**Introduction**

*"Good [morning/afternoon/evening] everyone! Today, I’ll be talking about Quarkus, functional programming there advantages and Angular frameworks: So Let’s start by understanding what is Quarkus and why it is used ..!"*

**What is Quarkus?**

Quarkus is a framework for building Java applications, just like Spring Boot. However, it is designed to be lightweight, super fast, and it is specifically optimized for cloud environments which is majorly used in todays development.

So there is a line in official definition of Quakrus which you can see on your screens i.e . “Quarkus is a Kubernetes-native Java framework” tailored for GraalVM " which basically Means Quarkus helps us to make a project which is easy to be loaded in Kubernetes which is a container that Run and deploy our application very fast.

**Why we should Quarkus Instead of Spring Boot?**

**Faster Startup** – Quarkus starts in milliseconds, while Spring Boot takes seconds. This is important for cloud applications where speed matters.

And It uses much less RAM compared to Spring Boot, making it cost-effective in cloud environments.

**It also support GraalVM Native Compilation** –which means Quarkus apps can be compiled into native executables,which makes them ultra-fast and even more memory-efficient.

**Developer-Friendly** – It has a live reload feature, meaning developers can see changes instantly without restarting the application. So many times when we run our application then due to some silly mistakes we can see the different output as expected So in that case ideally we will be correcting that things and then restarting our application But in Quarkus we do not have to restart our application, we can simply do our changes and it will reload automatically. until there are some major changes in our application Like if we add some dependency then we need to recompile the application manualy only.

**Slide 1: What is Functional Programming?**"Functional programming is a programming paradigm or we can say a methodology of writing a program that focuses on immutable data i.e the data that does not change and pure functions. So In functional programming Instead of modifying data, we create new versions of it, which makes the program more predictable.

"Unlike imperative programming, which focuses on writing step-by-step instructions, and object-oriented programming, which organizes code around objects, functional programming focuses on **what** needs to be done rather than **how** to do it." SO it basically focus on the logical part rather then the structural part.

**Slide 1: Key Concepts-** So there are some key concepts in functional programming which we need to understand before going any further.

**🔹 Pure Functions  
"**A pure function is nothing but a simple function which always returns the same output for the same input and has no side effects. For example, we have Math.sqrt() function which will always give a specific value for a specific input Like if we pass 9 then it will always return 3—no matter when or where it’s called**."**

**🔹 Immutability**"That means Once data is created, it cannot be modified. In Java, we achieve this using the final keyword, and in JavaScript, we use const.So in functional programming Instead of modifying existing data, we create new versions of it, which prevents unintended changes."

**🔹 First-Class Functions**"In functional programming, functions are treated like any other value. You can store them in variables, pass them as arguments, and even return them from other functions. Which ultimately makes code more flexible and reusable."

**Stored in variable**

A screenshot of a computer code

AI-generated content may be incorrect.

**Passed as Argument**

A screenshot of a computer code

AI-generated content may be incorrect.

**Returning Functions from Other Functions**

A screenshot of a computer program

AI-generated content may be incorrect.

**🔹 Higher-Order Functions**"These are functions that either take other functions as parameters or return them as results. A common example is it map(), which basically applies a function to each value in an array."

So by using all these key concepts functional programming offer good number of benefits like:

**Slide 1: Benefits of Functional Programming**

1. **Code Clarity & Readability –** Since functional code focuses on *what* rather than *how*, it’s easier to understand.
2. **Easier Testing & Debugging** – With pure functions, you don’t have to worry about hidden state changes.
3. **Better Concurrency & Parallelism –** Since functional programs avoid shared state, they work well in multi-threaded environments.
4. **Maintainability & Reusability –** By reducing dependencies between different parts of the code, functional programming makes it easier to modify and extend.
5. **Declarative Style –** Instead of writing loops, we write code in a more expressive and readable way, by using functions like map, filter, and reduce.

**Now moving on to next slide……..**

**Slide 2: Lambda Expressions and Stream API**

Modern Java programming has evolved with functional programming features. Lambda expressions and Stream API are two powerful additions which were introduced in Java 8.

So we will be digging deep into these in upcoming slides.

**Slide 3: What are Lambda Expressions?**

*"First, let’s talk about Lambda Expressions. So In simple words, a lambda is an* ***anonymous function****—or a function without a name or modifier. Which basically help us write shorter, cleaner code."*

*So this is the* ***syntax*** *of lambda which basically consists of 2 main parts the Parameter and expression …….and after that there is* ***example*** *showing how lambda works. So this lambda expression simply take 2 values as input and return the addition of them.*

*SO* instead of writing the entire method with Name , return type and all we can use lambda which also does the same task in much lesser lines.

