IDEAS2IT

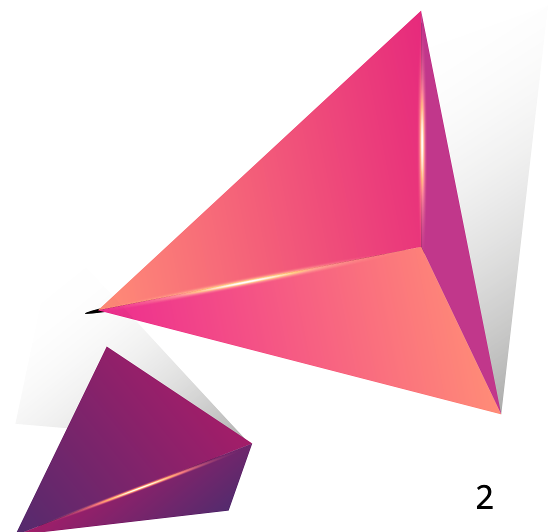
Best Practices we following in **Angular Development**

www.ideas2it.com

Angular is an open-source front-end framework developed by Google for creating dynamic, modern web applications. Introduced in 2009, Angular's popularity is based on its eliminating unnecessary code and ensuring lighter and faster apps. Having rapidly evolved from AngularJS in 2010, the front-end framework is today used by more than 40% of software engineers for creating user interfaces.

Angular helps build interactive and dynamic single page applications (SPAs) with its compelling features including templating, two-way binding, modularization, RESTful API handling, dependency injection, and AJAX handling. Designers can use HTML as template language and even extend HTML' syntax to easily convey the components of the application. You also don't need to rely on third-party libraries to build dynamic applications with Angular. Using this framework in your projects, you can reap multiple benefits.

While Angular Documentation is the right place to get to know best practices, this document focuses on other good practices that are not covered in the framework's documentation.



1. Use Angular CLI

Angular CLI is one of the most powerful accessibility tools available when developing apps with Angular. Angular CLI makes it easy to create an application and follows all the best practices! Angular CLI is a command-line interface tool that is used to initialize, develop, scaffold, maintain and even test and debug Angular applications.

So instead of creating the files and folders manually, use Angular CLI to generate new components, directives, modules, services, and pipes.

```
# Install Angular CLI

npm install -g @angular/cli
```

```
# Check Angular CLI version

ng version
```

2. Maintain proper folder structure

Creating a folder structure is an important factor we should consider before initiating our project. Our folder structure will easily adapt to the new changes during development.

```
-- app
     |-- modules
       |-- home
           |-- [+] components
           |-- [+] pages
           |-- home-routing.module.ts
           |-- home.module.ts
     |-- core
       |-- [+] authentication
       |-- [+] footer
       |-- [+] guards
       |-- [+] http
       |-- [+] interceptors
       |-- [+] mocks
       |-- [+] services
       |-- [+] header
       |-- core.module.ts
       |-- ensureModuleLoadedOnceGuard.ts
       |-- logger.service.ts
     |
```

```
|-- shared
         |-- [+] components
         |-- [+] directives
         |-- [+] pipes
         |-- [+] models
    |
    |-- [+] configs
|-- assets
    |-- scss
         |-- [+] partials
         |-- _base.scss
         |-- styles.scss
```

3. Follow consistent Angular coding styles

Here are some set of rules we need to follow to make our project with the proper coding standard.

- Limit files to 400 Lines of code.
- Define small functions and limit them to no more than 75 lines.
- Have consistent names for all symbols. The recommended pattern is feature.type.ts.
- If the values of the variables are intact, then declare it with 'const'.
- Use dashes to separate words in the descriptive name and use dots to separate the descriptive name from the type.
- Names of properties and methods should always be in lower camel case.
- Always leave one empty line between imports and modules; such as third party and application imports and third-party modules and custom modules.

4. Typescript

Typescript is a superset of Javascript, which is designed to develop large Javascript applications. You don't have to convert the entire Javascript code to Typescript at once. Migration from Javascript to Typescript can be done in stages.

