WSA | Forward looking IT finishing school

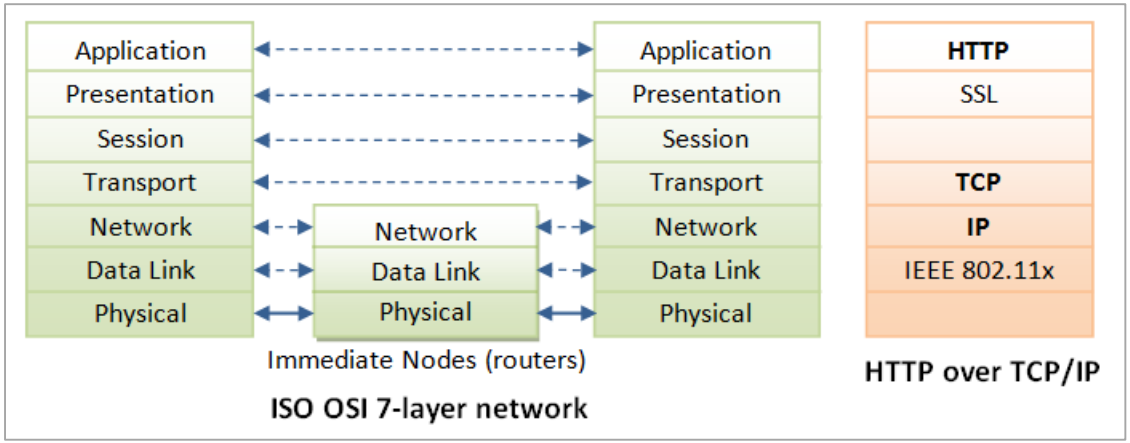
HTTP Services

Angular


WSA
Forward looking IT finishing school
HTTP Services
(Accessing services from the web)
www.webstackacademy.com

Introduction to HTTP

- Hyper Text Transfer Protocol (HTTP) is the client-server network protocol that has been in use by the World Wide Web (WWW) since 1990.
- HTTP protocol is defined in **RFC 7230** as per IETF standardization
- It is one of the application layer protocols in the TCP/IP suite of protocols.
- Whenever you are browsing the web, your browser will be sending HTTP request messages for various **Resources** (HTML pages, images etc..) and fetch them from the server



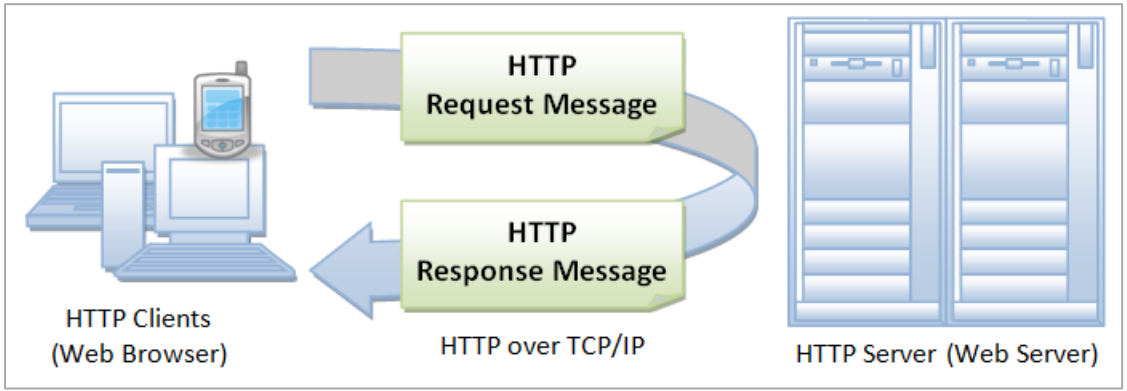
Introduction to HTTP

- Web servers handle these requests as a HTTP server by returning response messages that contain the requested resource. The client "pulls" the data from the server rather than the server "push"

- HTTP is a **Stateless** protocol. This means the current request does not know what has been done in the previous requests.

