MEAN STACK

Introduction to MEAN Stack

MEAN Stack is one of the most popular Technology Stack. It is used to develop a Full Stack Web Application. Although it is a Stack of different technologies, all of these are based on JavaScript language.

Mean Stack refers to a collection of JavaScript technologies used to develop web applications. Therefore, from the client to the server and from server to database, everything is based on JavaScript. MEAN is a full-stack development toolkit used to develop a fast and robust web applications.

This free and open-source stack offers a quick and organized method for creating rapid prototypes for web-based applications.

MEAN Stands for:

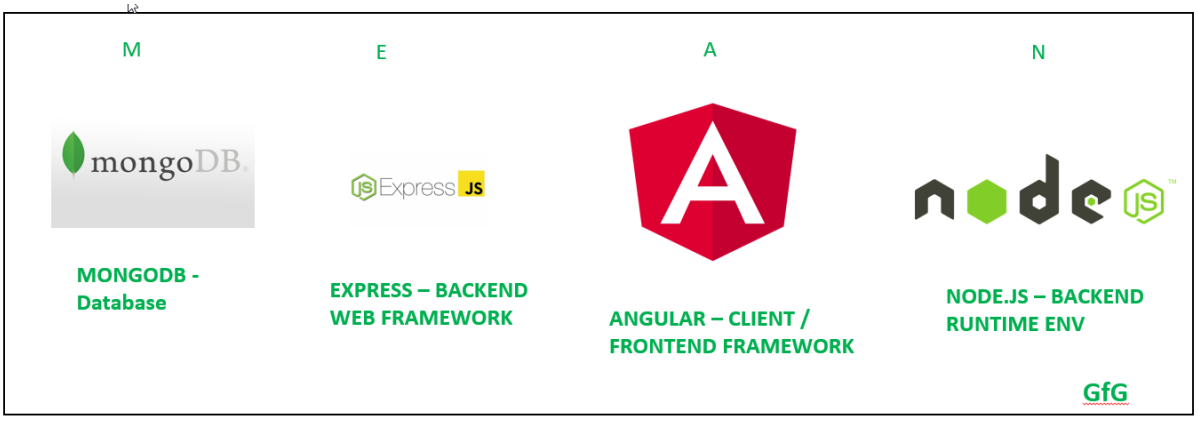
M – MongoDB: a schemaless NoSQL database system

E – Express: a framework used to build web applications in Node. It Make requests to Database and return a response

A – Angular: a JavaScript framework developed by Google. It Accept requests and display results to end user

N – Node.js: a server-side JavaScript execution environment. It Handle Client and Server Requests

This stack leads to faster development as well as the deployment of the Web Application. Angular is Frontend Development Framework whereas Node.js, Express, and MongoDB are used for Backend development as shown in the below figure.



Component of Mean Stack

Node.js

Node.js is an open-source platform and provides a runtime environment for executing the javascript code. It is mainly used for building the back-end application. Since there are two types of apps, such as web apps and mobile apps, where web apps run on the browser and mobile apps run on mobile devices. Both web app and mobile app are the client apps to which the user interacts. These apps require to communicate with the backend services to store the data, send emails, push notifications. Node.js is an ideal platform to build highly scalable, data-intensive, and real-time applications. It can be used for agile development and highly-scalable services. For example, PayPal is a java and spring-based application using Node.js.

Advantages of Node.js

- The node.js applications are faster than the other framework-based applications and require fewer people to build the app.
- It requires fewer lines of code.
- The Node app has a 35% faster response time than the other apps.
- The major advantage of using node.js is that node.js uses javascript. If you are a front-end developer, then you can easily transit from the front-end to the full stack developer.

Angular.js

Angular.js is a JavaScript framework that is used to develop web applications. This framework is developed by the Google. Now, the question arises that there are many javascript frameworks available in the market. Why do we prefer angular.js over the other frameworks for developing the web application?.

Advantages of Angular.js

- It is a two-way data binding which means that it keeps the model and view in sync. If any changes are made in the model, then automatically view will also be updated accordingly.
- The Angular.js is designed with testing in mind. The components of angular.js application can be tested with both the testing, such as unit testing and end to end testing.
- With the help of Angular.js, it is easy to develop the application in an MVC architecture.

MongoDB

MongoDB is the database used in web development. It is a NoSQL database, and a NoSQL database can be defined as a non-relational and document-oriented database management system. As it is a document-oriented database management system, so it stores the data in the form of documents. The SQL databases use SQL query language to query the database, whereas the MongoDB is a NoSQL database that uses BSON language to query the database. JSON is a text-based format, but it is inefficient in terms of speed and space. In order to make

MongoDB efficient in terms of space and speed, BSON was invented. BSON basically stores the JSON format in the binary form that optimizes the space, speed, and complexity.

Since the MongoDB is an unstructured schema and the data is not related to each other then the question arises 'why do we need to use the MongoDB?'. MongoDB is mainly useful for projects which are huge.

Express.js

Express.js is a free and open-source software used as a back-end web application framework. It is commonly used in the popular development stacks like MEAN with a MongoDB database. The Express.js was developed by TJ Holowaychuk.

Advantages of Express.js

- It is simple and lightweight software. It is not heavy to get installed in the machine and make the application running.
- It is easy to customize and configure as we can see it provides the flexibility that we require.
- It is a better choice for creating the API as when the application requires various APIs to communicate with different people then the Express.js is a good option

Advantages of Mean Stack

Here, are some most prominent reasons for using Mean Stack technology

- Allows creating a simple open source solution which can be used to build robust and maintainable solutions.
- Helps in rapid development of applications
- MEAN is full stack JavaScript which is 100% free. Leverage JavaScript's popularity
- Use a uniform language throughout your stack
- Uses very low memory footprint/overhead
- Helps you to avoid unnecessary groundwork and keeps your application organized
- MongoDB is built for the cloud
- Node.js simplifies the server layer
- MEAN makes code isomorphic

Disadvantages of Mean Stack

- MongoDB may be an ideal choice for small to the mid-sized application. However, it is not the best option for large-scale applications
- There are no specific general JS coding guidelines
- Once you have developed the first site using Mean stack technology, it's really hard to go back to the old approach
- It offers poor isolation of server from business logic
- You could potentially lose records

Mean Stack Architecture

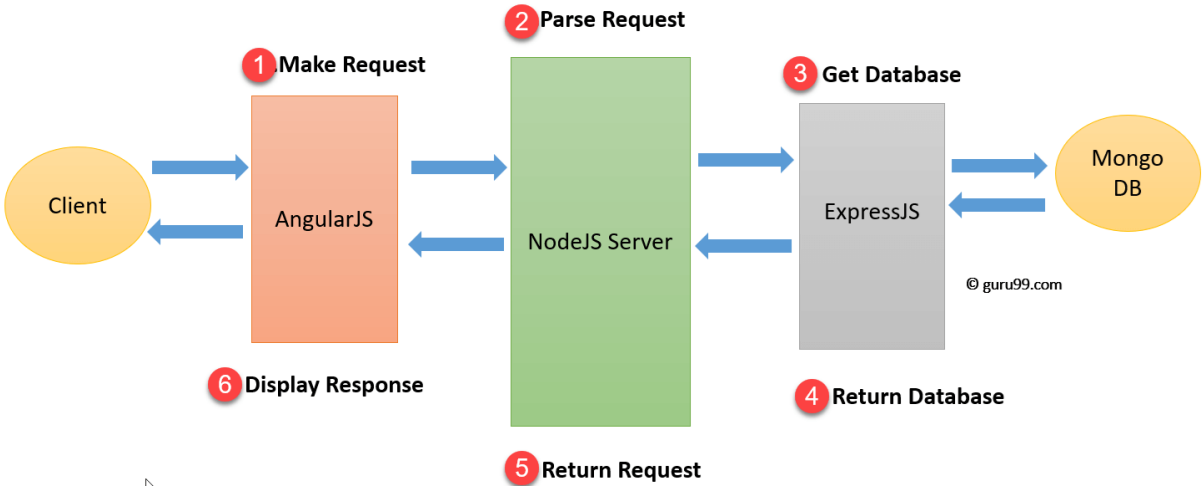
The primary function of various components of Mean Stack Architecture are as follows:

Angular JS: Accept requests and display results to end user

NodeJS: Handle Client and Server Requests

Express JS: Make requests to Database and return a response

MongoDB: Store and retrieve data.



1. Firstly, the client makes a request which is processed by the AngularJS
2. After that, the request moves to NodeJS which will parse the request.
3. Express.Js will make calls to MongoDB to get or set data.
4. MongoDB will retrieve the requested data and return that request to the Express JS
5. NodeJS will return the request to the client.
6. At the client side, AngularJS to display the result fetched from MongoDB.

Features of MEAN stack architecture

These are the following features of MEAN stack architecture:

- One of the most important features of the MEAN stack architecture is that the developer writes the entire code from the client to the server in JavaScript.
- MEAN stack architecture supports the MVC, i.e., Model View Controller architecture.
- The MEAN components are free and open-source.
- It is flexible to understand and easy to use.
- It helps the developers to customize as per the requirement.
- It uses JSON for data transferring and has a massive module library of node.js.

How secure is the MEAN stack?

- Always use HTTPS
- Preventing XSS & Request Forgery
- Preventing SQL Injections
- Security in Web API authentication

Why choose Mean Stack?

