Savitribai Phule Pune University
Semester VI (Information Technology)

Web Application Development

Unit III : Front End Technologies

Q. 1 What is web framework? Why we need Web Framework? **(5 Marks)**

Ans.:

Web Framework

- Web framework is a software framework that is designed to support the development of web applications including web services, web resources, and web APIs.
- Frameworks are, in short, libraries that help you develop your application faster and smarter. Web frameworks provide a standard way to build and deploy web applications on the World Wide Web. Web frameworks aim to automate the overhead associated with common activities performed in web development. For example, many web frameworks provide libraries for database access, templating frameworks, and session management, and they often promote code reuse.
- A web application or development framework can be considered as a pre-built structure that handles the more repetitive processes and features involved with developing a website/application. This means that a web developer will spend most of their time interacting with the different parts of the web framework through the use of code.
- There are many advantages of using web development frameworks like increased productivity, rapid development, reusable, clean and efficient code and added security.

Need of Web Framework

Web Frameworks provide tools and libraries to simplify common problems that developers face during web development operations.

(a) Help work directly with HTTP requests and responses : Web Frameworks help in simplifying the development by generating server-side code to work with requests and responses. Therefore, interacting with higher-level code becomes easier than with lower-level networking primitives.

(b) Help route requests to the appropriate handler : Web Frameworks are easily able to handle different resources provided by different sites that are accessible through distinct URL's. Web Frameworks map URL patterns to specific handler functions. This method eases the website maintenance as the URL used to deliver the code can be changed without the need to change the underlying code. Each framework handles the mapping in their way.

(c) Web Frameworks help in accessing data in request easily : Data can be encoded in an HTTP request in various ways. The data required in URL parameters or URL structure can be encoded in an HTTP get request. HTTP post request can update the resource on the server and include "Post data" within the body of the request. These may often include the session information or user in a client-side cookie. Web Frameworks provide just the right amount of information and help to streamline this information.

(d) Abstract and simplify database access : Web Frameworks provide a data layer to help websites in abstracting the data read, write, query, and delete operations. Such an abstraction layer is referred to as an Object-Relational Mapper (ORM). This serves in two ways – to replace the underlying database without changing the code and basic validation of data which helps check the correct sequencing of data.

(e) Rendering data : Web Frameworks come along with templating systems which allow specifying the structure of an output document, making use of placeholders for data that are added when a page is generated and restored. Web Framework templates provide a mechanism to generate other document formats from stored data including JSON and HTML.

Q. 2 Define the term MVC and explain MVC Architecture with practical example. **(10 Marks)**

Ans. :

MVC

- The Model-View-Controller (MVC) is an architectural pattern that separates an application into three main logical components: the model, the view, and the controller.
- Each of these components are built to handle specific development aspects of an application. MVC is one of the most frequently used industry-standard web development framework to create scalable and extensible projects. MVC model was first introduced in 1987 in the Smalltalk programming language.

Model

- The Model component corresponds to all the data-related logic that the user works with. This can represent either the data that is being transferred between the View and Controller components or any other business logic-related data.
- For example, a Customer object will retrieve the customer information from the database, manipulate it and update it data back to the database or use it to render data.

View

- The View component is used for all the UI logic of the application. View is a user interface. View display data using model to the user and also enables them to modify the data.
- For example, the Customer view will include all the UI components such as text boxes, dropdowns, etc. that the final user interacts with.

Controller

- Controllers act as an interface between Model and View components to process all the business logic and incoming requests, manipulate data using the Model component and interact with the Views to render the final output.
- For example, the Customer controller will handle all the interactions and inputs from the Customer View and update the database using the Customer Model. The same controller will be used to view the Customer data.

MVC Architecture

- Normally in MVC architecture, when a user enters a URL in the browser, it goes to the webserver and routed to a controller.
- A controller executes related view and models for that request and create the response and sends it back to the browser.
- First user interacts with view through browser or some client. View send alert to controller about particular event happed on view. Controller takes event details. it calls the model and gathers the requested data and update the model. Model alerts view that it has changed, then view grabs the model data and update itself.

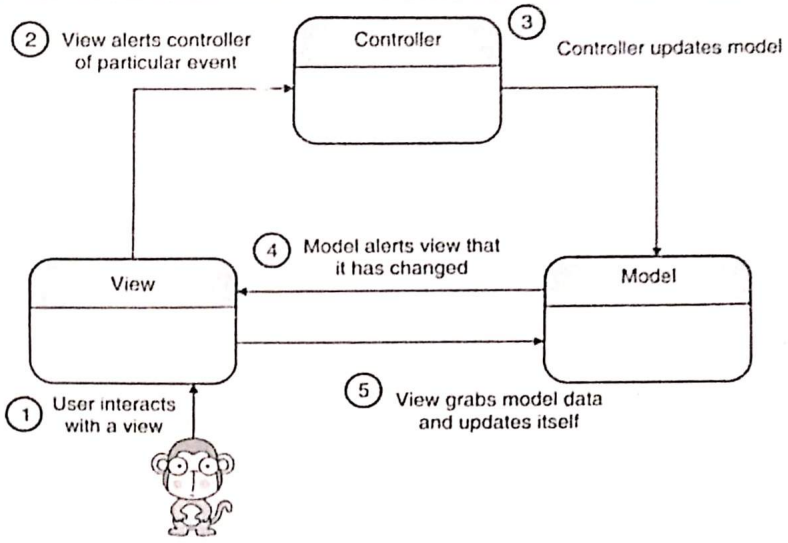


Fig. 3.1 : MVC Architecture

- The advantages of MVC architecture are as follows :
  1. MVC has the feature of scalability that in turn helps the growth of application.
  2. The components are easy to maintain because there is less dependency.
  3. A model can be reused by multiple views that provides reusability of code.
  4. The developers can work with the three layers (Model, View, and Controller) simultaneously.
  5. Using MVC, the application becomes more understandable.
  6. Using MVC, each layer is maintained separately therefore we do not require to deal with massive code.
  7. The extending and testing of application is easier.

MVC in Practical

Car driving mechanism is example of the MVC model, where every car consists of three main parts.

- View - User interface : (Gear lever, panels, steering wheel, brake, etc.)
- Controller - Mechanism (Engine)
- Model - Storage (Petrol or Diesel tank)

Car runs from engine take fuel from storage, but it runs only using mentioned user interface devices.

Example of MVC Design Pattern

```
class Student
{
   private String rollNo;
   private String name;

   public String getRollNo()
   {
      return rollNo;
```

```
    }

    public void setRollNo(String rollNo)
    {
      this.rollNo = rollNo;
    }

    public String getName()
    {
      return name;
    }

    public void setName(String name)
    {
      this.name = name;
    }
}
 class StudentView
{
   public void printStudentDetails(String studentName, String studentRollNo)
   {
     System.out.println("Student: ");
     System.out.println("Name: "+ studentName);
     System.out.println("Roll No: "+ studentRollNo);
   }
}

class StudentController
{
  private Student model;
  private StudentView view;
public StudentController(Student model, StudentView view)
  {
    this.model = model;
    this.view = view;
  }
  public void setStudentName(String name)
  {
```

```
        model.setName(name);
    }

    public String getStudentName()
    {
        return model.getName();
    }

    public void setStudentRollNo(String rollNo)
    {
        model.setRollNo(rollNo);
    }

    public String getStudentRollNo()
    {
        return model.getRollNo();
    }

    public void updateView()
    {
        view.printStudentDetails(model.getName(),  model.getRollNo());
    }
}
class MVCPattern
{
    public static void main(String[] args)
    {
        Student model  = retriveStudentFromDatabase();

        StudentView view = new StudentView();

        StudentController controller = new  StudentController(model, view);

        controller.updateView();

        controller.setStudentName("Vikram Sharma");

        controller.updateView();
```

```
    }

    private static Student retriveStudentFromDatabase()
    {
      Student student = new Student();
      student.setName("Tulshiram sule");
      student.setRollNo("1212");
      return student;
    }

}
```

Q. 3 What are the difference between TypeScript and JavaScript. **(4 Marks)**

Ans. :

Difference between TypeScript and JavaScript

TypeScript	JavaScript
It is an Object Oriented Language (Class based)	It is an Object Based Language (Prototype based)
Statically Typed language	Dynamically Typed language
Supports Modules	Does not Support Modules
Provides Errors at Compile time / during development	Doesn't provide Compile time errors
Takes more time as the code needs to be Compiled	No need of compilation

Q. 4 Explain variable and constant in type Script with example. **(6 Marks)**

Ans. :

Variables and Constants

- A variable, by definition, is "a named space in the memory" that stores values. In other words, it acts as a container for values in a program. TypeScript variables must follow the JavaScript naming rules :
- Variable names can contain alphabets and numeric digits.
- They cannot contain spaces and special characters, except the underscore (_) and the dollar ($) sign.
- Variable names cannot begin with a digit

Variables can be declared using : var, let, and const.

1. **Var** : Variables in TypeScript can be declared using var keyword, same as in JavaScript. The scoping rules remains the same as in JavaScript.
2. **Let** : To solve problems with var declarations, ES6 introduced two new types of variable declarations in JavaScript, using the keywords ***let and const.***

   TypeScript, being a superset of JavaScript, also supports these new types of variable declarations.

```
let employeeName = "John";
// or
let employeeName:string = "John";
```

The let declarations follow the same syntax as var declarations. Unlike variables declared with var, variables declared with let have a block-scope. This means that the scope of let variables is limited to their containing block, e.g. function, if else block or loop block.

3. **Const** : Variables can be declared using ***const*** similar to var or let declarations. The const makes a variable a constant where its value cannot be changed. Const variables have the same scoping rules as let variables.

   Example : Const Variable

```
const num:number = 100;
num = 200; //Compiler Error: Cannot assign to 'num' because it is a constant or read-only property
```

Const variables must be declared and initialized in a single statement. Separate declaration and initialization is not supported.

```
const num:number; //Compiler Error: const declaration must be initialized
num = 100;
```

Const variables allow an object sub-properties to be changed but not the object structure.

Q. 5 Explain Angular Architecture in detail. **(8 Marks)**

Ans. :

Angular Architecture

- Angular is a framework for building client applications in HTML and either JavaScript or a language like Type Script that compiles to JavaScript.
- The framework consists of several libraries, some of them core and some optional. You write Angular applications by composing HTML templates, writing component classes to manage those templates, adding application logic in services, and boxing components and services in modules.

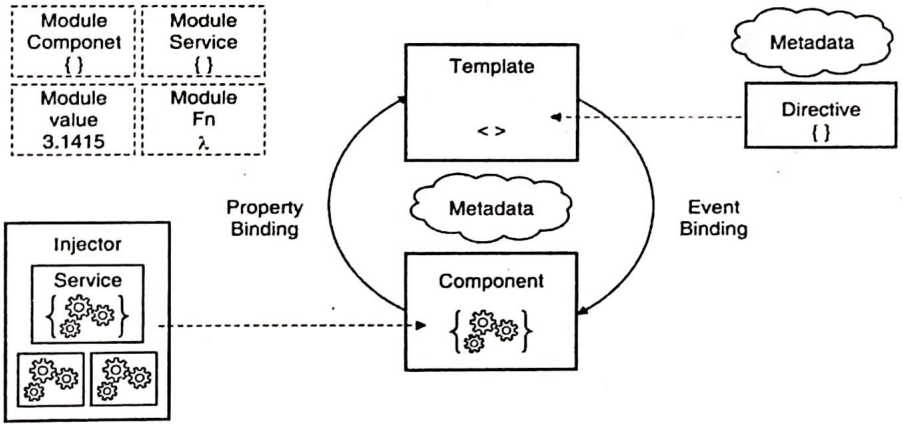


Fig. 3.2(a) : Angular architecture

The Eight main building blocks of an Angular application :

1. Modules
2. Components
3. Templates
4. Metadata
5. Data binding
6. Directives
7. Services
8. Dependency injection

1. Modules

- Every Angular app has a root module, conventionally named AppModule, which provides the bootstrap mechanism that launches the application. An app typically contains many functional modules.
- If we want to use another custom Angular module, then we need to register that module inside the app.module.ts file.
- Organizing your code into distinct functional modules helps in managing the development of complex applications, and in designing for re usability.
- Angular ships as a collection of JavaScript modules. You can think of them as library modules.
- Each Angular library name begins with the @angular prefix. You install them with the npm package manager and import parts of them with JavaScript import statements.
- For example, import Angular's Component decorator from the @angular/core library like this :

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

2. Components

- Every Angular project has at least one component, the root component and root component connects the component hierarchy with a page document object model (DOM). Each component defines the class that contains application data and logic, and it is associated with the HTML template that defines the view to be displayed in a target app. A component controls a patch of screen called a view.
- The @Component decorator identifies the class immediately below it as the component and provides the template and related component-specific metadata.

```
// app.component.ts@Component({
 selector: 'app-root',
 templateUrl: './app.component.html',
 styleUrls: ['./app.component.css']
})
```

3. Templates

- The angular template combines the HTML with Angular markup that can modify HTML elements before they are displayed. Template directives provide program logic, and binding markup connects your application data and the DOM.

4. Metadata

- Metadata tells Angular how to process a class. It is used to decorate the class so that it can configure the expected behavior of a class.
- Decorators are the core concept when developing with Angular . The user can use metadata to a class to tell Angular app that AppComponent is the component.
- Metadata can be attached to the TypeScript using the decorator.

```
// app.component.ts@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
```

- Here is the @Component decorator, which identifies the class immediately below it as a component class. The @Component decorator takes a required configuration object with the information Angular needs to create and present the component and its view.
- Here are a few of the most useful @Component configuration options :
  1. **selector** : CSS selector that tells Angular to create and insert an instance of this component where it finds a <app-root>tag which is Parent Component.
  2. **Template Url** : module-relative address of this component's HTML template.
  3. **providers** : array of dependency injection providers for services that the component requires.
- The metadata in the @Component tells Angular where to get the major building blocks you specify for the component. The template, metadata, and component together describe a view. Apply other metadata decorators in a similar fashion to guide Angular behavior. @Injectable, @Input, and @Output are a few of the more popular decorators.

5. Data binding

- Data binding plays an important role in communication between a template and its component.

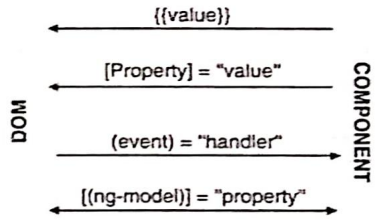


Fig. 3.2(b)

- Data binding is also important for communication between parent and child components. Angular allows defining communication between a component and the DOM, making it very easy to define interactive applications without worrying about pulling and pushing the data.
- There are two types of data binding :

1. **Event Binding** : Event binding is used to bind events to your app and respond to user input in the target environment by updating your application data.
2. **Property Binding** : Property binding is used to pass data from component class and facilitates you to interpolate values that are computed from your application data into the HTML.

6. Directives

- An Angular component isn't more than a directive with the template. When components are the building blocks of Angular applications, we are saying that directives are the building blocks of Angular projects. Let us use built-in Angular directive like ngClass, which is a better example of the existing Angular attribute directive.

```
<p [ngClass]="{'coffee'=true, 'red'=false}">
  Angular 10 Directives Example
</p><style>
  .coffee{color: coffee}
  .red{color: red}
</style>
```

- Here, based on the [ngClass] directive's value, the text has color. In our example, the text will be coffee because it is true.

7. Services

- For data or logic that isn't associated with a specific view, and that you want to share across components, you create a service class. The @Injectable decorator immediately precedes the service class definition. The decorator provides the metadata that allows your service to be injected into client components as a dependency.
- Angular distinguishes components from services to increase modularity and reusability.
- By separating a component's view-related functionality from other kinds of processing, you can make your component classes lean and efficient.

8. Dependency injection

- Dependency injection (DI) lets you keep your component classes lean and efficient. DI does not fetch data from a server, validate the user input, or log directly to the console instead they delegate such tasks to the services.
- DI is wired into a Angular framework and used everywhere to provide new components with the services or other things they need. Components consume services; that is, you can inject a service into a component, giving the component access to that service class.

Q. 6 Explain Angular Lifecycle with example. **(8 Marks)**

Ans. :

Angular Lifecycle

- Each Angular component goes through 8 phases in its lifecycle.
- When it is initialized, it creates and presents its root components. It is designed and it produces its heirs.
- For the components that get loaded during application development, it keeps checking when the data binding properties are getting changed and updated.
- When the component is not used anymore, it approaches the death phase and is decimated and expelled from the DOM.
- Your application can respond to events in the component lifecycle by using lifecycle hook methods.
- For instance, you may want to perform certain operations when the component is instantiated. Or perhaps you need to do some cleanup when the component is being destroyed.
- To respond to lifecycle events, you need to implement one or more of the lifecycle hook interfaces.
- Use these hook events in different phases of our application to obtains fine controls on the components.
- Since a component is a typescript class, for that reason every component must have a constructor method.
- The constructor of the component class executes first before the execution of any other life cycle hook event.

- If we need to inject any dependencies into the component, then the constructor is the best place to inject that dependency.
- After executing the constructor, Angular executes its life cycle hook methods in a specific order.
- Here is the complete list of life cycle hooks, which angular invokes during the component life cycle.

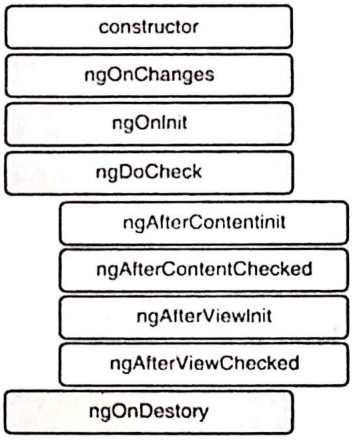


Fig. 3.3

1. **ngOnChanges**
   - This event executes every time when a value of an input control within the component has been changed.
   - Actually, this event fires first when a value of a bound property has been changed. It always receives a changed data map, containing the current and previous value of the bound property wrapped in a SimpleChange.
2. **ngOnInit**
   - This event initializes after angular first displays the data-bound properties or when the component has been initialized.
   - This event basically is called only once just after the ngOnChanges() events. This event is mainly used for initializing data in a component.
3. **ngDoCheck**
   - This event is triggered every time when the input properties of a component are checked. We can use this hook method to implement the check with our own logic check.
   - Basically, this method allows us to implement our own custom change detection logic or algorithm for any component.
4. **ngAfterContentInit**
   - This lifecycle method is executed when Angular performs any content projection within the component views.
   - This method executes only for the first time when all the bindings of the component need to be checked for the first time.
   - This event executes just after the ngDoCheck() method. This method is basically linked with the child component initializations.

