Amrita Vishwa Vidyapeetham

TIFAC-CORE in Cyber Security

20CYS312 - Principles of Programming Languages

Assignment-01: Exploring Programming Paradigms

Iniyan R

21st January, 2024

- Paradigm 1: Reactive
- Language for Paradigm 1: Angular
- Paradigm 1: Paradigm 2: Aspect-Oriented
- Paradigm 1: Language for Paradigm 2: Django
- Analysis
- Comparison
- Challenges Faced
- Conclusion
- References

1 Paradigm 1: Reactive

1.1 Overview

- Reactive programming is a declarative programming paradigm based on asynchronous event processing and data streams.
- Used for :
  - Smooth User Interface (UI)
  - Real-time updates
- Reactive programming enables improvement of responsive programs by reacting to changes and events in real-time. This responsiveness is important for interactive and dynamic user experiences.
- Reactive programming is based on event-driven architecture.So when use performs some actions which will cause a change in the state an equivalent reaction is performed throughout the system
- Reactive programming are now also being used in cloud computing.

1.2 Languages based on Reactive Paradigm

- JavaScript (with RxJS)
- Java (with Project Reactor)
- Kotlin
- Python (with RxPy)
- Scala
- Angular
- Dart (with RxDart)

1.3 Principles for building an reactive application

Figure 1: Principles of a reactive application

- Stay responsive - Respond in regular time intervals helps in low delay and good user experience.
- Accept uncertanity - Reliable ,for example, error handling.
- Embrace failure - if a failure occurs in system build a system that can recover from it .
- Assert autonomy - system should operate independantly but should do the task collabratively.
- Tailor consistency - balance availability and performance for each component
- Decouple time - System should work asychornously.
- Decouple space - flexible and distributed systems.
- Handle Dynamics - system to should adapt to varying demand and resources.

1.4 Asynchronous Event Processing

- In Asynchronous Event processing processing of one event doesn't block the processing of other event.
- So when an task takes long time for running and need to be started at first other tasks can be did simaltaneously without waiting for that task.
- Processing units don't block each other in asynchronous event processing.
- Working :
  - As Events arrive they are placed in the event queue.
  - Available processing units process the events from queue.
  - Each unit processes its event independently, emitting results as output streams.
  - The system remains responsive to new events even during processing.
  - Backpressure mechanisms manage event flow to maintain stability.
- Example - In an web application consider an user who uploads files or downloads so when at that time user can simaltaneously access other features of the application without any delay.

1.5 Data Streams

- Data stream are a series of data items generated sequentially over time from a particular source.
- Data streams are coherent and cohesive collections of digital signals that are generated continual or near-continual basis.
- They encapsulate asynchronous or event-driven data, like :
  - user inputs
  - network requests
  - and sensor readings.
- Data streams can be either finite or infinite, continuing indefinitely.

1.6 Example Code

- Imperative Paradigm :

  var y = 10
  var z = 90
  var x = y * z
  y = 100
  console.log(x)

  In this value of x remains 900 even after y is updated.to update it the value of x should be updated again.

- Reactive paradigm :

  In this consider an operation '$=' instead of = which updates the value of x when the value of y or z is updated.

  var y = 10
  var z = 90
  var x $= y * z
  y = 100
  console.log(x)

  In this the value of x will be 9000 as y is updated to 100 the operation is performed again and the value of y is updated.

1.7 Producer - Consumer Problem

- Producer - Producer is an entity that emits data or events to one or more consumer.It can be a source of data like databases, web services, sensors, or user inputs. Producers are classified as hot or cold based on whether they start emitting data before or after consumer subscribes for data. A producer can be unicast, emitting the same data to each consumer, or multicast, delivering different data to each consumer.

- Consumer - Consumer is an object designed to receive and process data from a source. Consumers can subscribe to one or more sources, engaging in actions on the data they receive. These actions may include printing, filtering, transforming, storing, or forwarding the data to another consumer.

1.8 Implementation

- Pull - The consumer regularly checks for values and reacts whenever a relavant value is available.This process of regularly checking for values is known as polling.
- Push - Whenever there the value contain all information and no further quering is needed the consumer pushes.
- Push-Pull - So,if there is a change in value the a change notification is received by the consumer(eg.some value changed).This is push.So the change notification wont contain all information so the consumer should query again from the producer.This is pull.