- Scalable and Flexible
- Excellent Speed
- One Stack, one language
- Free of Cost
- User-Friendly
- Avoids Rewriting

AngularJS

AngularJS is an open-source Model-View-Controller framework (MVC) that is similar to the JavaScript framework. AngularJS is probably one of the most popular modern-day web frameworks available today. It is a Javascript open-source front-end structural framework that is mainly used to develop single-page web applications(SPAs). This framework has been developed by a group of developers from Google itself.
It can be added to an HTML page with a <script> tag. It enables us to create single-page applications that only require HTML, CSS, and JavaScript on the client side.
AngularJS extends HTML attributes with Directives and Data binding to HTML with Expressions.

Angular Version History

- Google developed this web application framework in 2009. It is officially called AngularJS. Some people call this version Angular 1.0. This version came out on October 20, 2010.
- After Angular 1.0, in 2016, Angular 2 was released. It was written from scratch and is fully different from Angular 1 or JS.
- The third update, Angular 4, was launched in the year 2017. It is not a complete rewrite of the original version. Instead, it is the updated version of Angular 2.
- Angular 5 was released on 1st November 2017. The updates in this version help developers to create apps fast, as it removes unnecessary codes.
- Angular 6 was released on 3rd May 2018, and the version of Angular 7 was out in October 2018.
- Angular Team released Angular 8 on May 28, 2019. It features differential loading of all application code, web workers, supports TypeScript 3.4, dynamic imports for lazy routes, and Angular Ivy as an opt-in preview.
- Angular Team released Angular 9 on Feb 06, 2020. By default, version 9 moves all the applications to use the Ivy compiler and runtime.
- Angular 10 was released on Jun 24, 2020, with four months difference from the previous release.
- The latest angular version from Google is Angular 11. It was made available in the market from 14th November 2020.

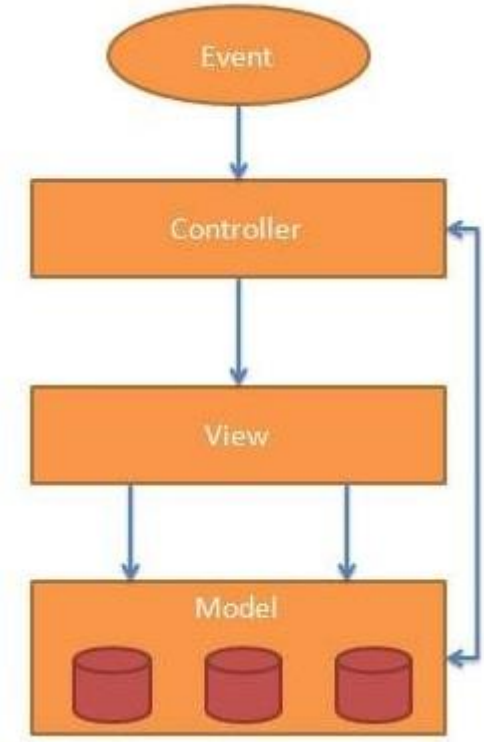
Why use AngularJS?

- **Easy to work with:** All you need to know to work with AngularJS is the basics of HTML, CSS, and Javascript, not necessary to be an expert in these technologies.

- **Time-saving:** AngularJs allows us to work with components and hence we can use them again which saves time and unnecessary code.
- **Ready to use a template:** AngularJs is mainly plain HTML, and it mainly makes use of the plain HTML template and passes it to the DOM and then the AngularJS compiler. It traverses the templates and then they are ready to use.
- **Directives:** AngularJS's directives allow you to extend HTML with custom elements and attributes. This enables you to create reusable components and define custom behaviors for your application. Directives make it easier to manipulate the DOM, handle events, and encapsulate complex UI logic within a single component.

AngularJS MVC Architecture

MVC stands for Model View Controller. It is a software design pattern for developing web applications. It is very popular because it isolates the application logic from the user interface layer and supports separation of concerns.



The MVC pattern is made up of the following three parts:

Let us understand the purpose of each component of MVC serve:

Model

The model basically helps us carry and manage the data or any logic or application rules. The model consists of databases and application data. It is responsible for maintaining the data. So, for example, you have two views, and you want to transfer some data to the other or maybe to some controller or some other source to your view; in that situation, we can go for the model that can bring the data to our UI.

View

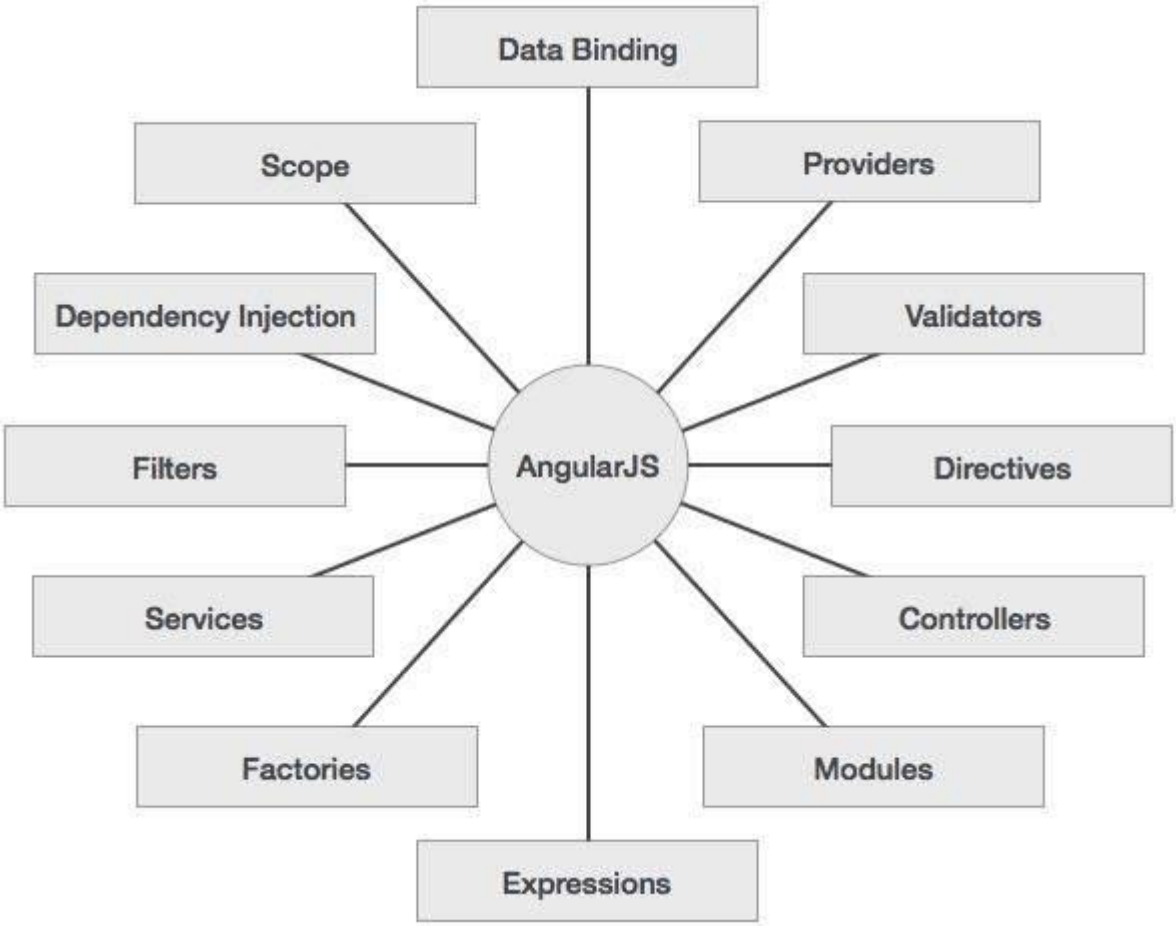
The view is simply the user interface, meaning whatever a user can see is a view that can be an HTML page, a graph, or any representational data. The view is the user interface part of an application. It is responsible for displaying data to the user.

Controller

The controller is responsible for controlling the interaction between the model and view. Accepts input and converts it to commands for the model or view. The user starts the execution. This user will send a request to the controller, which will get accepted by the controller. Then as per the request, it gathers the data and manipulates the model. Now the view receives the same data that the model is writing. The view is just an HTML template representing the view in Angular JS.

Concepts of AngularJS

Few things you need to know to before starting with AngularJS:



Modules - A module can be defined as a container that consists of various application parts. The module is a set of functions defined in a JavaScript file. Module divides an application into small and reusable components.

Directives -Directives indicate the compiler to associate a behavior to the DOM element or modify it. Angular JS contains several directives such as ng-app, ng-controller, ng-view, ng-if, etc.

Expressions -Angular JS expressions are expressed with {{ }} which indicate a data binding in HTML. These expressions can be added into the HTML templates. Expressions do not support control flow statements while support the filters.

Controller- It is a JavaScript object constructor function that controls the AngularJS applications.

Scope- It is a JavaScript object that acts as a bridge between the Controller and the View. It is the source of data in AngularJS. Each data manipulation and assignment takes place with the help of the Scope object.

Data Binding- It coordinates the model and views any changes in either of these two.

Validations- Validations take place with the help of AngularJS forms and controls.

Filters -These let you display the formatting of data on DOM in Angular and extend the behavior of directives and binding expressions. Filters format the values or apply specific.

Services -These are singletons that are used by directives, controllers, or other services.

Routing -The service $routeProvider handles the operations of Routing. It divides the map into various views. It helps split the Single Page Applications into different views.

Dependency Injection-It is a design pattern used to handle the dependencies of various components of a software. It lets you develop loosely-structured architectures.

Testing -The codes developed by Dependency Injections are tested. Some of the popular testing frameworks like Jasmine and Karma are two widely-used technologies.

