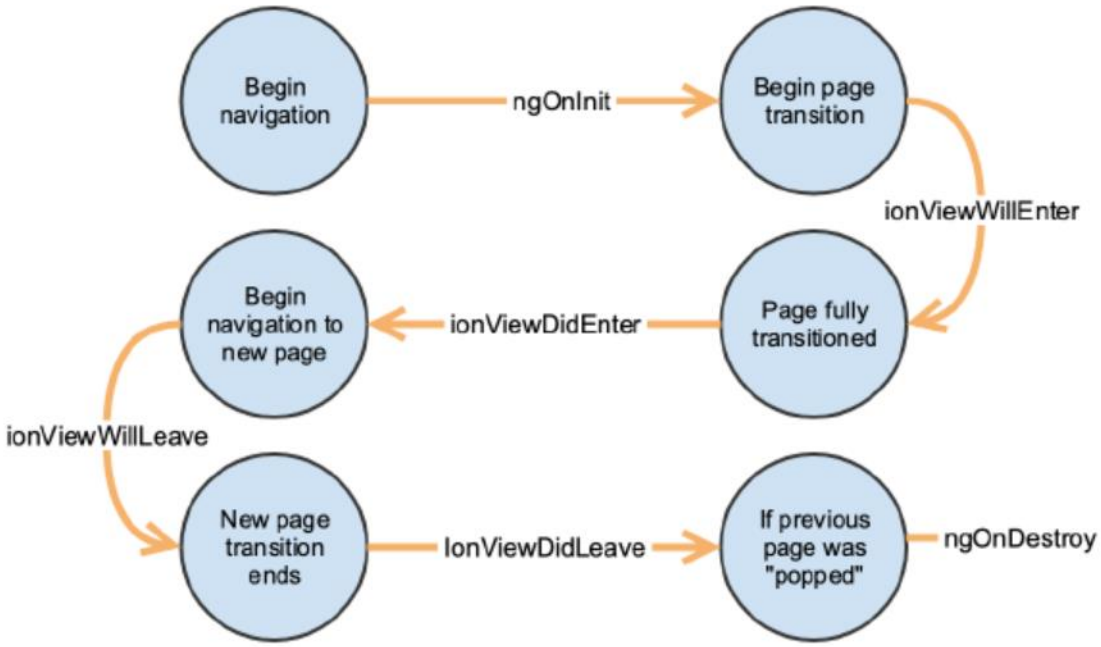
Q-1 Ionic Page Life Cycle:



- ngOnInit - Initialize your component and load data from services that don't need refreshing on each subsequent visit.
- ionViewWillEnter - Since ionViewWillEnter is called every time the view is navigated to (regardless if initialized or not), it's a good method to load data from services. However, if your data comes back during the animation, it can start lots of DOM manipulation, which can cause some janky animations.
- ionViewDidEnter - If you see performance problems from using ionViewWillEnter when loading data, you can do your data calls in ionViewDidEnter instead. This event won't fire until after the page is visible by the user, however, so you might want to use either a loading indicator or a skeleton screen, so content doesn't flash in un-naturally after the transition is complete.
- ionViewWillLeave - Can be used for cleanup, like unsubscribing from observables.
  Since ngOnDestroy might not fire when you navigate from the current page, put your cleanup code here if you don't want it active while the screen is not in view.
- ionViewDidLeave - When this event fires, you know the new page has fully transitioned in, so any logic you might not normally do when the view is visible can go here.

- ngOnDestroy - Cleanup logic for your pages that you don't want to clean up

  in ionViewWillLeave.

Q-2 web components

The Web Components are a set of web platform APIs that allow you to create custom, reusable, encapsulated HTML tags to use in web apps and web pages.

Their functionality is encapsulated away from the rest of your code and you can utilize them in your web apps.

Using Web Components, you can create almost anything that can be done with HTML, CSS and JavaScript. So, this way you can create a portable component that can be easily reused.

One of the aims of the web components is to help developers re using code as much as possible.

Web Components are based on 4 specifications:

Custom Elements. They define the bases and fundamentals to design and use new types of DOM elements.