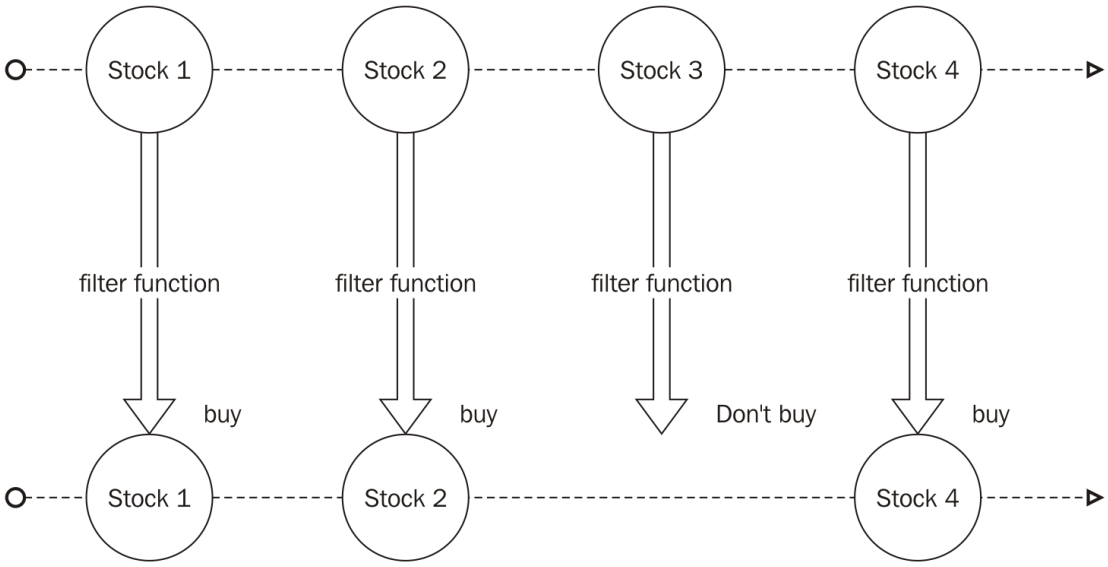
1.9 Challenges in implementing

- There can be too many requests to source(producer) after a change notification is sent overwhelming the producer.
- Backpressure - Producer produces more data than the consumer can receive it.Buffering,throttling can be used to control backpressure else the data may be dropped.
- Memory consumption is high because stream value should be stored.

1.10 Error handling

In reactive programming the errors are propagated . When an error occurs at any point in the stream, the normal flow of data is interrupted and the error is sent down the stream to consumers.Try-catch can be used for handling errors in Java similarly different languages different error handling mechanisms.

1.11 Reactive Programming in Trading



The changes in value of stock can be seen as event stream and it can be used for informing the user whether to buy a stock or not

1.12 Reactive Programming Advantages

- Responsiveness
- Scalability - In high-traffic applications concurrently the requests can be handled.
- Error handling
- Compatibility- Reactive frameworks are available for many programming languages available in different libraries

1.13 Applicationns using Reactive programming

- Social media - Instagram,Facebook...
- Streaming applications - Netflix
- Travel and navigation apps - google map
- IoT applications
- AWS cloud computing
- Trading platforms-Groww,Zerodha

---

2 Language for Paradigm 1: Angular

2.1 Introduction

- Angular is a MVVM (Model-View-ViewModel) framework.
- Angular is used for building Single-Web applications using html and typescript.
- Angular was written in typescript
- AngularJs version 1 was released in 2012.AngularJs was developed by Miško Hevery from 2009.And this was fully rewritten in typescript and and was named Angular 2.0,as of now angular version 17 is the latest.
- Angular is based on declarative and reactive programming.

2.2 MVVM(Model-View-ViewModel)

- Model - Itis application's data and business logic. It is used for retrieving, storing, and manipulating data.Model notifies the ViewModel of any changes in the data.
- View - It is used for displaying UI to the user. It is used for displaying the visual elements, like buttons, textboxes, that users interact with. View observes the ViewModel for changes in the data it needs to display.
- ViewModel - It is intermediary between Model and the View. It transforms data from Model into format that is easily consumable by View. It also exposes commands and methods that the View can bind to for handling user interactions. It does not have direct knowledge of View.
- Data Binding: MVVM frameworks provide mechanism for automatic data binding between ViewModel and View. So, changes in ViewModel will automatically update the View, and vice versa.So this makes,it an Reactive paradigm.
- The components can be tested seperately because Model,View,ViewModel are seperate components.

