

**SRM Institute of Science and Technology**

**College of Engineering and Technology**

**School of Computing**

**Department of Computational Intelligence**

**B.Tech - Artificial Intelligence**

**Regulations 2018**

**18AIC302J – Web programming for Artificial Intelligence**

**LAB MANUAL**

**SRM Institute of Science and Technology**

**College of Engineering and Technology**

**School of Computing**

**Department of Computational Intelligence**

**B.Tech - Artificial Intelligence**

**Regulations 2018**

**18AIC302J – Web programming for Artificial Intelligence**

**Department Vision**

To build a world-renowned academic platform in Computational Intelligence by providing unique learning and research experiences in collaboration with industries and world-class universities.

**Department Mission**

* To envision in creating, acquiring, and disseminating engineering knowledge on computational intelligence to elevate a student into a professional by imparting knowledge on mathematics, computing sciences, artificial intelligence, and software engineering along with the skills of cognitive computing.
* To offer a unique learning environment through world class faculty, curriculum, modernized lab facilities, and an interactive classroom environment with real-time experience from industrial experts that leads to a computing career in the latest technologies.
* To uplift the innovative research and development in computational intelligence and its allied fields by collaborating with renowned academic institutions and industries.
* To produce graduates who are global innovators and leaders in the development of computational intelligence-based systems, along with the commitment to ethical responsibilities and lifelong learning**.**

**List of Experiments**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.No** | **Topic(s)** | **CO** | **PO** |
| 1 | Demonstrate the various HTML Controls | 1 | 2 |
| 2 | Styling the page using different Style sheet approaches | 1 | 2 |
| 3 | Creation of web page using Simple Bootstrap | 1 | 2 |
| 4 | Validating the Web site using JS | 2 | 2 |
| 5 | Demonstration of Various Java Script Events | 2 | 2 |
| 6 | Forwarding the Client Request using AJAX | 2 | 2 |
| 7 | Validating the Web site using JQuery | 2 | 2 |
| 8 | Validating the Web site using AngularJS | 3 | 2 |
| 9 | Validation of Server-Side application using Nodejs | 3 | 2 |
| 10 | Simple website using Django Framework | 4 | 2 |
| 11 | Integration of Database to Django project | 4 | 2 |
| 12 | Simple Web scrapper | 5 | 2 |

# Ex. No. :1 a

# HTML – Element - Tables

**Aim:**

Program to design a class timetable using HTML basic elements.

# Procedure:

1. Tables are defined with the **table** element
2. Use the ***border*** attribute specifies the table’s border width in pixels. To create a table without a border, set **border** to **"0"**.
3. Use the ***tr*** element to define an individual *table row*.
4. The columns in the head section are defined with **th** elements.
5. *Data cells* contain individual pieces of data and are defined with **td** (*table data*) elements within each row.
6. Table cells are sized to fit the data they contain. Document authors can create larger data cells by using attributes ***rowspan*** and ***colspan***. The values assigned to these attributes specify the number of rows or columns occupied by a cell.
7. Use the attribute **rowspan = "2"** to allow the cell to use two vertically adjacent cells (thus the cell *spans* two rows).
8. Use the attribute **colspan = "4"** to widen the header cell to span four cells.

# Ex. No. :1 b

# HTML – Form Elements

**Aim:**

Program to design a email registration form using HTML Form elements.

# Procedure:

1. The form is defined by a ***form*** element.

**<form method = "post" action = "/cgi-bin/formmail">**

1. Use thea attribute ***method*** specifies how the form’s data is sent to the Web server.Using ***method = "post"*** appends form data to the browser request, which contains the protocol (i.e., HTTP) and the requested resource’s URL. Scripts located on the Web server’s computer (or on a computer accessible through the network) can access the form data sent as part of the request. For example, a script may take the form information and update an electronic mailing list. The other possible value, ***method = "get"*** appends the form data directly to the end of the URL.
2. The ***action*** attribute in the **<form>** tag specifies the URL of a script on the Web server; in this case, it specifies a script that e-mails form data to an address. Most Internet Service Providers (ISPs) have a script like this on their site; ask the Web site system administrator how to set up an XHTML document to use the script correctly.
3. Use the **type** of **input** as ***"text"* input** inserts a *text box* into the form. Users can type data in text boxes.
4. The **input** element’s ***size*** attribute specifies the number of characters visible in the text box. Optional attribute ***maxlength*** limits the number of characters input into the text box.
5. There are two types of **input** elements in lines

