

NPM :	2210010428
Nama :	Cut Fitriana Rahvi Putri
Kelas :	5 C Reguler Banjarbaru
Mata Kuliah :	UAS Visual 3

## 1. Screenshoot Source Code

### A. login.dart

```

1 import 'package:flutter/material.dart';
2 import 'package:uas_visual3_2210010428/home.dart';
3
4 class LoginPage extends StatefulWidget {
5   const LoginPage({super.key});
6
7   @override
8   _LoginPageState createState() => _LoginPageState();
9 }
10
11 class _LoginPageState extends State<LoginPage> {
12   final TextEditingController _usernameController = TextEditingController();
13   final TextEditingController _passwordController = TextEditingController();
14
15   final String _correctAdminUsername = "admin";
16   final String _correctAdminPassword = "admin";
17
18   final String _correctOperatorUsername = "operator";
19   final String _correctOperatorPassword = "operator";
20
21   String _errorMessage = "";
22
23   void _login() {
24     setState(() {
25       final username = _usernameController.text;
26       final password = _passwordController.text;
27
28       if (username == _correctAdminUsername &&
29         password == _correctAdminPassword) {
30         _errorMessage = "";
31         Navigator.push(
32           context,
33           MaterialPageRoute(
34             builder: (context) => const HomePage(),
35           ),
36         );
37       } else if (username == _correctOperatorUsername &&
38         password == _correctOperatorPassword) {
39         _errorMessage = "";
40         Navigator.push(
41           context,
42           MaterialPageRoute(
43             builder: (context) => const HomePage(),
44           ),
45         );
46       } else {
47         _errorMessage = "Username atau Password salah!";
48       }
49     });
50   }
51
52   @override
53   Widget build(BuildContext context) {
54     return Scaffold(
55       appBar: AppBar(),
56       body: Padding(
57         padding: const EdgeInsets.all(16.0),
58         child: Column(
59           mainAxisAlignment: MainAxisAlignment.center,
60           children: [
61             TextField(
62               controller: _usernameController,
63               decoration: const InputDecoration(
64                 labelText: 'Username',
65                 border: OutlineInputBorder(),
66               ),
67             ),
68             const SizedBox(height: 16.0),
69             TextField(
70               controller: _passwordController,
71               obscureText: true,
72               decoration: const InputDecoration(
73                 labelText: 'Password',
74                 border: OutlineInputBorder(),
75               ),
76             ),
77             const SizedBox(height: 16.0),
78             SizedBox(
79               width: double.infinity,
80               height: 35.0,
81               child: ElevatedButton(
82                 onPressed: _login,
83                 child: const Text('Login', style: TextStyle(fontsize: 18.0)),
84               ),
85             ),
86             const SizedBox(height: 16.0),
87             Text(
88               _errorMessage,
89               style: const TextStyle(color: Colors.red),
90             ),
91           ],
92         ),
93       ),
94     );
95   }
96 }
97

```

### B. Login.dart

```

1 import 'package:flutter/material.dart';
2 import 'package:uas_visual3_2210010428/login.dart';
3
4 void main() {
5   runApp(const MyApp());
6 }
7
8 class MyApp extends StatelessWidget {
9   const MyApp({super.key});
10
11   @override
12   Widget build(BuildContext context) {
13     return MaterialApp(
14       debugShowCheckedModeBanner: false,
15       home: Scaffold(
16         appBar: AppBar(
17           title: const Text('Login Page'),
18           centerTitle: true,
19         ),
20         body: const LoginPage(),
21       ),
22     );
23   }
24 }

