

VERSION 1.0

AUGUST, 2023



PEMROGRAMAN MOBILE

TESTING DAN BUILD APK – MODUL 6 MATERI

TIM PENYUSUN:

- DIDIH RIZKI CHANDRANEGARA, S.KOM., M.KOM.
- MUHAMMAD ZULFIQOR LILHAQ
- RIYAN PUTRA FIRJATULLAH

PRESENTED BY: LAB. INFORMATIKA UNIVERSITAS MUHAMMADIYAH MALANG

CAPAIAN PEMBELAJARAN PRAKTIKUM

1. Mahasiswa dapat memahami penggunaan Testing pada Framework Flutter.
 2. Mahasiswa dapat memahami jenis Testing pada Framework Flutter.
 3. Mahasiswa dapat memahami build APK pada aplikasi Framework Flutter.
-

SUB CAPAIAN PEMBELAJARAN PRAKTIKUM

1. Mahasiswa dapat mengimplementasikan Testing pada aplikasi menggunakan Framework Flutter.
 2. Mahasiswa dapat mengimplementasikan beberapa jenis Testing pada aplikasi menggunakan Framework Flutter.
 3. Mahasiswa dapat mengimplementasikan build APK menggunakan Framework Flutter.
-

KEBUTUHAN HARDWARE & SOFTWARE

1. PC/Laptop
 2. IDE Android Studio/ Visual Studio Code
 3. Flutter SDK: <https://docs.flutter.dev/release/archive?tab=windows>
-

MATERI POKOK

Pengertian Testing

Testing dalam pengembangan perangkat lunak merujuk pada proses verifikasi dan validasi program untuk memastikan bahwa itu berfungsi seperti yang diharapkan dan memenuhi persyaratan bisnis. Tujuan utama pengujian adalah untuk mengidentifikasi bug, kesalahan logika, atau masalah lain dalam kode sebelum aplikasi diterapkan ke pengguna akhir. Dengan melakukan pengujian secara menyeluruh, Anda dapat meningkatkan kualitas perangkat lunak, mengurangi kerugian waktu dan sumber daya yang dihabiskan untuk pemecahan masalah setelah peluncuran, dan meningkatkan kepercayaan pengguna terhadap produk Anda. Pada Flutter terdapat beberapa jenis testing diantaranya:

1. **Unit Testing:** Menguji unit-unit kecil kode, seperti fungsi atau metode, secara terisolasi dari komponen lainnya. Tujuannya adalah untuk memastikan bahwa setiap unit kode berfungsi dengan benar.
2. **Widget Testing:** Menguji widget dalam isolasi dan memastikan bahwa tampilan dan interaksi widget sesuai dengan yang diharapkan.
3. **Integration Testing:** Menggabungkan berbagai komponen dan menguji interaksi antara komponen-komponen tersebut.



Setiap kategori testing memiliki kelebihan dan kekurangan seperti ditunjukkan dalam tabel berikut:

	Unit	Widget	Integration
Confidence	Low	Higher	Highest
Maintenance Cost	Low	Higher	Highest
Dependencies	Few	More	Most
Execution Speed	Quick	Quick	Slow

Installation

Sebelum melakukan testing kita perlu mempersiapkan package [test](#). Package ini akan otomatis terpasang ketika Anda membuat project pertama kali. Pada berkas [pubspec.yaml](#) package termasuk ke dalam dependency [flutter_test](#). Jadi pastikan saja package ini sudah terpasang.

Penambahan package [flutter_test](#) juga berbeda dengan package yang biasa kita gunakan. Dimana package [flutter_test](#) diletakkan pada [dev_dependencies](#). Perbedaananya adalah dependency package terbagi ke dalam dua kategori, yaitu [dependencies](#) dan [dev_dependencies](#). [Dependencies](#) berisi package yang kita butuhkan untuk menjalankan [fitur aplikasi](#), sementara [dev_dependencies](#) berisi package yang hanya diperlukan selama [masa development](#). Artinya, package tersebut tidak akan disertakan ketika aplikasi di-build pada versi rilis.

