Slide Note

Which of the following checkout processes provide a native app experience?

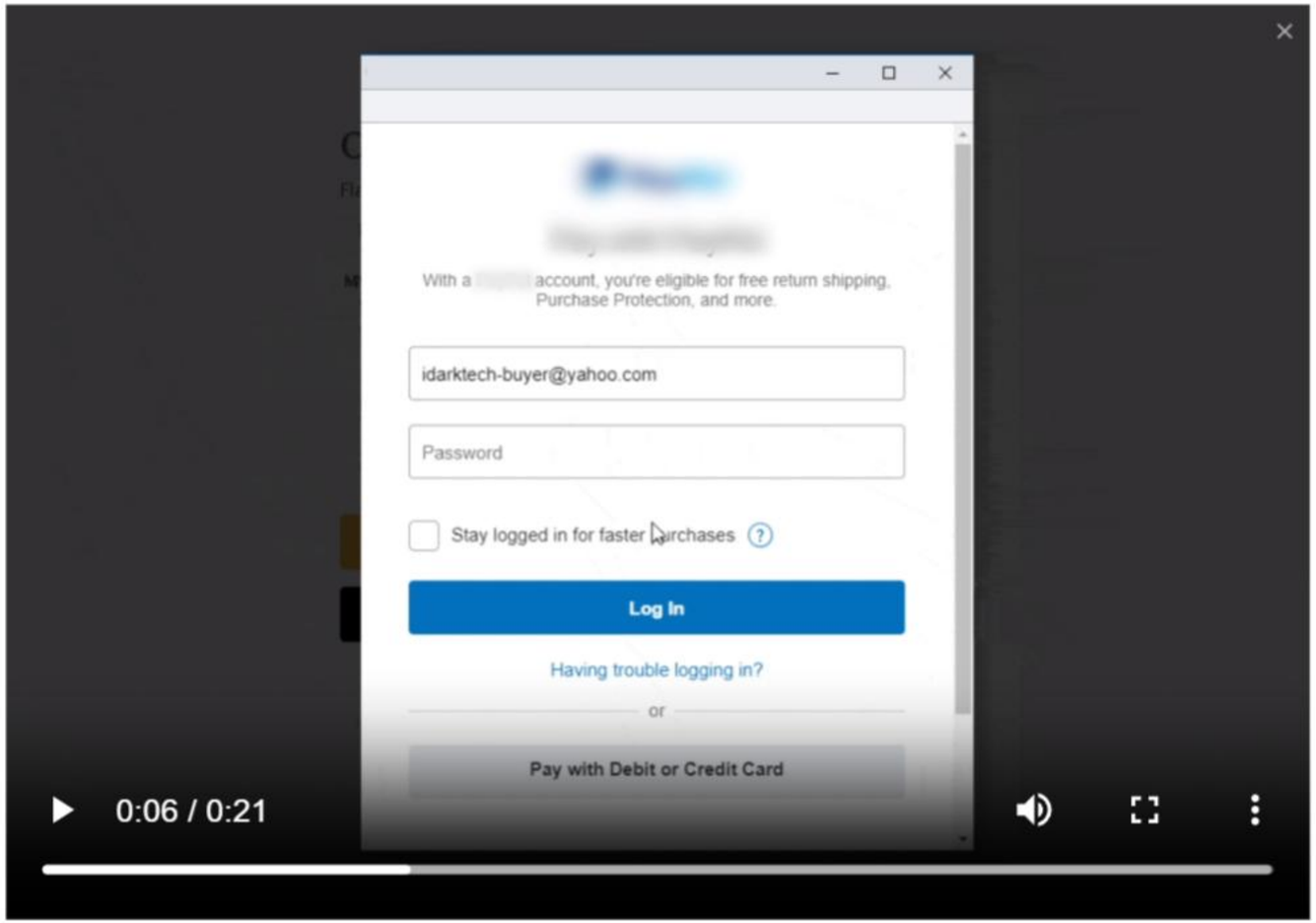
Menu

00:01:00:00:45

01/ 46

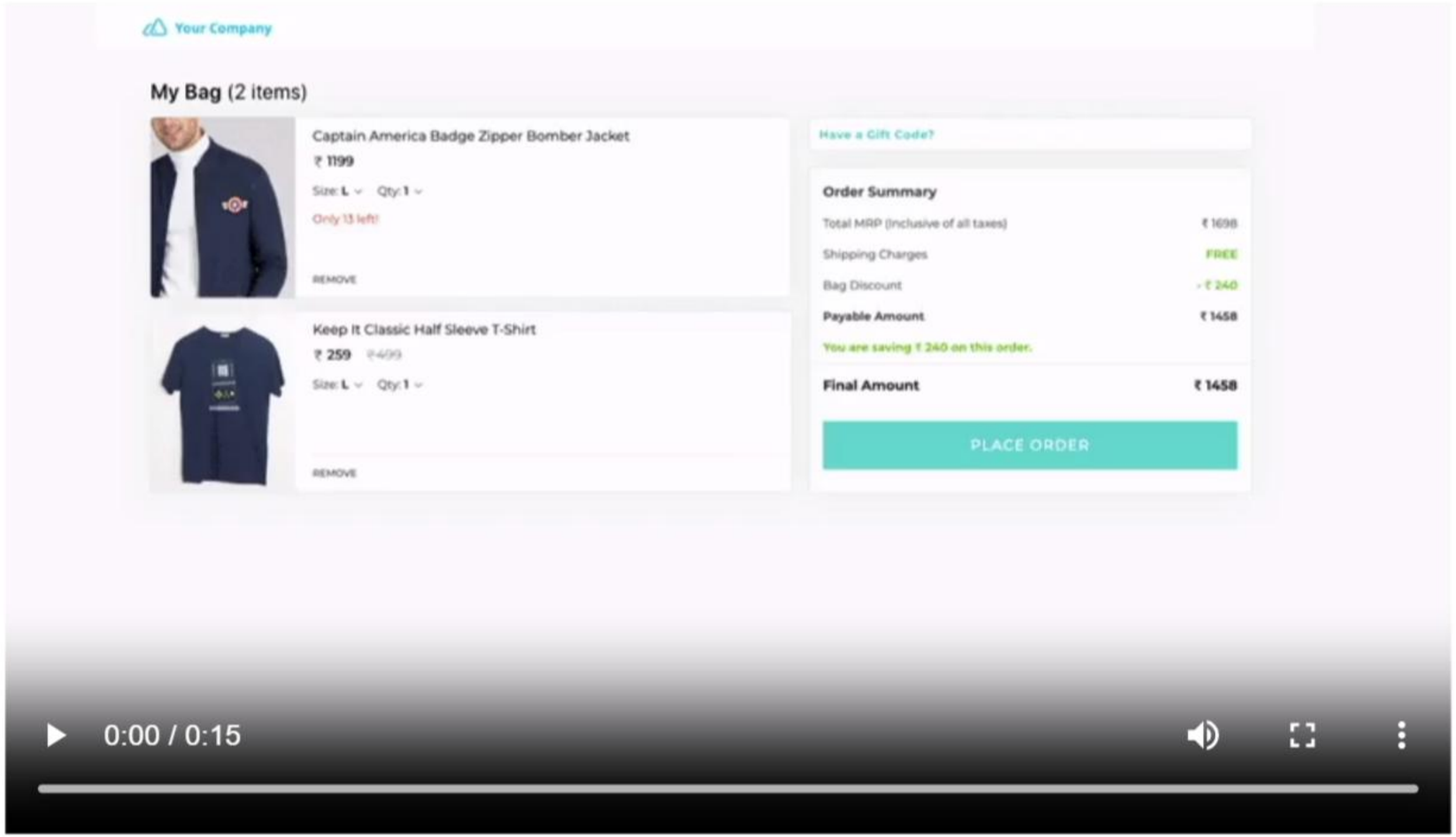
Slide Note

Multi-page Application (MPA)