2.3 Single-Page Web Applications (SPAs)

SPAs have a single HTML page that serves as entry point for application. The content of this page is dynamically updated based on user interactions. In SPA the content is dynamically loaded and updated as user interacts with application, eliminating need for full-page reloads.Instead of loading separate HTML pages for different views for the application, SPAs fetch data from server and update the existing page dynamically. This is done by AJAX (Asynchronous JavaScript and XML)
This process is asynchornous because data is fetched in background and the UI is updated in foreground automatically.This makes it an reactive paradigm.

2.4 Features

- Document Object Model :

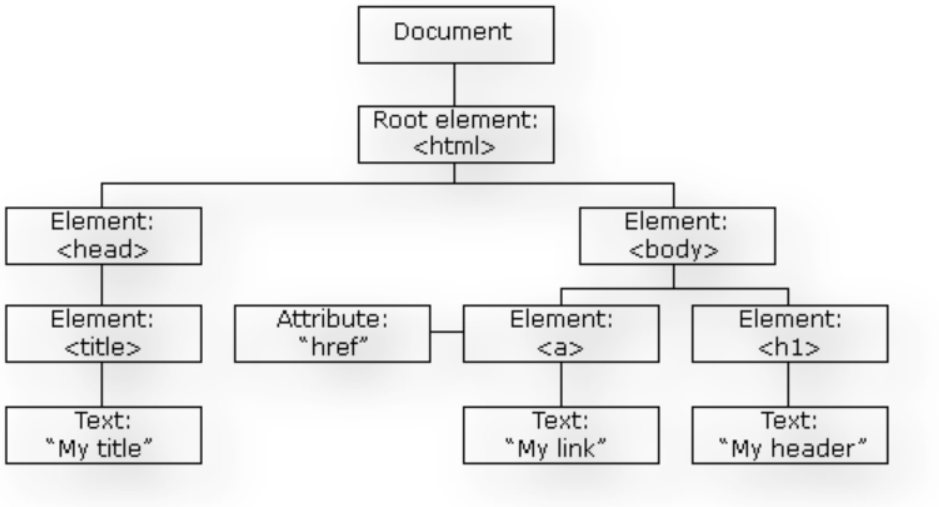


Figure 2: Document Object Model

To update the DOM angular uses Change detection.So,the model is checked for changes,so if there is a change the parts which have been modified are updated instead of updating the full model.

- Two-way binding :
  In Angular web application framework, two-way data binding is used . So when a change is made in the model state automatically it is reflected in the corresponding UI elements, and vice versa. It gives seamless connection between the Document Object Model (DOM) and the model data through the controller
- Testing :
  Angular primarily uses Jasmine testing framework for testing. Jasmine provides a behavior-driven development (BDD) syntax and structure for writing tests.

2.5 Architecture

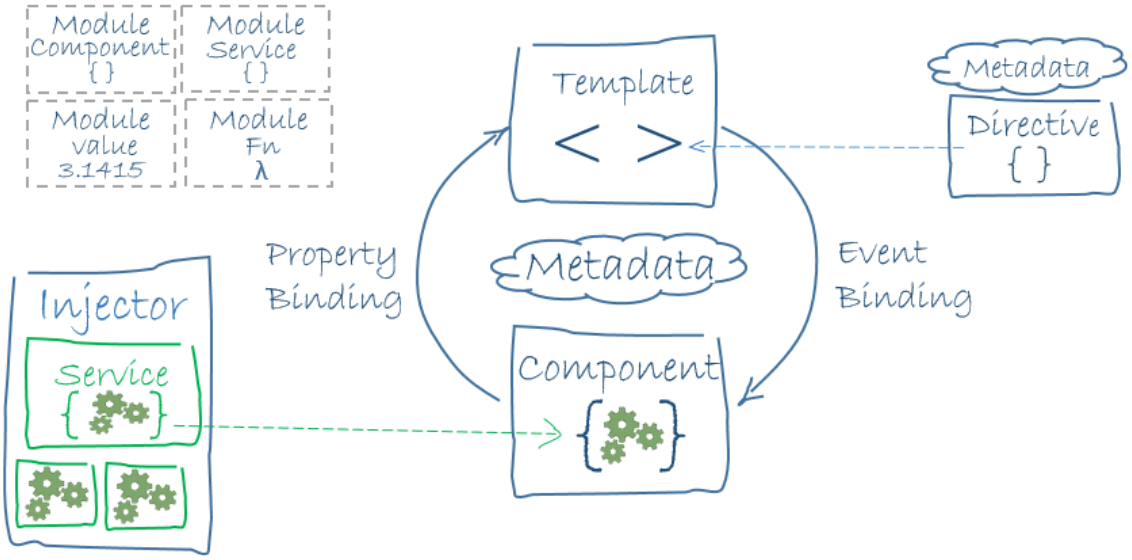


Figure 3: Architecture

- Module - An Angular module is a container for organizing and packaging components, directives, pipes, and services related to specific a feature. Modules are used for organizing codebase and facilitate modularity.Every angular application contains minimuma one module which is root module for small systems.Root module is bootstrapped when application is started.
  Ngmodule in angular:
  1)declarations: array of components, directives, and pipes that belong to the module.
  2)imports: modules whose exported components, directives, and pipes are needed by the module
  3)exports: Components, directives, and pipes declared in module and can be used by other modules.
  4)providers: Services that are provided at the module level, making them

available throughout the module.
5)bootstrap: The main application view, which is typically the root component.

