{c}odeBetter
Rise above the rest
401, Shekhar Central, Palasia Square, Indore, MP - 452001
www.codebetter.in
+91 88230 75444
+91 9993928766

Flutter APP Development

Section 1 – Dart Programming

- **Operators, var, let, const, type Conversion**
- **Control Statements (If else, switch,break,default)**
- **Control Statements (loop's)**
- **Intro to Function & types of function**
- **User-define Function Local & global variables**
- **Anonymous functions,Arrow Function.**
- **List, Map, Set**

*1. Write a program that takes two numbers as input and displays their sum using the addition operator.
*2. Write a program that swaps the values of two variable without using a temporary variable.
*3. Write a program that checks if a given number is even or odd using the modulus operator (%).
*4. Write a program that converts a string to a number and displays the result. For example, convert the string "42" to the number 42.
*5. Write a program that compares two strings and checks if they are equal using the equality (==) operator.
*6. Write a program that checks if a given number is positive, negative, or zero using an if-else statements.
*7. Write a program that prompts the user to enter a grade (A, B, C, D, or F) and displays a corresponding message using a switch statement.
*8. Write a program that finds the maximum of three numbers entered by the user using nested if-else statements.
*9. Write a program that prompts the user to enter a number and displays whether it is prime or not using a loop and if-else statements.
*10. Write a program that converts a given month number (1-12) into its corresponding name (January – December) using a switch statement.
*11. Write a program that prints the numbers from 1 to 10 using a for loop.
*12. Create a program that calculates the sum of all numbers from 1 to a given number entered by the user using a while loop.
*13. Write a program that prompts that user to enter a password and keeps asking until the correct password is entered using a do-while loop.
*14. Create a program that prints the multiplication table of a given number entered by the user using a for loop.
*15. Write a program that finds the factorial of a given number using a for loop.
*16. Write a Dart function that takes two integers as parameters and returns their sum.
*17. Write a Dart function that checks if a number is even or odd and returns a corresponding message.
*18. Create a Dart function that calculates the factorial of a given number using recursion.
*19. Write a Dart function that checks if a given number is prime and returns a Boolean value.
*20. Create a Dart function that generates a Fibonacci sequence up to a given number and returns a list.
*21. Write a Dart program that demonstrates the usage of a global variable.
*22. Write a Dart function that calculates the perimeter of a rectangle using local variables.

*23. Write a Dart function that calculates the factorial of a given number using a local variable.
*24. Write a Dart function that checks if a given number is prime using a local variable.
*25. Write a Dart program that uses an anonymous function to calculate the square of a given number.
*26. Write a Dart program that uses an anonymous function to filter even numbers from a list.
*27. Write a Dart program that uses an anonymous function to convert a list of string to uppercase.
*28. Write a Dart program that uses an arrow function to check if number is positive.
*29. Write a Dart program that uses an arrow function to reverse a string.
*30. Write a Dart program that creates a list of integers and prints the sum of all the numbers in the list.
*31. Create a Dart program that adds elements to a list using the **‘add’** method and prints the updated list.
*32. Create a Dart program that updates a value in a map using the key and prints the updated map.
*33. Create a Dart program that sorts a list of integers in ascending order and prints the sorted list.
*34. Create a Dart program that performs a union operation on two sets and prints the resulting set.

