**SET – 1**

**1. Explain briefly about various angular versions and explain various features of angular 8?**

**Ans:**

Google releases the version of Angular for the improvement of mobile and web development capabilities. All the released versions are backward compatible and can be updated easily to the newer version.

AngularJS

AngularJs is very powerful JavaScript framework. It was released in October 2010. AngularJS based on Model View Controller (MVC) architecture and automatically handles JavaScript code suitable for each browser.

Angular 2.0

Angular 2.0 was released in September 2016. It is re-engineered and rewritten version of AngularJS. AngularJs had a focus on controllers but, version 2 has changed focus on components. Components are the main building block of application. It supports features for speed in rendering, updating pages and building cross-platform native mobile apps for Google Android and iOS.

Angular 4.0

Angular 4.0 was released in March 2017. It is updated to TypeScript 2.2, supports ng ifelse conditions whereas Angular 2 supported only if conditions. Angular 4.0 introduces animation packages, Http search parameters and finally angular 4 applications are smaller and faster.

Angular 5.0

Angular 5.0 was released in November 2017. It supported some of the salient features such as HTTPClient API, Lambda support, Improved Compiler and build optimizer.

Angular 6.0

Angular 6.0 was released in May 2018. Features added to this version are updated Angular CLI, updated CDK, updated Angular Material, multiple validators and usage of reactive JS library.

Angular 7.0

Angular 7.0 was released in October 2018. Some of salient features are Google supported community, POJO based development, modular structure, declarative user interface and modular structure.

**Angular 8 New features**

Angular 8 comes up with the following new attractive features:

• Bazel support - If your application uses several modules and libraries, Bazel concurrent builds helps to load faster in your application.

• Lazy loading - Angular 8 splits AppRoutingModule into smaller bundles and loads the data in the DOM.

• Differential loading - When you create an application, Angular CLI generates modules and this will be loaded automatically then browser will render the data.

• Web worker - It is running in the background, without affecting the performance of a page.

• Improvement of CLI workflow - Angular 8 CLI commands ng-build, ng-test and ng-run are extended to third party libraries.

• Router Backward Compatibility - Angular router backward compatibility feature helps to create path for larger projects so user can easily add their coding with the help of lazy coding.

• Opt-in usage sharing - User can opt into share Angular CLI usage data.

**2. Explain the Architecture of Angular 8 with neat diagrams**

**Ans:**

Angular framework is based on four core concepts and they are as follows:

• Components.

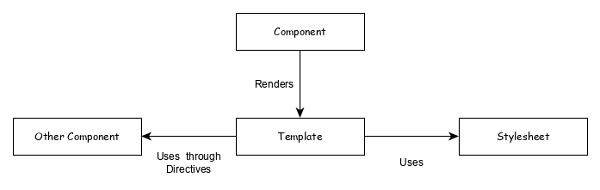
• Templates with Data binding and Directives.

• Modules.

• Services and dependency injection.

**Component**

The core of the Angular framework architecture is Angular Component. Angular Component is the building block of every Angular application. Every angular application is made up of one more Angular Component. It is basically a plain JavaScript / Typescript class along with a HTML template and an associated name. The HTML template can access the data from its corresponding JavaScript / Typescript class. Component’s HTML template may include other component using its selector’s value (name). The Angular Component may have an optional CSS Styles associated it and the HTML template may access the CSS Styles as well.



**AppComponent** property (title) can be used in the HTML template as mentioned below −

{{ title }}

Template

Template is basically a super set of HTML. Template includes all the features of HTML and provides additional functionality to bind the component data into the HTML and to dynamically generate HTML DOM elements.

template can be categorised into two items –

* Data binding

Used to bind the data from the component to the template.

{{ title }}

Here, **title** is a property in **AppComponent** and it is bind to template using **Interpolation**.

* Directives

Used to include logic as well as enable creation of complex HTML DOM elements.

<p \*ngIf="canShow">

This sectiom will be shown only when the \*canShow\* propery's value in the corresponding component is \*true\* </p>

<p [showToolTip]='tips' />

Here, **ngIf** and **showToolTip** (just an example) are directives. **ngIf** create the paragraph DOM element only when **canShow** is true. Similarly, **showToolTip** is **Attribute Directives**, which adds the tooltip functionality to the paragraph element.