Slide Note

Single-page Application (SPA)



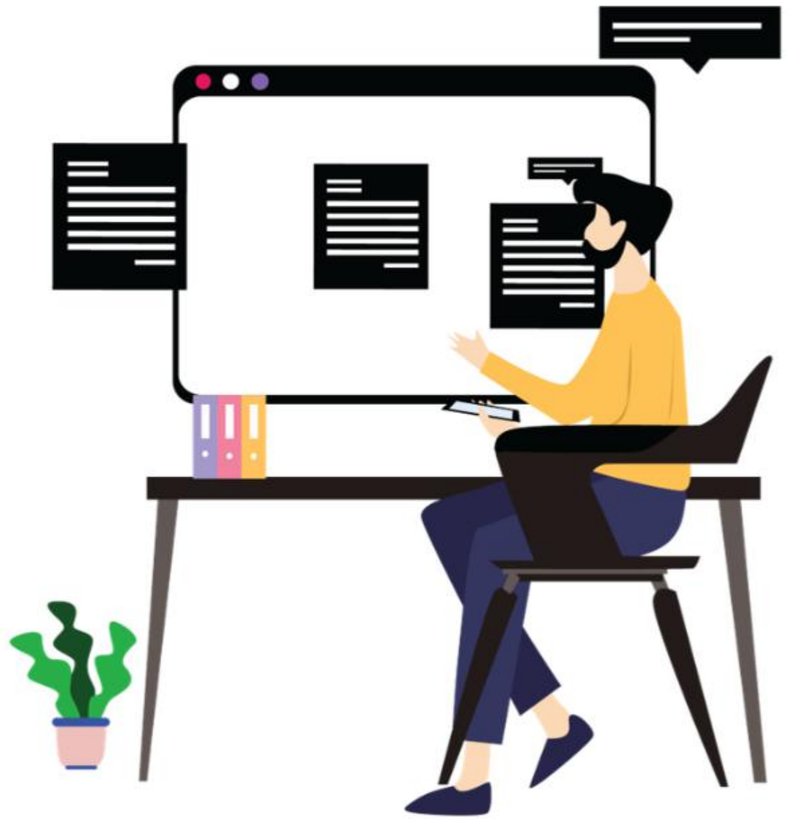
Slide Note

White Screen of Death (WSoD) occurs in multi-page applications (MPA) and leads to less user engagement.

Single Page Applications (SPA), on the other hand, are more engaging and give a native-like experience.

Slide Note

Develop SPA Using Angular Components



Menu

00:05:00:00:45 05/ 46

Slide Note

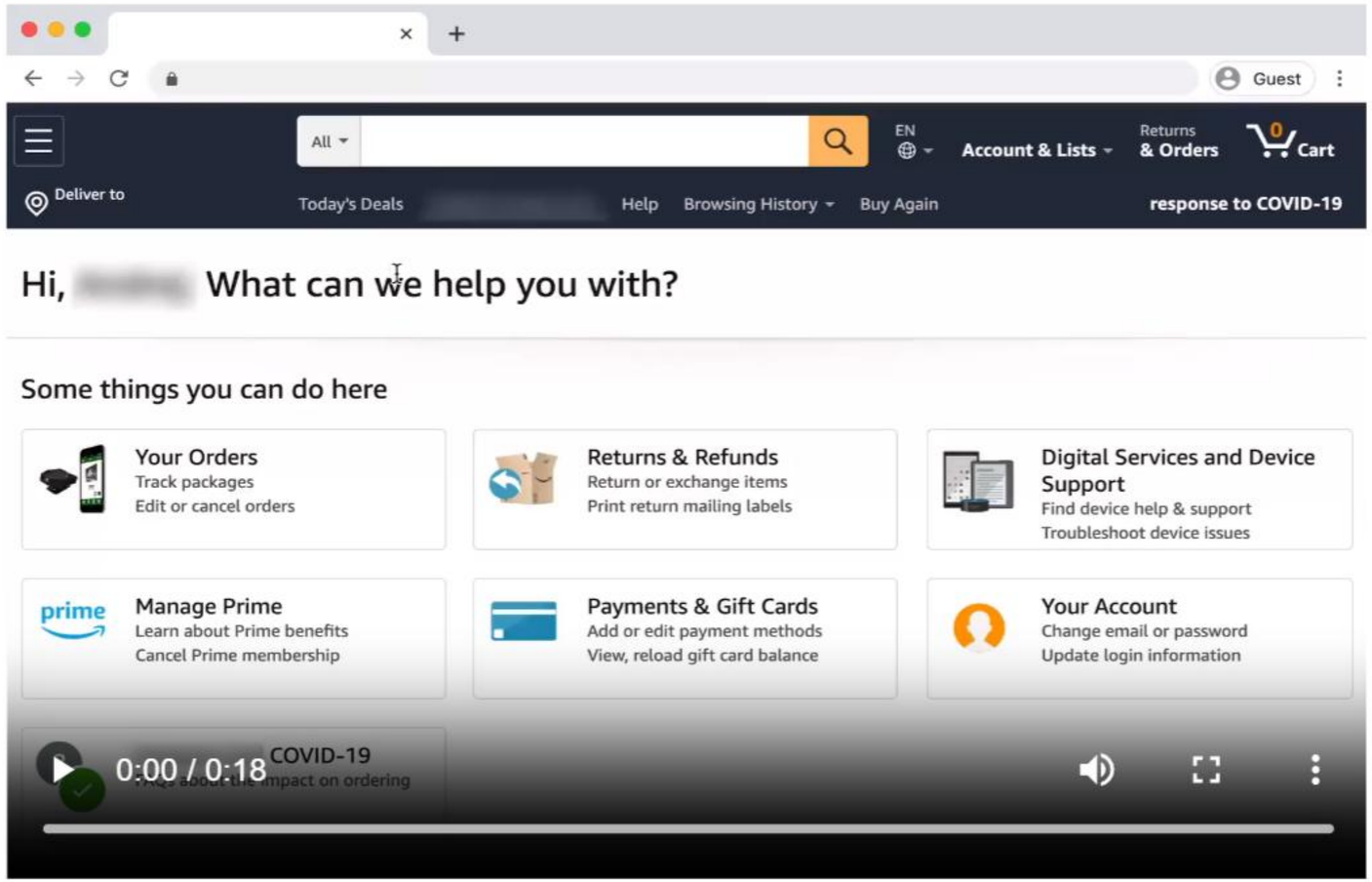
Learning Objectives

- Differentiate between Multi-page Application (MPA) and Single-page Application (SPA)
- Create an Angular project using the CLI tool
- Create Angular components, the main building block for Angular applications
- Design the user interface for views using component templates
- Explore the file and folder structure of an Angular project
- Add behavior and data to views using interpolation and property binding in templates
- Handle the events raised by user actions using event binding
- Explain two-way binding using ngModel

Slide Note

A popular e-commerce website is a typical example of MPA.