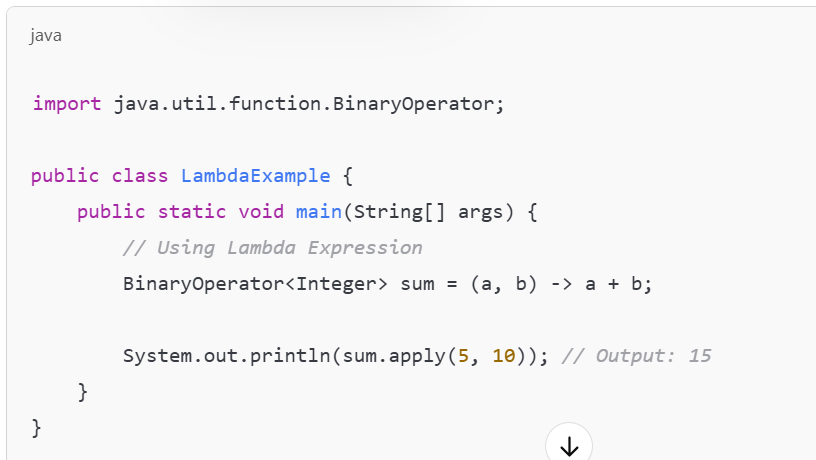
*Note that here we also does not have to write the return statement which further decreases our efforts.*

***Without Lambda***

***A screenshot of a computer program

AI-generated content may be incorrect.***

***With Lambda***



**Explanation:**

* **Without Lambda:** We have define an anonymous class that implements BinaryOperator<Integer>, overriding its apply() method.
* **With Lambda:** We replace the entire anonymous class with a simple expression: (a, b) -> a + b, making the code more concise and readable.

**Now Moving to next slide….**

**Slide 4: Why Use Lambda Expressions?**

*"Lambdas bring several advantages to our codebase. Let’s go through them quickly:"*

1. **Concise Code** – which simply Means Lambda Reduces boilerplate.
2. **Functional Programming** – Which Enables passing Functions as an argument as I explained above. In **functional programming**, we treat functions as **first-class citizens**, meaning we can pass them as arguments to other methods just like we pass objects or primitive values. This allows us to write more flexible and reusable code.
3. **Improved Readability** – The code will be Less cluttered.
4. **Parallel Processing** – Parallel processing allows operations to be executed concurrently, which utilizing **multiple CPU cores** for better performance. This is achieved using **parallel streams**, which divide the data into multiple chunks and process them in parallel.

* **So 1st**  The stream is divided into multiple substreams.
* **Then** Each substream is processed independently on different threads.
* **And after that** all the streams are combined into a final result

**SO**,

*"In short, lambdas help us write cleaner and more maintainable code."*

**Next slide…..**

**Slide 5: Predefined Functional Interfaces**

*"Java provides built-in functional interfaces to work with Lambdas. Some of the most commonly used ones include:"*

1. **Supplier<T>** – Supplier is one which Supplies a fix values without any input.
2. **Predicate<T>** – Return true or false based on the Tests conditions.
3. **Consumer<T>** – Consumer does not return anything it simply takes values as input and perform some task.
4. **Function<T, R>** – Functions has 2 generic inputs, one for input type and another one for return type. So it basically takes a value as input performs some operation and return the output.

**Next slide…**

**Slide 6**: These are some examples of each functional interface.

Here supplier is used to supply the List<String>

And predicate is used for checking the string length is > 5 or not

Comsumer simply taking string as input and printing it

and

*Function takes string as input and return the length of that string as output:"*

**Slide 7: What is Stream API?**

1. *"Now, let’s shift our focus to* ***Stream API****. Streams help us process data in a functional and declarative way. Means* Instead of writing loops, we can write code in a more expressive and readable way, by using functions like map, filter, and reduce etc.

*And some of the*

**Key Features of Streams:**

1. **Sequence of Elements** – We can create a sequence of elements with collections, arrays, Or any I/O channels by using different methods like .stream or .of
2. **Pipeline Processing** – which basically means Stream API Allows multiple operations in a row i.e Operations will occur in different stages on each element in row.
3. **Functional Style** – Means it Uses lambdas for cleaner code.
4. **Lazy Evaluation** – Lazy evaluation means that operations on a stream are **not executed immediately** but are **only performed when a terminal operation is invoked**.