5. **ngAfterContentChecked**
   - This lifecycle hook method executes every time when the content of the component has been checked by the change detection mechanism of the Angular.
   - This method called after ngAfterContentInit() method.
   - This method is also called on every subsequent execution of ngDoCheck(). This method is also mainly linked with the child component initializations.
6. **ngAfterViewInit**
   - This lifecycle hook method executes when the component's view has been fully initialized.
   - This method is initialized after Angular initializes the component's view and child views.
   - It is called only the first time after ngAfterContentChecked(). This lifecycle hook method only applies to components.
7. **ngAfterViewChecked**
   - This method is just called after the ngAterViewInit() method. It is executed every time the when the view of the given component has been checked by the change detection algorithm of Angular.
   - This method executes every subsequence execution of the ngAfterContentChecked().
   - This method also executes when any binding of the children directives has been changed. So this method is very useful when the component waits for some value which is coming from its child components.
8. **ngOnDestroy**
   - This method will be executed just before Angular destroys the components. This method is very useful for unsubscribing the observables and detaching the event handlers to avoid memory leaks.
   - Actually, it is called just before the instance of the component is finally destroyed.
   - This method called only once just before the component is removed from the DOM.

Example of Component Lifecycle

- The code below illustrates how component lifecycle hooks work.

```
export class SecondComponent implements  OnChanges, OnInit, DoCheck, AfterContentInit {
constructor(){
   alert('1. OnChanges() is called');
 }
 title ='Lifecycle Hooks Demo';
 ngAfterContentInit():void {
  alert('4. AfterContentInit() is called');
 }
 ngDoCheck():void {
  alert('3. ngOnDoCheck() is called');
 }
 ngOnInit():void {
  alert('2. ngOnInit() is called');
 }
 ngOnChanges(changes: import('@angular/core').SimpleChanges):void {
 }
}
```

Q. 7 Why Directives Required? Explain types of Directives. **(5 Marks)**

Ans. :

Directives

- Directives are functions that are executed when found in the DOM(Document Object Model) by the Angular compiler.
- They are used to extend the functionality of the regular HTML by adding new syntax.
- A directive modifies the DOM by changing the appearance, behaviour, or layout of DOM elements.

1. **Reusability** : In an Angular application, the directive is a self-sufficient part of the UI.

   As a developer, we can reuse the directive across the different parts of the application.

   This is very much useful in any large-scale applications where multiple systems need the same functional elements like search box, date control, etc.

2. **Readability** : Directive provides much more readability for the developers to understand the production functionality and data flow.
3. **Maintainability** : One of the main use of directive in any application is the maintainability. We can easily decouple the directive from the application and replace the old one with a new one directives.

Types of Directives

There are three types of directives :

1. **Component Directive**
   - Components are special kind of directive. So instead of manipulating the DOM, it comes with its own template.
   - However, behind the scenes, components use the directive API and provide a nicer way to define them. Just to add, the @Component annotation extends the @Directive annotation with template-related features.
2. **Attribute Directive**
   - Attribute directives are mainly used for changing the appearance or behavior of a component or a native DOM element. Attribute directives actually modify the appearance or behavior of an element. These directives actually act as a simple HTML attribute for any HTML tag. Attribute directives listen to and modify the behavior of other HTML elements, attributes, properties, and components.
   - The most common attribute directives are as follows:
     1. NgClass : adds and removes a set of CSS classes.
     2. NgStyle : adds and removes a set of HTML styles.
     3. NgModel : adds two-way data binding to an HTML form element.
3. **Structural Directive**
   - Structural directives are mainly used to change the design pattern of the UI DOM elements. In HTML, these directives can be used as a template tag.
   - Using this type of directives, we can change the structure of any DOM elements and can redesign or redecorate those DOM elements.
   - Structural directives are responsible for HTML layout. They shape or reshape the DOM's structure, typically by adding, removing, and manipulating the host elements to which they are attached.

- The most common built-in structural directives are as follows :
  - **NgIf :** Conditionally creates or disposes of subviews from the template.
  - **NgFor :** Repeat a node for each item in a list.
  - **NgSwitch :** A set of directives that switch among alternative views.

Q. 8 Write difference between component and directives. **(4 Marks)**

Ans. :

Component vs Directives

Sr. No.	Component	Directives
1.	A component is defined with the @ component decorator	A directive is defined with the @directives decorator.
2.	A component is a directive that uses a shadow DOM to create encapsulated visual behavior called a component. Components are typically used to create UI widgets.	Directive mainly used to provide new behavior within the existing DOM elements.
3.	With the help of the components, we can break down the application in multiple small parts.	With the help of the directive, we can design any type of reusable component.
4.	@view decorator or templateUrl template is mandatory in the component.	Directives don't use view.

Q. 9 Explain types of Pipes. **(6 Marks)**

Ans. :

Types of Pipes

- The pipes in Angular are of two types :

  1. Built-in Pipes 2. Custom Pipes

- The built-in pipes are those which are already given to us by the Angular and are ready to use. Whereas, in some scenarios, the built-in pipes are not sufficient to fulfill our use case. In such situations, create our own pipes, known as custom pipes.

1. Built-in Pipes

Angular contains built-in pipes for data transformations. The following are commonly used built-in pipes for data formatting

- DatePipe : Formats a date value.
- UpperCasePipe : Transforms text to uppercase.
- LowerCasePipe : Transforms text to lowercase.
- CurrencyPipe : Transforms a number to the currency string.
- PercentPipe : Transforms a number to the percentage string.
- DecimalPipe : Transforms a number into a decimal point string.

- **Example** : Observe the code for the uppercase pipe :

```
<h3>The name of the product is {{ name | uppercase }}</h3>
```

In the above code, the name of the product is converted to the uppercase using the UpperCasePipe. The way to implement or use a pipe is to put a pipe operator " | " right after the value that has to be transformed. After that, simply put the name of the pipe to be used on the value (uppercase in this scenario). Similarly, all other pipes can be used.

2. Custom Pipes

- Sometimes, you might need some functionality which just isn't built-in. In that case, you need to create your own pipe i.e you need to create a custom pipe.
- To create your own pipe you must create a new file. To create a pipe that shortens the length of the words if they exceed a given limit, say 15 letters. For this, first, we will create a pipe named shortenText (you can choose any name) by using the following terminal command :

```
ng generate pipe shortenText
```

Inside shortenText.pipe.ts file, we code in the following manner:

```
import { Pipe, PipeTransform } from '@angular/core';
@Pipe({
 name: 'shorten'
})
export class ShortenText implements PipeTransform {
 transform(value: any) {
  if(value.length > 15) {
   return value.substr(0,15);
  }
 return value;
 }
}
```

- In the above code, we used the Pipe decorator to tell Angular that we are creating a Pipe.
- The "name" property contains a value which will be used in the template through which we access this Pipe.
- It must be unique for all the Pipes present in an application. Then we have implemented an interface PipeTransform.
- It contains a method "transform()" that takes at least one parameter which is the value a pipe needs as an input. After that, it is a normal programming function and behaves according to the logic inside it.
- Now this pipe can be used just like built-in pipes in the following way :

```
<p>The product name is {{ name | shorten }}</p>
```

Q. 10 Explain Dependency Injections in Angular with example. **(10 Marks)**

Ans. :

Dependency Injections (DI)

- Dependency Injection (DI) is a technique in which a class receives its dependencies from external sources rather than creating them itself.

- Consider two classes, A and B. Let's assume that class A uses the objects of class B. Normally, in OOPS, an instance of class B is created so that class A can access the objects. Using DI, we move the creation and binding of the dependent objects outside of the class that depend on them.
- Angular dependency injection aims to decouple the implementation of services from components. This eases testing, overriding, and altering of services without affecting the components dependent on these services.

Types of Dependency Injection

- There are three types of Dependency Injections in Angular, they are as follows:
  1. Constructor injection : Here, it provides the dependencies through a class constructor.
  2. Setter injection : The client uses a setter method into which the injector injects the dependency.
  3. Interface injection : The dependency provides an injector method that will inject the dependency into any client passed to it.
- On the other hand, the clients must implement an interface that exposes a setter method that accepts the dependency.

Example

Step 1 : Creating an Angular Service using Angular CLI

Let's see how to use Angular CLI to generate a service. Open a new command-line interface, navigate to your project's folder and run the following command :

```
$ ng generate service data
```

The command will create the following files in the src/app folder of our project:

- src/app/data.service.spec.ts
- src/app/data.service.ts

Open the *src/app/data.service.ts* file, you should find the following initial code :

```
import { Injectable } from '@angular/core';
@Injectable({
 providedIn: 'root'
})
export class DataService {
 constructor() { }
}
```

- This is a typical TypeScript class decorated with the *@Injectable()* decorator that tells the Angular dependency injector that the class can be provided and injected as a dependency in the other components or services.
- The Injectable decorator takes a property called *provideIn* that have by default the *root* value. The *providedIn* property tells the Angular dependency injection with the "scope" of our service in the application i.e where it can be provided.

Step 2 : Implementing the method(s) of our service

- For sending HTTP requests, we need to import the *HttpClientModule* which provides the builtin *HttpClient* service.Open the *src/app/app.module.ts* file and import *HttpClientModule* then add it the *imports* array as follows:

```
import { HttpClientModule } from '@angular/common/http';

@NgModule({
```

```
  imports: [
    HttpClientModule
  ]
})
```

- Since *HttpClient* is also a service, we can inject it in our data service via the constructor as follows :

```
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs';
import { HttpClient } from '@angular/common/http';

@Injectable({
  providedIn: 'root'
})
export class DataService {

  apiUrl: string = 'YOUR_API_URL';
  constructor(private httpClient: HttpClient) { }

  fetchData(): Observable<any> {
    return this.httpClient.get(this.apiUrl)
  }
}
```

Step 3 : Accessing the service methods from components

- After defining and implementing the service, you need to inject it into the component(s) where you need to use it.
- Let's suppose that we want to use the service in the app component.
- Open the *src/app/app.component.ts* file and import the service as follows:

```
import { DataService } from './data.service';
```

Next, you need to inject the service class via the constructor of the component as follows:

```
export class AppComponent {
  constructor(private dataService: DataService) { }
}
```

- Next, we can now call the fetchData() method of our service, for example inside the *ngOnInit()* lifecycle hook of the component:

```
import { Component, OnInit } from '@angular/core';
import { DataService } from './data.service';

@Component({
  selector: 'app-root',
```

```
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent implements OnInit {
  constructor(private dataService: DataService) { }

  ngOnInit() {
    this.dataService.fetchData().subscribe(data => {
      console.log(data);
    });
  }
}
```

- Since the *get()* method of *HttpClient* returns an RxJS observable which gets returned from the *fetchData()* method, we need to subscribe to it in order to send the actual GET request and receive the response.

Q. 11 Write a short note on : Angular Routers. **(5 Marks)**

Ans. :

Angular Routers

- Routing in Angular allows the users to create a single-page application with multiple views and allows navigation between them. Users can switch between these views without losing the application state and properties.
- Routing in Angular is the way pages web pages are served based on a given url. Since Angular focuses on SPA, the URLs are not served from the server and also don't reload a page. The URLs are local in the browser and pages are served locally. Therefore, when a a URL is requested, the Angular router navigates to the new component. Then it renders it's template and finally updates the history.
- A route in Angular is a pair of a url pattern and a component. Such that when a requested url matches the pattern, then a component is loaded.

1. Configuring Angular Routing

The file angular.routing.ts is responsible for routing in an Angular application. Follow the steps below to configure routing:

Step 1 : Import the AppRoutingModule into the AppModule. Also add it to the imports array. Normally this is done automatically by Angular CLI if routing is enabled for your application.

```
imports: [
  BrowserModule,
  AppRoutingModule
],
```

Step 2 : Import the RouterModel and Routes into app.routing.module.ts. Also done automatically.

```
@NgModule({
 imports: [RouterModule.forRoot(routes)],
 exports: [RouterModule]
})
```

Step 3 : Define the routes in the Routes array. The code below show route for three components

```
const routes: Routes = [
  { path:'first-component', component: FirstComponent },
  { path:'second-component', component: SecondComponent },
  { path:'third-component', component: ThirdComponent },
];
```

Step 4 : Add the links to your template. This code below should be placed in the root component of the application (app.component.html).

```
<h1>Angular Routing Demo</h1>
<nav>
<ul>
<li><arouterLink="/first">1st Component</a></li>
<li><arouterLink="/second">2nd Component</a></li>
<li><arouterLink="/third">3rd Component</a></li>
</ul>
</nav>
<!-- The routed views are rendered the <router-outlet>-->
<router-outlet></router-outlet>
```

2. Passing Information to Components

- Sometimes, you may want to pass information from one component to another.
- For instance, in a shopping application that displays a list of products. When the user clicks on edit, the ProductEdit component is loaded.
- How do you pass data from the parent component to the ProductEdit component? The productId for example. You achieve this using the ActiveRoute interface. Here are the steps to follow :

Step 1 : Import the ActiveRoute and ParamMap interfaces into your component. Like so

```
import { Router, ActivatedRoute,  ParamMap } from '@angular/router';
```

Step 2 : Inject an instance of the ActiveRoute. Just add an ActiveRoute field to your component constructor.

```
constructor(
private route: ActivatedRoute,
) {}
```

Step 3 : In the ngOnInit() method, you can access the ActiveRoute and get the value of the productId variable are as follows :

```
ngOnInit() {
this.route.queryParams.subscribe(params => {
this.name = params['name'];
  });
}
```

3 Handling Errors

- When a user requests a path that does not exist, you should be able to handle this as well. You can achieve this by either using a Wildcard route and displaying a 404 page.

- A wildcard routes represents a path using **. Then the component could be a custom component that displays a friendly error message.

```
{ path:'**', component: PageNotFoundComponent },          // Wildcard route
```

- This route is selected if the requested url does not match any of the paths specified.

4. **Redirects**
   - You can also easily setup redirects in the same way. You simply specify the path you want to redirect from and the path you are redirecting to.

```
{ path:'/second', redirectTo:'/first', pathMatch:'full' },
```

   - Here, when the user visits /second, he will be redirected to /first

---

Q. 12 What are the features of react? **(4 Marks)**

Ans. :

Features of React

Some essential and most demanding features are as follows :

1. **JSX**
   - JSX means JavaScript XML. It is an extension of JS language syntax. It provides a way to render components using a syntax similar to HTML. It uses JSX to write its components.
   - It can use pure JavaScript also but prefers JSX. Babel uses it, a pre-processor, to convert text similar to HTML found in JavaScript files into standard JS objects.
   - The HTML code can be embedded in JavaScript to make HTML code more easily understandable, enhance the performance of JavaScript, and make the application robust.
2. **Virtual Document Object Model**
   - React make an in-memory data structure cache, then it computes the difference between the previous DOM and the new one and then updates the changes or mutations performed.
   - Thus, it updates only the changes, not the whole application. This helps to increase speed and performance and reduces memory wastage.
3. **Testability**
   - React views are used as the state's functions where the state determines the behavior of the component.
   - Therefore we can make changes to the state and then pass it to a view the ReactJS and then determine the output and the actions, functions, and events. This makes testing and debugging easy.
4. **SSR**
   - It stands for Server- Side Rendering. It permits to pre-render the initial state of the components at the server-side.
   - The browser can render without waiting for all the JavaScript to be executed or loaded. This makes web pages to load faster.
   - It helps the user to view the web pages even when React is still downloading the JavaScript, linking events or creating virtual DOM at the backend.
5. **One Way Data Binding**
   - It allows one-way flow of data, i.e. one-way data binding. Due to this feature, there is better control over the application.

- It makes the application's state to be contained in specific stores, and therefore all other components remain loosely coupled. This enhances the flexibility and efficiency of the application.

6. **Simplicity**
   - JSX files make the application simple and understandable. Standard JavaScript can be used to code, but the usage of JSX makes it easier. Several lifecycle methods and its component-based approach make it simpler to learn and execute.

Q. 13 Explain Inter Components Communication in details. **(6 Marks)**

Ans. :

Inter Components Communication

- React is a component-based UI library. When the UI is split into small, focused components, they can do one job and do it well. But in order to build up a system into something that can accomplish an interesting task, multiple components are needed. These components often need to work in coordination together and, thus, must be able to communicate with each other. Data must flow between them.
- React components are composed in a hierarchy that mimics the DOM tree hierarchy that they are used to create. There are those components that are higher (parents) and those components that are lower (children) in the hierarchy.

A. From Parent to Child with Props

- The simplest form of communication between components is via properties usually called props. Props are the parameters passed into child components by parents, similar to arguments to a function. Props is a bag of data, an object that can contain any number of fields.
- If a parent component wants to feed data to a child component, it simply passes it via props. Let's say that we have a BookList component that contains data for a list of books. As it iterates through the book list at render time, it wants to pass the details of each book in its list to the child Book component. It can do that through props. These props are passed to the child component as attributes in JSX.

```
function BookList(){
const list =[
{ title:'Web programming', author:'Tulshiram Sule'},
{ title:'Software development', author:'Poonam Sule'},
// ...
]

return(
<ul>
{list.map((book, i)=><Book title={book.title} author={book.author} key={i}/>)}
</ul>
)
}
```

- Then the Book component can receive and use those fields as contained in the props parameter to its function :

```
function Book(props) {
 return (
<li>
<h2>{props.title</h2>
<div>{props.author}</div>
</li>
 )
}
```

- Favor this simplest form of data passing whenever it makes sense.
- There is a limitation here, however, because props are immutable. Data that is passed in props should never be changed

B. From Child to Parent with Callbacks

- For a child to talk back to a parent (unacceptable, I know!), it must first receive a mechanism to communicate back from its parent. Parents pass data to children through props. A "special" prop of type function can be passed down to a child. At the time of a relevant event (eg, user interaction) the child can then call this function as a callback.
- A book can be edited from a BookTitle component :

```
function BookTitle(props) {
 return (
<label>
   Title:
<input onChange={props.onTitleChange} value={props.title} />
</label>
 )
}
```

- It receives a onTitleChange function in the props, sent from its parent. It binds this function to the onChange event on the <input /> field. When the input changes, it will call the onTitleChange callback, passing the change Event object.
- Because the parent, BookEditForm, has reference to this function, it can receive the arguments that are passed to the function :

```
import React, { useState } from 'react'
function BookEditForm(props) {
 const [title, setTitle] = useState(props.book.title)
 function handleTitleChange(evt) {
 setTitle(evt.target.value)
 }
 return (
<form>
```

```
<BookTitle onTitleChange={handleTitleChange} title={title} />
</form>
)
}
```

- In this case, the parent passed handleTitleChange, and when it's called, it sets the internal state based on the value of evt.target.value -- a value that has come as a callback argument from the child component.

