**Full stack Development**

**Unit-3**

**What is Node.js?**

* Node.js is an open source server environment
* Node.js is free
* Node.js runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)
* Node.js uses JavaScript on the server

**Why Node.js?**

**Node.js uses asynchronous programming!**

A common task for a web server can be to open a file on the server and return the content to the client.

Here is how PHP or ASP handles a file request:

1. Sends the task to the computer's file system.
2. Waits while the file system opens and reads the file.
3. Returns the content to the client.
4. Ready to handle the next request.

Here is how Node.js handles a file request:

1. Sends the task to the computer's file system.
2. Ready to handle the next request.
3. When the file system has opened and read the file, the server returns the content to the client.

Node.js eliminates the waiting, and simply continues with the next request.

Node.js runs single-threaded, non-blocking, asynchronous programming, which is very memory efficient.

**What Can Node.js Do?**

* Node.js can generate dynamic page content
* Node.js can create, open, read, write, delete, and close files on the server
* Node.js can collect form data
* Node.js can add, delete, modify data in your database

**What is a Node.js File?**

* Node.js files contain tasks that will be executed on certain events
* A typical event is someone trying to access a port on the server
* Node.js files must be initiated on the server before having any effect
* Node.js files have extension ".js"

**Download Node.js**

* The official Node.js website has installation instructions for Node.js: [https://nodejs.org](https://nodejs.org/)

**Getting Started**

Once you have downloaded and installed Node.js on your computer, let's try to display "Hello World" in a web browser.

Create a Node.js file named "myfirst.js", and add the following code:

myfirst.js

var http = require('http');  
  
http.createServer(function (req, res) {  
  res.writeHead(200, {'Content-Type': 'text/html'});  
  res.end('Hello World!');  
}).listen(8080);

Save the file on your computer: C:\Users\*Your Name*\myfirst.js

The code tells the computer to write "Hello World!" if anyone (e.g. a web browser) tries to access your computer on port 8080.

**Command Line Interface**

Node.js files must be initiated in the "Command Line Interface" program of your computer.

How to open the command line interface on your computer depends on the operating system. For Windows users, press the start button and look for "Command Prompt", or simply write "cmd" in the search field.

Navigate to the folder that contains the file "myfirst.js", the command line interface

window should look something like this:

C:\Users\*Your Name*>\_

Initiate the Node.js File

The file you have just created must be initiated by Node.js before any action can take place.

Start your command line interface, write node myfirst.js and hit enter:

Initiate "myfirst.js":

C:\Users\*Your Name*>node myfirst.js

Now, your computer works as a server!

If anyone tries to access your computer on port 8080, they will get a "Hello World!" message in return!

Start your internet browser, and type in the address: [http://localhost:8080](http://localhost:8080/)

**Node.js Modules**

What is a Module in Node.js?

Consider modules to be the same as JavaScript libraries.

A set of functions you want to include in your application.

**Built-in Modules**

Node.js has a set of built-in modules which you can use without any further installation.

Look at our [Built-in Modules Reference](https://nodejs.org/api/modules.html#modules-commonjs-modules) for a complete list of modules.

Include Modules

To include a module, use the require() function with the name of the module:

var http = require('http');

Now your application has access to the HTTP module, and is able to create a server:

http.createServer(function (req, res) {  
  res.writeHead(200, {'Content-Type': 'text/html'});  
  res.end('Hello World!');  
}).listen(8080);

Create Your Own Modules

You can create your own modules, and easily include them in your applications.

The following example creates a module that returns a date and time object:

Create a module that returns the current date and time:

exports.myDateTime = function () {  
  return Date();  
};

Use the exports keyword to make properties and methods available outside the module file.

Save the code above in a file called "myfirstmodule.js"

Include Your Own Module

Now you can include and use the module in any of your Node.js files.

