Angular

Lecture 3

Agenda

- Recap last lecture topics
- Sharing data between components
- Angular Material
- Ng bootstrap
- Pipes
- Custom pipes

*ngIf then and else

<div *ngIf="condition; then thenBlock else elseBlock">

<ng-template #thenBlock>Content to render when condition is true.</ng-template>

<ng-template #elseBlock>Content to render when condition is false.</ng-template>

More info for *ngIf

https://angular.io/api/common/NgIf

https://codecraft.tv/courses/angular/built-in-directives/ngif-and-ngswitch/

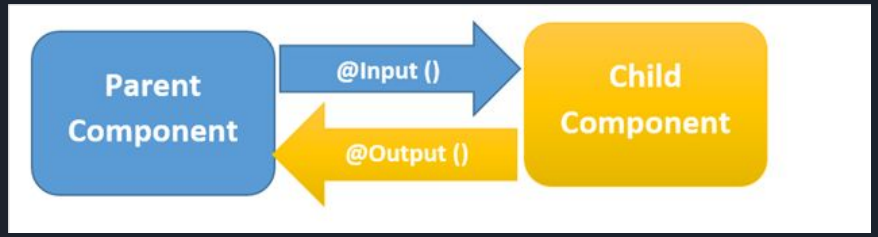
Sharing Data between Components

Sharing data between components

Parent to child :

This is probably the most common and straightforward method of sharing data. It works by using the @Input() decorator to allow data to be passed via the template.

Example : `@Input() Message: string;`



Sharing data between components

Child to parent

Using **output and event emitter** is a way to share data is to emit data from the child, which can be listed to by the parent. This approach is ideal when you want to share data changes that occur on things like button clicks, form entires, and other user events.

Example :

- `<app-child (messageEvent)="receiveMessage($event)"></app-child> - parent`
- `receiveMessage($event) {this.message = $event}- parent.ts`
- `@Output() messageEvent = new EventEmitter<string>();- Child.ts`
- `sendMessage() {this.messageEvent.emit(this.message)} - Child.ts`
- `<button (click)="sendMessage()">Send Message</button> - Child.html`

Sharing data between components

Child to parent

Another way to share from child to parent using **ViewChild** that allows a one component to be injected into another, giving the parent access to its attributes and functions. One caveat, however, is that child won't be available **until after the view has been initialized.** This means we need to implement the AfterViewInit lifecycle hook to receive the data from the child.

```
@ViewChild(ChildComponent) child;
```

Sharing data between components

Unrelated Components

- When passing data between components that lack a direct connection, such as siblings, grandchildren, etc, you should you a shared service.
- And you can also create a service to set and get values across unrelated components.

Will be covered in details later with services

Pipes

Pipes

Pipes are simple functions you can use in template expressions to accept an input value and return a transformed value.

Angular provides built-in pipes for typical data transformations like :

- DatePipe: Formats a date value according to locale rules.
  - `{{ today | date : 'MMMM YYYY' }}`
- UpperCasePipe: Transforms text to all upper case.
  - `{{ name | uppercase }}`
- LowerCasePipe: Transforms text to all lower case.

For More Info : https://angular.io/guide/pipes

Custom Pipes

- To generate custom pipe you need to run :
  - `ng generate pipe pipeName`

For example you can generate custom pipe that transform file size to MB :

```
transform(value: any, ...args: any[]): unknown {
   return (value / (1024 * 1024)).toFixed(2) + 'MB';
 }
```