```

## C. home.dart

```

1 import 'package:flutter/material.dart';
2 import 'package:uas_visual3_2210010428/pelanggan/pelanggan.dart';
3 import 'package:uas_visual3_2210010428/pelanggan/detail.dart';
4 import 'package:uas_visual3_2210010428/pelanggan/tambah.dart';
5 import 'package:uas_visual3_2210010428/pelanggan/edit.dart';
6 import 'package:uas_visual3_2210010428/layanan/layanan.dart';
7 import 'package:uas_visual3_2210010428/layanan/detail.dart';
8 import 'package:uas_visual3_2210010428/layanan/tambah.dart';
9 import 'package:uas_visual3_2210010428/layanan/edit.dart';
10 import 'package:uas_visual3_2210010428/layanan/tampil.dart';
11 import 'package:uas_visual3_2210010428/pelanggan/tampil.dart';
12
13 class HomePage extends StatelessWidget {
14   const HomePage({super.key});
15
16   @override
17   Widget build(BuildContext context) {
18     return Scaffold(
19       appBar: AppBar(
20         title: Text(
21           "Aplikasi Bengkel Putra",
22           style: TextStyle(
23             color: Colors.white,
24             fontWeight: FontWeight.bold,
25           ),
26         ),
27         backgroundColor: const Color.fromARGB(255, 83, 6, 6),
28         centerTitle: true,
29       ),
30       body: Center(
31         child: Column(
32           children: [
33             SizedBox(
34               height: 30,
35             ),
36             Row(
37               mainAxisAlignment: MainAxisAlignment.spaceEvenly,
38               children: [
39                 tomboli("Pelanggan", icon(Icons.account_circle_rounded),
40                   const Color.fromARGB(255, 255, 110, 112), context),
41                 tomboli("Layanan", icon(Icons.car_repair),
42                   const Color.fromARGB(255, 133, 216, 255), context)
43               ],
44             ),
45           ],
46         ),
47       ),
48     );
49   }
50 }
51
52 Widget tomboli(
53   String judul, Icon iconTombol, Color warna, BuildContext context) {
54   return InkWell(
55     onTap: () {
56       Navigator.push(
57         context, MaterialPageRoute(builder: (context) => TampilPelanggan());
58       ),
59     },
60     child: Container(
61       decoration: BoxDecoration(
62         color: warna, borderRadius: BorderRadius.circular(20)),
63       width: 120,
64       height: 120,
65       child: Column(
66         children: [
67           IconButton(
68             onPressed: () {
69               Navigator.push(context,
70                 MaterialPageRoute(builder: (context) => TampilPelanggan()));
71             },
72             icon: iconTombol,
73             iconSize: 70,
74           ),
75           Text(
76             judul,
77             style: TextStyle(
78               color: const Color.fromARGB(255, 245, 247, 247),
79               fontWeight: FontWeight.bold,
80             ),
81           ),
82         ],
83       ),
84     ),
85   );
86
87   String judul, Icon iconTombol, Color warna, BuildContext context) {
88     return InkWell(
89       onTap: () {
90         Navigator.push(
91           context, MaterialPageRoute(builder: (context) => TampilLayanan());
92         ),
93       },
94       child: Container(
95         decoration: BoxDecoration(
96           color: warna, borderRadius: BorderRadius.circular(20)),
97         width: 120,
98         height: 120,
99         child: Column(
100           children: [
101             IconButton(
102               onPressed: () {
103                 Navigator.push(context,
104                   MaterialPageRoute(builder: (context) => TampilLayanan()));
105               },
106               icon: iconTombol,
107               iconSize: 70,
108             ),
109             Text(
110               judul,
111               style: TextStyle(
112                 color: const Color.fromARGB(255, 245, 247, 247),
113                 fontWeight: FontWeight.bold,
114               ),
115             ),
116           ],
117         ),
118       ),
119     );
120   }
121 }

```

## D. dbhelper.dart

```

1 import 'package:path/path.dart';
2 import 'package:sqflite/sqflite.dart';
3 import 'package:uas_visual3_2210010428/layanan/layanan.dart';
4 import 'package:uas_visual3_2210010428/pelanggan/pelanggan.dart';
5
6 class DbHelper {
7   static final DbHelper _instance = DbHelper._internal();
8   static Database? _database;
9
10   DbHelper._internal();
11
12   factory DbHelper() {
13     return _instance;
14   }
15
16   Future<Database> get database async {
17     if (_database != null) return _database!;
18     _database = await _initDatabase();
19     return _database!;
20   }
21
22   Future<Database> _initDatabase() async {
23     String path = join(await getDatabasesPath(), 'bengkelaputra.db');
24     return await openDatabase(
25       path,
26       version: 1,
27       onCreate: (db, version) {
28         db.execute('''
29           CREATE TABLE pelanggan (
30             no_antrian TEXT PRIMARY KEY,
31             namalengkap TEXT,
32             no_telpn TEXT,
33             alamat TEXT,
34             layanan_servis TEXT,
35             tgl_servis TEXT
36           )
37         ''');
38
39         db.execute('''
40           CREATE TABLE bengkelan (
41             no_layanan TEXT PRIMARY KEY,
42             namalayanan TEXT,
43             deskripsi TEXT,
44             harga TEXT,
45             durasi TEXT,
46             kategori TEXT
47           )
48         ''');
49       },
50     );
51   }
52
53   Future<List<Pelanggan>> getPelanggan({String? query}) async {
54     final db = await database;
55     final List<Map<String, dynamic>> maps;
56     if (query != null && query.isNotEmpty) {
57       maps = await db.query("pelanggan",
58         where: "namalengkap like ?", whereArgs: ['$query%']);
59     } else {
60       maps = await db.query("pelanggan");
61     }
62     return List.generate(
63       maps.length,
64       (index) => Pelanggan.fromMap(maps[index]),
65     );
66   }
67
68   Future<int> insertPelanggan(Pelanggan Plgn) async {
69     final db = await database;
70     return await db.insert("pelanggan", Plgn.toMap());
71   }
72
73   Future<int> deletePelanggan(String no_antrian) async {
74     final db = await database;
75     return await db
76       .delete("pelanggan", where: "no_antrian=?", whereArgs: [no_antrian]);
77   }
78
79   Future<int> updatePelanggan(Pelanggan Plgn) async {
80     final db = await database;
81     return await db.update("pelanggan", Plgn.toMap(),
82       where: "no_antrian=?", whereArgs: [Plgn.no_antrian]);
83   }
84
85   Future<List<Layanan>> getLayanan({String? query}) async {
86     final db = await database;
87     final List<Map<String, dynamic>> maps;
88     if (query != null && query.isNotEmpty) {
89       maps = await db.query("bengkelan",
90         where: "namalayanan like ?", whereArgs: ['$query%']);
91     } else {
92       maps = await db.query("bengkelan");
93     }
94     return List.generate(
95       maps.length,
96       (index) => Layanan.fromMap(maps[index]),
97     );
98   }
99
100   Future<int> insertLayanan(Layanan Lyn) async {
101     final db = await database;
102     return await db.insert("bengkelan", Lyn.toMap());
103   }
104
105   Future<int> deleteLayanan(String no_layanan) async {
106     final db = await database;
107     return await db
108       .delete("bengkelan", where: "no_layanan=?", whereArgs: [no_layanan]);
109   }
110
111   Future<int> updateLayanan(Layanan Lyn) async {
112     final db = await database;
113     return await db.update("bengkelan", Lyn.toMap(),
114       where: "no_layanan=?", whereArgs: [Lyn.no_layanan]);
115   }
116 }
117