- Angular libraries - Angular library name begins with the @angular prefix.Angular libraries are distributed in Angular Package Format (APF).
- Components - component controls a view.
- Templates - A template is a HTML code that tells Angular how to render the component.
- Metadata - Metadata tells Angular how to process a class.
- Data binding - It is mechanism for coordinating parts of template with parts of a component. Add binding markup to the template HTML to tell Angular how to connect both sides.
- Directives - When Angular renders templates, it dynamically transforms the Document Object Model (DOM) based on the instructions provided by directives.
- Services - A service is a class with a narrow, well-defined purpose. It should do something specific and do it well.Example - data service,message bus,tax calculator.
- Dependency injection - Dependency injection is a mechanism for providing a class with its required dependencies, ensuring that a fully-formed instance is supplied. Services are often the dependencies, and dependency injection is utilized to providing new components with the necessary services.

2.6 Code

```
import { Component, OnInit } from '@angular/core'; import { Observable, of, range, from, fromEvent } from 'rxjs';
import { ajax } from 'rxjs/ajax';
import { filter, map, catchError } from 'rxjs/operators';
@Component({
   selector: 'app-root',
   templateUrl: './app.component.html',
   styleUrls: ['./app.component.css']
})
export class AppComponent implements OnInit {
   title = 'Reactive programming concept';
   numbers : number[] = [];
   val1 : number = 0;
   filteredNumbers : number[] = [];
   val2 : number = 0;
   processedNumbers : number[] = [];
   val3 : number = 0;
   apiMessage : string;
   counter : number = 0;
   ngOnInit() {
      // Observable stream of data Observable<number>
      // const numbers$ = of(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
      // const numbers$ = range(1,10);
      const numbers$ = from([1,2,3,4,5,6,7,8,9,10]);
      // observer
      const observer = {
         next: (num: number) => {this.numbers.push(num); this.val1 += num },
         error: (err: any) => console.log(err),
         complete: () => console.log("Observation completed")
      };
      numbers$.subscribe(observer);
      const filterFn = filter( (num : number) => num > 5 );
      const filteredNumbers = filterFn(numbers$);
      filteredNumbers.subscribe( (num : number) => {this.filteredNumbers.push(num); this.val2 += num } );
      const mapFn = map( (num : number) => num + num );
      const processedNumbers$ = numbers$.pipe(filterFn, mapFn);
      processedNumbers$.subscribe( (num : number) => {this.processedNumbers.push(num); this.val3 += num } );
      const api$ = ajax({
         url: 'https://httpbin.org/delay/1',
         method: 'POST',
         headers: {'Content-Type': 'application/text' },
         body: "Hello"
      });
      api$.subscribe(res => this.apiMessage = res.response.data );
      const clickEvent$ = fromEvent(document.getElementById('counter'), 'click');
      clickEvent$.subscribe( () => this.counter++ );
   }
}
```

Figure 4: src/app/app.component.ts

```
<h1>{{ title }}</h1>
<div>
   The summation of numbers ( <span *ngFor="let num of numbers"> {{ num }} </span> ) is {{ val1 }}
</div>
<div>
   The summation of filtered numbers ( <span *ngFor="let num of filteredNumbers"> {{ num }} </span> ) is {{ val2 }}
</div>
<div>
   The summation of processed numbers ( <span *ngFor="let num of processedNumbers"> {{ num }} </span> ) is {{ val3 }}
</div>
<div>
   The response from the API is <em>{{ apiMessage }}</em> </div>
<div>
   <a id="counter" href="#">Click here</a> to increment the counter value. The current counter value is {{ counter }}
<div>
```

Figure 5: src/app/app.component.html

Output :