When user mouse over the paragraph, a tooltip with be shown. The content of the tooltip comes from tips property of its corresponding component.

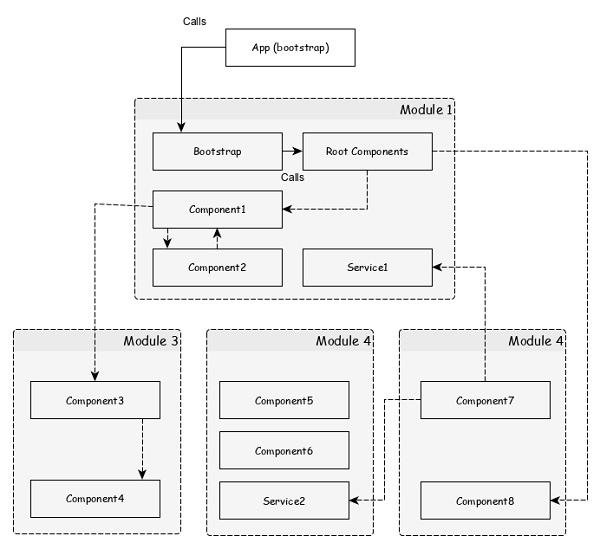
Modules

**Angular Module** groups multiple components and services under a single context.

For example, animations related functionality can be grouped into single module and Angular already provides a module for the animation related functionality, **BrowserAnimationModule** module.

An Angular application can have any number of modules but only one module can be set as root module, which will bootstrap the application and then call other modules as and when necessary. A module can be configured to access functionality from other module as well. In short, components from any modules can access component and services from any other modules.

Following diagram depicts the interaction between modules and its components.



Services

**Services** are plain Typescript / JavaScript class providing a very specific functionality. **Services** will do a single task and do it best. The main purpose of the service is reusability. Instead of writing a functionality inside a component, separating it into a service will make it useable in other component as well.

Also, **Services** enables the developer to organise the business logic of the application. Basically, component uses services to do its own job. **Dependency Injection** is used to properly initialise the service in the component so that the component can access the services as and when necessary without any setup.

**3. Explain any five CLI commands with examples**

**Ans:**

Angular CLI helps developers to create projects easily and quickly. As we know already, Angular CLI tool is used for development and built on top of Node.js, installed from NPM

**Add command**

It is used to add support for an external library to your project. It is specified by the below command: ng add [name]

**Build command**

It is used to compile or build your angular app. It is defined below:

ng build

After using this command, you could see the below response: Generating ES5 bundles for differential loading... ES5 bundle generation complete.

**Config command**

It is used to retrieve or set Angular configuration values in the angular.json file for the workspace. It is defined below:

ng config

**Doc command**

It is used to open the official Angular documentation (angular.io) in a browser, and searches for a given keyword.

ng doc

For example, if you search with component as ng g component then, it will open the documentation.

**e2e command**

It is used to build and serves an Angular app, then runs end-to-end tests using Protractor. It is stated below:

ng e2e [options]

**Help command**

It lists out available commands and their short descriptions. It is stated below:

ng help

**Serve command**

It is used to build and serves your app, rebuilding on file changes. It is given below:

ng serve

**Test command**

Runs unit tests in a project. It is mentioned below:

ng test

**Update command**

Updates your application and its dependencies. It is given below:

ng update

**Version command**

Shows Angular CLI version. It is stated below: ng version

**4. Define MEAN and Explain the MEAN stack application with neat diagram?**

**Ans:**

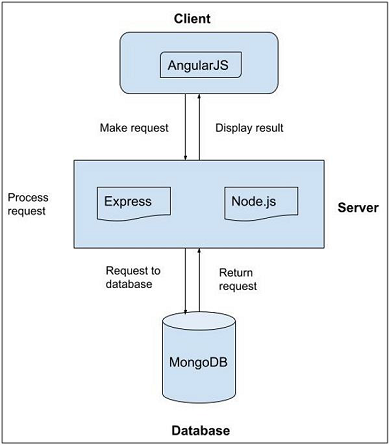
The term MEAN.js is a full stack JavaScript open-source solution, used for building dynamic websites and web applications.

