AngularJS

Unit 3

AngularJS UP & Running by Shyam Seshadri & Brad green

Faculty:Fiona Coutinho

Chapter 1: Introducing AngularJS

- AngularJS is a **JavaScript MVC framework** for the Web
- Apply standard tried-and-tested **software engineering practices** traditionally used on the server side in **client-side programming** to accelerate frontend development.
- It provides a **consistent scalable structure** thus easy to develop large, complex applications
- It’s all done in **pure JavaScript and HTML**. No need to learn new language

What Is MVC (Model-View-Controller)?

- **MVC** architectural **pattern** gives developers starting point to decide ,**how** and **where** to **split** responsibilities.
- The MVC architectural pattern divides an application into three parts:

1. **The *model:*** Its generally the **data** behind the application, fetched usually from the server. Any UI data the user sees is derived from here.
2. **The *view:*** is the **UI** that the user sees and interacts with. It is dynamic, and generated based on the current model of the application.
3. **The *controller:*** is the **business** logic and **presentation layer**, which performs actions such as **fetching data**, and makes decisions such as how to **present** the model, which **parts** of it to **display**, etc.

MVC Advantages

- Each unit is **responsible** for **one** and only one thing. The model is the data, the view is the UI, and the controller is the business logic. Figuring out where the **new code** we are working on **belongs**, as well as finding prior code, is easy .
- Each unit is **independent** from the others. This makes the code much more **modular** and **reusable**, as well as easy to **maintain**.

AngularJS Benefits

1. AngularJS is a **Single Page Application (SPA) meta-framework** so clients need to focus only on core application .
2. An AngularJS application will require **fewer lines of code**(clear code) to complete a task than a pure JavaScript solution using jQuery
3. AngularJS's **declarative** nature makes it **easier to write and understand** applications just by looking at the HTML and the controllers
4. AngularJS applications can be **styled using CSS** and HTML independent of their business logic and functionality
5. AngularJS application templates are **written in pure HTML**, so designers find it **easier to work** with and style them.
6. Its **simple** to unit **test** AngularJS applications
7. **Supports third-party** component libraries and gives hooks to integrate them

A Basic AngularJS Application

```
<!-- File: chapter1/basic-angularjs-app.html -->
<!DOCTYPE html>
<html ng-app>
<body>
<h1>Hello {{1 + 2}} </h1>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.2.19/angular.js">
</script>
</body>
</html>
```

A Basic AngularJS Application

- There are two parts to starting an AngularJS application:

1.Loading the AngularJS source code*: T*he unminified version directly from the Google Hosted Libraries is included(you could have your own local version). The Google CDN hosts all the latest versions of AngularJS that you can directly reference or download it from the AngularJS website.

2. ***Bootstrapping AngularJS:*** done with the **ng-app** directive. This denotes the section of HTML that AngularJS controls. Putting it on the <html> tag tells AngularJS to control the entire HTML application(can put it on <body> or any other element on the page).Any child element of that will be handled with AngularJS and anything outside would not be processed.

A Basic AngularJS Application

- The double curly **{{}}** denotes either **one-way data-binding or exp**ressions.
- If it refers to a variable,it keeps the UI up to date with changes in the value.
- If it is an expression, AngularJS evaluates it and keeps the UI up to date if the value of the expression changes.
- If for any reason AngularJS had not bootstrapped correctly, we would have seen {{1 +2}} in the UI, but if there are no errors, we should see 3 as output.

AngularJS Hello World Example

In this example we allow users to type in their name, as the user types, we update the UI with the latest value

```
<!-- File: chapter1/angularjs-hello-world.html -->
<!DOCTYPE html>
<html>
<body ng-app>
<input type="text“ ng-model="name“ placeholder="Enter your name">
<h1>Hello <span ng-bind="name"></span></h1>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.2.19/angular.js">
</script>
</body>
</html>
```

AngularJS Hello World:Explanation

- input tag, has a directive called ng-model on it.
- The ng-model directive is used with input fields whenever we want the user to enter any data and get access to the value in JavaScript.
- Here AngularJS stores the value that the user types into this field in a variable called name.
- Another directive ng-bind(interchangeable with double-curly notation)
- so instead of <span ng-bind="name"></span>, we can write {{ name }}. Both put the value of the variable name inside the tag, and keeping it up to date if it changes.

Chapter 2: Basic AngularJS Directives and Controllers: AngularJS Modules

- Modules :a way of packaging relevant code under a single name
- An AngularJS **module** has **two parts** to it:

1.A module can define its controllers, services, factories, and directives.
-These are functions and code that can be accessed throughout the module.

2.The module can depend on other modules as *dependencies*, which are defined when module is instantiated .

-AngularJS can be told what module to load as the main entry point for the application by passing the module name to the ng-app directive.

Chapter 2: Basic AngularJS Directives and Controllers:AngularJS Modules

- This is how we define a module named notesApp:

```
angular.module('notesApp', []);
```

- The *first argument* to the module function is the ***name of the module***.
- The *second argument* is an ***array of module names*** that this module depends on.
- Note the **empty square brackets** tells AngularJS to create a new module with the name notesApp, with **no dependencies**.
- This is how we define a module named notesApp, which depends on two other modules : **notesApp.ui**, which defines our UI widgets, and **thirdCompany.fusioncharts** , which is a third-party library for charts:

```
angular.module('notesApp',['notesApp.ui','thirdCompany.fusioncharts']);
```

Chapter 2: Basic AngularJS Directives and Controllers:AngularJS Modules

```
<!-- File: chapter2/module-example.html -->
<html ng-app="notesApp">
<head><title>Hello AngularJS</title></head>
<body>
Hello {{1 + 1}}nd time AngularJS <!--notesApp.htm-->
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.2.19/angular.js">
</script>
<script type="text/javascript">
angular.module('notesApp', []);
</script>
</body>
</html>
```

Chapter 2: Basic AngularJS Directives and Controllers:Creating Our First Controller

- It acts as the gateway between our model, which is the data that drives our application, and the view, which is what the user sees and interacts with.

Task of controller:

1.Fetching the right data from the server for the current UI

2.Deciding which parts of that data to show to the user

3. Presentation logic, such as how to display elements, which parts of the UI to show, how to style them, etc.

4.User interactions, such as what happens when a user clicks something or how a text input should be validated

Chapter 2: Basic AngularJS Directives and Controllers:Creating Our First Controller

```
<html ng-app="notesApp">
<head><title>Hello AngularJS</title></head>
<body ng-controller="MainCtrl">Hello {{1 + 1}}nd time AngularJS
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.2.19/angular.js">
</script>
<script type="text/javascript">
angular.module('notesApp', [])
.controller('MainCtrl', [function() {
// Controller-specific code goes here
console.log('MainCtrl has been created');
                                                  }]);
</script>
</body></html>
```

Chapter 2: Basic AngularJS Directives and Controllers: Controller code explanation

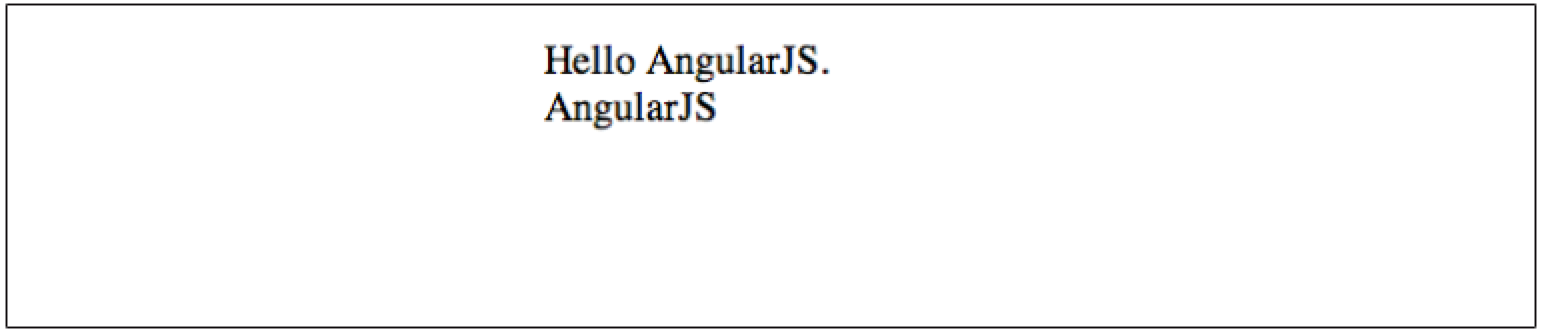
- We define a controller using the controller function that is exposed on an AngularJS module.
- The controller function takes the name of the controller as the first argument(MainCtrl),the second argument is the actual controller definition(what/how it does)
- The directive, ng-controller tells AngularJS to instantiate an instance of the controller with the given name, and attach it to the DOM element
- In the next AngularJS application the “hello world” message from the HTML is put in the controller, so get and display it from the controller.
- ng-controller directive associates an instance of a controller with a UI element (in this case, the body tag).

Chapter 2: Basic AngularJS Directives and Controllers

```
<html ng-app="notesApp"><!--notesApp1.htm-->
<head><title>Notes App</title></head><body ng-controller="MainCtrl as ctrl">
{{ctrl.helloMsg}} AngularJS.<br/>
{{ctrl.goodbyeMsg}} AngularJS
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.2.19/angular.js"></script>
<script type="text/javascript">
angular.module('notesApp', [])
.controller('MainCtrl', [function() {
                        this.helloMsg = 'Hello ';
                        var goodbyeMsg = 'Goodbye ';
                        }]);
</script>
</body>
</html>
```

Chapter 2: Basic AngularJS Directives and Controllers

- **Output:** we only see “Hello AngularJS.” The “Goodbye” message is not printed in the UI. As We defined the variable helloMsg on the controller’s instance and the variable goodbyeMsg as a local variable in the controller’s instance.



```
Hello AngularJS.
AngularJS
```

Chapter 2: Basic AngularJS Directives and Controllers

```
<html ng-app="notesApp"><head><title>Notes App</title></head><!-- notesApp2.htm -- >
<body ng-controller="MainCtrl as ctrl">
{{ctrl.message}} AngularJS.
<button ng-click="ctrl.changeMessage()">Change Message</button>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.2.19/angular.js">
</script>
<script type="text/javascript">
angular.module('notesApp', [])
.controller('MainCtrl', [function() {
var self = this;
self.message= 'Hello ';
self.changeMessage = function() {
self.message = 'Goodbye';
};}]);
</script></body></html>
```

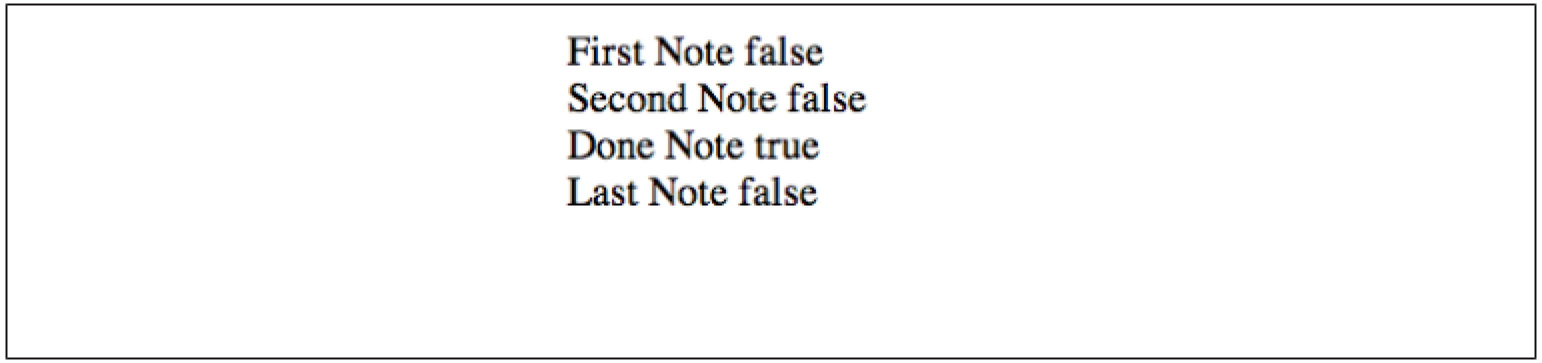
Chapter 2: Basic AngularJS Directives and Controllers

- In this code there is one binding the ctrl.message variable.
- There is a button with the label “Change Message.” To built-in directive ng-click we pass a function as an argument.
- The ng-click directive evaluates any expression passed to it when the button is clicked. In this case it triggers the controller’s changeMessage() function & sets the value of message to “Goodbye.”
- we use proxy variable ‘self’ instead of the this keyword.
- When the user clicked the button and changeMessage was triggered, UI updated automatically. For this reason AngularJS is a data-driven app.
- The HTML connects parts of the DOM to controllers, functions, and variables, and not vice versa.

Working with and Displaying Arrays

```
<html ng-app="notesApp"><!--notesApp4.htm -- >
<head><title>Notes App</title></head><body ng-controller="MainCtrl as ctrl">
<div ng-repeat="note in ctrl.notes">
<span class="label"> {{note.label}}</span>
<span class="status" ng-bind="note.done"></span>
</div><script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.2.19/angular.js">
</script>
<script type="text/javascript">
angular.module('notesApp', [])
.controller('MainCtrl', [function() { var self = this;
self.notes = [
{id: 1, label: 'First Note', done: false},
{id: 2, label: 'Second Note', done: false},
{id: 3, label: 'Done Note', done: true},
{id: 4, label: 'Last Note', done: false}
];
}]);
</script></body></html>
```

Working with and Displaying Arrays
Output:



```
First Note false
Second Note false
Done Note true
Last Note false
```

Working with and Displaying Arrays: code explanation

- Here we introduced an array of JSON objects in our MainCtrl. We exposed this on the controller instance with the name notes.
- In our HTML, we used a directive called **ng-repeat** which iterates over an array or over the keys and values of an object and displays them in the HTML.
- In ng-repeat directive: the contents of the element on which the directive is applied is considered as *template.*AngularJS picks this template,makes a copy of it, and then applies it for each instance of the ng-repeat.
- The label and status span elements were repeated four times, once for each item in the notes array
- The ng-repeat is the same as a for each loop in any programming language,the syntax is: **ng-repeat="eachVar in arrayVar"**
- Inside the context of the ng-repeat, we have a new variable, note, which is not present in our controller. This is created by ng-repeat, and each instance of the ng-repeat has its own version and value of note, obtained from each item of the array.
- we used the double-curly notation to print the note's label but used a directive ng-bind for the note's done field, Both take the value from the controller and display in UI.

ng-cloak Directive

- *ng-cloak* directive is a mechanism to hide sections of the page while AngularJS bootstraps and finishes loading.
- ng-cloak uses the following CSS rules, which are automatically included when you load *angular.js* or *angular.min.js*:

  .ng-cloak{

  display: none !important; }
- any section or element that needs to be hidden in your HTML needs to have class="ng-cloak" added to it.
- After AngularJS finishes loading, it goes through HTML and removes ng-cloak from all elements so that UI is visible after bootstrapping.

ng-cloak Directive

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js">
</script>
<body>
<div ng-app="">
<p class="ng-cloak">{{ 5 + 5 }}</p>
</div>
</body>
</html>
```

Advantage Using ng-bind Versus Double Curly

- For a portion of a second while the browser starts, you might see flashing double curly braces in the UI
- This is only for the very first page load because double-curly notation takes time to bootstrap and execute before replacing all the double curly braces from the HTML.
- Instead use ng-bind or use {{}} along with the ng-cloak directive.

How AngularJS is working behind?

1. Once **HTML** is loaded it triggers requests for all the **scripts** that are a part of it.
2. After the entire document has been loaded, AngularJS enters and looks for **ng-app** directive.
3. Once found, searches/loads the **module** specified and attaches it to element.
4. It traverses the **children DOM elements** of the root element with the ng-app and looks for **directives and bind** statements.
5. Each time it sees an **ng-controller** or an **ng-repeat directive**, it creates a *scope* *(*context for that element). The **scope** dictates what each DOM element has access to (functions, variables,etc).
6. AngularJS adds **watchers** and **listeners** on the **variables** that the HTML accesses and tracks current value of each. Once value changes, UI updated immediately.
7. It **checks** for UI updates only on certain **events**, which can change the model for eg. XHR or server calls, page loads, and user interactions like click or type.

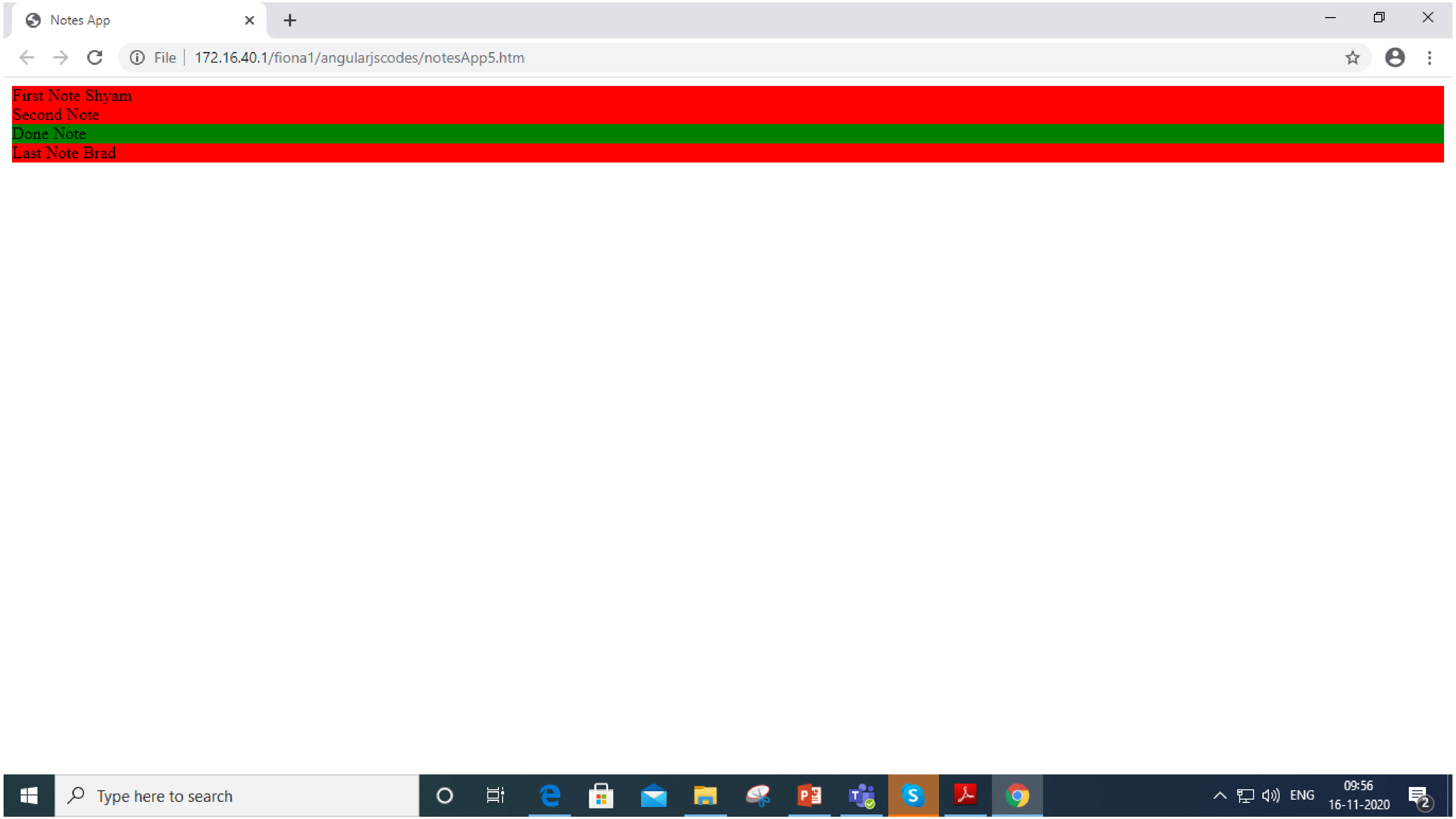
More Directives

```
<html ng-app="notesApp"><head><title>Notes App</title>
<style>
.done {
background-color: green;
}
.pending {
background-color: red;
}</style></head><body ng-controller="MainCtrl as ctrl">
<div ng-repeat="note in ctrl.notes“ ng-class="ctrl.getNoteClass(note.done)">
<span class="label"> {{note.label}}</span>
<span class="assignee“ ng-show="note.assignee“ ng-bind="note.assignee">
</span>
</div>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.2.19/angular.js"></script>
```

More Directives

```
<script type="text/javascript">
angular.module('notesApp', [])
.controller('MainCtrl', [function() {
var self = this;
self.notes = [{label: 'First Note', done: false, assignee: 'Shyam'},
                    {label: 'Second Note', done: false},
                    {label: 'Done Note', done: true},
                    {label: 'Last Note', done: false, assignee: 'Brad'}
];
self.getNoteClass = function(status) {return {done: status, pending: !status};};
}]);
</script>
</body>
</html>
```

More Directives



```
First Note Shyam
Second Note
Done Note
Last Note Brad
```

More Directives

1.ng-show: There are two directives in AngularJS that deal with hiding and showing HTML elements: **ng-show and ng-hide which** show or hide elements by finding the truthiness of value of a variable.

- previous program ,showed the assignee span only if note.assignee is true(true,nonempty strings, nonzero numbers, and non-null JS objects are truthy)

2.ng-class:used to selectively apply/remove CSS classes from elements it takes strings or objects as values.

- If it is a string, it simply applies the CSS classes directly.
- If it is an object it looks at each key of the object, and depending on the value for that key is true or false, applies/removes CSS class.

More Directives

- In our program:the CSS class done gets added and pending is removed if **note.done is true** and done gets removed and pending gets added if **note.done is false**.

Working with ng-repeat: ng-repeat Over an Object

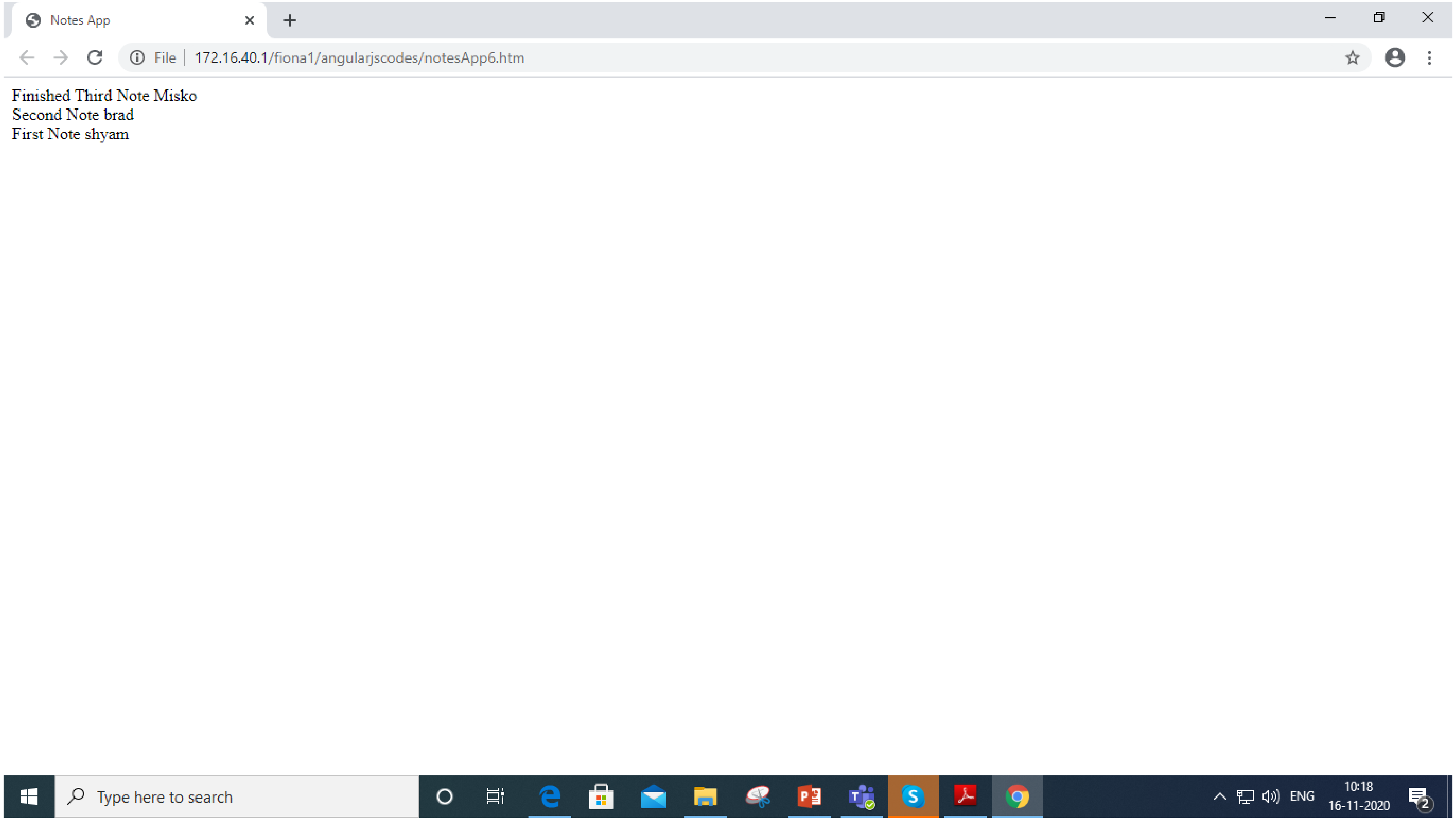
- **ng-repeat directive** can be used to show all the keys and values of an object just like array of elements :