Example

Use the module "myfirstmodule" in a Node.js file:

var http = require('http');  
**var dt = require('./myfirstmodule');**  
http.createServer(function (req, res) {  
  res.writeHead(200, {'Content-Type': 'text/html'});  
  res.write("The date and time are currently: " + **dt.myDateTime()**);  
  res.end();  
}).listen(8080);

Notice that we use ./ to locate the module, that means that the module is located in the same folder as the Node.js file.

Save the code above in a file called "demo\_module.js", and initiate the file:

Initiate demo\_module.js:

C:\Users\*Your Name*>node demo\_module.js

If you have followed the same steps on your computer, you will see the same result as the example: [http://localhost:8080](http://localhost:8080/)

**Node.js Events**

Node.js is perfect for event-driven applications.

Events in Node.js

Every action on a computer is an event. Like when a connection is made or a file is opened.

Objects in Node.js can fire events, like the readStream object fires events when opening and closing a file:

Example

var fs = require('fs');  
var rs = fs.createReadStream('./demofile.txt');  
rs.on('open', function () {  
  console.log('The file is open');  
});

Events Module

Node.js has a built-in module, called "Events", where you can create-, fire-, and listen for- your own events.

To include the built-in Events module use the require() method. In addition, all event properties and methods are an instance of an EventEmitter object. To be able to access these properties and methods, create an EventEmitter object:

var events = require('events');  
var eventEmitter = new events.EventEmitter();

The EventEmitter Object

You can assign event handlers to your own events with the EventEmitter object.

In the example below we have created a function that will be executed when a "scream" event is fired.

To fire an event, use the emit() method.

Example

var events = require('events');  
var eventEmitter = new events.EventEmitter();  
  
//Create an event handler:  
var myEventHandler = function () {  
  console.log('I hear a scream!');  
}  
  
//Assign the event handler to an event:  
eventEmitter.on('scream', myEventHandler);  
  
//Fire the 'scream' event:  
eventEmitter.emit('scream');

**What is Express.js?**

Express.js is a **fast, minimal, and flexible** web application framework for **Node.js**. It simplifies backend development by providing a set of robust features to build web applications and APIs.

**🔹 Why Use Express.js?**

✅ **Minimal & Lightweight** – Provides core features without unnecessary complexity  
✅ **Fast Performance** – Optimized for handling requests efficiently  
✅ **Middleware Support** – Allows adding custom functions for request handling  
✅ **Routing System** – Simplifies defining API endpoints  
✅ **Easy Integration** – Works well with databases like MongoDB, PostgreSQL, and MySQL  
✅ **Huge Community** – Widely used, with extensive documentation and support

**1️ Setting Up Express.js**

**Step 1: Install Node.js**

Download and install [**Node.js**](https://nodejs.org/) (includes npm).

**Step 2: Initialize a Project**

Open a terminal and run:

mkdir my-express-app && cd my-express-app

npm init -y # Creates a package.json file

**Step 3: Install Express.js**

npm install express

**2️ Creating a Simple Express Server**

Create a file **server.js** and add:

const express = require("express");

const app = express();

// Define a route

app.get("/", (req, res) => {

res.send("Hello, Express!");

});

// Start the server

const PORT = 3000;

app.listen(PORT, () => {

console.log(`Server running at http://localhost:${PORT}`);

});

**Run the Server**

node server.js

Now, visit **http://localhost:3000/** in your browser to see **"Hello, Express!"**. 🚀

**3️ Understanding Express.js Features**

**📌 Routing System**

Express makes handling different **URL routes** easy:

javascript

app.get("/about", (req, res) => {

res.send("About Page");

});

app.post("/submit", (req, res) => {

res.send("Form Submitted");

});

**📌 Middleware (Processing Requests)**

Middleware functions process requests **before sending a response**. Example:

javascript

CopyEdit

app.use((req, res, next) => {

console.log(`Request: ${req.method} ${req.url}`);

next(); // Pass control to the next middleware

});

**📌 Handling JSON & Form Data**

app.use(express.json()); // Parses JSON requests

app.use(express.urlencoded({ extended: true })); // Parses form data

**📌 Static File Serving**

To serve static files (CSS, images, etc.), use:

app.use(express.static("public"));