C. From Parent to Child with Context

- If we desire something to be globally available -- in many components and levels in the hierarchy -- props passing has the potential to be cumbersome. Think of some data that we might like to broadcast to all child components that they react to wherever they are, such as theming data. Instead of passing theme props to every component down the tree or a subtree in the hierarchy, we can define a theme context to be provided at the top and then consume it in whichever child needs it down the line.
- A list of books in BookList and had a parent component above that called BookPage. In that component we could provide a context for the theme:

```
const ThemeContext=React.createContext('dark')

function BookPage(){
return(
<ThemeContext.Provider value="light">
<BookList/>
</ThemeContext.Provider>
)
}
```

- The ThemeContext need only be created once and, thus, is created outside the component function. It is given a default of "dark" as a fallback theme name.
- The context object contains a Provider function which we wrap our rendered child component in.
- Specify a value to override the default theme. Here, we are saying our BookPage will always show the "light" theme.

```
import React, { useContext } from 'react'

function Book(props) {
  const theme = useContext(ThemeContext)
  const styles = {
    dark: { background: 'black', color: 'white' },
    light: { background: 'white', color: 'black' }
  }
  return (
<li style={styles[theme]}>
<h2>{props.title</h2>
<div>{props.author}</div>
</li>
  )
}
```

- Book needs to have access to the ThemeContext object created next to BookPage, and that context object is fed to the contextType book.
- From there, we create a simple styles map and select the appropriate styling based on the value of theme.
- The thing that's special about context is that theme did not come from props but rather was simply available because a parent component provided it to any and all children which used it.
- As with most global code patterns, use context sparingly. It creates coupling between components that can lead to less-reusable code and relationships or between components that are less clear.

Q. 14 Explain different ways to styling React js components. (5 Marks)

Ans. :

Components Styling

There are different ways to styling React js components.

1. Inline CSS	2. Normal CSS
3. CSS in JS	4. Styled Components
5. CSS modules	6. Sass & SCSS
7. Less	8. Stylable

1. **Inline CSS** : In inline styling basically we create objects of style. And render it inside the components in style attribute using the React technique to incorporate JavaScript variable inside the JSX (Using '{}')
2. **Normal CSS** : In the external CSS styling technique, we basically create an external CSS file for each component and to the required styling of classes and use those class names inside the component. It is a convention that name of the external CSS file same as the name of the component with '.css' extension.
3. **CSS in JS** : The 'react-jss' integrates JSS with react app to style components.

   It helps to write CSS with JavaScript and allows us to describe styles in a more descriptive way. It uses javascript objects to describe styles in a declarative way using 'createUseStyles' method of react-jss and incorporate those styles in functional components using className attribute.
4. **Styled Components** : The styled-components allows us to style the CSS under the variable created in JavaScript. style components is a third party package using which we can create a component as a JavaScript variable that is already styled with CSS code and used that styled component in our main component. styled-components allow us to create custom reusable components which can be less of a hassle to maintain. Command to install third-party styled-components package: npm install --save styled-components
5. **CSS Modules** : A CSS module is a simple CSS file but a key difference is by default when it is imported every class name and animation inside the CSS module is scoped locally to the component that is importing it also CSS file name should follow the format 'filename.module.css'. This allows us to use a valid name for CSS classes without worrying about conflicts with other class names in your application.

   To use CSS modules create a normal CSS file, import the module you have created from within your component using the syntax:

   import styles from './filename.module.css'
6. **Sass and SCSS** : Less is the most stable, powerful, professional-grade CSS extension language. It's a CSS preprocessor that adds special features such as variables, nested rules, and mixins or syntactic sugar in regular CSS. The aim is to make the coding process simpler and more efficient. Sass files are executed on the server and sends CSS to the browser.

---

7. **Less :** Less (Leaner Style Sheets) is an open-source, dynamic preprocessor style sheet language that can be compiled into CSS and run on the client side or server side. It takes inspiration from both CSS and Sass and is similar to SCSS. A few notable differences include variables starting with an @ sign in Less and with a $ in Sass.
8. **Stylable :** Stylable a preprocessor that enables you to scope styles to components so they don't leak and clash with other styles elsewhere in your app. It comes with several handy features such as the ability to define custom pseudo-classes so that you can apply styles to your components based on state. It is also inspired by Typescript, with the project's home page stating

Q. 15 Explain Redux- Architecture. **(5 Marks)**

Ans. : Redux Architecture

- Redux architecture revolves around a strict unidirectional data flow.
- This means that all data in an application follows the same lifecycle pattern, making the logic of your app more predictable and easier to understand. It also encourages data normalization, so that you don't end up with multiple, independent copies of the same data that are unaware of one another.
- **React :** Redux provides a store which makes the state inside components easier to maintain. Along with stores, react-redux introduces actions and reducers which work simultaneously with stores to make the state more predictable.
- The components of Redux architecture are as follows :
  1. **Store :** It contains the state of the components which need to be passed to other components. The store makes this passing along much easier as we no longer need to maintain a state inside a parent component in order to pass the same to its children components.
  2. **Actions :** The actions part of the react-redux basically contains the different actions that are to be performed on the state present in the store. The actions included must contain the type of the action and can also contain payload
  3. **Reducers :** The reducers in react-redux are the pure functions that contain the operations that need to be performed on the state. These functions accept the initial state of the state being used and the action type. It updates the state and responds with the new state. This updated state is sent back to the view components of the react to make the necessary changes.

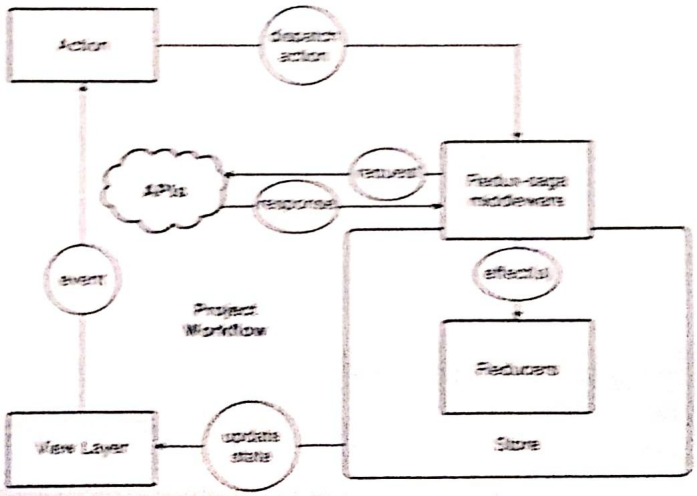


Fig. 3.4 : Redux architecture with Redux-Saga

- In this architecture data flow is as follows :
  - Component handles an event and dispatches an action
  - Redux calls the reducer functions
  - Related reducer function updates the state
  - Containers that are listening for the change of that state slice, gets notified
  - Containers updates Component with the new state and props
  - Component gets re-rendered if props are changed

Reduces Time : React-redux refreshes the parts of a page rather than reloading the whole page and therefore saves us time.

Effortless Debugging : React-redux introduces reducers which are pure functions operating on the state which makes the logic simpler and helps in achieving effortless debugging.

Q. 16 Explain basic Hooks in ReactJs. **(10 Marks)**

Ans. : Basics of Hooks

- Hooks are the new feature introduced in the React 16.8 version. It allows you to use state and other React features without writing a class. Hooks are the functions which "hook into" React state and lifecycle features from function components.
- It does not work inside classes. Hooks are backward-compatible, which means it does not contain any breaking changes.

Rules for Hooks

1. Never call Hooks from inside a loop, condition or nested function
2. Hooks should sit at the top-level of your component
3. Only call Hooks from React functional components
4. Never call a Hook from a regular function
5. Hooks can call other Hooks

Built-in Hooks

The built-in Hooks can be divided into two parts, which are given as follows :

Basic Hooks

- useState
- useEffect
- useContext

1. useState() hook

- The state of your application is bound to change at some point. This could be the value of a variable, an object, or whatever type of data exists in your component.
- To make it possible to have the changes reflected in the DOM, we have to use a React hook called useState.
- useState() hook allows functional components to have a dedicated state of their own.
- Whenever the useState() hook is used, the value of the state variable is changed and the new variable is stored in a new cell in the stack.Hook state is the new way of declaring a state in React app.
- Hook uses useState() functional component for setting and retrieving state.

```
import React, { useState } from 'react';
function CountApp() {
// Declare a new state variable, which we'll call "count"
 const [count, setCount] = useState(0);
 return (
<div>
<p>You clicked {count} times</p>
<button onClick={() => setCount(count + 1)}>
    Click me
</button>
</div>
 );
}
export default CountApp;
```

In the above example, useState is the Hook which needs to call inside a function component to add some local state to it. The useState returns a pair where the first element is the current state value/initial value, and the second one is a function which allows us to update it. After that, we will call this function from an event handler or somewhere else. The useState is similar to this.setState in class.

2. useEffect() Hook

- useEffect is short for 'use side effect'. useEffect Hook is used to eliminate the side-effects of using class-based components.
- For example, tasks like updating the DOM, fetching data from API end-points, setting up subscriptions or timers, etc can be lead to unwarranted side-effects.
- Since the render method is to quick to produce a side-effect one needs to use life cycle methods to observe the side effects.
- The Effect Hook helps you perform side effects in a functional component.
- The motivation behind the introduction of useEffect Hook is to eliminate the side-effects of using class-based components.
- React Hooks introduces the useEffect() method to replace a class component's lifecycle methods componentDidMount, componentDidUpdate, and componentWillUnmount.
- The method also allows side effects in your functional component, such as changing content in the document object model and fetching data. useEffect() will run after every component render.
- useEffect accepts two arguments. The second argument is optional.

  useEffect(<function>, <dependency>)

Example : Timer as an example

```
import { useState, useEffect }from"react";
import  ReactDOM from"react-dom";
function Timer(){
const [count, setCount]=useState(0);
useEffect(()=>{
```

```
setTimeout(() => {
setCount((count) => count + 1);
},1000);
});
return <h1>I've rendered {count} times!</h1>;
}
ReactDOM.render(<Timer />, document.getElementById('root'));
```

- useEffect runs on every render. That means that when the count changes, a render happens, which then triggers another effect.
- There are several ways to control when side effects run. We should always include the second parameter which accepts an array. We can optionally pass dependencies to useEffect in this array.

3. useContext() hook

- The main idea of using the context is to allow your components to access some global data and re-render when that global data is changed. Context solves the props drilling problem: when you have to pass down props from parents to children.
- You can hold inside the context:
  - global state
  - theme
  - application configuration
  - authenticated user name
  - user settings
  - preferred language
  - a collection of services
- Context provides a way to share values between components without having to explicitly pass a prop through every level of the tree.
- React's useContext hook makes it easy to pass data throughout your app without manually passing props down the tree.
- Generally, when we have two or more levels(height) in our component tree, it is viable to use a store instead of passing props and then lifting the state as this will create confusion and unnecessary lengthy code.
- To create context, you must import createContext and initialize it. Secondly use the Context Provider to wrap the tree of components that need the state Context.
- Wrap child components in the Context Provider and supply the state value. In order to use the Context in a child component, we need to access it using the useContext Hook.

```
import { useState, createContext, useContext }from"react";
import ReactDOM from"react-dom";
const UserContext =createContext();
function Component1(){
const[user, setUser]=useState("Jesse Hall");
return(
<UserContext.Provider value={user}>
<h1>{`Hello ${user}!`}</h1>
```

```
<Component2user={user}/>
</UserContext.Provider>
);
}
functionComponent2(){
return(
<>
<h1>Component 2</h1>
<Component5/>
</>
);
}
function Component5(){
const  user =useContext(UserContext);
return(
<>
<h1>Component 5</h1>
<h2>{`Hello ${user} again!`}</h2>
</>
);
}
ReactDOM.render(<Component1/>, document.getElementById("root"));
```

□□□

Unit IV : Back End Technologies

Q. 1 Write short note on NodeJS. SPPU - May 18, 4 Marks

Ans. :

Node JS

- Node JS is a runtime library and environment which is cross-platform and used for creating running JavaScript applications outside the browser.
- It is free and open-source and utilized for creating server-side JS applications.
- Node JS allows developers to execute their code on the server-side. It provides a faster way to write scripts that are scalable and light. Developers can write real-time applications, and at the same time, it provides scope for mobile application development.
- One can easily utilize Node JS for the front end as well as for back-end development as it allows the use of the same JavaScript.
- Server-side capabilities are provided extensively in Node JS, a developer can listen to and reply to HTTP requests on the computer, listen to traffic network and at the same time can access the database from a computer directly.
- Node JS uses an event-based model to address scalability, and allow rich JavaScript libraries for JavaScript modules which helps in simplify the coding.
- There are plenty of frameworks based on Node JS such as Express JS, Partial JS, etc. Basically, Node JS gives JavaScript the ability to interact with I/O (input/output) devices through its APIs, and connect with other external libraries written in various other languages.

Features of Node.js

Some of the key features of Node.js.

1. **Asynchronous event-driven IO helps concurrent request handling** : This is probably the most significant selling point of Node.js.

   This feature basically means that if a request is received by Node for some Input/Output operation, it will execute the operation in the background and continue with processing other requests.

2. Node uses the V8 JavaScript Runtime engine, the one which is used by Google Chrome. Node has a wrapper over the JavaScript engine which makes the runtime engine much faster and hence the processing of requests within Node also become faster.

3. **Handling of concurrent requests :** Another key functionality of Node is the ability to handle concurrent connections with a very minimal overhead on a single process.

4. **The Node.js library uses JavaScript :** This is another important aspect of development in Node.js. A major part of the development community is already well versed in JavaScript, and hence, development in Node.js becomes easier for a developer who knows JavaScript.

5. There is an active and vibrant community for the Node.js framework. Because of the active community, there are always keys updates made available to the framework.

   This helps to keep the framework always up-to-date with the latest trends in web development.

Q. 2 What is the difference between JavaScript and Node.js? **(4 Marks)**

Ans. :

Difference between JavaScript and Node

Parameters	JavaScript	Node JS
Type	JavaScript is a programming language. It running in any web browser with a proper browser engine.	It is an interpreter and environment for JavaScript with some specific useful libraries which JavaScript programming can use separately.
Utility	Mainly using for any client-side activity for a web application, like possible attribute validation or refreshing the page in a specific interval or provide some dynamic changes in web pages without refreshing the page.	It mainly used for accessing or performing any non-blocking operation of any operating system, like creating or executing a shell script or accessing any hardware specific information or running any backend job.
Running Engine	JavaScript running any engine like Spider monkey (FireFox), JavaScript Core (Safari), V8 (Google Chrome).	Node JS only run in a V8 engine which mainly used by google chrome. And javascript program which will be written under this Node JS will be always run in V8 Engine.

Q. 3 Explain Node.JS functions with example. **(6 Marks)**

Ans. :

Node.JS Functions

A. Defining and Calling a Function

NodeJs functions are exactly similar to functions in JavaScript. The structure of a function is given below:

```
function functionName(parameters){
//body of function
//optional return statement
}
```

- Now, functions in Node.js must return a value. So function definitions include an optional return statement. However, if the return statement is missing, then the function returns undefined.
- After a function is defined, you can then call it. When you call a function, then it begins executing. Let's take an example. Write a simple function that displays a greeting.

```
//Defining a function
function sayHello(){
  console.log("Hello Programmer!")
  console.log("Hope you're doing great!")
}
//we call the function here
sayHello()
```

B. Parameters to Function

- A parameter to a function is enclosed in brackets after the functionName. A parameter is a variable used by a function during execution.
- For example, the program below is a function that takes on parameter name, and prints out a message using the name.
- This means that the value of the parameter must be provided during the function call.

```
//Function definition
function greeting(name){
   console.log("Good morning "+ name)
}
//function call
greeting("tulshiram")
```

C. Function Scope

- Scope means the block of code where a variable is visible. So anytime a function is defined, the function block becomes a scope.
- Variables defined inside the function block is in the function scope.
- Therefore, these variables are not visible (or available) outside the function block.
- However, variables defined outside the function are visible inside the function scope.
- For example.

```
var scooter ='OLA';
function myFunction() {
    var scooter = 'TVS';
    console.log(scooter); // scooter in function scope
}
myFunction();
console.log(scooter); // scooter outside the function scope
```

- The scooter inside the function scope has the value 'TVS' while the scooter outside the function has a value of 'OLA'.
- That is if you have a variable with the same name inside and outside a function, then the variable declared inside has precedence within the function scope.

Q. 4 Explain types of modules in NodeJs. **(5 Marks)**

Ans. :

Node.JS Built-in Modules

- Module in Node.js is a simple or complex functionality organized into single or multiple JS files which can be used again throughout the Node.js application.
- Node.js is a light weight framework. The core modules include bare minimum functionalities of Node.js.
- These core modules are compiled into its binary distribution and load automatically when Node.js process starts. However, you need to import the core module first in order to use it in your application.

Type of Modules in Node JS

A. Core (Built-in) Modules
B. Local (Custom) Modules
C. Third Party (External) Modules

A. Core Modules

- Node.js has many built-in modules that are part of the platform and comes with Node.js installation.
- These modules can be loaded into the program by using the require function.

```
var module = require('module_name');
```

- The require() function will return a JavaScript type depending on what the particular module returns. The following list contains some of the important core modules in Node.js.

