DATA PERSISTENCE AND NETWORKING

Shuja Akbar
Lecturer
Department of Computer Science
CUI Lahore Campus

DATA PERSISTENCE AND NETWORKING

Data persistence and networking are important concepts in mobile app development, and Flutter provides several ways to implement them. Here's an overview of how you can implement data persistence and networking in your Flutter app:

- **Data Persistence**
- **Networking**

DATA PERSISTENCE

Data persistence is the ability of an app to store data locally on the device. Flutter provides several mechanisms for data persistence, including:

Shared Preferences

Shared Preferences is a simple key-value storage mechanism in Flutter that allows you to persistently store data on the device. It is typically used to store small amounts of data, such as user preferences, settings, or login credentials.

SQLite

SQLite is a lightweight relational database that allows you to store and query structured data in your Flutter app. It is a good choice for storing larger amounts of data or data that needs to be queried frequently.

Firebase

Firebase is a cloud-based platform that provides several services for mobile app development, including a real-time database and cloud storage. Firebase is a good choice for storing and syncing data across multiple devices or for apps that require real-time updates.

NETWORKING

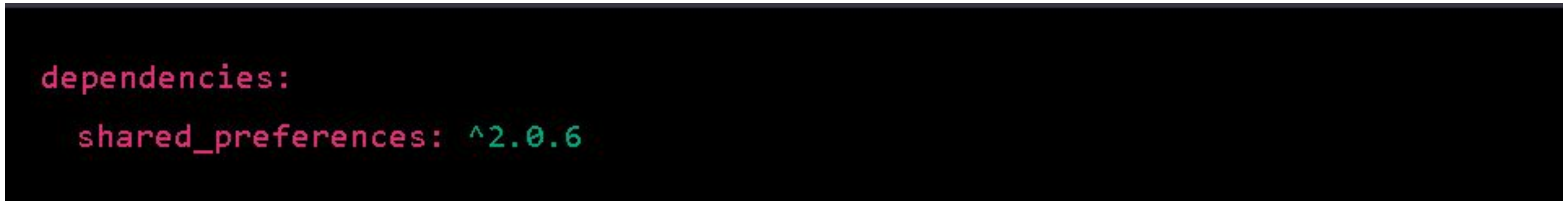
Networking is the process of exchanging data between an app and a server over the internet. Flutter provides several packages for networking, including:

- HTTP
  - The http package is a popular choice for making HTTP requests in Flutter apps. It provides a simple API for sending GET, POST, PUT, DELETE, and other HTTP requests and handling responses.
- Dio
  - The dio package is a more powerful alternative to the http package. It provides more features such as handling timeouts, interceptors, request cancellation, and more.
- GraphQL
  - GraphQL is a query language for APIs that provides a more efficient, powerful, and flexible alternative to REST. The graphql package in Flutter provides tools for working with GraphQL APIs.
- WebSockets
  - WebSockets are a protocol for real-time communication between a client and a server over the internet. The web_socket_channel package in Flutter provides tools for working with WebSockets.

In summary, Flutter provides several mechanisms for implementing data persistence and networking in your app. The choice of mechanism depends on the specific needs of your app, such as the amount of data to be stored or the type of API you're working with.

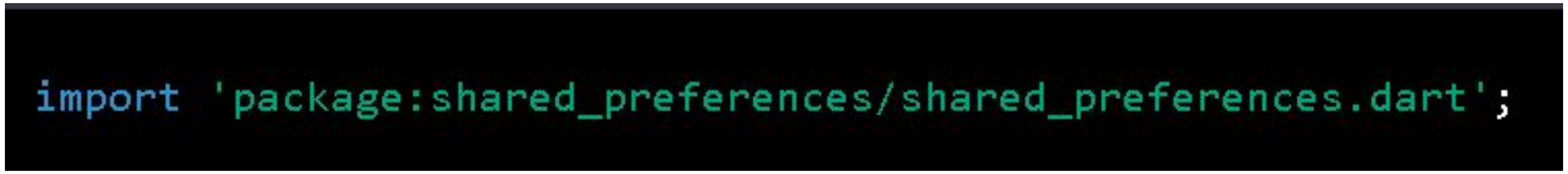
Shared Preferences is a simple key-value storage mechanism in Flutter that allows you to persistently store data on the device. It is typically used to store small amounts of data, such as user preferences, settings, or login credentials. Here's how you can use Shared Preferences to store and retrieve data in your Flutter app:

Add the Shared Preferences dependency to your pubspec.yaml file:



```yaml
dependencies:
  shared_preferences: ^2.0.6
```

Import the shared_preferences package in your Dart file:



```dart
import 'package:shared_preferences/shared_preferences.dart';
```

To write data to shared preferences, create an instance of SharedPreferences and call the setString(), setInt(), setBool(), etc. methods, depending on the data type you want to store:

```
// Store a string value
SharedPreferences prefs = await
SharedPreferences.getInstance();
await prefs.setString('username', 'John');

// Store an integer value
await prefs.setInt('age', 25);

// Store a boolean value
await prefs.setBool('is_logged_in', true);
```

To read data from shared preferences, create an instance of SharedPreferences and call the getString(), getInt(), getBool(), etc. methods:

```
// Read a string value
SharedPreferences prefs = await
SharedPreferences.getInstance();
String username = prefs.getString('username');

// Read an integer value
int age = prefs.getInt('age');

// Read a boolean value
bool isLoggedIn = prefs.getBool('is_logged_in');
```

To delete a value from shared preferences, call the remove() method and pass the key of the value to be removed:

```
SharedPreferences prefs = await
SharedPreferences.getInstance();
prefs.remove('username');
```

Note that Shared Preferences is not suitable for storing large amounts of data or sensitive data such as passwords, as the data is stored in plaintext on the device. In such cases, you should use other storage mechanisms such as SQLite, Firebase, or the device's secure storage.

FETCHING DATA FROM API AND STORE IN SHARED PREFERENCES

Import the http and shared_preferences packages in your Dart file:

```
import 'package:http/http.dart' as http;
import 'package:shared_preferences/shared_preferences.dart';
```

Write a func

```
Future<List<dynamic>> fetchData() async {
  final response = await
http.get(Uri.parse('https://jsonplaceholder.typicode.c
om/posts'));
  if (response.statusCode == 200) {
    // Return the response body as a List of dynamic
objects
    return jsonDecode(response.body);
  } else {
    throw Exception('Failed to fetch data from API');
  }
}
```

Call the fetchData() function and store the returned data in Shared Preferences:

```
SharedPreferences prefs = await
SharedPreferences.getInstance();
List<dynamic> data = await fetchData();
await prefs.setString('api_data', jsonEncode(data));
```

To read the data from Shared Preferences, use the getString() method and parse the JSON data:

```
SharedPreferences prefs = await
SharedPreferences.getInstance();
String apiData = prefs.getString('api_data');
List<dynamic> data = jsonDecode(apiData);
```