**4️ Creating a REST API with Express.js**

const users = [{ id: 1, name: "Alice" }, { id: 2, name: "Bob" }];

// GET all users

app.get("/users", (req, res) => {

res.json(users);

});

// GET user by ID

app.get("/users/:id", (req, res) => {

const user = users.find((u) => u.id == req.params.id);

user ? res.json(user) : res.status(404).send("User not found");

});

// POST new user

app.post("/users", (req, res) => {

const newUser = { id: users.length + 1, name: req.body.name };

users.push(newUser);

res.status(201).json(newUser);

});

**Node.js database Access**

Node.js can interact with various databases, including **SQL-based (MySQL, PostgreSQL, SQLite)** and **NoSQL-based (MongoDB, Firebase, Redis)** databases. The choice depends on project needs.

**1️ Types of Databases in Node.js**

**🔹 SQL Databases (Structured Data)**

* **MySQL** – Most popular open-source relational database
* **PostgreSQL** – Powerful, feature-rich database
* **SQLite** – Lightweight, serverless database
* **MSSQL (Microsoft SQL Server)** – Enterprise-grade database

**🔹 NoSQL Databases (Unstructured Data)**

* **MongoDB** – Document-based (JSON-like) database
* **Redis** – Key-value store, fast caching
* **Firebase Firestore** – Cloud-based NoSQL database

Node.js can be used in database applications.

One of the most popular NoSQL database is MongoDB.

Node.js can use this module to manipulate MongoDB databases:

var mongo = require('mongodb');

**Creating a Database**

To create a database in MongoDB, start by creating a MongoClient object, then specify a connection URL with the correct ip address and the name of the database you want to create.

MongoDB will create the database if it does not exist, and make a connection to it.

Create a database called "mydb":

var MongoClient = require('mongodb').MongoClient;  
var url = "mongodb://localhost:27017/mydb";  
  
MongoClient.connect(url, function(err, db) {  
  if (err) throw err;  
  console.log("Database created!");  
  db.close();  
});

Save the code above in a file called "demo\_create\_mongo\_db.js" and run the file:

Run "demo\_create\_mongo\_db.js"

C:\Users\*Your Name*>node demo\_create\_mongo\_db.js

Which will give you this result:

Database created!

**Important:** In MongoDB, a database is not created until it gets content!

MongoDB waits until you have created a collection (table), with at least one document (record) before it actually creates the database (and collection).

**Node.js MongoDB Create Collection**

**A collection in MongoDB is the same as a table in MySQL**

**Creating a Collection**

To create a collection in MongoDB, use the createCollection() method:

Create a collection called "customers":

var MongoClient = require('mongodb').MongoClient;  
var url = "mongodb://localhost:27017/";  
  
MongoClient.connect(url, function(err, db) {  
  if (err) throw err;  
  var dbo = db.db("mydb");  
  dbo.createCollection("customers", function(err, res) {  
    if (err) throw err;  
    console.log("Collection created!");  
    db.close();  
  });  
});

Save the code above in a file called "demo\_mongodb\_createcollection.js" and run the file:

Run "demo\_mongodb\_createcollection.js"

C:\Users\*Your Name*>node demo\_mongodb\_createcollection.js

Which will give you this result:

Collection created!

**Important:** In MongoDB, a collection is not created until it gets content!

MongoDB waits until you have inserted a document before it actually creates the collection.

**Node.js MongoDB Insert**

Insert Into Collection

To insert a record, or *document* as it is called in MongoDB, into a collection, we use the insertOne() method.

A **document** in MongoDB is the same as a **record** in MySQL

The first parameter of the insertOne() method is an object containing the name(s) and value(s) of each field in the document you want to insert.