- HTTP permits negotiating of data type and representation, so as to allow systems to be built independently of the data being transferred.

- In summary we can say HTTP is a application-level protocol for distributed, collaborative, hypermedia information systems.



HTTP methods

HTTP supports a set of methods, out of which four are very important and frequently used.

Method	Description
`GET`	The GET method is used to retrieve information from the given server using a given URI.
`DELETE`	Removes all current representations of the target resource given by a URI
`PUT`	Replaces all current representations of the target resource with the uploaded content
`POST`	A POST request is sent by the client to the server with all the data collected in the client end

HTTP response code

Upon receiving HTTP request, the server provides appropriate responses. They are categorized as follows:

Code	Category	Description
1XX	Informational	Request received, server is continuing the process.
2XX	Success	The request was successfully received, understood, accepted and serviced.
3XX	Redirection	Further action must be taken in order to complete the request.
4XX	Client Error	The request contains bad syntax or cannot be understood.
5XX	Server Error	The server failed to fulfil an apparently valid request.

HTTP response code - Examples

Code	Category	Description
100	Continue	The server received the request and in the process of giving the response.
200	OK	The request is fulfilled.
301	Resource moved	The client should issue a new request to the new location.
400	Bad Request	Server could not interpret or understand the request, probably syntax error in the request message.
401	Authentication Required	The requested resource is protected, and require client's credential (username/password).
403	Forbidden	Server refuses to supply the resource, regardless of identity of client.
404	Not Found	The requested resource cannot be found in the server.
500	Internal Server Error	Server is confused, often caused by an error in the server-side program responding to the request.
501	Method Not Implemented	The request method used is invalid


WSA
Forward looking IT finishing school
REST Interfaces
(Web services with REST)
www.webstackacademy.com

REST Interfaces

- REST stands for **RE**presentational **S**tate **T**ransfer.
- REST is a web standards based architecture and that uses HTTP as the underlying protocol for communication.
- It has a notion of **"resource"** where everything revolves around that. The resource is accessed by a common interface using HTTP standard methods.
- REST was first introduced by Roy Fielding in year 2000.

RESTful Web Services

- A web service is a collection of open protocols and standards used for exchanging data between applications over the internet.

- Web services based on REST Architecture are known as RESTful Web Services. These web services use HTTP methods to implement the concept of REST architecture.

- Since HTTP offers two way communication methods, RESTful web service added a layer on top of it to implement meaningful web services using URI.

RESTful
Web Services

Characteristics of REST

- Three characteristics of REST:
  - **Stateless:** Client data is not stored on the server between interactions and the session is stored client-side (typically in session storage).
  - **Client <-> Server:** There is a **“separation of concerns”** between the frontend (client) and the backend (server). They operate independently of each other and both are replaceable.
  - **Cache:** Data from the server can be cached on the client, which can improve performance speed.
- In addition to these three fundamental features of REST, there is also a uniform approach to the composition of URLs. This allows for a standardization of service, which prior to the introduction of REST, did not exist.

Characteristics of REST

- For example, a `GET` request to `/courses`, should yield all the courses in the database, whereas a `GET` request to `/courses/20` would render the course with ID of 20.

- Similarly, REST utilizes standard methods like GET, PUT, DELETE and POST to perform actions.