**<input type = "submit" value = "Submit Your Entries" />**

**<input type = "reset" value = "Clear Your Entries" />**

1. The ***"submit"* input** element is a button. When the user presses a **"submit"** button, the browser sends the data in the form to the Web server for processing. The ***value*** *attribute* sets the text displayed on the button (the default value is **Submit Query**).
2. The ***"reset"* input** element allows a user to reset all **form** elements to their default values. The **value** attribute of the **"reset" input** element sets the text displayed on the button (the default value is **Reset**).
3. The ***textarea*** element inserts a multiline text box, called a *text area*, into the form. The number of rows is specified with the ***rows*** *attribute* and the number of columns (i.e., characters) is specified with the ***cols*** *attribute.* In this example, the **textarea** is four rows high and 36 characters wide. To display default text in the text area, place the text between the **<textarea>** and **</textarea>** tags. Default text can be specified in other **input** types, such as text boxes, by using the **value** attribute.
4. The ***"password"*** input in lines inserts a password box with the specified **size**. A password box allows users to enter sensitive information, such as credit card numbers and passwords, by “masking” the information input with asterisks. The actual value input is sent to the Web server, not the character that mask the input.
5. Checkboxes enable users to select from a set of options. When a user selects a checkbox, a check mark appears in the check box. Otherwise, the checkbox remains empty. Each ***"checkbox"* input** creates a new checkbox. Checkboxes can be used individually or in groups. Checkboxes that belong to a group are assigned the same **name**.
6. Radio buttons are similar to checkboxes, except that only one radio button in a group of radio buttons may be selected at any time. The radio buttons in a group have the same **name** attributes and are distinguished by their different **value** attributes. The attribute- value pair ***checked = "checked"*** indicates which radio button, if any, is selected initially. The **checked** attribute also applies to checkboxes.
7. The ***select*** element provides a drop-down list of items from which the user can select an item. The **name** attribute identifies the drop-down list. The ***option*** element adds items to the drop-down list. The **option** element’s ***selected*** *attribute* specifies which item initially is displayed as the selected item in the **select** element.

**Ex. No. : 2**

**CSS**

# Aim:

Program to design web pages using basic elements, hyperlinks and to perform web navigation using CSS.

# Procedure:

Inline Style Sheets

1. Create inline styles that declare an individual element’s format using attribute style.
2. Apply inline styles to p elements to alter their font size and color.
3. Use the attribute style to specify the style for an element.
4. Create CSS property (the font-size property) followed by a colon and a value.
5. Use the two properties, font-size and color, separated by a semicolon.

Embedded Style Sheets

1. Use the style element to define the embedded style sheet.
2. Place the Styles in the head to apply matching elements in the entire document, not just to a single element.
3. Use the type attribute to specify specifies the Multipurpose Internet Mail Extension (MIME) type that describes a file’s content. CSS documents use the MIME type text/css.
4. Use the body of the style sheet to declare the CSS rules for the style sheet.
5. The body of each rule is enclosed in curly braces ({ and }).
6. Declare a style class. Class declarations are preceded with a period and are applied to elements only of that class.
7. Use the property name is followed by a colon (:) and the value of that property. Multiple properties are separated by semicolons (;).

Linking External Style Sheets

1. Create a link element, which uses the rel attribute to specify a relationship between the current document and another document.
2. Declare the linked document to be a stylesheet for this document.
3. Use the type attribute to specify the MIME type as text/css.
4. Use the href attribute provides the URL for the document containing the style sheet.

**Ex.No: 3**

**Creation of web page using Simple Bootstrap**

**Aim**

To create web sites which automatically adjust themselves to look good on all devices, from small phones to large desktops.

**Bootstrap**

* Bootstrap is a free front-end framework for faster and easier web development
* Bootstrap includes HTML and CSS based design templates for typography, forms, buttons, tables, navigation, modals, image carousels and many other, as well as optional JavaScript plugins
* Bootstrap also gives you the ability to easily create responsive designs

**Bootstrap CDN**

<!-- Latest compiled and minified CSS -->  
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css">  
  
<!-- jQuery library -->  
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.7.1/jquery.min.js"></script>  
  
<!-- Latest compiled JavaScript -->  
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/js/bootstrap.min.js"></script>

**Procedure**

1. Add the HTML5 doctype

Bootstrap uses HTML elements and CSS properties that require the HTML5 doctype.