Major benefits of Typescript include:

- Support for Classes and Modules
- Type-checking
- Access to ES6 and ES7 Features, before they are supported by major browsers
- Support for Javascript packaging and so on
- Great tooling support with IntelliSense

5. Use ES6 Features

ECMAScript is constantly updated with new features and functionalities. Currently, we have ES6 which has lots of new features and functionalities which could also be utilized in Angular.

Here are a few ES6 Features:

- Arrow Functions
- String interpolation
- Object Literals
- Let and Const
- Destructuring
- Default

6. Use trackBy along with ngFor

When using ngFor to loop over an array in templates, use it with a trackBy function which will return a unique identifier for each DOM item.

When an array changes, Angular re-renders the whole DOM tree. But when you use trackBy, Angular will know which element has changed and will only make DOM changes only for that element.

Use ngFor

```
<ul>
  <li *ngFor="let item of itemList;">{{item.id}}</li>
</ul>
```

```
<ul>
  <li *ngFor="let item of itemList;">{{item.id}}</li>
</ul>
```

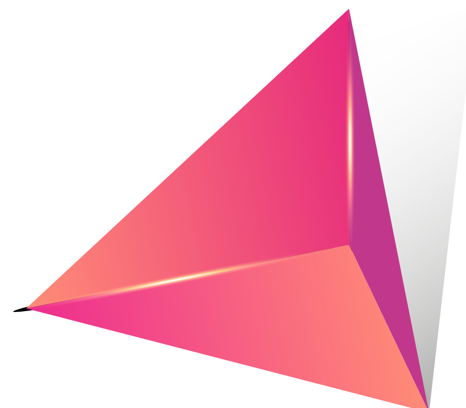
Now, each time the changes occur, the whole DOM tree will be re-rendered.

Using trackBy function

```
<ul>
  <li *ngFor="let item of itemList; trackBy: trackByFn">
    {{item.id}}
  </li>
</ul>
<button (click)="getItems()">Load items</button>
```

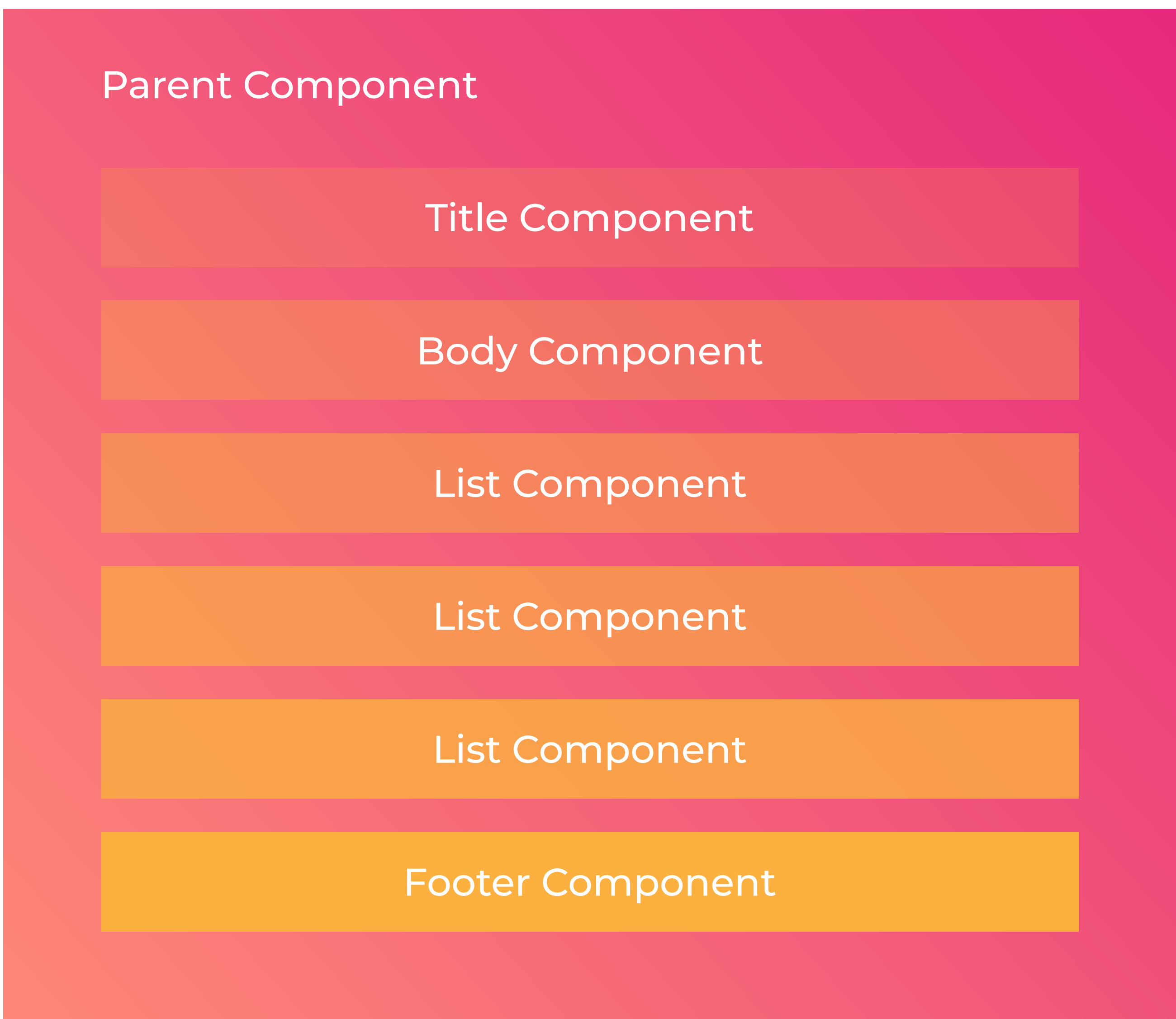
```
export class MyApp {

  getItems() { // load more items }

  trackByFn(index, item) {
    return index; // or item.id
  }
}
```