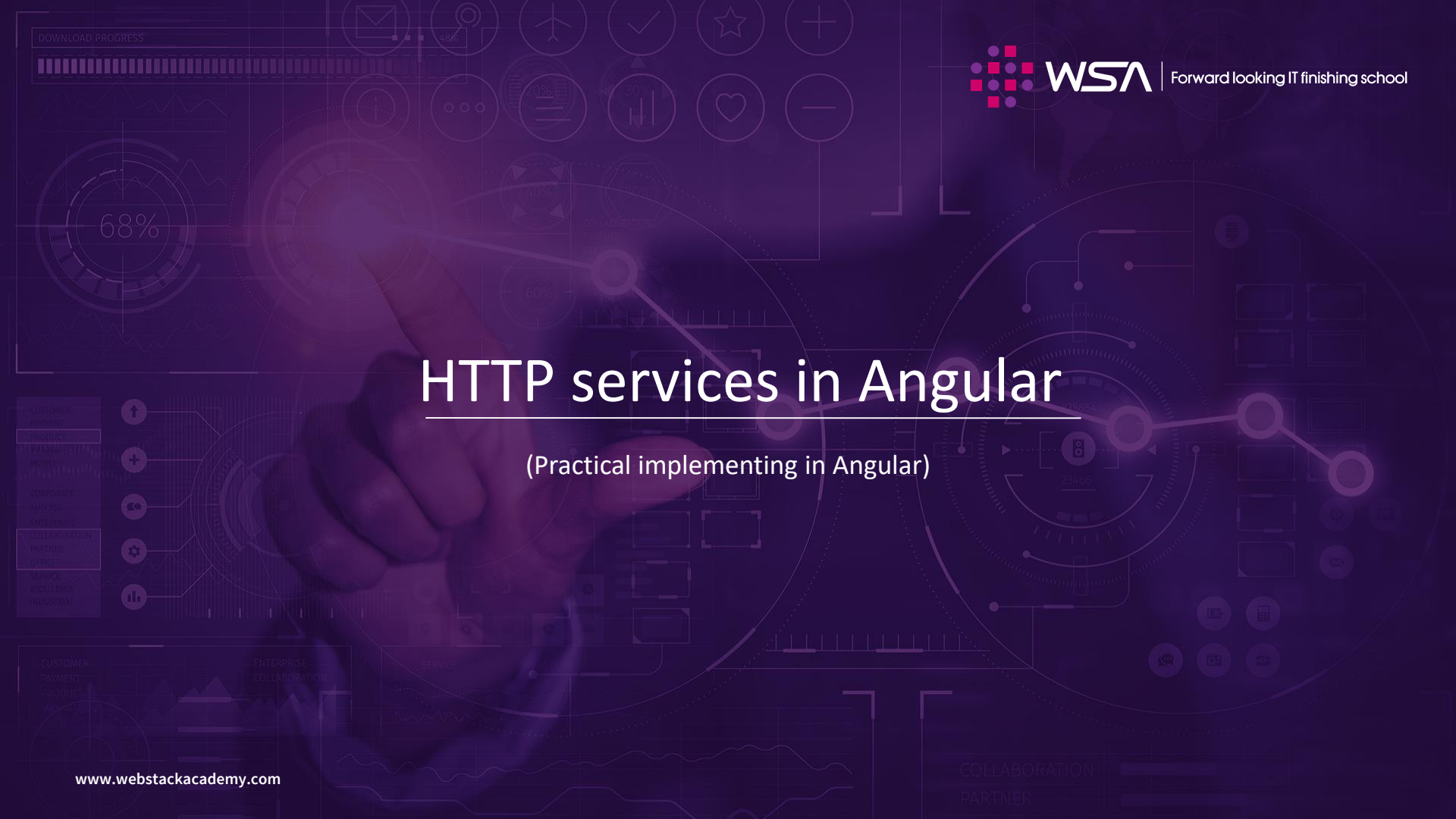
- Today every major web service provider expose RESTful interfaces (called as REST APIs) using which client applications can enable client application developers develop application with ease

A Brief about JSON

- **JSON:** JavaScript Object Notation.
- It is a lightweight data-interchange format, easy for humans to read and write.
- All programming languages support them, easy to parse and interpret
- One of the common standards used in data exchange for client-server communication. Servers as an alternative option for other data representations (ex: XML)
- Data is represented as name-value pairs and array data types.