- 1 Buka file [pubspec.yaml](#) pada project
 -  nama_project
 -  [pubspec.yaml](#)
- 2 Periksa pada bagian [dev_dependencies](#) [flutter_test](#) sudah terpasang.

```
dev_dependencies:
  flutter_test:
    sdk: flutter
```

Unit Testing


Pada unit testing ini kita akan melakukan testing pada kelas [ApiService](#) untuk menguji fungsi di dalam kelas tersebut yang mengambil data dari internet berfungsi dengan baik. [Berkas pengujian](#) harus diletakkan di dalam [folder test](#). Menurut convention dari Flutter [nama berkas pengujian](#) harus diakhiri dengan [_test.dart](#). Hal ini untuk membedakan antara berkas pengujian dan berkas yang akan diuji. Untuk pengujian unit testing kita akan menggunakan 2 package tambahan yaitu [mockito](#) dan [build_runner](#). [Unit testing documentation](#)

Mockito

Mockito adalah sebuah pustaka (library) dalam bahasa Dart yang digunakan untuk membuat objek palsu (mock objects) dalam proses pengujian unit (unit testing). Mock objects adalah objek yang meniru perilaku objek nyata, tetapi dapat dikendalikan untuk mensimulasikan berbagai skenario dan respons.

Installation

Tambahkan package mockito pada file `pubspec.yaml` [mockito](#).

- 1 Buka file `pubspec.yaml` pada project
 -  nama_project
 -  `pubspec.yaml`
- 2 Ubah dari code **sebelum** (kiri) menjadi **sesudah** (kanan) lalu **save**.

```
dev_dependencies:
  flutter_test:
    sdk: flutter
```

```
dev_dependencies:
  mockito: ^5.4.2
  flutter_test:
    sdk: flutter
```



- 3 Selanjutnya **mockito** sudah dapat dipakai oleh project.

Build Runner

Build Runner digunakan untuk menghasilkan kode tambahan berdasarkan anotasi dalam proyek Anda. Ini membantu Anda menghasilkan kode yang dibutuhkan untuk fungsionalitas tertentu, seperti serialisasi JSON atau kelas-kelas generik.

Installation

Tambahkan package **build_runner** pada file `pubspec.yaml` [build_runner](#).

- 1 Buka file `pubspec.yaml` pada project
 -  nama_project
 -  `pubspec.yaml`
- 2 Ubah dari code **sebelum** (kiri) menjadi **sesudah** (kanan) lalu **save**.

```
dev_dependencies:
  flutter_test:
    sdk: flutter
```

```
dev_dependencies:
  # mockito
  mockito: ^5.4.2
  # build_runner
  build_runner: ^2.4.6
  # flutter_test
  flutter_test:
    sdk: flutter
```

- 3 Selanjutnya **build_runner** sudah dapat dipakai oleh project.

Implementasi Unit Testing

Setelah menginstall mockito dan build_runner maka kita bisa melanjutkan untuk membuat testingnya.

1. Buatlah file **api_service.dart** yang di dalamnya berisi class **ApiService()** untuk menangani pengambilan data dari internet.

```
class ApiService extends GetxController {
  static const String _baseUrl = 'https://newsapi.org/v2/';
  static const String _apiKey =
    'YOUR_API_KEY'; //API KEY yang sudah didapat
  static const String _category = 'business';
  static const String _country = 'us'; //us maksudnya United States ya

  RxList<Article> articles = RxList<Article>([]);
  RxBool isLoading = false.obs; // Observable boolean for loading state
  static final http.Client _client = http.Client();

  Future<List<Article>> fetchArticles() async {
    try {
      isLoading.value = true; // Set loading state to true
      final response = await _client.get(Uri.parse(
        '${_baseUrl}top-headlines?country=$_country&category=$_category&apiKey=$_apiKey'));
      if (response.statusCode == 200) {
        final jsonData = response.body;
        final articlesResult = ArticlesResult.fromJson(json.decode(jsonData));
        articles.value = articlesResult.articles;
        return articlesResult.articles;
      } else {
        print('Request failed with status: ${response.statusCode}');
        return [];
      }
    } catch (e) {
      print('An error occurred: $e');
      return [];
    } finally {
      isLoading.value = false; // Set loading state to false when done
    }
  }
}
```

