Mobile Programming

Flutter Networking & APIs

Lecturer: Ali Najimi
Faculty of Computer Engineering
Sharif University of Technology
Fall 2027

HTTP | Dio | Interceptors | REST



Table of Contents

1. Networking in Mobile Apps
2. Dart's `http` Package
3. Dio — Advanced HTTP Client
4. Interceptors & Middleware
5. Authentication & Token Refresh
6. Error Handling Patterns
7. File Upload & Download
8. WebSocket Communication
9. TLS & Certificate Pinning
10. Mini Lab: Secure API Client
11. Summary & Key Takeaways

Networking in Mobile Apps

Mobile apps are **network-heavy** by nature. Almost every feature involves HTTP requests.

Challenges:

- **Unreliable connectivity:** Airplane mode, weak signal, network switches.
- **Latency:** Mobile networks are slower than WiFi/Ethernet.
- **Battery:** Excessive networking drains battery.
- **Security:** Data in transit must be encrypted.

Best Practices:

1. Always use **HTTPS** (TLS 1.2+).
2. Implement **timeouts** (connect + receive).
3. Handle **offline** gracefully.
4. Cache responses when possible.
5. Minimize payload size (JSON compression, pagination).

Dart's `http` Package

The simplest way to make HTTP requests in Flutter.

```
dependencies:
  http: ^1.2.0
```

```
import 'package:http/http.dart' as http;
Future<List<User>> fetchUsers() async {
  final response = await http.get(
    Uri.parse('https://api.example.com/users'),
    headers: {'Authorization': 'Bearer $token'},
  );

  if (response.statusCode == 200) {
    final List data = jsonDecode(response.body);
    return data.map((json) => User.fromJson(json)).toList();
  } else {
    throw ApiException('Failed: ${response.statusCode}');
  }
}
```

Dio — Advanced HTTP Client

Dio is a powerful HTTP client with interceptors, form data, and more.

```
dependencies:
  dio: ^5.4.0
```

Why Dio over `http`?

- **Interceptors** — Modify requests/responses globally.
- **Automatic retry** — Retry failed requests.
- **FormData** — File uploads with progress.
- **Cancel tokens** — Cancel in-flight requests.
- **Timeouts** — Per-request configuration.
- **Response types** — JSON, plain text, bytes, streams.

Dio — Basic Setup

```dart
class ApiClient {
  late final Dio _dio;
  ApiClient({required String baseUrl, String? token}) {
    _dio = Dio(BaseOptions(
      baseUrl: baseUrl,
      connectTimeout: const Duration(seconds: 10),
      receiveTimeout: const Duration(seconds: 10),
      headers: {
        'Content-Type': 'application/json',
        if (token != null) 'Authorization': 'Bearer $token',
      },
    ));
    _dio.interceptors.addAll([
      LogInterceptor(requestBody: true, responseBody: true),
      AuthInterceptor(_dio),
      RetryInterceptor(_dio),
    ]);
  }
  Future<List<User>> getUsers() async {
    final response = await _dio.get('/users');
    return (response.data as List).map((json) => User.fromJson(json)).toList();
  }
}
```

Interceptors & Middleware

Interceptors let you **intercept and modify** every request and response.

```
class LoggingInterceptor extends Interceptor {
  @override
  void onRequest(RequestOptions options, RequestInterceptorHandler handler) {
    print('→ ${options.method} ${options.uri}');
    handler.next(options); // Continue
  }
  @override
  void onResponse(Response response, ResponseInterceptorHandler handler) {
    print('← ${response.statusCode} ${response.requestOptions.uri}');
    handler.next(response);
  }
  @override
  void onError(DioException err, ErrorInterceptorHandler handler) {
    print('✗ ${err.type} ${err.message}');
    handler.next(err); // Pass error along
  }
}
```

Authentication & Token Refresh

The most critical interceptor — automatically refreshes expired tokens.

```
class AuthInterceptor extends Interceptor {
  final Dio dio;
  AuthInterceptor(this.dio);
  @override
  void onError(DioException err, ErrorInterceptorHandler handler) async {
    if (err.response?.statusCode == 401) {
      try {
        final newToken = await _refreshToken();
        // Update stored token
        await SecureStorage.saveToken(newToken);
        // Retry the original request with new token
        err.requestOptions.headers['Authorization'] = 'Bearer $newToken';
        final response = await dio.fetch(err.requestOptions);
        handler.resolve(response);
      } catch (e) { // Refresh failed → logout
        await SecureStorage.clearAll();
        handler.reject(err);
      }
    } else {
      handler.next(err);
    }
```