JSON - Example

```
{
        "firstName": “WebStack",
        "lastName“ : “Academy",
        “domain“   : “Education”,
        "phoneNumber": [
        {
                "type": “landline",
                "number": “91-80-41289576"
        },
        {
                "type": “mobile",
                "number": “8095557334"
        }
        ],
}
```


WSA
Forward looking IT finishing school
HTTP services in Angular
(Practical implementing in Angular)
www.webstackacademy.com

1. Import HTTP into app.module.ts

Make the Angular and related import paths

```
import { HttpModule } from '@angular/http';

@NgModule({
declarations: [
        AppComponent,
        HttpComponent,
],
imports: [
        BrowserModule,
        HttpModule      // Import HTTP module here
],
providers: [
        EndPointServiceService
],
})
```

2. Accessing HTTP

Access your HTTP service via constructor and access methods

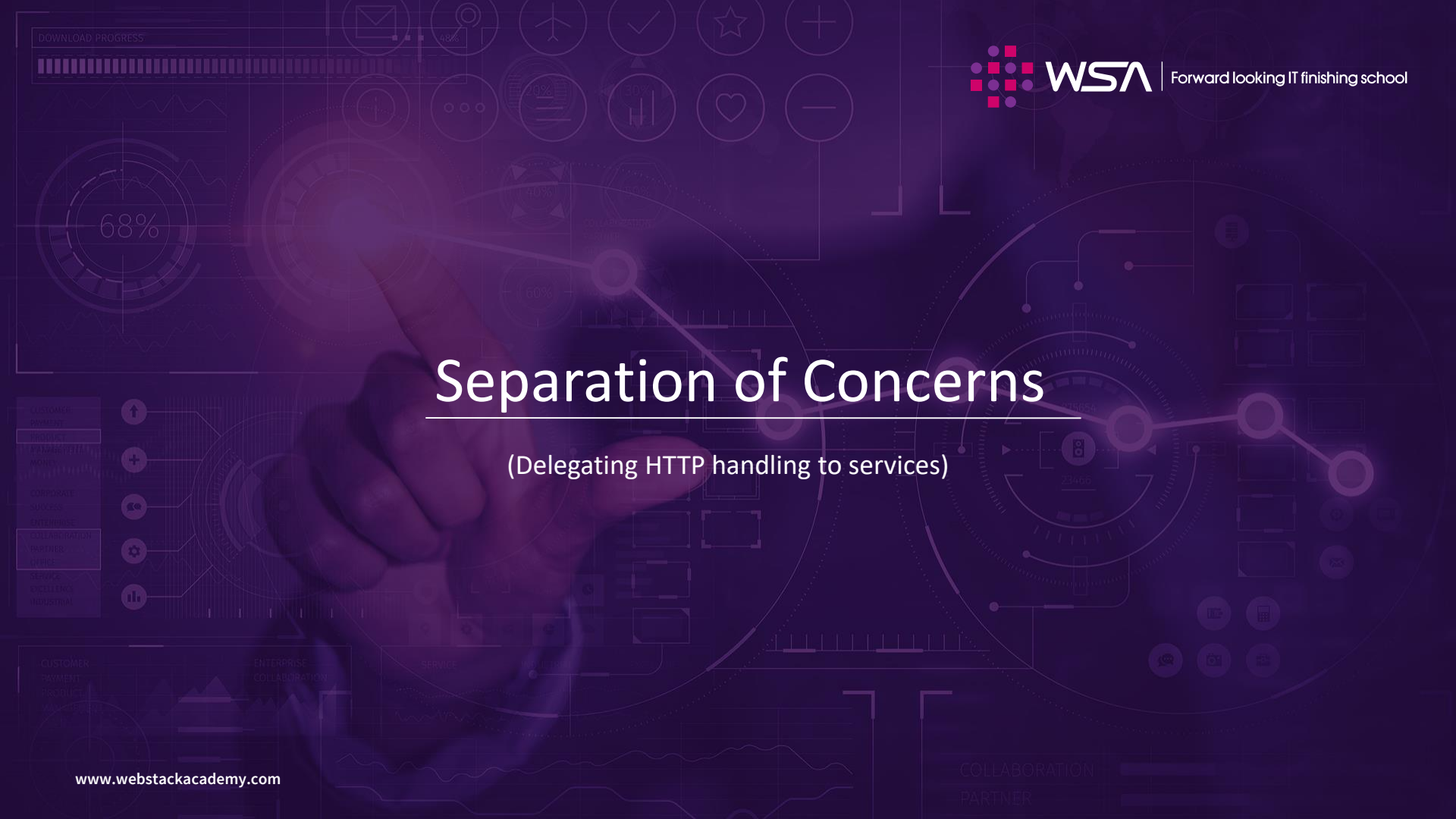
```
myPosts: any[];
private myURL = 'http://jsonplaceholder.typicode.com/posts';

constructor(private myHttp: Http) {

   // By default populate all posts....
   myHttp.get(this.myURL).subscribe(response => {
      console.log(response.json());
      this.myPosts = response.json();
   });

}
```

