

LECTURE 5

PLATFORM BASED DEVELOPMENT

ACKNOWLEDGEMENTS

- A bunch of Websites where I took all the pictures without asking. Apologies :)

TODAY'S LECTURE

@FRANCISCOVILCHEZV

- HTTP Methods
 - GET
 - POST
 - PUT
 - DELETE
- More details of HTTP requests
 - Request headers
 - Params
 - Query String
 - Data Payload
- Angular routes

HTTP METHODS

HTTP METHODS

REMEMBER

- Como recordamos, HTTP es un protocolo basado en TCP, utilizado para transferir Hipertexto.
- La información requerida por el protocolo HTTP para transferir la información incluye:
 - IP/Hostname
 - Puerto
 - URL (o Endpoint)
 - HTTP Method



HTTP METHODS

DEFINICIÓN

- El método HTTP es utilizado para especificar el tipo de acción que desea realizarse con la petición HTTP.
- La lista completa de métodos HTTP puede ser encontrada en la siguiente [url](#).
- En este curso, nos limitaremos al uso de los siguientes métodos:
 - GET: Obtener data
 - POST: Insertar data
 - PUT: Actualizar data
 - DELETE: Borrar data



HTTP METHODS

EXAMPLE

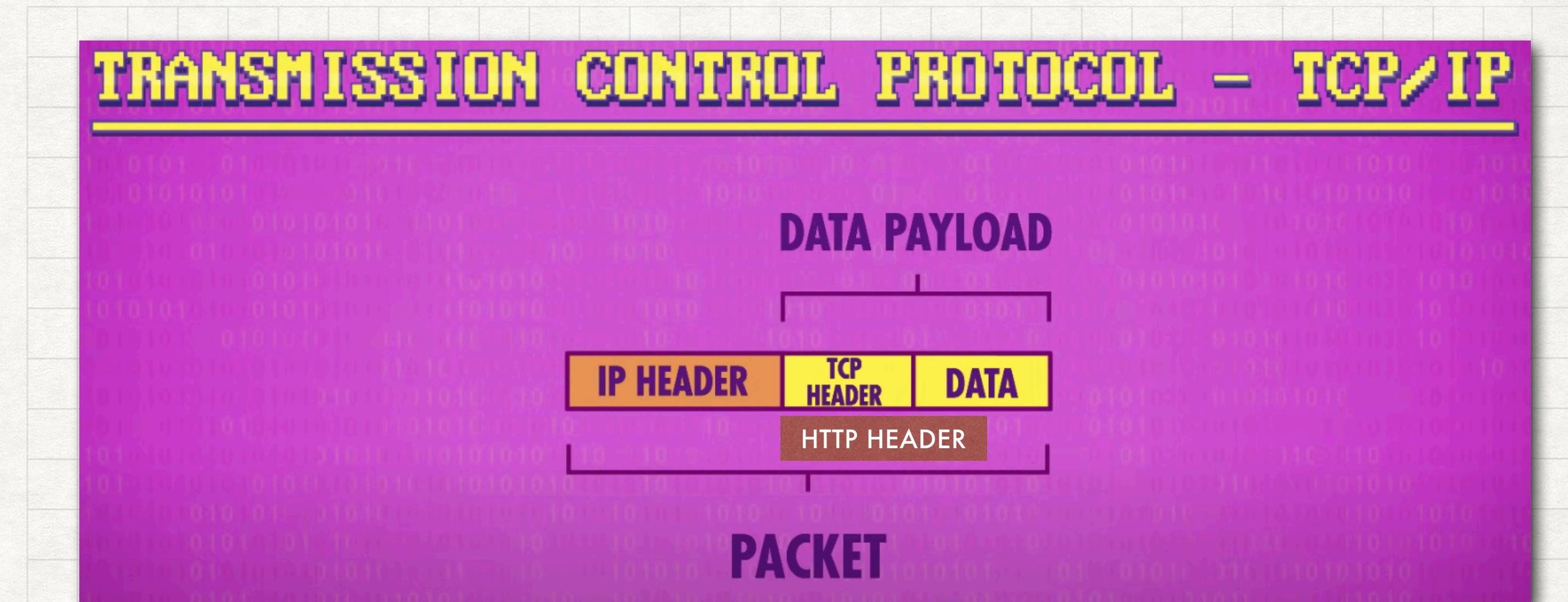
- Siguiendo el caso visto en el lab sobre el Club De Ajedrez
 - GET /members: Debería devolver los miembros del club de ajedrez
 - POST /members: Debería usarse para insertar un nuevo miembro del club.
 - PUT /members: Debería usarse para actualizar información de un miembro.
 - DELETE /members: Debería usarse para borrar a un miembro del club.
- Durante el desarrollo de nuestras APIs, tendremos la libertad de realizar cualquier funcionalidad deseada a cualquier método. En otras palabras, nuestra API `/DELETE members` no necesariamente estará obligada a borrar a un miembro. Sin embargo, es considerada una buena práctica que el código realice lo correspondiente al método HTTP, y ello deberá ser respetado durante el desarrollo del curso.