Typical Example of MPA

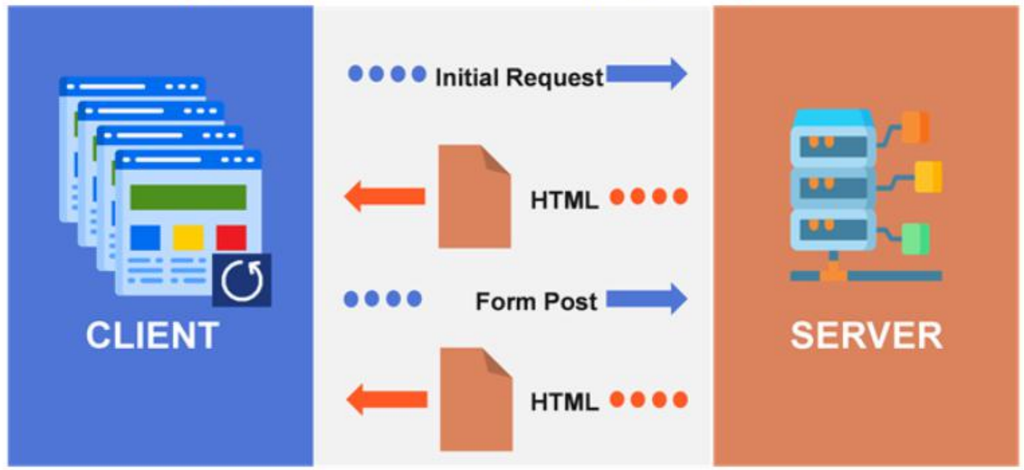


In a popular e-commerce website, when you navigate through the store categories and item details, you can see how the browser is reloading.

Slide Note

What Are Multi-page Applications (MPAs)?

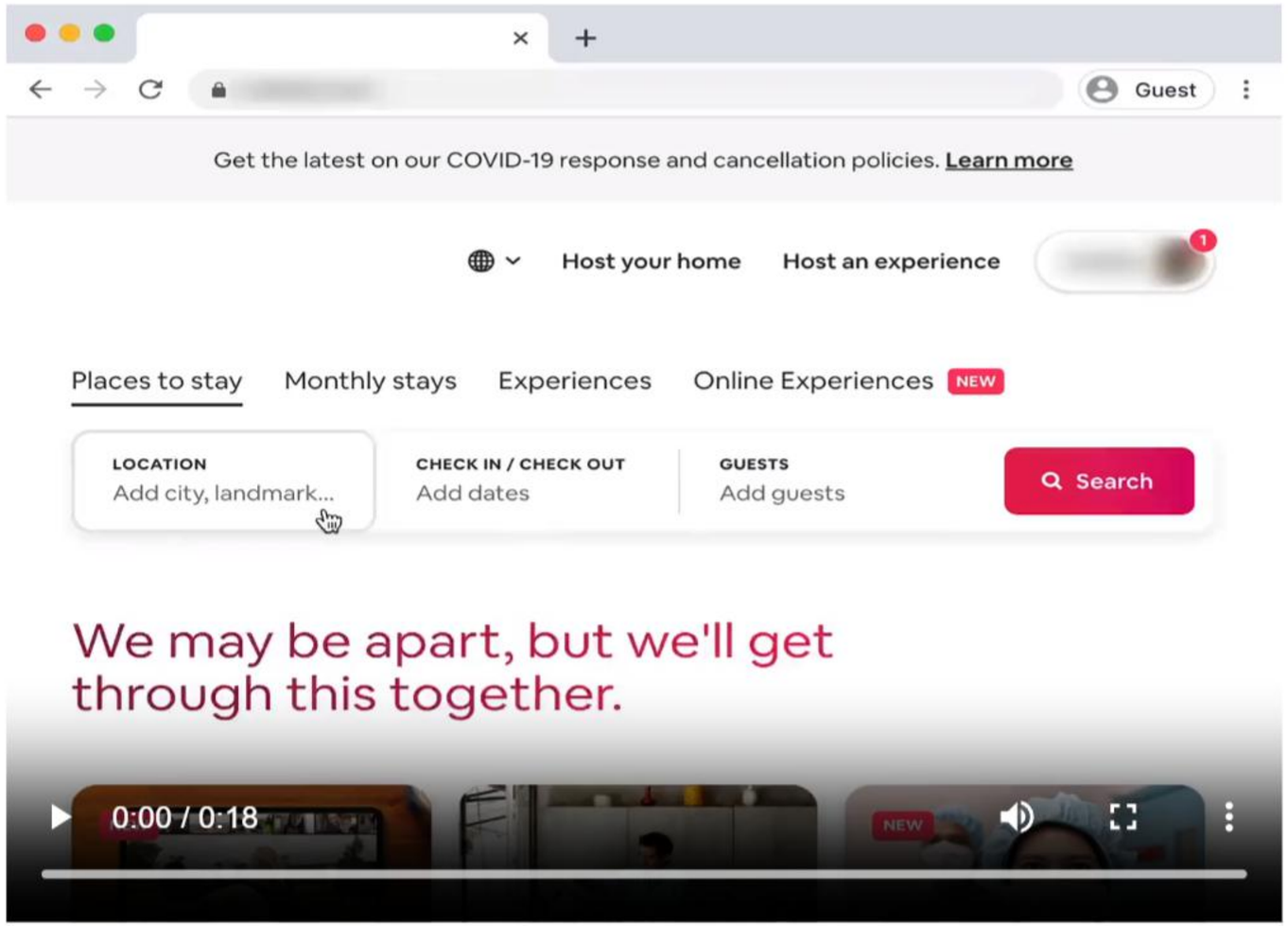
- Multi-page applications are the traditional web applications that reload the entire page and display the new one when a user interacts with the web app.
- An MPA consists of several pages with static information (text, images, etc.) and links to other pages with the same content.
- Each time the data is exchanged, a new page is requested from the server to display in the web browser.
- This process takes time to generate the pages on the server, send them to a client, and display them in the browser, which may affect the user experience.
- Most large-scale e-commerce apps are built as multi-page applications.



Slide Note

Airbnb is a good example of SPA.

Typical Example of SPA



Booking a stay through this popular website is very fast and smooth. Users don't see any delays between the loading of different pages.

Slide Note

What Are Single-page Applications (SPAs)?

- Single-page applications consist of just one page that works inside a browser and does not require page reloading at the time of use.
- A SPA presents the content quickly, elegantly, and effectively as it loads all the content on a single page rather than navigating the user to different pages.
- SPAs are faster than traditional web applications because they execute the logic in the web browser rather than on the server.
- After the initial page load, only the data is sent back and forth instead of the entire HTML, reducing the bandwidth.
- Gmail, Google Maps, Facebook, and GitHub are built on SPA.

Slide Note

MPA vs. SPA

- In an MPA, there are multiple pages with static information like text, images, etc., and links to the pages with the same content.
- **The browser reloads the page content completely,** including the already downloaded content upon moving to another page.
- But SPAs are applications that live on one page. You are still on a single page, but you can switch to other contents on the page, giving the impression of routing from one page to another.
- There will be static parts like the header, side nav bar, and footer, but some middle parts of the page will change by selecting a certain menu option.
- In an SPA, **views aren't complete HTML pages**. They are merely portions of the DOM that make up the viewable areas of the screen.
- Unlike an MPA, where we create multiple pages, **multiple views are created in SPA** to achieve the same functionality within the same page.
- Based on the user interaction, the views are changed on the page, but the page remains the same in a SPA.