HTTP methods in Angular

Here are the HTTP method details in Angular

Method	Definition
GET	`get(url: string, options?: RequestOptionsArgs): Observable<Response>`
DELETE	`delete(url: string, options?: RequestOptionsArgs): Observable<Response>`
PUT	`put(url: string, body: any, options?: RequestOptionsArgs): Observable<Response>`
POST	`post(url: string, body: any, options?: RequestOptionsArgs): Observable<Response>`

- **NOTE:** All functions return Observarable, you need to subscribe to get actual return values / error codes
- Response type has got multiple fields, which can be obtained from Angular official documentation (https://angular.io/api/http/Response)



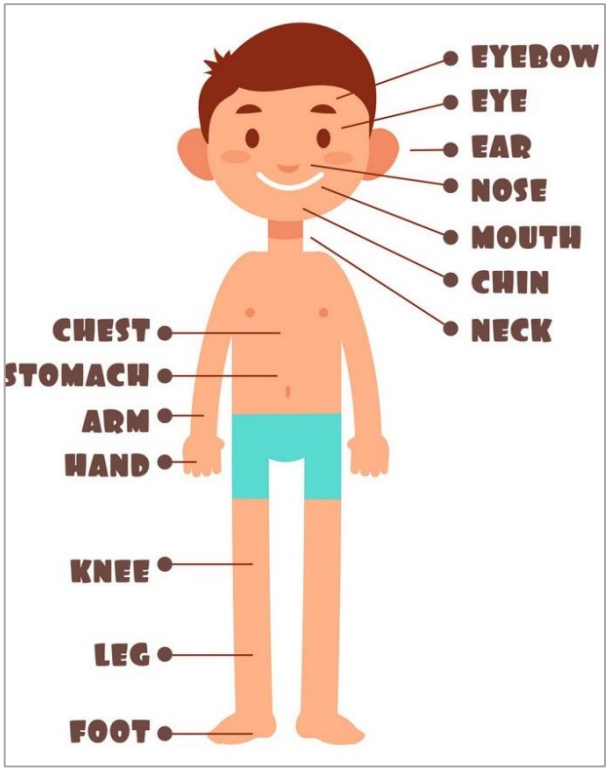
Separation of Concerns

(Delegating HTTP handling to services)