```

B. tambah.dart

[illegible]

## C. pelanggan.dart

```

1 class Pelanggan {
2   final String no_antrian;
3   final String namalengkap;
4   final String no_telpon;
5   final String alamat;
6   final String layanan_servis;
7   final String tgl_servis;
8
9   Pelanggan({
10     required this.no_antrian,
11     required this.namalengkap,
12     required this.no_telpon,
13     required this.alamat,
14     required this.layanan_servis,
15     required this.tgl_servis,
16   });
17
18   factory Pelanggan.fromMap(Map<String, dynamic> map) {
19     return Pelanggan(
20       no_antrian: map['no_antrian'],
21       namalengkap: map['namalengkap'],
22       no_telpon: map['no_telpon'],
23       alamat: map['alamat'],
24       layanan_servis: map['layanan_servis'],
25       tgl_servis: map['tgl_servis']);
26   }
27
28   Map<String, dynamic> toMap() {
29     return {
30       'no_antrian': no_antrian,
31       'namalengkap': namalengkap,
32       'no_telpon': no_telpon,
33       'alamat': alamat,
34       'layanan_servis': layanan_servis,
35       'tgl_servis': tgl_servis,
36     };
37   }
38 }
39

```

## D. edit.dart

```

1 import 'package:flutter/material.dart';
2 import 'package:http/http.dart' as http;
3 import 'package:shared_preferences/shared_preferences.dart';
4
5 final _httpClient = HttpClientOverrides();
6 var _httpClient = HttpClientOverrides();
7 var _httpClient = HttpClientOverrides();
8 var _httpClient = HttpClientOverrides();
9 var _httpClient = HttpClientOverrides();
10 var _httpClient = HttpClientOverrides();
11 var _httpClient = HttpClientOverrides();
12
13 class EditPelanggan extends StatefulWidget {
14   final String id;
15   EditPelanggan({Key? key, required this.id}) : super(key: key);
16
17   @override
18   State<EditPelanggan> createState() => _EditPelangganState();
19 }
20
21 class _EditPelangganState extends State<EditPelanggan> {
22   final TextEditingController _idController = TextEditingController();
23   final TextEditingController _noAntrianController = TextEditingController();
24   final TextEditingController _namaLengkapController = TextEditingController();
25   final TextEditingController _noTelponController = TextEditingController();
26   final TextEditingController _alamatController = TextEditingController();
27   final TextEditingController _layananServisController = TextEditingController();
28   final TextEditingController _tglServisController = TextEditingController();
29
30   @override
31   void initState() {
32     super.initState();
33     _idController.text = widget.id;
34     _noAntrianController.text = '';
35     _namaLengkapController.text = '';
36     _noTelponController.text = '';
37     _alamatController.text = '';
38     _layananServisController.text = '';
39     _tglServisController.text = '';
40   }
41
42   @override
43   void dispose() {
44     _idController.dispose();
45     _noAntrianController.dispose();
46     _namaLengkapController.dispose();
47     _noTelponController.dispose();
48     _alamatController.dispose();
49     _layananServisController.dispose();
50     _tglServisController.dispose();
51   }
52
53   Future<void> _updatePelanggan() async {
54     final url = 'http://localhost:8080/api/pelanggan/edit/${_idController.text}';
55     final response = await http.put(url, headers: {'Content-Type': 'application/json'}, body: {
56       'no_antrian': _noAntrianController.text,
57       'namalengkap': _namaLengkapController.text,
58       'no_telpon': _noTelponController.text,
59       'alamat': _alamatController.text,
60       'layanan_servis': _layananServisController.text,
61       'tgl_servis': _tglServisController.text,
62     });
63     if (response.statusCode == 200) {
64       // Success
65     } else {
66       // Error
67     }
68   }
69
70   Future<void> _deletePelanggan() async {
71     final url = 'http://localhost:8080/api/pelanggan/delete/${_idController.text}';
72     final response = await http.delete(url, headers: {'Content-Type': 'application/json'});
73     if (response.statusCode == 200) {
74       // Success
75     } else {
76       // Error
77     }
78   }
79
80   Future<void> _showSnackBar(String message) {
81     ScaffoldMessenger.of(context).showSnackBar(
82       SnackBar(content: Text(message)),
83     );
84   }
85
86   Future<void> _showDialog(String title, String message) {
87     showDialog(
88       context: context,
89       builder: (BuildContext context) {
90         return AlertDialog(
91           title: Text(title),
92           content: Text(message),
93           actions: [
94             TextButton(
95               onPressed: () {
96                 Navigator.of(context).pop();
97               },
98               child: Text('OK'),
99             ),
100           ],
101         );
102       },
103     );
104   }
105
106   Future<void> _showLoadingDialog() {
107     showDialog(
108       context: context,
109       builder: (BuildContext context) {
110         return Center(
111           child: CircularProgressIndicator(),
112         );
113       },
114     );
115   }
116
117   Future<void> _hideLoadingDialog() {
118     Navigator.of(context).pop();
119   }
120
121   Future<void> _showSuccessDialog() {
