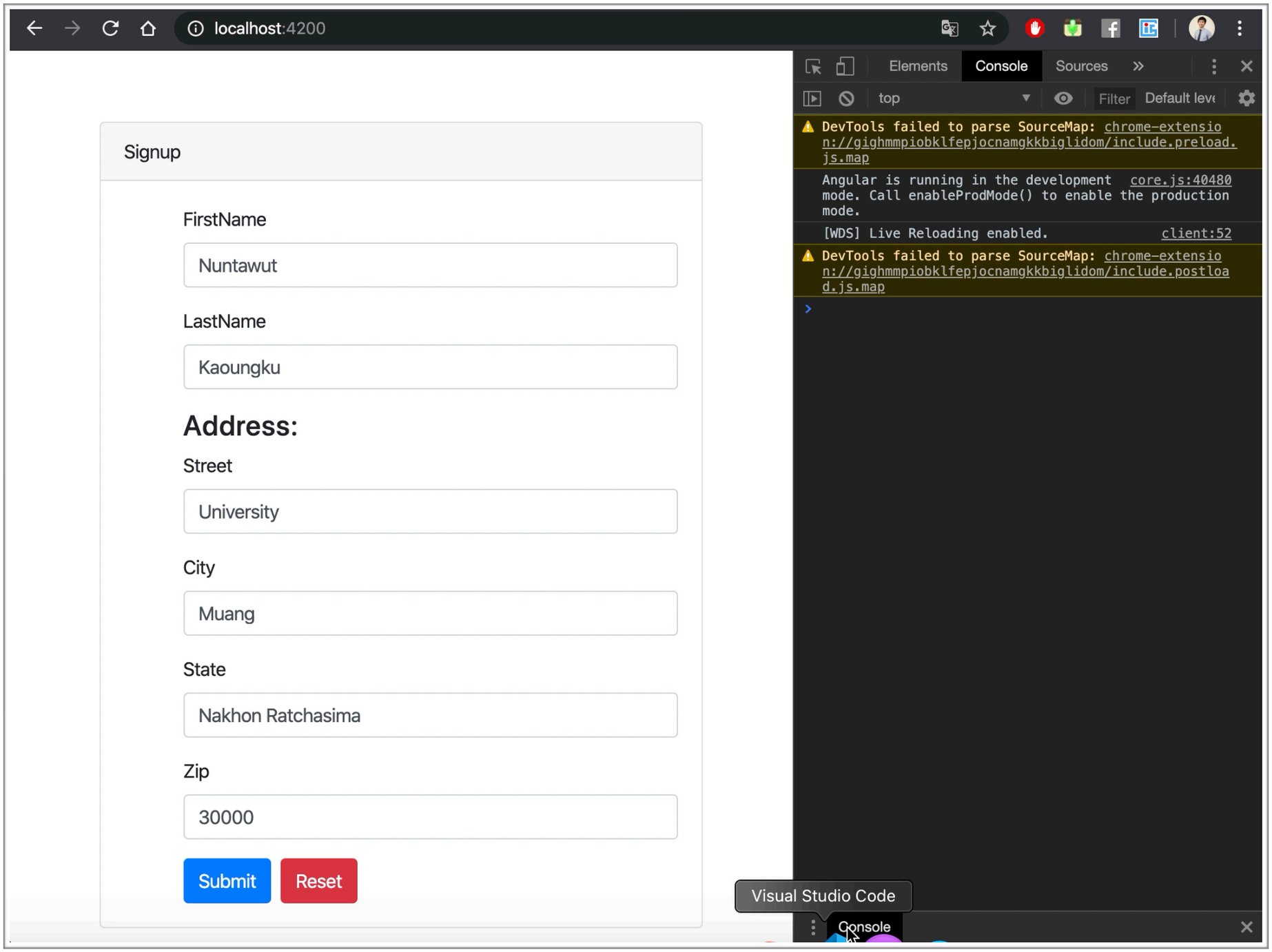
src > app > components > signup > TS signup.component.ts > ...

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

@Component({
  selector: 'app-signup',
  templateUrl: './signup.component.html',
  styleUrls: ['./signup.component.css']
})
export class SignupComponent implements OnInit {

  profileForm = new FormGroup({
    firstName: new FormControl(''),
    lastName: new FormControl(''),
    address: new FormGroup({
      street: new FormControl(''),
      city: new FormControl(''),
      state: new FormControl(''),
      zip: new FormControl('')
    })
  });

```

Grouping the nested form in the template

```
<h4>Address:</h4>
<div formGroupName="address">
  <div class="form-group">
    <label for="">Street</label>
    <input type="text" class="form-control" formControlName='street'>
  </div>
  <div class="form-group">
    <label for="">City</label>
    <input type="text" class="form-control" formControlName='city'>
  </div>
  <div class="form-group">
    <label for="">State</label>
    <input type="text" class="form-control" formControlName='state'>
  </div>
  <div class="form-group">
    <label for="">Zip</label>
    <input type="text" class="form-control" formControlName='zip'>
  </div>
</div>
```

Displaying the component



Generating form controls with FormBuilder

TS signup.component.ts ●

src > app > components > signup > TS signup.component.ts > ...