2. Buatlah **model** untuk menampung response dari API, berikut modelnya:

```

class ArticlesResult {
    final String status;
    final Int totalResults;
    final List<Article> articles;

    ArticlesResult({
        required this.status,
        required this.totalResults,
        required this.articles,
    });

    factory ArticlesResult.fromJson(Map<String, dynamic> json) => ArticlesResult(
        status: json["status"],
        totalResults: json["totalResults"],
        articles: List<Article>.from((json["articles"] as List)
            .map((x) => Article.fromJson(x))
            .where((article) =>
                article.author != null &&
                article.description != null &&
                article.urlToImage != null &&
                article.publishedAt != null &&
                article.content != null)),
    );
}

class Article {
    String? author;
    String title;
    String? description;
    String url;
    String? urlToImage;
    DateTime? publishedAt;
    String? content;

    Article({
        required this.author,
        required this.title,
        required this.description,
        required this.url,
        required this.urlToImage,
        required this.publishedAt,
        required this.content,
    });
}

```

```
factory Article.fromJson(Map<String, dynamic> json) => Article(
  author: json["author"],
  title: json["title"],
  description: json["description"],
  url: json["url"],
  urlToImage: json["urlToImage"],
  publishedAt: DateTime.parse(json["publishedAt"]),
  content: json["content"],
);
}
```

3. Kemudian buatlah folder baru di dalam folder **test** dengan nama **data**. Kemudian buat file **api_service_test.dart**, jadi struktur folder nya **test/data/api_service_test.dart**. Masukkan kode berikut:

```
import 'package:http/http.dart' as http;
import 'package:mockito/annotations.dart';

// Generate a MockClient using the Mockito package.
// Create new instances of this class in each test.
@GenerateMocks([http.Client])
void main() {
}
```

4. Kemudian **jalankan perintah** berikut pada **terminal** untuk generate mock nya:

```
flutter pub run build_runner build
```

5. Pada file **api_service_test.dart** tambahkan codingan berikut:

```
@GenerateMocks([http.Client, ApiService])
void main() {
  // Constants for API details
  const _apiKey = 'YOUR_API_KEY';
  const _baseUrl = 'https://newsapi.org/v2/';
  const String _category = 'business';
  const String _country = 'us';

  // Initialize ApiService and MockClient
}
```

```

late ApiService apiService;
late MockClient mockClient;

setUp() {
  // Set up MockClient and ApiService for each test
  mockClient = MockClient();
  apiService = ApiService();
};

group('ApiService', () {
  test('fetchArticles returns a list of articles if response is successful',
    () async {
      // Mock the HTTP response for successful case
      when(mockClient.get(Uri.parse(
        '${_baseUrl}top-headlines?country=$_country&category=$_category&apiKey=$_apiKey')))
        .thenAnswer((_) async =>
          http.Response('{"articles": []}', 200)); // Mock the HTTP response

      final articles = await apiService.fetchArticles();

      // Expect the fetched data to be a list of Article objects
      expect(articles, isA<List<Article>>());
    });

  test('fetchArticles returns an empty list if response fails', () async {
      // Mock the HTTP response for response failure
      when(mockClient.get(Uri.parse(
        '${_baseUrl}top-headlines?country=$_country&category=$_category&apiKey=$_apiKey')))
        .thenAnswer((_) async =>
          http.Response('Server error', 500)); // Mock the HTTP response

      final articles = await apiService.fetchArticles();

      // Expect the fetched data to be an empty list
      expect(articles, isA<List<Article>>());
    });

  test('fetchArticles returns an empty list if an error occurs', () async {
      // Mock an error response
      when(mockClient.get(Uri.parse(
        '${_baseUrl}top-headlines?country=$_country&category=$_category&apiKey=$_apiKey')))
        .thenThrow(Exception('Test error')); // Mock an error
    });

```



```

final articles = await apiService.fetchArticles();

// Expect the fetched data to be an empty list
expect.articles, isA<List<Article>>());
});
});
}

```

- **setUp**: Ini adalah blok yang dijalankan sebelum setiap tes dimulai. Disini, kita menginisialisasi **MockClient** dan **ApiService** agar siap digunakan dalam setiap tes.
 - **group**: Ini digunakan untuk mengelompokkan serangkaian tes yang berkaitan, dalam hal ini, tes untuk **ApiService**.
 - **test**: Blok ini berisi tes spesifik yang akan dijalankan. Komentar menjelaskan apa yang diharapkan dari tes tersebut.
6. Setelah itu **jalankan test** dengan menggunakan perintah seperti berikut di **terminal**:

```
flutter test test/data/api_service_test.dart
```

Widget Testing

Untuk pengujian widget testing kita bisa membuat halaman sederhana untuk menguji halaman tersebut menggunakan widget testing. [Widget testing documentation](#)