Slide Note

Need for Frameworks to Build SPA

- Most web applications have common functionality such as handling sessions, data validation, etc.
- Creating an application using vanilla JS from scratch requires re-writing the same code, thus wasting time and effort.
- Using a framework allows developers to focus on building a unique feature for their applications rather than re-inventing it by coding.
- Frameworks have fascinating templates, session management, and database access libraries.
- Therefore, frameworks improve not only productivity but also the quality of products since these pre-defined libraries are error-free.

React Angula Vue JS Polymer

SVELTE

METEOR

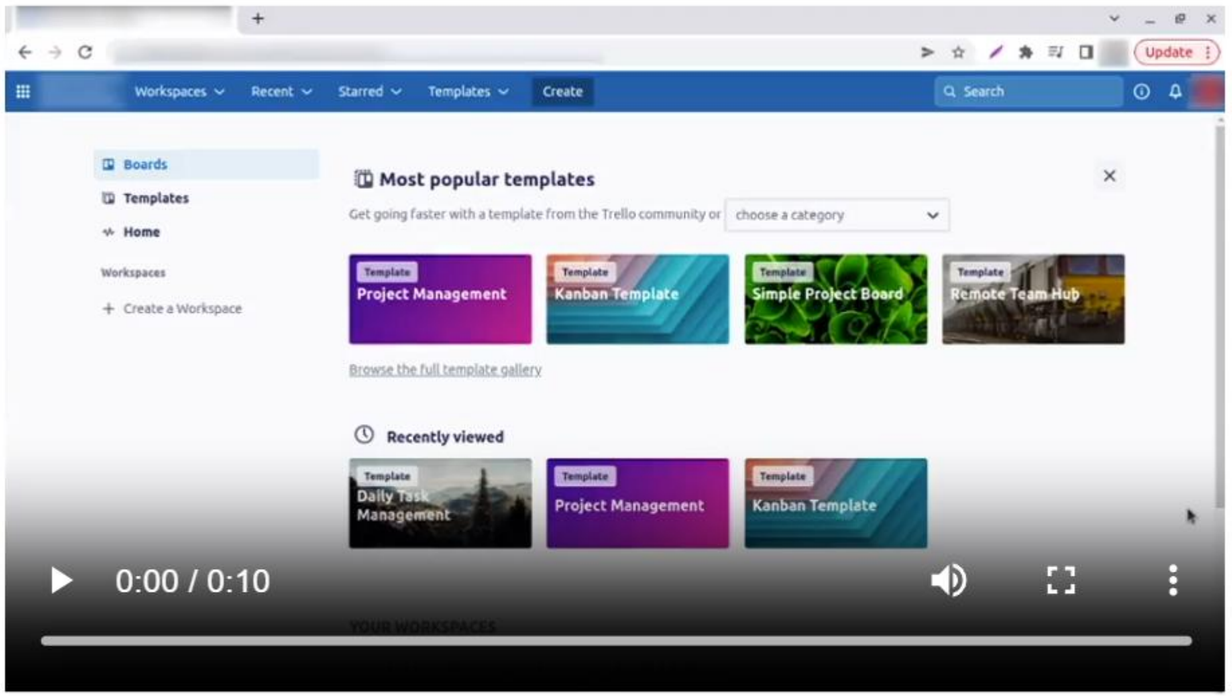
Slide Note

The website loads the default view with **Board View**. Clicking the template menu loads the main content with the lists of templates in the **template view.**

Selecting a particular template (Daily Task Management template) loads the **template details view.** Clicking on the "Use Template" loads the **implementation view** of that template.

Different Views in SPA

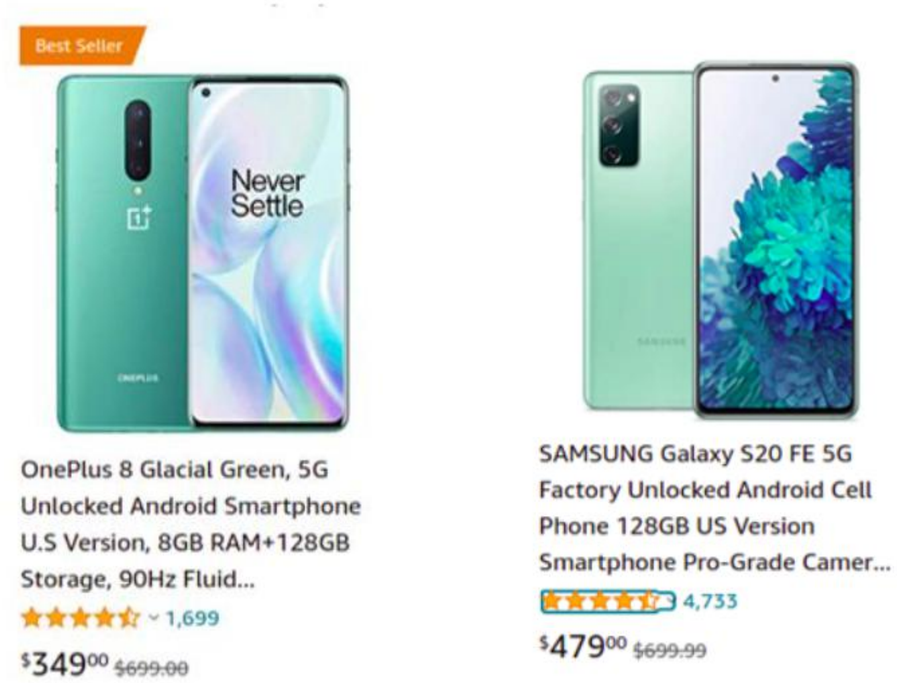
- On this website, you can see that the top navigation bar and sidebar almost remain the same, but the main content changes according to the user's selection.
- Different views are active on the same page.
- A view can be represented using one or more components.
- The component defines what content should be visible in a view based on the user's interaction.
- SPA is made of multiple components to create multiple views.



Slide Note

Web User Interface (UI) Design

- A web page is comprised of components that can be reused.
- Components let you split the UI into independent, reusable pieces and think about each piece in isolation.
- The key benefits of creating components to build UI are:
  - Modularity
  - Reusability
  - Easy to maintain
  - Easy to test



Slide Note

Identifying Components in SPA

Open a SPA like Pinterest and identify the various components inside the single page.