Always include the HTML5 doctype at the beginning of the page, along with the lang attribute and the correct character set.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

</head>

</html>

2. Bootstrap 3 is mobile-first

Bootstrap 3 is designed to be responsive to mobile devices. Mobile-first styles are part of the core framework.

To ensure proper rendering and touch zooming, add the following <meta> tag inside the <head> element:

<meta name="viewport" content="width=device-width, initial-scale=1">

The width=device-width part sets the width of the page to follow the screen-width of the device (which will vary depending on the device).

The initial-scale=1 part sets the initial zoom level when the page is first loaded by the browser.

3. Containers

Bootstrap also requires a containing element to wrap site contents.

There are two container classes to choose from:

The .container class provides a responsive fixed width container

The .container-fluid class provides a full width container, spanning the entire width of the viewport

**Example**

<!DOCTYPE html>

<html lang="en">

<head>

<title>Bootstrap Example</title>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css">

<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.7.1/jquery.min.js"></script>

<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/js/bootstrap.min.js"></script>

</head>

<body>

<div class="container">

<h1>My First Bootstrap Page</h1>

<p>This is some text.</p>

</div>

</body>

</html>

**Ex.No: 4**

**Web page Validation Using Java Script**

**Aim**

To verify whether the data entered into a form needs to be in the right format and certain fields need to be filled in order to effectively use the submitted form.

**Web Page Validation Procedure**

Data validation is the process of ensuring that user input is clean, correct, and useful.

Typical validation tasks are:

has the user filled in all required fields?

has the user entered a valid date?

has the user entered text in a numeric field?

Most often, the purpose of data validation is to ensure correct user input.

For example,

* Validate the name and password.
* The name can’t be empty and password can’t be less than 6 characters long.

**HTML Constraint Validation**

HTML5 introduced a new HTML validation concept called constraint validation.

HTML constraint validation is based on:

* Constraint validation HTML Input Attributes
* Constraint validation CSS Pseudo Selectors
* Constraint validation DOM Properties and Methods

**Constraint Validation HTML Input Attributes**

Attribute Description

disabled Specifies that the input element should be disabled

max Specifies the maximum value of an input element

min Specifies the minimum value of an input element

pattern Specifies the value pattern of an input element

required Specifies that the input field requires an element

type Specifies the type of an input element

**Constraint Validation CSS Pseudo Selectors**

Selector Description

:disabled Selects input elements with the "disabled" attribute specified

:invalid Selects input elements with invalid values

:optional Selects input elements with no "required" attribute specified

:required Selects input elements with the "required" attribute specified

:valid Selects input elements with valid values

**Example Code**

<script>

function validateform(){

var name=document.myform.name.value;

var password=document.myform.password.value;

if (name==null || name==""){

alert("Name can't be blank");

return false;

}else if(password.length<6){

alert("Password must be at least 6 characters long.");

return false;

}

}

</script>

<body>

<form name="myform" method="post" action="abc.jsp" onsubmit="return validateform()" >

Name: <input type="text" name="name"><br/>

Password: <input type="password" name="password"><br/>

<input type="submit" value="register">

</form>

**Ex.No: 5**

**Demonstration of Various Java Script Events**

**Aim**

To add various Java Script Events in a web page to provide a dynamic interface to a webpage.

**Events**

Event handlers can be used to handle and verify user input, user actions, and browser actions:

* Things that should be done every time a page loads
* Things that should be done when the page is closed
* Action that should be performed when a user clicks a button
* Content that should be verified when a user inputs data

Many different methods can be used to let JavaScript work with events:

* HTML event attributes can execute JavaScript code directly
* HTML event attributes can call JavaScript functions
* You can assign your own event handler functions to HTML elements
* You can prevent events from being sent or being handled

**Common HTML Events**

Event Description

onchange An HTML element has been changed

onclick The user clicks an HTML element

onmouseover The user moves the mouse over an HTML element

onmouseout The user moves the mouse away from an HTML element

onkeydown The user pushes a keyboard key

onload The browser has finished loading the page

**Example**

<!doctype html>

<html>

<head>

<script>

function hiThere() {

alert('Hi there!');

}

</script>

</head>

<body>

<button type="button"

onclick="hiThere()"

style="margin-left: 50%;">

Click me event

</button>

</body>

</html>

**Ex.No: 6**

**Forwarding the Client Request using AJAX**

**Aim:**

Program to create a simple XMLHttpRequest, and retrieve data froma TXTfile.