It also takes a callback function where you can work with any errors, or the result of the insertion:

Insert a document in the "customers" collection:

var MongoClient = require('mongodb').MongoClient;  
var url = "mongodb://localhost:27017/";  
  
MongoClient.connect(url, function(err, db) {  
  if (err) throw err;  
  var dbo = db.db("mydb");  
  var myobj = { name: "Company Inc", address: "Highway 37" };  
  dbo.collection("customers").insertOne(myobj, function(err, res) {  
    if (err) throw err;  
    console.log("1 document inserted");  
    db.close();  
  });  
});

Save the code above in a file called "demo\_mongodb\_insert.js" and run the file:

Run "demo\_mongodb\_insert.js"

C:\Users\*Your Name*>node demo\_mongodb\_insert.js

Which will give you this result:

1 document inserted

**Note:** If you try to insert documents in a collection that do not exist, MongoDB will create the collection automatically.

**Insert Multiple Documents**

To insert multiple documents into a collection in MongoDB, we use the insertMany() method.

The first parameter of the insertMany() method is an array of objects, containing the data you want to insert.

It also takes a callback function where you can work with any errors, or the result of the insertion:

Example

Insert multiple documents in the "customers" collection:

var MongoClient = require('mongodb').MongoClient;  
var url = "mongodb://localhost:27017/";  
  
MongoClient.connect(url, function(err, db) {  
  if (err) throw err;  
  var dbo = db.db("mydb");  
  var myobj = [  
    { name: 'John', address: 'Highway 71'},  
    { name: 'Peter', address: 'Lowstreet 4'},  
    { name: 'Amy', address: 'Apple st 652'},  
    { name: 'Hannah', address: 'Mountain 21'},  
    { name: 'Michael', address: 'Valley 345'},  
    { name: 'Sandy', address: 'Ocean blvd 2'},  
    { name: 'Betty', address: 'Green Grass 1'},  
    { name: 'Richard', address: 'Sky st 331'},  
    { name: 'Susan', address: 'One way 98'},  
    { name: 'Vicky', address: 'Yellow Garden 2'},  
    { name: 'Ben', address: 'Park Lane 38'},  
    { name: 'William', address: 'Central st 954'},  
    { name: 'Chuck', address: 'Main Road 989'},  
    { name: 'Viola', address: 'Sideway 1633'}  
  ];  
  dbo.collection("customers").insertMany(myobj, function(err, res) {  
    if (err) throw err;  
    console.log("Number of documents inserted: " + res.insertedCount);  
    db.close();  
  });  
});

Save the code above in a file called "demo\_mongodb\_insert\_multiple.js" and run the file:

Run "demo\_mongodb\_insert\_multiple.js"

C:\Users\*Your Name*>node demo\_mongodb\_insert\_multiple.js

Which will give you this result:

Number of documents inserted: 14

**AngularJS – Introduction**

**🔹 What is AngularJS?**

AngularJS is a **JavaScript framework** developed by **Google** for building **dynamic, single-page web applications (SPAs)**. It extends HTML with **two-way data binding** and a powerful **MVC (Model-View-Controller)** architecture.

**1️ Key Features of AngularJS**

✅ **Two-Way Data Binding** – Automatically syncs data between model and view.  
✅ **MVC Architecture** – Separates business logic (Model), UI (View), and control logic (Controller).  
✅ **Directives** – Custom HTML attributes (e.g., ng-model, ng-repeat).  
✅ **Dependency Injection (DI)** – Manages component dependencies efficiently.  
✅ **Single-Page Application (SPA) Support** – Handles routing without full page reloads.  
✅ **Templates & Filters** – Enhances HTML with dynamic content.

**2️ Setting Up AngularJS**

**Option 1: Use CDN**

Add this in an HTML file:

html

CopyEdit

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>

**Option 2: Install via npm**

npm install angular

**3️ Basic AngularJS Example**

<!DOCTYPE html>

<html lang="en" ng-app="myApp">

<head>

<title>AngularJS Example</title>

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>

</head>

<body>

<div ng-controller="MainController">

<h1>Hello, {{ name }}!</h1>

<input type="text" ng-model="name">

</div>

<script>

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

app.controller("MainController", function($scope) {

$scope.name = "AngularJS";

});

</script>

</body>

</html>

🔹 **Explanation:**

* ng-app="myApp" – Defines the AngularJS application.
* ng-controller="MainController" – Binds the controller to the view.
* ng-model="name" – Binds input field to the name variable dynamically.