1. **http** : Creates an HTTP server in Node.js.
2. **assert** : Set of assertion functions useful for testing.
3. **Fs** : Used to handle file system.
4. **Path** : Includes methods to deal with file paths.
5. **Process** : Provides information and control about the current Node.js process.
6. **Os** : Provides information about the operating system.
7. **Querystring** : Utility used for parsing and formatting URL query strings.
8. **url** : Module provides utilities for URL resolution and parsing.

B. Local Modules

Unlike built-in and external modules, local modules are created locally in your Node.js application. Let's create a simple calculating module that calculates various operations. Create a calc.js file that has the following code :

```
exports.add = function (x, y) {
    return x + y;
};
exports.sub = function (x, y) {
    return x - y;
};
exports.mult = function (x, y) {
    return x * y;
};
exports.div = function (x, y) {
    return x / y;
};
```

Since this file provides attributes to the outer world via exports, another file can use its exported functionality using the require() function.

Filename: index.js

```
var calculator = require('./calc');
var x = 50, y = 10;
console.log("Addition of 50 and 10 is " + calculator.add(x, y));
```

C. Thirdparty modules

Third-party modules are modules that are available online using the Node Package Manager (NPM). These modules can be installed in the project folder or globally. Some of the popular third-party modules are mongoose, express, angular, and react.

Example

- npm install express
- npm install mongoose

Q. 5 Explain Node Package Manager. **(5 Marks)**

Ans. : NPM

- NPM (Node Package Manager) is the default package manager for Node.js and is written entirely in JavaScript. Developed by Isaac Z. Schlueter, it was initially released in January 12, 2010.
- NPM manages all the packages and modules for Node.js and consists of command line client npm. It gets installed into the system with installation of Node.js. The required packages and modules in Node project are installed using NPM.
- NPM can install all the dependencies of a project through the package.json file. It can also update and uninstall packages. In the package.json file, each dependency can specify a range of valid versions using the semantic versioning scheme, allowing developers to auto-update their packages while at the same time avoiding unwanted breaking changes.
- NPM consists of three components:
- The website allows you to find third-party packages, set up profiles, and manage your packages.
- The command-line interface or npm CLI that runs from a terminal to allow you to interact with npm.
- The registry is a large public database of JavaScript code

A. Installing Modules using NPM

There is a simple syntax to install any Node.js module

```
$ npm install <Module Name>
```

For example, following is the command to install a famous Node.js web framework module called express –

```
$ npm install express
```

Use the following command to uninstall a Node.js module.

```
$ npm uninstall express
```

The npm list command outputs installed packages and their dependencies of the current project as a tree-structure to the stdout

```
$ npmnpm list
```

B. semantic versioning in npm

To make the JavaScript ecosystem more reliable and secure, you should update the version number in the package.json file that follows the semantic versioning specification:

```
major.minor.patch
```

- The major version when you make changes that are not compatible with the previous version.
- The minor version when you add a backward-compatible feature with the previous version.
- The path version when you make bug fixes that are backward-compatible with the previous version.
- For example: In this version: major version is 4, minor version is 17 and patch is 2.

Q. 6 Why to use NodeJS for creating Microservices? **(5 Marks)**

Ans. : Microservices

- A microservice is an application architecture that takes every application function and puts it in its own service, isolated from the others.
- These services are loosely coupled and independently deployable.

A. Why NodeJS for Microservices?

When building a microservice, there are a lot of top programming languages to choose from. One of them is NodeJS. here are several reasons

- **Improved execution time** - The V8 JavaScript engine that powers Node.js compiles functions written in JavaScript to native machine code. This makes Node.js a popular choice for building microservices that carry out low-latency CPU and IO-intensive operations.
- **Event-driven architecture** - Most of the objects in Node.js provide a way to emit and listen to events making it highly beneficial in building event-driven apps. NodeJS uses event-driven architectures built on a common pattern in software development known as publish-subscribe or observer pattern that enables powerful applications to be built, especially real-time applications.
- **Asynchronous and non-batch** - Most Node.js libraries support non-blocking calls which continue execution without waiting for the previous call to return. Additionally, data is returned as-is, streamed as chunks, and not batched in a buffer. NodeJS uses the event loop to execute the code, allowing asynchronous code to be executed, thereby enabling the server to use a non-blocking mechanism to respond.
- **Scalable technology** - The Node.js execution model supports scaling by delegating request handling to other worker threads. The runtime environment is built on top of one of the most powerful JavaScript engines, V8 JavaScript Engine, thus the code is executed quickly and facilitates the server to handle up to 10,000 concurrent requests.

B. Create Microservices with Node.js

Step 1 : Server Setup

Let us first start with original file server.js

```
var express = require('express');
var app = express();
var port =process.env.PORT || 3000;
var routes = require('./api/routes'); routes(app);
app.listen(port,function(){ console.log('Server started on port: ' + port);
});
```

Step 2 : Identify Routes

We will identify routes for the server and allocate every route to a target in the controller object. The routes will be defined by two endpoints to receive and send off requests.

For this project, we will use the ZIP codes as the two parameters to mark paths for distance endpoints.

```
'use strict';
var controller = require('./controller');
module.exports = function(app) {
  app.route('/about')
    .get(controller.about);
```

```
  app.route('/distance/:zipcode1/:zipcode2')
    .get(controller.get_distance);
};
```

- The first route transmits the GET requests on /about the endpoint. These requests are managed by the about function in the controller.
- The second route added to the app transmits GET requests on the /distance endpoint. These requests are managed by get_distance function in the controller.

Step 3 : Create Controller Logic

- The controller logic added to the microservices for inducing a few new functions in it. A controller object built on understanding user actions & intentions and establishing communication with the new or modified data for processing objects.
- For our project, we are developing a controller object with two properties. These two properties are functions to manage the requests transmitted by the routes module.

```
'use strict';
var properties = required('../package.json');
var distance = require('../service/distance');
var controllers = {
 about: function(req,res){
    var aboutInfo = { name:properties.name, version: properties.version }
res.json(aboutInfo);
 },
get_distance: function(req, res){
distance.find(req, res, function(err,dist){
     if(err) res.send(err); res.json(dist);
  });
 },
};
module.exports = controllers;
```

- Our code has two distinct parts, about and get_distance. The first functionality receives response objects and requests. The package.json file returns a new object as a response object for running the about functionality.
- The following functionality, get-distance, is built to maintain coordination between the distance module and find function. It accepts error & distance objects and returns the response object if an error is received.

Step 4 : Building the External API Call

We are setting an expired test key as a default key in our project for making external calls.

```
var request = require('request');
constapiKey = process.env.ZIPCODE_API_KEY ||
"hkCt1nW1wF1rppaEmoor7T9G4ta7R5wFSu8l1dokNz8y53gGZHDneWWVosbEYirC";
constzipCodeURL = 'https://www.zipcodeapi.com/rest/';
var distance = {
 find: function(req, res, next) {
```

```
    request(zipCodeURL + apiKey
        + '/distance.json/' + req.params.zipcode1 + '/'
        + req.params.zipcode2 + '/mile',
    function (error, response, body) {
      if (!error &&response.statusCode == 200) {
        response = JSON.parse(body);
res.send(response);
      } else {
     console.log(response.statusCode + response.body);
res.send({distance: -1});
      }
    });
  }
};
module.exports = distance;
```

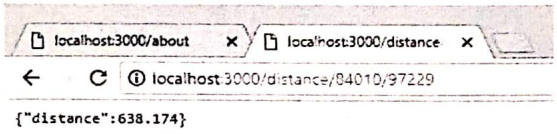
Step 5 : Execution

Finally, your application is ready to run. Open a console window and run the following command:

npm start

Now if you add two parameters, the two zip codes, you will see something like this:

http://localhost:3000/distance/84010/97229



Q. 7 What are the key characteristics of PM2 ? **(3 Marks)**

Ans. :

Key Characteristics of PM2

1. When a process goes down, PM2 will automatically restart the service and make it Live. We can enable the process to automatically start on system boot up or while restart.
2. It supports app monitoring, efficient management of micro-services/processes, running apps in cluster mode, graceful start and shutdown of apps.
3. With a configuration file (process.json), you specify what processes you want to run and how many you'd like to scale to. When starting PM2, you specify the process.json file and PM2 takes care of the rest.
4. PM2 allows you to keep your Node.js applications alive forever, and to reload them with zero downtime when you have updates to your application or server.
5. PM2 has built-in log management. It aggregates log data from all of your applications and writes it to a single source for viewing. You can even tail the logs in real-time to see what's going on behind the scenes with your application.

Q. 8 What do you mean by Express JS and what is its use? (5 Marks)

Ans. :

ExpressJS

- Express.js is a Node js web application server framework, which is specifically designed for building single-page, multi-page, and hybrid web applications.
- Express.js (Express) is a light web framework which sits on top of Node.js and it adds functionality like (middleware, routing, etc.) and simplicity to Node.js.
- Express is the backend part of something known as the MEAN stack. The MEAN is a free and open-source JavaScript software stack for building dynamic web sites and web applications which has the following components:
- MongoDB : The standard NoSQL database
- Express.js : The default web applications framework
- Angular.js : The JavaScript MVC framework used for web applications
- Node.js : Framework used for scalable server-side and networking applications.
- Express is a flexible Node.js web application framework that provides a wide set of features to develop both web and mobile applications. Express.js makes it much easier and simpler to build a web server with the use of middleware, which can handle requests and responses.
- Express.js is a web application framework that is built on top of Node.js. It provides a minimal interface with all the tools required to build a web application.

Features of Express.js

The main features of Express.js include:

- It can be used to design single-page, multi-page and hybrid web applications.
- It allows to setup middlewares to respond to HTTP Requests.
- It defines a routing table which is used to perform different actions based on HTTP method and URL.
- It allows to dynamically render HTML Pages based on passing arguments to templates.

Q. 9 Write a short note on : Template Engines. (5 Marks)

Ans. :

Template Engines

- Websites are built with HTML, you can dynamically generate HTML pages using Express. Dynamically generated HTML pages are useful when you want to show real time data or change a page's details based on the user.
- A template engine facilitates you to use static template files in your applications. At runtime, it replaces variables in a template file with actual values, and transforms the template into an HTML file sent to the client.
- There's a wide variety of template engines that work with Express. The default template engine found in Express is Jade, which is now known as Pug.
- However, the default Jade installed in Express is still using the old version.
- Pug is a very powerful templating engine which has a variety of features including filters, includes, inheritance, interpolation, etc.
- To install Pug and use it in our Express app, we have to use npm to install it first :

```
$ npm install pug
```

- Then you can set the view engine to pug when you initialize your app, then make a get call and render your view:

```
const express =require('express');
const app =express();
app.set('view engine','pug');

app.get('/home',(req, res)=>{
res.render('home');
});
```

Q. 10 How to serve static files in ExpressJS. **(5 Marks)**

Ans. : Serving Static Files

- Static files are files that clients download as they are from the server. Create a new directory, public. Express, by default does not allow you to serve static files. To serve static files such as images, CSS files, and JavaScript files, use the express.static built-in middleware function in Express.
- The function signature is :

```
express.static(root,[options])
```

- The root argument specifies the root directory from which to serve static assets.
- Express provides a built-in method to serve your static files :

```
app.use(express.static('public'));
```

- When you call app.use(), you're telling Express to use a piece of middleware. Middleware is a function that Express passes requests through before sending them to your routing functions, such as your app.get('/') route. express.static() finds and returns the static files requested. The argument you pass into express.static() is the name of the directory you want Express to serve files. Here, the public directory.
- In index.js, serve your static files below your PORT variable. Pass in your public directory as the argument :

index.js

```
const express =require('express');
const app =express();
constPORT=3000;
app.use(express.static('public'));
app.get('/',(req, res)=>{
res.send('Hello World!');
});
app.listen(PORT,()=> console.log(`Server listening on port: ${PORT}`));
```

- Navigate to your index.html file in the public directory. Populate the file with body and image elements

```
<html>
<head>
<title>Hello World!</title>
</head>
<body>
<h1>Hello, World!</h1>
```

```
<imgsrc="tuls.png"alt="tuls">
</body>
</html>
```

Q. 11 Explain basic HTTP Authentication with example. **(6 Marks)**

Ans. :

Applying basic HTTP Authentication

- Authentication is the process of verifying the identity of the user. HTTP Basic Authentication is a mechanism in which the server challenges anyone requesting for information and get a response in the form of a username and password. The information the server receives is encoded with base-64 and passed into the Authorization header.
- When the client makes a request to a resource on the server that required authorization, the server sends a response with a 401 status code accompanied with a WWW-Authenticate Basic header. Most browsers handle this response by requesting a username and a password from the user.
- When the web client obtains the username and password, it sends a response back to the server with ah Authorization header in the form of Authorization: Basic username: password.
- The username and the password provided by the client is only encrypted with base-64. This approach of authenticating users is not recommended since information exchanged between both parties could be intercepted when the connection between the two is not secured. The HTTP Basic authentication is only secure when the connection between the client and the server is secure.
- Express.js framework is mainly used in Node.js application because of its help in handling and routing different types of requests and responses made by the client using different Middleware.

```
const express = require("express");
const fs = require("fs");
varpath = require('path');
const app = express();
functionauthentication(req, res, next) {
  varauthheader = req.headers.authorization;
  console.log(req.headers);
  if(!authheader) {
    varerr = newError('You are not authenticated!');
    res.setHeader('WWW-Authenticate', 'Basic');
    err.status = 401;
    returnnext(err)
  }
  varauth = newBuffer.from(authheader.split(' ')[1],
  'base64').toString().split(':');
  varuser = auth[0];
  varpass = auth[1];
  if(user == 'admin'&& pass == 'password') {
    next();
```

```
    } else{
        varerr = newError('You are not authenticated!');
        res.setHeader('WWW-Authenticate', 'Basic');
        err.status = 401;
        returnnext(err);
    }
}
// First step is the authentication of the client
app.use(authentication)
app.use(express.static(path.join(__dirname, 'public')));
 // Server setup
app.listen((3000), () => {
   console.log("Server is Running ");
})
```

- Run index.js using the following command :

  node index.js.
- Open any browser with http://localhost:3000. A pop will occur near the address bar. Fill in the username and password that are mention in the code.

Q. 12 How to implement session authentication in express -js with example? **(5 Marks)**

Ans. : Implement Session Authentication

- Sessions are the traditional method and used by many applications. It is a straightforward and secure way to manage the user data your application needs for page loads. The express-session project is a handy library for adding session handling into your ExpressJS applications. It is the standard used by most organizations today.
- Express-session library will help us setup a Node.js session. Express-session an HTTP server-side framework used to create and manage a session middleware.

(a) Add the Express-session options

```
app.use(session({
   secret: '2C44-4D44-WppQ38S',
   resave: true,
   saveUninitialized: true
}));
```

1. **secret** : A random unique string key used to authenticate a session. It is stored in an environment variable and can't be exposed to the public. The key is usually long and randomly generated in a production environment.
2. **resave** : Takes a Boolean value. It enables the session to be stored back to the session store, even if the session was never modified during the request. The default value is true.
3. **saveUninitialized** : This allows any uninitialized session to be sent to the store. When a session is created but not modified, it is referred to as uninitialized.

(b) Session-based authentication flow:

- The user submits the login form and sends credentials (username and password) to the server,
- The server checks credentials in the database,
- If credentials are good, the server creates the session and store it in the database,
- The server sends the cookie with session ID back to the user,
- The user sends the cookie with each request,
- Server validate session ID against session in the database and grants access,
- When the user logout, the server destroys the session.

(c) Example

- There is a login endpoint, a logout endpoint and get post page. To see the post page, you have to login first, and your identity will be verified and saved in session. When you hit the logout endpoint, it will revoke your access by removing your identity from the session.
- If the user is tuls and if he has the admin access. A real web app will get the user and user access level from session, and then check against the user and user access lever from a database on the server.

```
varexpress = require('express'),
  app = express(),
  session = require('express-session');
app.use(session({
  secret: '2C44-4D44-WppQ38S',
  resave: true,
  saveUninitialized: true
}));
// Authentication and Authorization Middleware
varauth = function(req, res, next) {
 if(req.session&&req.session.user === "tuls"&&req.session.admin)
  returnnext();
 else
  returnres.sendStatus(401);
};
// Login endpoint
app.get('/login', function(req, res) {
 if(!req.query.username || !req.query.password) {
  res.send('login failed');
 } elseif(req.query.username === "tuls"|| req.query.password === "abc123") {
  req.session.user = "tuls";
  req.session.admin = true;
  res.send("login success!");
 }
```

```
});
// Logout endpoint
app.get('/logout', function(req, res) {
  req.session.destroy();
  res.send("logout success!");
});
// Get content endpoint
app.get('/content', auth, function(req, res) {
   res.send("You can only see this after you've logged in.");
});
app.listen(3000);
console.log("app running at http://localhost:3000");
```

- When we run the program and hit *localhost:3000/login?username=tuls&password=abc123*, the login url to check log the user in by saving the user and user access level in a session.
- The session will be different for each user, and also be unique for the same user using different browsers.
- For example, if the same user logged in using Chrome, and the open up Firefox, the user will have to login again in FireFox in order to gain protected resources.
- For demonstration purpose, this is a get request and passing in the info through query parameters. A real web app will usually be using a post request and passing in the data in the post form.

Q. 13 What is NoSQL ? Write difference between SQL and NOSQL. **(6 Marks)**

Ans. :

NoSQL

- NoSQL database stands for "Not Only SQL" or "Not SQL.". NoSQL database system encompasses a wide range of database technologies that can store structured, semi-structured, unstructured and polymorphic data.
- They allow you to store data in a flexible and fluid data model, provide scaling up capabilities to your database tier, and provide 100% uptime through replication.
- When the application grows relational databases start facing the scalability issue. If we talk about the big gigantic website (such as Facebook, Google, Amazon) that throws billions or trillions of queries within a small amount of time then relational databases get failed in handling the queries.
- To get rid of this limitation in relational databases NoSQL comes in the picture that mainly focuses on two things, high operations speed and flexibility in storing the data. These two are the main common things that gave birth to the NoSQL database.
- Vertical scaling means adding more resources to the existing machine whereas horizontal scaling means adding more machines to handle the data.
- NoSQL databases are horizontally scalable, which means that they can handle increased traffic simply by adding more servers to the database.
- NoSQL databases have the ability to become larger and much more powerful, making them the preferred choice for large or constantly evolving data sets.