MEAN is an acronym that stands for MongoDB, Express, Node.js and AngularJS, which are the key components of the MEAN stack. It was basically developed to solve the common issues with connecting those frameworks (Mongo, Express Nodejs, AngularJS), build a robust framework to support daily development needs, and help developers use better practices while working with popular JavaScript components.

Stack means using the database and web server in the back end, in the middle you will have logic and control for the application and interaction of user at the front end.

It includes following four building blocks to build an application.

* **MongoDB** − It is a document database, that stores data in flexible, JSON-like documents.
* **Express** − It is web application framework for Nodejs.
* **Node.js** − It is Web Server Platform. It provides rich library of various JavaScript modules which simplifies the development of web applications.
* **AngularJS** − It is a web frontend JavaScript framework. It allows creating dynamic, single page applications in a clean Model View Controller (MVC) way.

For more information on these, you can refer the [overview](https://www.tutorialspoint.com/meanjs/meanjs_overview.htm) chapter. The below diagram depicts architecture of MEAN stack application.

As shown in the above image, we have AngularJS as client side language which processes the request of a client.

* Whenever a user makes a request, it is first processed by AngularJS.
* Next, request enters second stage, where we have Node.js as server side language and *ExpressJS* as backend web framework.
* *Node.js* handles the client/server requests and *ExpressJS* makes request to the database.
* In the last stage, *MongoDB* (database) retrieves the data and sends the response to ExpressJS.
* ExpressJS returns the response to Nodejs and in turn to AngularJS and then displays the response to user.

**SET – 2**

**1.Define Directive? Explain the process of creating custom directives with example?**

**Ans:**

Angular 8 directives are DOM elements to interact with your application. Generally, directive is a TypeScript function. When this function executes Angular compiler checked it inside DOM element. Angular directives begin with ng- where ng stands for Angular and extends HTML tags with @directive decorator. Directives enables logic to be included in the Angular templates

Custom directives are used in AngularJS to extend the functionality of HTML. Custom directives are defined using "directive" function. A custom directive simply replaces the element for which it is activated. AngularJS application during bootstrap finds the matching elements and do one time activity using its compile() method of the custom directive then process the element using link() method of the custom directive based on the scope of the directive. AngularJS provides support to create custom directives for following type of elements.

* **Element directives** − Directive activates when a matching element is encountered.
* **Attribute** − Directive activates when a matching attribute is encountered.
* **CSS** − Directive activates when a matching css style is encountered.
* **Comment** − Directive activates when a matching comment is encountered.

Angular CLI provides a below command to create custom directive.

ng generate directive customstyle

After executing this command, you could see the below response:

CREATE src/app/customstyle.directive.spec.ts (244 bytes)

CREATE src/app/customstyle.directive.ts (151 bytes)

UPDATE src/app/app.module.ts (1115 bytes)

Open app.module.ts.

The directive will be configured in the AppModule through declarations meta data.

import { CustomstyleDirective } from './customstyle.directive';

@NgModule({

declarations: [

AppComponent,

TestComponent,

CustomstyleDirective ] })

Open customstyle.directive.ts file and add the below code:

import { Directive, ElementRef } from '@angular/core';

@Directive({ selector: '[appCustomstyle]' })

export class CustomstyleDirective {

constructor(el: ElementRef) { el.nativeElement.style.fontSize = '24px'; } }

Here, constructor method gets the element using CustomStyleDirective as el. Then, it accesses el’s style and set its font size as 24px using CSS property.