Now, it returns as a unique identifier for each item so only updated items will be re-rendered.



7. Break down into small reusable components

This might be an extension of the Single responsibility principle. Large components are very difficult to debug, manage and test. If the component becomes large, break it down into more reusable smaller components to reduce duplication of the code, so that we can easily manage, maintain and debug with less effort.



8. Use Lazy Loading

Try to lazy load the modules in an Angular application whenever possible. Lazy loading will load something only when it is used. This will reduce the size of the application load initial time and improve the application boot time by not loading the unused modules.

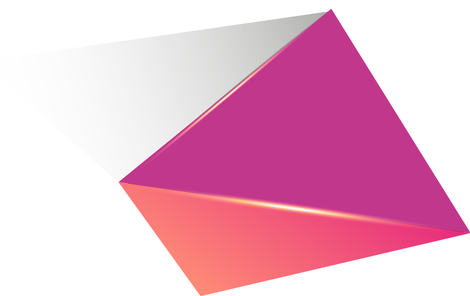
Without lazy loading

```
// app.routing.ts

import { WithoutLazyLoadedComponent } from
'./without-lazy-loaded.component';

{
  path: 'without-lazy-loaded',
  component: WithoutLazyLoadedComponent
}
```

With lazy loading

```
// app.routing.ts
{
  path: 'lazy-load',
  loadChildren: 'lazy-load.module#LazyLoadModule'
}
```

```
// lazy-load.module.ts

import { LazyLoadComponent } from './lazy-load.component';

@NgModule({
  imports: [
    RouterModule.forChild([
      {
        path: '',
        component: LazyLoadComponent
      }
    ])
  ],
  declarations: [
    LazyLoadComponent
  ]
})
export class LazyModule { }
```



9. Use Index.ts

index.ts helps us to keep all related things together so that we don't have to be bothered about the source file name. This helps reduce size of the import statement.

For example, we have /heroes/index.ts as

```
//heroes/index.ts

export * from './hero.model';
export * from './hero.service';
export { HeroComponent } from './hero.component';
```

We can import all things by using the source folder name.

```
import { Hero, HeroService } from '../heroes'; // index
is implied
```

10. Avoid logic in templates

All template logic will be extracted into a component. Which helps to cover that case in a unit test and reduce bugs when there is template change.