122     showDialog(
123       context: context,
124       builder: (BuildContext context) {
125         return AlertDialog(
126           title: Text('Success'),
127           content: Text('Data berhasil disimpan'),
128           actions: [
129             TextButton(
130               onPressed: () {
131                 Navigator.of(context).pop();
132               },
133               child: Text('OK'),
134             ),
135           ],
136         );
137       },
138     );
139   }
140
141   Future<void> _showErrorDialog() {
142     showDialog(
143       context: context,
144       builder: (BuildContext context) {
145         return AlertDialog(
146           title: Text('Error'),
147           content: Text('Data gagal disimpan'),
148           actions: [
149             TextButton(
150               onPressed: () {
151                 Navigator.of(context).pop();
152               },
153               child: Text('OK'),
154             ),
155           ],
156         );
157       },
158     );
159   }
160
161   Future<void> _showDeleteDialog() {
162     showDialog(
163       context: context,
164       builder: (BuildContext context) {
165         return AlertDialog(
166           title: Text('Delete'),
167           content: Text('Apakah anda yakin ingin menghapus data ini?'),
168           actions: [
169             TextButton(
170               onPressed: () {
171                 Navigator.of(context).pop();
172               },
173               child: Text('Batal'),
174             ),
175             TextButton(
176               onPressed: () {
177                 _deletePelanggan();
178               },
179               child: Text('Hapus'),
180             ),
181           ],
182         );
183       },
184     );
185   }
186
187   Future<void> _showEditDialog() {
188     showDialog(
189       context: context,
190       builder: (BuildContext context) {
191         return EditPelanggan(id: _idController.text);
192       },
193     );
194   }
195
196   Future<void> _showAddDialog() {
197     showDialog(
198       context: context,
199       builder: (BuildContext context) {
200         return AddPelanggan();
201       },
202     );
203   }
204
205   Future<void> _showDetailDialog() {
206     showDialog(
207       context: context,
208       builder: (BuildContext context) {
209         return DetailPelanggan(id: _idController.text);
210       },
211     );
212   }
213
214   Future<void> _showListDialog() {
215     showDialog(
216       context: context,
217       builder: (BuildContext context) {
218         return ListPelanggan();
219       },
220     );
221   }
222
223   Future<void> _showHomeDialog() {
224     showDialog(
225       context: context,
226       builder: (BuildContext context) {
227         return Home();
228       },
229     );
230   }
231
232   Future<void> _showAboutDialog() {
233     showDialog(
234       context: context,
235       builder: (BuildContext context) {
236         return About();
237       },
238     );
239   }
240
241   Future<void> _showSettingsDialog() {
242     showDialog(
243       context: context,
244       builder: (BuildContext context) {
245         return Settings();
246       },
247     );
248   }
249
250   Future<void> _showHelpDialog() {
251     showDialog(
252       context: context,
253       builder: (BuildContext context) {
254         return Help();
255       },
256     );
257   }
258
259   Future<void> _showPrivacyDialog() {
260     showDialog(
261       context: context,
262       builder: (BuildContext context) {
263         return Privacy();
264       },
265     );
266   }
267
268   Future<void> _showTermsDialog() {
269     showDialog(
270       context: context,
271       builder: (BuildContext context) {
272         return Terms();
273       },
274     );
275   }
276
277   Future<void> _showContactDialog() {
278     showDialog(
279       context: context,
280       builder: (BuildContext context) {
281         return Contact();
282       },
283     );
284   }
285
286   Future<void> _showFeedbackDialog() {
287     showDialog(
288       context: context,
289       builder: (BuildContext context) {
290         return Feedback();
291       },
292     );
293   }
294
295   Future<void> _showLegalDialog() {
296     showDialog(
297       context: context,
298       builder: (BuildContext context) {
299         return Legal();
300       },
301     );
302   }
303
304   Future<void> _showDisclaimerDialog() {
305     showDialog(
306       context: context,
307       builder: (BuildContext context) {
308         return Disclaimer();
309       },
310     );
311   }
312
313   Future<void> _showCookieDialog() {
314     showDialog(
315       context: context,
316       builder: (BuildContext context) {
317         return Cookie();
318       },
319     );
320   }
321
322   Future<void> _showGDPRDialog() {
323     showDialog(
324       context: context,
325       builder: (BuildContext context) {
326         return GDPR();
327       },
328     );
329   }
330
331   Future<void> _showAccessibilityDialog() {
332     showDialog(
333       context: context,
334       builder: (BuildContext context) {
335         return Accessibility();