#1. Create a program that calculates the area of a rectangle given its length and width using multiplication.
#2. Create a program that converts a temperature in Celsius to Fahrenheit using the formula: Fahrenheit = (Celsius* 9/5) + 32.
#3. Create a program that calculates the average of three numbers using the division and addition operators.
#4. Create a program that calculates the area of circle given its radius using the mathematical constant pi.
#5. Create a program that calculates the factorial of a given number using a loop and the multiplication operator.
#6. Create a program that determines whether a year entered by the user is a leap year or not using an if-else statement.
#7. Create a program that simulates a simple calculator. Prompt the user to enter two numbers and an operator (+,-,*, /), then perform the corresponding operation using a switch statement.
#8. Create a program that checks if a given character is a vowel or a consonant using a switch statement.
#9. Create a program that generates a random number between 1 and 10 and allows the user to guess it. Display a message indicating if the guess is correct or not using an if-else statement.
#10. Create a program that simulates a simple quiz. Ask the user a multiple-choice question and validate the answer using if-else statements. Provide appropriate feedback based on the user’s response.
#11. Create a program that prints the Fibonacci sequence up to a given number entered by the user using a while loop.
#12. Write a program that prompts the user to enter a series of numbers and finds the maximum among them using a for loop.
#13. Create a program that checks if a given number is prime or not using a for loop.
#14. Write a program that generates a random number between 1 and 100 and allows the user to guess it. Provide appropriate feedback and keep asking until the correct guess is made using a do-while loop.
#15. Create a program that prints the ASCII values of all uppercase letters from A to Z using a for loop.
#16. Create a Dart function that calculates the area of a rectangle given its length and width, and returns the result.
#17. Write a Dart function that converts a temperature in Celsius to Fahrenheit and returns the result.
#18. Create a Dart function that checks if a given string is a palindrome and returns a Boolean value.
#19. Write a Dart function that calculates the sum of all numbers in a list and returns the result.
#20. Create a Dart function that counts the number of vowels in a given string and returns the count.
#21. Create a Dart program that shows the difference between local and global variable.
#22. Create a Dart program that demonstrates the usage of a local variable inside a loop.
#23. Create a Dart program that demonstrates the scope of local variable.
#24. Create a Dart program that demonstrates the shadowing of local variables.
#25. Create a Dart program that uses an anonymous function to find the sum of two numbers.
#26. Create a Dart program that uses an anonymous function to find the maximum number in a list.

#27. Create a Dart program that uses an arrow function to calculate the area of a rectangle.
#28. Create a Dart program that uses an arrow function to calculate the sum of a list of a numbers.
#29. Create a Dart program that uses an arrow function to calculate the square root of a number.
#30. Create a Dart program that creates a map representing a person with keys “name”, “age”, and “city”. Print the values of the map.
#31. Write a Dart program that removes an element from a list using the **‘remove’** method and prints the updated list.
#32. Write a Dart program that checks if an element exists in a set using the **‘contains’** method and prints the result.
#33. Write a Dart program that retrieves all keys from a map and prints them.
#34. Write a Dart program that creates a set of strings and prints the number of unique elements in the set.

Section 2 –

- **Object Oriented Programming**
- **Object,Class, Methods,Access specifiers**
- **Constructor, named constructor, this, static**
- **Inheritance - single, multilevel, hierarchical,Multiple**
- **Runtime Polymorphism, Method overriding, super keyword**
- **Abstraction, Abstract Class, Interface**
- **Packages**
- **Exception Handling**
- **Multithreading, Isolets, Async, Concurrency.**

*1. Write a Dart program that creates a class called ‘Person’ with properties ‘name’ and ‘age’. Implement a default constructor and a named constructor.
*2. Write a Dart program that demonstrates the usage of static variables and methods in a class.
*3. Write a Dart program that demonstrates the concept of constructor chaining using the **‘this’** keyword.
*4. Write a Dart program that demonstrates the usage of default parameter values in a constructor.
*5. Write a Dart program that demonstrates the usage of the **‘assert’** keyword in a constructor.
*6. Write a Dart program that demonstrates single inheritance by creating a base class **‘Animal’** and a derived class **‘Cat’**.
*7. Write a Dart program that demonstrates hierarchical inheritance by creating a base class **‘Shape’** and two derived classes **‘Rectangle’** and **‘Circle’**.
*8. Write a Dart program that demonstrates method overriding in inheritance.
*9. Write a Dart program that demonstrates the concept of abstract classes and methods in inheritance.
*10.Write a Dart program that demonstrates the usage of mixins inheritance.
*11.Write a Dart program that demonstrates method overriding and runtime ploymorphism
*12.Write a Dart program that demonstrates the concept of runtime polumorphism by having a list of animals and calling the **‘makeSound()’** method for each animal.
*13. Write a Dart program that demonstrates method overriding and runtime polymorphism with a base class and its subclasses.
*14. Write a Dart program that demonstrates method overriding and runtime polymorphism with a base class and its subclasses implementing an interface.
*15. Write a Dart program that demonstrates method overriding and runtime polymorphism with a base class and its subclasses having different parameterized constructors.
*16. Write a Dart program that demonstrate abstraction and abstract classes.
*17. Write a Dart program that demonstrate the concept of abstraction and abstract classes with abstract methods.