Advantages of AngularJS

- It provides the capability to create Single Page Application in a very clean and maintainable way.
- It provides data binding capability to HTML. Thus, it gives user a rich and responsive experience.
- AngularJS code is unit testable.
- AngularJS uses dependency injection and make use of separation of concerns.
- AngularJS provides reusable components.
- With AngularJS, the developers can achieve more functionality with short code.
- In AngularJS, views are pure html pages, and controllers written in JavaScript do the business processing.
- On the top of everything, AngularJS applications can run on all major browsers and smart phones, including Android and iOS-based phones/tablets.

Disadvantages of AngularJS

- **Not Secure –** Being JavaScript only framework, application written in AngularJS are not safe. Server-side authentication and authorization is must to keep an application secure.
- **Not degradable –** If the user of your application disables JavaScript, then nothing would be visible, except the basic page.
- **Complex at times –** At times AngularJS becomes complex to handles as there are multiple ways to do the same thing. This creates confusion and requires considerable efforts.

General features

- We can create rich internet application by using Angular JS.
- Angular JS Allow us to write the client-side code using Javascript in an MVC way.
- AngularJS provides cross-browser compliant and it automatically handles JavaScript code suitable for each browser.
- The AngularJS is Completely free and opensource it is used by thousands of developers all around the world.
- AngularJS is used to build high-performance, large scale, and easy-to-maintain web applications.
- AngularJS Directives: AngularJS is Javascript based framework and it may divide into three major part.
    - **ng-app:** By using this directive we can link AngularJS application to HTML.
    - **ng-model:** By using this directive we can bind the value of AngularJS application data to HTML input controls.
    - **ng-bind:** By using this directive we can bind AngularJS application data to HTML tags.

UI benefits in Angular JS

The AngularJS framework employs basic HTML and offers extensions through directives that allow for the website to become more dynamic. Its capabilities to automatically synchronize with models and views make development on AngularJS an easy process. It exclusively follows DOM methodology, whose primary focus lies in improving application testability & performance. Therefore, the main features of AngularJS comprise; two-way data binding, templates, MVC structure, dependency injections, directives, and testing features.

Let's take a look at some of the reasons for why AngularJS development companies leverage this framework for front-end product engineering

Provides immense support to XAML Development

XAML is an XML based markup language which is used to instantiate object graphs and set values. It allows you to define various types of objects with properties. It makes it easy to layout complex, ever changing UIs. It also supports for data-binding which allows a symbiosis between the presentation layer and your data without creating hard dependencies between its components. It also enables a developer to conduct any number of testing scenarios. AngularJS translates very well with XAML principles. It also allows a parallel workflow

between different aspects that include the markup for the UI itself as well as the underlying logic that fetches and processes data.

Provides a simplistic approach to data binding

Data binding is very easy in the Angular world. The framework eliminates the need to derive from an existing object or place all your properties and dependencies cards on the table. AngularJS uses dirty tracking to enable this. Though several existing frameworks have evolved, the process of mapping everything explicitly to an interim object to data-bind to Angular is significantly easier and faster.

Reduces application side-effects by allowing developers to express declarative UI

Declarative UI has several advantages. A structured UI enables easy understanding and manipulation of the application. Using jQuery forces the developer to know a lot about the document structure which often creates two issues: first, a fairly unstable code working as "glue" that tightly grips the changes in the UI, and second, there is an uncertainty because it is hard to judge by studying the markup just how the UI would function. Placing markup directly in HTML, one can separate the presentation logic from imperative logic and keep it in one place. Understanding of the extended markup provided by Angular i.e. code snippets makes the whereabouts of the data amply clear. The addition, tools like directives and filters not only make the UI intent even more clear but also give clarity on how the information is being shaped.

Simplifies testing

AngularJS product engineering adeptly embraces Test-Driven Development, Behavior-Driven Development, or any of the driven-development methodologies of building an application. It helps in saving time and change the way an application is structured. Angular can help to test everything UI behavior to business logic with its ability to mock dependencies.

Gives Design development workflow a new meaning

Even though HTML and CSS support design, but AngularJS allows a developer to add a markup without completely breaking an application. It often depends on a certain id or structure to locate an element and perform tasks. Developers can rearrange portions of code by moving the elements around and the corresponding code that does the binding and filtering job moves with it.

Makes Single Page Applications easy

The growing popularity of Single Page Applications among AngularJS development companies is not unfounded as they cater to a very specific need. With more and more functionality finding its way to the web, developers are progressively realizing the potential of the browser as a distributed computing node. SPA boasts of more responsive design and can provide an experience of a native app on the web. AngularJS is an apt infrastructure that supports routing, templates, and even journaling making it viable accomplice of SPA.

Here are some additional advantages to leverage for awesome front-end product engineering:

- **Improved Plug & Play Features –** AngularJS makes it easy to add components from an existing application to a new application. It only needs a copy-paste command to make your existing assets available in a new environment.
- **Quicker Development Turnaround Time –** AngularJS supported by the MVC architecture ensures faster development, testing, and maintenance. The quicker turnaround time allows developers to enhance their productivity.
- **Superior Dependency Handling –** One of the biggest USPs of AngularJS is its "dependency injection". Testing and Single Page Application have been enormously simplified because of the dependency injection feature.
- **Makes Parallel Development Possible –** Apart from faster development, AngularJS in collaboration with MVC architecture allows the developer to perform parallel application development which makes it stand-out amongst the competition.
- **Gives More Control to the Developers –** AngularJS directives give superior control to developers, who get a free hand to experiment with HTML and attributes. The directives allow them more independence to help them create more responsive platforms.
- **State Management Made Easy –** AngularJS helps you manage any application state easily whether it is illusioned or disillusioned. It is highly conducive when it comes to managing properties, permissions, and other major concerns across the application.

Usage of Angular JS with HTML

AngularJS is a JavaScript framework. It can be added to an HTML page with a <script> tag. AngularJS extends HTML attributes with Directives, and binds data to HTML with Expressions.

AngularJS is a JavaScript framework written in JavaScript. AngularJS is distributed as a JavaScript file, and can be added to a web page with a script tag:

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

AngularJS Extends HTML

AngularJS extends HTML with ng-directives.

The **ng-app** directive defines an AngularJS application.

The **ng-model** directive binds the value of HTML controls (input, select, textarea) to application data.

The **ng-bind** directive binds application data to the HTML view.

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body>
<div ng-app="">
  <p>Name: <input type="text" ng-model="name"></p>
  <p ng-bind="name"></p>
</div>
</body>
</html>
```

Example explained:

AngularJS starts automatically when the web page has loaded.

The **ng-app** directive tells AngularJS that the element is the "owner" of an AngularJS application.

The **ng-model** directive binds the value of the input field to the application variable name.

The **ng-bind** directive binds the content of the <p> element to the application variable name.

AngularJS Directives

AngularJS directives are HTML attributes with an ng prefix.

The ng-init directive initializes AngularJS application variables.

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body>
<div ng-app="" ng-init="firstName='John'">
<p>The name is <span ng-bind="firstName"></span></p>
</div
</body>
</html>
```

The name is John

AngularJS Directives

Directive	Description
ng-app	Defines the root element of an application.
ng-bind	Binds the content of an HTML element to application data.
ng-bind-html	Binds the innerHTML of an HTML element to application data, and also removes dangerous code from the HTML string.
ng-bind-template	Specifies that the text content should be replaced with a template.
ng-blur	Specifies a behavior on blur events.
ng-change	Specifies an expression to evaluate when content is being changed by the user.
ng-checked	Specifies if an element is checked or not.
ng-class	Specifies CSS classes on HTML elements.
ng-class-even	Same as ng-class, but will only take effect on even rows.
ng-class-odd	Same as ng-class, but will only take effect on odd rows.
ng-click	Specifies an expression to evaluate when an element is being clicked.
ng-cloak	Prevents flickering when your application is being loaded.
ng-controller	Defines the controller object for an application.
ng-copy	Specifies a behavior on copy events.
ng-csp	Changes the content security policy.
ng-cut	Specifies a behavior on cut events.
ng-dblclick	Specifies a behavior on double-click events.
ng-disabled	Specifies if an element is disabled or not.
ng-focus	Specifies a behavior on focus events.
ng-form	Specifies an HTML form to inherit controls from.
ng-hide	Hides or shows HTML elements.
ng-href	Specifies a url for the <a> element.

ng-if	Removes the HTML element if a condition is false.
ng-include	Includes HTML in an application.
ng-init	Defines initial values for an application.
ng-jq	Specifies that the application must use a library, like jQuery.
ng-keydown	Specifies a behavior on keydown events.
ng-keypress	Specifies a behavior on keypress events.
ng-keyup	Specifies a behavior on keyup events.
ng-list	Converts text into a list (array).
ng-maxlength	Specifies the maximum number of characters allowed in the input field.
ng-minlength	Specifies the minimum number of characters allowed in the input field.
ng-model	Binds the value of HTML controls to application data.
ng-model-options	Specifies how updates in the model are done.
ng-mousedown	Specifies a behavior on mousedown events.
ng-mouseenter	Specifies a behavior on mouseenter events.
ng-mouseleave	Specifies a behavior on mouseleave events.
ng-mousemove	Specifies a behavior on mousemove events.
ng-mouseover	Specifies a behavior on mouseover events.
ng-mouseup	Specifies a behavior on mouseup events.
ng-non-bindable	Specifies that no data binding can happen in this element, or its children.
ng-open	Specifies the open attribute of an element.
ng-options	Specifies <options> in a <select> list.
ng-paste	Specifies a behavior on paste events.
ng-pluralize	Specifies a message to display according to en-us localization rules.
ng-readonly	Specifies the readonly attribute of an element.
ng-repeat	Defines a template for each data in a collection.
ng-required	Specifies the required attribute of an element.
ng-selected	Specifies the selected attribute of an element.

ng-show	Shows or hides HTML elements.
ng-src	Specifies the src attribute for the <img> element.
ng-srcset	Specifies the srcset attribute for the <img> element.
ng-style	Specifies the style attribute for an element.
ng-submit	Specifies expressions to run on onsubmit events.
ng-switch	Specifies a condition that will be used to show/hide child elements.
ng-transclude	Specifies a point to insert transcluded elements.
ng-value	Specifies the value of an input element.