Difference between SQL vs NoSQL

Sr. No.	SQL	NoSQL
1	Relational Database Management System (RDBMS)	Non-relational or distributed database system.
2	These databases have fixed or static or predefined schema	They have dynamic schema
3	These databases are not suited for hierarchical data storage.	These databases are best suited for hierarchical data storage.
4	These databases are best suited for complex queries	These databases are not so good for complex queries
5	Vertically Scalable	Horizontally scalable
6	Follows ACID property	Follows CAP(consistency, availability, partition tolerance)

2. 14 What are the different types of NoSQL Databases? Explain with suitable examples. (5 Marks)

Ans. : Types of NoSQL Databases

NoSQL databases fall into four main categories :

Key-value Pair Based

Column-oriented Graph

Graphs based

Document-oriented

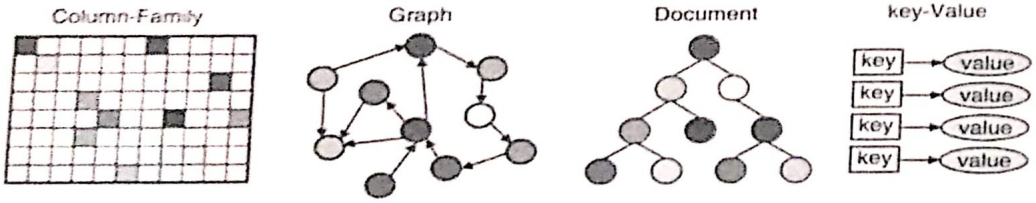


Fig. 4.1

Key-value Pair Based

- Data is stored in key/value pairs. It is designed in such a way to handle lots of data and heavy load. Key-value pair storage databases store data as a hash table where each key is unique, and the value can be a JSON, BLOB(Binary Large Objects), string, etc. This kind of NoSQL database is used as a collection, dictionaries, associative arrays, etc.
- Key value stores help the developer to store schema-less data. Key-value data stores are ideal for storing session information, user profiles, comments, and recommendations of products.
- **Examples**: Redis, Memcached. Pinterest uses Redis to store lists of images, galleries, users, boards, followers, and un-followers

Document-Oriented

Document-Oriented NoSQL DB stores and retrieves data as a key value pair but the value part is stored as a document. The document is stored in JSON or XML formats.

- The document type is mostly used for CMS systems, blogging platforms, real-time analytics & e-commerce applications.
- **Example** : Amazon SimpleDB, CouchDB, MongoDB, Riak, Lotus Notes, MongoDB. Cisco uses Couchbase to achieve greater scalability

C. Column-based

- Column-oriented databases work on columns and are based on BigTable paper by Google. Every column is treated separately. Values of single column databases are stored contiguously.
- They deliver high performance on aggregation queries like SUM, COUNT, AVG, MIN etc. as the data is readily available in a column.
- Column-oriented databases are designed like arrays with three-dimensional, the row identifier is the first dimension, a combination of a column identifier with family is the second and the timestamp is third. Column-based NoSQL databases are widely used to manage data warehouses, business intelligence, CRM.
- **Example** :HBase, Cassandra, HBase, Hypertable . Facebook uses HBase for various services like search indexing and the "Nearby Friends" feature.

D. Graph-Based

- A graph type database stores entities as well the relations amongst those entities. The entity is stored as a node with the relationship as edges.
- An edge gives a relationship between nodes. Every node and edge has a unique identifier. The graph can be traversed in a depth-first or breadth-first fashion. Starting from any arbitrary or start node.
- **Example** : FlockDB, Neo4j and HyperGraphDB. Walmart uses Neo4j to provide relevant product promotions and recommendations.

Q. 15 What a short note on CAP theorem. **(5 Marks)**

Ans. : CAP theorem

The three aspects of the CAP theorem are consistency, Availability, and Partition tolerance.

1. **Consistency** : Every read should give the most recent write. E.g. Consistency in SQL ACID property. Every node must be updated before allowing any further reads.
2. **Availability** : Every node executes the query if not failed. Availability is achieved by replicating the data across different machines. For example, HDFS (i.e.Hadoop Distributed File System).
3. **Partition-tolerance** : Tolerance to network partition. For example, two nodes can't communicate with each other, but their clients can talk to either one or both of the nodes.

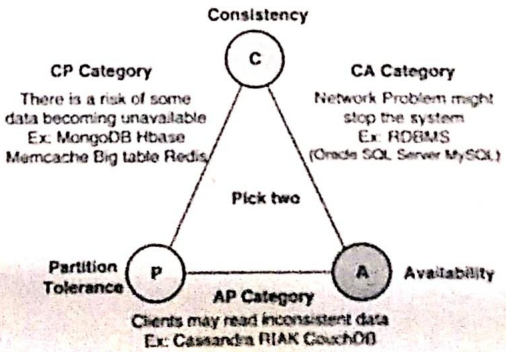


Fig. 4.2

- CAP theorem asserts that any networked shared-data system can have only two of three desirable properties. That is "C & A", "A & P", or "C & P".
- Consistency & Availability holds true in traditional SQL databases like MySQL, PostgreSQL, etc.
- Availability & Partition-tolerance holds true in NoSQL databases like Cassandra, CouchDB, etc.
- Consistency & Partition-tolerance holds true for HBase, MongoDB, Redis, MemcacheDB, etc.

Q. 16 What is MonogoDB? What are the features and components of MongoDB Architecture? **(6 Marks)**

Ans. :

MongoDB Basics

- MongoDB is a document-oriented NoSQL database used for high volume data storage. MongoDB makes use of collections and documents.
- Documents consist of key-value pairs which are the basic unit of data in MongoDB. Collections contain sets of documents and function which is the equivalent of relational database tables.
- MongoDB stores data in BSON documents. BSON is a binary representation of JSON (JavaScript Object Notation) documents.

A. Features of MongoDB

- **Indexing** : It makes use of indexes that helps in improving the search performance.
- **Replication** : MongoDB distributes the data across different machines.
- **Ad-hoc Queries** : It supports ad-hoc queries by indexing the BSON documents & using a unique query language.
- **Schemaless** : It enhances the flexibility of the data and needs no script to modify or update data.
- **Sharding** : It makes use of sharding which eases the deployment of very large data sets and provides high throughput operations.

B. Components of MongoDB Architecture

Few of the common terms used in MongoDB are as follows :

- **_id** : This is a field required in every MongoDB document. The _id field represents a unique value in the MongoDB document. The _id field is like the document's primary key. If you create a new document without an _id field, MongoDB will automatically create the field.
- **Collection** : This is a grouping of MongoDB documents. A collection is the equivalent of a table which is created in any other RDMS such as Oracle or MS SQL. A collection exists within a single database.
- **Cursor** : This is a pointer to the result set of a query. Clients can iterate through a cursor to retrieve results.
- **Database** : This is a container for collections like in RDMS wherein it is a container for tables. Each database gets its own set of files on the file system. A MongoDB server can store multiple databases.
- **Document** : A record in a MongoDB collection is basically called a document. The document, in turn, will consist of field name and values.
- **Field** : A name-value pair in a document. A document has zero or more fields. Fields are analogous to columns in relational databases.
- **JSON** :This is a human-readable, plain text format for expressing structured data.

Q. 17 What is the difference between MongoDB and RDBMS? **(4 Marks)**

Ans. : Difference between MongoDB & RDBMS

RDBMS	MongoDB	Description
Table	Collection	In RDBMS, the table contains the columns and rows which are used to store the data whereas, in MongoDB, this same structure is known as a collection. The collection contains documents which in turn contains Fields, which in turn are key-value pairs.
Row	Document	In RDBMS, the row represents a single, implicitly structured data item in a table. In MongoDB, the data is stored in documents.
Column	Field	In RDBMS, the column denotes a set of data values. These in MongoDB are known as Fields.
Joins	Embedded documents	In RDBMS, data is sometimes spread across various tables and in order to show a complete view of all data, a join is sometimes formed across tables to get the data. In MongoDB, the data is normally stored in a single collection, but separated by using Embedded documents. So there is no concept of joins in MongoDB.

Q. 18 Explain Mongoose ODM for middleware. **(4 Marks)**

Ans. : Mongoose ODM for Middleware

- Mongoose or MongooseJS is a MongoDB object modeling(ODM) tool designed to work in an asynchronous environment. Basically, it is a package that we will use to interact(query, update, manipulate) with our MongoDB database in our nodeJS app.
- Mongoose is a layer of abstraction over the regular MongoDB driver. We use a regular MongoDB driver to access our database, and it would work just fine, but instead we use Mongoose, because it gives us a lot more functionality out of the box, allowing for faster and simpler development of our applications.
- Some of the features Mongoose give us is schemas to model our data and relationship, easy data validation, a simple query API, middleware, and much more. A Mongoose model is a wrapper on the Mongoose schema.
- A Mongoose schema defines the structure of the document, default values, validators, etc., whereas a Mongoose model provides an interface to the database for creating, querying, updating, deleting records, etc.
- We then take that schema and create a model out of it. And the model is basically a wrapper around the schema, which allows us to actually interface with the database in order to create, delete, update, and read documents and create a simple schema and model.

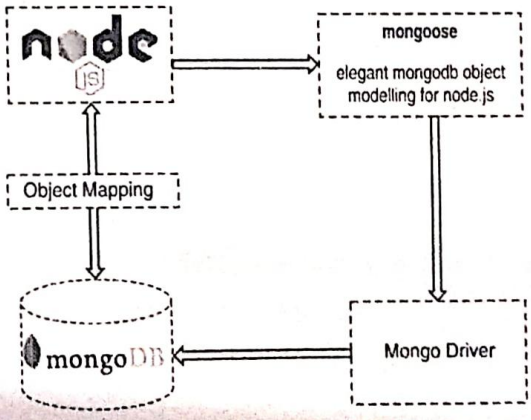


Fig. 4.3

Unit V : Mobile Web Development

Q. 1 What is mobile web? Explain benefits of the mobile web. **(5 Marks)**

Ans. : Mobile Web

- The mobile web refers to browser-based World Wide Web services accessed from handheld mobile devices, such as smartphones or feature phones, through a mobile or other wireless network.
- The shift to mobile Web access has accelerated since 2007 with the rise of larger multitouch smartphones, and since 2010 with the rise of multitouch tablet computers. Both platforms provide better Internet access, screens, and mobile browsers, or application-based user Web experiences than previous generations of mobile devices.
- The Mobile Web Initiative (MWI) was set up by the W3C to develop the best practices and technologies relevant to the mobile web. The goal of the initiative is to make browsing the web from mobile devices more reliable and accessible.
- The main aim is to evolve standards of data formats from Internet providers that are tailored to the specifications of particular mobile devices.

Benefits of the Mobile Web

1. **Constant Connectivity** : Web-enabled mobile devices provide owners with around-the-clock access to the Internet, regardless of location.
2. **Location-Aware** : Many of today's smartphones and pocket PCs have global positioning system (GPS) capabilities, which make them "aware" of where they are at all times.

   Cell phone users can search for businesses near their locations, retrieve directions to a desired destination, create location-based content, and discover nearby friends and contacts.
3. **Interactive Capabilities** : The mobile Web offers users the participatory experience of the read/write Web in the palm of their hand. Users can create content, such as photos and videos taken with their camera phones, share and rate media, make comments, write blog posts, tag resources, and form connections on social networks.

Q. 2 Explain different mobile devices. **(6 Marks)**

Ans. : Mobile Devices

- Wi-Fi or cellular access to the internet or a Bluetooth connection to another device.
- A battery that powers the device for several hours.
- A physical or on-screen keyboard for entering information.
- The size and weight allow it to be carried in one hand and manipulated with the other hand.
- Touch screen interface in almost all cases.
- A virtual assistant, such as Siri, Cortana, or Google Assistant.
- The ability to download data, such as apps or books, from the internet or another device. Some of the most common mobile devices we use today include:

1. Smartphones
2. Tablets computers
3. Wearables
4. E-readers
5. Handheld gaming consoles

1. **Smartphone**
   - Smartphone is a more powerful version of a traditional cell phone. In addition to the same basic features : phone calls, voicemail, text messaging. Smartphones can connect to the Internet over Wi-Fi or a cellular network.
   - This means you can use a Smartphone for the same things you would normally do on a computer, such as checking your email, browsing the Web, or shopping online. For many people, a Smartphone can actually replace electronics like an old laptop, digital music player, and digital camera in the same device.
2. **Tablet computers**
   - Tablet computers are designed to be portable. However, they provide a different computing experience.
   - The most obvious difference is that tablet computers don't have keyboards or touchpads. Instead, the entire screen is touch-sensitive, allowing you to type on a virtual keyboard and use your finger as a mouse pointer.
   - A tablet really shines when performing tasks that would be more suitable for a larger display with more battery life. Some examples include work presentations, heavy gaming, or even live streaming of shows in high definition on Netflix.
3. **Wearables**
   - Smart watches and fitness trackers are among the newest additions to the mobile device landscape.
   - The main benefits a smart watch can provide us with are the ability to get notifications and necessary information on our wrist without having to pick up another device.
   - With some of the new standalone models, we can take and receive phone calls just like our smartphones can.
4. **E-readers**
   - E-readers are similar to tablet computers, except they are mainly designed for reading e-books (digital, downloadable books).
   - Notable examples include the Amazon Kindle, Barnes & Noble Nook, and Kobo.
   - Most e-readers use an e-ink display, which is easier to read than a traditional computer display. These devices changed the whole concept of reading and helped to bring books into a new era.
5. **Handheld gaming**

   Handheld gaming consoles changed how we think about mobile gaming. Some of the most well-known mobile gaming devices we have today include the Nintendo Switch and the Nintendo 3DS. Gaming is now one of the most popular forms of entertainment, and there will always be a need for such devices.

Q. 3 List different features of jQuery mobile. **(4 Marks)**

Ans. :

Features of jQuery Mobile

- **Great simplicity and usability** : The jQuery Mobile framework is uncomplicated and flexible.
- Since the framework's configuration interface is mark-up driven, developers can easily build their complete basic application interfaces in HTML, with minimal or no JavaScript code.
- **Support for user friendly inputs** : Developers can include an uncomplicated API to support touch, mouse, and cursor focus-based user input functionalities
- **Progressive enhancement :** Brings a unique functionality to all mobile, tablet, and desktop platforms and adds efficient page loads and wider device support.

- **Accessibility :** Accessibility features such as WAI-ARIA are tightly integrated throughout the framework to provide support for screen readers and other assistive technologies.
- **Theming and UI widgets :** Provides a set of touch-friendly form inputs and UI widgets.
- **Accessibility :** The framework comes with support for Accessible Rich Internet Applications (WAI-ARIA) to assist disabled persons using screen readers and other assistive technologies easily access web pages.
- **Responsiveness :** It enables the same underlying codebases to fit comfortably in different types of screens, from mobile devices to desktop-sized screens.
- **Ajax Navigation :** It enables animated page transitions while maintaining back button, bookmarking and and clean URLs though push State.

Q. 4 What are the advantages and disadvantages of jQuery mobile ? **(4 Marks)**

Ans. :

Advantages of jQuery Mobile

- jQuery Mobile uses HTML5 along with JavaScript for ease of development.
- It takes care of cross-platform and cross-device (iPhone, Android, Windows, Blackberry, and other platforms) issues so you don't have to worry about writing different code for each device or resolution.
- This is a huge advantage that results in a unique, clean code that saves time and resources.
- It is built in a way that allows the same code to automatically scale from the mobile screen to desktop screen.
- You can create your own theme using the ThemeRoller. It is a useful tool for creating themes without writing any line of code.

Disadvantages of jQuery Mobile

- jQuery Mobile works fine for simple designs where you can use a default theme or create your own theme using the ThemeRoller. However, when you have something more custom, like our client's design, there are a lot of things that have to be overwritten to obtain the results that you need.
- Applications which are developed using jQuery Mobile are slower on mobiles.
- It becomes more time consuming when you combine jQuery mobile with other mobile frameworks.

Q. 5 Explain Mobile page structure. **(5 Marks)**

Ans. :

Mobile page structure

- A jQuery Mobile site must start with an HTML5 doctype to take full advantage of all of the framework's features. In the head, references to jQuery, jQuery Mobile and the mobile theme CSS are all required to start things off.
- The easiest way to get started is to link to files hosted on the jQuery CDN or for best performance, build a custom bundle.

Example

```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="stylesheet" href="https://code.jquery.com/mobile/[version]/jquery.mobile-[version].min.css" />
```

```
<script src="https://code.jquery.com/jquery-[version].min.js"></script>
<script src="https://code.jquery.com/mobile/[version]/jquery.mobile-[version].min.js"></script>
</head>

<body>
<div data-role="page">

 <div data-role="header">
   <h1>Welcome To Siya'world</h1>
 </div>

 <div data-role="main" class="ui-content">
   <p>You are in pune Now!!</p>
 </div>

 <div data-role="footer">
   <h1>Footer Text</h1>
 </div>

</div>
</body>
</html>
```

- In above there is a meta viewport tag in the head to specify how the browser should display the page zoom level and dimensions.
- By setting the viewport attributes to content="width=device-width, initial-scale=1", the width will be set to the pixel width of the device screen.<meta name="viewport" content="width=device-width, initial-scale=1">
- Inside the <body> tag, each view or "page" on the mobile device is identified with an element (usually a div) with the data-role="page" attribute
- The data-role="page" is the page displayed in the browser
- The data-role="header" creates a toolbar at the top of the page
- The data-role="main" defines the content of the page, like text, images, buttons, forms, etc.
- The "ui-content" class adds extra padding and margin inside the page content
- The data-role="footer" creates a toolbar at the bottom of the page.

Q. 6 Explain different icons in Jquery mobile. **(5 Marks)**

Ans. : Icons

In jQuery Mobile a group of built-in icons can be applied to buttons, collapsibles, listview buttons and more. There is an SVG and PNG image of each icon. By default the SVG icons, that look great on both SD and HD screens, are used.