DEMO

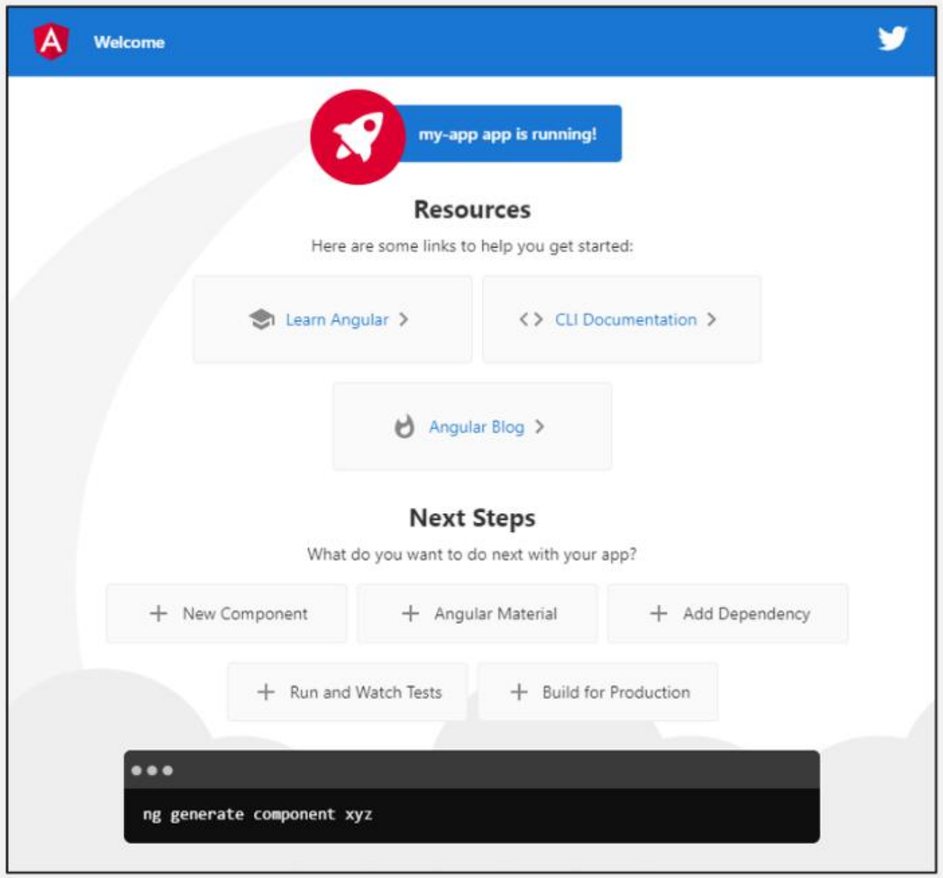
Slide Note

Angular Framework

- Angular is a component-based framework for building scalable web applications.
- It is a development platform built on TypeScript for creating efficient and sophisticated single-page apps.
- It is a well-integrated library collection that covers various features, including routing, form management, client-server communication, and much more.
- Angular includes a suite of developer tools to help you develop, build, test, and update your code.

Slide Note

Delete the default content present in the app.component.html to start working with your project.



Angular Application Output created using Angular CLI command

Install Angular CLI

- Install Angular CLI using the command in the terminal window:

  npm install -g @angular/cli

- To create a new workspace and initial starter app, use the "ng new" command, which prompts for information about features to include in the initial app.

  ng new first-app

- Run the application with the included server after navigating to the workspace folder.

  cd first-app

  ng serve --open

Slide Note

Angular CLI

- The Angular CLI is a command-line interface tool to initialize, develop, scaffold, and maintain Angular applications directly from a command shell.
- The Angular framework is provided with the Angular CLI tool to create projects more easily and quickly.
- The Angular CLI is the most recommended way to develop Angular applications.
- Some examples of CLI commands are:
  - ng new: Creates a new Angular workspace
  - ng build: Compiles an Angular app into an output directory
  - ng serve: Builds and serves your application, rebuilding on file changes
  - ng generate: Generates or modifies files based on a schematic
  - ng test: Runs unit tests on a given project
  - ng e2e: Builds and serves an Angular application, then runs end-to-end tests

Slide Note

- Install Angular CLI using the command in the terminal window:

npm install -g @angular/cli

- To create a new workspace and initial starter app, use the "ng new" command, which prompts for information about features to include in the initial app.

ng new hello-world

- Run the application with the included server after navigating to the workspace folder.

cd hello-world

ng serve --open

Create an Angular "Hello, World!" App Using CLI

Create a "Hello, World!" application using the CLI command and execute it to see the app running in the browser with the default port number.

Explore the files and folders generated by angular CLI.

Explore the configuration files like the angular.json, package.json, tsconfig.json, and generated source files under the src folder.

Click here for the demo solution.

DEMO

Slide Note

Angular Component

Components are the main building block for Angular applications. Each component consists of:

- An HTML template that declares what renders on the page.
- A TypeScript class that defines behavior.
- A CSS selector that defines how the component is used in a template.
- Optionally, CSS are styles applied to the template.

- View layout
- Created with HTML

- Code supporting the view
- Created with TypeScript
- Properties: data
- Methods: logic

- Extra data for Angular

Slide Note

app.component.ts: A TypeScript class that includes the code for the component.

CSS Selector: Every component requires a CSS selector. A selector instructs Angular to instantiate this component wherever it finds the corresponding tag in template HTML. In this example code, **app-root** is the CSS selector for AppComponent.

app.component.html: HTML template that tells how to render the component on the browser.

app.component.css: Styles are declared for the AppComponent in this file.

Decorators are design patterns that is used to separate modification or decoration of a class without modifying the source code.

The **Component decorator** marks a class as an Angular component and provides additional metadata that determines how the component should be processed, instantiated, and used at runtime.

Angular Component (Cont'd.)

Metadata using decorator

CSS selector

```
import { Component } from
'@angular/core';

@Component({
selector: 'app-root',
templateUrl: './app.component.html',
styleUrls: ['./app.component.css']
})
export class AppComponent {
title= 'Angular Web Page';
}
```

app.component.ts

app.component.html

```
<h1>
 {{title}}
</h1>
```

```
h1{
  color: midnightblue;
}
```

app.component.css

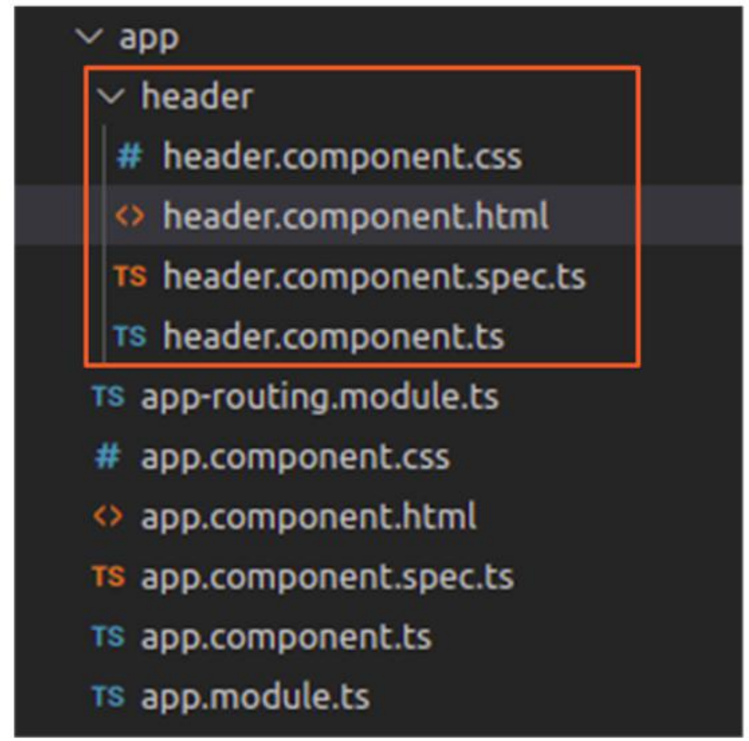
Slide Note

Creating a Component Using Angular CLI

Giving the following command in the terminal window generates a component called **header**.

ng generate component header

- By default, this command creates the following:
  - A folder naming header
  - A component file, header.component.ts
  - A template file, header.component.html
  - A CSS file, header.component.css
  - A testing specification file, header.component.spec.ts
- Adds the newly created HeaderComponent to the root module called app.module.ts



Slide Note

The above image lists the various terms used in defining an Angular component.

• The import statement in the first line is used to import the Angular Component decorator from the @angular/core library.