1. Pertama buat halaman widget nya kita bisa menamai file nya dengan **todo_page.dart**.

```

class TodoPage extends StatefulWidget {
  // const TodoList({super.key});

  @override
  State<TodoPage> createState() => _TodoPageState();
}

class _TodoPageState extends State<TodoPage> {
  static const _appTitle = 'Todo List';
  final todos = <String>[];
  final controller = TextEditingController();

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: _appTitle,
      home: Scaffold(

```

```

appBar: AppBar(
  title: const Text(_appTitle),
),
body: Column(
  children: [
    TextField(
      controller: controller,
    ),
    Expanded(
      child: ListView.builder(
        itemCount: todos.length,
        itemBuilder: (context, index) {
          final todo = todos[index];

          return Dismissible(
            key: Key('$todo$index'),
            onDismissed: (direction) => todos.removeAt(index),
            background: Container(color: Colors.red),
            child: ListTile(title: Text(todo)),
          );
        },
      ),
    ],
  ),
floatingActionButton: FloatingActionButton(
  onPressed: () {
    setState(() {
      todos.add(controller.text);
      controller.clear();
    });
  },
  child: const Icon(Icons.add),
),
);
}

```

2. Kemudian pada **folder test** kita bisa membuat folder baru dengan nama **page** untuk membedakan antara unit testing dan widget testing, kemudian di dalam folder **page** buat file **todo_page_test.dart**. Jadi struktur folder nya **test/page.todo_page_test.dart**.

```

void main() {
  testWidgets('TodoList UI elements', (WidgetTester tester) async {
    // Build the TodoList widget
    await tester.pumpWidget(TodoPage());

    // Verify if the app title is displayed
    expect(find.text('Todo List'), findsOneWidget);

    // Verify if the TextField is displayed
    expect(find.byType(TextField), findsOneWidget);

    // Verify if the FloatingActionButton is displayed
    expect(find.byType(FloatingActionButton), findsOneWidget);
  });

  testWidgets('Adding and removing todos', (WidgetTester tester) async {
    // Build the TodoList widget
    await tester.pumpWidget(TodoPage());

    // Add a todo
    await tester.enterText(find.byType(TextField), 'Test Todo');
    await tester.tap(find.byType(FloatingActionButton));
    await tester.pump();

    // Verify if the added todo is displayed
    expect(find.text('Test Todo'), findsOneWidget);

    // Swipe to delete the todo
    await tester.drag(find.text('Test Todo'), const Offset(500.0, 0.0));
    await tester.pumpAndSettle();

    // Verify if the todo is removed
    expect(find.text('Test Todo'), findsNothing);
  });
}

```

3. Terakhir pada **terminal** kita bisa **jalankan test** nya.



```
flutter test test/page/todo_page_test.dart
```

Integration Testing

Untuk menjalankan integration testing kita memerlukan package **integration_test** yang akan ditambahkan atau diinstall pada file **pubspec.yaml**. [Integration testing documentation](#)

Installation

Tambahkan package **build_runner** pada file **pubspec.yaml** [build_runner](#).

- 1 Buka file **pubspec.yaml** pada project
 -  nama_project
 -  pubspec.yaml
- 2 Ubah dari code **sebelum** (kiri) menjadi **sesudah** (kanan) lalu **save**.

```
dev_dependencies:  
  flutter_test:  
    sdk: flutter
```

```
dev_dependencies:  
  # mockito  
  mockito: ^5.4.2  
  # build_runner  
  build_runner: ^2.4.6  
  # integration_test  
  integration_test:  
    sdk: flutter  
  # flutter_test  
  flutter_test:  
    sdk: flutter
```

- 3 Selanjutnya **integration_test** sudah dapat dipakai oleh project.

Implementasi Integration Testing

Setelah menginstall **integration_test** kita bisa melanjutkan ke pengujiannya.