1. **Icon set :** It sets the icon in the button. You can set the icon in the button using ***ui-icon*** class and ***ui-btn-icon-pos_name*** class for specifying the position for the icon.

Example

```
<button class="ui-btn ui-shadow ui-corner-all ui-btn-icon-left ui-icon-action">action</button>
<button class="ui-btn ui-shadow ui-corner-all ui-btn-icon-left ui-icon-arrow-d-l">arrow-d-l</button>
<button class="ui-btn ui-shadow ui-corner-all ui-btn-icon-left ui-icon-arrow-d-r">arrow-d-r</button>
```

2. **Custom Icons**

   Icons are displayed as background image of :after pseudo elements. Target the pseudo element to button class="ui-btn ui-shadow ui-corner-all ui-btn-icon-left ui-icon-myicon">myicon</button>

3. **Icon Positioning**

   It specifies the position of the icon in the button. You can determine the position of the icon (top, right, left, bottom in the button) using the ***ui-btn-icon-[value]*** class.

```
<a href="index.html" class="ui-btn ui-shadow ui-corner-all ui-icon-arrow-u ui-btn-icon-top">Top</a>
<a href="index.html" class="ui-btn ui-shadow ui-corner-all ui-icon-arrow-d ui-btn-icon-bottom">Bottom</a>
```

4. **Icon Only**

   It displays only an icon in the button. Use "notext" as value for icon position if you want to create an icon-only button.

```
<a href="index.html" class="ui-btn ui-shadow ui-corner-all ui-icon-delete ui-btn-icon-notext">Delete</a>
```

5. **Icon shadow :** It adds an icon shadow in your button . You can add an icon shadow in your button using the ***ui-shadow-icon*** class.

```
<a href="index.html" class="ui-shadow-icon ui-btn ui-shadow ui-corner-all ui-icon-delete ui-btn-icon-left">Icon shadow</a>
<a href="index.html" class="ui-shadow-icon ui-btn ui-shadow ui-corner-all ui-btn-b ui-icon-delete ui-btn-icon-left">Icon shadow</a>
```

Q. 7 What is transitions in Jquery mobile ? **(5 Marks)**

Ans. : Transitions

- jQuery Mobile includes CSS3 effects that lets you choose the way a page should open. jQuery Mobile has a variety of effects for how to transition from one page to the next.
- Need to set the value of data-transition attribute to any of the available CSS3 effects.
- Change the direction in which the effect occurs by setting the value of data-direction attribute to reverse. The transition effect can be applied to any link or form submission by using the data-transition attribute :

```
<a href="#anylink" data-transition="slide">Slide to Page Two</a>
```

Table 5.1

Transition	Description
fade	Default. Fades to the next page
flip	Flips to the next page from back to front
flow	Throws the current page away and comes in with the next page
pop	Goes to the next page like a popup window
slide	Slides to the next page from right to left

Transition	Description
slidefade	Slides from right to left and fades in the next page
slideup	Slides to the next page from bottom to top
slidedown	Slides to the next page from top to bottom
Turn	Turns to the next page
none	No transition effect

Q. 8 Explain types of widgets. **(5 Marks)**

Ans. :

Types of Widgets

- **Buttons :** It specifies clickable button that includes content like text or images.
- **Checkbox :** Checkboxes are used when more than one option is required to be selected.
- **Radiobox :** Radio buttons are used when out of many options, just one option is required to be selected.
- **Datepicker :** It is focused on the input to open an interactive calendar in a small overlay.
- **Collapsible :** Collapsible allows you to expand or collapse the content whenever clicked on it. It is very helpful for mobile device, which presents a brief content.
- **Listview :** The purpose of listview component is to render complex and customized content in lists.
- **Navbar :** The navbar widget is a set of buttons which links you to other web pages or sections.
- **Popups :** Popup is a user interface that appears within a small window to display text, images, and other content.
- **Table :** jQuery Mobile uses the table to represent the data in terms of rows and columns, i.e. displays the data in a tabular format.
- **Rangeslider :** Rangeslider widget provides you with a pair of handles allowing you to select a numeric value range.
- **Selectmenu :** A select menu provides various options in the form of dropdown list, from where a user can select one or more options.
- **Slider :** Slider allows you to choose a value by sliding the handle of the slider.
- **Tabs :** The tabs widget is jQuery ui tabs widget's extension, which accepts all the methods and options.
- **Toolbar :** The jQuery mobile toolbar widget allows you to create headers and footers.
- **Textinput :** The <input> tag is used to declare an input element, a control that allows the user to input data.

Q. 9 Explain events for the mobile devices. **(5 Marks)**

Ans. :

Events

- You can use all standard jQuery events in jQuery Mobile. jQuery Mobile also offers several events that are tailor-made for mobile-browsing.

Syntax for jQuery Mobile pagecreate event.

```
<script>
$(document).on("pagecreate","#pageone",function(){
```

```
// jQuery events go here...
});
</script>
```

- In jQuery Mobile, we use the pagecreate event, which occurs when the page has been been created in the DOM, but before enhancement is complete. The second parameter ("#pageone") points to the id of the page to specify the event(s) for.
- Following are events for the mobile devices, which are supported by jQuery Mobile.

1. Touch Events

- The touch events are triggered when the user taps on an element or swipes over an element.
- The tap event is triggered after a quick touch event and the taphold event is triggered after a sustained touch event.
- The swipe event is triggered when a horizontal drag of 30px or more or a vertical drag of less than 75px occurs within one second.
- The swipeleft event is triggered when a swipe occurs in the left direction and a swiperight event is triggered when a swipe occurs.

2. Scroll Events

- Triggers when a user scrolls up and down.
- The scrollstart event is triggered when the user starts to scroll the page and the scrollstop event is triggered when the user stops scrolling.

3. Orientation Events

- The orientation change event is triggered when the user changes the orientation from portrait to landscape or vice versa.
- Portrait orientation is when the device is held in vertical position and landscape orientation is when the device is held in a horizontal position.
- Using the orientation property of the window object, you can check whether the device is in landscape or portrait mode.
- The value of orientation property 0 means that the device is in portrait mode and the value is 90 or -90 means the device is in landscape mode.

4. Page Events

- Page events is triggered when a page is shown, hidden, created, loaded and/or unloaded.
- During the page initialization, pagebeforecreate event gets triggered when the page is about to be initialized and pagecreate event gets triggered when the page is created.
- When an external page is loaded, pagecontainerbeforeload event gets triggered before the page is loaded, pagecontainerload event gets triggered after the page is loaded successfully and pagecontainerloadfailed event gets triggered if the page is not loaded successfully.
- During the transition from one page to another, pagebeforehide and pagehide events get triggered on the from page.

Q. 10 Explain form controls in Jquery Mobile. **(5 Marks)**

Ans. : Forms

- Mobile forms are the same thing as forms but they fit in well to a mobile device display (aspect ratio). jQuery mobile forms are responsive forms that are styled and made dynamic with jQuery libraries.
- In jQuery Mobile you can use the following form controls :
  - Text Inputs
  - Search Inputs
  - Radio Buttons
  - Checkboxes
  - Select Menus
  - Sliders
  - Flip Toggle Switches

1. Form Structure

- The <form> element must have a method and an action attribute
- Each form element must have a unique "id" attribute. The id must be unique across the pages in the site. This is because jQuery Mobile's single-page navigation model allows many different "pages" to be present at the same time
- Each form element must have a label. Set the for attribute of the label to match the id of the element

```
<form method="post" action="demoform.asp">
 <label for="fname">First name:</label>
 <input type="text" name="fname" id="fname">
</form>
```

2. jQuery Mobile Form Select

Select menu creates a simple menu with select options to choose from. The **<select></select>** elements are used to define a selection list and the **<option></option>** tags are used to define an item in a selection list.

Example :

```
<fieldset class = "ui-field-contain">
<label for = "fname">Select City</label>
<select id = "select" name = "fname">
<option value = "Pune">Pune</option>
<option value = "Mumbai">Mumbai</option>
<option value = "Chennai">Chennai</option>
<option value = "Bangalore">Bangalore</option>
</select>
</fieldset>
```

3. jQuery Mobile Slider Controls

- A slider allows you to select a value from a range of numbers. To create a slider, use <input type="range">.
- Use the following attributes to specify restrictions:
  - max : Specifies the maximum value allowed
  - min : Specifies the minimum value allowed
  - step : Specifies the legal number intervals
  - value : Specifies the default value

Example

```
<form method="post" action="demoform.asp">
<label for="marks">Percentage of Marks :</label>
<input type="range" name="mark" id="marks" value="35" min="0" max="100">
<input type="submit" data-inline="true" value="Submit">
</form>
```

Q. 11 Write a short note on : Theme classes. **(5 Marks)**

Ans. : Theme Classes

- Assigning color swatches through the data-theme attribute is one way to leverage the theme system, but it's also possible to apply any of the theme swatches directly to your markup through classes to apply the colors, textures and font formatting of your theme to any markup.
- This is especially useful when creating your own custom layout elements or UI widgets. Here are a few common theme classes
  - **ui-bar-(a-z) :** Applies the toolbar theme styles for the selected swatch letter. Commonly used in conjunction with ui-bar structural class to add the standard bar padding styles.
  - **ui-body-(a-z) :** Applies the content body theme styles for the selected swatch letter. Commonly used in conjunction with ui-body structural class to add the standard content block padding styles.
  - **ui-btn-up-(a-z) :** Applies the button/clickable element theme styles for the selected swatch letter. Commonly used in with the ui-btn-hover-(a-z) and ui-btn-down-(a-z) interaction class states to provide visual feedback and ui-btn-active to indicate the selected or "on" state.
  - **ui-shadow :** Applies the theme's global drop shadow to any element using CSS box-shadow property.

Q. 12 Explain formatting lists with example. **(8 Marks)**

Ans. :

Formatting Lists

- Create mobile optimized lists in jQuery Mobile using standard HTML lists and then assigning the value listview to data-role attribute of the <ol> or <ul> element.
- You can also style your lists, group the list items with dividers, add icons, thumbnails and count bubbles to the items and even make the list filterable.

1. Creating Lists

- You need to assign the value listview to the data-role attribute of the <ul> or <ol> element to make a list mobile optimized with jQuery Mobile.
- You need to add links inside list items to make them clickable. When you add links inside a list item, the item automatically becomes a button without assigning the value button to the data-role.

Example

```
<!DOCTYPE html>
<html>
<head>
<title>list</title>
<link rel="stylesheet" href="http://code.jquery.com/mobile/1.3.2/jquery.mobile-1.3.2.min.css" />
```

```
<script src="http://code.jquery.com/jquery-1.9.1.min.js"></script>
<script src="http://code.jquery.com/mobile/1.3.2/jquery.mobile-1.3.2.min.js"></script>
</head>
<body>
<div data-role="page" id="lists">
<div data-role="content">
<h3>Names</h3>
<ul data-role="listview">
<li>Siya</li>
<li>Arnav</li>
<li><a href="#">Vedangi</a></li>
<li><a href="#">Poonam</a></li>
</ul>
</div>
</div>
</body>
</html>
```

2. Styling Lists

- You can add rounded corners to your list with some margin by adding an attribute data-inset and setting its value to true.
- Thus, you can style your list to make it more professional.

Example

```
<div data-role="page" id="lists">
<div data-role="content">
<h3>Names List with Styling</h3>
<ul data-role="listview" data-inset="true">
<li>Siya</li>
<li>Arnav</li>
<li><a href="#">Vedangi</a></li>
<li><a href="#">Poonam</a></li>
</ul>
</div>
</div>
```

3. Dividing Lists

- You might have to divide the list items into different groups or sections to better organize your list. You can use list dividers to achieve this.
- You need to assign the value list-divider to the data-role attribute of the required <li> element.

Example

```
<div data-role="page" id="lists">
<div data-role="content">
<h3>Names divide</h3>
<ul data-role="listview" data-inset="true">
<li data-role="list-divider">Child</li>
<li>Siya</li>
<li>Aman</li>
<li data-role="list-divider">Adult</li>
<li><a href="#">Vedang</a></li>
<li><a href="#">Poonam</a></li>
</ul>
</div>
</div>
```

Q. 12 Explain different types of CSS classes to style the elements. (6 Marks)

Ans. : CSS Classes

Different types of CSS classes to style the elements are as follows :

1. Global Classes

These classes can be added on any jQuery Mobile widgets (buttons, toolbars, panels, tables, lists, etc.)

- **ui-corner-all** : Adds rounded corners to the element
- **ui-shadow** : Adds shadow to the element
- **ui-overlay-shadow** : Adds an overlay shadow to the element
- **ui-mini** : Makes the element smaller

2. Grid Classes

- Columns in a grid are of equal width (and 100% wide in total), with no border, background, margin or padding. There are five layout grids that can be used.

Grid Class	Columns	Column Widths
ui-grid-solo	1	100%
ui-grid-a	2	50% / 50%
ui-grid-b	3	33% / 33% / 33%
ui-grid-c	4	25% / 25% / 25% / 25%
ui-grid-d	5	20% / 20% / 20% / 20% / 20%

3. Button Classes

In addition to global classes, you can add the following classes to <a> or <button> elements (not <input> buttons)

- **ui-btn** : It specifies that the element will be styled as button.

- **ui-btn-inline :** It shows the button as inline element which saves the space as needed for the label.
- **ui-btn-icon-top :** It places the icon above the text.
- **ui-btn-icon-right :** It places the icon right of the text.
- **ui-btn-icon-bottom :** It places the icon below the text.
- **ui-btn-icon-left :** It places the icon left of the text.
- **ui-btn-icon-notext :** It shows the only icon.

4. **Theme Classes**

- **ui-bar-(a-z) :** Applies the toolbar theme styles for the selected swatch letter. Commonly used in conjunction with ui-bar structural class to add the standard bar padding styles.
- **ui-body-(a-z) :** Applies the content body theme styles for the selected swatch letter. Commonly used in conjunction with ui-body structural class to add the standard content block padding styles.
- **ui-btn-up-(a-z) :** Applies the button/clickable element theme styles for the selected swatch letter. Commonly used in with the ui-btn-hover-(a-z) and ui-btn-down-(a-z) interaction class states to provide visual feedback and ui-btn-active to indicate the selected or "on" state.
- **ui-shadow :** Applies the theme's global drop shadow to any element using CSS box-shadow property.

□□□

Unit VI : Web Application Deployment

Q. 1 Examine cloud computing models and explain types of clouds. (5 Marks)

Ans. :

Cloud Computing

- The word cloud computing is very popular nowadays. Cloud computing provides a simple way to access servers, storage, databases and a broad set of application services over the Internet.
- A cloud services platform such as Amazon Web Services owns and maintains the network-connected hardware required for these application services, while you provision and use what you need via a web application.
- The benefit of cloud computing is to replace the capital infrastructure expenses with low variable costs that scale your business.

Types of Clouds

Three types of clouds are available i.e. Public, Private, and Hybrid cloud.

1. Public Cloud
2. Private Cloud
3. Hybrid Cloud

1. **Public Cloud :** A cloud where the third-party service providers make resources and services available to their customers via the internet. Related data and security are with the service providers' owned infrastructure.

   Some of the largest public cloud providers include Alibaba Cloud, Amazon Web Services (AWS), Google Cloud, IBM Cloud, and Microsoft Azure.

2. **Private Cloud :** This is almost similar features as the public cloud, but the data and services are managed by the organization or by the third party only for the customer's organization. In this type of cloud, major control is over the infrastructure so security-related issues are minimized which makes it different from a public cloud.

3. **Hybrid Cloud :** As the name suggests Hybrid, is the combination of both private and public cloud. The decision to choose a type of cloud i.e. private or public usually depends on various parameters like the sensitivity of data and applications, industry certifications and required standards, regulations, and many more.

Q. 2 What are the various cloud computing models? (6 Marks)

Ans. :

Cloud Computing Models

There are three main models for cloud computing. Each model represents a different part of the cloud computing stack.

1. Infrastructure as a Service (IaaS)
2. Platform as a Service (PaaS)
3. Software as a Service (SaaS)

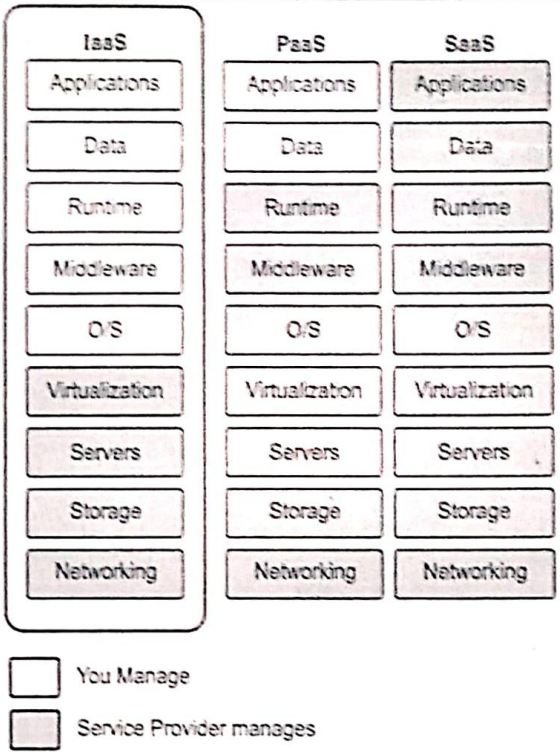


Fig. 6.1 : Cloud computing models

1. **Infrastructure as a Service (IaaS)**
   - IaaS, contains the basic building blocks for cloud IT and typically provide access to networking features, computers (virtual or on dedicated hardware), and data storage space.
   - IaaS means a cloud service provider manages the infrastructure for you the actual servers, network, virtualization, and data storage through an internet connection.
   - The user has access through an API or dashboard, and essentially rents the infrastructure. The user manages things like the operating system, apps, and middleware while the provider takes care of any hardware, networking, hard drives, data storage, and servers; and has the responsibility of taking care of outages, repairs, and hardware issues.
2. **Platform as a Service (PaaS)**
   - PaaS means the hardware and an application-software platform are provided and managed by an outside cloud service provider, but the user handles the apps running on top of the platform and the data the app relies on.
   - Platforms as a service remove the need for organizations to manage the underlying infrastructure (usually hardware and operating systems) and allow you to focus on the deployment and management of your applications.
   - This helps you be more efficient as you don't need to worry about resource procurement, capacity planning, software maintenance, patching, or any of the other undifferentiated heavy lifting involved in running your application.