Finally, start your application (if not done already) using the below command:

ng serve

Now, run your application and you could see the below response:

**2.Explain the various methods to register a Service for utilizing a Dependency Injection?**

**Ans:**

To use Dependency Injection, every service needs to be registered into the system. Angular provides multiple option to register a service. They are as follows:

• ModuleInjector @ root level

• ModuleInjector @ platform level

• ElementInjector using providers meta data

• ElementInjector using viewProviders meta data

• NullInjector

**ModuleInjector @ root**

ModuleInjector enforces the service to used only inside a specific module. ProvidedIn meta data available in @Injectable has to be used to specify the module in which the service can be used. The value should refer to the one of the registered Angular Module (decorated with @NgModule). root is a special option which refers the root module of the application. The sample code is as follows:

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

@Injectable({

providedIn: 'root',

})

export class DebugService {

constructor() { }

}

**ModuleInjector @ platform**

Platform Injector is one level higher than ModuleInject and it is only in advanced and rare situation. Every Angular application starts by executing PreformBrowserDynamic().bootstrap method (see main.js), which is responsible for bootstrapping root module of Angular application. PreformBrowserDynamic() method creates an injector configured by PlatformModule. We can configure platform level services using platformBrowser() method provided by PlatformModule.

**NullInjector**

NullInjector is one level higher than platform level ModuleInjector and is in the top level of the hierarchy. We could not able to register any service in the NullInjector. It resolves when the required service is not found anywhere in the hierarchy and simply throws an error.

**ElementInjector using providers**

ElementInjector enforces the service to be used only inside some particular components. providers and ViewProviders meta data available in @Component decorator is used to specify the list of services to be visible for the particular component. The sample code to use providers is as follows:

ExpenseEntryListComponent

// import statement

import { DebugService } from '../debug.service';

// component decorator

@Component({

selector: 'app-expense-entry-list',

templateUrl: './expense-entry-list.component.html',

styleUrls: ['./expense-entry-list.component.css'],

providers: [DebugService]

})

Here, DebugService will be available only inside the ExpenseEntryListComponent and its view. To make DebugService in other component, simply use providers decorator in necessary component.

**ElementInjector using viewProviders**

viewProviders is similar to provider except it does not allow the service to be used inside the component’s content created using ng-content directive.

**ExpenseEntryListComponent**

// import statement

import { DebugService } from '../debug.service';

// component decorator

@Component({

selector: 'app-expense-entry-list',

templateUrl: './expense-entry-list.component.html',

styleUrls: ['./expense-entry-list.component.css'],

viewProviders: [DebugService]

})

Parent component can use a child component either through its view or content. Example of a parent component with child and content view is mentioned below:

Parent component view / template

<div>

child template in view

<child></child>

</div>

<ng-content></ng-content>

child component view / template

<div>

child template in view

</div>

Parent component usage in a template (another component)

<parent><!- - child template in content - -> <child></child> </parent>

Here,

• child component is used in two place. One inside the parent’s view. Another inside parent content.

• Services will be available in child component, which is placed inside parent’s view.

• Services will not be available in child component, which is placed inside parent’s content.

1. **Explain the process of creating and accessing the routes with an example?**

**Ans:**

Creating routes

Creating a route is simple and easy. The basic information to create a route is given below:

• Target component to be called.

• The path to access the target component.

The code to create a simple route is mentioned below:

const routes: Routes = [

{ path: 'about', component: AboutComponent },

];

Here,

• Routes is the variable in the AppRoutingModule.

• about is the path and AboutComponent is the target / destination component. When user requests http://localhost:4200/about url, the path matches with about rule and then AboutComponent will be called.

Accessing routes

Let us learn how to use the configured routes in the application. Accessing the route is a two step process. Include router-outlet tag in the root component template.

<router-outlet></router-outlet>

Use routerLink and routerLinkActive property in required place.

<a> routerLink=”/about” routerLinkActive=”active”>First Component </a>

Here,

• routerLink set the route to be called using the path.

• routerLinkActive set the CSS class to be used when the route is activated. Sometime, we need to access routing inside the component instead of template. Then, we need to follow below steps:

Inject instance of Router and ActivatedRoute in the corresponding component.

import { Router, ActivatedRoute } from '@angular/router';

constructor(private router: Router, private route: ActivatedRoute)

Here,

• Router provides the function to do routing operations.

• Route refers the current activate route.

Use router’s navigate function.

this.router.navigate(['about']);

Here, navigate function expects an array with necessary path information.