Logic in templates

```
// template
<span *ngIf="status === 'inActive' || 'hold'"> Status:
Unavailable </span>
```

We can import all things by using the source folder name.

```
// component
ngOnInit (): void {
  this.status = apiRes.status;
}
```

Logic in component

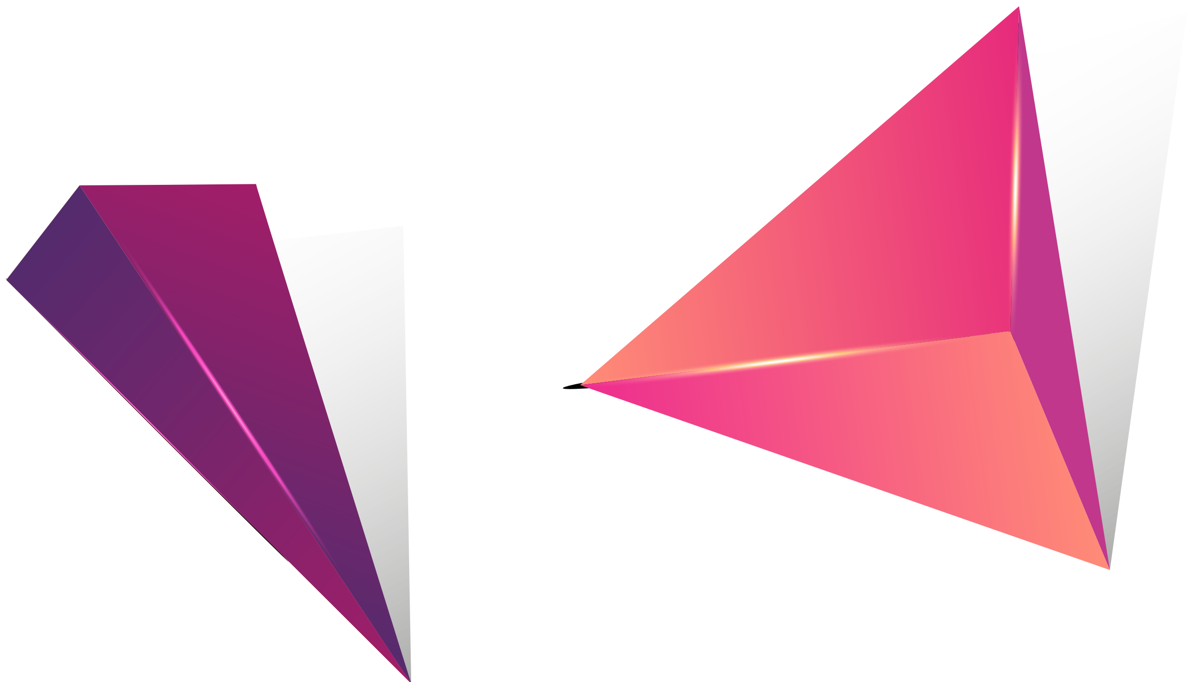
```
// template
<span *ngIf="isUnavailable"> Status: Unavailable </span>
```

```
// component
ngOnInit (): void {
  this.status = apiRes.status;
  this.isUnavailable = this.status === 'inActive' ||
'hold';
}
```

11. Cache API calls

When making API calls, responses from some of them do not change frequently. In those cases, we can add a caching mechanism and store the value from an API. When another request to the same API is made, we get a response from the check, if there is no value available in the cache then we make an API call and store the result.

We can introduce a cache time for some API calls the value change, but not frequently. Caching API calls and avoiding unwanted duplicate API calls improves speed of the application and also ensures we do not download the same information repeatedly.



12. Use async pipe in templates

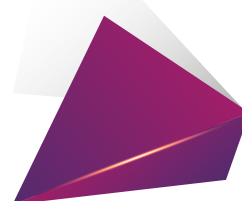
Observables can be directly used in templates with the async pipe, instead of using plain JS values. Under the hood, an observable would be unwrapped into plain value. When a component using the template is destroyed, the observable is automatically unsubscribed.

Without Using async pipe

```
//template
<p>{{ text }}</p>
```

Using async pipe

```
// template
<p>{{ text | async }}</p>

// component
this.text = observable.pipe(
 map(value => value.item)
);
```



13. Declare safe strings

The variable of type string has only some set of values and we can declare the list of possible values as the type. So the variable will accept only the possible values. We can avoid bugs while writing the code during compile time itself.