3. Software as a Service (SaaS)

- Software as a Service provides you with a completed product that is run and managed by the service provider.
- With a SaaS offering you do not have to think about how the service is maintained or how the underlying infrastructure is managed; you only need to think about how you will use that particular piece of software.
- A common example of a SaaS application is web-based email where you can send and receive email without having to manage feature additions to the email product or maintaining the servers and operating systems that the email program is running on.

Q. 3 Briefly describe services offered by AWS. **(8 Marks)**

Ans. :

Services Offered by AWS

1. Analytics
2. Storage
3. Compute
4. Blockchain
5. Database
6. Developer Tool
7. Networking and Content Delivery
8. Security, Identity, and Compliance
9. Machine Learning
10. IOT

1. Analytics

- Analytics has been a primary source of growth for businesses.
- Data is always valued as it gives valuable information. Customers need a fast and scalable system to handle a big amount of data and provide useful insights.
- It offers a variety of applications for this purpose. Amazon EMR provides the Hadoop framework to process big data.
- Amazon Kinesis helps in analyzing real-time streaming data.
- AWS Data Pipeline and Glue provide pipeline structures schedule data load and processing.
- AWS provides many more applications for almost every operation.

2. Storage

- The data need to be stored somewhere to process it.
- It offers storage in three broad categories: object, block, and file storage. Amazon Simple Storage Service (S3) provides scalable data storage with backup and replication.
- Amazon Glacier offers storage for archived data and affordable retrieval.
- AWS backup service manages the backup of data. It automates the backup process.

3. **Compute**
   - Compute service is necessary for running any organization.
   - The computer is necessary from hosting a complete web app to executing a function in a serverless environment.
   - It offers a comprehensive portfolio for computing services like Amazon Elastic Compute Cloud (EC2) provides virtual servers or instances for computing.
   - It is auto-scalable as per the requirement. Amazon Elastic Container Service is a high-performance container service that supports Docker containers. AWS Lambda offers serverless computing to run applications.
   - Lightsail is an easy-to-use service that provides virtual servers, storage, DNS management, etc. It provides all the services required for the development of applications.
4. **Blockchain**
   - Blockchain is the new technology that helps customers work with multiple parties to maintain a verified transaction record.
   - Amazon Managed Blockchain creates and manages a blockchain network.
   - In addition, Amazon Quantum Ledger Database (QLDB) offers a fully managed ledger database to maintain transactions.
5. **Database**
   - A database can be used to store structured data.
   - It provides a broad range of database services to supports relational and non-relational databases.
   - In addition, it offers a service to handle all application-specific use cases. Amazon Relational Database Service provides a fully managed database service that includes Oracle, SQL, MySQL, etc.
   - Amazon Aurora offers a high-performance, fully managed relational database service. Amazon Timestream provides a fully managed time-series database.
   - Amazon DynamoDB provides database services for the NoSQL database. Along with these databases, AWS offers many other database services to support almost every type of requirement.
6. **Developer Tool**
   - Developer tools help a developer to deliver software quickly and safely.
   - It helps DevOps to automatically build, test and deploy the application in different environments. It also helps in maintaining source code and version control.
   - AWS Codestar helps the user set up a continuous delivery pipeline in minutes. AWS X-Ray helps to debug production applications.
   - With the help of an X-Ray, the user can analyze and identify performance issues and application components.
   - AWS CodeCommit provides fully managed private GIT repositories to store code and manage versions.
   - Apart from these services, AWS provides AWS CodePipeline, AWS CodeBuild, AWS CodeDeploy, AWS CLoud9 to support development and deployment.
7. **Networking and Content Delivery**
   - AWS is a virtual private cloud; it offers services over a network. Hence it ensures that AWS can run any workload over the network with security, performance, manageability, and availability.

- It offers a set of resources over the network by connecting it privately. It gives administrative control to users over a virtual network.
- It provides an application for load balancing in the networks. It also offers DNS to route end users to the application.

8. **Security, Identity, and Compliance**
   - While offering services over internet security is the utmost priority. With AWS Identity and access management, users can control user access and manage encryption keys.
   - AWS Firewall Manager helps manage firewall rules for applications.
   - Amazon Inspector is an automated security scan that helps in improving the security and compliance of applications.
   - Amazon Macie is a machine learning-powered service to identify, classify and protect sensitive data. Apart from these security check services, AWS provides a lot more applications to keep the hosted applications secure and safe.

9. **Machine Learning**
   - Learning from existing data is called machine learning or artificial intelligence.
   - AWS offers a wide range of services and pre-defined models for AI. Amazon SageMaker provides services to quickly build, train and deploy models at a big scale.
   - It also supports custom model building. Amazon Recognition is used to analyze images and videos. Along with these, AWS offers ML service for speech recognition, language translation, chatbots, and many other scenarios, with high speed and scalability.

10. **Internet of Things (IoT)**
    - AWS offers Internet of Things (IoT) services and solutions to connect and manage billions of devices.
    - Collect, store, and analyze IoT data for industrial, consumer, commercial, and automotive workloads.
    - AWS IoT offers a range of services to connect with the devices, operate and collect data from the devices and perform specific actions on the data and store the information in the AWS cloud.
    - Free RTOS - AWS offers Free RTOS, an open-source and real-time operating system, to deploy and manage devices such as sensors (microcontrollers). Greengrass is an open-source runtime and cloud service service that allows businesses to build, deploy, and manage device software faster and in an intelligent manner.
    - The AWS IoT core service allows all connected devices to securely interact with the applications running on the AWS cloud infrastructure as well as other devices.
    - AWS IoT events is used to detect and respond to events from various IoT sensors and applications.

Q. 4 What are the features of AWS EC2. **(4 Marks)**

Ans. :

Features of AWS EC2

Following is the list of some of the silent features of EC2

- **Scalable Computing** : Amazon EC2 provides scalable computing where the number of instances can be increased or decreased within seconds. Scaling multiple services at the same time can be done using the technique known as Auto Scaling.
- **Reliable** : Amazon EC2 agrees to a highly reliable environment where rapid replacement of instances is.

- **Fully-Controlled** : Similar to normal machines, users have complete control on the root in Virtual machines. By keeping the data in the Boot partition, Instances can be stopped and restarted using the APIs. Also, the user can access the console output of the instance.
- **Easy to Start** : AWS EC2 can be started easily by using AWS Management Console, AWS SDKs or AWS Command Line Tools (CLI).
- **Designed for AWS** : Amazon EC2 works well with different Amazon services like S3, RDS, Dynamo DB, and SQS.
- **Secure** : Amazon EC2 works with Amazon Virtual Private Cloud to offer a secure network to compute resources. Users can choose their own range of IP Addresses, routers and create subnets based on their needs. It is because the user controls their virtual environment completely through Amazon VPC.
- **Flexible Tools** : Amazon EC2 offers different operating systems, software's of different operating systems, software's of different configurations.
- **Inexpensive** : Amazon EC2 follows the 'Pay Per Use' method in which users have to pay for the resources they chose without any hidden or sudden expenses.

Q. 5 Explain EC2 fundamentals concepts. **(6 Marks)**

Ans.: EC2 Fundamentals Concepts

Following are some key terms needs to understand before creating EC2 instance

- **Regions and Availability Zones :** Amazon EC2 is hosted in multiple locations world-wide.
- These locations are composed of regions and Availability Zones. Each region is a separate geographic area. Each region has multiple, isolated locations known as Availability Zones.
- Each region is completely independent. Each Availability Zone is isolated, but the Availability Zones in a region are connected through low-latency links.
- AZ-1a, AZ-1b, and AZ-1c are availability zones in the region. When you launch an EC2 instance, you must select an AMI that's in the same region. Now select an Availability Zone or let AWS choose for you. After creating the EC2 instance, it will show up in selected Availability Zone.

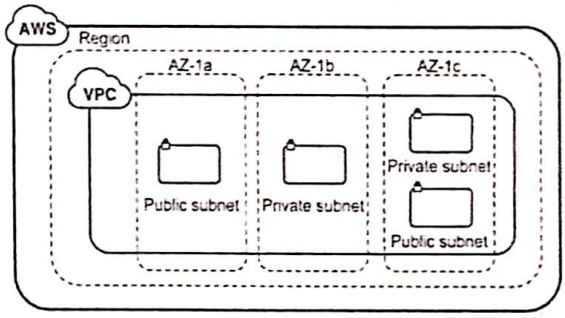


Fig. 6.2

- **Amazon Machine Image :** An Amazon Machine Image (i.e. AMI) provides the information required to launch an EC2instance.
- You can launch multiple instances from a single AMI when you need multiple instances with the same configuration. You can launch an instance from an existing AMI, customize the instance, and then save this updated configuration as a custom AMI.
- Instance type : Instance types comprise varying combinations of CPU, memory, storage, and networking capacity and give you the flexibility to choose the appropriate mix of resources for your applications.

- **Security Groups :** EC2 Security Groups acts as a virtual firewall that controls the traffic for one or more instances. When you launch an instance, you can specify one or more security groups. If you don't a default security group will be used.
- You can add rules in terms of protocols, ports, and source IP ranges in each security group that allow traffic to or from its associated instances. You can modify the rules for a security group at any time.
- When you launch an instance in a VPC, you must specify a security group that's created for that VPC. After you launch an instance, you can change its security groups.
- Security groups are associated with network interfaces. Changing an instance's security groups changes the security groups associated with the primary network interface
- EC2 Key Pairs : The public and private keys are known as a key pair. Public-key cryptography enables you to securely access your instances using a private key instead of a password.
- You can use Amazon EC2 to create your key pair. Each key pair requires a name. Be sure to choose a name that is easy to remember. Amazon EC2 stores the public key only, and you store the private key.
- Anyone who possesses your private key can decrypt your login information, so it's important that you store your private keys in a secure place.
- Elastic IP Address : Within an AWS infrastructure, you will have VPCs, within the VPCs, you have instances. When you launch an EC2 instance, you receive a Public IP address by which that instance is reachable from internet.
- Once you stop that instance and restart the instance you get a new Public IP for the same instance. This is a problem to connect your instance from internet for not having a static IP.
- To overcome this problem, you need to attach an Elastic IP to an Instance which doesn't change after you stop & start the instance. In short, an Elastic IP is a permanent IP for your instance.
- EC2 tagging : AWS allows customers to assign metadata to their AWS resources in the form of tags. Each tag is a simple label consisting of a customer-defined key and an optional value that can make it easier to manage, search for, and filter resources.
- Tags enable you to categorize your AWS resources in different ways, for example, by purpose, owner, or environment.
- This is useful when you have many resources of the same type, you can quickly identify a specific resource based on the tags you've assigned to it.
- For example, you could define a set of tags for your account's Amazon EC2 instances that helps you track each instance's OWNER (E.g. DBAdmin, User, etc) and ENV (E.g. TEST, PROD, etc).
- On-demand instance : On-Demand instances are servers that run in EC2 or AWS Relational Database Service(RDS) and are purchased at a fixed rate per hour.
- They are also suitable for use during testing and development of applications on EC2. This is generally the most expensive purchasing option for AWS instances, and the hourly prices vary depending on the operating system and size of the instance.
- Spot instance : Spot instances are spare EC2 capacity that can save you up 90% off of On-Demand prices that AWS can interrupt with a 2-minute notification.
- Spot uses the same underlying EC2 instances as On-Demand and Reserved Instances, and is best suited for fault-tolerant, flexible workloads.
- Spot instances provides an additional option for obtaining compute capacity and can be used along with On-Demand and Reserved Instances

- **Terminating an instance :** You cannot restart a terminated instance, but you can restart a stopped instance, but the data stored on the instance will be lost.
- You can stop an Amazon EBS backed instance, but not an s3-backed instance. When the instance is stopped, you're not charged for any instance hours, only for the storage volume.
- When an instance is terminated, the instance performs a normal shutdown, then the attached Amazon EBS volumes are deleted unless the volume's delete On Termination attribute is set to false.

Q. 6 Explain types of elastic load balancer. **(6 Marks)**

Ans. :

Types of Elastic Load Balancer

AWS supports four types of load balancers.

1. Classic Load Balancer(CLBs)
2. Application Load Balancers(ALBs)
3. Network Load Balancers (NLBs)
4. Gateway Load Balancers(GLBs)

1. Classic Load Balancer (CLBs)

- The classic load balancer operates at layer 4 and layer 7.
- So, it supports protocols like TCP and TLS, which are the transport layer protocols, and it supports protocol like HTTP and HTTPs, which are the application layer protocols. classic load balancer is not recommended by AWS at all.
- Classic Load Balancer supports SSL termination, including offloading SSL decryption from application instances, centralized management of SSL certificates, and encryption to back-end instances with optional public key authentication.
- Flexible cipher support allows you to control the ciphers and protocols the load balancer presents to clients.
- Classic Load Balancer supports the use of both the Internet Protocol version 4 and 6 (IPv4 and IPv6) for EC2-Classic networks. Classic Load Balancer has quite a few limitations.
- For example, it cannot forward traffic on more than one port per instance. It also doesn't support forwarding to IP addresses or WebSocket.

2. Application Load Balancer (ALBs)

- The application load balancer supports layer7 protocols which are HTTP, HTTPS, web sockets, and things like that.
- It also supports advanced routing approaches. You can look at the request, let's say the request header or you can look at the request path or the request hostname and based on it, you can route it to appropriate targets. So, this is a highly flexible and a widely used load balancer type in AWS. ALB supports HTTP/2 and websockets.
- It has a wide range of routing rules for incoming requests based on host name, path, query string parameter, HTTP method, HTTP headers, source IP, or port number
- An interesting feature of ALB is that it supports user authentication via a variety of methods, including OIDC, SAML, LDAP, Microsoft AD, and well-known social identity providers such as Facebook and Google.
- This can help you off-load the user authentication part of your application to the load balancer.

- ALBs are typically used for web applications. If you have a microservices architecture, ALB can be used as an internal load balancer in front of EC2 instances or Docker containers that implement a given service. You can also use them in front of an application implementing a REST API

3. **Network Load Balancer (NLBs)**
   - Network load balancer is a new generation load balancer. It works at layer 4, the transport layer. This supports the layer 4 protocols, TCP, TLS, and UDP.
   - Its main feature is that it has a very high performance. Also, it uses static IP addresses and can be assigned Elastic Ips not possible with ALB and ELB.
   - A typical use case would be a near real-time data streaming service (video, stock quotes, etc.) Another typical case is that you would need to use an NLB if your application uses non-HTTP protocols.

4. **Gateway Load Balancer (GLBs)**
   - Gateway Load Balancer helps you easily deploy, scale, and manage your third-party virtual appliances.
   - It gives you one gateway for distributing traffic across multiple virtual appliances while scaling them up or down, based on demand.
   - This decreases potential points of failure in your network and increases availability.
   - You can find, test, and buy virtual appliances from third-party vendors directly in AWS Marketplace.
   - This integrated experience streamlines the deployment process so you see value from your virtual appliances more quickly whether you want to keep working with your current vendors or try something new.

Q. 7 What are the various components of VPC? **(10 Marks)**

Ans. : Components of AWS Virtual Private Cloud

- There are six core components which are fundamental to a VPC and will be created by a user or by AWS as part of a default VPC. These components are:

1. Subnet
2. Route Table
3. Security Group
4. Network Access Control Lists (ACLs)
5. CIDR Block
6. Gateways

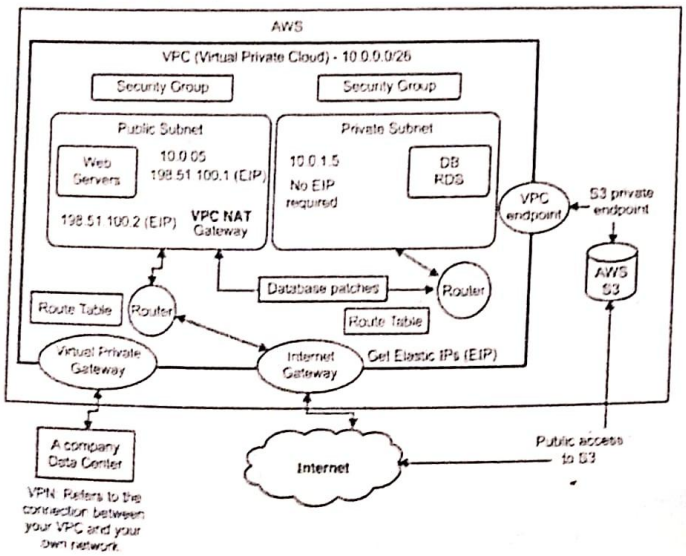


Fig. 6.3(a) : VPC & its components

1. **Subnet**
   - A subnet is a range of IP addresses in your VPC. You can launch AWS resources into a specified subnet. Use a public subnet for resources like a web server that must be connected to the internet, and a private subnet for resources that won't be connected to the internet like application servers and database servers.
   - By default all EC2 instances is assigned with a private ip addresses for internal communication in the VPC. Providing an EC2 instance with a public ip address allow your EC2 instance to communicate outside a VPC with an internet.
   - An Elastic IP is a combination of a public IP address and a static IP address. It allows you to continue to advertise AWS instances within your AWS network infrastructure. Static IP addresses are IPs which do not change. A public IP address is the IP address that communicates your internet connected device to the public internet.
   - A public IP address is assigned to your instance from Amazon's pool of public IP addresses. It's not associated with your account.
   - When a public IP address is disassociated from your instance, it's released back into the pool, and is no longer available for you to use. If you require a persistent public IP address that can be assigned to and removed from instances as you require, use an Elastic IP (i.e EIP) address instead.
2. **Route table**
   - A route table contains a set of rules, called routes, that are used to determine where network traffic is directed. When you create a VPC, it automatically has a default main route table.
   - Your VPC can have route tables other than the default table. One way to protect your VPC is to leave the main route table in its original default state (with only the local route), and explicitly associate each new subnet you create with one of the custom route tables you've created.
   - This ensures that you explicitly control how each subnet routes outbound traffic. Each route in a table specifies a destination CIDR (E.g.) and a target.
3. **Security Groups**
   - AWS offers virtual firewalls via Security Groups. AWS security groups (SGs) are associated with EC2 instances and provide security at the protocol and port access level.
   - Each security group – working much the same way as a firewall – contains a set of rules that filter traffic coming into and out of an EC2 instance.
   - Security groups do not deny traffic, which means all the rules in security groups are positive, and allow traffic.
   - Whilst security group rules can be set to specify a traffic source, or a destination, they cannot specify both on the same rule.
   - A single security group can be applied to multiple instances, or multiple security groups can be applied to a single instance.
   - The actual rule set that filters traffic is made up of two tables: 'Inbound' and 'Outbound'. AWS Security groups are stateful, meaning you do not need the same rules for both outbound traffic and inbound.
   - Therefore any rule that allows traffic into an EC2 instance, will allow responses to pass back out without an explicit rule in the Outbound rule set.
   - Each rule is comprised of four fields: 'Type', 'Protocol', 'Port Range', and 'Source'.