**4.How does Node.js work? Explain briefly with a neat diagram?**

**Ans:**

Node.js is a platform built on [Chrome's JavaScript runtime](https://code.google.com/p/v8/) for easily building fast and scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.

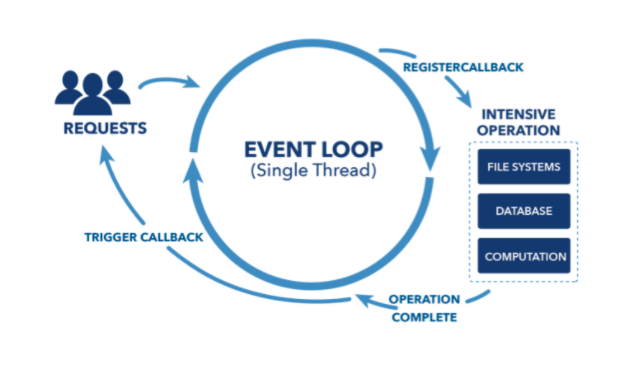
Node.js is an open source, cross-platform runtime environment for developing server-side and networking applications. Node.js applications are written in JavaScript, and can be run within the Node.js runtime on OS X, Microsoft Windows, and Linux.

Node.js also provides a rich library of various JavaScript modules which simplifies the development of web applications using Node.js to a great extent.

Node.js = Runtime Environment + JavaScript Library

Working of Node.js

* If you are familiar with JavaScript, then you know it is asynchronous and single threaded. The single thread is the event loop which is responsible for running all functions and requests. The asynchronous behavior is extremely important when using node, because it guarantees that the event loop is never blocked by a synchronous function.



* Even though there is only one event loop, when a request is made the loop passes the request to an asynchronous function which does the work.
* When this function is done and a response is returned, it can then be passed back to the event loop to be executed by the callback and sent to the user.
* If the functions were synchronous then the event loop would get locked up with one clients request and response, and all other clients would have to wait till that client was done. Because of the the asynchronous nature of JavaScript, the applications using node can handle many requests happening at the same time. This means that when programming in Node.js it is important to always keep in mind that the functions being written are not synchronous.
* It is also very important to catch errors on the server before it is passed back to the client. This prevents any errors from getting to the event loop which could crash the program and all clients would suffer.

SET- 3

**1.Explain the Process of creating a Observable and subscribing process in detail with an example?**

**Ans:**

Observables are **created** using new Observable or a creation operator, are **subscribed** to with an Observer, **execute** to deliver next / error / complete notifications to the Observer, and their execution may be **disposed**.

**Observer**

 An Observer is a consumer of values delivered by an Observable. Observers are simply a set of callbacks, one for each type of notification delivered by the Observable: next, error, and complete.

**Subscription**

 A Subscription is an object that represents a disposable resource, usually the execution of an Observable.

Creating Observables

The Observable constructor takes one argument: the subscribe function.

The following example creates an Observable

import { Observable } from 'rxjs';

// create observable

const simpleObservable = new Observable((observer) => {

// observable execution

observer.next('Hello Observable');

observer.complete();

});

// subscribe to the observable

simpleObservable.subscribe();

As you can see in the example observables are **created** by using the **new Observable()** call, then **subscribed** to by an observer, **executed** by calling the **next().**

*Observables can be created with new Observable. Most commonly, observables are created using creation functions, like of, from, interval, etc.*

In the example above, the subscribe function is the most important piece to describe the Observable.

Subscribing to Observables

The Observable observable in the example can be *subscribed* to, like this:

content\_copyopen\_in\_simpleObservable.subscribe(x => console.log(x));

It is not a coincidence that observable.subscribe and subscribe in new Observable(function subscribe(subscriber) {...}) have the same name. In the library, they are different, but for practical purposes you can consider them conceptually equal.

This shows how subscribe calls are not shared among multiple Observers of the same Observable. When calling observable.subscribe with an Observer, the function subscribe in new Observable(function subscribe(subscriber) {...}) is run for that given subscriber. Each call to observable.subscribe triggers its own independent setup for that given subscriber.