AngularJS Expressions

AngularJS expressions are written inside double braces: {{ expression }}.

AngularJS will "output" data exactly where the expression is written:

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body>
<div ng-app="">
  <p>Input something in the input box:</p>
  <p>Name:<input type=”text” ng-model=”name”></p>
  <p>{{name}}</p>
</div>
</body>
</html>
```

AngularJS expressions bind AngularJS data to HTML the same way as the ng-bind directive.

The ng-bind directive tells AngularJS to replace the content of an HTML element with the value of a given variable, or expression.

If the value of the given variable, or expression, changes, the content of the specified HTML element will be changed as well.

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body>
<div ng-app="" ng-init="myText='HelloWorld!' ">
  <p ng-bind="myText"></p>
</div>
</body>
</html>
```

Event Handling in Angular JS

AngularJS Events

AngularJS Events are the functionalities that allow web applications to interact with user inputs like mouse click, keyboard inputs, mouse hover, etc. Events need to be handled in web-based applications in order to perform specific tasks and actions. It is triggered when a specific action is performed by the user.

When creating web-based applications, sooner or later your application will need to handle DOM events like mouse clicks, moves, keyboard presses, change events, etc.

AngularJS can add functionality which can be used to handle such events.

For example, if there is a button on the page and you want to process something when the button is clicked, we can use the Angular ng-click event directive.

- **ng-mousemove:** Movement of mouse leads to the execution of event.
- **ng-mouseup:** Movement of mouse upwards leads to the execution of event.
- **ng-mousedown:** Movement of mouse downwards leads to the execution of event.
- **ng-mouseenter:** Click of the mouse button leads to the execution of event.
- **ng-mouseover:** Hovering of the mouse leads to the execution of event.
- **ng-cut:** Cut operation leads to the execution of the event.
- **ng-copy:** Copy operation leads to the execution of the event.
- **ng-keypress:** Press of key leads to the execution of the event.
- **ng-keyup:** Press of upward arrow key leads to the execution of the event.
- **ng-keydown:** Press of downward arrow key leads to the execution of the event.
- **ng-click:** Single click leads to the execution of the event.

- **ng-dblclick:** Double click leads to the execution of the even

What is ng-click Directive in AngularJS?

The “ng-click directive” in AngularJS is used to apply custom behavior when an element in HTML is clicked. This directive is normally used for buttons because that is the most commonplace for adding events that respond to clicks performed by the user. It is also used to popup alerts when a button is clicked.

Syntax of ng-click in AngularJS

```
<element
 ng-click="expression">
</element>
```

Let’s look a simple example of how we can implement the click event.

Example of ng-click in AngularJS

In this ng-click Example, we will have a counter variable which will increment in value when the user clicks a button.

```
<!DOCTYPE html>
<html>
<head>
   <meta chrset="UTF 8">
   <title>Event Registration</title>
</head>
<body ng-app="">
<script src="https://code.angularjs.org/1.6.9/angular.js"></script>
<script src="https://code.jquery.com/jquery-3.3.1.min.js"></script>
<h1>ADIT Global Event</h1>
<button ng-click="count = count + 1" ng-init="count=0">
   Increment
</button>
<div>The Current Count is {{count}}</div>
</body>
</html>
```

Code Explanation:

We are first using the ng-init directive to set the value of a local variable count to 0.

We are then introducing the ng-click event directive to the button. In this directive, we are writing code to increment the value of the count variable by 1.

Here we are displaying the value of the count variable to the user.

If the code is executed successfully, the following Output will be shown when you run your code in the browser.

Output:

Mouse Events

Mouse events occur when the cursor moves over an element, in this order:

- ng-mouseover
- ng-mouseenter
- ng-mousemove
- ng-mouseleave

Or when a mouse button is clicked on an element, in this order:

- ng-mousedown
- ng-mouseup
- ng-click

You can add mouse events on any HTML element.

ng-mousemove

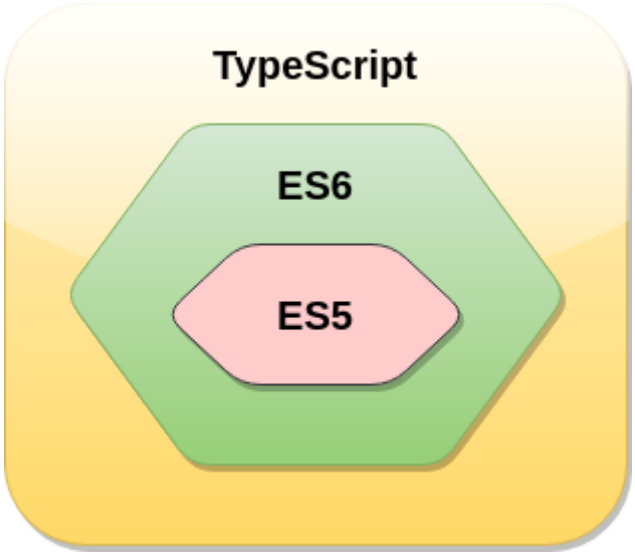
```
<!DOCTYPE html>

<html>

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

<body>

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

<h1 ng-mousemove="count = count + 1">Mouse Over Me!</h1>
```

```
<h2>{{ count }}</h2>
</div>
<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
   $scope.count = 0;
});
</script>
</body>
</html>
```

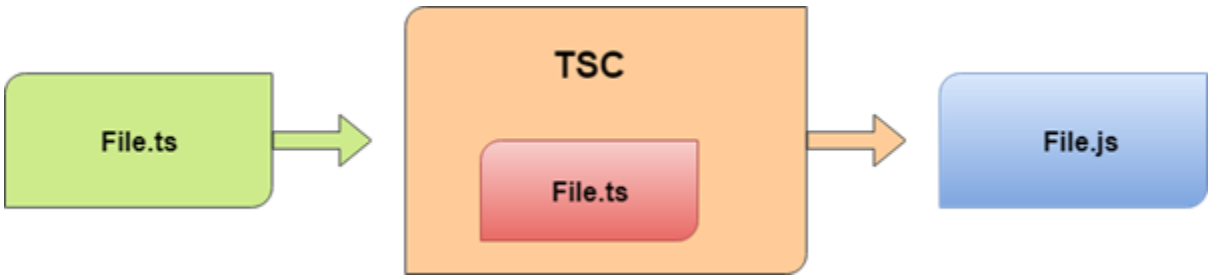
TypeScript

What is TypeScript?

TypeScript is an open-source pure object-oriented programming language. It is a strongly typed superset of JavaScript which compiles to plain JavaScript. It contains all elements of the JavaScript. It is a language designed for large-scale JavaScript application development, which can be executed on any browser, any Host, and any Operating System. The TypeScript is a language as well as a set of tools. TypeScript is the ES6 version of JavaScript with some additional features.



TypeScript cannot run directly on the browser. It needs a compiler to compile the file and generate it in JavaScript file, which can run directly on the browser. The TypeScript source file is in ".ts" extension. We can use any valid ".js" file by renaming it to ".ts" file. TypeScript uses TSC (TypeScript Compiler) compiler, which convert Typescript code (.ts file) to JavaScript (.js file).



The first version of TypeScript was released to the public in the month of 1st October 2012 and was labeled as version 0.8. Now, it is maintained by Microsoft under the Apache 2 license. The latest version of Typescript is TypeScript 3.5, which was released to the public on May 2019.

Why use TypeScript?

We use TypeScript because of the following benefits.

- TypeScript supports Static typing, Strongly type, Modules, Optional Parameters, etc.
- TypeScript supports object-oriented programming features such as classes, interfaces, inheritance, generics, etc.
- TypeScript is fast, simple, and most importantly, easy to learn.
- TypeScript provides the error-checking feature at compilation time. It will compile the code, and if any error found, then it highlighted the mistakes before the script is run.
- TypeScript supports all JavaScript libraries because it is the superset of JavaScript.
- TypeScript support reusability because of the inheritance.
- TypeScript make app development quick and easy as possible, and the tooling support of TypeScript gives us autocompletion, type checking, and source documentation.
- TypeScript has a definition file with .d.ts extension to provide a definition for external JavaScript libraries.
- TypeScript supports the latest JavaScript features, including ECMAScript 2015.
- TypeScript gives all the benefits of ES6 plus more productivity.
- Developers can save a lot of time with TypeScript.

Difference between JavaScript and TypeScript

JavaScript

JavaScript is the most popular programming language of HTML and the Web. JavaScript is an object-based scripting language which is lightweight and cross-platform. It is used to create client-side dynamic pages. The programs in JavaScript language are called scripts. The scripts are written in HTML pages and executed automatically as the page loads. It is provided and executed as plain text and does not need special preparation or compilation to run.