**4️ AngularJS Directives**

Directives are special HTML attributes that add behavior to elements.

| **Directive** | **Function** |
| --- | --- |
| ng-app | Initializes an AngularJS app |
| ng-model | Binds input fields to variables |
| ng-repeat | Loops through arrays (similar to forEach) |
| ng-if, ng-show, ng-hide | Conditional rendering |

**Example – ng-repeat Directive**

html

<ul>

<li ng-repeat="fruit in fruits">{{ fruit }}</li>

</ul>

<script>

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

app.controller("FruitController", function($scope) {

$scope.fruits = ["Apple", "Banana", "Cherry"];

});

</script>

**Is AngularJS Still Used? (Alternatives)**

AngularJS is now considered **deprecated** (since 2022) and has been replaced by **Angular (2+)**, which is more modern and performance-optimized.

**🔹 Alternatives to AngularJS**

✔ **Angular (2+)** – Successor of AngularJS, using TypeScript  
✔ **React.js** – Component-based UI library by Facebook  
✔ **Vue.js** – Lightweight and easy-to-learn framework

**AngularJS Directives**

**🔹 What are AngularJS Directives?**

Directives in AngularJS are special attributes or elements that **extend HTML functionality** by adding dynamic behavior. They help in creating reusable UI components and manipulating the DOM efficiently.

**1️Types of AngularJS Directives**

| **Directive** | **Function** |
| --- | --- |
| ng-app | Defines an AngularJS application |
| ng-model | Binds input fields to variables (two-way data binding) |
| ng-bind | Binds data to an HTML element |
| ng-repeat | Loops through arrays (for dynamic lists) |
| ng-if, ng-show, ng-hide | Controls element visibility |
| ng-click, ng-dblclick | Handles click events |
| ng-style, ng-class | Dynamically applies CSS styles and classes |
| ng-src, ng-href | Dynamically sets image sources and links |
| **Custom Directives** | Allows creating reusable components |

**2️Basic AngularJS Directives**

**🔹 ng-model (Two-Way Data Binding)**

<div ng-app="myApp" ng-controller="MainController">

<input type="text" ng-model="name">

<p>Hello, {{ name }}!</p>

</div>

<script>

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

app.controller("MainController", function($scope) {

$scope.name = "AngularJS";

});

</script>

✅ **Two-way data binding** updates name instantly as the user types.

**🔹 ng-repeat (Looping Through an Array)**

<ul>

<li ng-repeat="fruit in fruits">{{ fruit }}</li>

</ul>

<script>

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

app.controller("FruitController", function($scope) {

$scope.fruits = ["Apple", "Banana", "Cherry"];

});

</script>

✅ Loops through an array and dynamically generates list items.

**🔹 ng-if, ng-show, ng-hide (Conditional Rendering)**

<button ng-click="isVisible = !isVisible">Toggle</button>

<p ng-if="isVisible">This text appears when isVisible is true.</p>

<p ng-show="isVisible">Visible when true.</p>

<p ng-hide="isVisible">Hidden when true.</p>

✅ ng-if removes the element from the DOM if false.  
✅ ng-show & ng-hide toggle element visibility using display: none.

**🔹 ng-click (Handling Events)**

<button ng-click="count = count + 1">Click Me</button>

<p>Clicked {{ count }} times</p>

<script>

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

app.controller("ClickController", function($scope) {

$scope.count = 0;

});

</script>

✅ Increments the count every time the button is clicked.

**3️ Custom AngularJS Directives**

You can create your own directives using **app.directive()** to build reusable components.

**🔹 Creating a Simple Custom Directive**

<div ng-app="myApp">

<custom-message></custom-message>

</div>

<script>

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

app.directive("customMessage", function() {

return {

template: "<h3>This is a custom directive!</h3>"

};

});

</script>

✅ The <custom-message></custom-message> element gets replaced with the custom template.