Shadow DOM. It defines how to use encapsulated styles and markup within a web component. We will learn more about Shadow DOM below.

ES Modules. It defines how to include and reuse JS documents in a standards based, modular, performant way.

HTML Template. It defines how to declare code fragments that will not be used while the page is loading, but could be later instantiated at runtime.

Q-3 types of alert in ionic

1.Action Sheet

2.Alert Controller

1.An Action Sheet is a dialog which allows us to choose to confirm or cancel an action from a set of options.

When the Action Sheets are triggered, the rest of the page darkens to give more focus to the options of the Action Sheets.

Code:

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

import {ActionSheetController } from '@ionic/angular';


@Component({
```

```
  selector: 'app-home',
  templateUrl: 'home.page.html',
  styleUrls: ['home.page.scss'],
})
export class HomePage {
  constructor(
    public actionsheetCtrl: ActionSheetController
  ) { }

  async openMenu() {
    const actionSheet = await this.actionsheetCtrl.create({
      header: 'Modify your album',
      buttons: [
        {
          text: 'Destructive',
          role: 'destructive',
          handler: () => {
            console.log('Destructive clicked');
          }
        },{
          text: 'Archive',
          handler: () => {
            console.log('Archive clicked');
          }
        }, {
          text: 'Cancel',
          role: 'cancel',
          handler: () => {
            console.log('Cancel clicked');
          }
        }
```

```
      ]
    });
    await actionSheet.present();
  }
}
```

Html code

```
<ion-header>
  <ion-toolbar>
    <ion-title>
      Action Sheets
    </ion-title>
  </ion-toolbar>
</ion-header>

<ion-content class="ion-padding" class="action-sheets-home-page">
  <button ion-button block (click)="openMenu()">
    Show Action Sheet
  </button>
</ion-content>
```

2. The alert controller is responsible for **creating** an alert in the Ionic application. It uses **create()** method to create an alert and can be customized by passing alert option in the create() method.

Code:[single button]

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

@Component({
  selector: 'app-home',
  templateUrl: 'home.page.html',
  styleUrls: ['home.page.scss'],
})
```

```
export class HomePage {
  constructor(public alertCtrl: AlertController) { }

  async showAlert() {
    const alert = await this.alertCtrl.create({
      header: 'Alert',
      subHeader: 'SubTitle',
      message: 'This is an alert message',
      buttons: ['OK']
    });
    await alert.present();
    const result = await alert.onDidDismiss();
    console.log(result);
  }
}
```

Home.page.html

```
<ion-header translucent>
  <ion-toolbar color="danger">
    <ion-title>Ionic Alert</ion-title>
  </ion-toolbar>
</ion-header>

<ion-content class="ion-padding" color="light">
  <div>
    <ion-button (click)="showAlert()">Basic Alert</ion-button>
  </div>
</ion-content>
```

Code:[multiple button]

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

@Component({
  selector: 'app-home',
  templateUrl: 'home.page.html',
  styleUrls: ['home.page.scss'],
})
export class HomePage {
  constructor(public alertCtrl: AlertController) { }
  async showMultipleAlertButtons() {
    const alert = await this.alertCtrl.create({
      header: 'MultipleButtonAlert',
      subHeader: 'SubTitle',
      message: 'This is an multiple button alert message',
      buttons: ['Cancel', 'Save', 'Open']
    });
    await alert.present();
  }
}
```

Home.page.html

```
<ion-header translucent>
  <ion-toolbar color="danger">
    <ion-title>Ionic Alert</ion-title>
  </ion-toolbar>
</ion-header>