*18. Write a Dart program that demonstrates abstraction and abstract classes with named constructors.
*19. Write a Dart program that demonstrates the usage of abstract classes and interfaces in a hierarchical structure.
*20. Write a Dart program that demonstrates abstraction and abstract classes with static members.
*21. Write a Dart program that uses the **‘http’** package to make an HTTP GET request and fetch data from a remote server.
*22. Write a Dart program that uses the **‘path’** package to manipulate file and directory paths.
*23. Write a Dart program that uses the **‘flutter_launcher_icons’** package to generate app launcher icons for a flutter project.
*24. Write a Dart program that uses the **‘flutter_localizations’** package to add internationalization support to a Flutter app.
*25. Write a Dart program that uses the **‘connectivity’** package to check the internet connectivity status in a Flutter app.
*26. Write a Dart program that demonstrates the use of try-catch blocks to handle a specific exception.
*27. Write a Dart program that uses the **‘on’** keyword to catch specific exceptions.
*28. Write a Dart program that uses the **‘finally’** block to perform cleanup operations.
*29. Write a Dart program that demonstrates the use of the **‘finally**’ block with a return statement.
*30. Write a Dart program that demonstrates the use of stack traces to debug exceptions.
*31. Write a Dart program that uses the **‘async’** and **‘await’** keywords to perform an asynchronous operation.
*32. Write a Dart program that uses isolates to perform computationally intensive tasks in parallel.
*33. Write a Dart program that uses the ‘Stream’ class to handle a stream of data asynchronously.
*34. Write a Dart program that uses the **‘await for’** loop to iterate over a stream of data asynchronously.
*35. Write a Dart program that uses the **‘StreamTransformer’** class to perform and filter a stream of data.

#1. Create a Dart program that uses the **‘this’** keyword to refer to the current instance of a class.
#2. Create a Dart program that initialize an object using a static factory method.
#3. Create a Dart program that uses a private constructor to restrict the instantiation of a class.
#4. Create a Dart program that demonstrates the usage of constant constructors.
#5. Create a Dart program that demonstrates the usage of the **‘required’** keyword in a constructor.
#6. Create a Dart program that demonstrates multilevel inheritance by creating a base class **‘Animal’** an intermediate class **‘Mammal’** and a derived class **‘Cat’**.
#7. Create a Dart program that demonstrates multiple inheritance using interfaces by creating two classes **‘Flyer’** and **‘Swimmer’** implementing the interface **‘Animal’**.
#8. Create a Dart program that demonstrates the usage of the **‘super’** keyword to call the base class method.
#9. Create a Dart program that demonstrates the usage of the **‘is’** and **‘as’** keywords for type checking and type casting in inheritance.
#10. Create a Dart program that demonstrates the usage of abstract classes and multiple inheritance using interfaces.
#11. Create a Dart program that demonstrates the usage of the **‘super’** keyword to call the base class method.
#12. Create a Dart program that demonstrates method overriding and runtime polymorphism with a parameterized method.
#13. Create a Dart program that demonstrates the usage of the ‘super’ keyword with parameters in a method override.
#14. Create a Dart program that demonstrates runtime polymorphism with a base class and its subclasses implementing multiple interfaces.
#15. Create a Dart program that demonstrates the usage of the **‘super’** keyword with named constructors in method overriding.

#16. Create a Dart program that demonstrate abstraction and interfaces.
#17. Create a Dart program that demonstrate abstraction and interfaces with multiple inheritance.
#18. Create a Dart program that demonstrates abstraction and interfaces with default implementations.
#19. Create a Dart program that demonstrates abstraction and interfaces with getters and setters.
#20. Create a Dart program that demonstrates abstraction and interfaces with private members.
#21. Create a Dart program that uses the **‘int1’** package to format a date and time according to a specific locale.
#22. Create a Dart program that uses the **‘shared_preferences’** package to persist data locally on the device.
#23. Create a Dart program that uses the **‘firebase_core’** package to initialize Firebase in a Flutter project.
#24. Create a Dart program that uses the **‘flutter_svg’** package to display SVG images in a Flutter app.
#25. Create a Dart program that uses the **‘sqflite’** package to perform SQLite database operation in a Flutter app.
#26. Create a Dart program that uses try-catch-finally blocks to handle exceptions and ensure cleanup of resources.
#27. Create a Dart program that demonstrates the use of the **‘rethrow’** keyword to rethrow an exception.
#28. Create a Dart program that demonstrates the use of custom exceptions.
#29. Create a Dart program that uses the **‘on’** keyword to catch multiple exceptions.
#30. Create a Dart program that uses the **‘assert’** function to validate inputs and detect logical errors.
#31. Create a Dart program that uses the **‘Future’** class to handle a long-running computation asynchronously.
#32. Create a Dart program that demonstrates the use of ‘**Future.wait’** to perform multiple asynchronous operations concurrently.
#33. Create a Dart program that demonstrates the use of **‘StreamController’** to create and control a stream of data.
#34. Create a Dart program that demonstrates the use of **‘Completer’** to handle a future result asynchronously.