336       },
337     );
338   }
339
340   Future<void> _showSecurityDialog() {
341     showDialog(
342       context: context,
343       builder: (BuildContext context) {
344         return Security();
345       },
346     );
347   }
348
349   Future<void> _showDataPrivacyDialog() {
350     showDialog(
351       context: context,
352       builder: (BuildContext context) {
353         return DataPrivacy();
354       },
355     );
356   }
357
358   Future<void> _showDataProtectionDialog() {
359     showDialog(
360       context: context,
361       builder: (BuildContext context) {
362         return DataProtection();
363       },
364     );
365   }
366
367   Future<void> _showDataRetentionDialog() {
368     showDialog(
369       context: context,
370       builder: (BuildContext context) {
371         return DataRetention();
372       },
373     );
374   }
375
376   Future<void> _showDataDeletionDialog() {
377     showDialog(
378       context: context,
379       builder: (BuildContext context) {
380         return DataDeletion();
381       },
382     );
383   }
384
385   Future<void> _showDataPortabilityDialog() {
386     showDialog(
387       context: context,
388       builder: (BuildContext context) {
389         return DataPortability();
390       },
391     );
392   }
393
394   Future<void> _showDataAnonymizationDialog() {
395     showDialog(
396       context: context,
397       builder: (BuildContext context) {
398         return DataAnonymization();
399       },
400     );
401   }
402
403   Future<void> _showDataMinimizationDialog() {
404     showDialog(
405       context: context,
406       builder: (BuildContext context) {
407         return DataMinimization();
408       },
409     );
410   }
411
412   Future<void> _showDataLimitationDialog() {
413     showDialog(
414       context: context,
415       builder: (BuildContext context) {
416         return DataLimitation();
417       },
418     );
419   }
420
421   Future<void> _showDataStorageDialog() {
422     showDialog(
423       context: context,
424       builder: (BuildContext context) {
425         return DataStorage();
426       },
427     );
428   }
429
430   Future<void> _showDataTransferDialog() {
431     showDialog(
432       context: context,
433       builder: (BuildContext context) {
434         return DataTransfer();
435       },
436     );
437   }
438
439   Future<void> _showDataAccessDialog() {
440     showDialog(
441       context: context,
442       builder: (BuildContext context) {
443         return DataAccess();
444       },
445     );
446   }
447
448   Future<void> _showDataControlDialog() {
449     showDialog(
450       context: context,
451       builder: (BuildContext context) {
452         return DataControl();
453       },
454     );
455   }
456
457   Future<void> _showDataSecurityDialog() {
458     showDialog(
459       context: context,
460       builder: (BuildContext context) {
461         return DataSecurity();
462       },
463     );
464   }
465
466   Future<void> _showDataIntegrityDialog() {
467     showDialog(
468       context: context,
469       builder: (BuildContext context) {
470         return DataIntegrity();
471       },
472     );
473   }
474
475   Future<void> _showDataAvailabilityDialog() {
476     showDialog(
477       context: context,
478       builder: (BuildContext context) {
479         return DataAvailability();
480       },
481     );
482   }
483
484   Future<void> _showDataReliabilityDialog() {
485     showDialog(
486       context: context,
487       builder: (BuildContext context) {
488         return DataReliability();
489       },
490     );
491   }
492
493   Future<void> _showDataAccuracyDialog() {
494     showDialog(
495       context: context,
496       builder: (BuildContext context) {
497         return DataAccuracy();
498       },
499     );
500   }
501
502   Future<void> _showDataCompletenessDialog() {
503     showDialog(
504       context: context,
505       builder: (BuildContext context) {
506         return DataCompleteness();
507       },
508     );
509   }
510
511   Future<void> _showDataConsistencyDialog() {
512     showDialog(
513       context: context,
514       builder: (BuildContext context) {
515         return DataConsistency();
516       },
517     );
518   }
519
520   Future<void> _showDataValidityDialog() {
521     showDialog(
522       context: context,
523       builder: (BuildContext context) {
524         return DataValidity();
525       },
526     );
527   }
528
529   Future<void> _showDataTrustworthinessDialog() {
530     showDialog(
531       context: context,
532       builder: (BuildContext context) {
533         return DataTrustworthiness();
534       },
535     );
536   }
537
538   Future<void> _showDataConfidentialityDialog() {
539     showDialog(
540       context: context,
541       builder: (BuildContext context) {
542         return DataConfidentiality();
543       },
544     );
545   }
546
547   Future<void> _showDataIntegrityDialog() {
548     showDialog(
549       context: context,
550       builder: (BuildContext context) {
551         return DataIntegrity();
552       },
553     );
554   }
555