<ion-content class="ion-padding" color="light">
  <div>
```

<ion-button (click)="showMultipleAlertButtons()" expand="block">Multiple Button Alert</ion-button>

</ion-content>

Q-4 what is hybrid app development?

Hybrid development adopts an approach to write the code once, run everywhere to build an app.

Web technologies like HTML5, CSS and JavaScript are the core of these apps while deploying a native container.

Some popular examples of applications that amaze customers with their high performance, interface & functionality include Twitter, Instagram, Uber, Gmail, and more.

Hybrid apps use shared code to deploy across multiple platforms – Android and iOS.

It offers a perfect blend of both native and web features.

It provides a unique avenue for enterprises looking for an app that works on various devices to expand their business operations.

It lets you engage customers irrespective of the platform they use.

Whether you want an app to entice, be a game, or take photos, these types of apps offer tons of excellent options to leap into the mobility world.

Q-5 native and hybrid and cross platform

Native

The term native app development refers to building a mobile app exclusively for a single platform.

Pros Of Native App Development

1. *Broad Functionality*

2. *Better Store Support*

3. *Increased Scalability*

4. *High Performance And Great UX*

Cons Of Native App Development

1. *Costly*

2. *Time Consuming*

Cross platform

Cross-platform development points to the process of creating an app that works on several platforms.

Pros Of Cross-Platform App Development

1. *Less Costly*

2. *Faster Development*

3. *Single Code Base*

Cons Of Cross-Platform App Development

1. *Slower App*

2. *Limited Functionality*

3. *Limited UX*

Hybrid

Hybrid app development is the combination of native and web solutions where developers need to embed the code written with the languages such as CSS, HTML, and JavaScript into a native app with

the help of plugins including Ionic's Capacitor, Apache Cordova, and so on which enables to get the access of native functionalities.

Benefits of Hybrid Apps

1. Rapid Time to Market
2. Easy Maintenance
3. Minimized Cost Backed with Ease of Development
4. Improved UI/UX

Cons of Hybrid App Development

1. No Offline Support
2. OS Inconsistencies

Q-6 Code for creating list

```
<ion-header>
  <ion-toolbar>
    <ion-title>
      Lists Example
    </ion-title>
  </ion-toolbar>
</ion-header>
<ion-content color="light">
  <ion-list inset>
    <ion-list-header>Action Game</ion-list-header>
    <ion-item>Pok?mon Yellow</ion-item>
    <ion-item>Super Metroid</ion-item>
    <ion-item>Mega Man X</ion-item>
  </ion-list>
</ion-content>
```

Q-7 code for textbox value in next page

Page1

Html file

```
<ion-row>
  <ion-col size="12">
```

```
    <ion-item>
      <ion-label position="stacked">E-Mail</ion-label>
      <ion-input type="email" [(ngModel)]="email"></ion-input>
    </ion-item>
    <ion-item>
      <ion-label position="stacked">Password</ion-label>
      <ion-input type="password" [(ngModel)]="password"></ion-input>
    </ion-item>
    <ion-item>
      <ion-label>Country</ion-label>
      <ion-select placeholder="Select Country" [(ngModel)]="country">
        <ion-select-option value="India">India</ion-select-option>
        <ion-select-option value="Usa">Usa</ion-select-option>
      </ion-select>
    </ion-item>
  </ion-col>
  <ion-col size="12" class="ion-text-center">
    <ion-button (click)="goToAboutPage()" expand="block" shape="round">
      Click me
    </ion-button>
  </ion-col>
 </ion-row>
```

Ts file

```
public country: string;
 public email: string;
 public password: string;

constructor(public nav: NavController) {}

goToAboutPage() {
   let params: any = {
     country: this.country,
     email: this.email,
     password: this.password
   }
   this.nav.navigateForward('/identity', { state: params }); // params to pass object/array
 }
```

Page 2

```
constructor(public router: Router){
if (router.getCurrentNavigation().extras.state) {
     const params = this.router.getCurrentNavigation().extras.state;
console.log(params.email)
console.log(params.password)
console.log(params.country)
   }
}
```

Q-8 show prompt box

```
showPrompt() {
   this.alertController.create({
    header: 'Prompt Alert',
    subHeader: 'Enter information requested',
    message: 'Enter your favorate place',
    inputs: [
     {
      name: 'Place',
      placeholder: 'Eg.NY',

     },
    ],
    buttons: [
     {
      text: 'Cancel',
      handler: (data: any) => {
       console.log('Canceled', data);
      }
     },
     {
      text: 'Done!',
      handler: (data: any) => {
       console.log('Saved Information', data);
      }
     }
    ]
   }).then(res => {
    res.present();
   });
  }
```