Normal String declaration

```
private vehicleType: string;

// We can assign any string to vehicleType.
this.vehicleType = 'four wheeler';
this.vehicleType = 'two wheeler';
this.vehicleType = 'car';
```

Safe string declaration

```
private vehicleType:  'four wheeler' | 'two wheeler';

this.vehicleType = 'four wheeler';
this.vehicleType = 'two wheeler';

this.vehicleType = 'car';  // This will give the below
error

Type '"car"' is not assignable to type '"four wheeler" |
"two wheeler"'
```

14. Avoid any type

Declare variables or constants with proper types other than any. This will reduce unintended problems. Another advantage of having good typings in our application is that it makes refactoring easier. If we had a proper typing, we would get a compile-time error as shown below:

```
interface IProfile {
  id: number;
  name: string;
  age: number;
}

export class LocationInfoComponent implements OnInit {
  userInfo: IProfile;

  constructor() { }

  ngOnInit() {
    this.userInfo = {
      id: 12345,
      name: 'test name',
      mobile: 121212
```

```
    }
  }
}
// Error

Type '{ id: number; name: string; mobile: number; }' is
not assignable to type 'IProfile'.
Object literal may only specify known properties, and
'mobile' does not exist in type 'IProfile'.
```

15. State Management

One of the most challenging things in software development is state management. State management in Angular helps in managing state transitions by storing the state of any form of data. In the market, there are several state management libraries for Angular like NGRX, NGXS, Akita, etc. and all of them have different usages and purposes.

We can choose suitable state management for our application before we implement it.

Some benefits of using state management.

- It enables sharing data between different components
- It provides centralized control for state transition
- The code will be clean and more readable
- Makes it easy to debug when something goes wrong
- Dev tools are available for tracing and debugging state management libraries

16. Use CDK Virtual Scroll

Loading hundreds of elements can be slow in any browser. But CDK virtual scroll support displays large lists of elements more efficiently. Virtual scrolling enables a performant way to simulate all items being rendered by making the height of the container element the same as the height of the total number of elements to be rendered, and then only rendering the items in view.

```
//Template
<ul>
  <cdk-virtual-scroll-viewport itemSize="100">
    <ng-container *cdkVirtualFor="let item of items">
      <li> {{item}} </li>
    </ng-container>
  </cdk-virtual-scroll-viewport>
</ul>
```

```
// Component
export class CdkVirtualScroll {
 items = [];
 constructor() {
   this.items = Array.from({length: 100000}).map((_, i)
=> `scroll list ${i}`);
 }


}
```

17. Use environment variables

Angular provides environment configurations to declare variables unique for each environment. The default environments are *development* and *production*. We can even add more environments, or add new variables in existing environment files.

Dev environment

```
// environment.ts environment variables
export const environment = {
  production: false,
  apiEndpoint: 'http://dev.domain.com',
  googleMapKey: 'dev-google-map-key',
  sentry: 'dev-sentry-url'
};
```

Dev environment

```
// environment.prod.ts environment variables
export const environment = {
  production: true,
  apiEndpoint: 'https://prod.domain.com',
  googleMapKey: 'prod-google-map-key',
  sentry: 'prod-sentry-url'
};
```