**Procedure:**

1. Create a text document file rec.txt.

2. Type some context in that file.

3. Create a HTML document file File.html.

4. Inside the BODY tag create one h2 element, one p element and one button.

5. Make the button to calls a function named loadDoc(), if it is clicked.

6. Add a <script> tag to the page's body section.

7. Inside the script section create the loadDoc() function.

8. To send a request to a server, use the open() method of the XMLHttpRequest object.

9. Use the url parameter of the open() method, an address to a file on a server.

10. Use the responseText property returns the response as a string, and can use it accordingly.

**Example**

**File.html**

<!DOCTYPE html>

<html>

<head>

<meta charset="ISO-8859-1">

<title>AJAX File</title>

</head>

<body>

<h2>AJAX File Example</h2>

<p id="demo">Let AJAX change this text.</p>

<button type="button" onclick="loadDoc()">Change Content</button>

<script>

function loadDoc() {

var xhttp = new XMLHttpRequest();

xhttp.onreadystatechange = function()

{ if (this.readyState == 4 && this.status == 200) {

document.getElementById("demo").innerHTML = this.responseText;

}

};

xhttp.open("GET", "rec.txt", true);

xhttp.send();

}

</script>

</body>

</html>

**rec.txt**

<hr />

<h1>Dr.R.Babu </h1>

<hr />

<b>School of Computing</b> <br />

<i>Department of Computational Intelligence</i>

<br />

<i>University Building, SRMIST </i>

<br />

<i>Chennai </i>

**Ex.No: 7**

**Validating the Web site using JQuery**

**Aim**

To validate a simple form that consists of a username, password, and a confirmed password using jQuery.

**Procedure**

1. First, you need to create an index.html file consisting of an HTML form with username, email, password, and confirm password as input fields. We will use Bootstrap 4 to use Bootstrap’s form controls. At the bottom of the <body> tag, include the “app.js” file having jQuery code for form validation.
2. Create an app.js file that validates the whole form as given below in the code.
3. In the app.js file, when the document is ready, hide all the error messages. Perform the validation task for all the input fields such as username, email, password, and confirm password.

**Example**

HTML

<!DOCTYPE html>

<html>

<head>

<!-- Latest compiled and minified CSS -->

<link rel="stylesheet" href=

"https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css">

<!-- jQuery library -->

<script src=

"https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">

</script>

<!-- Popper JS -->

<script src=

"https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.12.9/umd/popper.min.js">

</script>

<!-- Latest compiled JavaScript -->

<script src=

"https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootstrap.min.js">

</script>

</head>

<body>

<h1 class="text-center text-success">

Welcome to SRM

</h1>

<p class="text-center">

FORM VALIDATION USING JQUERY

</p>

<div class="container">

<div class="col-lg-8

m-auto d-block">

<form>

<div class="form-group">

<label for="user">

Username:

</label>

<input type="text"

name="username"

id="usernames"

class="form-control">

<h5 id="usercheck"

style="color: red;">

\*\*Username is missing

</h5>

</div>

<div class="form-group">

<label for="user">

Email:

</label>

<input type="email"

name="email"

id="email" required

class="form-control">

<small id="emailvalid"

class="form-text text-muted invalid-feedback">

Your email must be a valid email

</small>

</div>

<div class="form-group">

<label for="password">

Password:

</label>

<input type="password"

name="pass"

id="password"

class="form-control">

<h5 id="passcheck"

style="color: red;">

\*\*Please Fill the password

</h5>

</div>

<div class="form-group">

<label for="conpassword">

Confirm Password:

</label>

<input type="password"

name="username"

id="conpassword"

class="form-control">

<h5 id="conpasscheck"

style="color: red;">

\*\*Password didn't match

</h5>

</div>

<input type="submit"

id="submitbtn"

value="Submit"

class="btn btn-primary">

</form>

</div>

</div>

<!-- Including app.js jQuery Script -->

<script src="app.js"></script>

</body>

</html>

**JS**

<!DOCTYPE html>

<html>

<head>

<!-- Latest compiled and minified CSS -->

<link rel="stylesheet" href=

"https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css">

<!-- jQuery library -->

<script src=

"https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">

</script>

<!-- Popper JS -->

<script src=

"https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.12.9/umd/popper.min.js">

</script>

<!-- Latest compiled JavaScript -->

<script src=

"https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootstrap.min.js">

</script>

</head>

<body>

<h1 class="text-center text-success">

Welcome to SRM

</h1>

<p class="text-center">

FORM VALIDATION USING JQUERY

</p>

<div class="container">

<div class="col-lg-8

m-auto d-block">

<form>

<div class="form-group">

<label for="user">

Username:

</label>

<input type="text"

name="username"

id="usernames"

class="form-control">

<h5 id="usercheck"

style="color: red;">

\*\*Username is missing

</h5>

</div>

<div class="form-group">

<label for="user">

Email:

</label>

<input type="email"

name="email"

id="email" required

class="form-control">

<small id="emailvalid"

class="form-text text-muted invalid-feedback">

Your email must be a valid email

</small>

</div>

<div class="form-group">

<label for="password">

Password:

</label>

<input type="password"

name="pass"

id="password"

class="form-control">

<h5 id="passcheck"

style="color: red;">

\*\*Please Fill the password

</h5>

</div>

<div class="form-group">

<label for="conpassword">

Confirm Password:

</label>

<input type="password"

name="username"

id="conpassword"

class="form-control">

<h5 id="conpasscheck"

style="color: red;">

\*\*Password didn't match

</h5>

</div>

<input type="submit"

id="submitbtn"

value="Submit"

class="btn btn-primary">

</form>

</div>

</div>

<!-- Including app.js jQuery Script -->

<script src="app.js"></script>

</body>

</html>

**Ex.No: 8**

**Validating the Web site using AngularJS**

**Aim**

To create a form with various elements, such as checkboxes, input textboxes, number text, and email text and to validate form and input field controls on the client side.

**Procedure**

There are three states that are used to detect the errors:

* $dirty: This says that the value has been altered.
* $error: This displays the exact error.
* $invalid: This states that entered value is invalid.

States of AngularJS Form Validation

AngularJS stores the data of changes in the input field. Let's have a look at the states of form input fields.

|  |  |
| --- | --- |
| States | Description |
| $valid | It indicates that the field's content is valid. |
| $invalid | It indicates that the field's content is not valid. |
| $dirty | It shows that the field is modified. |
| $pristine | It shows that the field is not modified yet. |
| $touched | It indicates that the field is touched. |
| $untouched | It indicates that the field is not touched. |
| $submitted | It indicates that the form has been submitted. |

Validation Directives

Now, let's see the validation directives in the AngularJS Form Validation.

|  |  |
| --- | --- |
| Directive | Description |
| ng-required | It is used to make a field required to fill. |
| ng-minlength | It is used to set a minimum length of the input. |
| ng-maxlength | It is used to set the maximum length of the input. |
| ng-pattern | It shows a pattern validation error key if the value of the ngModel parameter does not match the given RegEx expression. |

**Example 1**

<!DOCTYPE html>

<html>

   <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>

   <body ng-app="">

       <p>Enter your name in the input box:</p>

       <form name="myForm">

           <input name="myInput" ng-model="myInput" required />

       </form>

       <h3>Validation State is:</h3>

       <h1>{{myForm.myInput.$valid}}</h1>

   </body>

</html>

**Explanation**

* The validation state will show the message "false" if the input box is empty.
* Similarly, the validation state will show the message "true" as soon as we enter something in the input box.

**Example 2**

<!DOCTYPE html>

<html>

   <head>

       <title>AngularJs Form Validation</title>

       <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

       <script type="text/javascript">

           var app = angular.module('formApp', []);

           app.controller('formCtrl', function ($scope) {

               $scope.sendForm = function () {

                   $scope.msg = 'Form Validated Successfully!';

               };

           });

       </script>

   </head>

   <body>

       <div ng-app="formApp" ng-controller="formCtrl">

           <form name="personForm" ng-submit="sendForm()">

               Name: <input type="text" name="name" ng-model="person.name" required />

               <span ng-show="personForm.name.$error.required"> Required! </span>

               <br /><br />

               Age: <input type="number" name="age" ng-model="person.age" required />

               <span ng-show="personForm.age.$error.required"> Required! </span>

               <br /><br />

               <button type="submit">Submit</button><br /><br />

               <span>{{msg}}</span>

           </form>

       </div>

   </body>

</html>

**Explanation**

Let's understand the output one by one.

At first, the form will ask for the Name and the Age. It has a submit button also.

Now the question is, Will our form still validate if we did not fill or misses any field? The answer is "No". The form will ask you to fill out the fields you missed.

The message "Form Validated Successfully!" will be displayed when the details are entered correctly.

In this example, the first field (name) will accept any text, but it will only take a number in the second field (age).

**Custom Validation**

Custom Validation is a term used while creating conditions that are not declared by default. For instance, you can set your validations at any specific place that will be checked while validating any result.