**🔹 Custom Directive with Scope & Attributes**

<div ng-app="myApp">

<user-card name="Alice" email="alice@example.com"></user-card>

</div>

<script>

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

app.directive("userCard", function() {

return {

scope: {

name: "@",

email: "@"

},

template: "<div><h3>{{ name }}</h3><p>{{ email }}</p></div>"

};

});

</script>

✅ Attributes like name="Alice" pass values dynamically into the directive.

**4️ When to Use Directives?**

✔ When creating **reusable UI components** (like buttons, cards, or forms).  
✔ When manipulating the **DOM dynamically**.  
✔ When reducing **code duplication** across the application.

**AngularJS Controller**

**🔹 What is an AngularJS Controller?**

An **AngularJS Controller** is a JavaScript function that manages application data and business logic. It acts as the bridge between the **View (HTML)** and the **Model (Data)**.

📌 **Key Responsibilities of a Controller:**  
✅ Controls application logic.  
✅ Interacts with the View via **$scope**.  
✅ Fetches, processes, and updates data.

**1️ Creating a Simple AngularJS Controller**

<!DOCTYPE html>

<html lang="en" ng-app="myApp">

<head>

<title>AngularJS Controller Example</title>

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>

</head>

<body>

<div ng-controller="MainController">

<h2>Hello, {{ name }}!</h2>

<input type="text" ng-model="name">

</div>

<script>

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

app.controller("MainController", function($scope) {

$scope.name = "AngularJS"; // Default value

});

</script>

</body>

</html>

**🔹 How it Works:**

1. **ng-controller="MainController"** – Links the HTML to the controller.
2. **$scope.name = "AngularJS"** – Defines a variable inside the controller.
3. **ng-model="name"** – Two-way binds the input field to the $scope.name.

**AngularJS Form Validation**

**🔹 What is AngularJS Form Validation?**

AngularJS provides built-in form validation features that help in validating **user input** before submitting a form.

✅ Uses **ng-model** to bind input fields to variables.  
✅ Uses **ng-required**, **ng-minlength**, and **ng-pattern** for validation.  
✅ Uses **$valid**, **$invalid**, and **$dirty** to check form states.

**1️ Basic Form Validation Example**

<!DOCTYPE html>

<html lang="en" ng-app="myApp">

<head>

<title>AngularJS Form Validation</title>

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>

</head>

<body ng-controller="FormController">

<form name="userForm">

<label>Name:</label>

<input type="text" name="name" ng-model="user.name" ng-required="true">

<span ng-show="userForm.name.$touched && userForm.name.$invalid">Name is required</span>

<br>

<label>Email:</label>

<input type="email" name="email" ng-model="user.email" ng-required="true">

<span ng-show="userForm.email.$touched && userForm.email.$invalid">Enter a valid email</span>

<br>

<label>Password:</label>

<input type="password" name="password" ng-model="user.password" ng-minlength="6">

<span ng-show="userForm.password.$touched && userForm.password.$error.minlength">Minimum 6 characters</span>

<br>

<button type="submit" ng-disabled="userForm.$invalid">Submit</button>

</form>

<script>

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

app.controller("FormController", function($scope) {

$scope.user = {};

});

</script>

</body>

</html>

**2️ Key AngularJS Form Validation Directives**

| **Directive** | **Description** |
| --- | --- |
| ng-required | Makes a field required. |
| ng-minlength="x" | Sets a minimum length for input. |
| ng-maxlength="x" | Sets a maximum length for input. |
| ng-pattern="regex" | Ensures input matches a pattern. |
| ng-model | Binds input to a variable. |
| name="fieldName" | Assigns a name to the input for validation. |

**3️ Checking Validation Status ($valid, $invalid, $dirty)**

| **Form State** | **Description** |
| --- | --- |
| $valid | True if the form is valid. |
| $invalid | True if the form is invalid. |
| $dirty | True if a field has been modified. |
| $pristine | True if a field is untouched. |

**4️ Advanced Validation with ng-pattern**

**🔹 Phone Number Validation**

html

CopyEdit

<label>Phone:</label>

<input type="text" name="phone" ng-model="user.phone" ng-pattern="/^\d{10}$/" required>

<span ng-show="userForm.phone.$error.pattern">Enter a valid 10-digit phone number</span>

✅ Uses **ng-pattern** to enforce a **10-digit number** rule.