Section 3 – Flutter App Development – (Basic UI)

- **Intro to Mobile UI & Mobile Programming(Backend)**
- **Flutter Architecture**
- **Intro to widgets(stateless & statefull)**
- **Intro to layouts,types(Widget supporting a single child,Widget supporting a multiple child)**
- **Material Widgets(TextField,ButtonBar,Checkbox) with properites(Height,Width,padding,etc...)**
- **Material Widgets (ListTile,FloatingActionButton,FlatButton,IconButton,etc..)**
- **Async,await**
- **Gestures (Touch based device)**
- **Animation, Menu, Gallery, Camera**
- **Navigaton & Routing**
- **state Management (Like login,SignUp,etc...)**

*1. Create a Dart program that displays a simple mobile UI with buttons and text fields.
*2. Implement a Dart program that retrieves data from a remote server using HTTP requests and displays it in a mobile app.
*3. Create a Dart program that uses local storage to store and retrieve data persistently in a mobile app.
*4. Implement a Dart program that communicates with a RESTful API to fetch and display data in a

mobile app.
*5. Create a Dart program that implements a real-time chat functionality using WebSocket communication in a mobile app.
*6. Implement the BLOC (Business Logic Component) pattern in Flutter to manage the state of a simple counter app.
*7. Develop a Flutter app that uses the MVVM (Model-View-ViewModel) architecture to separate the business logic and presentation logic.
*8. Implement the Clean Architecture in a Flutter app by separating the app into layers and defining clear boundaries between the presentation layer, domain layer, and data layer.
*9. Develop a Flutter app that follow the Flutter Modular architecture and uses modules to organize the app into smaller, reusable components.
*10. Implement the MVP (Model-View-Presenter) architecture in a Flutter app, separating the logic of the app into distinct layers and improving testability.
*11. Create a Flutter app with a stateless widget that displays a static text message.
*12. Build a Flutter app with a stateful widget that displays a list of items fetched from an API.
*13. Create a Flutter app with a stateless widget that displays a random quote every time the screen is refreshed.
*14. Build a Flutter app with a stateful widget that fetches data from an API periodically and displays it on the screen.
*15. Create a Flutter app with a stateful widget that validates user input in a form and displays appropriate error message.
*16. Create a Flutter app with a **'Container'** widget that contains a single child widget and applies custom styling properties like color and padding.
*17. Build a Flutter app with a **'Column'** widget that displays multiple child widgets vertically.
*18. Create a Flutter app with a **'ListView'** widget that displays a scrollable list of child widgets.
*19. Build a Flutter app with a **'Wrap'** widget that arranges multiple child widgets to the next line when there's not enough horizontal space.
*20. Create a Flutter app with a **'Card'** widget that displays multiple child widgets within a
*21. Create a Flutter app with a **'TextField'** widget that allows the user to input text and displays the entered text on a button press.
*22. Build a Flutter app with a **'Checkbox'** widget that allows the user to toggle a value and displays the current state.
*23. Create a Flutter app with a **'ButtonBar'** widget that has a specified height, width, and padding.
*24. Develop a Flutter app with a **'Scaffold'** widget that displays an **'AppBar'** and a simple body.
*25. Build a Flutter app with a **'TabBar'** and **'TabBarView'** widget that displays different content based on the selected tab.
*26. Create a Flutter app with a **'BottomNavigationBar'** that includes badges on the icons.
*27. Create a Flutter app with a **'ListTile'** widget that displays a title and subtitle.
*28. Build a Flutter app with a **'FlatButton'** that changes its color when pressed.
*29. Create a Flutter app with a **'RaisedButton'** that shows a snackbar when pressed.
*30. Create a Dart function that simulates an asynchronous task using **'async'** and **'await'**.
*31. Build a Dart function that reads a file asynchronously using **'async'** and '**await'**.
*32. Create a Dart function that performs an asynchronously operation with error handling using '**try-catch'** and **'async'/'await'**,
*33. Implement a Dart function that executes a series of asynchronous tasks sequentially using **'async'** and **'await'**.
*34. Create a Dart function that performs an asynchronous task with a timeout using **'async'** and **'await'**.
*35. Create a Flutter app that detects a tap gesture on a specific widget and performs an action.
*36. Build a Flutter app that detects a swipe gesture in different directions and displays the direction in a text widget.
*37. Create a Flutter app that detects a pinch gesture and scales an image widget.