```
1   import { Component, OnInit } from '@angular/core';
2   import { FormControl, FormGroup, FormBuilder } from '@angular/forms';
3
4   @Component({
5     selector: 'app-signup',
6     templateUrl: './signup.component.html',
7     styleUrls: ['./signup.component.css']
8   })
9   export class SignupComponent implements OnInit {
10
11
12    constructor(private fb: FormBuilder) { }
13
14    ngOnInit(): void {
15
16    }
17
18  }
19
```

Generating form controls

Compare using the form builder to creating the instances manually.

```
profileForm = new FormGroup({
  firstName: new FormControl(''),
  lastName: new FormControl(''),
  address: new FormGroup({
    street: new FormControl(''),
    city: new FormControl(''),
    state: new FormControl(''),
    zip: new FormControl('')
  })
});
```

Instances

```
profileForm = this.fb.group({
  firstName: [''],
  lastName: [''],
  address: this.fb.group({
    street: [''],
    city: [''],
    state: [''],
    zip: ['']
  }),
});
```

Form Builder

Dynamic controls using form arrays

TS signup.component.ts ●

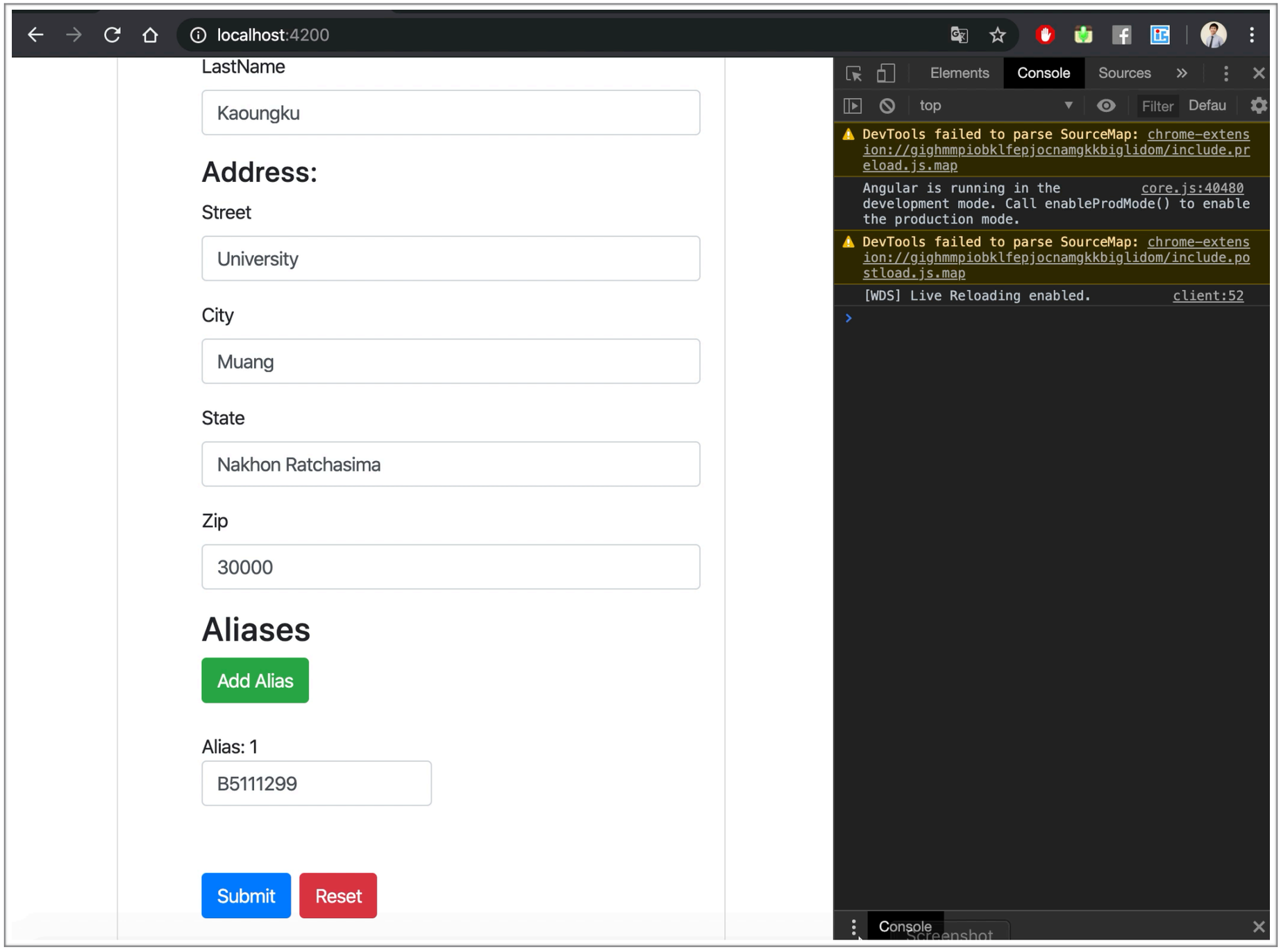
src > app > components > signup > TS signup.component.ts > SignupComponent > profileForm