Error Handling Patterns

DioException Types

- `connectionTimeout` — Server too slow.
- `sendTimeout` — Upload took too long.
- `receiveTimeout` — Download took too long.
- `badResponse` — Server returned error (4xx/5xx).
- `cancel` — Request was cancelled.
- `connectionError` — No internet.

```
try {
  final users = await api.getUsers();
} on DioException catch (e) {
  switch (e.type) {
    case DioExceptionType.connectionTimeout:
      showError('Connection timed out');
    case DioExceptionType.connectionError:
      showError('No internet connection');
    case DioExceptionType.badResponse:
      final code = e.response?.statusCode;
      if (code == 404) showError('Not found');
      else if (code == 500) showError('Server error');
    default:
      showError('Something went wrong');
  }
}
```

File Upload & Download

Upload with progress

```dart
Future<String> uploadFile(File file) async {
  final formData = FormData.fromMap({
    'file': await MultipartFile.fromFile(
      file.path,
      filename: file.path.split('/').last,
    ),
    'description': 'My upload',
  });

  final response = await _dio.post(
    '/upload',
    data: formData,
    onSendProgress: (sent, total) {
      final progress = (sent / total * 100).toStringAsFixed(0);
      print('Upload: $progress%');
    },
  );
  return response.data['url'];
}
```

File Upload & Download

Download with progress

```dart
await _dio.download(
  'https://example.com/file.pdf',
  '/save/path.pdf',
  onReceiveProgress: (received, total) {
    print('Download: ${(received / total * 100).toStringAsFixed(0)}%');
  },
);
```

File Upload & Download

Use `path_provider` for paths

```dart
final Directory tempDir = await getTemporaryDirectory();

final Directory appDocumentsDir = await getApplicationDocumentsDirectory();

final Directory? downloadsDir = await getDownloadsDirectory();
```

WebSocket Communication

For real-time features (chat, live updates, notifications).

```dart
import 'package:web_socket_channel/web_socket_channel.dart';
class ChatService {
  late final WebSocketChannel _channel;
  void connect(String url) {
    _channel = WebSocketChannel.connect(Uri.parse(url));
    _channel.stream.listen( // Listen to incoming messages
      (message) {
        print('Received: $message');
        // Parse and update UI
      },
      onError: (error) => print('WS Error: $error'),
      onDone: () => print('WS Connection closed'),
    );
  }
  void sendMessage(String text) {
    _channel.sink.add(jsonEncode({'type': 'message', 'text': text}));
  }
  void disconnect() {
    _channel.sink.close();
  }
}
```

TLS & Certificate Pinning

Protect against **Man-in-the-Middle (MITM)** attacks.

What is Certificate Pinning?

- Embed a known certificate hash in your app.
- At runtime, compare the server's certificate against the embedded hash.
- If they don't match → reject the connection.

```
// Dio with certificate pinning
final dio = Dio();
dio.httpClientAdapter = IOHttpClientAdapter(
  createHttpClient: () {
    final client = HttpClient();
    client.badCertificateCallback = (cert, host, port) {
      final hash = cert.sha256.toString(); // Compare certificate SHA-256 hash
      return hash == expectedHash;
    };
    return client;
  },
);
```

Repository Pattern for Networking

```dart
abstract class UserRepository {
  Future<List<User>> getAll();
  Future<User> getById(int id);
  Future<User> create(User user);
  Future<void> delete(int id);
}
class ApiUserRepository implements UserRepository {
  final ApiClient _api;
  ApiUserRepository(this._api);
  @override
  Future<List<User>> getAll() async {
    final response = await _api.get('/users');
    return (response.data as List).map((j) => User.fromJson(j)).toList();
  }
  @override
  Future<User> getById(int id) async {
    final response = await _api.get('/users/$id');
    return User.fromJson(response.data);
  }
}
```

Exercise

Secure API Client Wrapper

Objective: Build a production-ready API client with auth, retry, and error handling.

Requirements:

1. `ApiClient` class with Dio under the hood.
2. Auth interceptor with automatic token refresh.
3. Retry interceptor (max 3 retries with backoff).
4. Logging interceptor (debug mode only).
5. Repository pattern for user CRUD.
6. Proper error handling with custom exceptions.

Summary: Key Takeaways

Fundamentals

- Use HTTPS, always.
- Set timeouts (connect + receive).
- Handle all error types.
- Cache when possible.

Dio

- Interceptors for cross-cutting concerns.
- Token refresh on 401.
- File upload/download with progress.

Security

- Certificate pinning prevents MITM.
- Never store tokens in plain text.
- Use secure storage for credentials.

Architecture

- Repository pattern separates data sources.
- Custom exceptions for domain errors.
- Cancel requests when leaving screens.

Thank You! 🙌

Questions?

"A mobile developer's prayer:
'Lord, give me the strength to handle
every possible network error,
and the wisdom to never trust a 200 OK.'"