*38. Build a Flutter app that defects a scale gesture and zooms in/out an image.
*39. Create a Flutter app that detects a force press gesture and displays the pressure level.
*40. Create a Flutter app with a bouncing animation effect.
*41. Create a Flutter app that displays a gallery of images.
*42. Implement a Flutter app with an animated menu.
*43. Develop a Flutter app with a circular menu.
*44. Create a Flutter app with a gallery using a PageView.
*45. Create a Flutter app with two screens and navigate between them.
*46. Create a Flutter app with a bottom navigation bar.
*47. Create a Flutter app with a drawer menu.
*48. Develop a Flutter app with a bottom navigation bar and different screens for each tab.
*49. Implement a Flutter app with a named route and passing arguments.
*50. Implement a simple login form with state management.
*51. Implement a signup form using BLoC pattern for state management.
*52. Implement a theme toggle using Riverpod for state management.
*53. Implement a timer app using Provider for state management.
*54. Implement a user profile screen using BLoC pattern for state management.

#1. Develop a Dart program that retrieves data from a remote server using HTTP requests and displays it in a mobile app.
#2. Build a Dart program that uses geolocation services to fetch the current location of a user's mobile device.
#3. Develop a Dart program that integrates with the device's camera to capture photos and display them in a mobile app.
#4. Build a Dart program that utilize Firebase Cloud Messaging to send push notifications to a mobile device.
#5. Develop a Dart program that utilizes the device's accelerometer to detect motion and perform actions in a mobile app.
#6. Create a Flutter app that follows the Provider pattern to manage global state and demonstrate how to access and update the state from different widgets.
#7. Build a Flutter app that implements the redux architecture for state management and demonstrates how to dispatch actions and handle state changes.
#8. Create a flutter app that uses the Repository pattern to handle data access and provides an abstraction layer between the data sources and the rest of the app.
#9. Build a Flutter app that uses the Provider and Riverpod libraries to implement dependency injection and manage dependencies between different parts of the app.
#10. Create a flutter app that uses the Flutter Hooks package to implement a functional programming style and manage state in a more concise and declarative way.
#11. Develop a Flutter app with a stateful widget that increments a counter when a button is pressed.
#12. Implement a Flutter app with a stateful widget that toggles the visibility of a widget when a button is pressed.
#13. Develop a Flutter app with a stateful widget that fetches data from an API periodically and displays it on the screen.
#14. Implement a Flutter app with a stateful widget that loads and displays images from the internet.
#15. Develop a Flutter app with a stateful widget that uses a timer to update the UI periodically.
#16. Develop a Flutter app with a **'Row'** widget that displays multiple child widgets horizontally.
#17. Implement a Flutter app with a **'Stack'** widget that overlays multiple child widgets on top of each other.
#18. Develop a Flutter app with a **'GridView'** widget that arranges multiple child widgets in a grid layout.
#19. Implement a Flutter app with an **'Expanded'** widget that distributes available space evenly among multiple child widgets.