```
9   export class SignupComponent implements OnInit {
10
11    profileForm = new FormGroup({
12      firstName: new FormControl(''),
13      lastName: new FormControl(''),
14      address: new FormGroup({
15        street: new FormControl(''),
16        city: new FormControl(''),
17        state: new FormControl(''),
18        zip: new FormControl('')
19      }),
20      aliases: this.fb.array([
21        this.fb.control('')
22      ])
23    });
24
25    get aliases() {
26      return this.profileForm.get('aliases') as FormArray;
27    }
28
29    constructor(private fb: FormBuilder) { }
30
31    ngOnInit(): void {
32    }
33
34    addAlias() {
35      this.aliases.push(this.fb.control(''));
36    }
```

Displaying the form array in the template

```
<div formArrayName="aliases">
  <h3>Aliases</h3>

  <button (click)="addAlias()" class="btn btn-success">Add Alias</button>

  <div *ngFor="let alias of aliases.controls; let i=index">
  <!-- The repeated alias template -->
    <label>
      Alias: {{i+1}}
      <input type="text" class="form-control" [formControlName]="i">
    </label>
  </div>
</div>
```

Displaying the component



Reactive form validation

- There are two types of validator functions: sync validators and async validators.

    - Sync validators: functions that take a control instance and immediately return either a set of validation errors or null. You can pass these in as the second argument when you instantiate a FormControl.

    - Async validators: functions that take a control instance and return a Promise or Observable that later emits a set of validation errors or null. You can pass these in as the third argument when you instantiate a FormControl.

Built-in validators

```
class Validators {
  static min(min: number): ValidatorFn
  static max(max: number): ValidatorFn
  static required(control: AbstractControl): ValidationErrors | null
  static requiredTrue(control: AbstractControl): ValidationErrors | null
  static email(control: AbstractControl): ValidationErrors | null
  static minLength(minLength: number): ValidatorFn
  static maxLength(maxLength: number): ValidatorFn
  static pattern(pattern: string | RegExp): ValidatorFn
  static nullValidator(control: AbstractControl): ValidationErrors | null
  static compose(validators: ValidatorFn[]): ValidatorFn | null
  static composeAsync(validators: AsyncValidatorFn[]): AsyncValidatorFn | null
}
```

Adding to reactive forms

TS signup.component.ts

src > app > components > signup > TS signup.component.ts > SignupComponent > aliases

```
1   import { Component, OnInit } from '@angular/core';
2   import { FormControl, FormGroup, FormBuilder, FormArray, Validators } from '@angular/forms';
3
4   @Component({
5     selector: 'app-signup',
6     templateUrl: './signup.component.html',
7     styleUrls: ['./signup.component.css']
8   })
9   export class SignupComponent implements OnInit {
10
11    profileForm = new FormGroup({
12      firstName: new FormControl('',[Validators.required,Validators.minLength(4)]),
13      lastName: new FormControl(''),
14      address: new FormGroup({
15        street: new FormControl(''),
16        city: new FormControl(''),
17        state: new FormControl(''),
18        zip: new FormControl('')
19      }),
20      aliases: this.fb.array([
21        this.fb.control('')
22      ])
23    });
24
25    get firstName() { return this.profileForm.get('firstName'); }
26
```

Control status CSS classes

- Like in AngularJS, Angular automatically mirrors many control properties onto the form control element as CSS classes.
- You can use these classes to style form control elements according to the state of the form.
- The following classes are currently supported:
  - .ng-valid: The field content is valid
  - .ng-invalid: The field content is not valid
  - .ng-pending: The field has been pending
  - .ng-pristine: The field has not been modified yet
  - .ng-dirty: The field has been modified
  - .ng-untouched: The field has not been touched yet
  - .ng-touched: The field has been touched

Displaying the form validators in the template

```
<div class="form-group">
  <label>FirstName</label>
  <input type="text" class="form-control" formControlName='firstName'>

  <div *ngIf="firstName.invalid && (firstName.dirty || firstName.touched)"
      class="alert alert-danger">

      <div *ngIf="firstName.errors.required">
        Name is required.
      </div>

      <div *ngIf="firstName.errors.minlength">
        Name must be at least 4 characters long.
      </div>
  </div>
</div>
```

Displaying the component