*Subscribing to an Observable is like calling a function, providing callbacks where the data will be delivered to.*

A subscribe call is simply a way to start an "Observable execution" and deliver values or events to an Observer of that execution.

**2.Explain the process of configuring the HTTP Client with an Example?**

**Ans:**

Configure Http client

HttpClient service is available inside the HttpClientModule module, which is available inside the @angular/common/http package.

To register HttpClientModule module:

Import the HttpClientModule in AppComponent

import { HttpClientModule } from '@angular/common/http';

Include HttpClientModule in imports meta data of AppComponent.

@NgModule({

imports: [

BrowserModule,

// import HttpClientModule after BrowserModule.

HttpClientModule,

]

})

export class AppModule {}

**3. Explain the Configuration of Angular Material with an example?**

**Ans:**

Angular Material provides a huge collection of high-quality and ready-made Angular component based on Material design. Let us learn how to include Angular material in Angular application and use its component.

## Configure Angular Material

Let us see how to configure Angular Material in Angular application.

Open command prompt and go to project root folder.

cd /go/to/expense-manager

Add Angular material package using below command −

ng add @angular/material

Angular CLI will ask certain question regarding theme, gesture recognition and browser animations. Select your any theme of your choice and then answer positively for gesture recognition and browser animation.

Installing packages for tooling via npm.

Installed packages for tooling via npm.

Choose a prebuilt theme name, or "custom" for a custom theme: Indigo/Pink [ Preview: https://material.angular.i

o?theme=indigo-pink ]

Set up HammerJS for gesture recognition? Yes

Set up browser animations for Angular Material? Yes

Angular material packages each UI component in a separate module. Import all the necessary module into the application through root module **(src/app/app.module.ts)**

import { MatTableModule } from '@angular/material/table';

import { MatButtonModule } from '@angular/material/button';

import { MatIconModule } from '@angular/material/icon';

@NgModule({

imports: [

MatTableModule,

MatButtonModule,

MatIconModule

]

})

Change the edit button using ExpenseEntryListComponent template (src/app/expense-entry-list/expense-entry-list.component.html) as specified below −

<div class="col-sm" style="text-align: right;">

<!-- <button type="button" class="btn btn-primary">Edit</button> -->

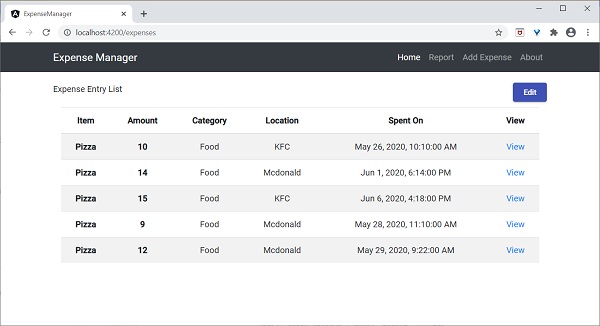
<button mat-raised-button color="primary">Edit</button>

</div>

Run the application and test the page.

ng serve

The output of the application is as follows −



Here, the application clearly shows the Angular Material button.

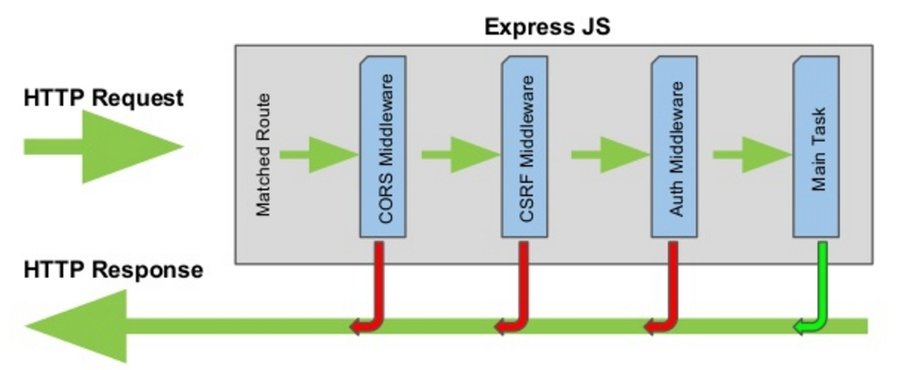
**4.Which are the most popular Node.js frameworks have used?**

**Ans:**

There are some frameworks of the node which you can use to build your applications. Some popular frameworks of node are…**Express.js, Socket.io, Koa.js, Meteor.js, Sail.js**.