```
<html ng-app="notesApp">
<head><title>Notes App</title></head>
<body ng-controller="MainCtrl as ctrl">
<div ng-repeat="(author, note) in ctrl.notes">
<span class="label"> {{note.label}}</span>
<span class="author" ng-bind="author"></span>
</div>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.2.19/angular.js">
</script>
```

Working with ng-repeat

```
<script type="text/javascript">angular.module('notesApp', [])
.controller('MainCtrl', [function() {
var self = this;
self.notes = { shyam: {id: 1,label: 'First Note',done: false},
                    Misko: {id: 3,label: 'Finished Third Note',done: true},
                   brad: {id: 2,label: 'Second Note',done: false}
                  }
                                                    ;}
                                     ]
                );
</script>
</body>
</html>
```

Working with ng-repeat



Notes App

File | 172.16.40.1/fiona1/angularjscodes/notesApp6.htm

```
Finished Third Note Misko
Second Note brad
First Note shyam
```

Working with ng-repeat

- In this example, Misko is capitalized while brad and shyam lowercase.
- ng-repeat directive over an object sorts the keys of the object in case-sensitive, alphabetic order(uppercase first, and then sorted by alphabet). So the items would be shown in the following order: Misko, brad, shyam.
- The ng-repeat directive takes an argument in the form variable in arrayExpression or (key, value) in objectExpression.
- When used with an array, the items will be in the order in which they exist in the array.

Helper Variables in ng-repeat

- The ng-repeat directive exposes some variables within the HTML context that gets repeated, and gives insight into the current element:

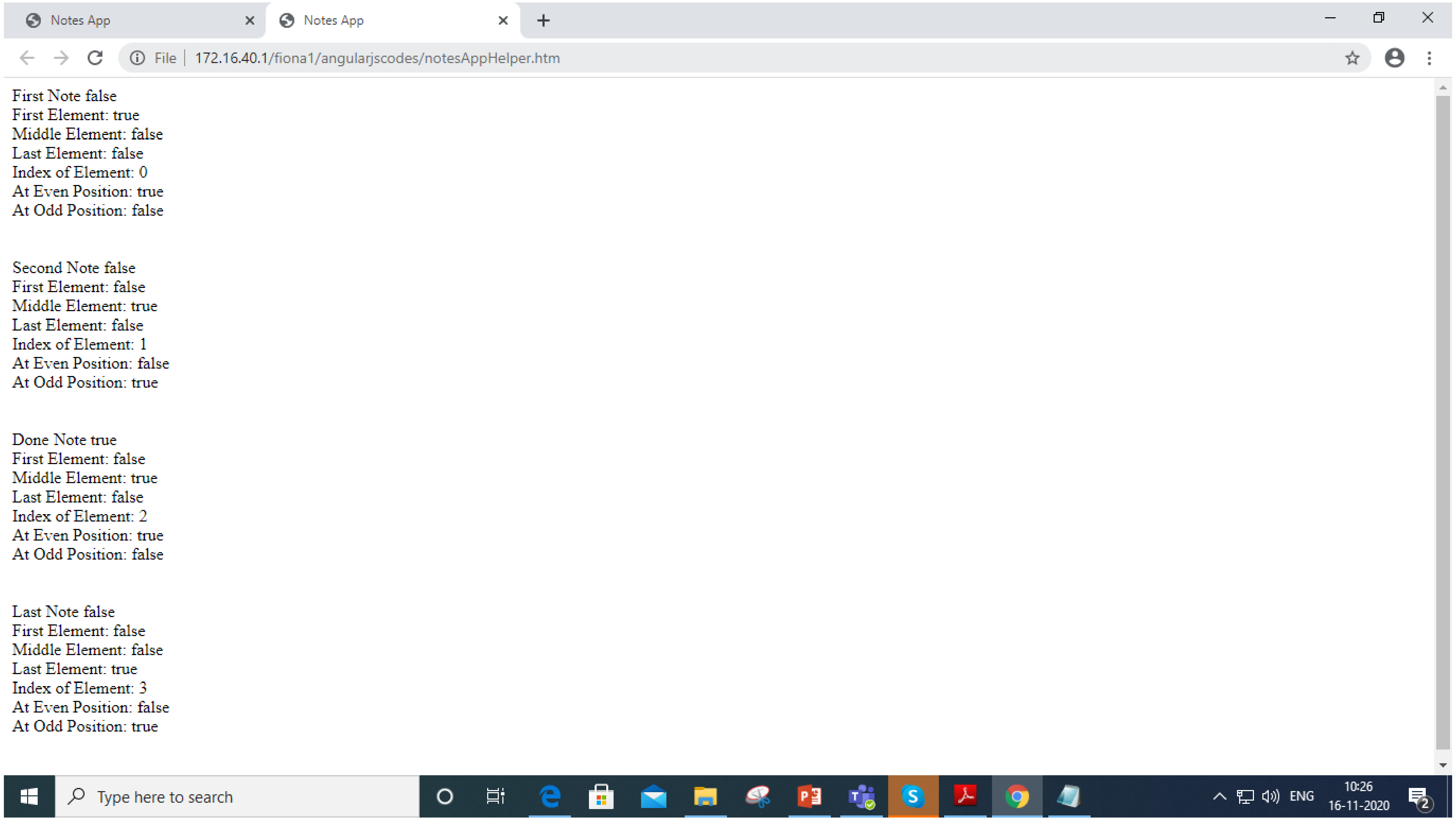
```
<html ng-app="notesApp">
<head><title>Notes App</title></head>
<body ng-controller="MainCtrl as ctrl">
<div ng-repeat="note in ctrl.notes">
<div>First Element: {{$first}}</div>
<div>Middle Element: {{$middle}}</div>
<div>Last Element: {{$last}}</div>
<div>Index of Element: {{$index}}</div>
<div>At Even Position: {{$even}}</div>
<div>At Odd Position: {{$odd}}</div>
<span class="label"> {{note.label}}</span>
<span class="status" ng-bind="note.done"></span>
<br/><br/>
</div>
```

Helper Variables in ng-repeat

```
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.2.19/angular.js"></script>
<script type="text/javascript">
angular.module('notesApp', [])
.controller('MainCtrl', [function() {
var self = this;self.notes = [
{id: 1, label: 'First Note', done: false},
{id: 2, label: 'Second Note', done: false},
{id: 3, label: 'Done Note', done: true},
{id: 4, label: 'Last Note', done: false}];}]);
</script>
</body>
</html>
```

- In this example, we use the same array as before but the difference is we display more state about the item being repeated in the HTML. For each item, we display which index the item is in, and whether it is the first, middle, last, odd, or even item.

Helper Variables in ng-repeat



```
First Note false
First Element: true
Middle Element: false
Last Element: false
Index of Element: 0
At Even Position: true
At Odd Position: false


Second Note false
First Element: false
Middle Element: true
Last Element: false
Index of Element: 1
At Even Position: false
At Odd Position: true


Done Note true
First Element: false
Middle Element: true
Last Element: false
Index of Element: 2
At Even Position: true
At Odd Position: false


Last Note false
First Element: false
Middle Element: false
Last Element: true
Index of Element: 3
At Even Position: false
At Odd Position: true
```

Helper Variables in ng-repeat

- The $ prefixed variables used within the ng-repeat provided by AngularJS refer to the state of the repeater for that particular element.
- They include: $first, $middle, and $last (Boolean values) that tell if that particular element is the first, between the first and last, or the last element in the array or object.
- $index gives us the index or position of the item in the array.
- $odd and $even tell us if the item is in an index that is odd or even (we could use this for conditional styling of elements)

Track by ID

- By default, ng-repeat creates a new DOM element for each value in the array or object that we iterate over.
- Optimize performance, it caches or reuses DOM elements if the objects are exactly the same, according to the hash of the object
- If we want to reuse the same DOM element, even if the object instance does not hash to the same value. For eg. if we have objects coming from a database & we want AngularJS to treat two objects with the same ID as identical for the purpose of the repeat. For this purpose, AngularJS provides a tracking expression when specifying our ng-repeat:

```
<html ng-app="notesApp">
<body ng-controller="MainCtrl as ctrl">
<button ng-click="ctrl.changeNotes()">Change Notes</button>
<br/>DOM Elements change every time someone clicks
<div ng-repeat="note in ctrl.notes1">
{{note.$$hashKey}}
<span class="label"> {{note.label}}</span><span class="author" ng-bind="note.done"></span>
</div>
```

- DOM Elements are reused every time someone clicks

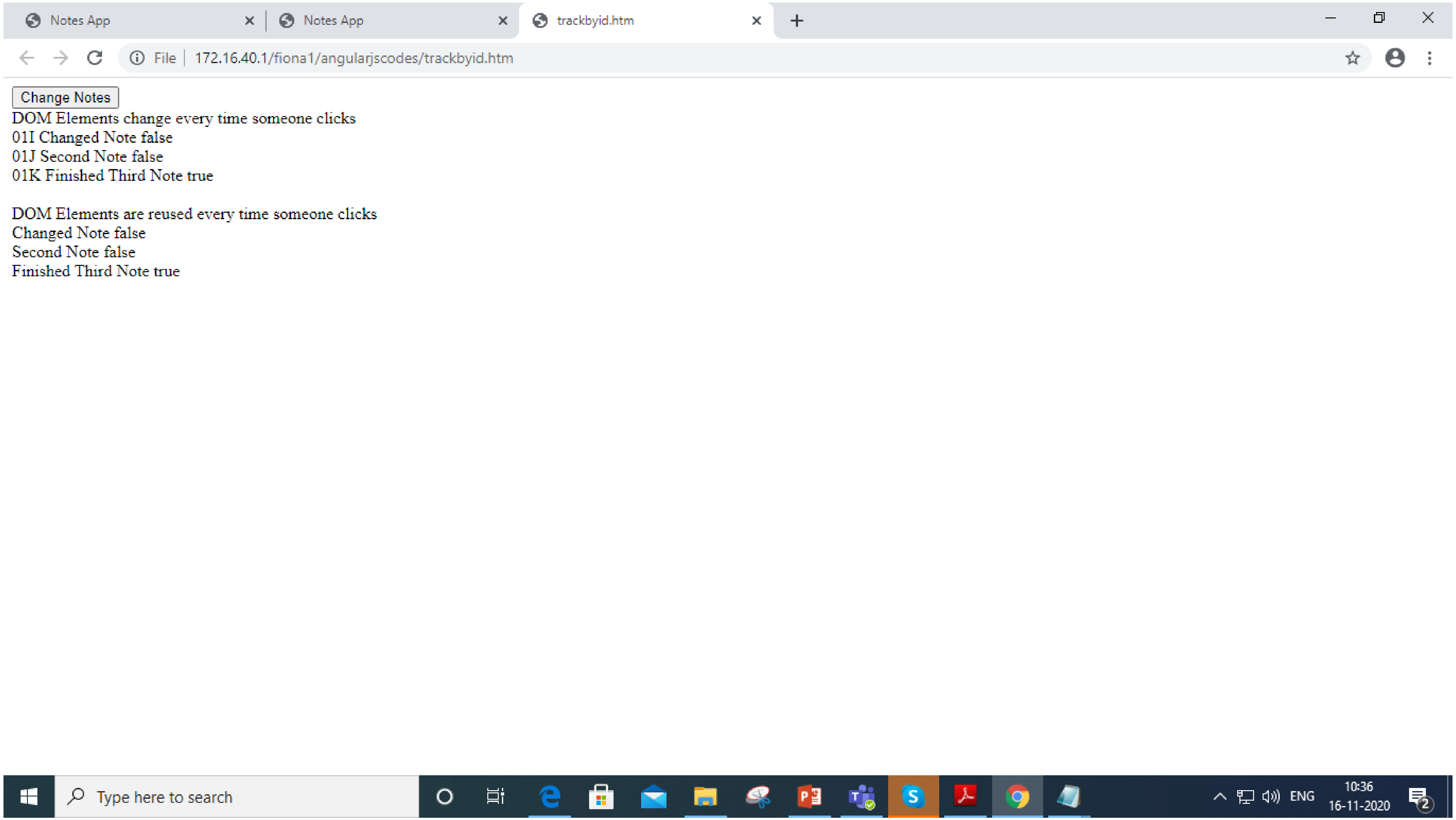
```
<div ng-repeat="note in ctrl.notes2 track by note.id">{{note.$$hashKey}}
<span class="label"> {{note.label}}</span>
<span class="author" ng-bind="note.done"></span></div>
```

Track by ID

```
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.2.19/angular.js">
</script>
<script type="text/javascript">
angular.module('notesApp', [])
.controller('MainCtrl', [function() {
var self = this; var notes = [
{id: 1,label: 'First Note',done: false,someRandom: 31431},
{id: 2,label: 'Second Note',done: false},
{id: 3,label: 'Finished Third Note',done: true}
];
```

Track by ID

```
self.notes1 = angular.copy(notes);
self.notes2 = angular.copy(notes);
self.changeNotes = function() { notes = [
        {id: 1,label: 'Changed Note',done: false,someRandom: 4242},
        {id: 2,label: 'Second Note',done: false},
        {id: 3,label: 'Finished Third Note',done: true}
];
self.notes1 = angular.copy(notes);
self.notes2 = angular.copy(notes);
};}]);
</script></body></html>
```



```
DOM Elements change every time someone clicks
01I Changed Note false
01J Second Note false
01K Finished Third Note true

DOM Elements are reused every time someone clicks
Changed Note false
Second Note false
Finished Third Note true
```

Track by ID

- Here we have two arrays, notes1 and notes2,(identical). Both displayed in UI using ng-repeat. The difference is one uses the plain ng-repeat (ng-repeat="note in ctrl.notes1") and the other uses the track by ID version (ng-repeat="note in ctrl.notes2 track by note.id").
- ng-click triggers a function in controller when someone clicks that element.Here changeNotes() is called which changes the notes arrays to a new array.
- Notice the hashKeys and the DOM elements in the first ng-repeat are getting changed every time we click a button.
- In the second ng-repeat, no $$hashKey needs to generate, as unique identifier for each element is mentioned. So DOM elements are reused based on ID of object.
- Do not use any variables that start with $$ in your application(they denote private variables (for its own Purpose))
- Track by ID ensure AngularJS reuses DOM elements even if we fetch the data multiple times from the server.

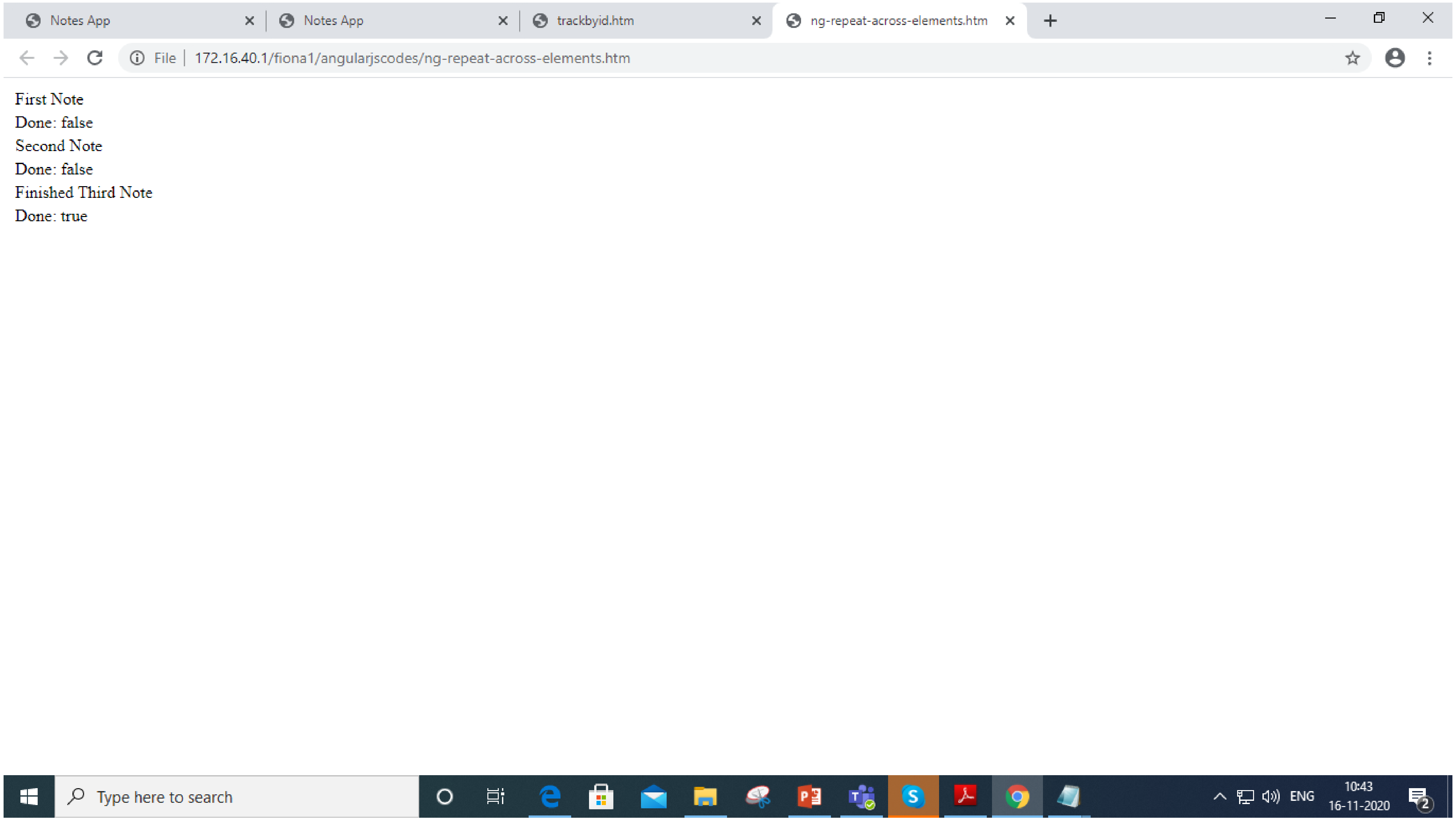
ng-repeat Across Multiple HTML Elements

- Requirement to repeat multiple sibling HTML elements that may not be in a single container element.
- For example, repeat two table rows (<tr>) for each item in our array, maybe one as a header row and one as a child row.
- For these situations, AngularJS provides the ability to mark where our ng-repeat starts and which HTML element is considered,It uses ng-repeat-start and ng-repeat-end directives:

```
<html ng-app="notesApp">
<body ng-controller="MainCtrl as ctrl">
```

ng-repeat Across Multiple HTML Elements

```
<table><tr ng-repeat-start="note in ctrl.notes"><td>{{note.label}}</td></tr>
<tr ng-repeat-end><td>Done: {{note.done}}</td>
</tr></table>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.2.19/angular.js">
</script>
<script type="text/javascript">
angular.module('notesApp', [])
.controller('MainCtrl', [function() { var self = this;
self.notes = [ {id: 1, label: 'First Note', done: false},
{id: 2, label: 'Second Note', done: false},
{id: 3, label: 'Finished Third Note', done: true}];}]);
</script></body>
</html>
```


Notes App
Notes App
trackbyid.htm
ng-repeat-across-elements.htm
File
172.16.40.1/fiona1/angularjscodes/ng-repeat-across-elements.htm
Type here to search
ENG
10:43
16-11-2020

```
First Note
Done: false
Second Note
Done: false
Finished Third Note
Done: true
```

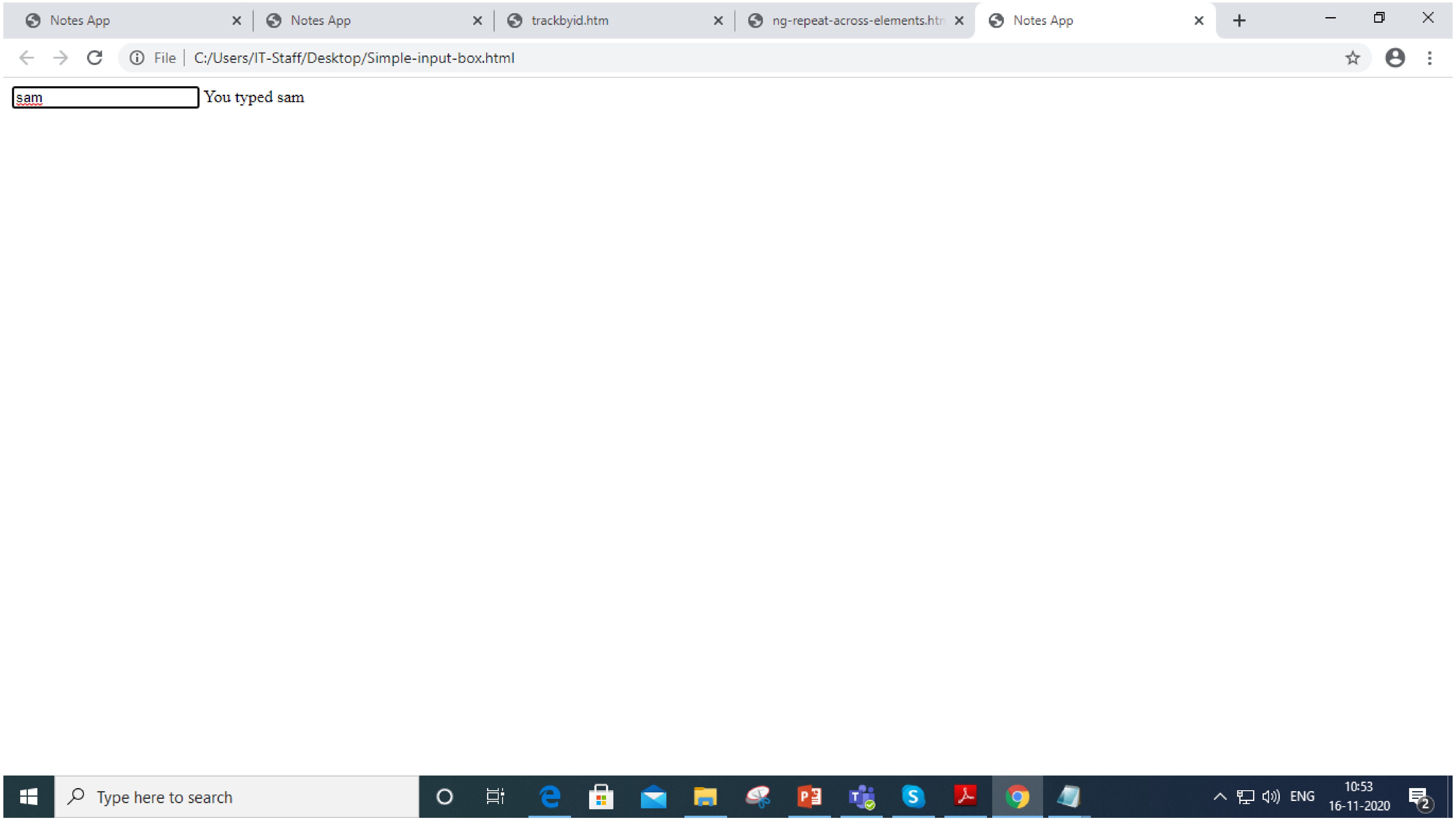
ng-repeat Across Multiple HTML Elements

- In this example, we create a table to display the list of notes.
- For each note, we have a row displaying the label, followed by a second table row displaying the status of the note.
- We mark the first tr as where our ng-repeat starts, and use our traditional ng-repeat expression as the argument.
- We then defined our template containing the first element with label, and then in second table row element we mark this element as where our repeater ends using ng-repeat-end directive.
- AngularJS ensures that it creates both tr elements for each element in the array

Chapter 4: Forms, Inputs, and Services
Working with ng-model

- ng-bind directive or double-curly {{ }} notation offer one-way data-binding
- But most applications have user interaction,From registration forms to profile information.AngularJS provides ng-model directive to get inputs and for two-way data-binding:

```
<html ng-app="notesApp">
<head><title>Notes App</title></head>
<body ng-controller="MainCtrl as ctrl">
<input type="text" ng-model="ctrl.username"/>
You typed {{ctrl.username}}
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.2.19/angular.js">
</script>
<script type="text/javascript">
angular.module('notesApp', [])
.controller('MainCtrl', [function() {
this.username = 'nothing';}]);
</script>
</body>
</html>
```



C:/Users/IT-Staff/Desktop/Simple-input-box.html

sam You typed sam

Chapter 4: Working with ng-model

- In this example, we have a controller with a variable username. we get its value out into the HTML using the ng-controller and the double-curly for one-way data-binding. we also have a text box on it we have attached the ng-model directive.
- value for the ng-model is pointed to the same username variable on the MainCtrl.

This Does the following:

- When the user types the value in the input box, it updates the model in controller.
- When the value of the variable changes in the controller input field updated

Chapter 4:Working with ng-model

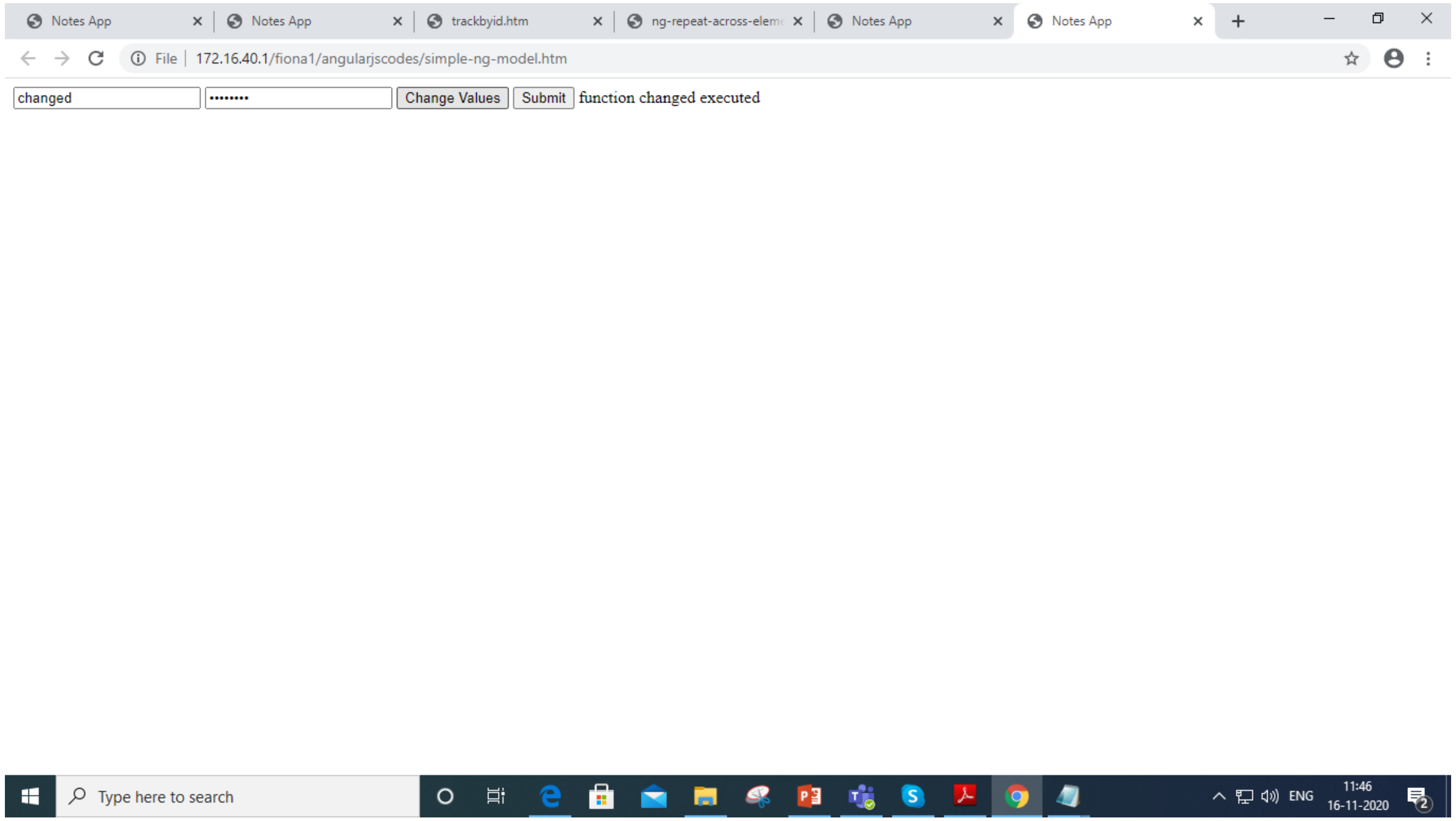
```
<html ng-app="notesApp">
<head><title>Notes App</title></head>
<body ng-controller="MainCtrl as ctrl">
<input type="text" ng-model="ctrl.username">
<input type="password" ng-model="ctrl.password">
<button ng-click="ctrl.change()">Change Values</button>
<button ng-click="ctrl.submit()">Submit</button>
<span ng-bind="ctrl.msg"/>
<script  src="https://ajax.googleapis.com/ajax/libs/angularjs/1.2.19/angular.js">
</script>
```

Chapter 4: Forms, Inputs, and Services
Working with ng-model

```
<script type="text/javascript">angular.module('notesApp', [])
.controller('MainCtrl', [function() {
var self = this;
self.str1 = 'User clicked submit with';
self.change = function() {self.username = 'changed';self.password =
'password';self.msg='function changed executed';};
self.submit = function() {self.msg = self.str1.concat(self.username);};}]);
</script>
</body>
</html>
```

Chapter 4: Forms, Inputs, and Services

Working with ng-model



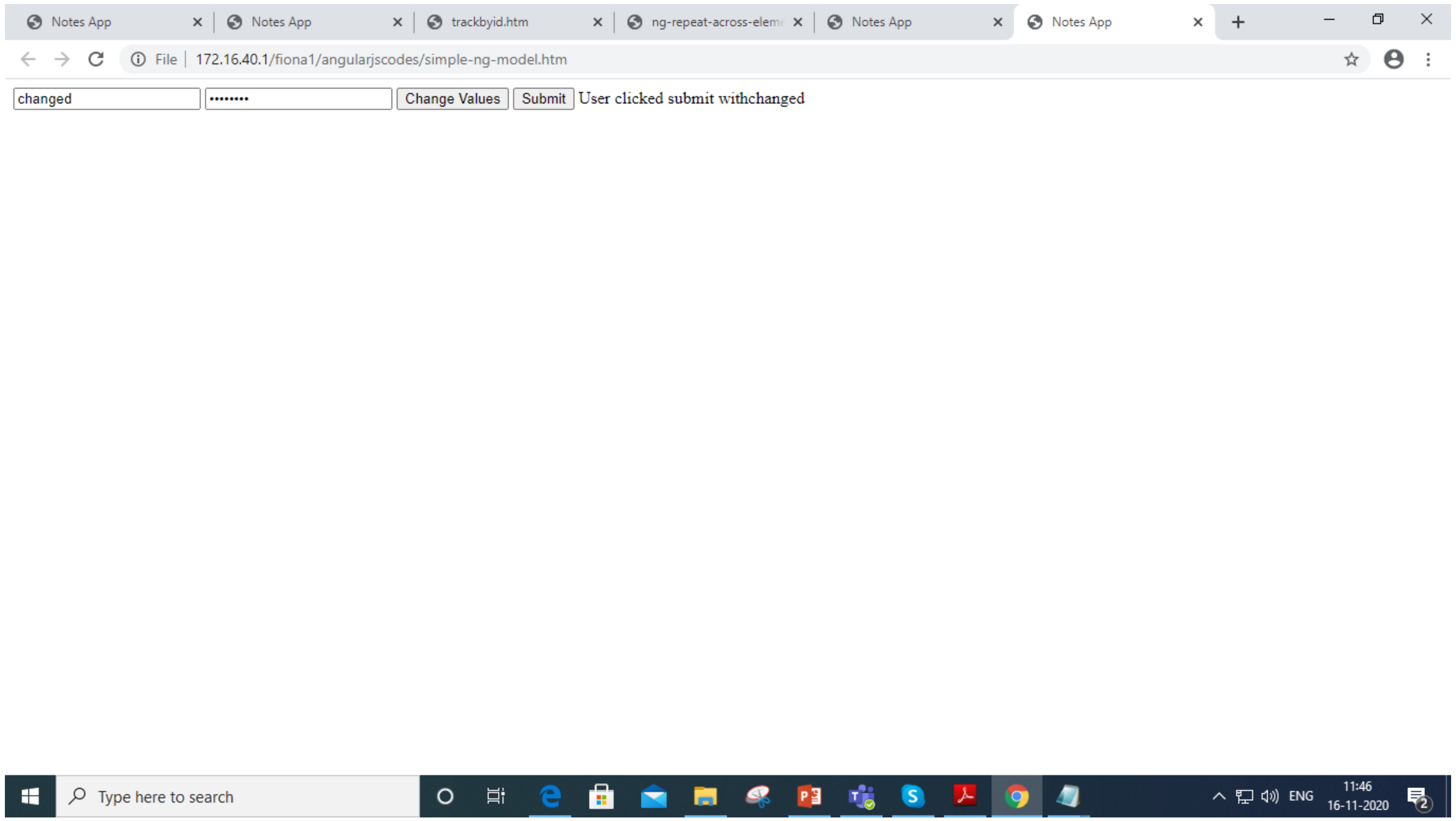
changed

••••••••

function changed executed

Chapter 4: Forms, Inputs, and Services

Working with ng-model



User clicked submit withchanged

Chapter 4: Forms, Inputs, and Services
Working with ng-model

- Here we added password input field bound to a field password on the

controller’s instance. we added two buttons:

- The first ChangeValues of username and password fields in the controller with the latest values.
- The second button, Submit, submits the form to the server.
- Point to note is the controller never reached out into the UI.
- There was no jQuery selector, no findElementById, or anything like that.
- When we need to update the UI, we just update the model fields in the controller.
- When we need to get the latest value, we just grab it from the controller.

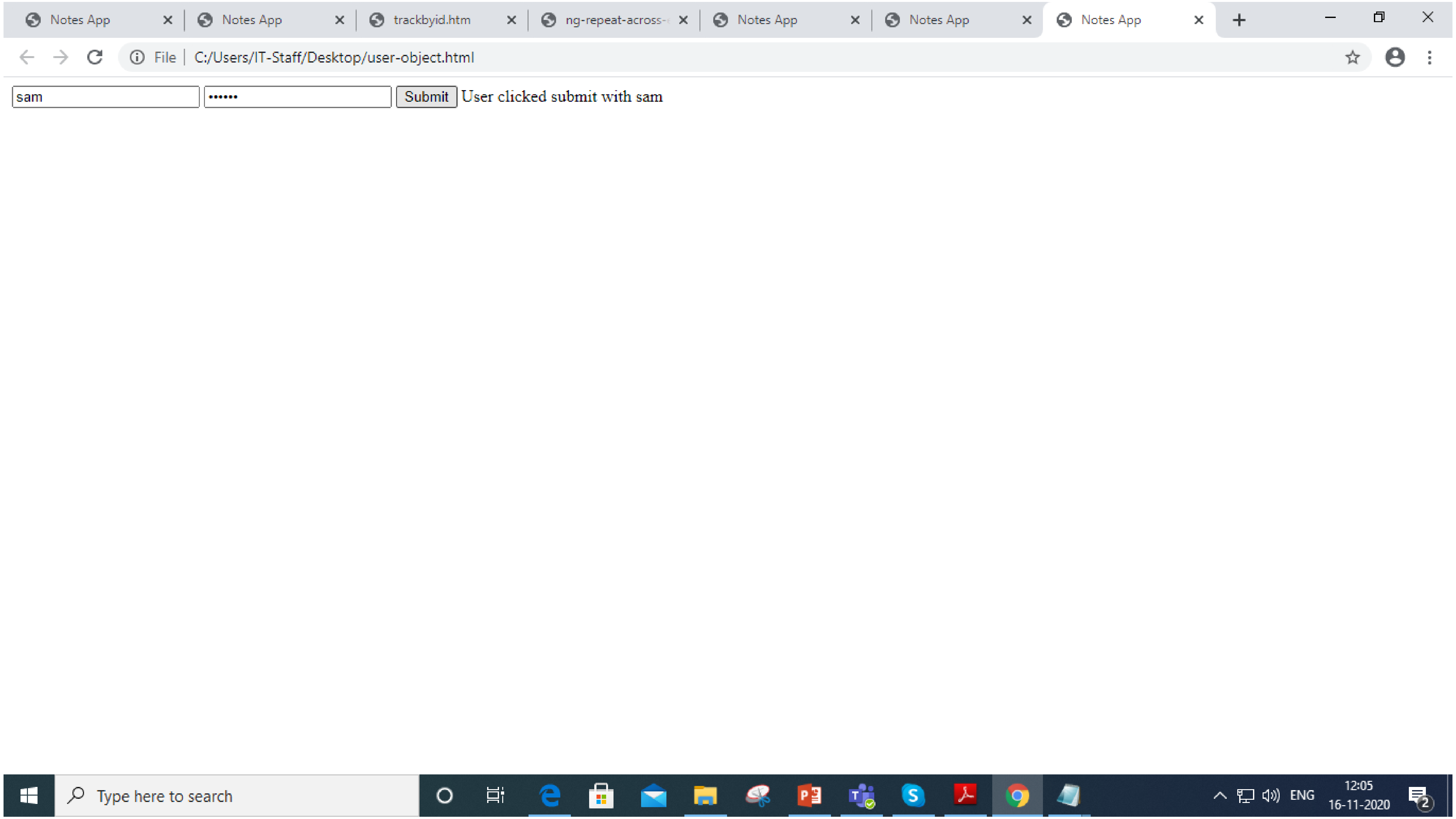
Chapter 4: Working with Forms

When using forms in AngularJS ng-model heavily used to get data into and out of the form :

```
<html ng-app="notesApp">
<head><title>Notes App</title></head>
<body ng-controller="MainCtrl as ctrl">
<form ng-submit="ctrl.submit()">
<input type="text" ng-model="ctrl.user.username">
<input type="password" ng-model="ctrl.user.password">
<input type="submit" value="Submit">
<span ng-bind="ctrl.msg"/>
</form>
```

Chapter 4: Forms, Inputs, and Services
Working with Forms

```
<scriptsrc="https://ajax.googleapis.com/ajax/libs/angularjs/1.2.19/angular.js">
</script>
<script type="text/javascript">
angular.module('notesApp', [])
.controller('MainCtrl', [function() {var self = this;
self.str1 = 'User clicked submit with  ';
self.submit = function() {self.msg =
self.str1.concat(self.user.username);};}]);</script>
</body>
</html>
```



User clicked submit with sam

Chapter 4: Forms, Inputs, and Services Working with Forms

- In this example text fields and button put in a form. instead of ng-click we have ng-submit directive
- The ng-submit directive advantages: A form submit event can be triggered by clicking the Submit button, or   hitting Enter on a text field unlike ng-click which is triggered on click of the button.
- Instead of binding to ctrl.username and ctrl.password, we bind to ctrl.user.username and ctrl.user.password.we did not declare a user object in controller on using ng-model, AngularJS automatically creates the objects and keys necessary for a data-binding connection.
- Only once user types into the username or password field, the object created. (The first letter typed into either the username or password field causes the user object to be created)

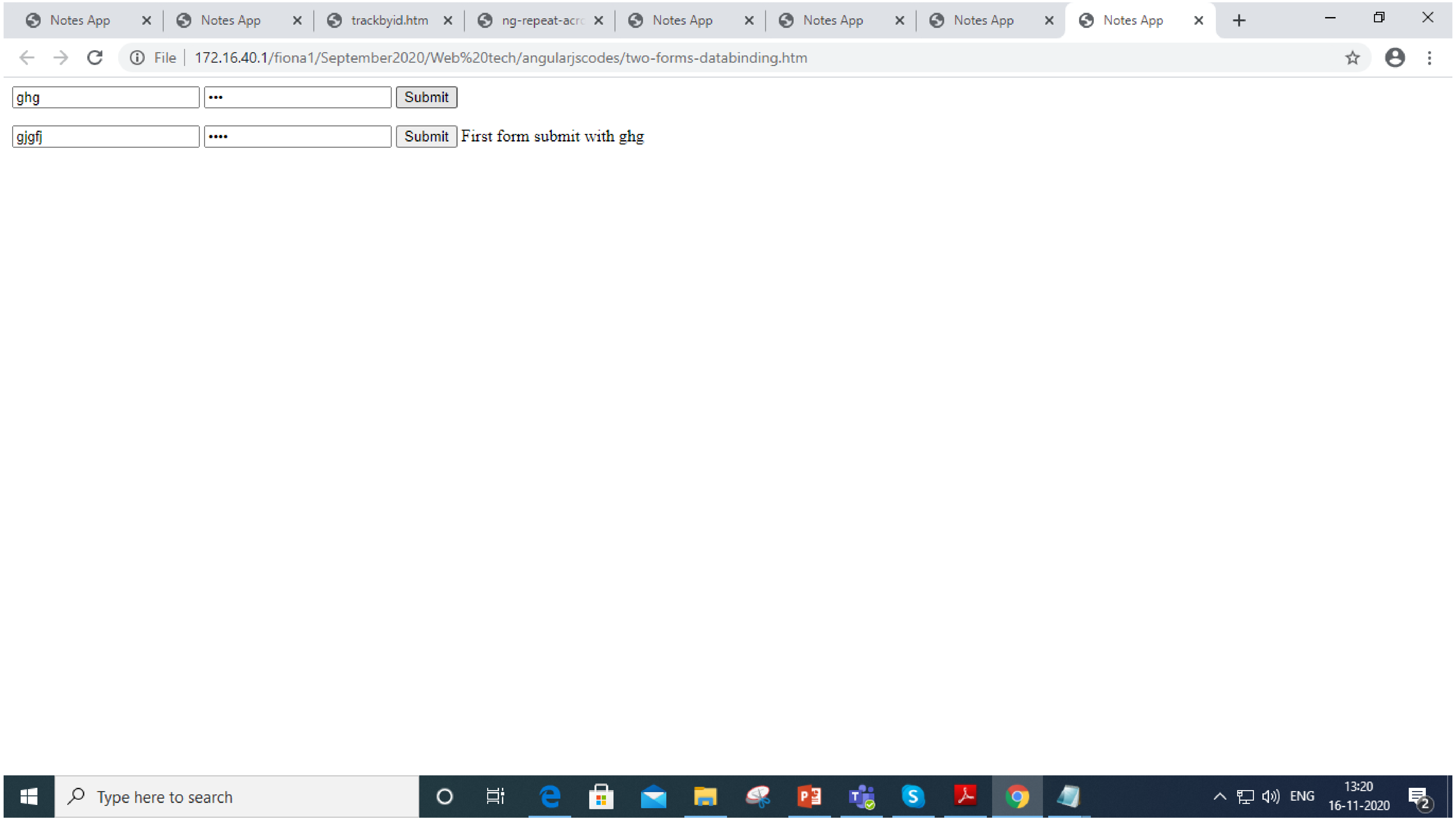
Chapter 4: Forms, Inputs, and Services Leverage Data-Binding and Models

- When designing your forms and deciding which fields to bind the ng-model to, always consider what format you need the data in.

```
<html ng-app="notesApp">
<head><title>Notes App</title></head>
<body ng-controller="MainCtrl as ctrl">
<form ng-submit="ctrl.submit1()">
<input type="text" ng-model="ctrl.username">
<input type="password" ng-model="ctrl.password">
<input type="submit" value="Submit"></form>
<form ng-submit="ctrl.submit2()"><input type="text" ng-model="ctrl.user.username">
<input type="password" ng-model="ctrl.user.password"><input type="submit" value="Submit">
<span ng-bind="ctrl.msg"/>
</form>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.2.19/angular.js">
</script>
```

Chapter 4: Forms, Inputs, and Services Leverage Data-Binding and Models