1. Pertama hampir sama dengan widget testing kita bisa membuat halaman yang akan diuji. Buat file **todo_page.dart**.

```

class TodoList extends StatefulWidget {
  @override
  State<TodoList> createState() => _TodoListState();
}

class _TodoListState extends State<TodoList> {
  static const _appTitle = 'Todo List';
  final todos = <String>[];
  final controller = TextEditingController();

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: _appTitle,
      home: Scaffold(
        appBar: AppBar(
          title: const Text(_appTitle),
        ),
        body: Column(
          children: [
            TextField(
              key: Key('textfield'), // Adding a key to the TextField
              controller: controller,
            ),
            Expanded(
              child: ListView.builder(
                itemCount: todos.length,
                itemBuilder: (context, index) {
                  final todo = todos[index];

                  return Dismissible(
                    key: Key('$_todo$index'), // Adding a unique key
                    onDismissed: (direction) {
                      setState() {
                        todos.removeAt(index);
                      }
                    },
                    background: Container(color: Colors.red),
                    child: ListTile(key: Key('listtile_$index'), title: Text(todo)),
                  );
                },
              ),
            ),
          ],
        ),
      ),
    );
  }
}

```

```

    ],
  ),
  floatingActionButton: FloatingActionButton(
    onPressed: () {
      setState(() {
        todos.add(controller.text);
        controller.clear();
      });
    },
    child: const Icon(Icons.add),
  ),
),
);
}
}

```

2. Perbedaan ketika menggunakan integration test yaitu **integration test** tidak dijalankan dalam proses yang sama dengan **unit** atau **widget test**, sehingga kita perlu meletakkan berkas pengujian pada folder yang berbeda. Konvensi Flutter menyarankan berkas diletakkan pada **folder integration test**. Buatlah folder tersebut pada **root directory** dari project.

```

project_name/
  lib/
  test/
  integration_test/

```

3. Di dalam folder **integration_test**, buatlah file **app_test.dart** Pada berkas ini siapkan inisialisasi testing agar terhubung dengan perangkat mobile.

```

void main() {
  IntegrationTestWidgetsFlutterBinding.ensureInitialized(); // Ensure initialization
  //code test
}

```

4. Kemudian kita bisa menuliskan kode pengujiannya.

```

void main() {
  IntegrationTestWidgetsFlutterBinding.ensureInitialized(); // Ensure initialization
  app.main(); // Start the app
  testWidgets('TodoList displays UI elements', (WidgetTester tester) async {

```

```

await tester.pumpWidget(MaterialApp(home: TodoList()));

// Verify if the correct UI elements are displayed
expect(find.text('Todo List'), findsOneWidget);
expect(find.byType(TextField), findsOneWidget);
expect(find.byType(ListTile), findsNothing); // No items initially
expect(find.byIcon(Icons.add), findsOneWidget);
});

testWidgets('TodoList adds and removes items', (WidgetTester tester) async {
  await tester.pumpWidget(MaterialApp(home: TodoList()));

  // Verify if the correct UI elements are displayed
  expect(find.byType(Dismissible), findsNothing); // No items initially

  // Add an item
  await tester.enterText(find.byType(TextField), 'Task 1');
  await tester.tap(find.byIcon(Icons.add));
  await tester.pump();

  // Verify if the item is added
  expect(find.byType(ListTile), findsOneWidget);
  expect(find.text('Task 1'), findsOneWidget);

  // Dismiss the item
  await tester.drag(find.byType(Dismissible), Offset(500.0, 0.0));
  await tester.pumpAndSettle();

  // Verify if the item is removed
  expect(find.byType(ListTile), findsNothing);
});
}

```

5. Pastikan IDE Anda telah terhubung ke emulator atau perangkat fisik karena pengujian akan dilakukan secara otomatis pada perangkat. Jalankan perintah pada terminal untuk menjalankan testing nya.

```
flutter test integration_test/app_test.dart
```

Build APK

Build APK (Android Package) adalah proses mengkompilasi dan mengemas aplikasi Android Anda ke dalam format yang dapat diinstal di perangkat Android. Ini adalah langkah penting dalam siklus pengembangan

aplikasi, karena APK adalah bentuk akhir dari aplikasi yang dapat didistribusikan dan diinstal di perangkat pengguna. [Build APK documentation](#)

Konfigurasi AndroidManifest.xml

Kita perlu mengatur settingan **AndroidManifest.xml** sebelum membuild aplikasi. File ini berada pada folder **android/app/src/main/AndroidManifest.xml** yang berisikan informasi mengenai aplikasi Android yang akan di-build. Informasi-informasi tersebut berupa **nama aplikasi**, **ikon**, **permission**, **screen orientation**, dan lain-lain.