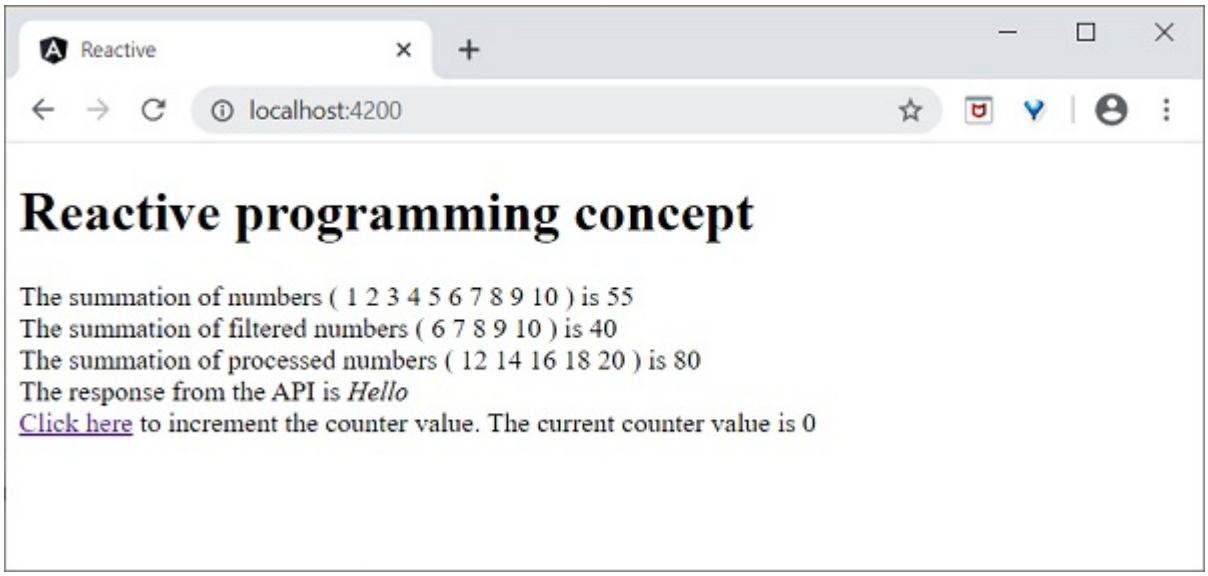


Figure 6: Before clicking 'click here'

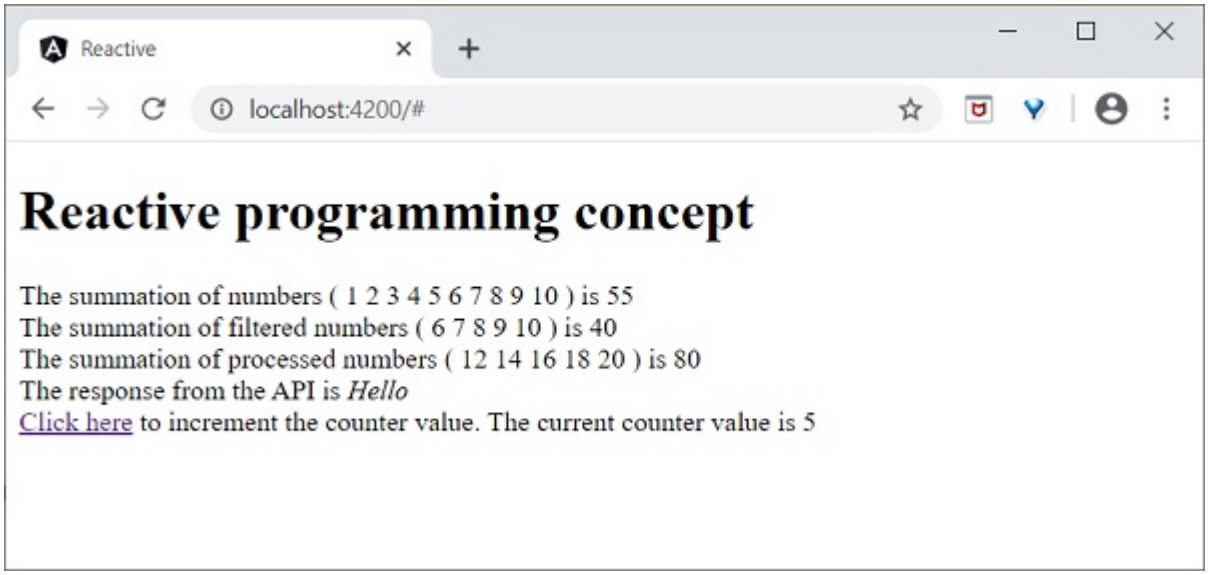


Figure 7: After clicking 'click here' 5 times

2.7 Advantages

- Data binding
- Dependency injection
- Browser Compatability
- Custom components(Users can implement their own functions and can be reused)
- Testing