TypeScript

TypeScript is an open-source pure object-oriented programing language. It is a strongly typed superset of JavaScript which compiles to plain JavaScript. TypeScript is developed and maintained by Microsoft under the Apache 2 license. It is not directly run on the browser. It needs a compiler to compile and generate in JavaScript file. TypeScript source file is in ".ts" extension. We can use any valid ".js" file by renaming it to ".ts" file. TypeScript is the ES6 version of JavaScript with some additional features.

Advantage of TypeScript over JavaScript

- TypeScript always highlights errors at compilation time during the time of development, whereas JavaScript points out errors at the runtime.
- TypeScript supports strongly typed or static typing, whereas this is not in JavaScript.
- TypeScript runs on any browser or JavaScript engine.
- Great tooling supports with IntelliSense which provides active hints as the code is added.
- It has a namespace concept by defining a module.

Disadvantage of TypeScript over JavaScript

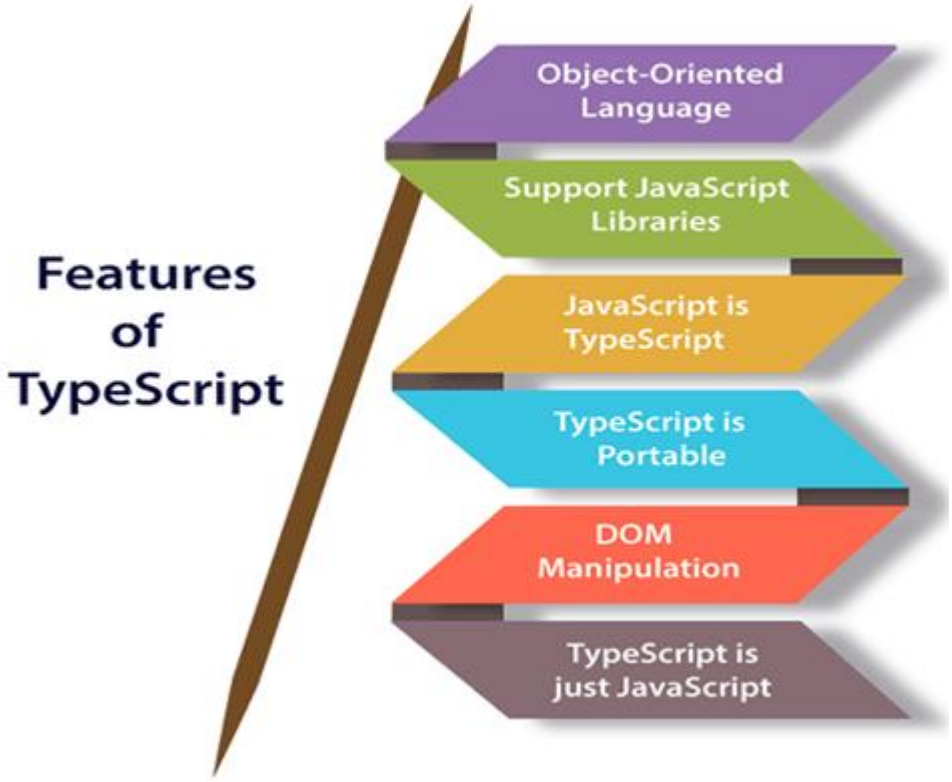
- TypeScript takes a long time to compile the code.
- TypeScript does not support abstract classes.
- If we run the TypeScript application in the browser, a compilation step is required to transform TypeScript into JavaScript.

TypeScript Vs. JavaScript

SN	JavaScript	TypeScript
1.	It doesn't support strongly typed or static typing.	It supports strongly typed or static typing feature.
2.	Netscape developed it in 1995.	Anders Hejlsberg developed it in 2012.
3.	JavaScript source file is in ".js" extension.	TypeScript source file is in ".ts" extension.
4.	It is directly run on the browser.	It is not directly run on the browser.
5.	It is just a scripting language.	It supports object-oriented programming concept like classes, interfaces, inheritance, generics, etc.
6.	It doesn't support optional parameters.	It supports optional parameters.
7.	It is interpreted language that's why it highlighted the errors at runtime.	It compiles the code and highlighted errors during the development time.
8.	JavaScript doesn't support modules.	TypeScript gives support for modules.
9.	In this, number, string are the objects.	In this, number, string are the interface.
10.	JavaScript doesn't support generics.	TypeScript supports generics.

11.	**Example:**
	<script>
	function addNumbers(a, b) {
	  return a + b;
	}
	var sum = addNumbers(15, 25);
	document.write('Sum of the numbers is: ' + sum);
	</script>
		Example:
		function addNumbers(a, b) {
		  return a + b;
		}
		var sum = addNumbers(15, 25);
		console.log('Sum of the numbers is: ' + sum);

Features of TypeScript



Object-Oriented language: TypeScript provides a complete feature of an object-oriented programming language such as classes, interfaces, inheritance, modules, etc. In TypeScript, we can write code for both client-side as well as server-side development.

TypeScript supports JavaScript libraries: TypeScript supports each JavaScript elements. It allows the developers to use existing JavaScript code with the TypeScript. Here, we can use all of the JavaScript frameworks, tools, and other libraries easily.

JavaScript is TypeScript: It means the code written in JavaScript with valid .js extension can be converted to TypeScript by changing the extension from .js to .ts and compiled with other TypeScript files.

TypeScript is portable: TypeScript is portable because it can be executed on any browsers, devices, or any operating systems. It can be run in any environment where JavaScript runs on. It is not specific to any virtual-machine for execution.

DOM Manipulation: TypeScript can be used to manipulate the DOM for adding or removing elements similar to JavaScript.

TypeScript is just a JS: TypeScript code is not executed on any browsers directly. The program written in TypeScript always starts with JavaScript and ends with JavaScript. Hence, we only need to know JavaScript to use it in TypeScript. The code written in TypeScript is compiled and converted into its JavaScript equivalent for the execution. This process is known as Trans-piled. With the help of JavaScript code, browsers can read the TypeScript code and display the output.

Advantage of TypeScript over JavaScript

- TypeScript always highlights errors at compilation time during the time of development, whereas JavaScript points out errors at the runtime.
- TypeScript supports strongly typed or static typing, whereas this is not in JavaScript.
- TypeScript runs on any browser or JavaScript engine.
- Great tooling supports with IntelliSense, which provides active hints as the code is added.
- It has a namespace concept by defining a module.

Disadvantage of TypeScript over JavaScript

- TypeScript takes a long time to compile the code.
- TypeScript does not support abstract classes.
- If we run the TypeScript application in the browser, a compilation step is required to transform TypeScript into JavaScript.

Components of TypeScript

The TypeScript language is internally divided into three main layers. Each of these layers is divided into sublayers or components. In the following diagram, we can see the three layers and each of their internal components. These layers are:

- Language
- The TypeScript Compiler
- The TypeScript Language Services

Language

It features the TypeScript language elements. It comprises elements like syntax, keywords, and type annotations.

The TypeScript Compiler

The TypeScript compiler (TSC) transform the TypeScript program equivalent to its JavaScript code. It also performs the parsing, and type checking of our TypeScript code to JavaScript code.

Browser doesn't support the execution of TypeScript code directly. So the program written in TypeScript must be re-written in JavaScript equivalent code which supports the execution of code in the browser directly. To perform this, TypeScript comes with TypeScript compiler named "tsc." The current version of TypeScript compiler supports ES6, by default. It compiles the source code in any module like ES6, SystemJS, AMD, etc.

We can install the TypeScript compiler by locally, globally, or both with any npm package. Once installation completes, we can compile the TypeScript file by running "tsc" command on the command line.

```
$ tsc helloworld.ts   // It compiles the TS file helloworld into the helloworld.js file.
```

The TypeScript Language Services

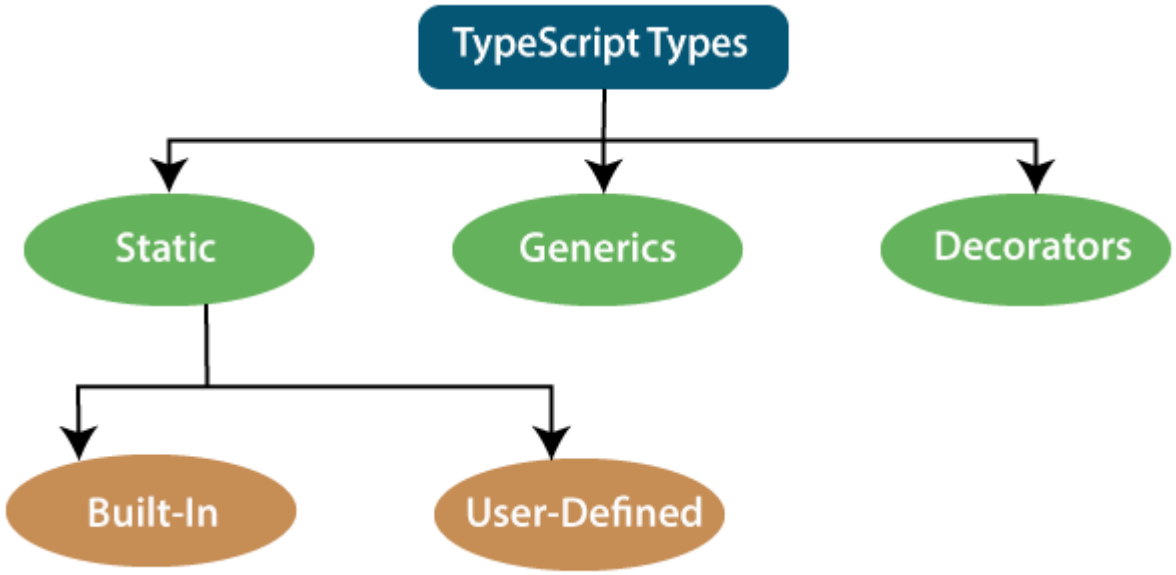
The language service provides information which helps editors and other tools to give better assistance features such as automated refactoring and IntelliSense. It exposes an additional layer around the core-compiler pipeline. It supports some standard typical editor operations like code formatting and outlining, colorization, statement completion, signature help, etc.