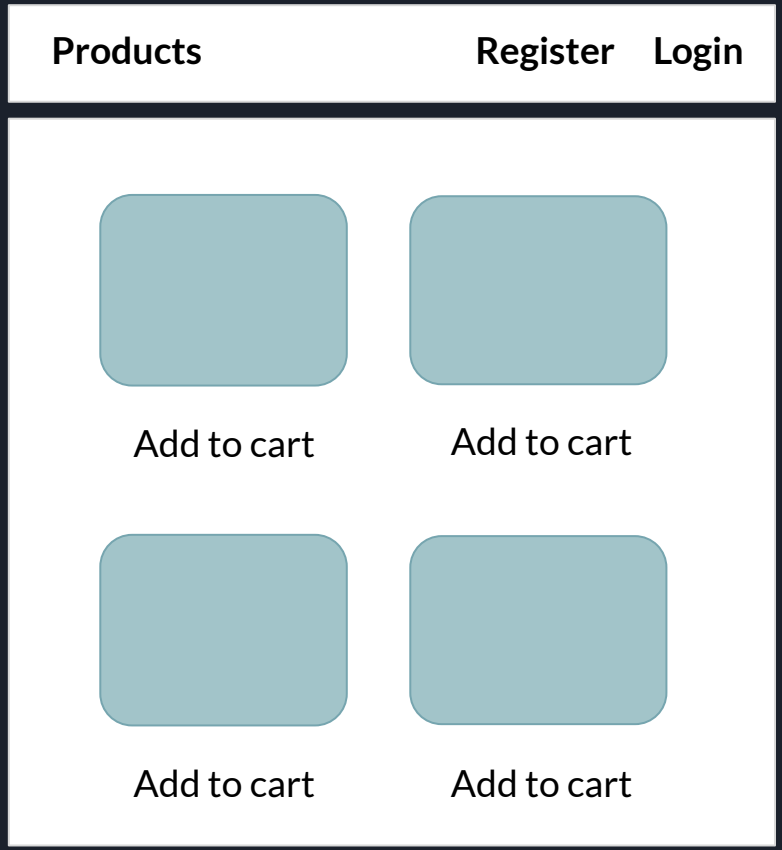
Thank you

Lap

Products

Create a new project with Routing Module for products.

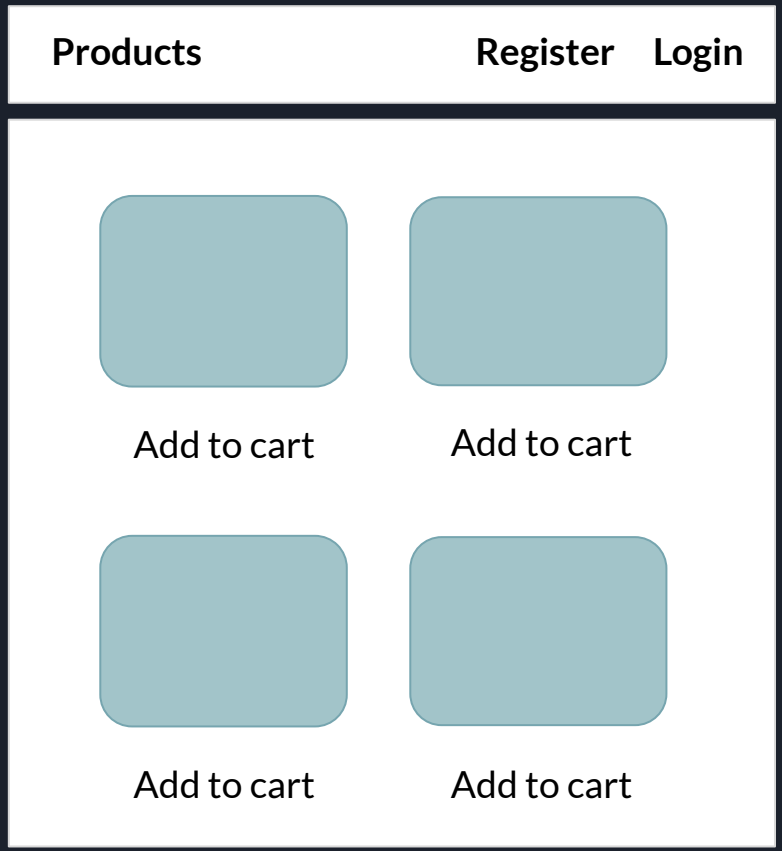
- Create a separate components for navbar and product list and product card
- Render Products cards from the provided products array [attached]
- Each product card is separate component so you will need to pass data to it with one of the ways the components could share data between them
- If key [count_in_stock] == 0 value then show text ‘not available in stock’ with red color if more than 0 then show ‘In stock’ with green color [custom pipe]



Products

Each product card item should have:

- Product images
- Product name
- Product category
- Product price [Use pipes to format the price to be shown as the following : **20 EGP**]
- “Add to cart” button



To do app

Create a to do app to create self notes with the following features:

- Red is Parent Component for the To-Do
- Yellow borders are child components one for the input with add button [which will emit the input value to the parent component]
- Other yellow border is for the other child component of the items list]
- User can add new task
- User can delete Task
- User can mark as completed and when mark as completed will be marked with linethrough.