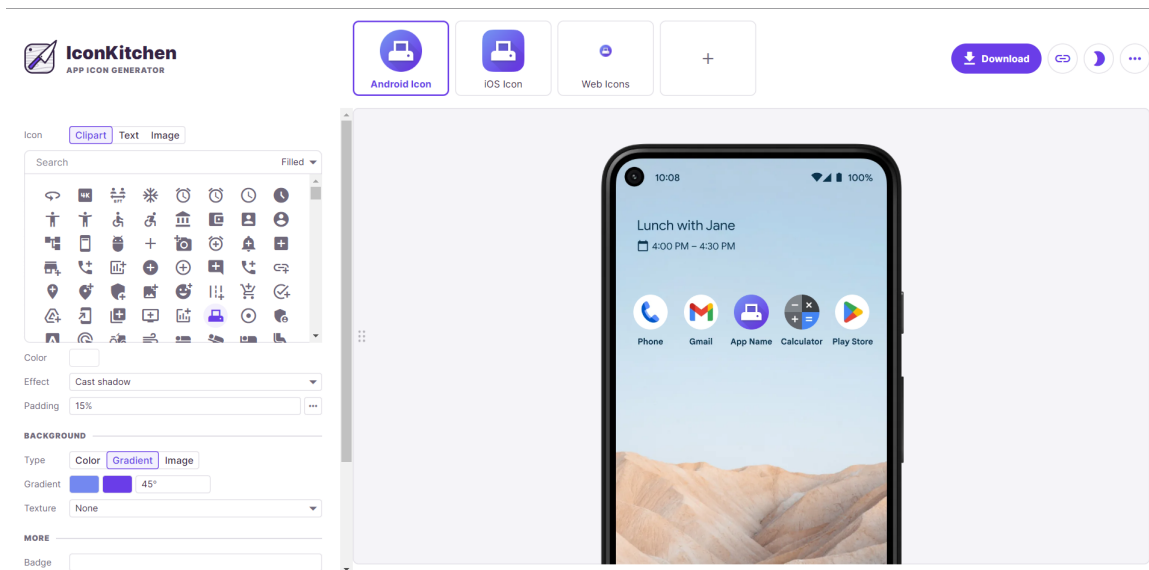
Konfigurasi Nama Aplikasi

Untuk mengatur nama aplikasi caranya dengan mengubah properti **android:label** yang ada pada file **AndroidManifest.xml**.

```
<application
  android:label="news_app" //ganti label untuk mengubah nama aplikasi
  android:name="${applicationName}"
  android:icon="@mipmap/ic_launcher">
```

Konfigurasi Ikon Aplikasi

Secara **default ikon** aplikasi Flutter kita adalah ikon Flutter. Untuk mengubah ikon aplikasi dengan mudah, kita akan mengganti gambar **ic_launcher.png** yang berada pada folder **android/app/src/main/res/** yang terbagi menjadi berbagai **mipmap** (ukuran resolusi ikon). Kita bisa membuat aplikasi dengan bantuan [IconKitchen](#). Dengan aplikasi tersebut kita dapat membuat ikon aplikasi dengan mudah dan nantinya akan terbentuk dalam berbagai resolusi (**mipmap**). Setelah membuat ikon sesuai dengan keinginan, tekan tombol **download** yang ada di kanan atas.



Tampilan Aplikasi IconKitchen

Setelah mengunduh, **unzip-lah** berkas tersebut dan temukan folder **res/** di dalamnya. Lalu **copy folder res/** ke **android/app/src/main/res/** untuk mengganti **ic_launcher.png** pada setiap **mipmap** dengan ikon aplikasi yang baru.

Konfigurasi Perizinan Aplikasi

Ketika aplikasi dalam mode **debug** atau **profil**, perizinan internet akan secara otomatis ditambahkan. Namun ketika Anda ingin menjalankan atau membuatnya dalam mode rilis, Anda perlu menambahkan semua perizinan yang dibutuhkan pada **AndroidManifest**.

Untuk menambahkan perizinan pada aplikasi Android, Anda bisa menambahkan **tag uses-permission** pada **AndroidManifest**, di dalam **tag manifest** dan sejajar **tag application** seperti contoh berikut.

```
<uses-permission android:name="android.permission.INTERNET"/>
```

Melakukan Build APK

Langkah selanjutnya adalah melakukan build aplikasi menjadi APK. Sebelumnya terdapat tiga (3) jenis mode aplikasi yang perlu diketahui, yaitu **debug**, **profile**, dan **release**. APK **debug** umumnya digunakan untuk pengujian dan penggunaan aplikasi secara internal. Mode debug digunakan secara default ketika menjalankan aplikasi menggunakan perintah **flutter run**. Sementara untuk bisa dirilis melalui Google Play Store, Anda perlu membuat APK **release**. Sedangkan mode **profile** sama hal nya dengan **release** hanya saja tetap dapat di-debug menggunakan tools seperti DevTools dan tidak dapat dijalankan di emulator atau simulator.