AngularJS offers HTML input types like text, number, date, radio, checkbox, etc., and validation directives like min, max, pattern, required, etc., to execute custom validation. We can make our validations using the $validators. The $validators accept two parameters, modelValue and viewValue.

In the below example, the $parsers command takes myValidation as a parameter. An expression can be parsed and evaluated using the function that the $parsers returns.

<!DOCTYPE html>

<html>

   <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>

   <body ng-app="myApp">

       <p>Fill the given form:</p>

       <form name="myForm">

           Name:

           <input

               name="myInput"

               ng-model="myInput"

               required

               my-directive

           /><br /><br />

           Age:

           <input name="age" type="number" required /><br /><br />

       </form>

       <p>

           The Name field must contain the character "N" to be considered as valid.

       </p>

       <h3>Validation State is:</h3>

       <h1>{{myForm.myInput.$valid}}</h1>

       <script>

           var app = angular.module('myApp', []);

           app.directive('myDirective', function () {

               return {

                   require: 'ngModel',

                   link: function (scope, element, attr, mCtrl) {

                       function myValidation(value) {

                           if (value.indexOf('N') > -1) {

                               mCtrl.$setValidity('charE', true);

                           } else {

                               mCtrl.$setValidity('charE', false);

                           }

                           return value;

                       }

                       mCtrl.$parsers.push(myValidation);

                   },

               };

           });

       </script>

   </body>

</html>

Output:

Explanation:

The output shows the empty fields with a false validation state. This will come to your screen when you first run the program.

The validation state is true in the output as the Name field has the character ‘N’ in it.

The validation state will remain false until it encounters the character ‘N’ in the Name field.

**Ex.No: 9**

**Validation of Server-Side application using Nodejs**

**Aim**

To do some server-side form validation using the Express-validator.

**Procedure**

**Step 1:** Install Libraries

* npm install ejs
* npm install express
* npm install body-parser
* npm install express-validator

**Step 2:** Create a file named: **SampleForm.ejs**

<!DOCTYPE html>

<html>

    <head>

        <title>Validation using Express-Validator</title>

    </head>

<body>

<h1>Demo Form</h1>

<form action="saveData" method="POST">

<pre>

    Enter your Email : <input type="text" name="email"> <br>

    Enter your Name  : <input type="text" name="name"> <br>

    Enter your Number : <input type="number" name="mobile"> <br>

    Enter your Password : <input type="password" name="password"> <br>

    <input type="submit" value="Submit Form">

</pre>

</form>

</body>

</html>

**Step 3:** Create a file named: **index.js**

const { check, validationResult }

    = require('express-validator');

const bodyparser = require('body-parser')

const express = require("express")

const path = require('path')

const app = express()

var PORT = process.env.port || 3000

// View Engine Setup

app.set("views", path.join(\_\_dirname))

app.set("view engine", "ejs")

// Body-parser middleware

app.use(bodyparser.urlencoded({ extended: false }))

app.use(bodyparser.json())

app.get("/", function (req, res) {

    res.render("SampleForm");

})

// check() is a middleware used to validate

// the incoming data as per the fields

app.post('/saveData', [

    check('email', 'Email length should be 10 to 30 characters')

                    .isEmail().isLength({ min: 10, max: 30 }),

    check('name', 'Name length should be 10 to 20 characters')

                    .isLength({ min: 10, max: 20 }),

    check('mobile', 'Mobile number should contains 10 digits')

                    .isLength({ min: 10, max: 10 }),

    check('password', 'Password length should be 8 to 10 characters')

                    .isLength({ min: 8, max: 10 })

], (req, res) => {

    // validationResult function checks whether

    // any occurs or not and return an object

    const errors = validationResult(req);

    // If some error occurs, then this

    // block of code will run

    if (!errors.isEmpty()) {

        res.json(errors)

    }

    // If no error occurs, then this

    // block of code will run

    else {

        res.send("Successfully validated")

    }

});

app.listen(PORT, function (error) {

    if (error) throw error

    console.log("Server created Successfully on PORT ", PORT)

})

Step 4: node index.js

**Ex.No: 10**

**Simple website using Django Framework**

**Aim**

To create basic HelloWorld application using Django Framework.

**Procedure**

1. From the command line, cd into a directory where you’d like to store your code, then run the following command:

$ django-admin startproject mysite

This will create a mysite directory in your current directory

1. Look at [startproject](https://docs.djangoproject.com/en/4.2/ref/django-admin/#django-admin-startproject) created:

mysite/

manage.py

mysite/

\_\_init\_\_.py

settings.py

urls.py

asgi.py

wsgi.py

These files are:

The outer mysite/ root directory is a container for your project. Its name doesn’t matter to Django; you can rename it to anything you like.