Separation of Concern (SoC) - Concept

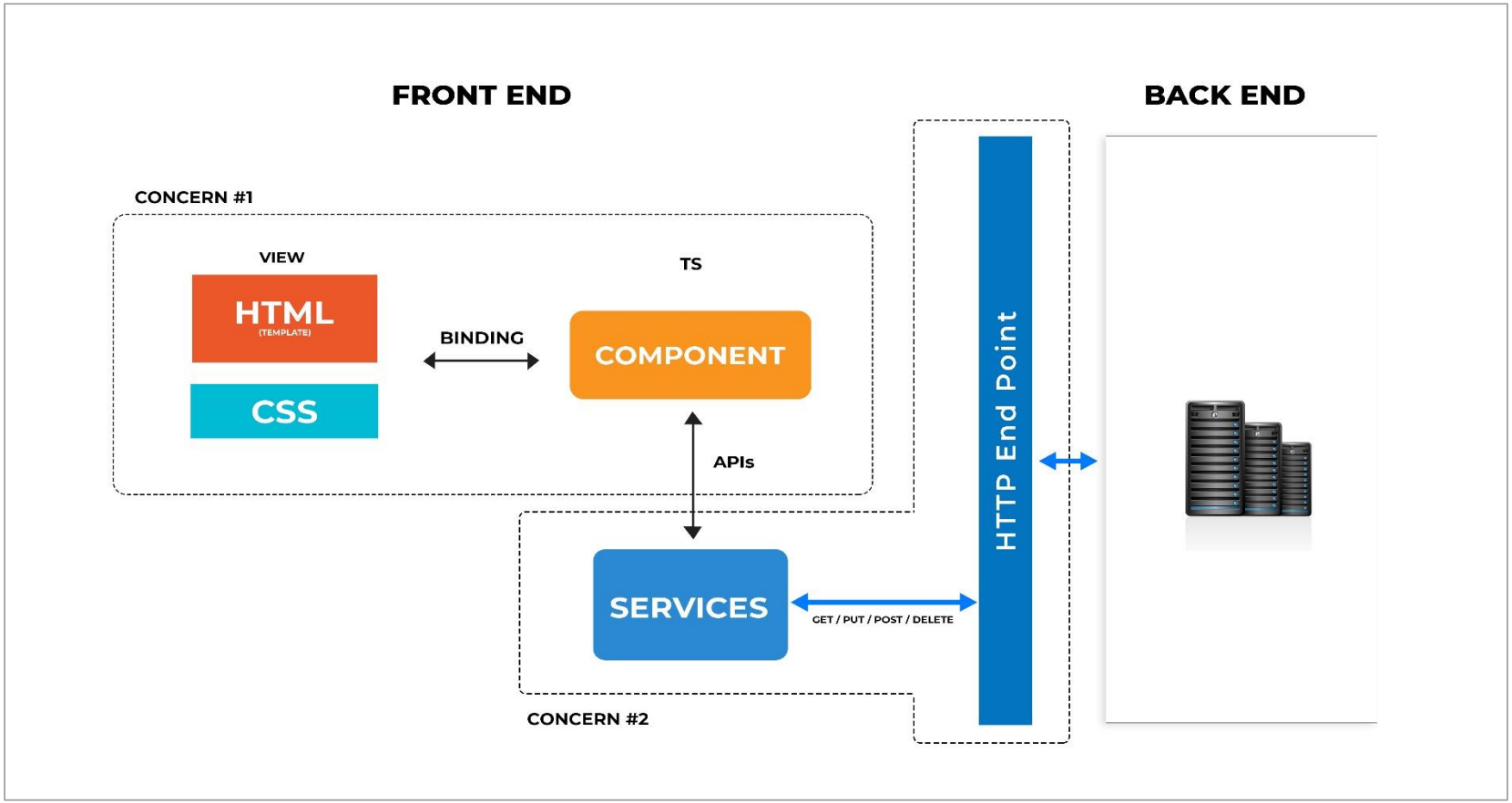
- In computer science, separation of concerns (SoC) is a design principle for separating a computer program into distinct sections, such that each section addresses a separate concern.
- A concern is a set of information that affects the code of a computer program (Functionality).
- It is always a good practise to implement SoC well in your program, so that is becomes modular.
- By making it modular we get other benefits in terms of re-usability, maintainability and simplifies development process.