2.8 Advantages

- Limited SEO
- Steep learning-curve(directives,modules,deocrators need to be leanred)
- Complex directives

2.9 Applications using Angular

- Youtube for PS3
- Paypal
- Linkedin(Mobile version)
- Google workspace(Gmail,G-drive)

3 Paradigm 2: Aspect-Oriented

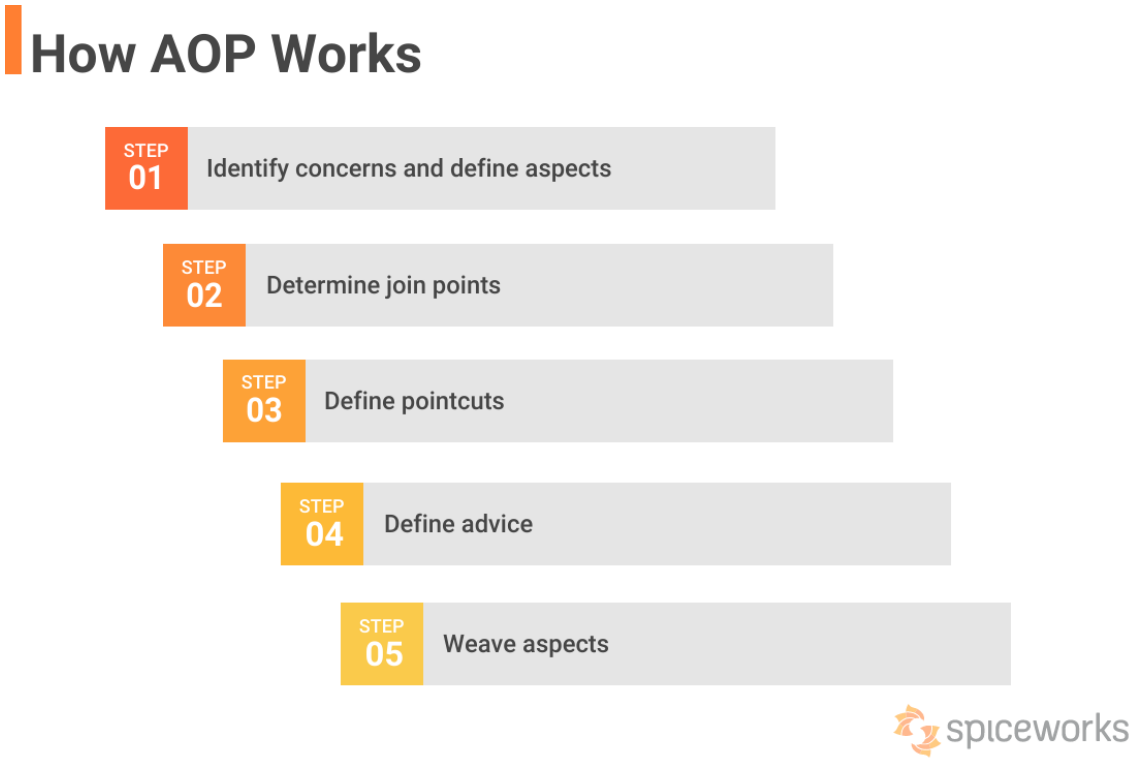
3.1 Introduction

- Aspect-oriented programming was developed by Gregor Kickzales and Xerox PARC in 2001.
- It was developed based on subject-oriented programming,adaptive programming.
- The aim of aspect-oriented programming is increasing the usage of modularity(modularisation).It is a programming paradigm that aims to modularize crosscutting concerns, such as logging, transaction management security,and error hadlling by separating them from main business logic
- It allows developers to separate concerns into different aspects rather than mixing them all in one module
- So for example in a banking system Security is a cross-cutting concern,so it should be applied in many part of the application so here we define the functionality in a code instead of implementing in every method and is called when needed by the application

3.2 Languages based on Aspect-Oriented Programming

- AspectJ
- Django
- java(Spring framework)
- C# (Post sharp)
- Python(spyct and aspectlib)
- Ruby(AspectR)
- Javascript(aspect.js provides basic aop features)

3.3 Working of Aspect-Oriented Programming



- Identifying concerns - The responsibilites of program need to be found first . In order processing payment processing , order validation are concerns.
- Defining aspects - aspect is a modular unit of code that encapsulates a specific behaviour. In the above example payment processing,order validation each can be an aspect.
- Determine join points - Point where aspect is applied. So when an order is placed the appropriate actions need to be taken like payment processing , order validation so this is the aspect.
- Define pointcuts - set of jointpoints where aspect is applied.
- Advice - additional code that is applied to existing model.in this case code for logging.
- Defining advice - behaviour an aspect provides at an joint point.
- Weaving aspects - applying aspects on program execution
- Execution - The behavior the aspects provide will be triggered at the appropriate join points.