```
<script type="text/javascript">
angular.module('notesApp', [])
.controller('MainCtrl', [function() {var self = this;
self.str1 = 'First form submit with  ';
self.str2 = 'Second form submit with  ';
self.submit1 = function() {
// Create user object to send to the server
var user = {var user = {username: self.username, password: self.password};
 self.msg = self.str1.concat(user.username);};
self.submit2 = function() {self.msg = self.str2.concat(self.user.username);};}]);
</script></body>
</html>
```

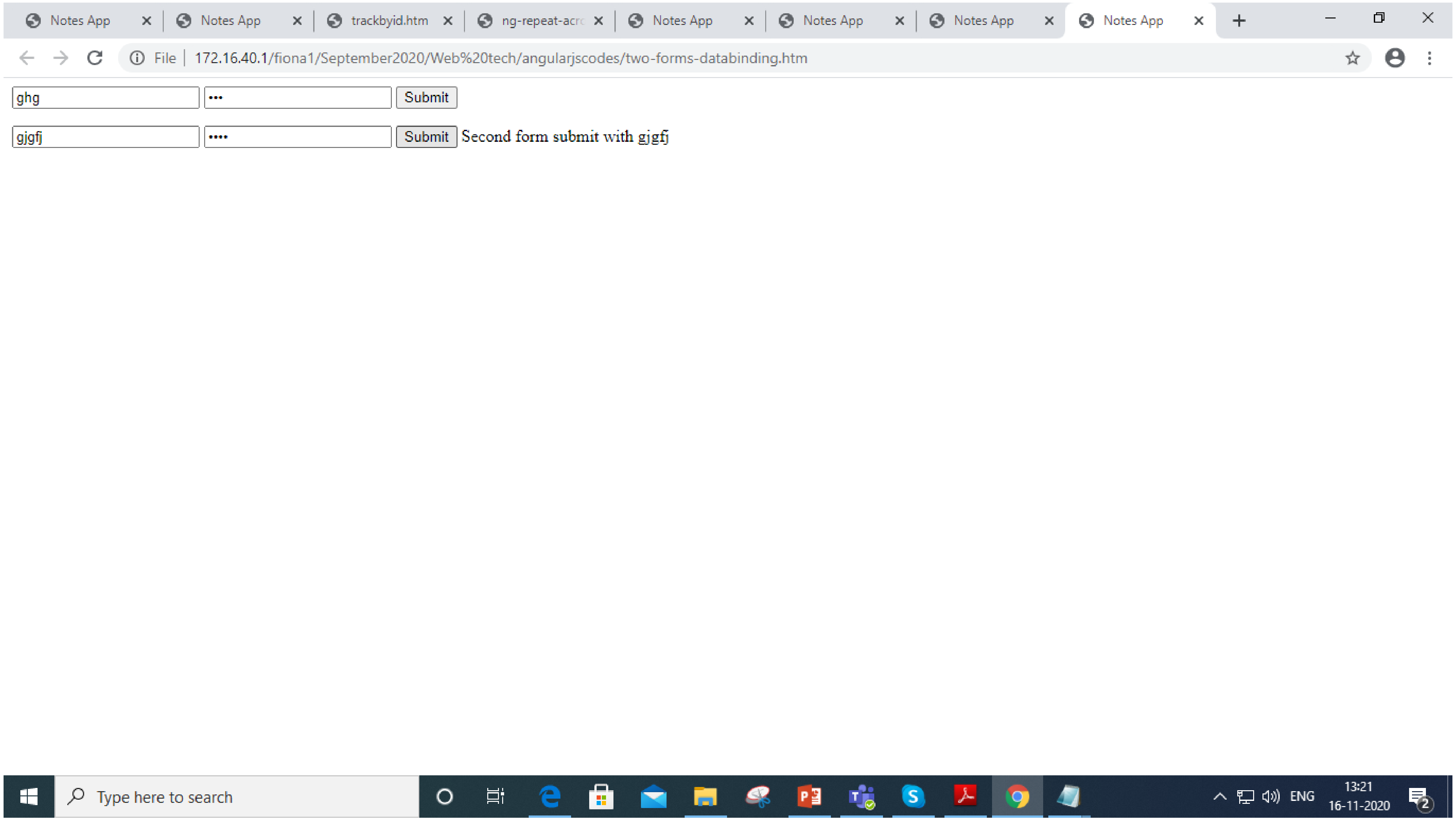


Notes App | Notes App | trackbyid.htm | ng-repeat-acr | Notes App | Notes App | Notes App | Notes App

File | 172.16.40.1/fiona1/September2020/Web%20tech/angularjscodes/two-forms-databinding.htm

ghg ••• Submit

gjgfj •••• Submit First form submit with ghg



Notes App
Notes App
trackbyid.htm
ng-repeat-acr
Notes App
Notes App
Notes App
Notes App
File
172.16.40.1/fiona1/September2020/Web%20tech/angularjscodes/two-forms-databinding.htm

ghg ••• Submit

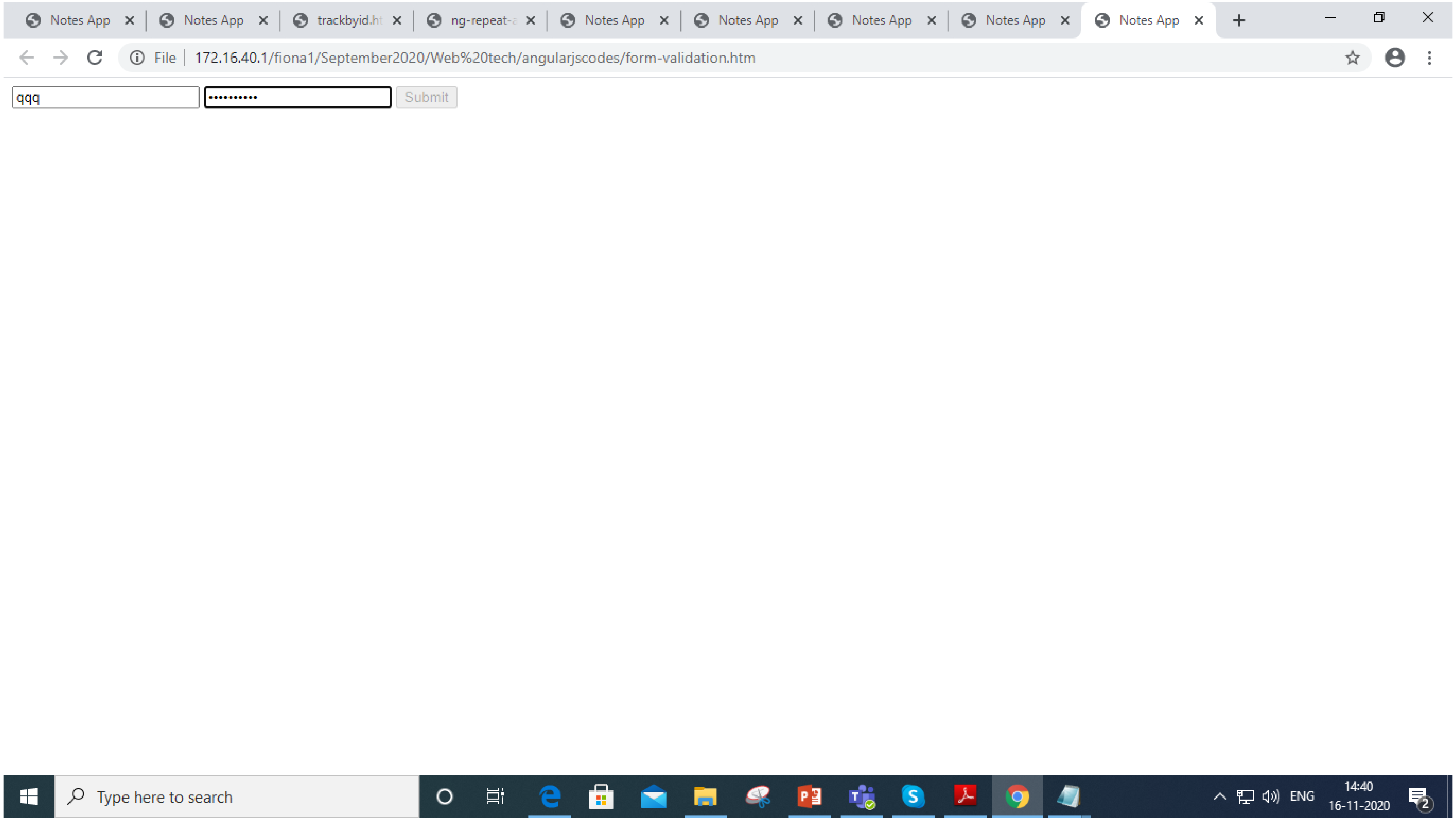
gjgfj •••• Submit Second form submit with gjgfj

Chapter 4: Forms, Inputs, and Services Leverage Data-Binding and Models

- There are two forms in this example, both with the same fields.
- The first form is bound to a username and password directly on the controller, while the second form is bound to a username and password key on a user object in the controller.
- Both trigger ng-submit function on submission. Now in first form,we have to take those fields from the controller and put them into an object, before we send it to the server.
- In the second case, we can directly take the user object from the controller and pass it around.This makes more sense, because we are directly modeling how we want to represent the form as an object in the controller.

Chapter 4: Form Validation and States

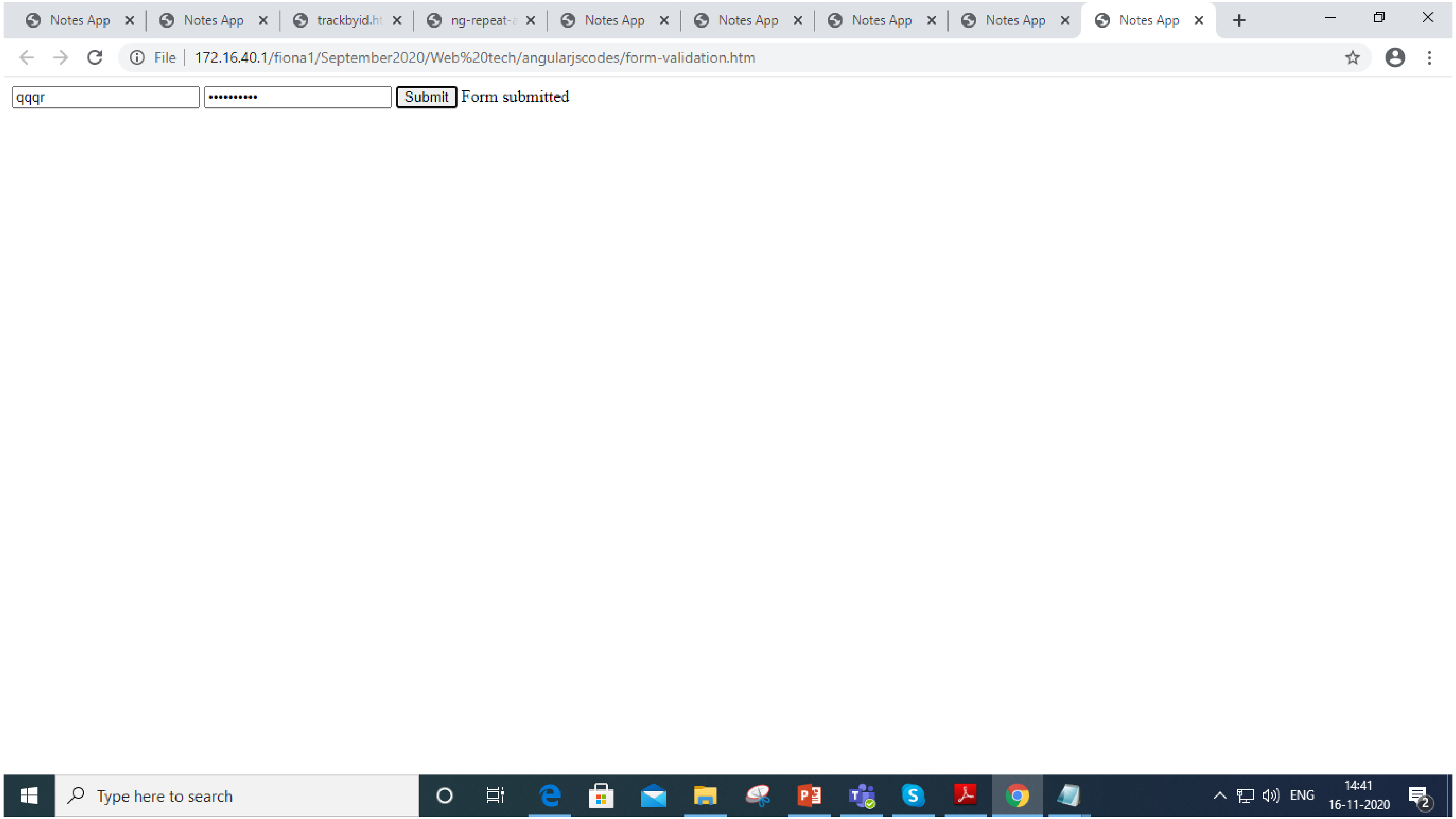
```
<html ng-app="notesApp"><head><title>Notes App</title></head>
<body ng-controller="MainCtrl as ctrl"><form ng-submit="ctrl.submit()" name="myForm">
<input type="text“ ng-model="ctrl.user.username“ required ng-minlength="4">
<input type="password"ng-model="ctrl.user.password“ required>
<input type="submit“ value="Submit“ ng-disabled="myForm.$invalid">
</form>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.2.19/angular.js">
</script>
<script type="text/javascript">
angular.module('notesApp', [])
.controller('MainCtrl', [function() {var self = this;
self.submit = function() {self.msg = 'Form submitted'; };
}]);
</script>
</body>
</html>
```



Notes App | Notes App | trackbyid.ht | ng-repeat-a | Notes App | Notes App | Notes App | Notes App | Notes App

File | 172.16.40.1/fiona1/September2020/Web%20tech/angularjscodes/form-validation.htm

qqq | •••••••••• | Submit



Notes App | Notes App | trackbyid.ht | ng-repeat-a | Notes App | Notes App | Notes App | Notes App | Notes App

File | 172.16.40.1/fiona1/September2020/Web%20tech/angularjscodes/form-validation.htm

qqqr •••••••••• Submit Form submitted

Chapter 4: Form Validation and States

- In this example we added some validation
- **Submit button disabled if the user has not filled the required fields.**

How do we accomplish this?

1. We give the form a name to refer later(myForm)

2. We use HTML5 validation tags and add required attribute on each input field.

3. We add a validator, ng-minlength(enforces minimum length of the username as 4 characters)

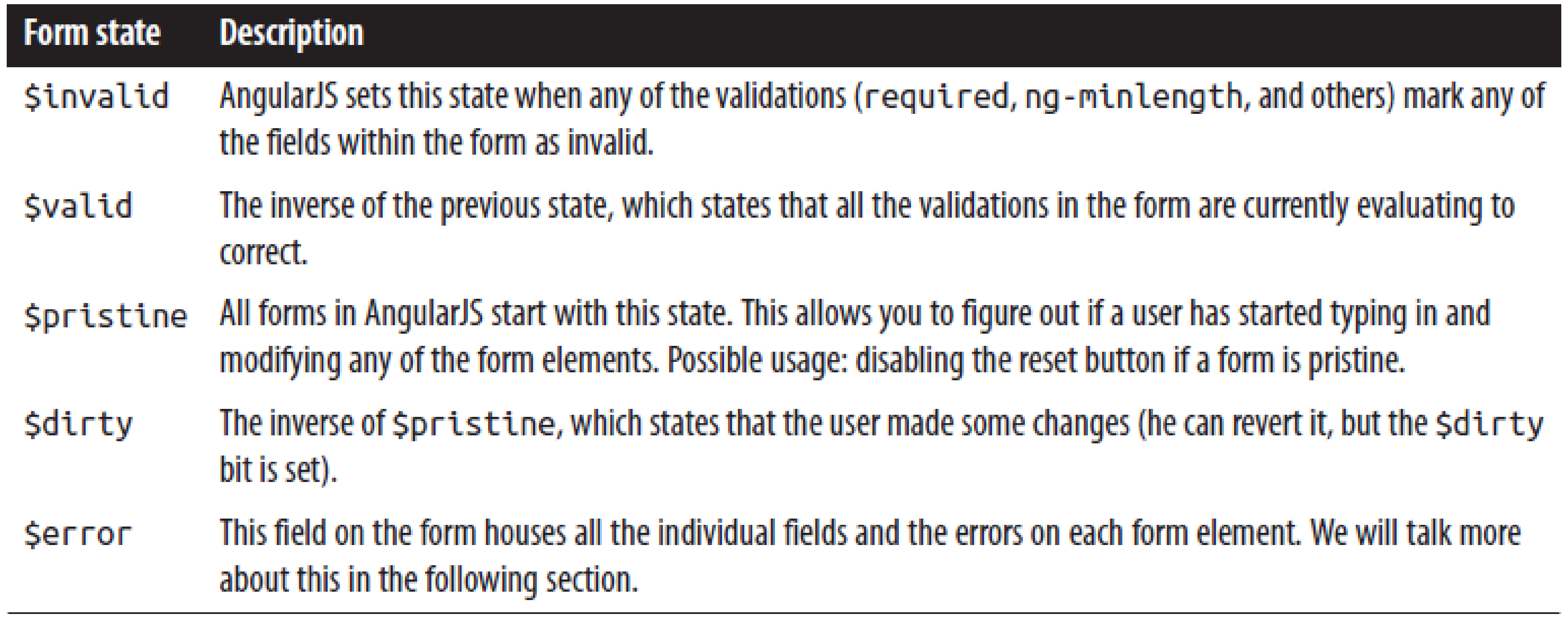
4.On the Submit button, we add an ng-disabled directive(disables the element if condition is true).button disables itself if myForm is $invalid.

Chapter 4: Form Validation and States

- AngularJS creates FormController that has the current state of the form as well as some helper methods for forms.
- You can access the FormController for a form using the form’s name. Things exposed as the state and kept updated with data-binding are shown in Table 4-1.
- Each of the states in the table (except $error) are Booleans and can be used to conditionally hide, show, disable, or enable HTML elements in the UI.

Chapter 4: Form Validation and States

Table 4-1. Form states in AngularJS



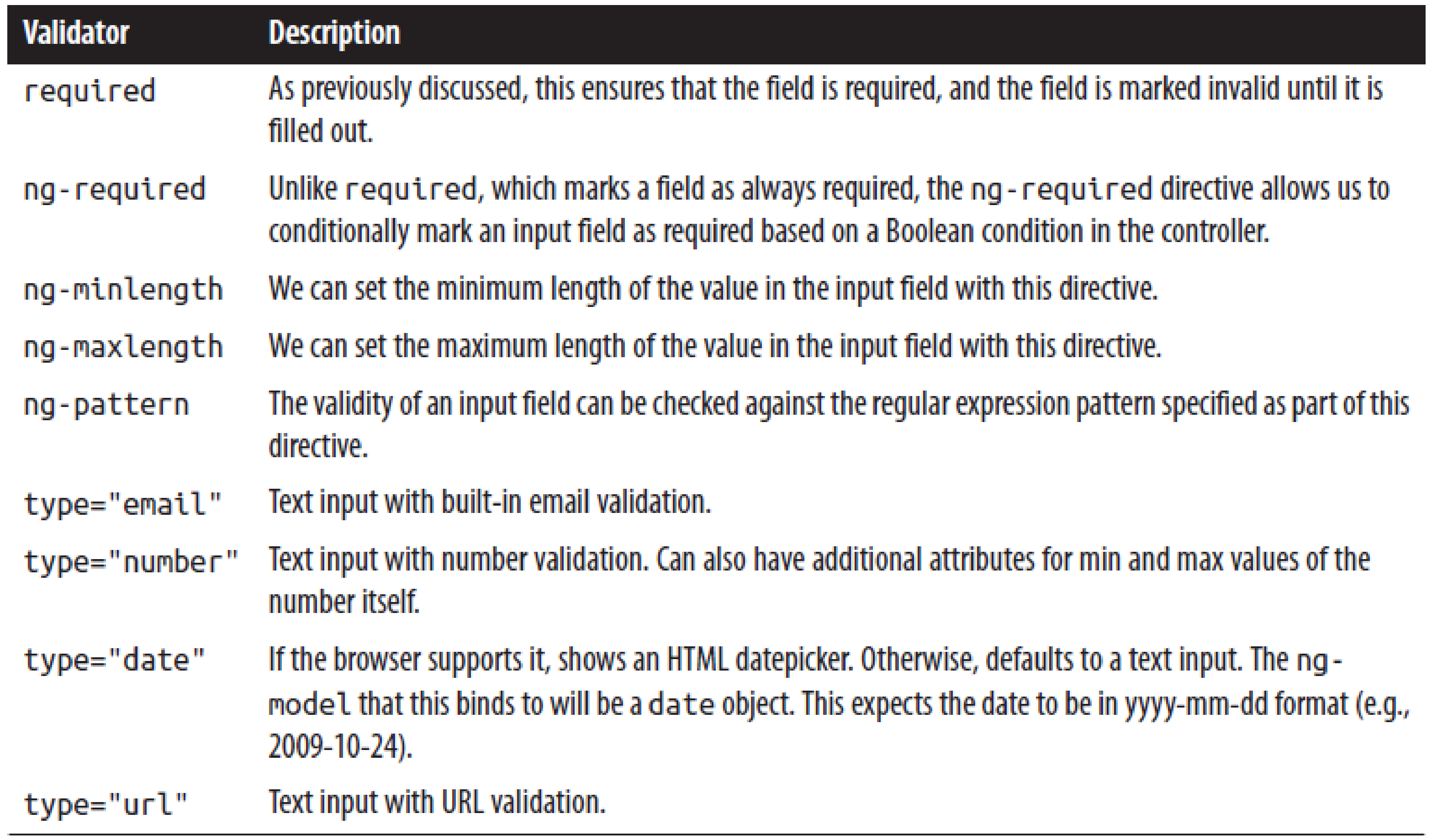
Form state	Description
`$invalid`	AngularJS sets this state when any of the validations (`required`, `ng-minlength`, and others) mark any of the fields within the form as invalid.
`$valid`	The inverse of the previous state, which states that all the validations in the form are currently evaluating to correct.
`$pristine`	All forms in AngularJS start with this state. This allows you to figure out if a user has started typing in and modifying any of the form elements. Possible usage: disabling the reset button if a form is pristine.
`$dirty`	The inverse of `$pristine`, which states that the user made some changes (he can revert it, but the `$dirty` bit is set).
`$error`	This field on the form houses all the individual fields and the errors on each form element. We will talk more about this in the following section.

Chapter 4: Error Handling with Forms

- In our previous example, we ensured that both input fields were required fields , and that the minimum length on the username was four
- Table 4-2 contains some built-in validations that AngularJS offers.
- we can write our own validators too.

Chapter 4: Error Handling with Forms

Table 4-2. Built-in AngularJS validators



Validator	Description
`required`	As previously discussed, this ensures that the field is required, and the field is marked invalid until it is filled out.
`ng-required`	Unlike `required`, which marks a field as always required, the `ng-required` directive allows us to conditionally mark an input field as required based on a Boolean condition in the controller.
`ng-minlength`	We can set the minimum length of the value in the input field with this directive.
`ng-maxlength`	We can set the maximum length of the value in the input field with this directive.
`ng-pattern`	The validity of an input field can be checked against the regular expression pattern specified as part of this directive.
`type="email"`	Text input with built-in email validation.
`type="number"`	Text input with number validation. Can also have additional attributes for min and max values of the number itself.
`type="date"`	If the browser supports it, shows an HTML datepicker. Otherwise, defaults to a text input. The `ng-model` that this binds to will be a `date` object. This expects the date to be in yyyy-mm-dd format (e.g., 2009-10-24).
`type="url"`	Text input with URL validation.

Chapter 4: Displaying Error Messages

- Why validators? check the validity of the form, and disable the Save or Update button accordingly. But we also want to tell the user what went wrong and how to fix it. AngularJS offers two things to solve this problem:
- A **model** that reflects what exactly is wrong in the form, which we can use to **display nicer error messages**
- **CSS classes** automatically added and removed from each of these fields allow us to **highlight problems** in the form

<html ng-app="notesApp"**>**

<head><title>Notes App**</title></head>**

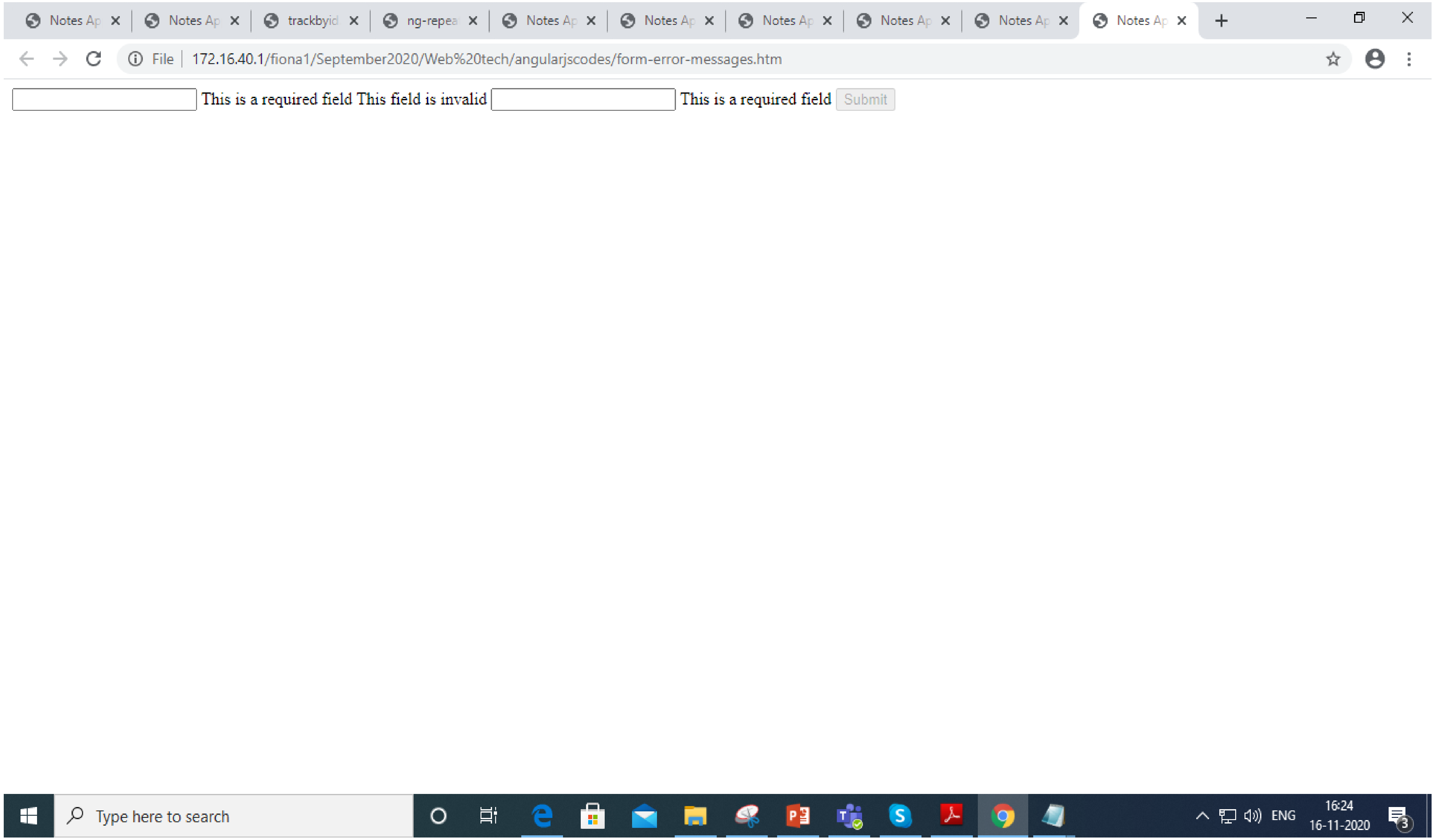
<body ng-controller="MainCtrl as ctrl"**>**

<form ng-submit="ctrl.submit()" name="myForm"**>**

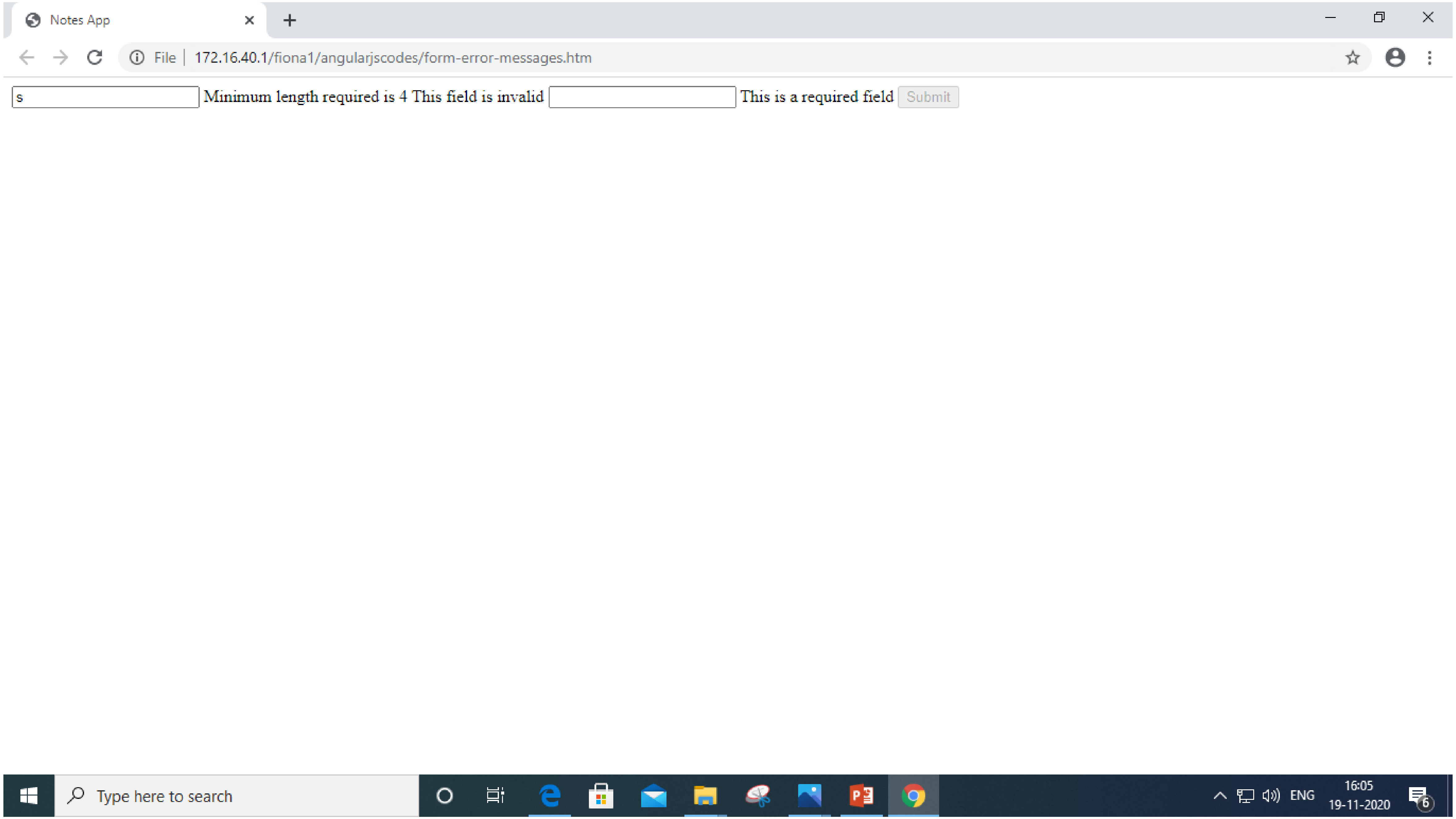
<input type="text“ name="uname“ ng-model="ctrl.user.username“ required ng-minlength="4"**>**

Chapter 4: Forms, Inputs, and Services Displaying Error Messages