*SO,*

*Instead of using loops, we can process collections easily with Stream API."*

**Now Moving on to next slide……**

**Slide 8: Why Use Stream API?**

Like I just mentioned in previous slide that there are multiple advantages of using Stream API over traditional way

*"Streams make our code more readable and efficient.And Here are some key points why we should use them:"*

1. **Readable Code** – Streams make complex operations clear i.e No need for nested loops. We can simply use stream which will do the same things in just few line .
2. **Parallel Processing** – We can Easily switch to parallel processing by using parallelStream to utilize multiple CPU cores for better performance.



1. **Data Filtering** – It Allows easy filtering of elements based on conditions.

*"For example, filtering a list of names that start with 'A'.*

*TO achieve this we can simply convert the list to stream and then use .filter method of stream to filter the names based on some condition. Which will be much cleaner than writing a traditional for-loop!"*

A close-up of a computer code

AI-generated content may be incorrect.

1. **Data Transformation** –Which means it Convert one data type to another smoothly.

**Slide 9: How to create Stream API**

*"Streams consist of three main components:"*

1. **Stream Creation** – Streams can be created from collections, arrays, or specific methods. SO while using stream API the 1st part is to create a stream.After that we have apply some intermediate operation according to our use case.
2. **Intermediate Operations** – It Transform or filter the data and returns a new stream and some Eg of intermediate operations are filter, map so these are **lazy** and do not execute immediately.

These operations **don’t modify the original data** but create a new transformed stream.

Like I sad

* **filter()** → Is one Which Selects elements based on a condition
* **map()** → Is used to Transforms elements, it is basically used to traverse the stream of data
* **sorted()** → which Sorts elements

1. **Terminal Operations** –It Produces a final result which basically Consumes the stream and no other operation can be done after the terminal operation.

**And** Eg of terminal operations are forEach(), collect(), reduce(), etc. and here forEach is used

 **forEach()** → Is used to Performs an action on each element

 **collect()** → Gathers elements into a list, set, etc.

 **count()** → simply Counts the elements

 **reduce()** → It basically Performs aggregation means it return less number of data as compared to original data based on some conditions.

So to conclude this in one line we can

*"Think of it like an assembly line where person standing on 1st place have a drawing paper and it is passed throw the line and each student is adding some things to the drawing before the final result."*

*The same operations are performed in Stream also.*

*Now Moving on to next slide……*

**Slide 10: Advantages of Stream API**

*"Streams offer several advantages over traditional loops:"*

1. **Declarative Style –** Which meansDevelopers can focus on "what" needs to be done rather than "how" to do it. Or In simple terms we can say, Stream API allows developers to concentrate on the logical operations rather than the structural details of iteration and data processing. Which can increase the productivity by 2 times.
2. **Less Boilerplate** – By using stream we can Reduces unnecessary code by 68%.
3. **Parallel Execution** – Can Boosts performance effortlessly By 8 times on 8 core processors.
4. **Method References** – which Allows cleaner syntax……. like String::length.
5. **Pipeline Processing** – Means it Chain multiple operations together. Which ultimately decrease the boiler plate code

*"This makes Streams a game-changer when working with collections."*

**Slide 11: Exception Handling in Streams**

*"* *The* ***Stream API*** *in Java does not allow checked exceptions to be thrown directly inside lambda expressions. This creates challenges when dealing with methods that may throw exceptions, such as* ***file exception, database exception, or parsing****. So To handle exceptions in streams, we need to use different workarounds.:"*

1. **Using try-catch inside lambdas-** A simple way to handle exceptions is by wrapping the code inside a try-catch block within the lambda expression.

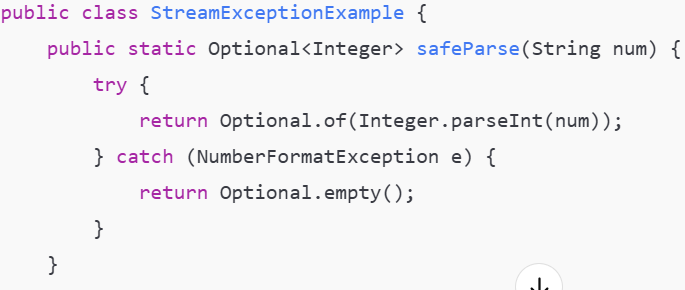
A screenshot of a computer code

AI-generated content may be incorrect.

💡 **How it works?**

* Tries to parse each string into an integer.
* If parsing fails (e.g., for "abc"), it catches the exception and returns 0.