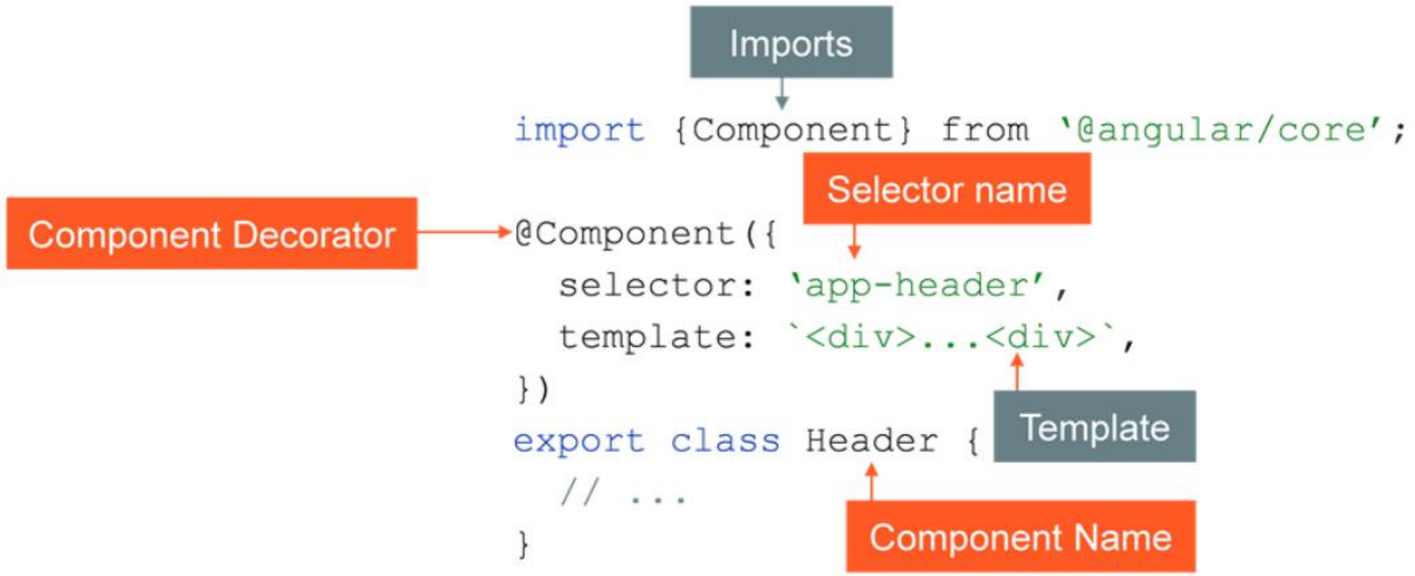
• @Component decorator: The metadata for a component tells Angular where to get the major building blocks it needs to create and present the component and its view.

• Inside the @Component decorator, the component's CSS selector is added using a selector property.

• A template property is added inside the @Component decorator to define the template within the component.

• The component class Header is exported to be used in other modules.

Angular Component Class



Slide Note

In Angular, a template is a chunk of HTML that renders a view in the browser, but with lot more functionality.

Component's Template

- A component controls a patch of the screen called a view, defined by its companion called the Template.
- A template is a block of HTML that tells Angular how to render the component in your application.
- A template contains Angular template syntax, which alters the HTML based on your application's logic and the state of application and DOM data.
- Angular Template is part of an overall webpage and not the entire page. So, there is no need to include elements such as <html>, and <body> as it can only focus on that part of the developed page.

```
<div>
  <h1>Angular Web Page</h1>
</div>
```

header.component.html

Slide Note

Create Fruit Fantasy App

Create a new application called Fruit Fantasy using CLI commands. Create a Header component to display the header for the Fruit Fantasy app.

Modify the template to display the title text in the app component using text interpolation.

Click here for the demo solution.

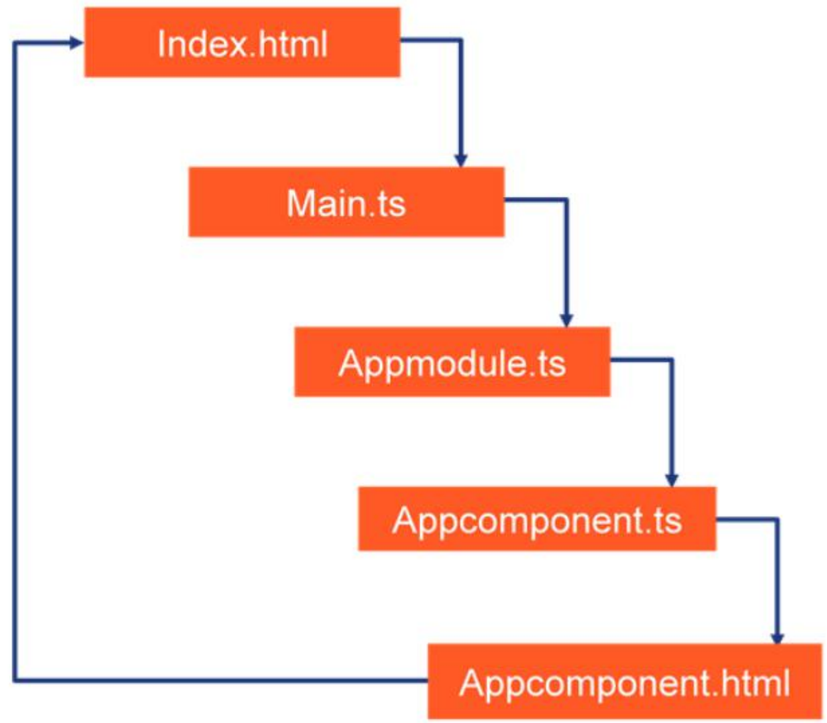
DEMO

Slide Note

Angular Application Workflow Execution

- The important parts of an Angular application are index.html, module, component, and main.
- The angular runs `index.html` whenever the angular application runs on a browser by hitting 'localhost:4200'.
- The `index.html` file contained a selector of a component that is registered in the body tag as shown below.

```
<body>
<app-root></app-root>
</body>
```



Slide Note

main.ts

- The **main.ts** file is the first code executed and is an important part of an angular application.
- The job of **main.ts** is to bootstrap the application. It loads everything and controls the startup of the application.
- The bootstrap section of the **main.ts** registers a module that the user wants to execute first.
- It tells the builder to start the app from the AppModule. AppModule refers to the **app.module.ts** file.

```
import { enableProdMode } from
'@angular/core';
import { platformBrowserDynamic } from
'@angular/platform-browser-dynamic';

import { AppModule } from
/'./app/app.module';
import { environment } from
/'./environments/environment';

if (environment.production) {
enableProdMode();
}

platformBrowserDynamic().bootstrapModule(App
Module)
.catch(err => console.error(err));
```

Slide Note

```
import {BrowserModule} from ' @angular/platfo
rm-browser';
import { NgModule} from '@angular/core';

import { AppRoutingModule } from './app-
routing.module';
import { AppComponent} from './app/component'
;

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

app.module.ts

- The module is the third most important part of an Angular application, which is user dependent.
- The user can add several modules to the application.
- The module contains a few sections in which the user can import all components, modules, and services that are used in the app.
- The bootstrap section registers the component that the user wants to display first on the browser. In the code, the AppComponent is the first component to be loaded.

Slide Note

app.component.ts

- AppComponent is a logical part of the Angular application that contains a template, classes, and metadata.
- The template comprises of HTML code used to render views in the browser.
- Class is defined in TypeScript and has properties and methods.
- Additional metadata is provided using the @Component decorator, which marks a class as an Angular component.