3.4 Example - Program calculation 'click here' in a website

- Joint point is click here method which need to be called upon action
- The cross cutting point is error handling and logging as logging is used to see the flow of execution or to record errors.
- to apply aspect on join point weaving is used which is used to add the aspect code to original code.
- After woven the aspect will be executed when the join point i.e, after clicking 'click here', in the code is reached.

3.5 Places AOP is used

- Web applications - Used in web applications for transaction management,logging,security concerns in banking(authentication and authorization).
- Mobile applications - Device compatibility, data synchronization, and user engagement.
- IoT- fault tolerence,data processing
- Finance sector - auditing ,transaction management,compliance
- Healthcare - privacy of reports of patients.
- Gaming - optimize game rendering and animation.

4 Language for Paradigm 2: Django

4.1 Introduction

- Django is a python-based web framework that was created in 2003 and maintained by Django Softwre Foundation(DSF)
- Aim of the project was reusability and pluggability of components.
- It offers tools for data validation,caching, logging, pagination, authentication, and static file management in web apps.
- Django follows DRY(Dont Repeat Yourself) principle.
- Django provides administrative create, read, update and delete interface which is generated dynamically through introspection and configured via admin models.

4.2 How Django is Aspect-Oriented paradigm?

Django is aspect-oriented by using the futures of Middleware,Decorators and Signls.

- Middleware - Middleware components are used to process requests and responses globally before they reach the view or after the view has processed the request. Middleware functions are applied in a chain-like fashion, so now cross-cutting concerns such as authentication, logging, or caching are performed.
- Decorators - Decorators are metaprogramming and are used to implement aspects of AOP.
  Example: `login_required` decorator is a cross-cutting concern that ensures that a user is authenticated before accessing a particular view.
- Signals - Django signals are used to allow decoupled applications to get notified when certain actions occur elsewhere in the application. By connecting signal handlers to specific signals, the code can be executed in response to certain events without directly coupling the code to the code that triggers the event.

4.3 Security

1)Django templates protect against Cross site scripting (XSS).
2)Django protects against Sql injection attacks
3)Django uses the Host header provided by the client to construct URLs
4)Cross site request forgery (CSRF) is protected using django

4.4 Advantages of Django

- User authentication, Content administration, Site maps.
- django is a lightweight and standalone web server for development and testing
- Django supports middleware that can intervene at various stages of request processing and can be used to carry out custom functions
- Security
- Scalability - high traffic demand can be met.
- Verstaile - content management systems to scientific computing.

4.5 Applications using Django

- Social Media - Instagram,Pinterest,Youtube
- Browsers - Firefox
- NASA
- Bitbucket
- Cloud storages - Dropbox

4.6 Code

```python
from django.contrib.auth import authenticate, login
from django.contrib.auth.models import User
from django.http import JsonResponse
from django.views.decorators.csrf import csrf_exempt
import json

@csrf_exempt
def custom_authenticate(request):
    if request.method == 'POST':
        try:
            data = json.loads(request.body)
            username = data.get('username')
            password = data.get('password')

            user = custom_auth(username, password)

            if user is not None:
                login(request, user)
                return JsonResponse({'message': 'Authentication successful'})
            else:
                return JsonResponse({'message': 'Authentication failed'}, status=401)

        except json.JSONDecodeError:
            return JsonResponse({'message': 'Invalid JSON data'}, status=400)

    return JsonResponse({'message': 'Invalid request method'}, status=400)


def custom_auth(username, password):
    try:
        user = User.objects.get(username=username)
        if user.check_password(password):
            return user
    except User.DoesNotExist:
        pass

    return None
```

5 Analysis

5.1 Reactive paradigm and Angular

Strengths of reactive paradigm :

- Asynchronous Programming
- Responsive User Interfaces
- Event-Driven Architecture
- Backpressure Handling
- Concurrency and Parallelism
- Scalability

Strengths of Angular :

- Two-way Data Binding
- Dependency Injection
- MVVM (Model-View-ViewModel) Architecture
- Mobile Support
- Modularity

Weakness of reactive paradigm :