Typescript Datatype and Operator

TypeScript Type

The TypeScript language supports different types of values. It provides data types for the JavaScript to transform it into a strongly typed programing language. JavaScript doesn't support data types, but with the help of TypeScript, we can use the data types feature in JavaScript. TypeScript plays an important role when the object-oriented programmer wants to use the type feature in any scripting language or object-oriented programming language. The Type System checks the validity of the given values before the program uses them. It ensures that the code behaves as expected.

TypeScript provides data types as an optional Type System. We can classify the TypeScript data type as following.

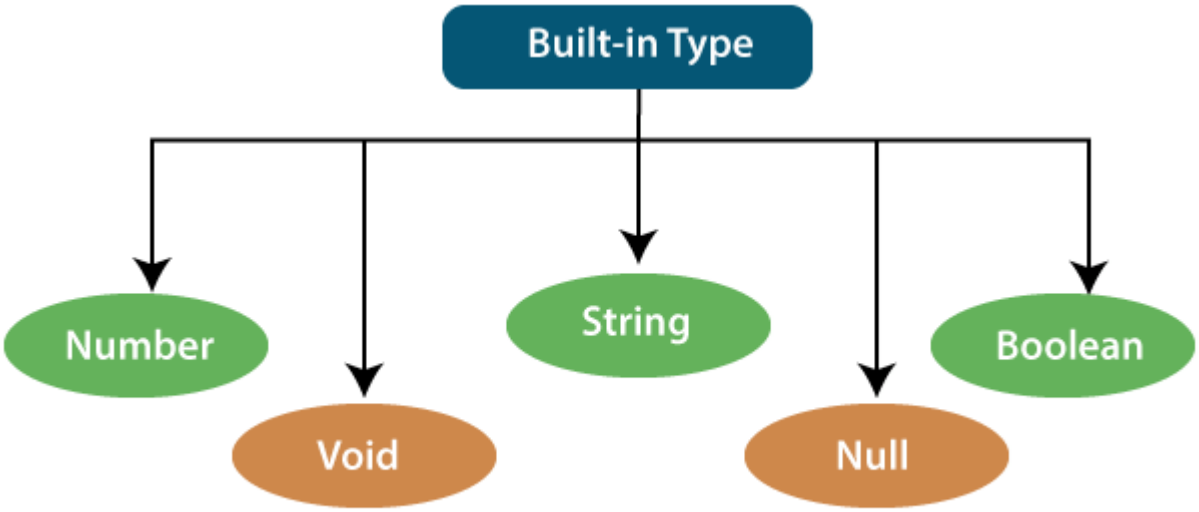


Static Types

In the context of type systems, static types mean "at compile time" or "without running a program." In a statically typed language, variables, parameters, and objects have types that the compiler knows at compile time. The compiler used this information to perform the type checking.

Built-in or Primitive Type

The TypeScript has five built-in data types, which are given below.



Built-in Datatypes: TypeScript has some pre-defined data-types-

Built-in Datatype	Keyword	Description
Number	number	It is used to represent both Integer as well as Floating-Point numbers
Boolean	boolean	Represents true and false
String	string	It is used to represent a sequence of characters
Void	void	Generally used on function return-types
Null	null	It is used when an object does not have any value

Number

Like JavaScript, all the numbers in TypeScript are stored as floating-point values. These numeric values are treated like a number data type. The numeric data type can be used to represents both integers and fractions. TypeScript also supports Binary(Base 2), Octal(Base 8), Decimal(Base 10), and Hexadecimal(Base 16) literals.

Example:

```
let a:number=1;
console.log(a);
```

Output:

```
1
```

In the above code, we have declared the variable named a, and since we are going to store an arithmetical value in it so we are using the data type number.Run the code in your editor for a clear explanation.

String:

We will use the string data type to represents the text in TypeScript. String type work with textual data. We include string literals in our scripts by enclosing them in single or double quotation marks. It also represents a sequence of Unicode characters. It embedded the expressions in the form of ${expr}.

Syntax:

```
let identifier: string = " ";
 Or
let identifier: string = ' ';
```

Example:

```
let name1:string='yash';
let name2:string="yash1";
console.log(name1);
console.log(name1);
```

Output:

```
yash
yash1
```

In the above code, we have declared a variable named name1 and name2 since we are going to store text data in it so we are using the data type string.

Boolean

The string and numeric data types can have an unlimited number of different values, whereas the Boolean data type can have only two values. They are "true" and "false." A Boolean value is a truth value which specifies whether the condition is true or not.

Syntax

```
let identifier: BooleanBoolean = Boolean value;
```

Example:

```
let a:boolean=true;
let b:boolean=false;
console.log(a);
console.log(b);
```

Output:

```
true
false
```

In the above code, we have declared a variable named a and b since we are going to check whether the variables are true or false so we are using the data type boolean. Run the code in your editor for a clear explanation.

Void

A void is a return type of the functions which do not return any type of value. It is used where no data type is available. A variable of type void is not useful because we can only assign undefined or null to them. An undefined data type denotes uninitialized variable, whereas null represents a variable whose value is undefined.

Syntax

```
let unusable: void = undefined;
```

Example:

```
let a:void=undefined;
a=3;//error
```

Output:

```
Line 2:Type 'number' is not assignable to type 'void'.
```

If we assign a variable using the void data type only undefined can be assigned to it.No other data types can be assigned to the variable when we declare the variable using the void data type. We can also use void as the return type of those function that does not return any value. Run the code in your editor for a clear explanation.

Let's see another example of when the void is useful:

Example:

```
function tsscaler(): void {
  console.log('scaler!')
}

let a: void = tsscaler();
console.log(a);
```

Output:

```
Undefined
```

As we know there is no meaning to assigning void to a variable, as only null or undefined is assignable to void.

Null

Null represents a variable whose value is undefined. Much like the void, it is not extremely useful on its own. The Null accepts the only one value, which is null. The Null keyword is used to define the Null type in TypeScript, but it is not useful because we can only assign a null value to it.

Example:

```
let flag: null;
flag=10;
```

Output:

```
Type '10' is not assignable to type 'null'.
```

In this code, we have created a variable named flag and set it to null, then we assigned the value 10 to the variable flag which is throwing an error because no other data type can be assigned to the variable when we have set the variable as null. Run the code in your editor for a clear explanation.

Now let us see another example where we are assigning null to values:

Example:

```
let a: number;
console.log(a);//Output:Undefined
```

```
        // Assigning 'null' to variable
    a = null;
    console.log(a);
```
Output:
```
    undefined
    null
```
When we were printing an empty string the output was undefined but when we are assigning null to the variable then the output is null.

Undefined

The Undefined primitive type denotes all uninitialized variables in TypeScript and JavaScript. It has only one value, which is undefined. The undefined keyword defines the undefined type in TypeScript, but it is not useful because we can only assign an undefined value to it.

Example:
```
    var name1:undefined;
    name1="Scaler";
```
Output:
```
    Type '"Scaler"' is not assignable to type 'undefined'.
```
In the above code, we have declared a variable named name1 and then set it to undefined, and then we were assigning a string type data to it which is throwing an error because undefined can only be assigned to variables with any data type, and undefined data types. Run the code in your editor for a clear explanation.

Any Type

It is the "super type" of all data type in TypeScript. It is used to represents any JavaScript value. It allows us to opt-in and opt-out of type-checking during compilation. If a variable cannot be represented in any of the basic data types, then it can be declared using "Any" data type. Any type is useful when we do not know about the type of value (which might come from an API or 3rd party library), and we want to skip the type-checking on compile time.

Syntax

```
let identifier: any = value;
```
Example:
```
    let a: any;
    let b: any;
    a="scaler";
    b=1;
    console.log(a);
    console.log(b);
```
Output:
```
    scaler
    1
```

In the above code, we have declared two variables a and b, and set them to any and they are assigning string and numeric values in it which will not throw any error as the variables are set as any, and any disables type checking. This behavior is similar to javascript's variable Run the code in your editor for a clear explanation.

The important difference between Null and Undefined are:

SN	Null	Undefined
1.	It is an assignment value. It can be assigned to a variable which indicates that a variable does not point any object.	It is not an assignment value. It means a variable has been declared but has not yet been assigned a value.
2.	It is an object.	It is a type itself.
3.	The null value is a primitive value which represents the null, empty, or non-existent reference.	The undefined value is a primitive value, which is used when a variable has not been assigned a value.
4.	Null indicates the absence of a value for a variable.	Undefined indicates the absence of the variable itself.
5.	Null is converted to zero (0) while performing primitive operations.	Undefined is converted to NaN while performing primitive operations.

TypeScript Variables

A variable is the storage location, which is used to store value/information to be referenced and used by programs. It acts as a container for value in code and must be declared before the use. We can declare a variable by using the var keyword. In TypeScript, the variable follows the same naming rule as of JavaScript variable declaration. These rules are

- The variable name must be an alphabet or numeric digits.
- The variable name cannot start with digits.
- The variable name cannot contain spaces and special character, except the underscore(_) and the dollar($) sign.

In ES6, we can define variables using let and const keyword. These variables have similar syntax for variable declaration and initialization but differ in scope and usage. In TypeScript, there is always recommended to define a variable using let keyword because it provides the type safety.