556   Future<void> _showDataAvailabilityDialog() {
557     showDialog(
558       context: context,
559       builder: (BuildContext context) {
560         return DataAvailability();
561       },
562     );
563   }
564
565   Future<void> _showDataReliabilityDialog() {
566     showDialog(
567       context: context,
568       builder: (BuildContext context) {
569         return DataReliability();
570       },
571     );
572   }
573
574   Future<void> _showDataAccuracyDialog() {
575     showDialog(
576       context: context,
577       builder: (BuildContext context) {
578         return DataAccuracy();
579       },
580     );
581   }
582
583   Future<void> _showDataCompletenessDialog() {
584     showDialog(
585       context: context,
586       builder: (BuildContext context) {
587         return DataCompleteness();
588       },
589     );
590   }
591
592   Future<void> _showDataConsistencyDialog() {
593     showDialog(
594       context: context,
595       builder: (BuildContext context) {
596         return DataConsistency();
597       },
598     );
599   }
600
601   Future<void> _showDataValidityDialog() {
602     showDialog(
603       context: context,
604       builder: (BuildContext context) {
605         return DataValidity();
606       },
607     );
608   }
609
610   Future<void> _showDataTrustworthinessDialog() {
611     showDialog(
612       context: context,
613       builder: (BuildContext context) {
614         return DataTrustworthiness();
615       },
616     );
617   }
618
619   Future<void> _showDataConfidentialityDialog() {
620     showDialog(
621       context: context,
622       builder: (BuildContext context) {
623         return DataConfidentiality();
624       },
625     );
626   }
627
628   Future<void> _showDataIntegrityDialog() {
629     showDialog(
630       context: context,
631       builder: (BuildContext context) {
632         return DataIntegrity();
633       },
634     );
635   }
636
637   Future<void> _showDataAvailabilityDialog() {
638     showDialog(
639       context: context,
640       builder: (BuildContext context) {
641         return DataAvailability();
642       },
643     );
644   }
645
646   Future<void> _showDataReliabilityDialog() {
647     showDialog(
648       context: context,
649       builder: (BuildContext context) {
650         return DataReliability();
651       },
652     );
653   }
654
655   Future<void> _showDataAccuracyDialog() {
656     showDialog(
657       context: context,
658       builder: (BuildContext context) {
659         return DataAccuracy();
660       },
661     );
662   }
663
664   Future<void> _showDataCompletenessDialog() {
665     showDialog(
666       context: context,
667       builder: (BuildContext context) {
668         return DataCompleteness();
669       },
670     );
671   }
672
673   Future<void> _showDataConsistencyDialog() {
674     showDialog(
675       context: context,
676       builder: (BuildContext context) {
677         return DataConsistency();
678       },
679     );
680   }
681
682   Future<void> _showDataValidityDialog() {
683     showDialog(
684       context: context,
685       builder: (BuildContext context) {
686         return DataValidity();
687       },
688     );
689   }
690
691   Future<void> _showDataTrustworthinessDialog() {
692     showDialog(
693       context: context,
694       builder: (BuildContext context) {
695         return DataTrustworthiness();
696       },
697     );
698   }
699
700   Future<void> _showDataConfidentialityDialog() {
701     showDialog(
702       context: context,
703       builder: (BuildContext context) {
704         return DataConfidentiality();
705       },
706     );
707   }
708
709   Future<void> _showDataIntegrityDialog() {
710     showDialog(
711       context: context,
712       builder: (BuildContext context) {
713         return DataIntegrity();
714       },
715     );
716   }
717
718   Future<void> _showDataAvailabilityDialog() {
719     showDialog(
720       context: context,
721       builder: (BuildContext context) {
722         return DataAvailability();
723       },
724     );
725   }
726
727   Future<void> _showDataReliabilityDialog() {
728     showDialog(
729       context: context,
730       builder: (BuildContext context) {
731         return DataReliability();
732       },
733     );
734   }
735
736   Future<void> _showDataAccuracyDialog() {
737     showDialog(
738       context: context,
739       builder: (BuildContext context) {
740         return DataAccuracy();
741       },
742     );
743   }
744
745   Future<void> _showDataCompletenessDialog() {
746     showDialog(
747       context: context,
748       builder: (BuildContext context) {
749         return DataCompleteness();
750       },
751     );
752   }
753
754   Future<void> _showDataConsistencyDialog() {
755     showDialog(
756       context: context,
757       builder: (BuildContext context) {
758         return DataConsistency();
759       },
760     );
761   }
762
763   Future<void> _showDataValidityDialog() {
764     showDialog(
765       context: context,
766       builder: (BuildContext context) {
767         return DataValidity();
768       },
769     );
770   }
771