- Learning Curve
- Complexity(multiple asynchronous operations and complex data flows)
- Limited Ecosystem Support
- Potential for Overhead
- Backpressure Handling Complexity

Weakness of Angular :

- Steep Learning Curve
- Performance Overhead
- Verbosity

5.2 Aspect-Oriented paradigm and Django

Strengths of Aspect-Oriented paradigm

- Modularity and Separation of Concerns
- Cross-Cutting Concerns Management
- Aspect Reusability
- Cross-Cutting Concerns Encapsulation

Strengths of Django

- Model-View-Controller (MVC) Architecture
- URL Routing and Views
- Middleware Support
- DRY(don't repeat yourself)

Weakness of Aspect-Oriented paradigm

- Steep Learning Curve
- Potential Performance Overhead
- Potential for Code Scattering

Weakness of Django

- Heavyweight for Small Projects
- Asynchronous Support (not as powerfull as in other languages)

6 Comparison

- The aim of reactive programming is to handle asynchronous and event-driven systems while the aim of aspect-oriented programming is modularizing cross-cutting concerns.
- The main concept of reactive programming is observer and observable state(producer and consumer) whereas in aop it is aspects.
- Aop doesnt concern about asyncornous properties.
- AOP is used mainly for logging,authentication and authorization whereas reactive is used for responsiveness and scalability
- Both languages have a steep learning curve
- Although reactive paradigm mainly foucses on asynchronous property but it also uses modularity for data streams.
- Both paradigm use declarative approach meaning expressing the desired outcome without specifying step-by-step procedure for achieving it.
- AOP supports dynamic adaptation by allowing aspects to be woven into code during runtime, providing flexibility in changing behavior of a system without modifying code while reactive systems use dynamic adaptation by reacting to changes in the environment or data streams, adjusting their behavior in real-time.Django implements using middleware while angular implements in client side making components and services to react to changes in the application state.
- AOP implements aspects related to event-driven architecture by capturing events and handling them in separate aspects whereas in reactice programmming components react to changes or events in system giving real-time updates.
- In Django event-driven architecture is implement through its support for signals whereas in angular it is implemented in client side like HTTP respones.
- Django implements cross-cutting concerns in backend(authentication, security, and database access) while angular implements it in front end(routing and HTTP requests).

7 Challenges Faced

- The main challenge I faced was understanding the architecture of angular.I addressed this by learning about the architecture from the official website of angular.
- I wasn't able to understand some terms relative to the context give like event-driven architecture,asynchornous,single-page web applications.So I first explored about them first.
- I didnt get code for django with aspect-oriented paradigm so I used chatgpt for that to address the issue and then I understood how modularity works in django.
- Some websites had contrasting information for same specification so I had to do more research on those.

8 Conclusion

So for server side django can be used based on aspect-oriented paradigm so that the admin can either autheticate the user or handle error in case and concern security issues by preventing from attacks like sql injection.In client side, Angular can be used based on reactive programming for scalability.Using angular helps having a responsive web application and a good user experience.Django helps encapsulating the code.Angular based on reactive paaradigm is mainly useful in developing real-time applications.Both django and angular are usefull when developing the code is reused again,so in that case modularity can be used.If i was needed to develop a website concerning security django would be preffered(like banking applications) whereas the purpose is streaming or gaming angular would be preffered.

9 References

- Paradigm 1: Reactive
  - https://www.baeldung.com/cs/reactive-programming
  - https://en.wikipedia.org/wiki/Reactive_programming
  - https://www.techtarget.com/searchapparchitecture/definition/reactive-programming
  - https://subscription.packtpub.com/book/web-development/9781786463388/1/ch01lvl1sec0/the-reactive-paradigm
- Language for Paradigm 1: Angular
  - https://www.simplilearn.com/tutorials/angular-tutorial/what-is-angul
  - https://www.tutorialspoint.com/angular8/angular8_reactive_programming.html
  - https://v2.angular.io/docs/ts/latest/guide/architecture.html
- Paradigm 2 : Aspect-Oriented
  - https://en.wikipedia.org/wiki/Aspect-oriented_programming#Implementation
  - https://www.spiceworks.com/tech/devops/articles/what-is-aop/
  - https://medium.com/hprog99/aspect-oriented-programming-b9a06ca256db
- Language for Paradigm 2 : Django
  - https://en.wikipedia.org/wiki/Django_(web_framework)
  - https://www.djangoproject.com/
  - https://www.tutorialspoint.com/python_web_development_libraries/python_web_development_libraries_django_framework.html
  - Chatgpt for code.Prompt : give an code in django which authenticates an user which is base on aspect oriented programming.