1. **Returning Optional to manage errors -** Instead of returning fallback values like we return using try catch block, we can use Optional to decrease the possibility of failure.



💡 **Why use Optional?**

* Helps avoid null values.
* Ensures safer handling of errors.

1. **Using a custom exception wrapper -** If we need to distinguish between successful and failed operations, we can create a custom wrapper.

*"For example, handling exceptions inside map():"*

stream.map(element -> {

try { return processElement(element); }

catch (Exception e) { return fallbackValue; }

});

*"This ensures smooth execution even when errors occur."*

💡 **Benefits:**

* We explicitly separate **successful** and **failed** cases.
* Useful when handling **multiple error types**.

***Next Slide………***

**Slide 15:** Ok…. So Now, we'll be exploring a powerful framework for building dynamic and engaging web applications and that is Angular.

*Next Slide Please……….*

**Slide 16:** So, what exactly is Angular? At its core, it's a robust and well-structured framework designed for building modern web applications. And It's not just a library but it provides a complete set of tools to help developers create scalable and maintainable applications.

1. As written in the 1st point…..It is developed and maintained by Google which ensures continuous development and a large, active community for support.
2. And A key characteristic of Angular is its focus on building **single-page applications**, which offer a smooth and responsive user experience by loading a single HTML page that is index.html and dynamically updating content.
3. Furthermore, Angular leverages TypeScript, which adds static typing to JavaScript instead of dynamic, which leading to more robust and easier-to-maintain code.
4. And Finally, its component-based architecture promotes modularity and reusability.

Now Moving on to the next Slide…………..

Here we will be discussing the benefits of using angular.

**Slide 17:** So….There are several reasons to choose Angular for your web development projects. Number 1 is Its **component-based architecture**

Its **component-based architecture** is a significant advantage, which allow you to break down your application into smaller, independent, and reusable pieces just like we have microservice concept in backend in which we can use our microservice in any other service.

This makes development more organized, testing easier, and long-term maintenance more manageable.

Another benefit is the use of **TypeScript** which brings the benefits of static typing, catching errors early in the development process and improving code readability and by saying static tying I mean that in .TS file we have to declare the type of variable we are defining unlike JavaScript where its not mandatory to define the type. So this is another + point of using angular.

Apart from this Angular's powerful CLI is a game-changer, which automate many common development tasks like creating the project and its files which ultimately boosts productivity.

And It also comes with a **rich set** of built-in features, which handls crucial aspects like navigation between different pages, managing application data, dealing with user input through forms, and communicating with backend services.

It also have **Strong community** support which ensures that you'll find many resources and support when you need it, and the framework's design makes it highly scalable for even the most complex applications.

So these all are the key benefits of using angular framework.

*Now moving on to the next Slide……….*

**Slide 18:** So these all are the Key Concepts in Angular which one should know to understand this framework:-

I will be briefing about each one of it one by one so that you all understand it better.

Starting with the 1st one that is Component.

**Components:** At the heart of every Angular application lies the concept of components. Think of them as the fundamental building blocks of your user interface. Each component is responsible for a specific part of the UI and has its own structure (the HTML template), its own styling (the CSS), and its own behavior (the JavaScript or TypeScript code). This encapsulation makes components highly reusable –and you can use them in different parts of your application or even in other projects. Furthermore, you can compose complex UIs by nesting and combining smaller components.

Another one is Modules

**Modules:** As your application grows, you'll need a way to organize your components and other related code. This is where modules come in picture. Modules are essentially containers that group together components, directives, services, and other elements that are related to a specific feature or functionality of your application. They also provide a logical structure to your codebase, making it easier to understand and maintain. Additionally, Angular modules support lazy loading, which means that certain parts of your application are only loaded when they are actually needed, leading to faster initial load times and a improved performance.

Next is Templates

**Templates:** The structure and presentation of each component's UI are defined in its template, which is nothing but HTML extended with Angular-specific syntax. This syntax allows for dynamic behavior. For example, you can display data from your component's code directly in the template using double curly braces – this is called interpolation.

Another is Directives which are powerful tools that let you manipulate the structure of the DOM based on certain conditions or iterate over collections of data. And some Common examples include \*ngIf to conditionally display elements and \*ngFor to loop through arrays,….. these all Directives can be used in our HTML page and after angular 17 we can use @if or @for instead of \*ngif.

And one more Important concept is, *event binding (e.g., (click))* which allows you to respond to user interactions like clicks, mouse movements, and form submissions, which triggers actions in your component's logic.