772   Future<void> _showDataTrustworthinessDialog() {
773     showDialog(
774       context: context,
775       builder: (BuildContext context) {
776         return DataTrustworthiness();
777       },
778     );
779   }
780
781   Future<void> _showDataConfidentialityDialog() {
782     showDialog(
783       context: context,
784       builder: (BuildContext context) {
785         return DataConfidentiality();
786       },
787     );
788   }
789
790   Future<void> _showDataIntegrityDialog() {
791     showDialog(
792       context: context,
793       builder: (BuildContext context) {
794         return DataIntegrity();
795       },
796     );
797   }
798
799   Future<void> _showDataAvailabilityDialog() {
800     showDialog(
801       context: context,
802       builder: (BuildContext context) {
803         return DataAvailability();
804       },
805     );
806   }
807
808   Future<void> _showDataReliabilityDialog() {
809     showDialog(
810       context: context,
811       builder: (BuildContext context) {
812         return DataReliability();
813       },
814     );
815   }
816
817   Future<void> _showDataAccuracyDialog() {
818     showDialog(
819       context: context,
820       builder: (BuildContext context) {
821         return DataAccuracy();
822       },
823     );
824   }
825
826   Future<void> _showDataCompletenessDialog() {
827     showDialog(
828       context: context,
829       builder: (BuildContext context) {
830         return DataCompleteness();
831       },
832     );
833   }
834
835   Future<void> _showDataConsistencyDialog() {
836     showDialog(
837       context: context,
838       builder: (BuildContext context) {
839         return DataConsistency();
840       },
841     );
842   }
843
844   Future<void> _showDataValidityDialog() {
845     showDialog(
846       context: context,
847       builder: (BuildContext context) {
848         return DataValidity();
849       },
850     );
851   }
852
853   Future<void> _showDataTrustworthinessDialog() {
854     showDialog(
855       context: context,
856       builder: (BuildContext context) {
857         return DataTrustworthiness();
858       },
859     );
860   }
861
862   Future<void> _showDataConfidentialityDialog() {
863     showDialog(
864       context: context,
865       builder: (BuildContext context) {
866         return DataConfidentiality();
867       },
868     );
869   }
870
871   Future<void> _showDataIntegrityDialog() {
872     showDialog(
873       context: context,
874       builder: (BuildContext context) {
875         return DataIntegrity();
876       },
877     );
878   }
879
880   Future<void> _showDataAvailabilityDialog() {
881     showDialog(
882       context: context,
883       builder: (BuildContext context) {
884         return DataAvailability();
885       },
886     );
887   }
888
889   Future<void> _showDataReliabilityDialog() {
890     showDialog(
891       context: context,
892       builder: (BuildContext context) {
893         return DataReliability();
894       },
895     );
896   }
897
898   Future<void> _showDataAccuracyDialog() {
899     showDialog(
900       context: context,
901       builder: (BuildContext context) {
902         return DataAccuracy();
903       },
904     );
905   }
906
907   Future<void> _showDataCompletenessDialog() {
908     showDialog(
909       context: context,
910       builder: (BuildContext context) {
911         return DataCompleteness();
912       },
913     );
914   }
915
916   Future<void> _showDataConsistencyDialog() {
917     showDialog(
918       context: context,
919       builder: (BuildContext context) {
920         return DataConsistency();
921       },
922     );
923   }
924
925   Future<void> _showDataValidityDialog() {
926     showDialog(
927       context: context,
928       builder: (BuildContext context) {
929         return DataValidity();
930       },
931     );
932   }
933
934   Future<void> _showDataTrustworthinessDialog() {
935     showDialog(
936       context: context,
937       builder: (BuildContext context) {
938         return DataTrustworthiness();
939       },
940     );
941   }
942
943   Future<void> _showDataConfidentialityDialog() {
944     showDialog(
945       context: context,
946       builder: (BuildContext context) {
947         return DataConfidentiality();
948       },
949     );
950   }
951
952   Future<void> _showDataIntegrityDialog() {
953     showDialog(
954       context: context,
955       builder: (BuildContext context) {
956         return DataIntegrity();
957       },
958     );
959   }
960
961   Future<void> _showDataAvailabilityDialog() {
962     showDialog(
963       context: context,
964       builder: (BuildContext context) {
965         return DataAvailability();
966       },
967     );
968   }
969
970   Future<void> _showDataReliabilityDialog() {
971     showDialog(
972       context: context,
973       builder: (BuildContext context) {
974         return DataReliability();
975       },
976     );
977   }
978
979   Future<void> _showDataAccuracyDialog() {
980     showDialog(
981       context: context,
982       builder: (BuildContext context) {
983         return DataAccuracy();
984       },
985     );