The let keyword is similar to var keyword in some respects, and const is an let which prevents prevents re-assignment to a variable.

Variable Declaration

TypeScript provides the following 4 ways to declare a variable. Common syntax for a variable declaration is to include a colon (:) after the variable name which followed by its type. We can use var or let keyword to declare a variable.

1. Declare a variable with type and value.

By using var keyword:

```
var variableName:data_type = value;
```

By using let keyword:

```
let variableName:data_type = value;
```

2. Declare a variable with type but no value. Variable's value will be set to undefined by default.

   By using var keyword:

   ```
   var variableName:data_type;
   ```

   By using let keyword:

   ```
   let variableName:data_type;
   ```

3. Declare a variable with value but no type. Variable's type will be set to any by default.

   By using var keyword:

   ```
   var variableName = value;
   ```

   By using let keyword:

   ```
   let variableName = value;
   ```

4. Declare a variable without type and value. Variable's type will be set to any and value will be set to undefined by default.

   By using var keyword:

   ```
   var variableName;
   ```

   By using let keyword:

   ```
   let variableName;
   ```

Difference between let and var keyword

var keyword

The var statement is used to declare a variable in JavaScript. A variable declared with the var keyword is defined throughout the program.

Example

```
var greeter = "hey hi";
var times = 5;
if (times > 3) {
  var greeter = "Say Hello NSTI";
}
console.log(greeter) //Output: Say Hello NSTI
```

let keyword

The let statement is used to declare a local variable in TypeScript. It is similar to the var keyword, but it has some restriction in scoping in comparison of the var keyword. The let keyword can enhance our code readability and decreases the chance of programming error. A variable declared with the let keyword is limited to the block-scoped only.

Note: The key difference between var and let is not in the syntax, but it differs in the semantics.

Example

```
let greeter = "hey hi";

let times = 5;

if (times > 3) {

  let hello = "Say Hello JavaTpoint";

  console.log(hello) // Output: Say Hello JavaTpoint

}

console.log(hello) // Compile error: greeter is not defined
```

Var vs. Let Keyword

SN	var	let
1.	The var keyword was introduced with JavaScript.	The let keyword was added in ES6 (ES 2015) version of JavaScript.
2.	It has global scope.	It is limited to block scope.
3.	It can be declared globally and can be accessed globally.	It can be declared globally but cannot be accessed globally.

4.	Variable declared with var keyword can be re-declared and updated in the same scope. **Example:**
	function varGreeter(){
	var a = 10;
	var a = 20; //a is replaced
	console.log(a);
	}
	varGreeter();
		Variable declared with let keyword can be updated but not re-declared. **Example:**
		function varGreeter(){
		let a = 10;
		let a = 20; //SyntaxError:
		//Identifier 'a' has already been declared
		console.log(a);
		}
		varGreeter();

5.	It is hoisted.
	Example:
	{
	console.log(c); // undefined.
	//Due to hoisting
	var c = 2;
	}
		It is not hoisted.
		Example:
		{
		console.log(b); // ReferenceError:
		//b is not defined
		let b = 3;
		}

Typescript OOPS

In object-oriented programming languages like Java, classes are the fundamental entities which are used to create reusable components. It is a group of objects which have common properties. In terms of OOPs, a class is a template or blueprint for creating objects. It is a logical entity.

A class definition can contain the following properties:

- **Fields**: It is a variable declared in a class.
- **Methods**: It represents an action for the object.
- **Constructors**: It is responsible for initializing the object in memory.
- **Nested class and interface**: It means a class can contain another class.

Class

In TypeScript, a class is a blueprint for creating objects that define a set of properties and methods that are common to all instances of the class. A class is declared using the class keyword, followed by the class name and a set of curly braces containing the class definition.

Syntax

The syntax of declaring a class in TypeScript is as follows −

```
class ClassName {
  // properties and methods
}
```

Example

Here we have declared a Car class that contains make, model and year attributes and their corresponding getter and setter methods.

```
class Car {
  private make: string;
  private model: string;
  private year: number;
  constructor(make: string, model: string, year: number) {
    this.make = make;
    this.model = model;
    this.year = year;
```

```
  }
  public getMake(): string {
    return this.make;
  }
  public setMake(make: string): void {
    this.make = make;
  }
  public getModel(): string {
    return this.model;
  }
  public setModel(model: string): void {
    this.model = model;
  }
  public getYear(): number {
    return this.year;
  }
  public setYear(year: number): void {
    this.year = year;
  }
}
```

Object

A class creates an object by using the new keyword followed by the class name. The new keyword allocates memory for object creation at runtime. All objects get memory in heap memory area. We can create an object as below.

Syntax

```
let object_name = new class_name(parameter)
Or
let objectName = new ClassName(arguments);
```

- **new keyword**: it is used for instantiating the object in memory.
- The right side of the expression invokes the constructor, which can pass values.

Example

This creates an object named myCar with make: Toyota, model: Camry, and year: 2022

```
let myCar = new Car('Toyota', 'Camry', 2022);

console.log(myCar.getMake(), myCar.getModel(), myCar.getYear());
```

Object Initialization

Object initialization means storing of data into the object. There are three ways to initialize an object. These are:

1. By reference variable

Example

```
//Creating an object or instance

let obj = new Student();


//Initializing an object by reference variable

obj.id = 101;

obj.name = "Virat Kohli";
```

2. By method

A method is similar to a function used to expose the behavior of an object.

Advantage of Method

- Code Reusability
- Code Optimization

Example

```
//Defining a Student class.

class Student {

  //defining fields

  id: number;

  name:string;


  //creating method or function

  display():void {
```

```
        console.log("Student ID is: "+this.id)
        console.log("Student Name is: "+this.name)
    }
}

//Creating an object or instance
let obj = new Student();
obj.id = 101;
obj.name = "Virat Kohli";
obj.display();
```

Constructor

In TypeScript, a constructor is a special method that is called when an object is created from a class. The constructor is used to initialize the object with default values. The constructor method is declared using the constructor keyword, followed by any necessary arguments.

Syntax

The syntax for writing a constructor function in TypeScript −

```
class ClassName {
  constructor(arguments) {
    // initialization code
  }
}
```

Example

This defines a constructor with an empty body and three parameters, namely: make, model, and year.

```
class Car {
  constructor(private make: string, private model: string, private year: number) {}
}
```

Inheritance

In TypeScript, inheritance is a mechanism that allows a class to inherit properties and methods from another class. A class that inherits from another class is called a subclass or derived class, and the class that is inherited from is called the superclass or base class. In TypeScript, inheritance is declared using the extends keyword.

Syntax

The syntax for creating a child class (sub-class) from a parent class (superclass) −

```
class SubclassName extends SuperclassName {
  // additional properties and methods
}
```

Example

Here the child class: ElectricCar, inherits the properties and methods from the parent class: Car. Additionally, it has its own property range and its getter and setter methods which are not present in the parent class.

```
class ElectricCar extends Car {
  private range: number;
  constructor(make: string, model: string, year: number, range: number) {
    super(make, model, year);
    this.range = range;
  }
  public getRange(): number {
    return this.range;
  }
  public setRange(range: number): void {
    this.range = range;
  }
}
const tesla = new ElectricCar("Tesla", "Model S", 2019, 300);
console.log(
  tesla.getMake(),
  tesla.getModel(),
  tesla.getYear(),
```

```
  tesla.getRange()
);
```

Polymorphism

In TypeScript, polymorphism is the ability of an object to take on many forms or types, allowing objects of different classes to be treated as if they were objects of a common class. Polymorphism is achieved through inheritance and interfaces. Here is an example demonstrating polymorphism

Example

Here, we have first created an interface Shape which is a blueprint similar to classes in TypeScript. It has a method-declared area. This interface is implemented twice, once for the Circle class and later on for the Rectangle class. In both of these classes, we have defined the body of the area function, which is defined differently, but the name of the function is exactly the same. This demonstrates the polymorphism property in OOP (same name but different forms).

```
interface Shape {
  area(): number;
}

class Circle implements Shape {
  constructor(private radius: number) {}

  public area(): number {
    return Math.PI * this.radius ** 2;
  }
}

class Rectangle implements Shape {
  constructor(private width: number, private height: number) {}

  public area(): number {
    return this.width * this.height;
  }
```

```
}

let shapes: Shape[] = [new Circle(5), new Rectangle(10, 20)];

shapes.forEach(shape => console.log(shape.area()));
```

Output

78.53981633974483

200

Interface

In TypeScript, an interface is a blueprint for creating objects that define a set of properties and methods that must be implemented by any object that implements the interface. An interface is declared using the interface keyword, followed by the interface name and a set of curly braces containing the interface definition.

Syntax

```
interface InterfaceName {
  // properties and methods
}
```

Example

```
interface Person {
  firstName: string;
  lastName: string;
  age: number;
}
class Employee implements Person {
  constructor(public firstName: string, public lastName: string, public age: number, public jobTitle: string) {}
}


let employee: Person = new Employee('John', 'Doe', 30, 'Software Engineer');

console.log(employee.firstName, employee.lastName, employee.age);
```

Output

John Doe 30

Data Bindings

Data binding in TypeScript is a mechanism that allows you to synchronize data between your application's model and its view. In the context of web development, this often involves connecting data in your TypeScript classes with the HTML elements on the page.

There are several ways to achieve data binding in TypeScript, especially in the context of frameworks like Angular.

one way data binding

One-way data binding involves binding data from your TypeScript classes to the HTML elements in a single direction. This means that changes in your TypeScript classes will update the associated HTML elements, but changes in the HTML elements won't affect the TypeScript classes directly.