Now coming to service and dependency injection

**Service and dependency injections:** To keep your components concise and neat, it's crucial to extract reusable logic and data handling into services. Services are nothing but simply classes that encapsulate specific functionalities, such as fetching data from an API, performing calculations, or managing application state. And

Angular use a powerful concept called Dependency Injection to make these services available to the components that need them. Instead of components creating their own instances of services, Angular's DI framework provides the necessary dependencies. This promotes loose coupling, making your code more testable and maintainable.

And lastly is

**Routing:** SO In a single-page application, you need a way to handle navigation between different parts of the application without full page reloads. This is the role of Angular's Router. You configure the router by defining a set of routes, where each route maps a specific URL path to a particular component. When the user navigates to a certain URL, the Router displays the corresponding component, providing a seamless user experience.

***Moving to Next Slide…***

**Slide 19:** Here we will be discussing about the Angular CLI

The Angular CLI is an valuable tool that significantly streamlines the development process. It's a command-line interface that allows you to perform a wide range of tasks quickly and efficiently.

With some simple commands, you can create new Angular projects with a pre-configured structure, generate boilerplate code for components, services, and other Angular artifacts,

It also run a development server to preview your application, and build your application for deployment to a production environment.

This angular CLI enforces best practices to be used in project and greatly enhances developer productivity.

**Slide 20:** Steps to install Angular

So these are the basic steps to install Angular in your system.

**Slide 21:** These are the real world Angular applications

**Slide 22: Conclusion**

*So yaaa this is all from my side if you have any doubts please feel free to ask.*

*"Thank you”*

**1. Introduction:**

*"Now, I will walk you through the overall flow of my Mobile CRUD application built using Quarkus.Here I'll explain you the entry point, the processing flow, how I have used webSocket, and how we manage exceptions globally."*

**2. Entry Point:**

*"So The entry point of the application is the REST API exposed in the MobileResource class. This resource is mapped with the base path /mobiles and here base path is localhost8080. Whenever a client, like a frontend application or Postman, sends a request, it first hits the corresponding endpoint in MobileResource."*

**3. Request Handling and Flow:**

*"Once the request comes to the MobileResource layer, it handles different operations like creating a mobile, fetching mobile details, updating, or deleting a mobile record based on different annotations like PUT, POST, DELETE.*

*And Inside the resource class, we seperate the main business logic to a separate service layer called MobileService."*

*"The service layer is responsible for handling the core operations, such as validating the request data, interacting with the database, and applying any business rules."*

*"The service layer further calls the repository layer, which uses PanacheRepository to interact with the database. The repository directly performs CRUD operations on the Mobile entity."*

*"The Mobile entity is a JPA entity mapped to a database table, where each record stores fields like mobile name, brand, price, RAM, and external storage."*

**4. WebSocket Usage:**

*"Apart from REST APIs, I have also uses a WebSocket endpoint.*

*The WebSocket server is implemented to provide real-time updates. For example, whenever a new mobile is added or an existing mobile is updated or deleted, the server sends real-time notifications to all connected clients. Such as Angular-UI"*

*"This WebSocket server is annotated with @ServerEndpoint, and it manages client sessions.*

*Whenever an event happens — like a new mobile creation — after saving the data in the database, we broadcast a message through WebSocket to inform all connected users without needing them to refresh manually."*

*"This improves user experience by keeping the frontend synced with the latest data."*

**5. Global Exception Handling:**

So when we make any application there can be N number of cases which can cause error in our application…… SO instead of handling each error separately which is a bad practice I have used a global exception handler which handles all the exceptions of entire application.

In which *Instead of showing raw server errors which is very difficult to understand, we catch exceptions centrally by implementing an ExceptionMapper." Interface.*

*And we have to annotate this class by @Provider which tell the Quarkus to bring all the exception to this particular class*

*"For example, if a requested mobile is not found or a database error occurs, the global exception handler captures it and returns a clean, user-friendly error response with proper HTTP status codes."*

*"This ensures that the client always receives a predictable, readable error format and helps in debugging and logging."*

*"That's the overall flow of the application."*

**GraphQL**

***1. Introduction to GraphQL:***

*"AND Apart from normal REST APIs, my Quarkus application also supports GraphQL endpoint."*

*So before showing that 1st I will just quickly explain what is GraphQL.*

*"GraphQL is a query language for APIs that allows the client to specify exactly what data they need, and the server responds with only that data — nothing extra, nothing missing."*