* manage.py: A command-line utility that lets you interact with this Django project in various ways. You can read all the details about manage.py in [django-admin and manage.py](https://docs.djangoproject.com/en/4.2/ref/django-admin/).
* The inner mysite/ directory is the actual Python package for your project. Its name is the Python package name you’ll need to use to import anything inside it (e.g. mysite.urls).
* mysite/\_\_init\_\_.py: An empty file that tells Python that this directory should be considered a Python package.
* mysite/settings.py: Settings/configuration for this Django project.
* mysite/urls.py: The URL declarations for this Django project; a “table of contents” of your Django-powered site.
* mysite/asgi.py: An entry-point for ASGI-compatible web servers to serve your project.
* mysite/wsgi.py: An entry-point for WSGI-compatible web servers to serve your project.

1. Change into the outer mysite directory, if you haven’t already, and run the following commands:

$ python manage.py runserver

1. To create your app, make sure you’re in the same directory as manage.py and type this command:

$ python manage.py startapp polls

That’ll create a directory polls, which is laid out like this:

polls/

\_\_init\_\_.py

admin.py

apps.py

migrations/

\_\_init\_\_.py

models.py

tests.py

views.py

1. Open the file polls/views.py and put the following Python code in it:

polls/views.py

from django.http import HttpResponse

def index(request):

return HttpResponse("Hello, world. You're at the polls index.")

1. In the polls/urls.py file include the following code:

polls/urls.py

from django.urls import path

from . import views

urlpatterns = [

path("", views.index, name="index"),

]

1. The next step is to point the root URLconf at the polls.urls module. In mysite/urls.py, add an import for django.urls.include and insert an include() in the urlpatterns list, so you have:

mysite/urls.py

from django.contrib import admin

from django.urls import include, path

urlpatterns = [

path("polls/", include("polls.urls")),

path("admin/", admin.site.urls),

]

The include() function allows referencing other URLconfs. Whenever Django encounters include(), it chops off whatever part of the URL matched up to that point and sends the remaining string to the included URLconf for further processing.

1. $ python manage.py runserver

**Ex.No: 11**

**Integration of Database to Django project**

**Aim**

To use sqlite3 to create database and integrate with the application created using Djano

**Procedure**

* 1. Database setup

Now, open up mysite/settings.py. It’s a normal Python module with module-level variables representing Django settings. The default value, BASE\_DIR / 'db.sqlite3', will store the file in your project directory.

* 1. $ python manage.py migrate

The migrate command looks at the INSTALLED\_APPS setting and creates any necessary database tables according to the database settings in your mysite/settings.py file and the database migrations shipped with the app (we’ll cover those later). You’ll see a message for each migration it applies. If you’re interested, run the command-line client for your database and type \dt (PostgreSQL), SHOW TABLES; (MariaDB, MySQL), .tables (SQLite), or SELECT TABLE\_NAME FROM USER\_TABLES; (Oracle) to display the tables Django created.

* 1. Edit the polls/models.py file so it looks like this:

polls/models.py¶

from django.db import models

class Question(models.Model):

question\_text = models.CharField(max\_length=200)

pub\_date = models.DateTimeField("date published")

class Choice(models.Model):

question = models.ForeignKey(Question, on\_delete=models.CASCADE)

choice\_text = models.CharField(max\_length=200)

votes = models.IntegerField(default=0)

* 1. to include the polls app. Let’s run another command:

$ python manage.py makemigrations polls

* 1. Run migrate again to create those model tables in your database:

$ python manage.py migrate

* 1. To invoke the Python shell, use this command:

$ python manage.py shell

* 1. explore the [database API](https://docs.djangoproject.com/en/4.2/topics/db/queries/):

**>>> from** **polls.models** **import** Choice, Question *# Import the model classes we just wrote.*

# No questions are in the system yet.

**>>>** Question.objects.all()

<QuerySet []>

# Create a new Question.

# Support for time zones is enabled in the default settings file, so

# Django expects a datetime with tzinfo for pub\_date. Use timezone.now()

# instead of datetime.datetime.now() and it will do the right thing.

**>>> from** **django.utils** **import** timezone

**>>>** q = Question(question\_text="What's new?", pub\_date=timezone.now())

# Save the object into the database. You have to call save() explicitly.

**>>>** q.save()

# Now it has an ID.

**>>>** q.id

1

# Access model field values via Python attributes.

**>>>** q.question\_text

"What's new?"