```
import { Component }
from '@angular/core';
@Component({
selector: 'app-root',
templateUrl: './app.component.html',
styleUrls: ['./app.component.css']
})
export class AppComponent{

}
```

Slide Note

@NgModule is a class decorator that allows us to tell Angular that a particular class is a module. It defines this effect without modifying anything inside the code.

Angular Modules

- An Angular module is used to group an inter-related set of Angular components that are closely related and meant to be used together.
- Every application has at least one NgModule class, the *root module*, conventionally named as AppModule and resides in app.module.ts.
- A NgModule is defined by a class decorated with @NgModule().
- The @NgModule metadata has an important role in guiding the Angular compilation process that converts the application code into a highly performant JavaScript code.

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

Slide Note

Export and Import Modules

- An Angular project consists of multiple files to keep the code easy to maintain.
- The **'export'** keyword is used to provide code to other modules.
- The **'import'** keyword reads the code exported from another module.
- Angular uses a collection of JavaScript modules as libraries.
- These Angular library names begin with the `@angular` prefix and are imported using JavaScript import statements.
- To access the data from the imported modules, we need to add the imported libraries to the @NgModule metadata imports.

```
import {BrowserModule} from '@angular/platfor
m-browser';
import { NgModule} from '@angular/core';

import { AppRoutingModule } from './app-
routing.module';
import { AppComponent} from './app/component'
;

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

Slide Note

Quick Check

What is the purpose of the Ng serve command?

1. It compiles an Angular app into an output directory named dist/at the given output path.

2. It builds and serves an Angular app and then runs end-to-end tests.

3. It builds and serves your app, rebuilding on file changes.

4. None of the above.

Slide Note

Quick Check: Solution

What is the purpose of the Ng serve command?

1. It compiles an Angular app into an output directory named dist/at the given output path.

2. It builds and serves an Angular app and then runs end-to-end tests.

3. **It builds and serves your app, rebuilding on file changes.**

4. None of the above.

Slide Note

One-way Binding: Text Interpolation

- Text interpolation allows you to incorporate dynamic string values into your HTML templates.
- With interpolation, you can dynamically change what appears in an application view, such as displaying a custom greeting that includes the user's name.
- Interpolation refers to embedding expressions into marked-up text.
- By default, interpolation uses the double curly braces {{ and }} as delimiters.

```
title: 'World Health Organization';
itemImageUrl: './images/abc.png';
```

app.component.ts

```
<p>{{title}}</p>
<div>
   <img src='{{itemImageUrl}}'>
</div>
```

app.component.html

Slide Note

One-way Binding: Property Binding

- Property binding in Angular helps you set values for properties of HTML elements.
- With property binding, you can implement toggle button functionality, set paths programmatically, and share values between components.
- Property binding moves a value in one direction, from a component's property into a target element property.
- To bind to an element's property, enclose it in square brackets, [ ], which identifies the property as a target property.
- The target property is the DOM property to which you want to assign a value.

```
itemImageUrl = './images/abc.png';
isUnchanged = true;
classes = "special";
```

app.component.ts

```
<img [src] = 'itemImageUrl'>
<button [disabled] = 'isUnchanged'>
Disabled Button
</button>
```

app.component.html

Slide Note

Built-in Directives

- Directives are classes that add additional behavior of elements in your Angular applications.
- Angular built-in directives manage forms, lists, styles, etc.
- Different types of directives:
  - Components: Most common directive used with the template
  - Attribute Directives: To change the appearance or behavior of an element, component, or another directive
  - Structural Directives: To change the DOM layout by adding and removing DOM elements
- Built-in Structural Directives:
  - *NgIf: Conditionally creates or disposes subviews from the template

```
<div *ngIf="fruit" class="name">{{fruit.name}}</div>
```

  - *NgFor: Repeat a node for each item in a list

```
<div *ngFor="let fruit of fruits">{{fruit.name}}</div>
```

  - *NgSwitch: A set of directives that switch among alternative views
  - Check this link to know more about built-in attribute directives.

Slide Note

Display Fruit List in the Fruit Fantasy App

Modify the Fruit Fantasy app component to display the list of fruits.

Create a model class called Fruit and an array to store lists of fruits.

Use property binding and ngFor directive to iterate through the list of fruits in the app component template.

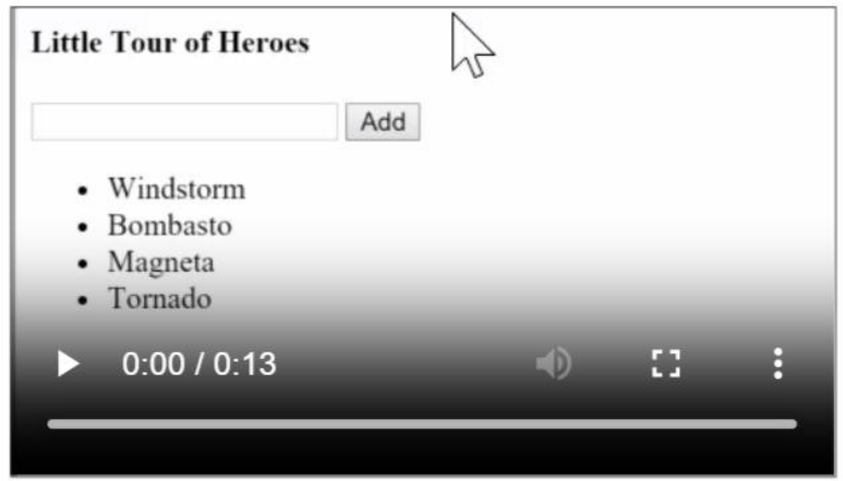
Click here for the demo solution.

DEMO

Slide Note

One-Way Binding: Event Binding

- One-way data binding from the view to the component can be achieved using the event binding technique.
- In the above video, the Hero-List component contains a text box, an add button, and a container to display the list of heroes.
- The button click event will invoke an event handler method to add the new hero.
- In the event handler method, the component's hero array property is changed by adding a new hero value to the array.
- In turn, it displays the list of heroes with the newly added hero.
- Check this link for the angular component code.



Capture Data From User View: Event Binding

- **Event Binding**: This allows us to listen to and respond to user actions such as keystrokes, mouse movements, clicks, and touches.
  - To bind to an event, the Angular event binding syntax is used. This syntax consists of a target event name within parentheses to the left of an equal sign and a quoted template statement to the right.

```
<button (click)="onSave()">Save</button>
```