*"This solves a common problem with REST APIs, where sometimes we over-fetch or under-fetch data."*

***2. How GraphQL Works in My Application:***

*"In my application, I have created a GraphQL endpoint using Quarkus extensions for GraphQL."*

*"Instead of hitting different REST APIs for different operations, with GraphQL, we expose a single endpoint — which is localhost:8080 /graphql — where clients can send their queries or mutations. Here for fetching any data from DB we have to use Query annotation and for any updation operation on DB we uses Mutation annotation.*

*"For example, if the client wants to fetch only mobile name and price, they can do this very easily in GraphQL unlike in Rest API where we have to create one DTO layer for that"*

***4. Demonstration in Postman:***

*"So now I will just quickly show you how we can use the GraphQL endpoint ."*

*"Instead of selecting 'GET' or 'POST' and entering a URL like we do for REST, we select the GraphQL option in Postman."*

*"We write the query or mutation in the body and hit send — and Postman will show the exact result."*

***ANGULAR***

**1. Introduction:**

"Now I'll explain the flow of the Angular frontend of the Mobile CRUD application — starting from the entry point, the important configuration files, and how the entire application works."

**2. Entry Point:**

"The entry point of the Angular application is the main.ts file.

This file is responsible for bootstrapping the main Angular module, which is the AppModule."

"The AppModule is like the heart of the application. It declares all components, imports required modules, and configures the basic structure of the app."

**3. index.html:**

"After bootstrapping, the main HTML page is index.html.

This is a very simple file that contains a basic structure with a <body> tag, and inside it, there is only one Angular component tag called <app-root>."

"The <app-root> is connected to the AppComponent, which is the first component loaded when the application starts."

**4. AppComponent and Routing:**

"The AppComponent acts like a shell or layout. Inside it, we usually have a <router-outlet> where all other pages or components are loaded based on user navigation."

And here I have fixed the header and footer component with <router-outlet> so that these 2 component is always present with every other component that is rendered.

" And here We manage navigation between different pages using the Angular Router.

The routes are defined in the app.routes.ts, where different paths are mapped to their respective components."

"For example, we have different routes for Admin, Mobile, Home Component."

**5. Global Styling (styles.scss):**

"The styles.scss file is used for global styling across the entire application.

Any CSS or SCSS code written here is available to all components unless they override it with their own styles."

"For example, things like fonts, background color, button styles, or global variables are generally placed inside styles.scss."

Like here I have give the background colour as black for the entire application instead of giving in each component.

**6. Important Configuration Files:**

"Now coming to some important configuration files:"

angular.json

"This file is the main workspace configuration for Angular.

It defines how the application should be built, where the source code is located, how assets like images and stylesheets should be bundled, and also allows us to customize build options."

package.json

"This file manages all the dependencies and libraries that our project uses, like Angular itself, Bootstrap for styling, or any other third-party packages.

It is very much similar to POM.XML file that we have in Quakrus or springboot

tsconfig.json

"This is the TypeScript configuration file. It defines how the TypeScript compiler should behave — like which files to include, what target JavaScript version to compile to, and setting strictness rules for code checking."

And we don’t have to worry about all these files because these files are automatically created while creating the angular application and are handled by Angular.

The files on which we have to mainly work and focus are components and services that we have to create using angular CLI or we can also created them manually.

And in each component there are 4 files that are created by Angular that are HTML, SCSS, Spec.TS, and .TS

Here HTML is used to provide the structure to the component and SCSS is used to give the styling to that Structure and in TS file we have to write the logical part such as behaviours of some elements. And lastly in SPEC.TS file we have to write the test related things.

Apart from this we also have to work with service layer which contains all the logic which is used to Hit the API of backend so that data can be fetched from Database, and it also contain the logic that we are using repeatedly in our application

 "That's the complete flow of the frontend application. If you have any doubts please feel free to ask!"

**7. How the Entire Application Works Together:**

"To summarize the flow:"

First, when we run ng serve, Angular compiles the application using the instructions from angular.json and tsconfig.json.

The entry point is main.ts, which bootstraps the AppModule.

The index.html loads the AppComponent inside the <app-root> tag.

The AppComponent contains a <router-outlet>, and depending on the URL, Angular loads the appropriate component using routing.

Components interact with backend REST APIs (like our Quarkus MobileResource) to perform operations.

Global styling is handled via styles.scss, and component-specific styling can be applied inside each component's .scss file.

The package.json keeps track of project dependencies, making it easy to install and manage libraries."