```
<span ng-show="myForm.uname.$error.required">This is a required field</span>
<span ng-show="myForm.uname.$error.minlength">Minimum length required is 4</span>
<span ng-show="myForm.uname.$invalid">This field is invalid</span>
<input type="password“ name="pwd“ ng-model="ctrl.user.password“ required>
<span ng-show="myForm.pwd.$error.required">This is a required field</span>
<input type="submit“ value="Submit“ ng-disabled="myForm.$invalid">
</form><script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.2.19/angular.js">
</script>
<script type="text/javascript">
angular.module('notesApp', [])
.controller('MainCtrl', [function () {var self = this;
self.submit = function () {console.log('User clicked submit with ', self.user);
};
}]);
</script>
</body>
</html>
```


Notes Ap
trackbyid
ng-repea
172.16.40.1/fiona1/September2020/Web%20tech/angularjscodes/form-error-messages.htm
Type here to search
ENG
16:24
16-11-2020

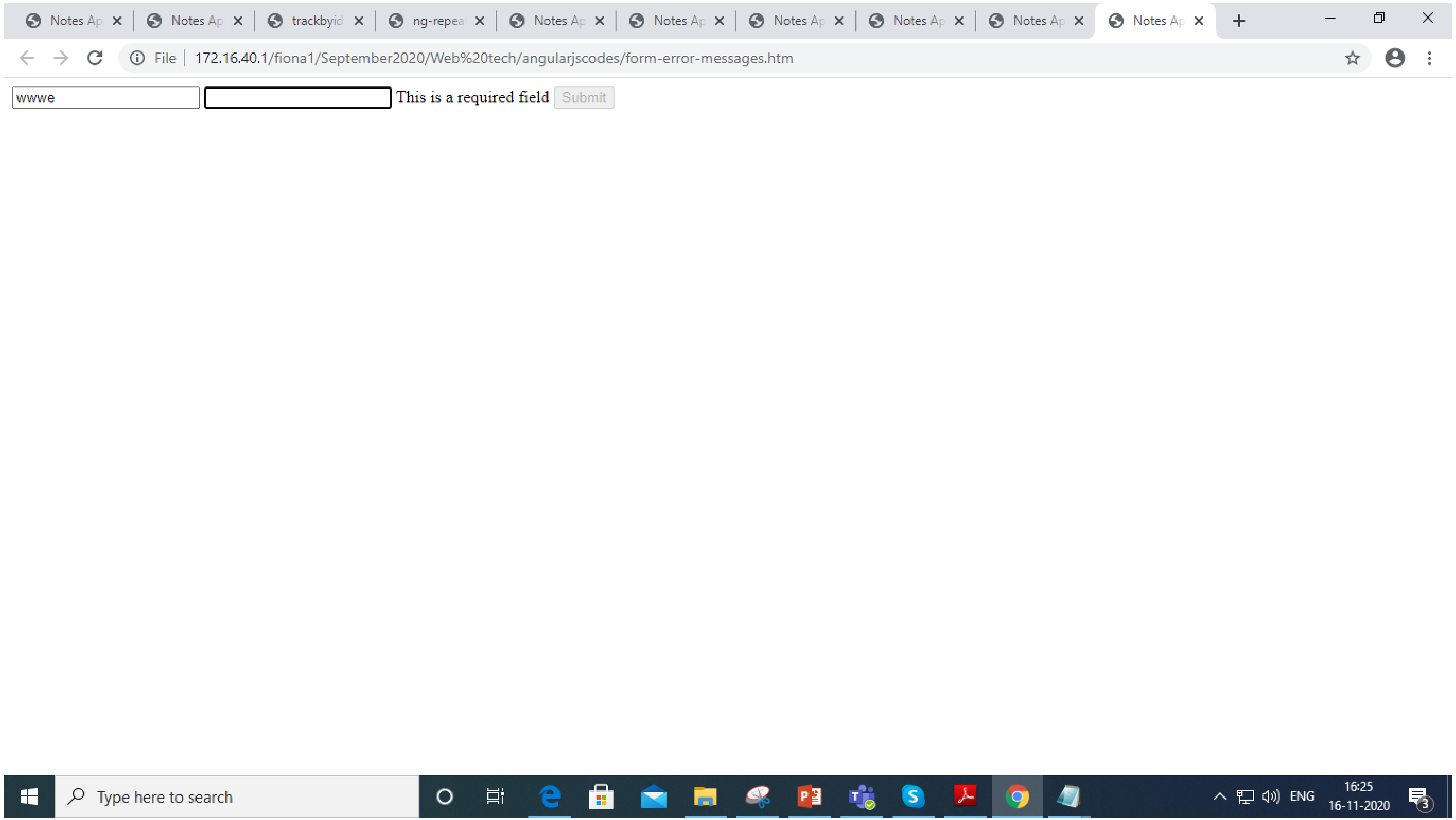
This is a required field This is a required field This field is invalid This is a required field Submit



Notes App

172.16.40.1/fiona1/angularjscodes/form-error-messages.htm

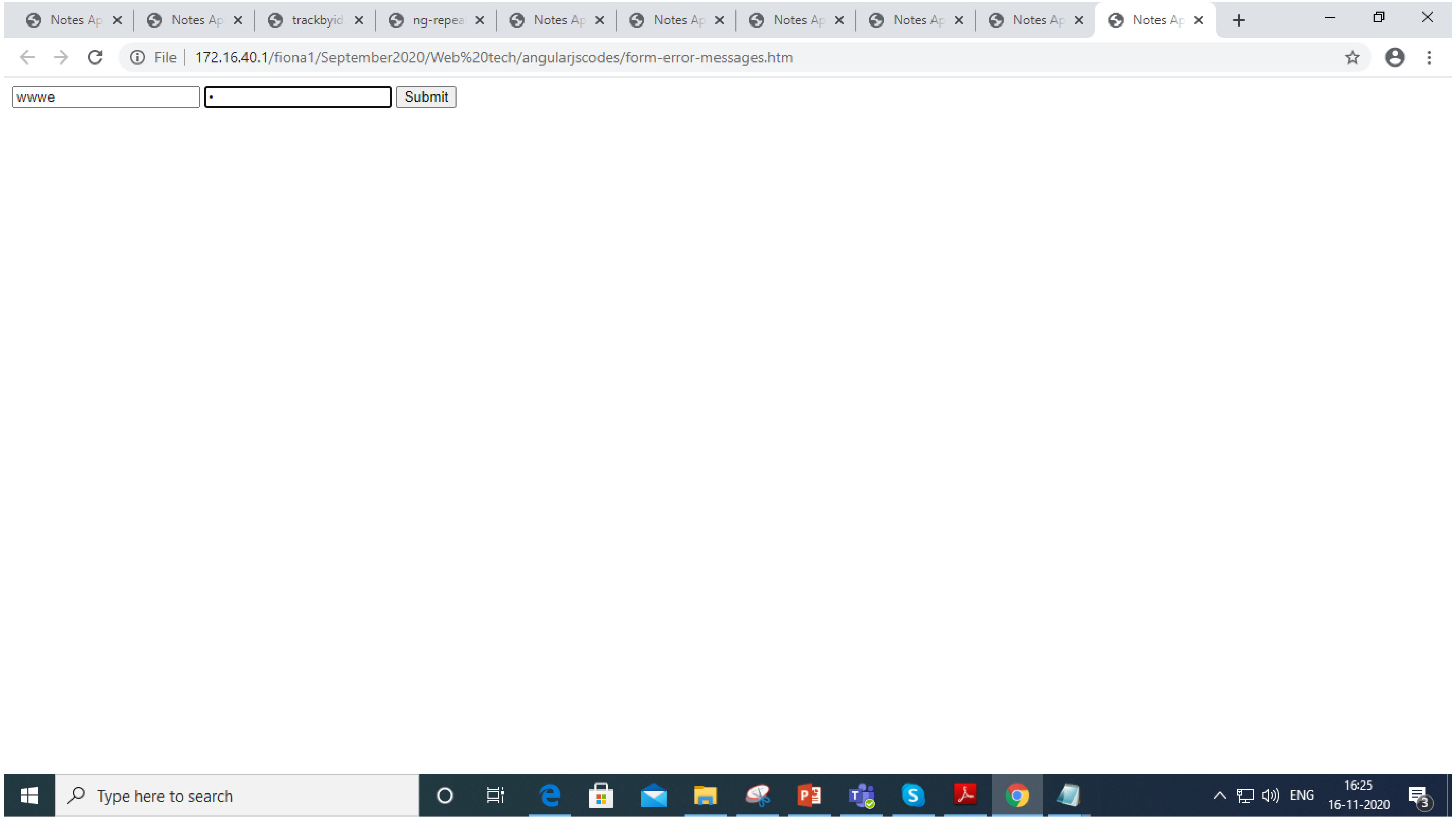
s Minimum length required is 4 This field is invalid This is a required field Submit



Notes Ap
trackbyid
ng-repea
File
172.16.40.1/fiona1/September2020/Web%20tech/angularjscodes/form-error-messages.htm

wwwe This is a required field Submit

Type here to search
ENG
16:25
16-11-2020


Notes Ap
trackbyid
ng-repea
File
172.16.40.1/fiona1/September2020/Web%20tech/angularjscodes/form-error-messages.htm
wwwe
•
Submit
Type here to search
ENG
16:25
16-11-2020

Chapter 4: Forms, Inputs, and Services Displaying Error Messages

1. we added the name attribute to both the input fields that need validation: uname and pwd.

2. Then we leverage AngularJS's form bindings to be able to pick out the errors for each individual field. When we add a name to any input, it creates a model on the form for that particular input, with the error state.

3. So for the username field, we can access it if the field was not entered by accessing myForm.uname.$error.required. Similarly, for ng-minlength, the field would be myForm.uname.$error.minlength. For the password, we look at myForm.pwd.$error.required to see if the field was filled or not.

4. We also accessed the state of the input, similar to the form, by accessing myForm.uname.$invalid($valid, $pristine, $dirty) .With this, we have an error message that shows only when a certain type of error is triggered.

- Each of the validators we saw in Table 4-2 exposes a key on the $error object, so that we can pick it up and display the error message for that particular error to the user.

Styling Forms and States

- How to highlight certain input fields or form states using UI and CSS?
- One option is to use the form and input states along with ng-class directive.Eg. add a class dirty when myForm.$dirty is true.
- Second option is For each of the states described previously, AngularJS adds and removes the CSS classes shown in Table 4-3 to and from the forms and input elements. Similarly, for each of the validators that we add on the input fields, we also get a CSS class in a similarly named fashion, as demonstrated in Table 4-4.
- Other than the basic input states, AngularJS takes the name of the validator (number,maxlength, pattern, etc.) and depending on whether or not that particular validator has been satisfied, adds the ng-valid-validator_name or ng-invalid-validator_name class, respectively.
- Example of how its used to highlight the input in different ways is shown.

Styling Forms and States

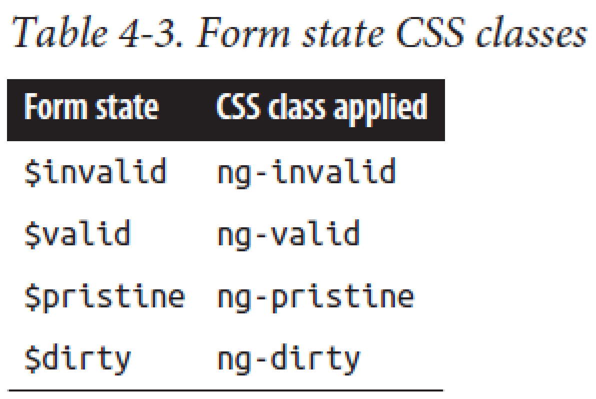


Table 4-3. Form state CSS classes

Form state	CSS class applied
`$invalid`	`ng-invalid`
`$valid`	`ng-valid`
`$pristine`	`ng-pristine`
`$dirty`	`ng-dirty`

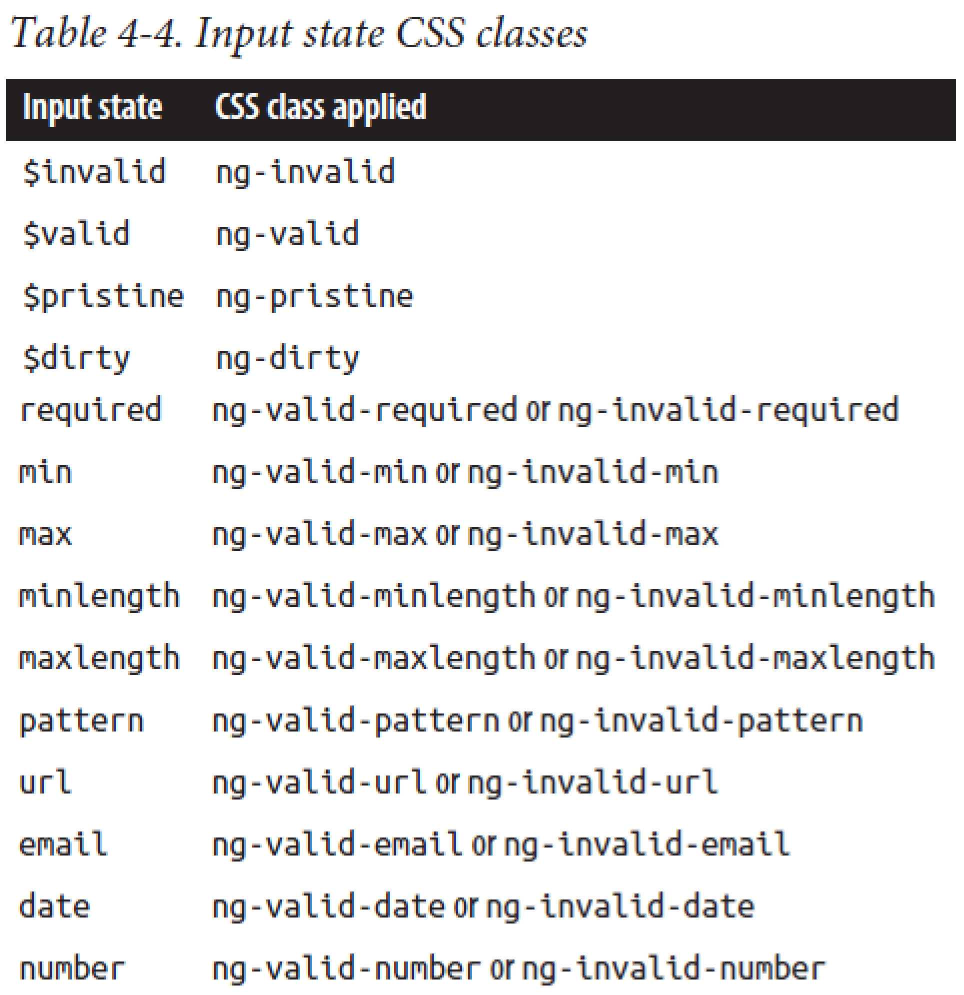
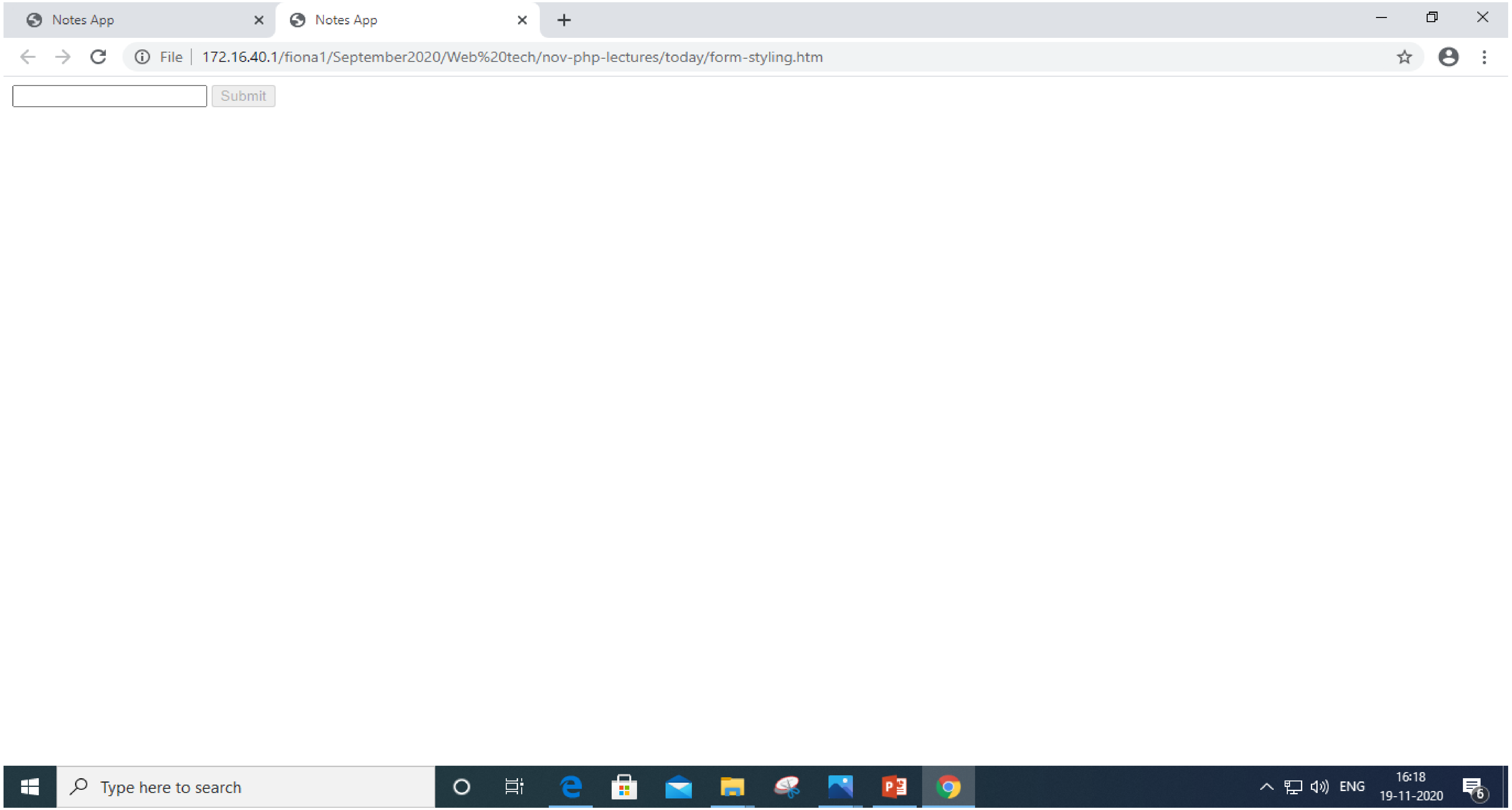


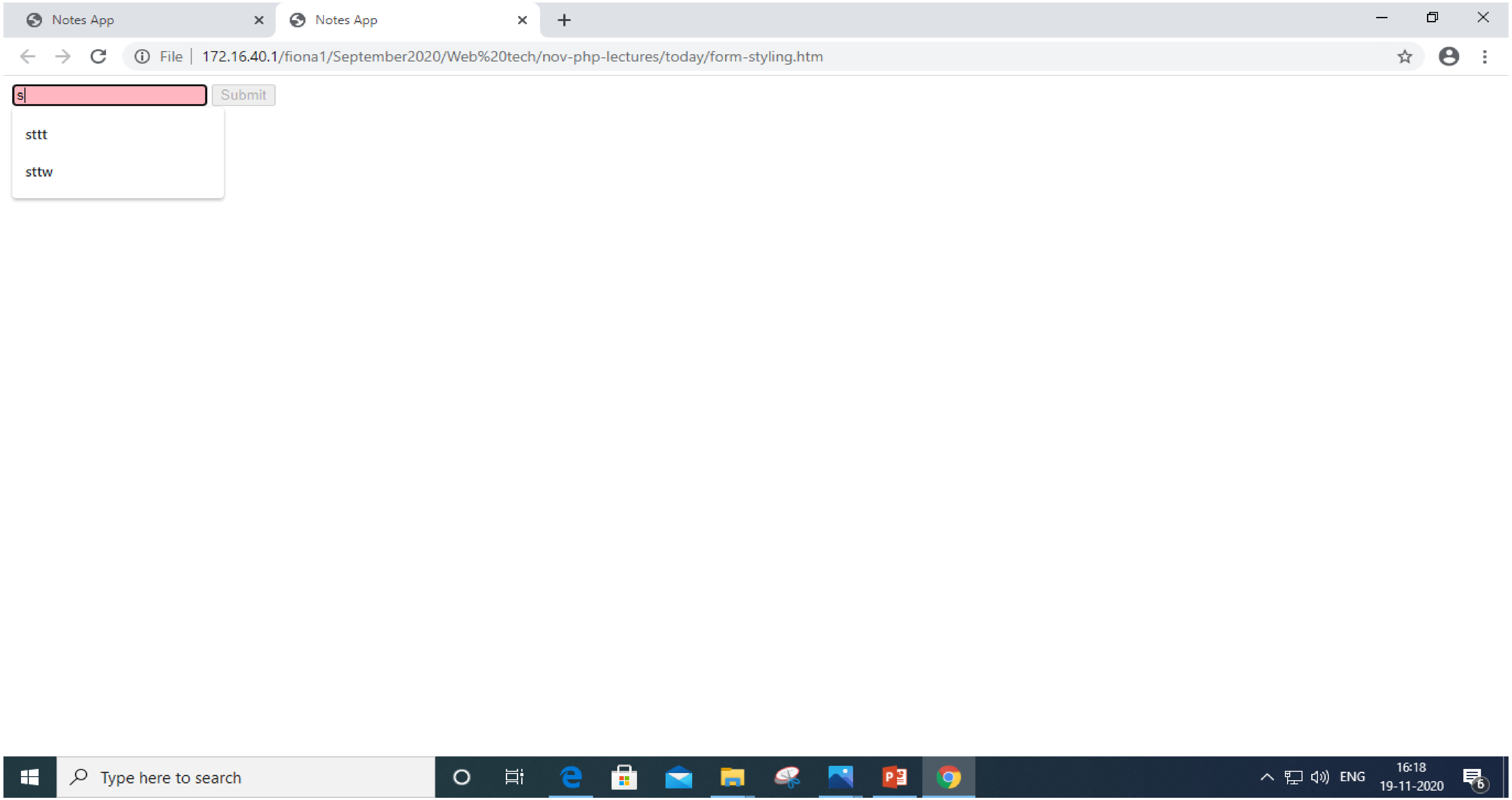
Table 4-4. Input state CSS classes

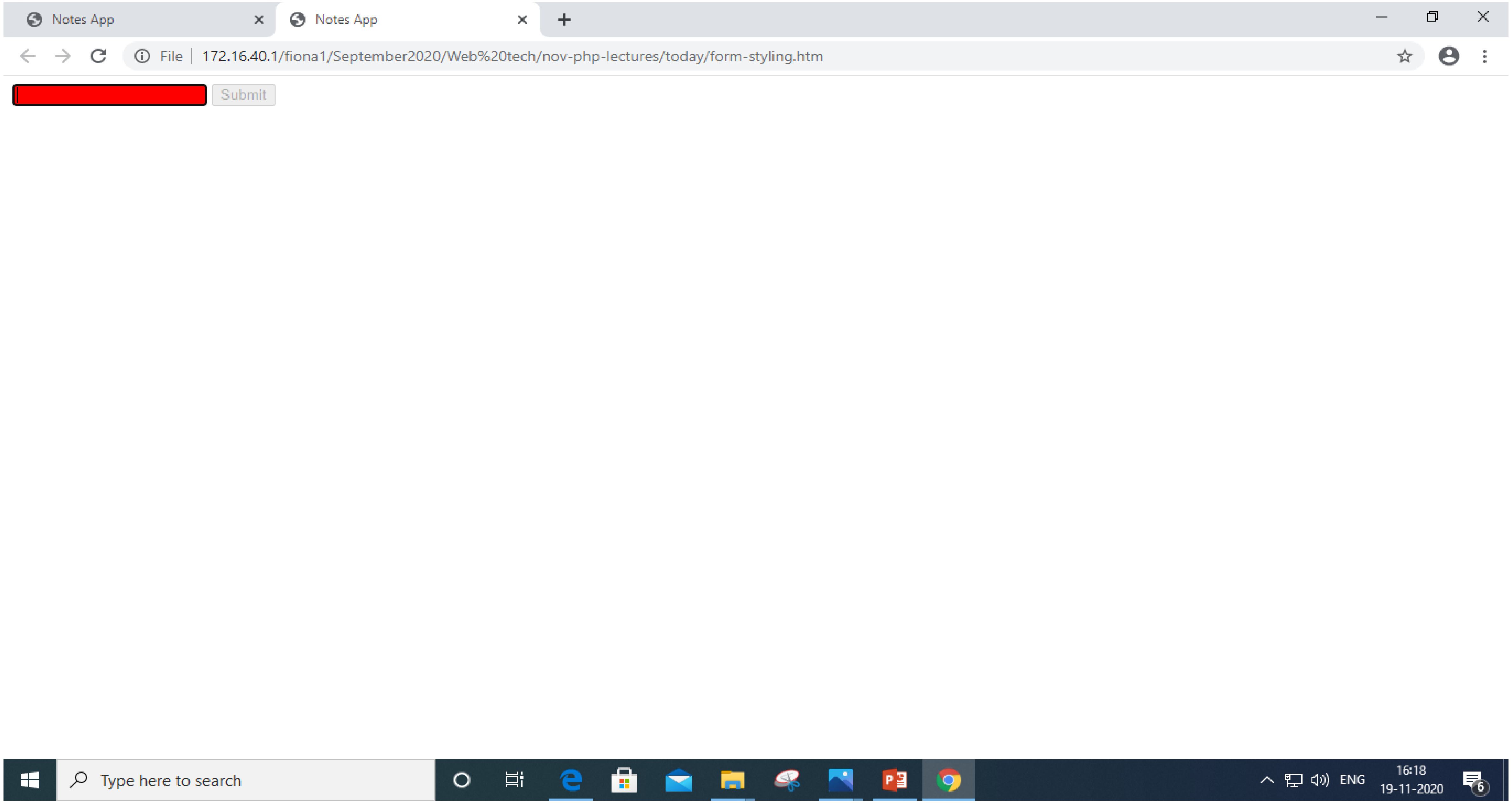
Input state	CSS class applied
`$invalid`	`ng-invalid`
`$valid`	`ng-valid`
`$pristine`	`ng-pristine`
`$dirty`	`ng-dirty`
`required`	`ng-valid-required` or `ng-invalid-required`
`min`	`ng-valid-min` or `ng-invalid-min`
`max`	`ng-valid-max` or `ng-invalid-max`
`minlength`	`ng-valid-minlength` or `ng-invalid-minlength`
`maxlength`	`ng-valid-maxlength` or `ng-invalid-maxlength`
`pattern`	`ng-valid-pattern` or `ng-invalid-pattern`
`url`	`ng-valid-url` or `ng-invalid-url`
`email`	`ng-valid-email` or `ng-invalid-email`
`date`	`ng-valid-date` or `ng-invalid-date`
`number`	`ng-valid-number` or `ng-invalid-number`

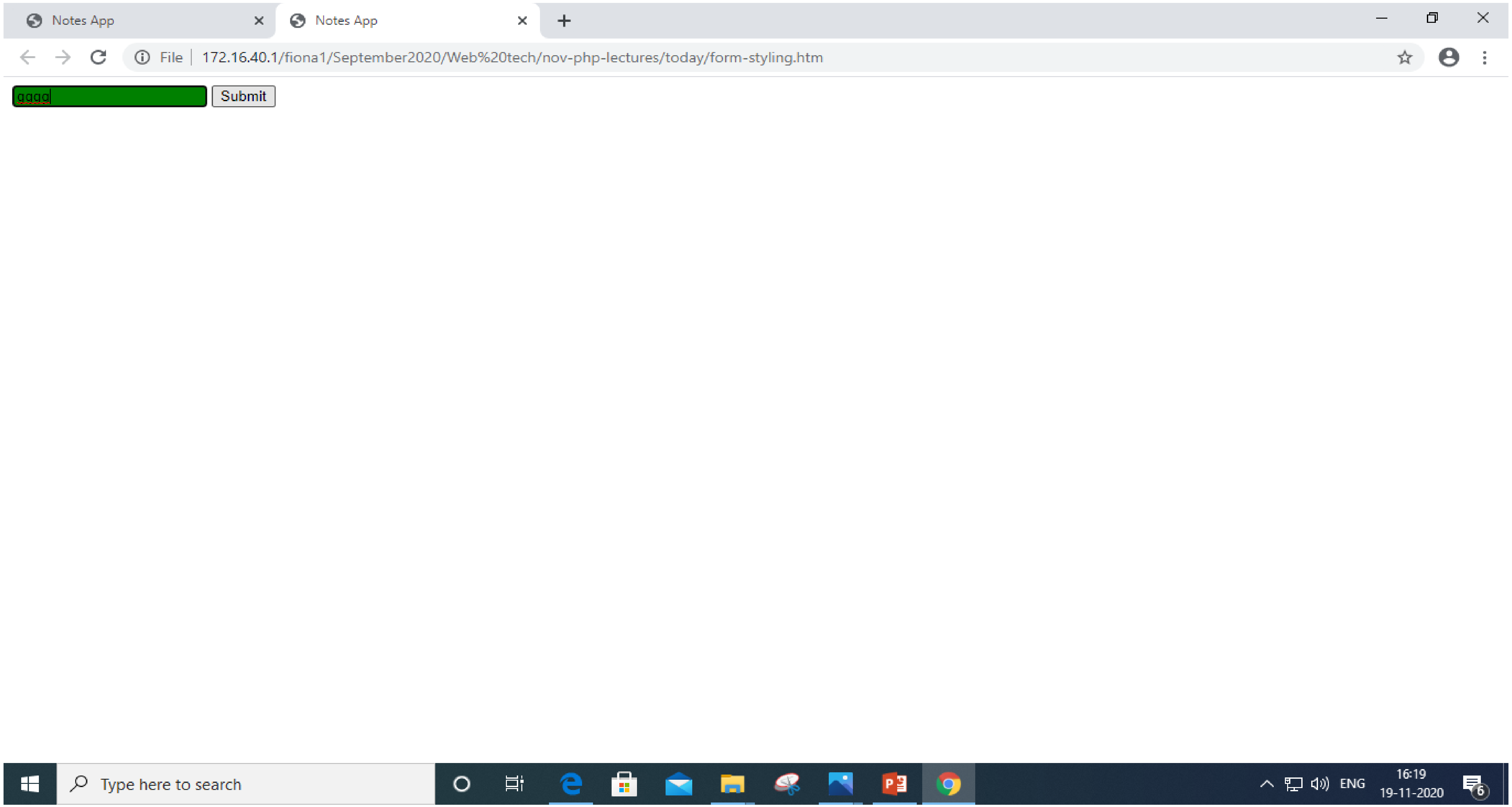
Styling Forms and States

