PoiemaWeb

13.19 Angular Forms - Basics

HTML 표준 폼과 Angular 폼

#1. 폼이란?

폼(Form)은 애플리케이션에서 사용자의 데이터를 입력받는 인터페이스를 의미한다. 다시 말해 사용자는 폼을 통해 애플리케이션에 데이터를 제공할 수 있으며 이 데이터는 일반적으로 유효성 검증 절차를 통과한 후 서버로 전송된다. 하지만 사용자가 애플리케이션이 기대하는 유효한 형식의 데이터를 입력하리라는 낙관은 금물이다. 예를 들어 이메일을 입력받는 폼이 있다고 할 때, 애플리케이션이 기대하는 데이터는 이메일 형식에 부합하는 데이터이지만 반드시 사용자가 그 형식에 맞는 데이터를 입력할 지는 알 수 없다.

따라서 사용자가 입력한 폼 데이터는 애플리케이션이 기대하는 데이터 형식에 부합하는지 그렇지 아니한지 체크가 필요한데 이를 유효성 검증(Form data validation)이라 한다. 유효성 검증을 통해 부적절한

데이터가 데이터 처리 로직으로 넘어가는 것을 방지하여야 하고 사용자에게는 애플리케이션이 기대하는 데이터를 형식에 맞게 입력하도록 효과적인 방식으로 안내하여야 한다.

HTML 표준 폼으로도 어느 정도의 유효성 검증이 가능하고 사용자가 입력한 폼 데이터를 서버로 전송할 수 있지만, 여러모로 단점과 한계가 있기 때문에 애플리케이션 개발에 적용하기 어렵다. Angular는 HTML 표준 폼의 단점과 한계를 보완한 템플릿 기반 폼과 리액티브(모델 기반) 폼을 제공한다.

2. HTML 표준 폼

HTML이 제공하는 표준 폼을 사용하여도 어느 정도의 유효성 검증이 가능하며 폼 데이터를 서버로 전송할 수 있다. 단, HTML 표준 폼만으로는 세밀한 유효성 검증과 에러 정보 표시에 한계가 있다. HTML 표준 폼으로 회원 가입 폼을 구성하여 보자.

HTML

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Signup</title>
</head>
<body>
  <form action="/signup" method="POST">
    <input type="email" name="email" placeholder="Email"
      pattern="^[0-9a-zA-Z]([-_\.]?[0-9a-zA-Z])*@[0-9a-zA-Z]([-_\.]?[0-9a-zA
-Z])*\.[a-zA-Z]{2,3}$"
      required>
    <input type="password" id="password1" name="password1"
      placeholder="Password" pattern="^[a-zA-Z0-9]{4,10}$"
      required>
    <input type="password" id="password2" name="password2"
      placeholder="Confirm Password" pattern="^[a-zA-Z0-9]{4,10}$"
      required>
    <input type="submit" name="submit" value="Signup">
  </form>
  <script>
    const pw1 = document.getElementById('password1');
```

```
    const pw2 = document.getElementById('password2');

    function validatePassword() {
      if (pw1.value !== pw2.value) {
        pw2.setCustomValidity('패스워드가 일치하지 않습니다');
      } else {
        pw2.setCustomValidity('');
      }
    }

    pw1.addEventListener('change', validatePassword);
    pw2.addEventListener('change', validatePassword);
  </script>
</body>
</html>
```

RESULT

HTML 표준 폼은 아래와 같은 단점이 있다.

1. submit 버튼이 클릭되면 서버로 폼 데이터를 전송하고 페이지를 전환한다.
2. HTML 표준 폼이 제공하는 required, pattern, max, min, maxlength, minlength, step과 같은 유효성 검증 어트리뷰트를 사용하여 유효성을 검증할 수 있다. 하지만 명확한 에러 정보를 제공하지 않으며 포커스를 옮기면 에러 정보가 사라진다. 또한 에러 메시지 팝업의 스타일도 변경하기 어렵다.
3. 세밀한 유효성 검증을 위해서는 로직을 추가하여야 하며 연관된 필드의 유효성 검증이 필요한 경우, 연관 필드 모두를 검사하여야 한다.

이와 같은 단점이 있는 HTML 표준 폼으로 애플리케이션의 요구사항을 만족시키기 어렵다.

3. Angular 폼

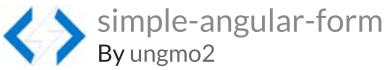
이번에는 Angular의 폼을 구성하여 보자. 하나의 입력 폼 컨트롤 요소를 작성하고 사용자 입력 데이터의 유효성을 검증하는 케이스이다.

TYPESCRIPT

```typescript
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  template: `
    <input type="text" (keyup.enter)="checkValue($event.target.value)">
    <em>{{ checkResult }}</em>
  `
})
export class AppComponent {
  checkResult: string;

  // 데이터를 입력하고 엔터 키를 눌렀을 때 입력 데이터의 길이를 검사
  checkValue(value) {
    this.checkResult = value.length > 3 ? '' : '4자 이상 입력하세요';
  }
}
```



이와 같이 템플릿 문법으로 거의 대부분의 폼을 구축할 수 있다. 그런데 만약 입력 필드가 하나가 아니라 여러 개 있다면 어떻게 해야 할까? 각각의 입력 필드마다 유효성 검증을 위한 이벤트 핸들러를 작성해야 하며, 이를 컴포넌트 템플릿의 폼과 이벤트 바인딩해야 한다. 또한 입력 필드 중에 하나라도 오류가 발생한 경우, 폼 전체의 처리를 중단시켜야 하는 등 처리가 복잡해 진다. 코드에는 중복이 발생할 것이고 복잡한 코드가 만들어져서 테스트도 힘들어 질 것이다.

위 예제와 같은 방식은 비교적 간단한 폼일 때는 유용한 방법이지만 복잡한 폼의 경우, 보다 효과적인 폼 데이터 변경 추적과 유효성 검증 및 에러 처리가 필요하다.

Angular는 HTML 표준 폼의 단점과 한계를 보완하고 효과적인 폼 데이터 변경 추적과 유효성 검증 및 에러 처리를 지원하는 템플릿 기반 폼과 리액티브(모델 기반) 폼을 제공한다.

Reference

- Form data validation
- Form Validation UX in HTML and CSS