4. **Network Access Control List**

- In addition to the security groups to further enrich its security filtering capabilities, AWS offers a feature called Network Access Control Lists (NACLs). Like security groups, each NACL is a list of rules, but there are two important differences between NACLs and security groups. Security Groups are applied at the instance level.
- Firstly, NACLs are not directly tied to EC2 instances, but are tied with the subnet within your AWS VPC.
- This means that the rules in a NACL apply to all of the EC2 instances within the subnet, in addition to all the rules from the security groups.
- Secondly, the NACLs support 'deny' rules to block traffic from a particular set of IP addresses which are known to be compromised.
- The ability to write 'deny' actions is a crucial part of NACL functionality. When you have the ability to write both 'allow' rules and 'deny' rules, the order of the rules becomes important.
- "Rule numbers' are used within each NACLs to identify the correct order of the rules for your needs. You can choose which traffic you deny and which traffic you allow.
- For inbound traffic, AWS's infrastructure first assesses the NACL rules.
- If traffic gets through the NACL, then all the security groups that are associated with that specific instance are evaluated.
- For outbound traffic, this order is reversed, where the traffic is first evaluated against the security groups, and then finally against the NACL that is associated with the relevant subnet.

5. **CIDR block**

- A system called Classless Inter-Domain Routing, or CIDR, was developed as an alternative to traditional subnetting.
- The idea is that you can add a specification in the IP address itself as to the number of significant bits that make up the routing or networking portion.
- The CIDR notation of 192.168.0.15/24 means that the first 24 bits (i.e. the first three octets) of the IP address are considered significant for the network routing.
- The last octet can have 255 ip addresses from 0 to 255.
- There are various calculators and tools online that will help you understand some of these concepts and get the correct addresses and ranges that you need by typing in certain information.

6. **Gateways**

- Your EC2 instances will require connectivity outside of AWS to the Internet or to a user's corporate network via the use of gateways.
- AWS provides several options for connecting to an Amazon virtual private cloud (VPC) like
- VPN, AWS Direct Connect, VPC endpoints, VPC peering, EC2 ClassicLink, Internet Gateway and NAT Gateway.

i. **Internet Gateway**

- For communication with the Internet, a VPC must be attached to an Internet gateway. An Internet gateway is a fully managed AWS service that performs bi-direction source and destination network address translation for your EC2 instances.

- Optionally, a VPC may use a virtual private gateway to grant instances secure access to a user's corporate network via VPN or direct connect links. Instances in a subnet can also be granted outbound only Internet access through a NAT gateway.

ii. VPC endpoints

- VPC endpoints enable you to create a private connection between your VPC and another AWS service that resides outside the VPC like Amazon S3 & DynamoDB without the need for Internet access.
- If you wanted your EC2 instances in a VPC private subnet to be able to privately access resources outside VPC like S3 buckets & DynamoDB, you had to use an Internet Gateway, and potentially manage some NAT instances, which can be expensive as you have to pay an hourly fee and a data processing fee.
- VPC Endpoints simplify access to S3 resources from EC2 instances within a private subnet in VPC.
- These endpoints are easy to configure, highly reliable, and provide a secure connection to S3 that does not require a gateway or NAT instances.
- EC2 instances running in private subnets of a VPC can have controlled access to S3 buckets, objects, and API functions that are in the same region as the VPC using the endpoints.
- You can use an S3 bucket policy to indicate which VPCs and which VPC Endpoints have access to your S3 buckets.

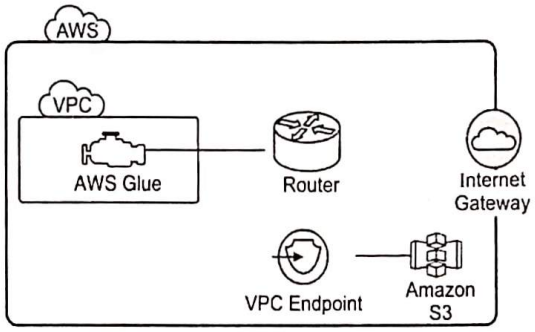


Fig. 6.3(b)

iii. VPC peering

- A VPC peering connection allows you to route traffic between two Virtual Private Cloud's using IPv4 or IPv6 private addresses.
- Instances in either VPC can communicate with each other as if they are within the same network.
- You can create a VPC peering connection between your own VPCs, or with a VPC in another AWS account. A VPC peering connection helps you to facilitate the transfer of data

iv. NAT Gateway

- You can use a network address translation (NAT) instance in a public subnet in your VPC to enable instances in the private subnet to initiate outbound IPv4 traffic to the Internet or other AWS services, but prevent the instances from receiving inbound traffic initiated by someone on the Internet.

v. VPN

- You can connect your Amazon VPC to remote networks like your own on premise data centre over the public internet by using a VPN (i.e. Virtual Private Network) connection.
- On the AWS side of the VPN connection, a virtual private gateway provides two VPN endpoints (tunnels) for automatic failover. You configure your customer gateway on the remote side of the VPN connection.
- The maximum throughput of a Site-to-Site VPN connection is typically less than 4.0 Gbps.

vi. **AWS Direct Connect**

- You can also use AWS Direct Connect to create a dedicated private connection from a remote network to your VPC. AWS Direct Connect bypasses the public Internet and establishes a secure, dedicated connection from your infrastructure into AWS.
- This dedicated connection occurs over a standard 1 GB or 10 GB Ethernet fiber-optic cable with one end of the cable connected to your router and the other to an AWS Direct Connect router.
- AWS Direct Connect is a great option for businesses that are seeking secure, ultra-low latency connectivity into AWS.
- While provisioning AWS Direct Connect can sometimes be more involved, but it is worth the effort once the connectivity is established as it has predictable network performance.

Q. 8 Explain types of AWS storage services. **(5 Marks)**

Ans. : Types of AWS Storage Services

AWS offers seven types of storage services with choices for back-up, archiving and recovery of lost data. Let's see what those services are and their features :

1. **Simple Storage Service (S3)**

- Amazon S3 is the most popular, very, very flexible, and inexpensive storage service provided by AWS. Amazon S3 is typically recommended for storing large objects with a key-value approach.
- So, this is very, very similar to how you store values into a hashmap. You assign a key and put a object against it.
- So, that's exactly how you'd store stuff in to S3.You can store a file and associate it with a key. Amazon S3 also provides you with a REST API to access and modify the objects which are stored in S3.
- Amazon S3 provides you with unlimited storage. The default storage class for S3 provides you with 99.99 percent availability.
- The chance that you would lose some data which is put into S3 is very remote.
- To achieve durability, your objects are replicated inside a single region across multiple AZs, across multiple availability zones.
- You can use S3 to store any kind of files: text, binary, backup, archives.
- It can store data for any business such as web applications, mobile applications, backup, archive, analytics.
- Thus, S3 provides unlimited storage with very good availability and very, very high durability.

2. **Elastic Block Storage (EBS)**

- Block Storage is your hard disk which is attached to your computer. Typically, one Block Storage device can be attached to one virtual server, one computer, or one laptop.
- However, you can connect multiple different block storage devices to one virtual server. So, over here Virtual Server A is attached with a block storage, Virtual Server B is attached with multiple block storage devices. But one block storage device cannot be attached with multiple virtual servers.
- Typically, block storage devices are used in two ways.
- One is Direct-attached storage, similar to a hard disk. The hard disk is directly connected to your computer.
- The other option is on the network. You can also create a pool of block storage devices together. You can create a network between them and you can use them.
- Creating a network of block storage devices and using them is called a Storage Area Network.
- Typically, this is done when you need huge performance. Let's say you would want to create a cluster of databases. For example, Oracle or Microsoft SQLServer clusters.

- EBS volumes are placed in availability zones so that they are replicated to prevent loss of data due to single component failures.
- They provide absolute low-latency performance and you can also scale up or down your resources as and when required.
- EBS is available in both SSD and HDD formats depending on your requirement of speed and volume.

3. Elastic File System (EFS)

- EFS is a managed network file system that is easy to set up right from the amazon console or CLI.
- When you have multiple EC2 instances needed to access the same file system EFS helps in providing just that.
- The typical use cases would be something like a media workflow where you are editing a video file and a lot of people are involved in editing.
- So, you put the files on a file share and you do the editing together.
- You might be in an enterprise where you would want to be able to share your files with other users.
- These are the use cases where you go for file storage and file storage are shared between multiple virtual servers. The file storage options which are provided by AWS are Amazon EFS, Elastic File Server.
- EFS is much more expensive than EBS as it can be used on very large analytical workloads.
- EFS scales up or down based on the size of the files you store and is also accessible from multiple availability zones.
- The distributed nature of the file system can tempt you to use it as a CDN. But the costs of a CDN outweigh the benefits of using EFS.
- Hence it is better to use a CDN and use EFS in conjunction with files that can't be stored on a CDN.

4. Amazon FSx for Lustre

- Luster is a file system used for compute-intensive workloads.
- This mainly comes into the picture when you run machine learning operations on large data sets or when you need to run media encoding workloads.
- Running Lustre separately requires a lot of expertise in setting it up and configuring it for the right workloads.
- With the help of Amazon FSx, this can be avoided and a simple interface on the console helps you to quickly get started and start working on your data.
- The ability to connect it seamlessly to S3 and the option of running it in VPC provides a low cost yet a performant way to achieve your compute-intensive workloads leveraging luster without the administrative overhead of running it.

5. Amazon S3 Glacier

- The glacier is used mainly for archival and long-term data storage.
- This means that there is a low retrieval rate on this storage system due to which it is offered at an extremely cheap rate.
- It does also come with compliant security features to encrypt your data.
- Glacier allows you to run queries and analytics on it directly and you will be charged only for the few minutes or hours when you read the data. In terms of durability, it offers 99.(11 nines)% durability which is one of the highest in the industry.
- Glacier hopes to replace the legacy on-premise tape-based backup service with a much more cost-effective and durable solution.

6. **Amazon FSx for Windows File Server**

- Whenever you need to run your windows specific software that needs to access the proprietary windows file system on the cloud, AWS provides you with Amazon FSx to easily achieve that.
- Windows-based .Net applications, ERPs and CRMs require shared file storage to move workloads between them.
- Also, Amazon FSx provides support for all native windows based technologies such as NTFS, SMB protocol, Active Directory (AD) and Distributed File System (DFS).
- Similar to luster, Amazon FSx eliminates the administrative overhead for setting up and maintaining a windows file server and provides you with a simple cost-effective way to run your windows file server on AWS.

7. **AWS Storage Gateway**

- Storage Gateway is a simple way to let your on-premise applications store, access or archive the data into the AWS cloud.
- This is achieved by running on a hypervisor on one of the machines in your data center which contains the storage gateway and then is available on AWS to connect to S3, Glacier or EBS.
- It provides a highly optimized, network resilient and low-cost way to move your data from on-prem to the cloud. Local caching is also available on your on-prem to allow for accessing the more active data.
- Storage gateway also supports legacy backup stores such as tapes as virtual tapes backed up directly into AWS Glacier.

Q. 9 Explain services are used to deploy your website/ web application. **(8 Marks)**

Ans. : Deploy Website or Web Application on AWS

- There are multiple options for you if you decide to host your website /application on the AWS cloud.
- Following services are used to deploy your website/ web application.

1. **Amazon Elastic Compute Cloud**

- Amazon Elastic Compute Cloud (Amazon EC2) provides you a virtual server with different capacities to best suit your needs allowing you to scale up or down to handle changes in requirements or spikes in traffic.
- EC2 is technically the most challenging one how you can host your own website on the AWS cloud as the entire server configuration, OS updates, security settings falls on your shoulders (both initial setup + ongoing management).
- Hosting your website on EC2 is the right choice for you if you wish to maintain the maximum control and flexibility of your web server configuration and administration.

2. **Amazon S3**

- Amazon S3 provide a built-in (native) support making it possible for you to host static websites.
- As it is a serverless service, you don't need to manage any underlying server infrastructure - you just need to upload your files to your S3 bucket and configure it for public access.
- If you want to optimize your website performance while effectively managing costs, you can additionally set up Amazon CloudFront to work with your S3 bucket and serve the content for your website visitors.
- CloudFront is a content delivery network (CDN) service that delivers website content around the world, securely and at scale. By design, delivering data out of CloudFront can be more cost effective than delivering it from S3 directly.

---

3. **AWS Amplify**
   - AWS Amplify offers a fully managed service for deploying and hosting static web applications, with built-in CI/CD workflows that accelerate your application release cycle.
   - Simply connect your application's code repository in the Amplify console, and any changes committed to the repository will trigger the pipeline which will build, test and deploy the changes to your target environment.
   - AWS Amplify also provides instant content delivery network (CDN) cache invalidation, atomic deploys, password protection, and redirects without the need to manage any servers.
   - Amplify has native support for JavaScript single page applications (SPA) written in a variety of frameworks like React, Vue, or Angular.
   - AWS Amplify runs S3, Cloudfront and Route 53 for you in the background so you don't have to worry about configuring these services.
4. **Amazon Lightsail**
   - Amazon Lightsail is nothing else than a preconfigured virtual private server plan that includes everything you may need to easily deploy and manage your dynamic web applications that rely on using PHP, Node.js or Ruby backends (like Wordpress, Django, Drupal, Joomla, Magento).
   - You can run both Linux and Windows instances, you get the ability to set up DNS, static IP addresses, load balancers, and connectivity to a VPC to access private resources like RDS hosted databases. Amazon Lightsail was introduced by AWS as a simpler alternative to EC2 so that new users could get started quickly with AWS since EC2 required considerable effort and expertise for its setup and configuration.
   - Website hosting using Amazon Lightsail is recommended only if your web applications can afford some downtime.
   - Unlike EC2 or AWS Lambda which can scale based on incoming requests, Lightsail can only work with the computing power that you have purchased - there is no automatic upscaling. Amazon Lightsail uses clear and fixed pricing model
5. **AWS Elastic Beanstalk**
   - AWS Elastic Beanstalk is an orchestration service that will setup infrastructure services as you desire to support applications developed in Go, Java, .NET, Node.js, PHP, Python, and Ruby.
   - When you deploy your application, Elastic Beanstalk builds the selected supported platform version and provisions one or more AWS resources, such as Amazon EC2 instances, to run your application.

Q. 10 Explain steps for deploying web application on AWS. **(10 Marks)**

Ans. : Deploy web application on AWS

- A Single Page Application is a web application that doesn't need to reload the page during its use and works within a browser that interacts with user without making a request to the server to fetch new HTML.
- Following are steps required to deploy SPA to AWS using S3

1. **Sign Up for AWS and Create IAM Users**
   - The first step is signing-up for a Amazon Web Services account. Once you sign up for an AWS account, you get automatic access to hosting services and will be charged for only the services that you use.
   - If you already have an AWS account, skip this step and go straight to the creation of IAM (Identity and Access Management) users. All services within the scope of AWS require valid credentials to gain access, which ensures that only authorized users are able to access the resources of the service.

- According to AWS recommendations, IAM roles are a must to restrict unauthorized access to the resources. An administrator, therefore, should create an IAM user and grant administrative permissions so that the user can access AWS services through a special URL.

2. Create Buckets for the Website

- A bucket can be considered as a file folder on the internet that has its unique naming convention over the entire web.
- IT administrators can use Amazon Simple Storage Service, or Amazon S3, to be precise, to store all content such as CSS files, HTML, and Javascript pages that are the building blocks for your website.
- If you intend to attach a domain name to the website, you need to book the domain first before going ahead with the creation of buckets.
- This is because your bucket name should be the same as the domain name, so that Amazon S3 can resolve the host headers sent by web browsers, whenever someone requests content from your website.

3. Configure Buckets

- After the bucket creation is over, the next step is to configure the root domain buckets, so that normal users can access the website data on the browser.
- When buckets are created, only the administrator has the default permissions to access them and their contents in such a way that the data is not accidentally exposed to unauthorised users.
- The objective of the exercise is creating a website to get traction from visitors and sell products and services. For that to happen, you have to manually configure the buckets and add special permissions for visitors to your website.
- AWS hosting also allows you to track the visitors accessing your website, including their IP address and the in-out traffic, through the bucket.
- There are no charges to enable logging on the bucket, but you may need to pay more to access the information contained in the bucket.

4. Deploy the Web Application

- Deploy web application include creation of an index and custom error document, uploading necessary files to the bucket, configuring the bucket as a website, setting up a redirect, and finally the most important step, which is testing the website to check for any bugs or tune ups that may be required.
- Choosing a redirect also has many advantages as you can maintain a single version of website files in Amazon S3, which supports the root and sub-domains of the web address.

5. Speed Up the Website

- you can use Amazon Cloudfront to improve the performance of the deployed website or web application.
- CloudFront ensures that copies of your website content are available on data centers around the world, called edge locations, and once a visitor requests access, the nearest data center entertains the request, leading to faster download times and improved performance.
- The feature consistently checks the origin server for a newer version of the content, and once available, it gets replicated or passed on to the nearest edge location.
- This step is primarily concerned with optimising website performance, which involves three steps – creation of cloudfront distribution, updating the record sets for domains and subdomains, and checking the log files.

□□□