```
<html ng-app="notesApp"><head><title>Notes App</title><style>
.username.ng-valid {background-color: green;}
.username.ng-dirty.ng-invalid-required {background-color: red;}
.username.ng-dirty.ng-invalid-minlength {background-color: lightpink;}
</style></head>
<body ng-controller="MainCtrl as ctrl">
<form ng-submit="ctrl.submit()" name="myForm">
<input type="text“ class="username“ name="uname“ ng-model= "ctrl.user.username“ required ng-
minlength="4">
<input type="submit“ value="Submit“ ng-disabled="myForm.$invalid">
</form><script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.2.19/angular.js"></script>
<script type="text/javascript">angular.module('notesApp', [])
.controller('MainCtrl', [function() {var self = this;self.submit = function() {console.log('User clicked
submit with ', self.user);};}]);
</script>
</body></html>
```


Notes App
Notes App
File
172.16.40.1/fiona1/September2020/Web%20tech/nov-php-lectures/today/form-styling.htm
Submit
Type here to search
ENG
16:18
19-11-2020


Notes App
Notes App
File
172.16.40.1/fiona1/September2020/Web%20tech/nov-php-lectures/today/form-styling.htm
s
Submit
sttt
sttw
Type here to search
ENG
16:18
19-11-2020


Notes App
Notes App
File
172.16.40.1/fiona1/September2020/Web%20tech/nov-php-lectures/today/form-styling.htm
Submit
Type here to search
ENG
16:18
19-11-2020


Notes App
Notes App
File
172.16.40.1/fiona1/September2020/Web%20tech/nov-php-lectures/today/form-styling.htm
gggg
Submit
Type here to search
ENG
16:19
19-11-2020

Styling Forms and States

- This example accomplishes:

1.When the field is correctly filled out, it turns the input box green. (done by setting the background color when the CSS class ng-valid is applied to our input field)

2.display the background as dark red if the user starts typing in, and then undoes it.(set the background color to red if the CSS classes ng-dirty (which marks that the user has modified it) and ng-invalid-minlength (which marks that the user has not typed in the necessary amount of characters) applied)

Nested Forms with ng-form

- how to deal with more complicated grouping of elements. For eg. subsections of form to be valid as a group.
- Nesting not possible with HTML form as they are not meant to be nested.
- AngularJS **ng-form** directive similar to form but allows nesting, so group related form fields under sections.

<html ng-app**><head><title>**Notes App**</title>**

</head><body>

<form novalidate name="myForm"**>**

<input type="text“ class="username“ name="uname“ ng-model= "ctrl.user.username“ required="“ placeholder="Username“ ng-minlength="4" **/>**

Nested Forms with ng-form

<input type="password“ class="password“ name="pwd“ ng-model="ctrl.user.password“ placeholder="Password“ required="" **/>**

<ng-form name="profile"**>**

<input type="text“ name="firstName“ ng-model= "ctrl.user.profile.firstName“ placeholder="First Name“ required**>**

<input type="text“ name="middleName“ placeholder="Middle Name"

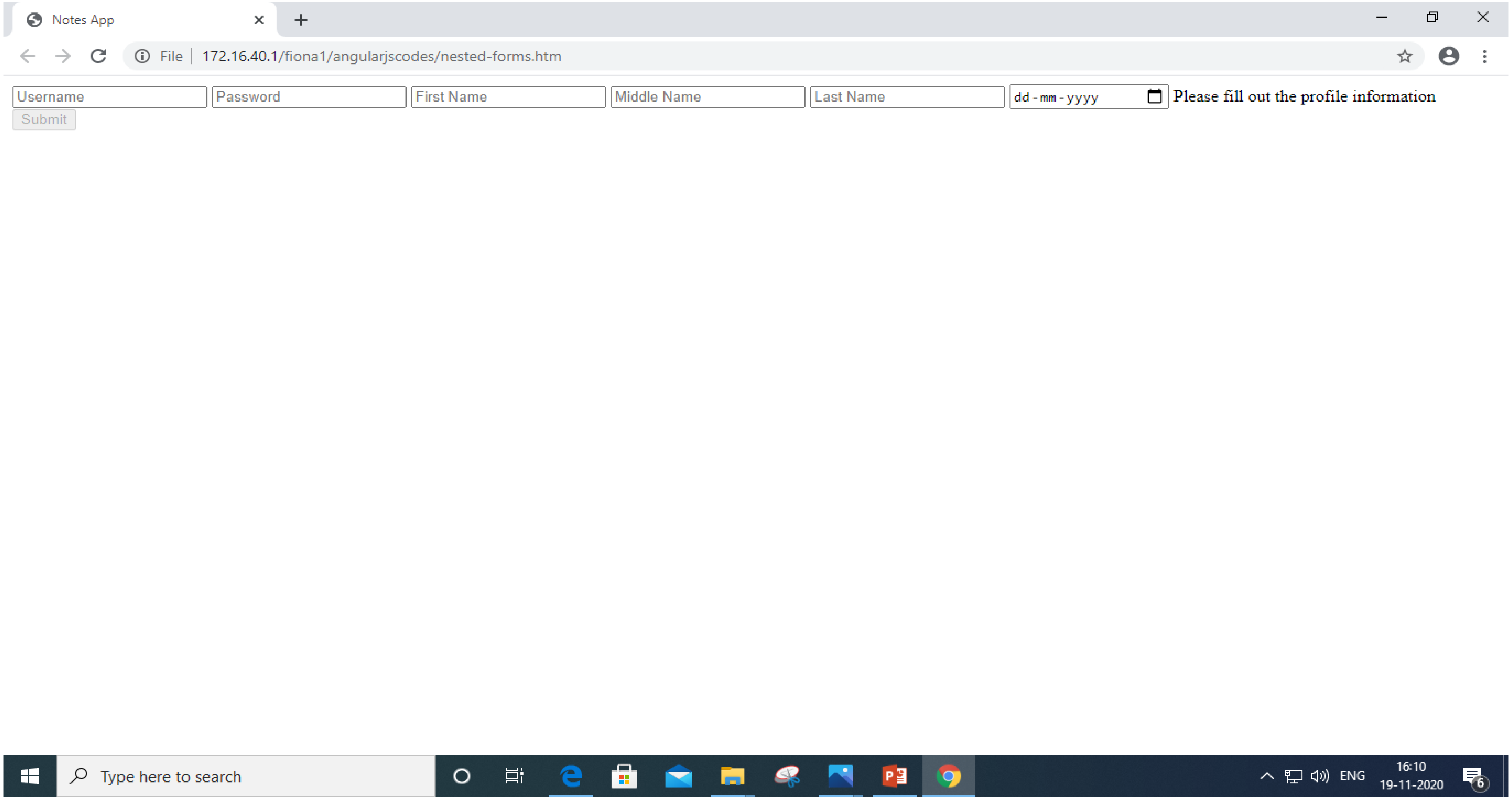
ng-model="ctrl.user.profile.middleName"**>**

<input type="text“ name="lastName"placeholder="Last Name"

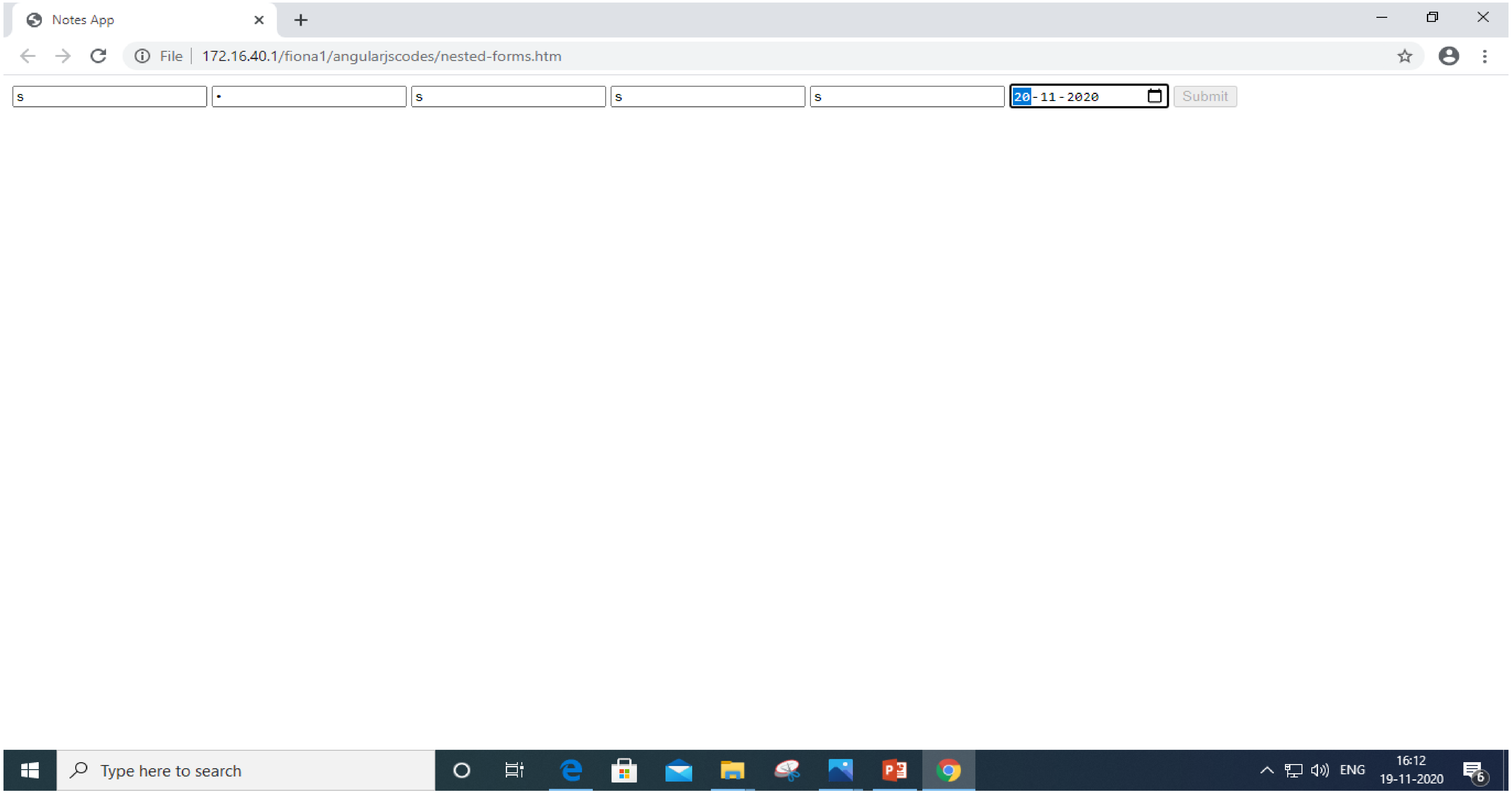
ng-model="ctrl.user.profile.lastName“ required**>**

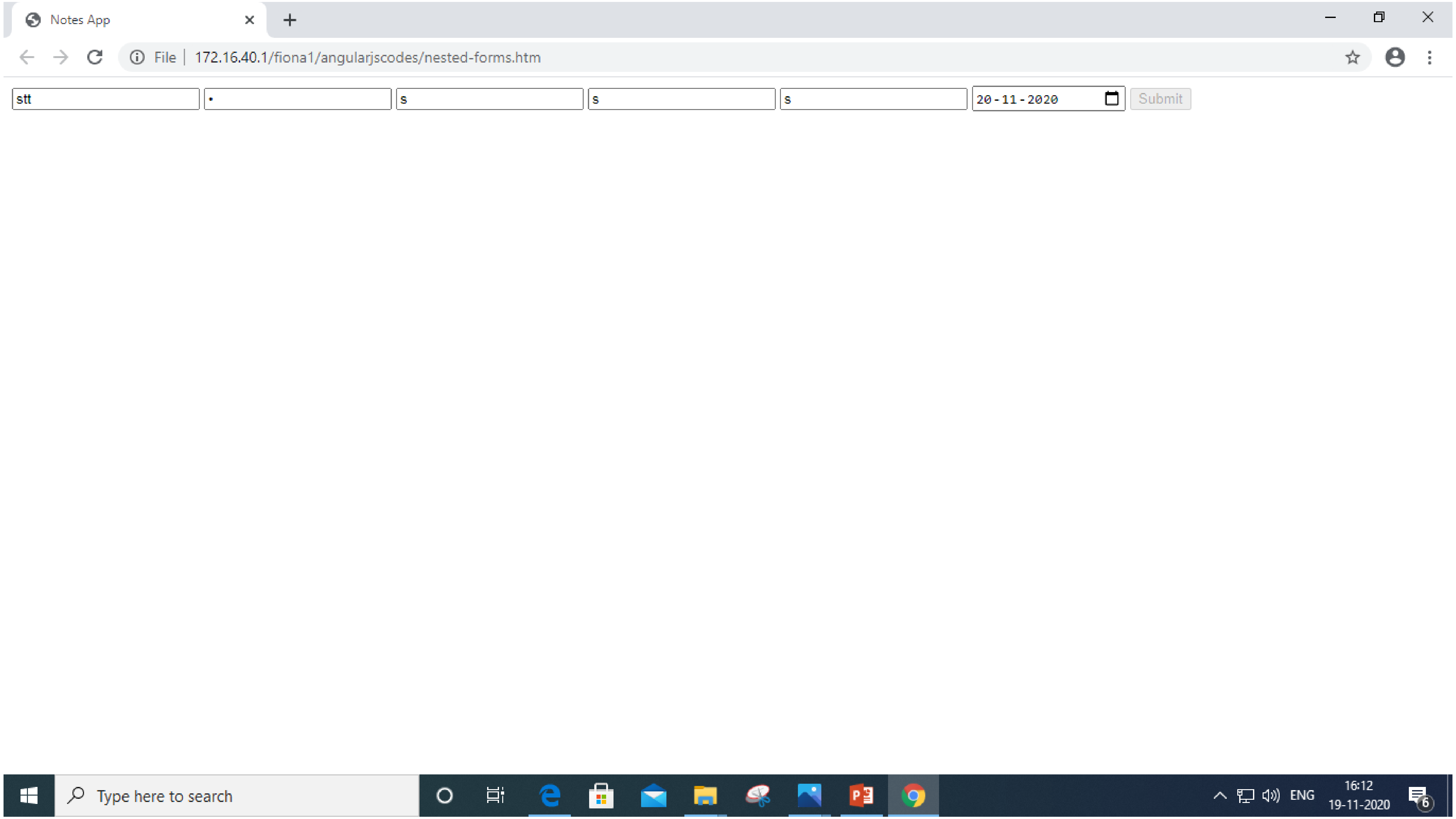
Nested Forms with ng-form