Separation of Concern (SoC) - Implementation



- Practically it is achieved by encapsulating information inside a section of code that has a well-defined interface.
- The section of code which is interested, will invoke this interface and obtain required information
- This is also called as data encapsulation is a means of information hiding. The calling section will not know the detail orientation but only concerned about its objective
- Typically these interfaces are methods / APIs and implemented via various OOP mechanisms – Classes, Constructors etc..
- In Angular also we have used these concepts already!

Separation of Concern (SoC) in Angular



Separation of Concern (SoC) in Angular

- In Angular, we have seen separation of concern happening at component level as follows:
  - Template - HTML file
  - Style - CSS file
  - Business Logic - TS file
- The main goal of the component is to deal with business logic related to view (Template + Style)
- When it comes to dealing with HTTP services (ex: Fetching course list from Firebase) it should NOT be done as a part of component to adhere to SoC
- The idea is to implement a service and access them via a method. This way you are “delegating" the action to a service. In this process you also achieve benefits of SoC.

Separation of Concern (SoC) in Angular - Problem

```
export class HttpComponent implements OnInit {

myPosts: any[];
private myURL = 'http://jsonplaceholder.typicode.com/posts';

// Method for creating a new post
createNewPost(userTitle: HTMLInputElement) {

    let newPost = { title: userTitle.value };
    this.myHttp.post(this.myURL,JSON.stringify(newPost)).subscribe(
        response => {
                    console.log(response.json());
                    this.myPosts.splice(0,0,newPost);
        });
}
```

Separation of Concern (SoC) in Angular - Solution

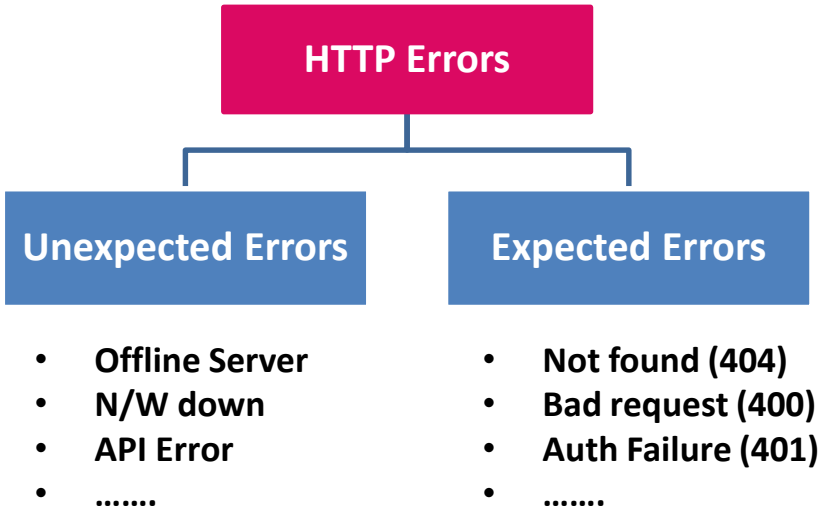
```
export class NewHttpComponentComponent implements OnInit {
    myPosts: any[];
    constructor(private service: EndPointServiceService) {

    }

    createNewPost(userTitle: HTMLInputElement) {

       let newPost = { title: userTitle.value };

       this.service.createPost(newPost).subscribe (
          response => {
            this.myPosts.splice(0,0,newPost);
       });
}
```

Error handling in HTTP

(Expected & Unexpected Errors)

HTTP Errors

- During the HTTP communication there could be many errors that could be happening
- The application should capture errors and handle them appropriately



HTTP Errors – Handling in Angular

```
deleteExistingPost (dPost){
    this.service.deletePost(dPost).subscribe (
        response => {
            console.log("Post deleted successfully");
            let dPostIndex = this.myPosts.indexOf(dPost);
            this.myPosts.splice(dPostIndex,1); },
        error => {
            if (error.status == 404)    // Expected Error
               alert ("Post already deleted");
            else {                      // Unexpected Error
               alert ("An unexpected error occurred");
               console.log(error);
            }
    });
}
```