GETTING DEEP INTO HTTP

GETTING DEEP INTO HTTP HEAD

MORE DATA? (1)

- Adicionalmente a la URL y al HTTP method, existen otros campos adicionales incluidos en el header de la petición HTTP, entre ellos:
 - Request Header: Espacio reservado para mandar información sobre User-Agent, Cookies, Host, etc...
 - Toda esa información puede ser accedida por nosotros dependiendo del framework que usamos. Express lo recibe en una variable `req`.



```
app.get('/members', getMembers);

function getMembers(req, res){
  console.log(req.headers) // access http headers in the request
```

GETTING DEEP INTO HTTP

MORE DATA? (2)

- **Query string:** Se refiere a información que puede ser incluida en la URL, por ejemplo: /members?name=Francisco&country=PE
- **Params:** No es propiamente un atributo de HTTP, pero en algunos frameworks se le refiere a algunos campos pasados en la URL para referirse a un recurso en específico, por ejemplo: /members/23 para referirse al member número 23.
- **Data payload:** Es data enviada en la petición HTTP. Tiene el mayor espacio reservado y es utilizado generalmente en peticiones POST y PUT para mandar la data que deseé agregarse/cambiarse. También es conocido como body.

```
console.log(req.query) // access query string in the request  
console.log(req.params) // access params  
console.log(req.body) // access body (data payload)
```

GETTING DEEP INTO HTTP

REQUEST DATA

- Durante la definición de nuestras APIs, usualmente usaremos los siguientes atributos del HTTP header dependiendo del HTTP method que estamos definiendo.
Por ejemplo:
 - GET: querystring y/o params
 - POST: body (data payload)
 - PUT: params y body
 - DELETE: params

GETTING DEEP INTO HTTP

MORE ABOUT DATA PAYLOAD

- Existen algunos estándares para el envío de información a través del payload (body).
- Los dos formatos más usados son XML y JSON.
- El tipo de formato en el cual se está enviando la información está definido en el Request Header por un atributo conocido como Content-type.
- Durante el desarrollo de nuestra api, es probable que tengamos que especificar el formato en el cual estamos recibiendo/enviando nuestra data.
- Dependiendo del framework utilizado, es probable que haya un tipo de formato predeterminado, o que el mismo framework lo infiera.

XML

```
1 <member>
2   <fullname>Francisco Vilchez</fullname>
3   <email>fvilchez@utec.edu.pe</email>
4   <rating></rating>
5   <birthday>31/02/1981</birthday>
6 </member>
7
8
```