target event name

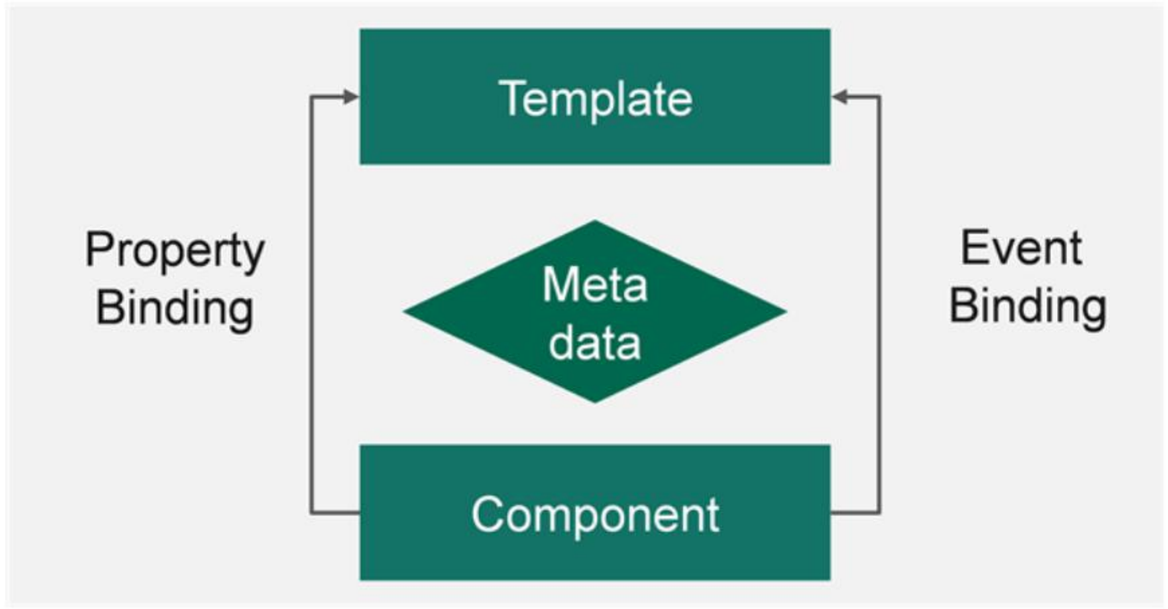
template statement

  - The event binding listens to the button's click events and calls the component's `onSave()` method whenever a click occurs.

Two-way Data Binding

- Two-way data binding combines property binding with event binding.
- Property binding sets a specific element property.
- Event binding listens for an element change event.
- Angular's two-way data binding syntax combines square brackets and parentheses [()]. The [()] syntax combines the brackets of property binding, [ ], with the parentheses of event binding, ( )

```
<app-sizer [(size)] = "fontSizePx"></app-sizer>
```



Search Fruits in the Fruit Fantasy App

Add search functionality to the Fruit Fantasy app, which allows users to search for fruits.

Use ngModel to establish a binding between the component and the template.

Click here for the demo solution.

DEMO

Capture Data From User View: ngModel

NgModel: Used for displaying and updating properties when the user makes changes.

To use NgModel in the code:

1. Import FormsModule and add it to the NgModule's imports list.
2. Add an [(ngModel)] binding on an HTML <form> element and set it equal to any property. This [(ngModel)] syntax can only set a data-bound property.

app.component.html

```
<div>
 <input type ="text" placeholder = "Search Fruit"
[(ngModel)] = "searchText" />
 <button (click) = "search()" type = "button"> Go
</button>
</div>
```

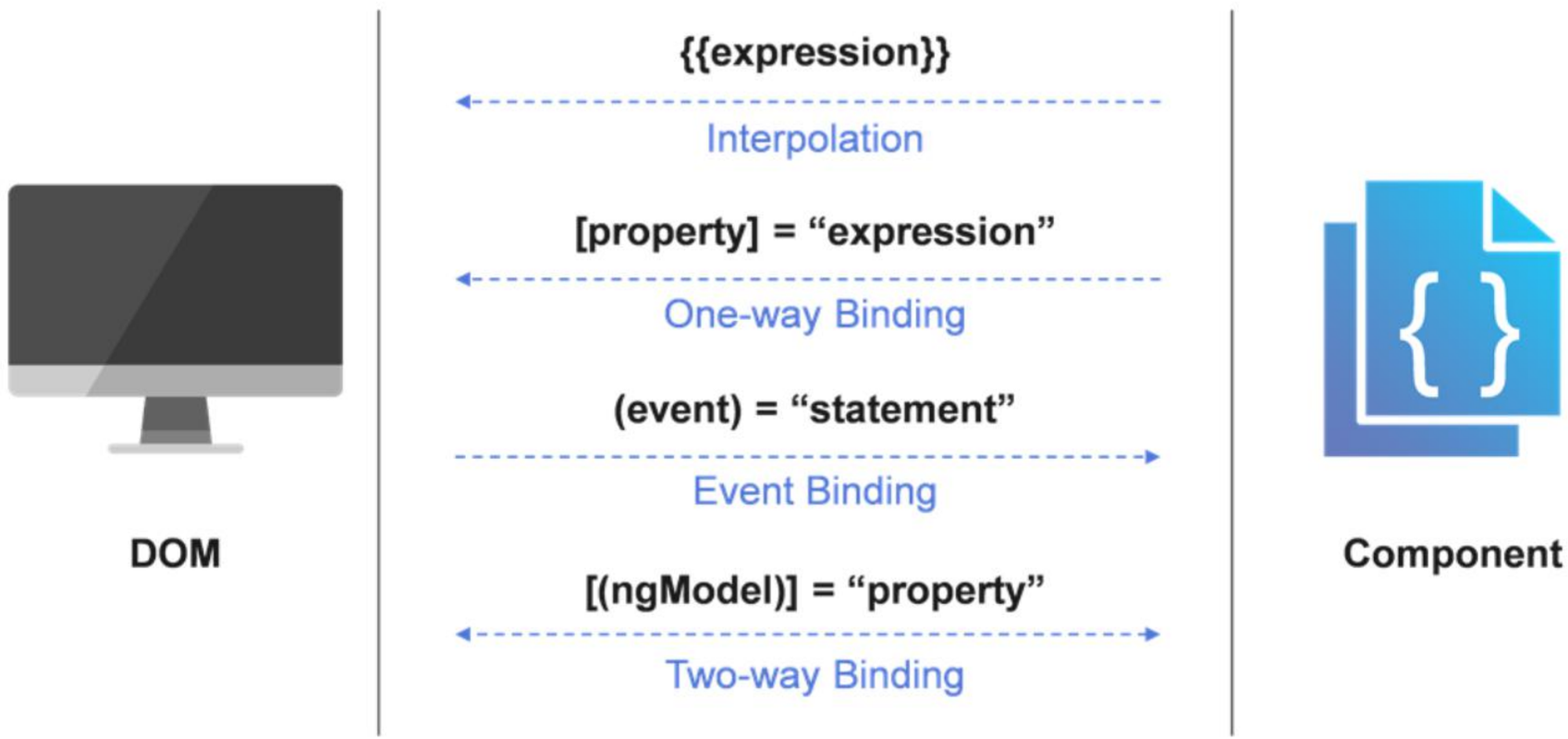
```
import { BrowserModule } from '@angular/platform-
browser';
import { NgModule } from '@angular/core';
import {FormsModule} from '@angular/forms';
@NgModule({
  declarations: [ //components declaration
  ],
  imports: [
  FormsModule, BrowserModule
  ],
})
export class AppModule { }
```

app.module.ts

```
export class AppComponent{
fruits = FRUITS;
searchText: string = '';
search() {
//Write logic to display the searched fruit
}
```

app.component.ts

Data Binding Markup: Summary



Quick Check

Consider the below code snippet from the class file and the corresponding template of the component. How would you bind the 'src' attribute of the image tag with the 'url' property?

```
export class TestComponent{
    public url: string = 'http://www.images.com/1.jpg';
}
```

Test.component.html

<img _____>

1. [src]='url';

2. {{ src | url }}

3. src=[url]

4. (src)=url

Quick Check: Solution

Consider the below code snippet from the class file and the corresponding template of the component. How would you bind the 'src' attribute of the image tag with the 'url' property?

```
export class TestComponent{
    public url: string = 'http://www.images.com/1.jpg';
}
```

Test.component.html

<img _____>

1. [src]='url';

2. {{ src | url }}

3. src=[url]

4. (src)=url