**5️ Disabling Submit Button Until Form is Valid**

html

CopyEdit

<button type="submit" ng-disabled="userForm.$invalid">Submit</button>

✅ The button stays **disabled** until the form is **completely valid**.

**6️ Highlighting Errors with CSS**

**🔹 Adding Custom Styles**

css

CopyEdit

input.ng-invalid.ng-touched {

border: 2px solid red;

}

✅ **Red border** appears if the field is **touched but invalid**.

**7️ When to Use AngularJS Form Validation?**

✔ When validating **required fields**.  
✔ When ensuring **correct email or phone number formats**.  
✔ When **disabling submit** until inputs are valid.

**AngularJS Filters**

**🔹 What are AngularJS Filters?**

Filters in AngularJS **modify data** before displaying it in the view. They are commonly used for **formatting strings, numbers, dates, and arrays**.

✅ Filters can be used in **expressions ({{ }})** and **directives (ng-repeat)**.  
✅ Filters are **chained** to apply multiple transformations.

**1️Using Filters in Expressions**

Syntax:

html

CopyEdit

{{ expression | filterName }}

**Example: Using the uppercase Filter**

html

CopyEdit

<p>{{ "hello world" | uppercase }}</p> <!-- Output: HELLO WORLD -->

✔ Converts the text to uppercase.

**2️ Common Built-in AngularJS Filters**

| **Filter** | **Description** | **Example** |
| --- | --- | --- |
| uppercase | Converts text to uppercase | "hello" → "HELLO" |
| lowercase | Converts text to lowercase | "Hello" → "hello" |
| currency | Formats numbers as currency | 1234 → $1,234.00 |
| date | Formats a date | new Date() → Jan 1, 2024 |
| limitTo | Limits array length | limitTo:3 (first 3 items) |
| orderBy | Sorts an array | orderBy:'name' |

**3️ Using Filters in ng-repeat**

**🔹 orderBy Filter (Sorting an Array)**

html

CopyEdit

<ul>

<li ng-repeat="user in users | orderBy:'name'">

{{ user.name }} - {{ user.age }}

</li>

</ul>

<script>

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

app.controller("UserController", function($scope) {

$scope.users = [

{ name: "Alice", age: 25 },

{ name: "Bob", age: 22 },

{ name: "Charlie", age: 30 }

];

});

</script>

✅ Users are sorted alphabetically by name.

**AngularJS Module**

**🔹 What is an AngularJS Module?**

An **AngularJS module** is a container for different parts of an AngularJS application, such as **controllers, directives, filters, services, and routes**.

✅ Acts as the **main application container**.  
✅ Helps in **organizing** code into reusable blocks.  
✅ Uses **angular.module()** to create a module.

**1️ Creating an AngularJS Module**

**Basic Example**

<!DOCTYPE html>

<html lang="en" ng-app="myApp">

<head>

<title>AngularJS Module Example</title>

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>

</head>

<body ng-controller="MainController">

<h2>Hello, {{ name }}!</h2>

</body>

<script>

// Creating an AngularJS module

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

// Creating a controller inside the module

app.controller("MainController", function($scope) {

$scope.name = "AngularJS";

});

</script>

</html>

✔ The **myApp** module is created using angular.module("myApp", []).  
✔ The **controller** is attached to the module.  
✔ The **ng-app="myApp"** directive links the HTML to the module.

**When to Use AngularJS Modules?**

✔ When **organizing** different parts of an application.  
✔ When **creating reusable components** (services, directives, etc.).  
✔ When **structuring large applications** into smaller modules.