WSA
Forward looking IT finishing school
Building re-usable services
(HTTP methods remain similar!)
www.webstackacademy.com

Building re-usable services

- In HTTP based services, the common operations remain the same
- In such cases each service may not want to implement the same methods. In order to achieve that services can be made re-usable using inheritance

Step-1: Create a new TS file (ex: generic.service.ts) and import all HTTP methods into it

```
export class GenericService {
constructor(myURL: string, myHttp : Http) {

}
  getPosts() { … }
  createPost(userPost) { … }
  updatePost(userPost) { … }
  deletePost(userPost) { … }

 }
}
```

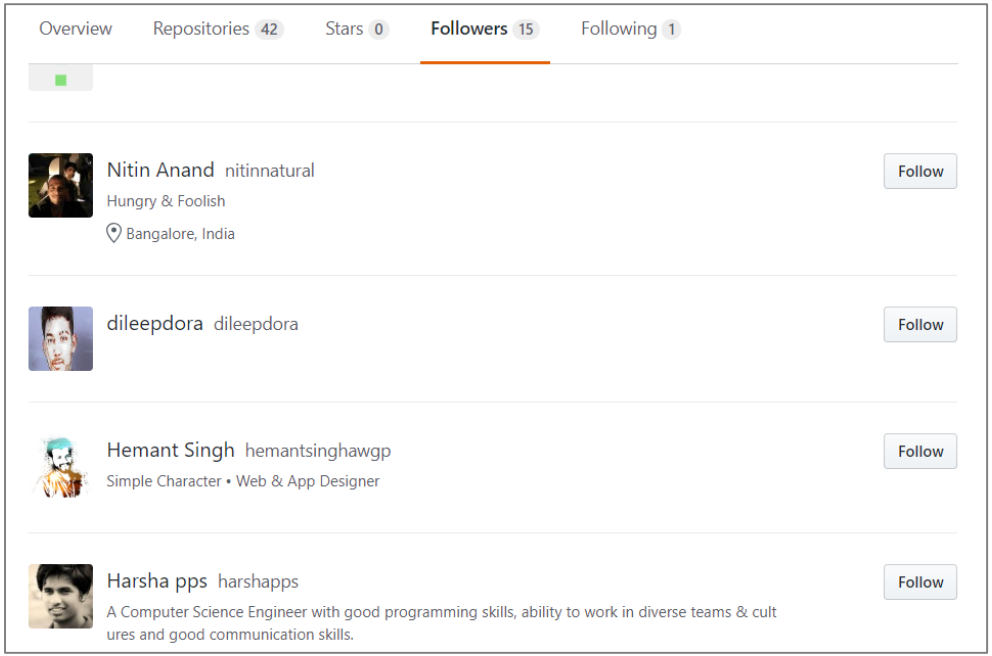
Building re-usable services

Step-2: From the service's main class, inherit this class. Also use constructor of the parent to pass parameters and initialization.

```
export class EndPointServiceService extends GenericService {

super(myURL, myHttp);

}
```

Exercise

- Create a GITHub followers page by accessing REST API provided by the GIT
  - Example URL: https://api.github.com/users/devendradora/followers



Exercise

- **Ensure you apply following learnings:**
  - New component to handle the main business logic
  - New re-usable service interacting with REST API
  - HTTP error handling
  - Directives for rendering all followers
  - CSS changes to display the follower names in a user friendly way
  - Display the following attributes:
    - ✓ Avatar (Profile picture)
    - ✓ GIT profile URL
    - ✓ Login name

WSA | Forward looking IT finishing school

WebStack Academy

#83, Farah Towers,
1st Floor, MG Road,
Bangalore – 560001

M: +91-809 555 7332
E: training@webstackacademy.com

WSA in Social Media:

www.webstackacademy.com