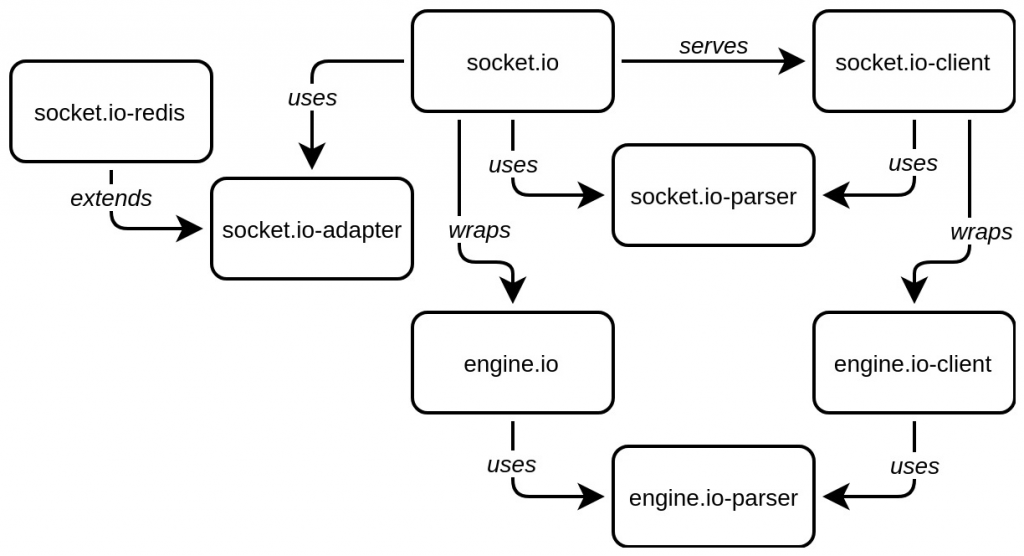
##### **Express.js**

[Express.js](https://wiredelta.com/technologies/frameworks/) is a node.js framework that’s designed for a developer to develop, build and run web apps quickly. The popular platform for developing node.js apps is now being extended to become a framework for building any other web app. But, a lot of developers have found it difficult to work with this. Because of its event-driven nature, when developers start reading instructions on how to do it, they will start to write all the code in callbacks which makes it difficult to maintain in the future.



##### **Socket.io**

[Socket.io](https://socket.io/) was made in 2010 by Guillermo Rauch. It is a very popular platform for creating web services that can be run in the browser. Socket.io provides real-time, bidirectional, and event-based communication. Many businesses use this framework including Microsoft and IBM. It is written in node.js and uses HTML5’s WebSockets to communicate, and the socket.io library is used for client and server.



##### **Sail.js**

[Sail.js](https://sailsjs.com/) was released in 2012 by Mike McNeil, it is a [Model View Controller (MVC)](https://wiredelta.com/cms-mvc-framework-comparison/) web application framework and authorizes a user to build an enterprise-ready application. MVC is a software design when organizing your codes, it is often used for developing the user interface. Sails have embedded features like API creator, and its socket integration which makes it very useful and speeds up development. Many industries use sails.js. However, below you can see that the software industry is the most popular in utilizing this framework.

##### **Koa.js**

[Koa.js](https://koajs.com/) was made by the team that also develop express.js, it is considered as the future generation of the node.js framework. The purpose of koa.js is for building client-side web applications. Koa.js allows a client to access features such as event delegation, callback channels, events, and to execute asynchronous code. The highlight and tools delivered by koa.js are to support programmers to accelerate the development of web applications and APIs.

##### **Meteor.js**

[Meteor](https://www.meteor.com/) is for building modern web and mobile applications. It is considered a full-stack JavaScript platform. This framework is simplified between the app and JavaScript. Its features are, it gives real-time updates, easy to understand for the developers, and organized documentation.