Here's an example of one-way data binding in TypeScript:

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


@Component({
  selector: 'app-root',
  standalone: true,
  imports: [RouterOutlet],
  templateUrl: './app.component.html',
  styleUrl: './app.component.css'
})
export class AppComponent {
  title = 'Data binding';
  numberA: number=10;
  numberB: number=20;


  addTwoNumbers() {
    return this.numberA + this.numberB;
  }


}
```

HTML

```
<h2>
 {{title}}
</h2>>
<p>Addition is: {{addTwoNumbers()}}</p>
```

In this example:

- The TypeScript class User has firstName and lastName properties.
- The HTML file includes two span elements with specific IDs (firstName and lastName).
- In the embedded <script> tag, we access the user object from the global scope and update the content of the span elements with the corresponding properties.

This is a basic example to illustrate one-way data binding. In more complex scenarios or when working with modern web frameworks, one-way data binding is often handled more elegantly through frameworks or libraries like Angular, React, or Vue.js. These frameworks provide abstractions and tools for managing the interaction between data and the user interface in a more efficient and maintainable way.

Event Binding

Angular provides us with other types of binding, i.e., event binding, which is used to handle the events raised from the DOM like button click, mouse move etc. Let's understand this with the help of an example –

Suppose we have a button in the HTML template and we want to handle the click event of this button. To implement event binding, we will bind click event of a button with a method of the component.

```
// app.component.ts


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


@Component({
 selector: 'app-root',
 templateUrl: './app.component.html',
```

```
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  onSave(){
    console.log("Save operation is performed!");
  }
}
```

```
// app.module.ts


import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';


// Other imports and module configurations go here


@NgModule({
  declarations: [
    // ... other components
    AppComponent,
  ],
  bootstrap: [AppComponent],
  // Other module configurations go here
})
export class AppModule { }
```

add.component.html

```
<h2>Event Binding Demo</h2>
   <button (click)="onSave()">Save</button>"
```

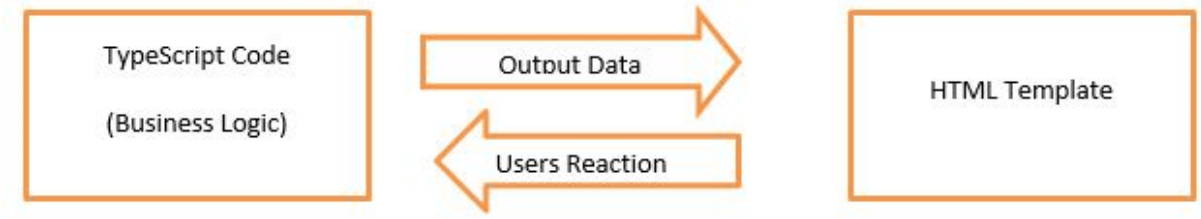
Output

When a user clicks on this button, the button click event will fire and onSave method will be called, which will log a message in the console.

Two way Data Binding

In two-way databinding, automatic synchronization of data happens between the Model and the View. Here, change is reflected in both components. Whenever you make changes in the Model, it will be reflected in the View and when you make changes in View, it will be reflected in Model.

This happens immediately and automatically, ensures that the HTML template and the TypeScript code are updated at all times.



Let's consider a simple TypeScript class representing a user:

app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import {FormsModule} from '@angular/forms';
import { AppComponent } from './app.component';
@NgModule({
 declarations: [
  AppComponent
 ],
 imports: [
  BrowserModule,
  FormsModule
 ],
 providers: [],
 bootstrap: [AppComponent]
})
export class AppModule { }
```

Now, let's create an HTML form that binds to the properties of this User class:

```
<h2>Two-way Binding Demo</h2>
  <input [(ngModel)]="fullName" /> <br/><br/>
<p> {{fullName}} </p>
```

App.component.ts

```
import { Component } from "@angular/core";

@Component({
 selector: "app-root",
 templateUrl: "./app.component.html",
 styleUrls: ["./app.component.css"]
})
export class AppComponent {
 fullName: string = "Rahul";
}
```

Now, ngModel will trigger the input event and update the value of input textbox in our view and when we will change the value of the textbox, it will automatically update in the “fullName” property of our component and vice-versa (as shown with the help of including string interpolation inside a paragraph).

Angular Routing

Routing is a core feature in AngularJS. This feature is useful in building SPA (Single Page Application) with multiple views. In SPA application, all views are different Html files and we use Routing to load different part of application and its help to divide application logically and make it manageable. In other words, Routing helps us to divide our application in logical views and bind them with different controllers.

Step-1: Create angular-App

```
ng new angular-routing
cd angular-routing
```

Step 2: Create Components

Create two components: HomeComponent and AboutComponent.

```
ng generate component home
ng generate component about
```

Step 3: Set Up Routing

1. Open src/app/app-routing.module.ts and define the routes:

```
// app-routing.module.ts


import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { HomeComponent } from './home/home.component';
import { AboutComponent } from './about/about.component';
const routes: Routes = [
  { path: '', component: HomeComponent },
  { path: 'about', component: AboutComponent },
];
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule],
})
export class AppRoutingModule {}
```

2. Open src/app/app.module.ts and import the AppRoutingModule:

```
// app.module.ts

import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppRoutingModule } from './app-routing.module'; // Import this line
import { AppComponent } from './app.component';
import { HomeComponent } from './home/home.component';
import { AboutComponent } from './about/about.component';

@NgModule({
 declarations: [
  AppComponent,
  HomeComponent,
  AboutComponent,
 ],
 imports: [
  BrowserModule,
  AppRoutingModule, // Add this line
 ],
 providers: [],
 bootstrap: [AppComponent],
})
export class AppModule {}
```

Step 4: Use Router Outlet

Open src/app/app.component.html and add the <router-outlet></router-outlet> where you want the routed components to be displayed:

```
<!-- app.component.html -->

<div>

 <h1>Angular Routing Example</h1>

 <nav>

  <a routerLink="/">Home</a>

  <a routerLink="/about">About</a>

 </nav>

 <router-outlet></router-outlet>

</div>
```

Step 5: Navigation Links

In your components or templates, you can use [routerLink] to navigate between routes:

```
<!-- home.component.html -->

<p>Welcome to the Home Component!</p>

<!-- about.component.html -->

<p>Welcome to the About Component!</p>
```

Step 6: Serve and Test

Run your Angular app:

Angular JS $http Service

The $http service is one of the most common used services in AngularJS applications. The service makes a request to the server, and lets your application handle the response.

Syntax:

```
function studentController($scope,$https:) {
  var url = "data.txt";

  $https:.get(url).success( function(response) {
    $scope.students = response;
  });
}
```

Method: There are lots of methods that can be used to call $http service, this are also shortcut methods to call $http service.

- .post()
- .get()
- .head()
- .jsonp()
- .patch()
- .delete()
- .put()

Properties: With the help of these properties, the response from the server is an object.

- .headers : To get the header information (A Function).
- .statusText: To define the HTTP status (A String).
- .status: To define the HTTP status (A Number).
- .data: To carry the response from the server (A string/ An Object).
- .config: To generate the request (An Object).

Example:

Step 1

First, include the AngularJS script in your HTML file.

```
//app.component.html
<!DOCTYPE html>
<html ng-app="myApp">

<head>
  <title>AngularJS HTTP Example</title>
  <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></s
cript>
  <script src="app.js"></script>
</head>

<body ng-controller="MyController">
  <!-- Your content goes here -->
</body>

</html>
```

Step 2: Create a Controller with HTTP Request

```
// app.js

angular.module('myApp', [])
   .controller('MyController', function ($scope, $http) {
      $scope.data = [];

      $scope.loadData = function () {
```

```
        $http.get('https://jsonplaceholder.typicode.com/todos')
            .then(function (response) {
                $scope.data = response.data;
            })
            .catch(function (error) {
                console.error('Error fetching data:', error);
            });
    };
});
```

Step 3: Use the Controller in HTML

```
//app.component.html
<body ng-controller="MyController">
  <h1>AngularJS HTTP Example</h1>
  <button ng-click="loadData()">Load Data</button>

  <div ng-if="data.length > 0">
    <h2>Data:</h2>
    <ul>
      <li ng-for="item in data">{{ item.title }}</li>
    </ul>
  </div>
</body>
```

Step 4: Create Your Angular Module and Component

```
// app.component.ts

import { Component } from '@angular/core';
import { HttpClient } from '@angular/common/http';

@Component({
  selector: 'app-root',
  template: `
    <h1>Angular HTTP Example</h1>
    <button (click)="loadData()">Load Data</button>

    <div *ngIf="data.length > 0">
      <h2>Data:</h2>
      <ul>
        <li *ngFor="let item of data">{{ item.title }}</li>
      </ul>
    </div>
  `,
})
export class AppComponent {
  data: any[] = [];

  constructor(private http: HttpClient) {}

  loadData() {
    this.http.get<any[]>('https://jsonplaceholder.typicode.com/todos')
      .subscribe(
        (response) => {
```

```
        this.data = response;
      },
      (error) => {
        console.error('Error fetching data:', error);
      }
    );
  }
}
```

Step 5: Update Your Module

Make sure you have the necessary module setup in your app.module.ts.

```
import { NgModule } from '@angular/core';
import { BrowserModule, provideClientHydration } from '@angular/platform-browser';
import { HttpClientModule } from '@angular/common/http';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,HttpClientModule
  ],
  providers: [
    provideClientHydration()
  ],
```

```
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Step 6: Run Your Application