#20. Develop a Flutter app with a **'Table'** widget that arranges multiple child widgets in a tabular layout with rows and columns.
#21. Develop a Flutter app with a **'ButtonBar'** widget that displays a row of buttons with custom padding and alignment.
#22. Implement a Flutter app with a **'TextField'** widget that has a specified height, width, and padding.
#23. Develop a Flutter app with a **'Checkbox'** widget that has a specified height, width, and padding.
#24. Develop a Flutter app with a **'BottomNavigateBar'** widget that switches between different screens.
#25. Implement a Flutter app with a **'Scaffold'** widget that includes a **'FloatingActionButton'**.
#26. Develop a Flutter app with a **'TopBar'** and **'TabBarView'** that includes custom colors and styling.
#27. Develop a Flutter app with a **'FloatingActionButton'** that performs an action when pressed.
#28. Implement a Flutter app with an **'IconButton'** that shows a dialog when pressed.
#29. Develop a Flutter app with a **'Card'** widget that displays an image and some text.
#30. Develop a Dart program that fetches data from an API using **'async'** and **'await'**.
#31. Implement a Dart program that performs multiple asynchronous tasks concurrently using **'async'** and **'await'**.
#32. Develop a Dart program that performs a time-consuming calculation asynchronously using **'async'** and **'await'**.
#33. Develop a Dart program that fetches data from multiple APIs concurrently using **'async'** and '**await'**.
#34. Implements a Dart program that performs an asynchronous operation with progress updates using **'Stream'** and **'async'/'await'**.
#35. Develop a Flutter app that detects a long press gesture and displays a dialog.
#36. Implement a Flutter app that detects a double tap gesture and changes the color of a widget.
#37. Develop a Flutter app that detects a vertical drag gesture and scrolls list.
#38. Implement a Flutter app that defects a long press move gesture and drags an object on the screen.
#39. Develop a Flutter app that detects a rotate gesture and rotates an image.
#40. Implement a Flutter app with a dropdown menu.
#41. Develop a Flutter app that accesses the camera and captures an image.
#42. Create a Flutter app with a sliding menu.
#43. Implement a Flutter app with a tab bar.
#44. Develop a Flutter app with a camera preview.
#45. Implement a Flutter app with named routes and passing data between screens.
#46. Develop a Flutter app with a tab-based navigation.
#47. Implement a Flutter app with a login screen and navigate to the home screen on successful login.
#48. Create a Flutter app with a bottom navigation bar and persistent state between screens.
#49. Create a Flutter app with a drawer menu and navigate to different screens.
#50. Implement a counter app using Provider for state management.
#51. Implement a shopping cart using MobX for state management.
#52.Implement a todo list using GetX for state management.
#53. Implement a weather app using Riverpod for state management.
#54. Implement a favorites list using MobX for state management.

Section 4 – Flutter App Development – Advanced

- **Create SQlite Based Project**
- **API Using Postman (GET,POST,PUT,DELETE,etc..)**
- **Rest API Use in Application**
- **OTP Verification**
- **Login with Gmail**
- **Firebase Remote Notification**
- **Payment Integration**

*1. Create a Flutter application that stores and retrieves user information (name, email, age) in a SQLite database.
*2. Use the ‘http’ package in Dart to make a GET request to a public API (e.g., weather API) and display the response data in your application.
*3. Build a Flutter app that consumes a Restful API to fetch and display a list of products.
*4. Create a Dart program that generates a random OTP and sends it to a user’s phone number or email using an external service (e.g., Twilio, SendGrid).
*5. Integrate the Google Sign-In API in your Flutter app to allow users to log in using their Gmail accounts.
*6. Set up Firebase Cloud Messaging (FCM) in your Flutter app to send remote push notifications to the device.
*7. Integrate a payment gateway SDK (such as Stripe or PayPal) into your Dart application to enable online payment processing.
*8. Implement JWT (JSON Web Tokens) authentication in your Dart app to secure your Restful API endpoints.
*9. Create a Flutter app with a login/register screen using Firebase Authentication.
*10. Use Firebase Real-time Database in your Dart app to synchronize data across multiple devices in real – time.

#1. Implement CRUD (Create, Read, Update, and Delete) operations on the user data using SQLite.
#2. Implement a POST request to send data to a server (e.g., user registration) using the ‘http’ package.
#3. Implement pagination functionality in your app by making subsequent requests to retrieve additional data from the API.
#4. Implement OTP verification by comparing the entered OTP with the generated OTP in your Dart application.
#5. Retrieve the user’s basic profile information (name, email, profile picture) after successful authentication.
#6. Implement handling of incoming push notifications and display them as system notifications or within the app.
#7. Implement a checkout flow where users can enter payment details and complete a purchase using the integrated payment gateway.
#8. Validate and decode JWT tokens received from the client to authenticate and authorize requests.
#9. Implement different user roles (e.g. admin, regular user) and restrict access to certain features based on the user’s role.
#10. Implement real-time chat functionality where messages sent from one device are instantly visible on other devices.