```
<input type="date“ name="dob“ placeholder="Date Of Birth“ ng-model=
"ctrl.user.profile.dob">
</ng-form>
<span ng-show="myForm.profile.$invalid">
Please fill out the profile information
</span><input type="submit“ value="Submit“ ng-disabled="myForm.$invalid"/>
</form>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.2.19/angular.js">
</script>
</body>
</html>
```

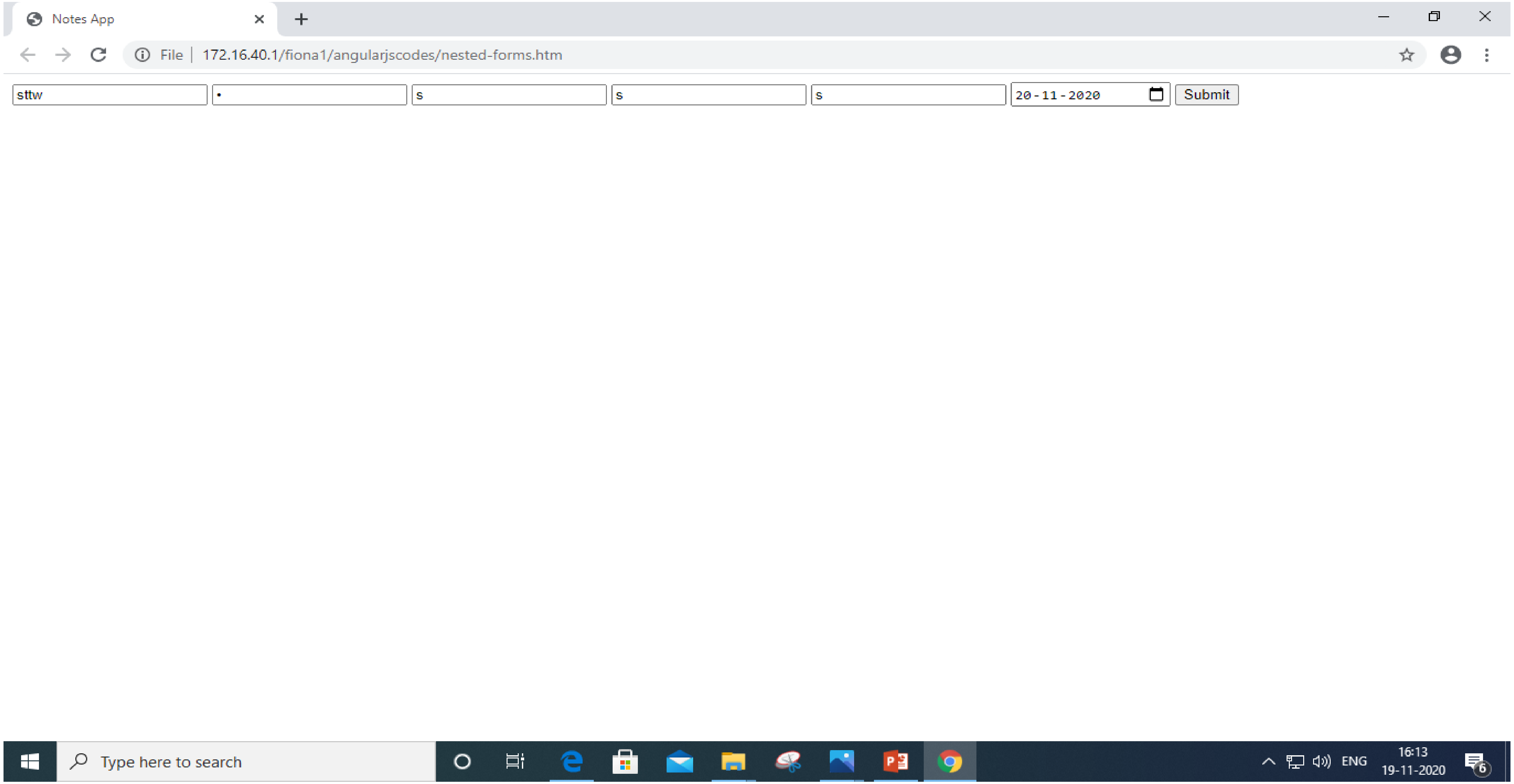


Please fill out the profile information


Notes App
File
172.16.40.1/fiona1/angularjscodes/nested-forms.htm
s
•
s
s
s
20-11-2020
Submit
Type here to search
ENG
16:12
19-11-2020



stt	•	s	s	s	20-11-2020	Submit



sttw	•	s	s	s	20-11-2020	Submit

Nested Forms with ng-form

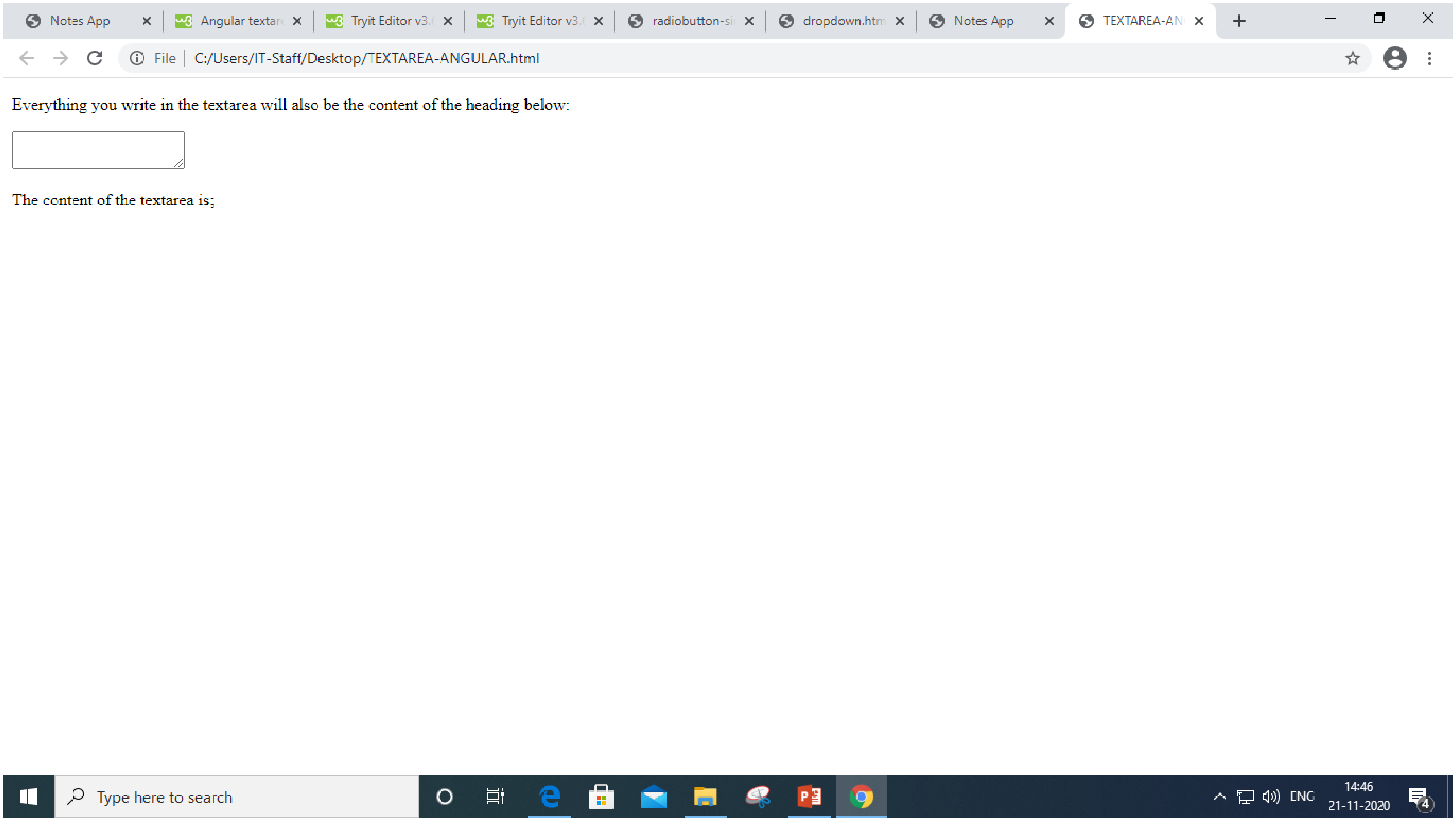
- In the example subform nested in main form so **ng-form** directive provides **substate** within our form, to evaluate section validity.
- Give subform a name to identify and get its state.
- The state can be accessed directly (**profile.$invalid**) or through parent form (**myForm.profile.$invalid**).
- Individual elements of the form can be accessed as: (**profile**.firstName.$error.required).
- Subforms and nested forms affect the outer form (the myForm.$invalid is true because of the use of the required tags).

Other Form Controls

1.Textareas:

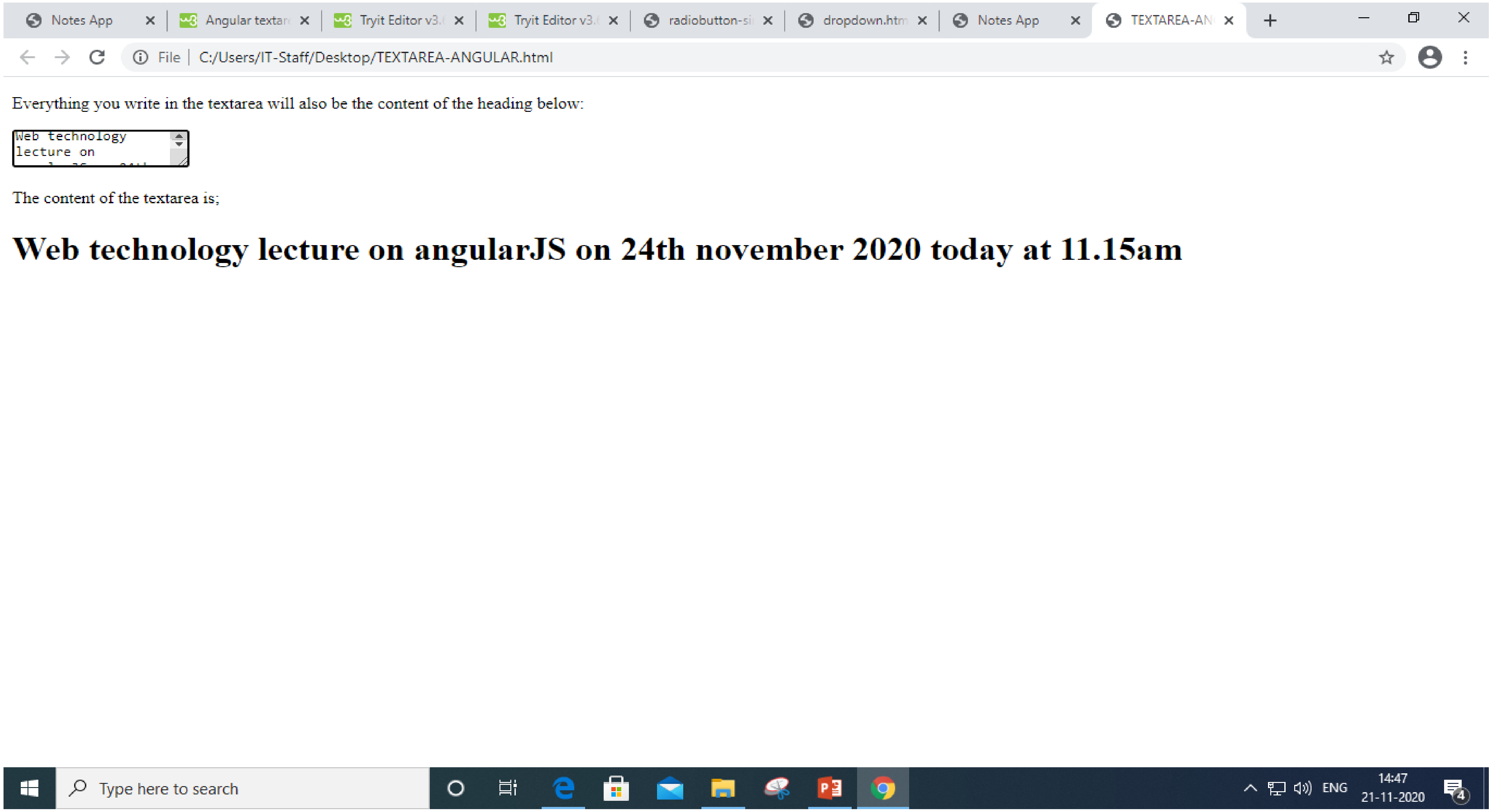
```
<textarea ng-model="ctrl.user.address" required></textarea>
```

- two-way data-binding with a textarea, and make it a required field



Everything you write in the textarea will also be the content of the heading below:

The content of the textarea is;



Everything you write in the textarea will also be the content of the heading below:

```
Web technology
lecture on
```

The content of the textarea is;

Web technology lecture on angularJS on 24th november 2020 today at 11.15am

Other Form Controls

2.Checkboxes: can only have true or false values .

- An ng-model two-way data-binding to the checkbox takes a Boolean value and assigns the checked state based on it. After that, any changes to the checkbox toggles the state of the model:

<input type="checkbox" ng-model="ctrl.user.agree">

- What if we wanted to assign the string YES or NO to our model, or have the checkbox checked when the value is YES?
- AngularJS gives two attribute arguments to the checkbox that allow us to specify our custom values for the true and false values. We could accomplish this as :

<input type="checkbox“ ng-model="ctrl.user.agree“ ng-true-value="YES“ ng-false-value = “NO">

- This sets the **value** of the agree field to **YES** if the user **checks** the checkbox, and NO if

the user **unchecks** it.

Other Form Controls

- what if we want one-way data-binding: where the state of the checkbox changes when the value behind it changes, but the value doesn’t change on checking or unchecking the checkbox.
- Accomplished using the **ng-checked directive**, which binds to an expression.
- Whenever the **value** is **true**, AngularJS will **set** the **checked** property for the input element, and remove and **unset** it when the **value** is **false**. Let’s demonstrate :

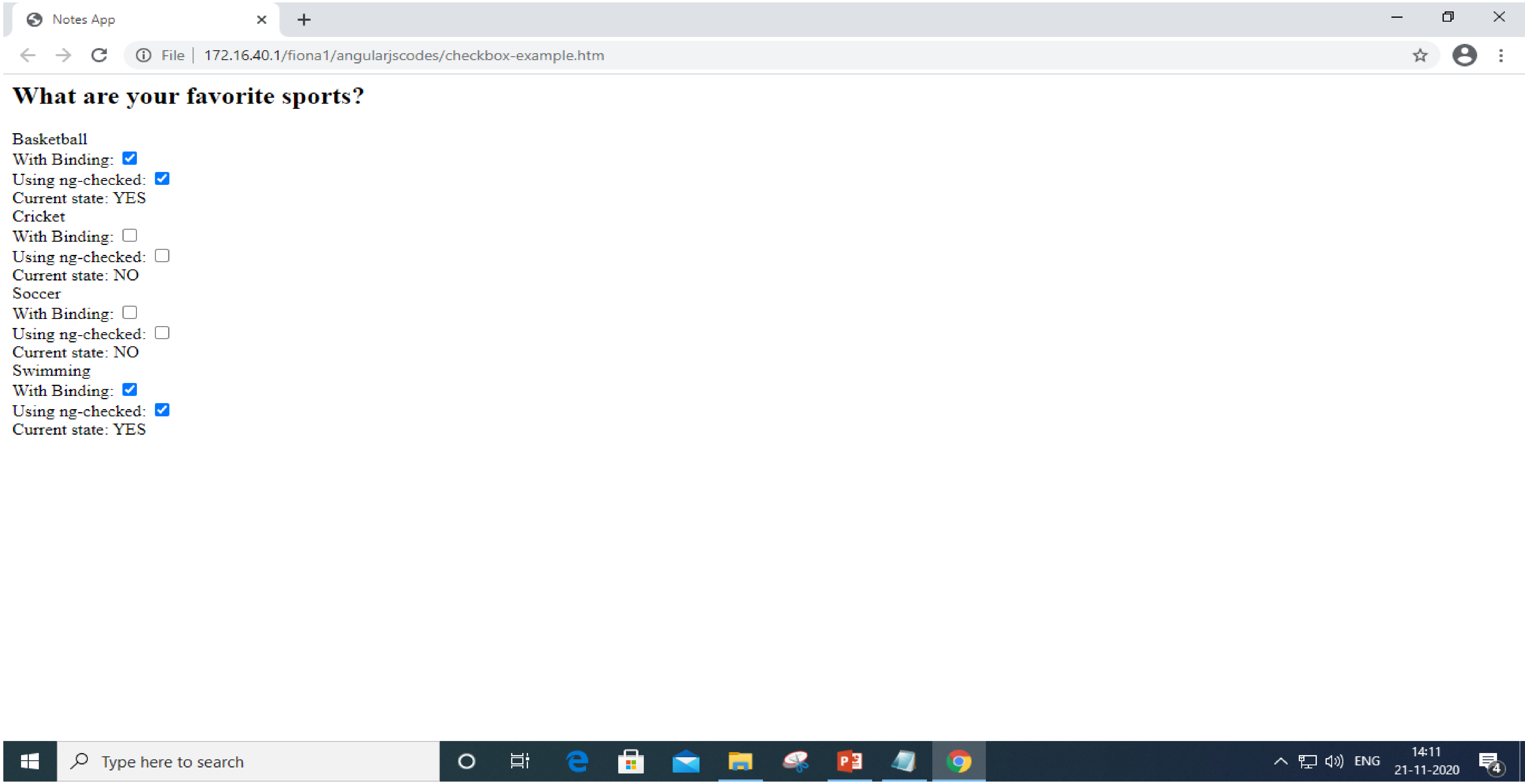
Other Form Controls

```
<html ng-app="notesApp">
<head><title>Notes App</title></head>
<body ng-controller="MainCtrl as ctrl">
<div><h2>What are your favorite sports?</h2>
<div ng-repeat="sport in ctrl.sports">
<label ng-bind="sport.label"></label>
<div>With Binding:
<input type="checkbox“ ng-model="sport.selected“ ng-true-value="YES"
ng-false-value="NO">
</div>
<div>
Using ng-checked:
<input type="checkbox“ ng-checked="sport.selected === 'YES'">
```

Other Form Controls

```
</div>
<div>
Current state: {{sport.selected}}
</div>
</div>
</div>
<script  src="https://ajax.googleapis.com/ajax/libs/angularjs/1.2.19/angular.js">
</script>
<script type="text/javascript">
angular.module('notesApp', [])
.controller('MainCtrl', [function() {var self = this;
self.sports = [{label: 'Basketball', selected: 'YES'},
{label: 'Cricket', selected: 'NO'},
{label: 'Soccer', selected: 'NO'},
{label: 'Swimming', selected: 'YES'}];
}]);
</script>
</body>
</html>
```

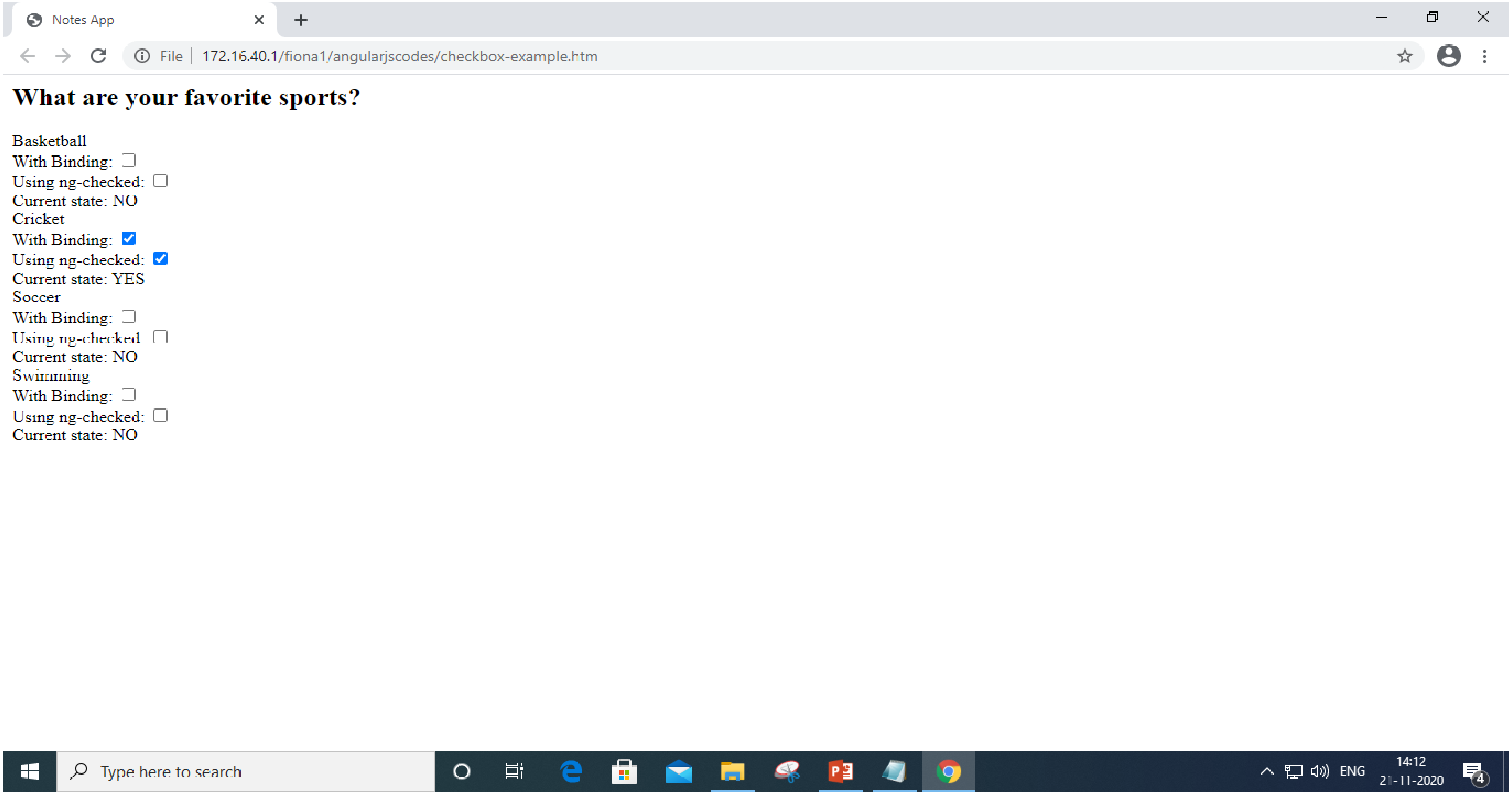
Output :when form loads


Notes App
File | 172.16.40.1/fiona1/angularjscodes/checkbox-example.htm
Type here to search
ENG
14:11
21-11-2020

What are your favorite sports?

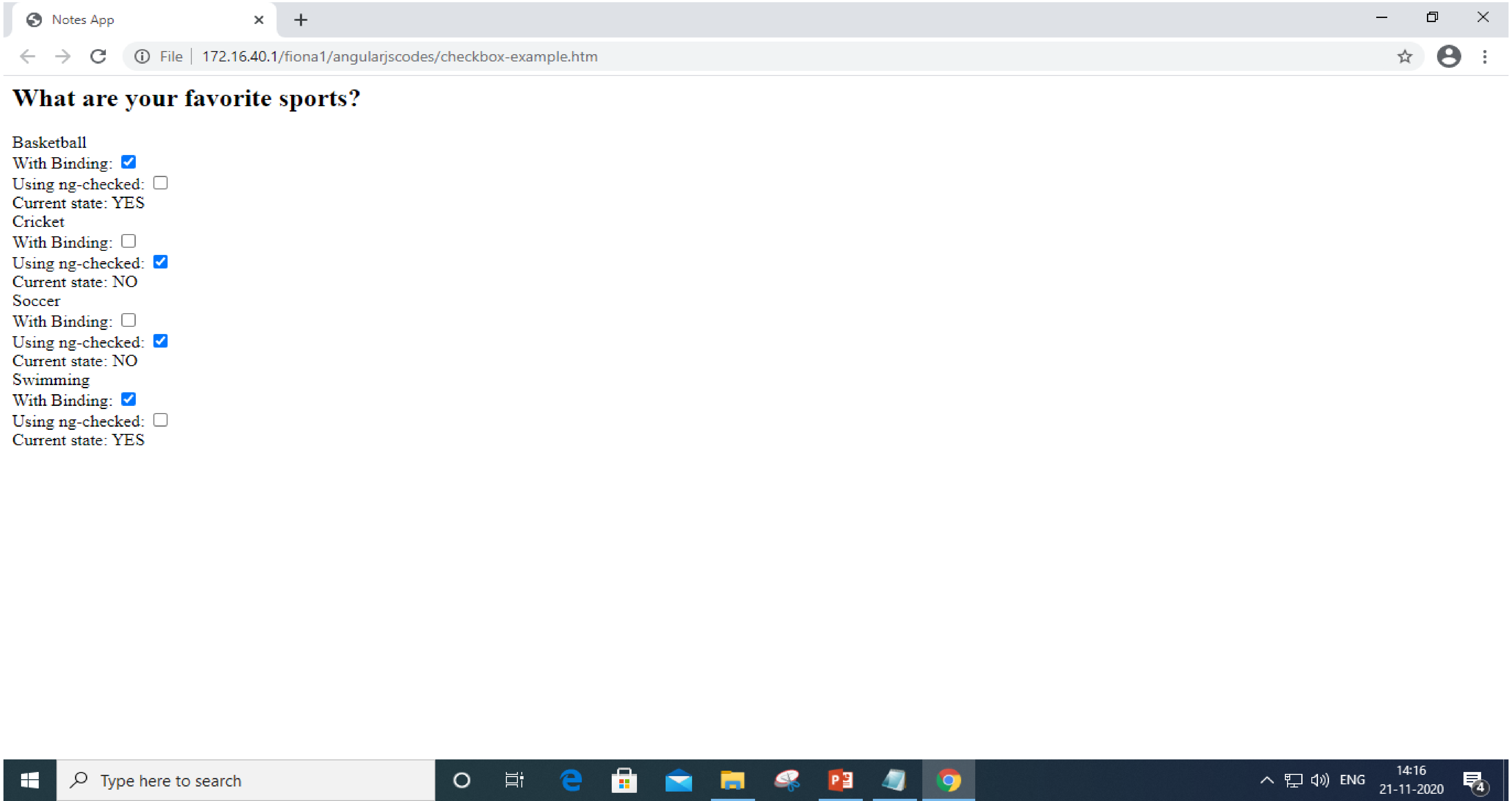
```
Basketball
With Binding: ☑
Using ng-checked: ☑
Current state: YES
Cricket
With Binding: ☐
Using ng-checked: ☐
Current state: NO
Soccer
With Binding: ☐
Using ng-checked: ☐
Current state: NO
Swimming
With Binding: ☑
Using ng-checked: ☑
Current state: YES
```

Output :when binding checkbox(unselected/selected)


Notes App
File | 172.16.40.1/fiona1/angularjscodes/checkbox-example.htm
Type here to search
ENG
14:12
21-11-2020

```
What are your favorite sports?

Basketball
With Binding: ☐
Using ng-checked: ☐
Current state: NO
Cricket
With Binding: ✓
Using ng-checked: ✓
Current state: YES
Soccer
With Binding: ☐
Using ng-checked: ☐
Current state: NO
Swimming
With Binding: ☐
Using ng-checked: ☐
Current state: NO
```

Output :when ng-checked checkbox(unselected/selected)


Notes App
File | 172.16.40.1/fiona1/angularjscodes/checkbox-example.htm
Type here to search
ENG
14:16
21-11-2020

```
What are your favorite sports?

