1. Difference between Angular2 and Angular4
2. What is Static Class in OOPS
3. Why Angular4 comes with TS where Angular1 is already running with JS
4. What type of storage will work on client end
5. What is the routing structure of Angular4
6. What is the LazyLoading in Angular4 and why we need that
7. if we want to get the API list from server and save it to the local for the next call where we can apply that code and what method we can use.
8. Explain Steps for Calling an Angular 4 Service in the Angular 4 Component class.
9. Explain Steps for creating an Angular 4 Service.
10. What are structural directives?
11. Is it possible to have a multiple router-outlet in the same template?
12. How to use Dependency Injection (DI) correctly in Angular 4?
13. Why type definition (.d.ts) with Typescript in Angular 4?
14. What is an Interface in TypeScript?
15. Why Angular 4? What's New in Angular 4?

1. In general, Angular4 is smaller and faster in comparison with Angular2 applications. For architecture, the upgrade of the version from Angular2 to Angular4 has reduced its bundled file size by 60%. In Angular4, animations now have their own package which Angular2 not have. Angular4 importing Animations module from @angular/platform-browser/animations. Angular4 has enhanced *ngFor and *ngIf from now onwards if/else style syntax to *ngIf while Angular2 has not this facility. The template is now ng-template which means we should use "ng-tmeplate" tag instead of "template" tag. The code generated is reduced and has accelerated the application development. Angular4 is compatible with newer versions TypeScript2.1 and TypeScript2.2 which helps with better type checking and also enhanced IDE features for Visual Studio Code.

2. Static keyword can be used with class, variable, method and block. Static members belong to the class instead of a specific instance, that is, we can access the static member without object. For example, in Java, a class can be made static only if it is a nested class. Nested static class does not need reference of Outer class. A static class cannot access non-static members of the Outer class and it can only access static members of Outer class.

3. TypeScript provides advanced autocompletion, navigation, and refactoring. Having such tools is almost a requirement for large projects. TypeScript is a superset of JavaScript, so we do not need to go through a big rewrite to migrate it. TypeScript support core ES2015 features as well as ES2016/2017 features like decorators, async/await and others. It provides supports for types and allows us to use classes and functional programming techniques. Besides, TypeScript code can be debugged directly in the browser as long as the proper map files are constructed during build time.

4. On client end, localStorage stores information as long as the user does not delete them, and sessionStorage stores information as long a the session goes until the user closes the browser/tab. Both of them stores string data with key-value storage type. Different browsers cannot share the information in localStorage or sessionStorage while different pages share the same localStorage.

5. Routing rules can navigate users to certain page when they click or input the link in browser. For example, we define a router pointing to a TestComonent component.
   import { RouterModule } from '@angular/router';

   ```
   RouterModule.forRoot([
       {
           path: 'test',
           component: TestComponent
       }
   ]}
   ```

   Route will map every URL to certain component to allow the navigation in different views.

6. Normally, Angular4 will load all components when starting up the application which makes the application slow when user visit the site at first time. Since we need to download all components at first time but what user can see is only home page, that is whey we need lazy loading to improve the performance of loading time of the application. Lazy loading means that we only load component what we need to use first starting up the application. If user navigate to a new page, the component for that page will load immediately and it load only for the first visit page, that is, it will not load when we revisit that page again.

7. We can apply the code into a service with $http service, we can request the API list on server and get the response from server to store it in local.

8. The first step requires importing the service at the top of the component.

```
import { TestService } from './test.service';
```

Next, within the constructor, we have to import it through dependency injection.

```
export class AppComponent {
    constructor(private testService:TestService) {
    }
}
```

Then, we can use testService to access its associated properties and methods.

Underneath the constructor() {}, we can add the ngOnInit() lifecycle hook, which runs when the components loads:

```
SomePropoerty:string = '';
        ngOnInit() {
                console.log(this.testService.cars);
                this.someProperty = this.testService.myCar();
}
```

Firstly, we are console logging the cars array, and then bind someProperty to the myCar() method that we defined in testService. In the template property of the @Component() decorator, add:

```
@Component({
        template: `
        <p>{{ someProperty }}</p>
        `
})
```

9. to create a service, we command at the console in the project folder type:
> ng g service test
Then the service can be installed and src/app/test.service.spec.ts and test.service.ts can be created.
We have to add it to the providers property of the NgModule decorator in src/app/app.module.ts with the following code:

```
import { TestService } from './test.service';

@NgModule({
        providers: [TestService],
})
```

10. Structural Directives are directives which change the structure of the DOM by adding or removing elements. After applying a structural directive to a host element, the directive then does whatever it's supposed to do with that host element and its descendants. We can prepend the directive name with * to sip having to define a <ng-template> and have the directive use the element it's attached to as the template.

11. Yes, it is possible to have a multiple router-outlet in the same template. We need to use aux routing and give a name to the router-outlet.

```
<router-outlet name="auxPathName"></router-outlet>
```

and then setup the routing config:

```
@RouteConfig([
        {path:'/', name: 'RegularPath', component: FirstComponent, useAsDefault: true},
        {aux:'/auxRoute', name: 'AuxPath', component: SecondComponent}
])
```

12. Dependency Injection (DI) is a way to create objects that depend upon other objects. Since Angular team has found a nice API that hides all the injector machinery when building components in Angular, we do not need to create injectors manually when we build Angular components.
For example, the following is a simple Angular component.

```
@Component({
        selector: 'app',
        template: '<h1>Hello</h1>'
})
class App {
        name = 'World';
}
```

We can extend this component by using a NameService that is used in the component's constructor.

```
class NameService {
        name = 'Pascal';
        getName() {
                return this.name;
        }
}
```

To make it available in application as an injectable, we need to pass some provider configurations to application's injector.

```
@NgModule({
        imports: [BrowserModule],
        providers: [NameService],
        declarations: [App],
        bootstrap: [App]
})
export class AppModule {}
```

To actually inject it, we use the tools, @Inject decorators.

```
class App {
        constructor(NameService: NameService) {
                this.name = NameService.getName();
        }
}
```

13. In Angular application, TypeScript complier can extract type APIs from *.ts files. However, we need to tell the TypeScript complier to expect "ambient values" that are provided outside of the scope of the known *.ts files. To implement this, we need to install a custom Typing file - *.d.ts in Angular application.

14. In TypeScript, an interface is a syntactical contact that an entity should conform to. In other words, an interface defines the syntax that any entity must adhere to. Interfaces

define properties, methods, and events which are the members of the interface. Interfaces contain only the declaration of the members.

15. Angular4 introduces a new “titlecase” pipe “l” to change the first letter of each word into uppercase. Angular4 adds search parameters to an “HTTP request”. A new service has been introduced to easily get or update “Meta Tags”. A new interface “paramMap” and “queryParamMap” has been added to represent the parameters of a URL. In conclusion, Angular4 is for more experienced developers while Angular2 is a bit confusing for those who are in the learning phase.