JSON

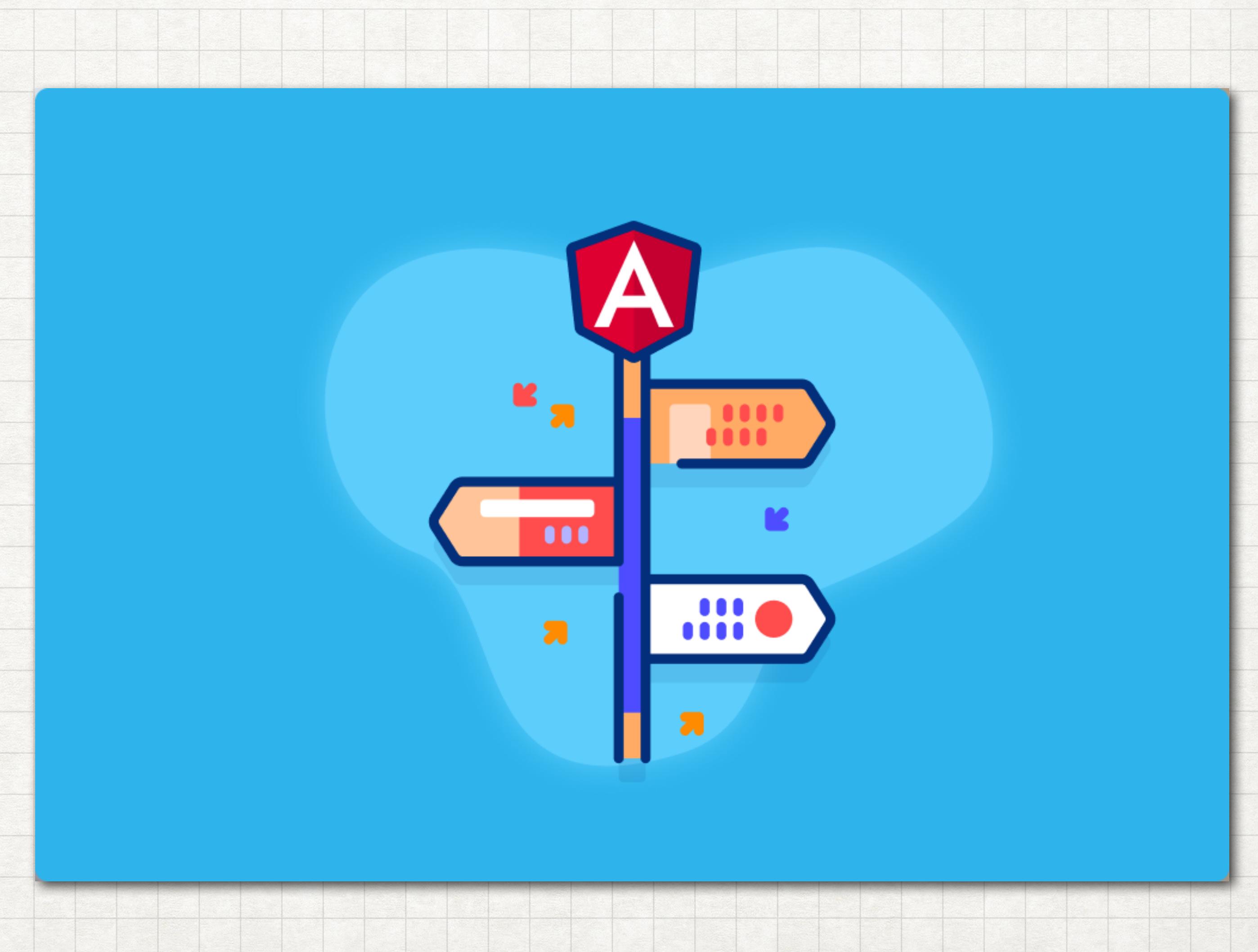
```
1 {
2   "member": {
3     "fullname": "Francisco Vilchez",
4     "email": "fvilchez@utec.edu.pe",
5     "rating": "",
6     "birthday": "31/02/1981"
7   }
8 }
```

ANGULAR ROUTES

ANGULAR ROUTES

INTRODUCTION

- Una página web generalmente te provee muchas vistas, las cuales son accedidas por distintas URLs.
- Por ejemplo, la página web google.com, te ofrece:
 - News: google.com/news
 - Maps: google.com/maps
 - Gmail: google.com/drive
 - Others
- Angular te permite definir qué URLs quieres usar en tu proyecto.



ANGULAR ROUTES

USAGE

- Cada route de tu aplicación Angular te permite especificar qué componentes quieres mostrar cuando el usuario navegue por una URL determinada.
- El tag <router-outlet> en tu HTML principal será reemplazado por el código HTML, CSS y JS del componente especificado.
- Por ejemplo, mi aplicación Web puede tener una cabecera o título que es global para todos los componentes, y el contenido en el <router-outlet> cambiaría para cada componente.

```
0 |  const routes: Routes = [
1 |  { path: '', redirectTo: 'welcome', pathMatch: 'full' },
2 |  { path: 'welcome', component: WelcomeComponent },
3 |  { path: 'members', component: MembersComponent },
4 |  { path: 'login', component: LoginComponent },
5 |  { path: 'members/new', component: NewMembersComponent }
6 | ];
```

```
<> app.component.html M X
src > app > <> app.component.html > ⚙ router-outlet
1 |   <h1>My app</h1>
2 |
3 |   <router-outlet> </router-outlet>
```

THANKS