Basketball
With Binding: ☑
Using ng-checked: ☐
Current state: YES
Cricket
With Binding: ☐
Using ng-checked: ☑
Current state: NO
Soccer
With Binding: ☐
Using ng-checked: ☑
Current state: NO
Swimming
With Binding: ☑
Using ng-checked: ☐
Current state: YES
```

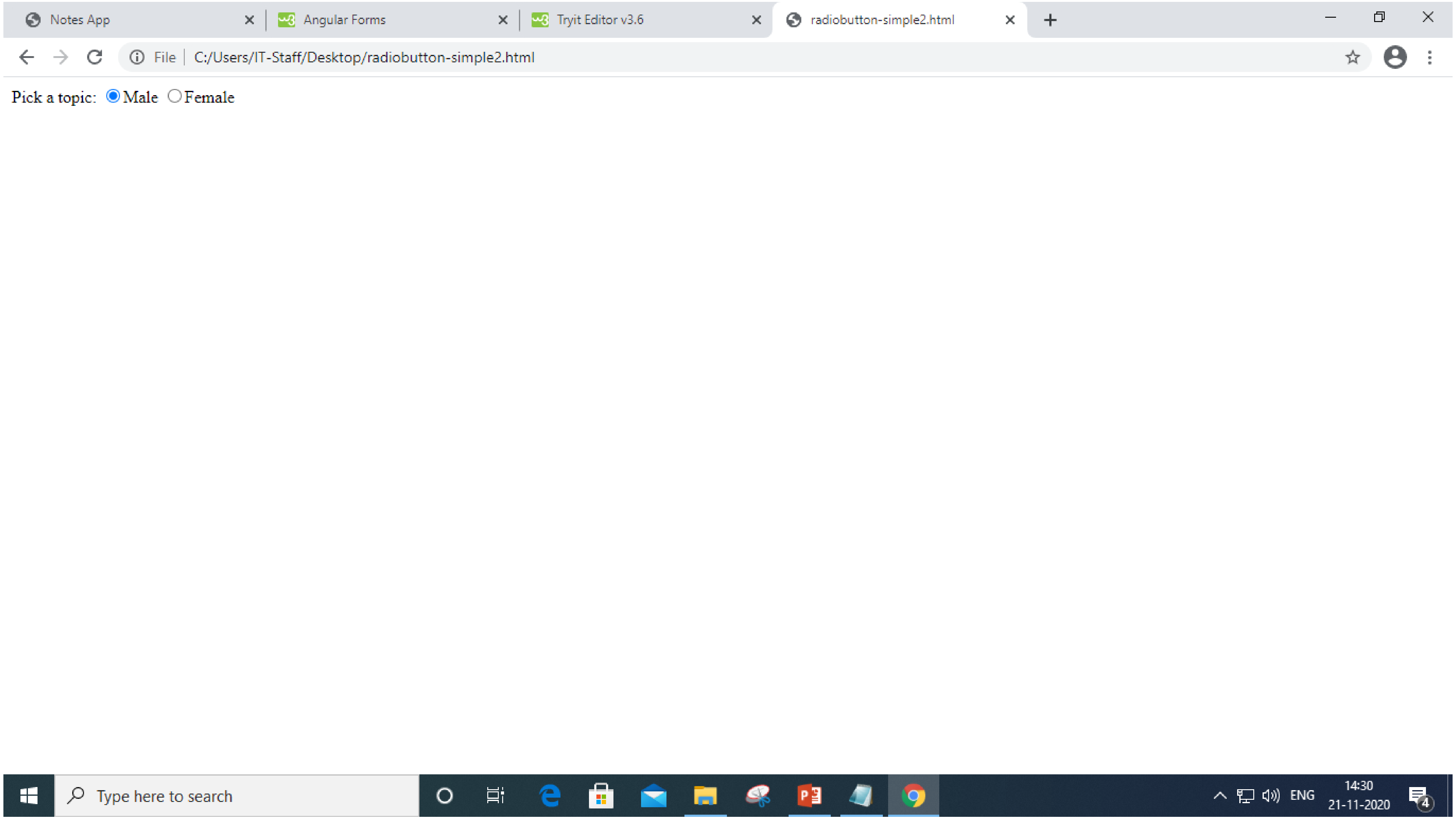
Other Form Controls

- This example has a checkbox with ng-model, a checkbox with ng-checked, and a div with the current state bound to it.
- The first checkbox uses the traditional two-way data-binding with the ng-model directive.
- The second uses ng-checked so when the user checks the first checkbox, the value of selected becomes YES as the true value, set using ng-true-value, is YES. This triggers the ng-checked and sets the second box as checked (or unchecked).
- When the user unchecks the first box, the value of selected is set to NO because of the ng-false-value.The second checkbox in each repeater element displays the state of the ng-model using ng-checked.
- This updates the state of the checkbox whenever the model backing ng-model changes.
- Checking or unchecking the second checkbox itself has no effect on the value of the model.
- So if you need two-way data-binding, use ng-model. If you need one-way data-binding with checkboxes, use ng-checked.

3.Radio Buttons

```
<div ng-init="user = {gender: 'female'}">
<input type="radio“ name="gender“ ng-model="user.gender“ value="male">
<input type="radio“ name="gender“ ng-model="user.gender“
value="female">
</div>
```

- Here we have two radio buttons having same name so when one is selected, the other gets deselected. Both are bound to the same ng-model (user.gender).
- each of them has a value, which is the value that gets stored in user.gender (male if it is the first radio button; female, otherwise).
- There is **ng-init** block surrounding it, which sets the value of user.gender to be female by default. This ensures that the second radiobutton is selected when this snippet of HTML loads.



Pick a topic: [✓ Male] Female

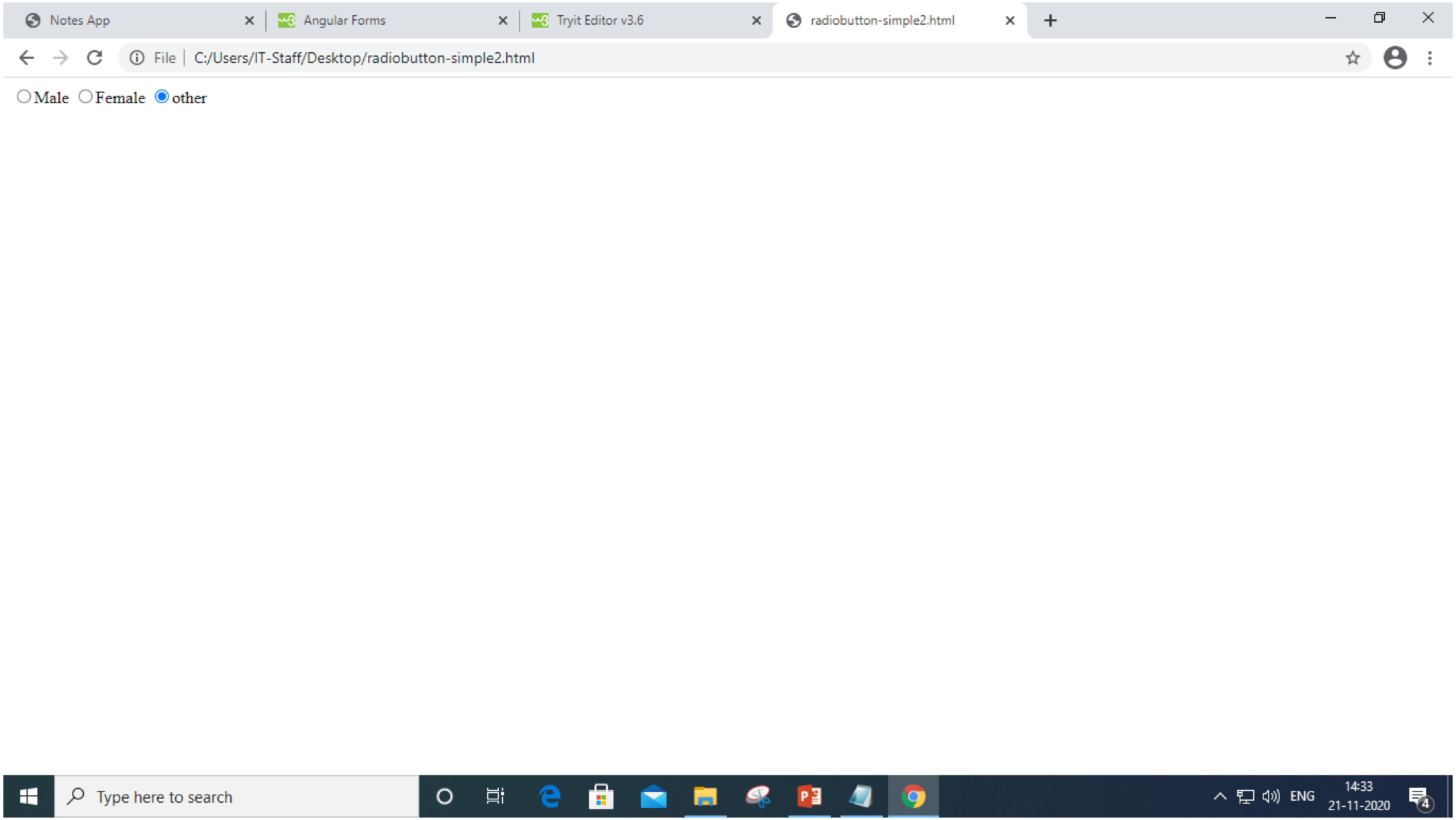
3.Radio Buttons

what if values are dynamic? What if the value are decided in our controller? Use the ng-value attribute, which you can use along with the radio buttons. ng-value takes expression, and the return value of the expression becomes the value that is assigned to the model:

```
<div ng-init="otherGender = 'other'">

<input type="radio“ name="gender"ng-model="user.gender“
value="male">Male

<input type="radio“ name="gender“ ng-model="user.gender“
value="female">Female

<input type="radio“ name="gender“ ng-model="user.gender"

ng-value="otherGender">{{otherGender}}

</div>
```

3.Radio Buttons

- In this example, the third option box takes a dynamic value.
- we assign it as part of the initialization block (ng-init), but in a real application, the initialization could be done from within a controller.
- ng-value="otherGender", it doesn’t assign otherGender as a string to user.gender, but the value of the otherGender variable, which is *other*.



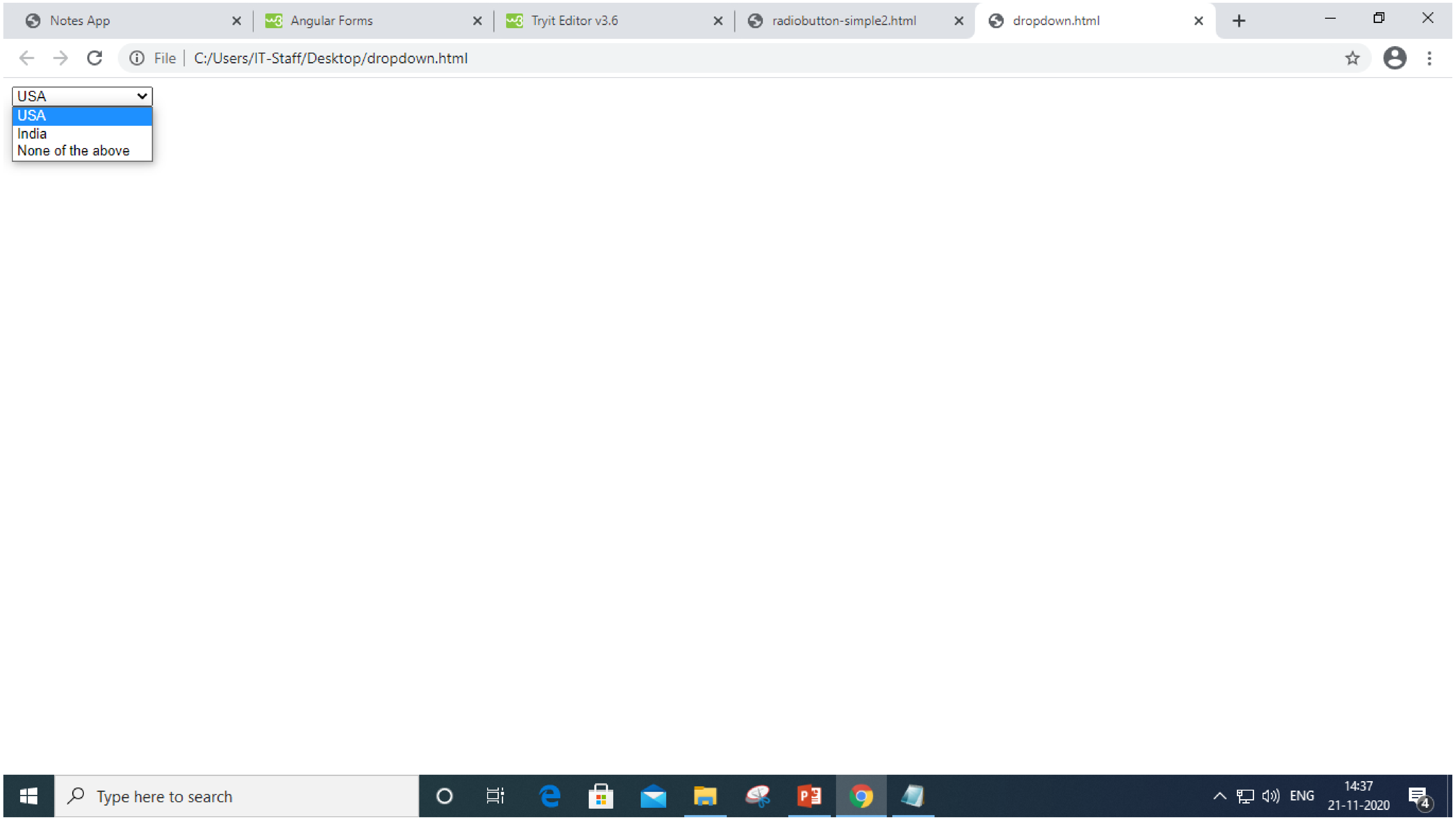
○ Male ○ Female [✓ ● other]

4.Combo Boxes/Drop-Downs

- The HTML form element (can be used outside forms also) is the select box, or the drop-down/combo box.

```
<div ng-init="location = 'India'">
<select ng-model="location">
<option value="USA">USA</option>
<option value="India">India</option>
<option value="Other">None of the above</option>
</select>
</div>
```

- we have a simple select box that is data-bound to the variable location.
- We also initialize the value of location to India, on loads,India is the selected.
- When the user selects any other options, the value of the value attribute gets assigned to the ng-model. The standard validators and states also apply to this field.
- This has a few restrictions, though:
    - You need to know the values in the drop-down up front.
    - They need to be hardcoded.
    - The values can only be strings.



USA

USA
India
None of the above

4.Combo Boxes/Drop-Downs

- In dynamic app the select HTML element can dynamically generate the list of options, and **work with objects** instead of pure string ng-models using ng-options directive

<html ng-app="notesApp"**>**

<head><title>Notes App**</title></head>**

<body ng-controller="MainCtrl as ctrl"**>**

<div>

<select ng-model="ctrl.selectedCountryId“ ng-options="c.id as c.label for c in ctrl.countries"**>**

</select>

Selected Country ID : {{ctrl.selectedCountryId}}

</div>

<div>

<select ng-model="ctrl.selectedCountry“ ng-options="c.label for c in ctrl.countries"**>**

</select>

Selected Country : {{ctrl.selectedCountry}}

4.Combo Boxes/Drop-Downs

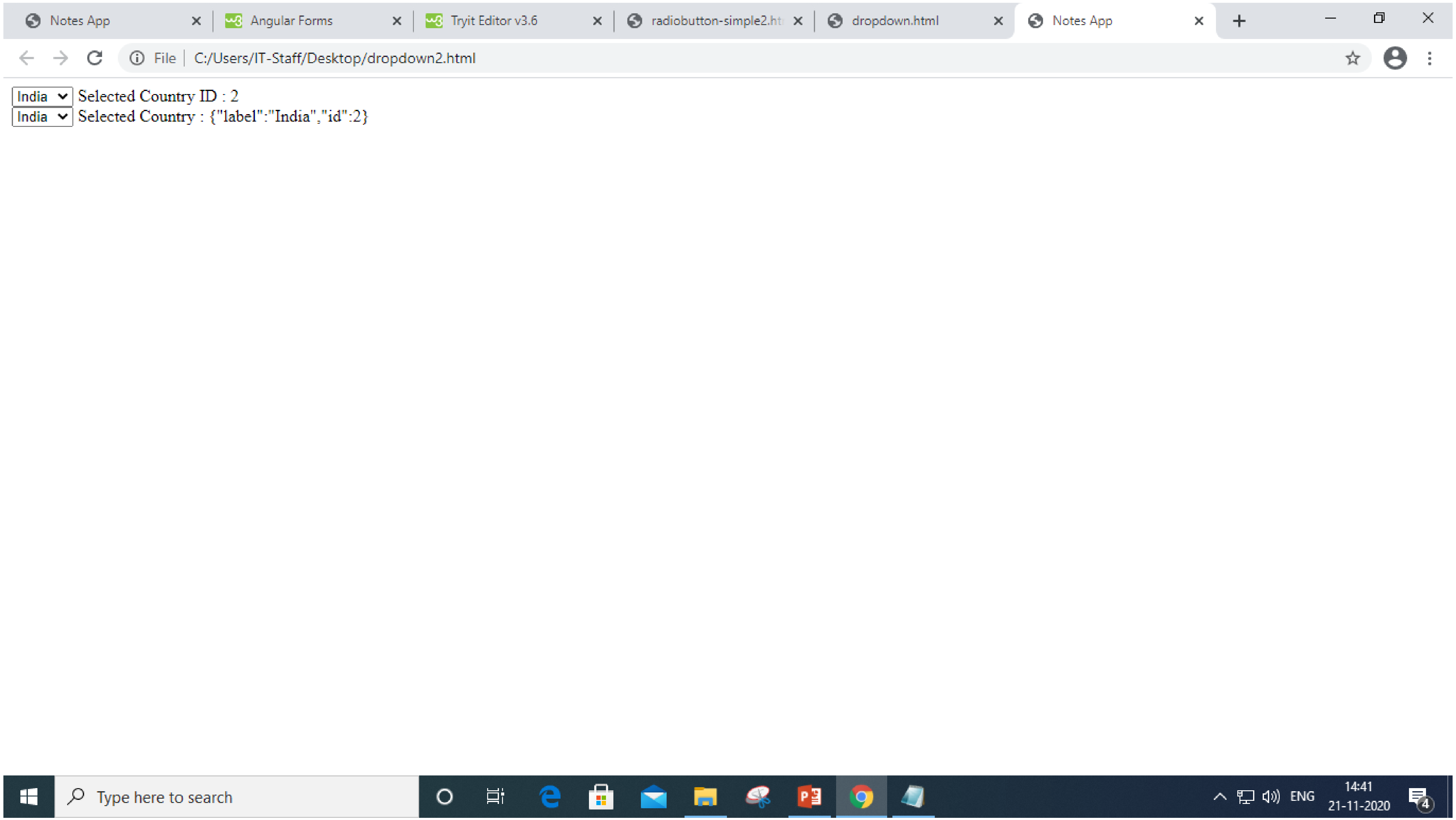
```
</div>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.2.19/angular.js">
</script>
<script type="text/javascript">
angular.module('notesApp', [])
.controller('MainCtrl', [function() {
this.countries = [{label: 'USA', id: 1},{label: 'India', id: 2},{label: 'Other', id: 3}];
this.selectedCountryId = 2;this.selectedCountry = this.countries[1];
}]);
</script>
</body>
</html>
```

4.Combo Boxes/Drop-Downs

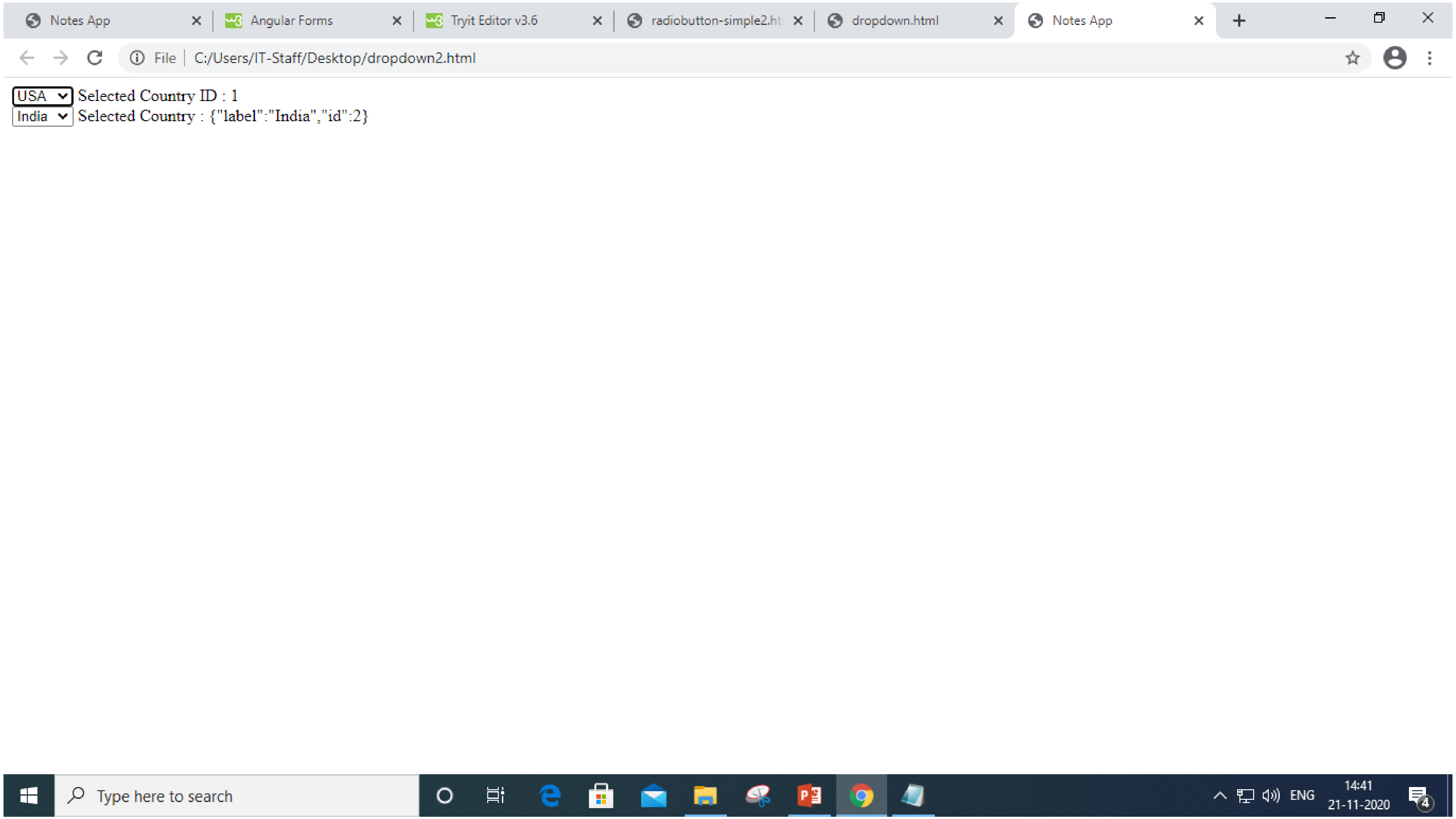
- Here, we have two select boxes, both bound to different models in our controller.
- The first select element is bound to ctrl.selectedCountryId and the second bound to ctrl.selectedCountry.
- Note one is a number, while the other is an actual object.
- ng-options attribute on the select allows to repeat an array (or object) and display dynamic options.
- similar to ng-repeat with additional ability to select what is displayed as the label, and what is bound to the model.
- In first select box, ng-options="c.id as c.label for c in ctrl.countries" tells AngularJS to create one option for each country in the array of countries.
- The syntax is : modelValue as labelValue for item in array.Here our modelValue is ID of each element, the label value is the label key of each array item, and then our typical for each loop.
- In second select ng-options="c.label for c in ctrl.countries“ we assume each item in the repeat is the actual model value, so when we select an item the country object (c) of that option box gets assigned to ctrl.selectedCountry.

4.Combo Boxes/Drop-Downs

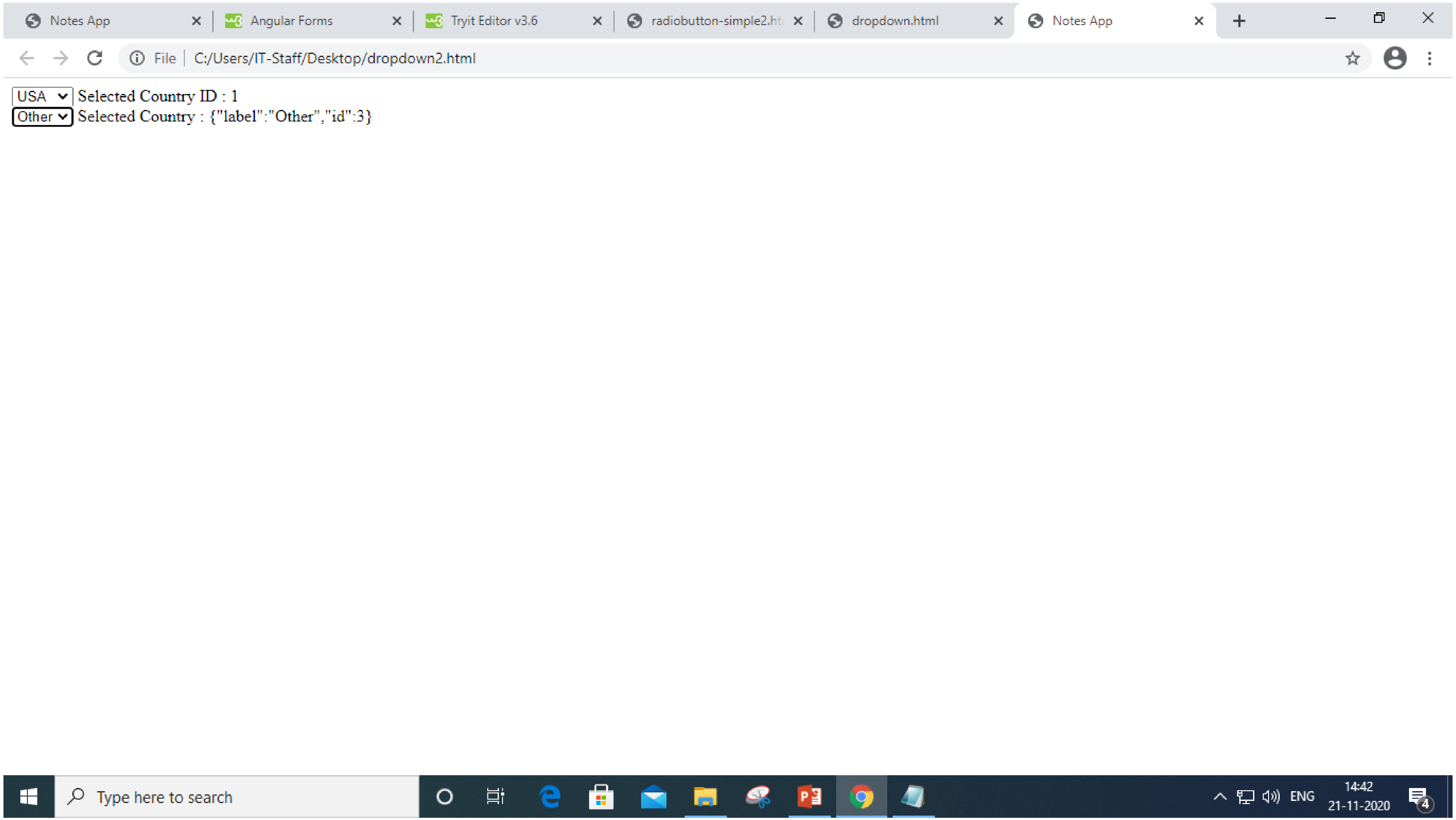
- You can also optionally give a grouping clause, for which the syntax would be ngoptions="modelValue as labelValue group by groupValue for item in array".
- Similar to how we specified the model and label values, we can point the groupValue at another key in the object (say, continent).
- When you use objects, the clause changes as follows: modelValue as labelValue group by groupValue for (key, value) in object.



India Selected Country ID : 2
India Selected Country : {"label":"India","id":2}



```
USA    Selected Country ID : 1
India  Selected Country : {"label":"India","id":2}
```



USA Selected Country ID : 1
Other Selected Country : {"label":"Other","id":3}