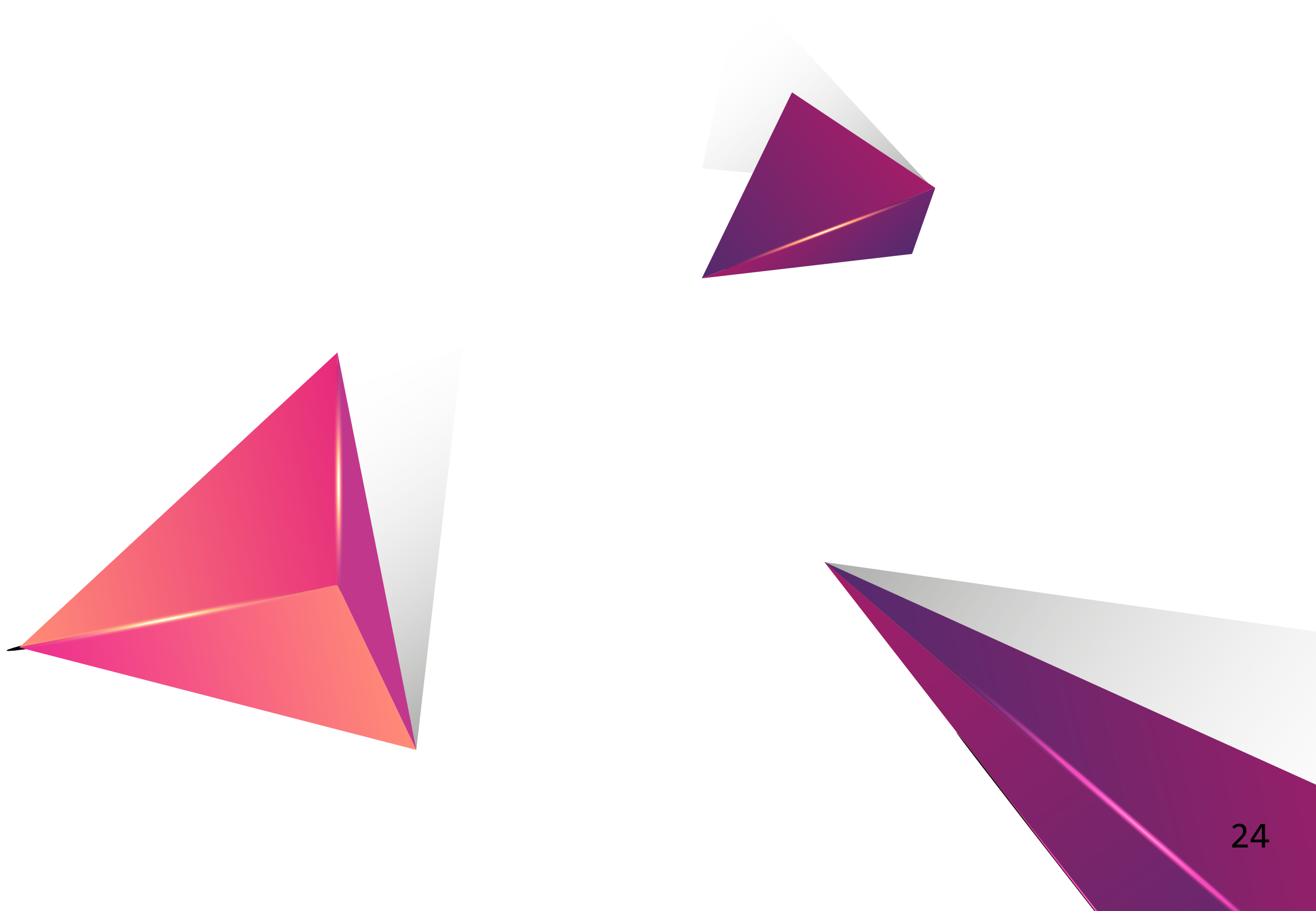
During the application build, the environment variables are applied automatically from the specified environment file.

18. Use lint rules for Typescript and SCSS

tslint and stylelint have various inbuilt options, it forces the program to be cleaner and more consistent. It is widely supported across all modern editors and can be customized with your own lint rules and configurations. This will ensure consistency and readability of the code.

Some inbuilt options in tslint: no-any, no-magic-numbers, no-debugger, no-console, etc.

Some inbuilt options in stylelint: color-no-invalid-hex, selector-max-specificity, function-comma-space-after, declaration-colon-space-after, etc.



19. Always Document

Always document the code as much as possible. It will help the new developer involved in a project to understand its logic and readability.

It is a good practice to document each variable and method. For methods, we need to define it using multi-line comments on what task the method actually performs and all parameters should be explained.

```
/**
* This is the foo function
* @param bar This is the bar parameter
* @returns returns a string version of bar
*/
function foo(bar: number): string {
    return bar.toString()
}
```

We can use an extension to generate comments.

TIP: 'Document this' is a visual studio extension that automatically generates detailed JSDoc comments for both TypeScript and JavaScript files.a

contact@ideas2it.com