Untuk melakukan build APK debug cukup jalankan perintah berikut pada **terminal** project anda.

```
flutter build apk --debug
```

Tunggu hingga proses build berhasil. Setelah berhasil, hasil build yang berupa berkas **apk-debug.apk** akan terletak di folder **build/app/outputs/apk/debug/** atau akan muncul direktori tempat tersimpannya berkas ketika proses build selesai pada Terminal.

Untuk bisa mem-build apk release dan mengunggahnya melalui Google Play Store, Anda memerlukan signing key. Signing key ini digunakan sebagai tanda tangan supaya aplikasi Anda lebih aman. Secara default Flutter menggunakan debug key sebagai signing key sehingga Anda sebenarnya bisa membuat apk release cukup menjalankan perintah berikut di terminal.

```
flutter build apk
```

Link Github

Berikut merupakan link github yang akan digunakan pada modul pemrograman mobile:

[Link Github](#)

KEGIATAN PRAKTIKUM

A. Latihan Unit Testing

1. Buatlah unit testing untuk melakukan pengujian terhadap fungsi **fetchPost()** berikut:

Fungsi fetchPosts():

```
Future<Posts> fetchPosts(http.Client client) async {  
  final response = await client  
    .get(Uri.parse('https://jsonplaceholder.typicode.com/posts/1'));  
  
  if (response.statusCode == 200) {  
    // If the server did return a 200 OK response,  
    // then parse the JSON.  
    return Posts.fromJson(jsonDecode(response.body));  
  } else {  
    // If the server did not return a 200 OK response,  
    // then throw an exception.  
    throw Exception('Failed to load album');  
  }  
}
```

Model Posts:

```
class Posts {  
  int userId;  
  int id;  
  String title;  
  String body;  
  
  Posts({  
    required this.userId,  
    required this.id,  
    required this.title,  
    required this.body,  
  });  
  
  factory Posts.fromJson(Map<String, dynamic> json) => Posts(  
    userId: json["userId"],  
    id: json["id"],  
    title: json["title"],  
    body: json["body"],  
  );  
}
```

```
);

Map<String, dynamic> toJson() => {
  "userId": userId,
  "id": id,
  "title": title,
  "body": body,
};
}
```

2. Test case atau test yang ingin dilakukan bebas.
3. Jalankan unit testing seperti biasanya.
4. Untuk memudahkan silahkan bisa membaca dokumentasi unit testing.
[Unit testing documentation](#)

B. Latihan Widget Testing

1. Buatlah widget testing untuk halaman login berikut:

Login_page.dart

```
class LoginPage extends StatefulWidget {
  @override
  State<LoginPage> createState() => _LoginPageState();
}

class _LoginPageState extends State<LoginPage> {
  final TextEditingController _emailController = TextEditingController();
  final TextEditingController _passwordController = TextEditingController();

  @override
  void dispose() {
    _emailController.dispose();
    _passwordController.dispose();
    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text('Login'),
```

```

    ),
    body: Padding(
      padding: const EdgeInsets.all(16.0),
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        crossAxisAlignment: CrossAxisAlignment.center,
        children: [
          TextField(
            controller: _emailController,
            decoration: InputDecoration(labelText: 'Email'),
          ),
          TextField(
            controller: _passwordController,
            obscureText: true,
            decoration: InputDecoration(labelText: 'Password'),
          ),
          SizedBox(height: 16),
          ElevatedButton(
            onPressed: () {
              // Handle login logic here
            },
            child: Text('Login'),
          ),
        ],
      ),
    ),
  ),
);
}
}

```

2. Test case atau test yang ingin dilakukan bebas.
3. Jalankan unit testing seperti biasanya.
4. Untuk memudahkan silahkan bisa membaca dokumentasi unit testing.
[Widget_testing_documentation](#)

RUBRIK PENILAIAN MODUL 6 MATERI

Bobot Penilaian Modul 6 Materi (20%)

Membuat Unit Testing	40
----------------------	----

Membuat Widget Testing	40
Unit Testing dan Widget Testing berjalan lancar	20
Total	100