**>>>** q.pub\_date

datetime.datetime(2012, 2, 26, 13, 0, 0, 775217, tzinfo=datetime.timezone.utc)

# Change values by changing the attributes, then calling save().

**>>>** q.question\_text = "What's up?"

**>>>** q.save()

# objects.all() displays all the questions in the database.

**>>>** Question.objects.all()

<QuerySet [<Question: Question object (1)>]>

**Ex.No: 12**

**Simple Web scrapper**

**Aim**

To scrape the IMDB website and extract some of its data into a JSON file.

**Procedure**

Step 1: Creating a Virtual Environment

First, install the virtualenv using the below command.

$ pip install virtualenv

shell

Now create a virtual environment with Python

$ virtualenv scrapyvenv

powershell

For Linux/Mac, you can mention the Python version.

$ virtualenv -p python3 scrapyvenv

shell

You can also mention which Python version you want to create the virtual environment.

After creating a virtual environment, activate it.

For Windows:

$ cd scrapyvenv

$ .\Scripts\activate

powershell

For Linux/Mac:

$ cd scrapyvenv

$ source bin/activate

shell

Step 2: Installing Scrapy

Most of the dependencies will automatically get installed. They're available for Python 2.7+.

* pip install: To install using pip, open the terminal and run the following command:

1$ pip install scrapy

shell

* conda Install: To install using conda, open the terminal and run the following command:

1$ conda install -c anaconda scrapy

shell

If you have a problem installing the twisted library, you can download it [here](https://www.lfd.uci.edu/~gohlke/pythonlibs/#twisted) and then install it locally.

Step 3: Creating a Scrapy Project

Since Scrapy is a framework, we need to follow some standards of the framework. To create a new project in scrapy, use the command startproject. I have named my project webscrapy.

1$ scrapy startproject webscrapy

shell

Moreover, this will create a webscrapy directory with the following contents:

1webscrapy

2├── scrapy.cfg -- deploy configuration file of scrapy project

3└── webscrapy -- your scrapy project module.

4 ├── \_\_init\_\_.py -- module initializer(empty file)

5 ├── items.py -- project item definition py file

6 ├── middlewares.py -- project middleware py file

7 ├── pipelines.py -- project pipeline py file

8 ├── settings.py -- project settings py file

9 └── spiders -- directory where spiders are kept

10 ├── \_\_init\_\_.py

docs

Create a Spider

Use the command genspider, which takes the name of spider and the URL it will crawl :

$ cd webscrapy

$ scrapy genspider imdb www.imdb.com

terminal

After running this command, Scrapy will automatically create a Python file named imdb in the spider folder.

When you open that spider imdb.py Python file, you will see a class named imdbSpider that inherits scrapy.Spider class and contains a method named parse.

import scrapy

class ImdbSpider(scrapy.Spider):

name = 'imdb'

allowed\_domains = ['www.imdb.com']

start\_urls = ['http://www.imdb.com/']

def parse(self, response):

pass

To run this spider, use the below command. Before running this command, make sure that you in the right directory.

$ scrapy crawl imdb

terminal

Scrape on IMDB

Create the spider imdb.py, which has been created earlier.

# importing the scrapy

import scrapy

class ImdbSpider(scrapy.Spider):

name = "imdb"

allowed\_domains = ["imdb.com"]

start\_urls = ['http://www.imdb.com/chart/top',]

def parse(self, response):

# table coloums of all the movies

columns = response.css('table[data-caller-name="chart-top250movie"] tbody[class="lister-list"] tr')

for col in columns:

# Get the required text from element.

yield {

"title": col.css("td[class='titleColumn'] a::text").extract\_first(),

"year": col.css("td[class='titleColumn'] span::text").extract\_first().strip("() "),

"rating": col.css("td[class='ratingColumn imdbRating'] strong::text").extract\_first(),

}

python

Run the above imdb spider:

$ scrapy crawl imdb

shell

You will get the following output:

1{'title': 'The Shawshank Redemption', 'year': '1994', 'rating': '9.2'}

2{'title': 'The Godfather', 'year': '1972', 'rating': '9.1'}

3...

4{'title': 'Swades', 'year': '2004', 'rating': '8.0'}

5{'title': 'Song of the Sea', 'year': '2014', 'rating': '8.0'}