986   }
987
988   Future<void> _showDataCompletenessDialog() {
989     showDialog(
990       context: context,
991       builder: (BuildContext context) {
992         return DataCompleteness();
993       },
994     );
995   }
996
997   Future<void> _showDataConsistencyDialog() {
998     showDialog(
999       context: context,
1000       builder: (BuildContext context) {
1001         return DataConsistency();
1002       },
1003     );
1004   }
1005
1006   Future<void> _showDataValidityDialog() {
1007     showDialog(
1008       context: context,
1009       builder: (BuildContext context) {
1010         return DataValidity();
1011       },
1012     );
1013   }
1014
1015   Future<void> _showDataTrustworthinessDialog() {
1016     showDialog(
1017       context: context,
1018       builder: (BuildContext context) {
1019         return DataTrustworthiness();
1020       },
1021     );
1022   }
1023
1024   Future<void> _showDataConfidentialityDialog() {
1025     showDialog(
1026       context: context,
1027       builder: (BuildContext context) {
1028         return DataConfidentiality();
1029       },
1030     );
1031   }
1032
1033   Future<void> _showDataIntegrityDialog() {
1034     showDialog(
1035       context: context,
1036       builder: (BuildContext context) {
1037         return DataIntegrity();
1038       },
1039     );
1040   }
1041
1042   Future<void> _showDataAvailabilityDialog() {
1043     showDialog(
1044       context: context,
1045       builder: (BuildContext context) {
1046         return DataAvailability();
1047       },
1048     );
1049   }
1050
1051   Future<void> _showDataReliabilityDialog() {
1052     showDialog(
1053       context: context,
1054       builder: (BuildContext context) {
1055         return DataReliability();
1056       },
1057     );
1058   }
1059
1060   Future<void> _showDataAccuracyDialog() {
1061     showDialog(
1062       context: context,
1063       builder: (BuildContext context) {
1064         return DataAccuracy();
1065       },
1066     );
1067   }
1068
1069   Future<void> _showDataCompletenessDialog() {
1070     showDialog(
1071       context: context,
1072       builder: (BuildContext context) {
1073         return DataCompleteness();
1074       },
1075     );
1076   }
1077
1078   Future<void> _showDataConsistencyDialog() {
1079     showDialog(
1080       context: context,
1081       builder: (BuildContext context) {
1082         return DataConsistency();
1083       },
1084     );
1085   }
1086
1087   Future<void> _showDataValidityDialog() {
1088     showDialog(
1089       context: context,
1090       builder: (BuildContext context) {
1091         return DataValidity();
1092       },
1093     );
1094   }
1095
1096   Future<void> _showDataTrustworthinessDialog() {
1097     showDialog(
1098       context: context,
1099       builder: (BuildContext context) {
1100         return DataTrustworthiness();
1101       },
1102     );
1103   }
1104
1105   Future<void> _showDataConfidentialityDialog() {
1106     showDialog(
1107       context: context,
1108       builder: (BuildContext context) {
1109         return DataConfidentiality();
1110       },
1111     );
1112   }
1113
1114   Future<void> _showDataIntegrityDialog() {
1115     showDialog(
1116       context: context,
1117       builder: (BuildContext context) {
1118         return DataIntegrity();
1119       },
1120     );
1121   }
1122
1123   Future<void> _showDataAvailabilityDialog() {
1124     showDialog(
1125       context: context,
1126       builder: (BuildContext context) {
1127         return DataAvailability();
1128       },
1129     );
1130   }
1131
1132   Future<void> _showDataReliabilityDialog() {
1133     showDialog(
1134       context: context,
1135       builder: (BuildContext context) {
1136         return DataReliability();
1137       },
1138     );
1139   }
1140
1141   Future<void> _showDataAccuracyDialog() {
1142     showDialog(
1143       context: context,
1144       builder: (BuildContext context) {
1145         return DataAccuracy();
1146       },
1147     );
1148   }
1149
1150   Future<void> _showDataCompletenessDialog() {
1151     showDialog(
1152       context: context,
1153       builder: (BuildContext context) {
1154         return DataCompleteness();
1155       },
1156     );
1157   }
1158
1159   Future<void> _showDataConsistencyDialog() {
1160     showDialog(
1161       context: context,
1162       builder: (BuildContext context) {
1163         return DataConsistency();
1164       },
1165     );
1166   }
1167
1168   Future<void> _showDataValidityDialog() {
1169     showDialog(
1170       context: context,
1171       builder: (BuildContext context) {
1172         return DataValidity();
1173       },
1174     );
1175   }
1176
1177   Future<void> _showDataTrustworthinessDialog() {
1178     showDialog(
1179       context: context,
1180       builder: (BuildContext context) {
1181         return DataTrustworthiness();
1182       },
1183     );
1184   }
1185
1186   Future<void> _showDataConfidentialityDialog() {
1187     showDialog(
1188       context: context,
1189       builder: (BuildContext context) {
1190         return DataConfidentiality();
1191       },
1192     );
1193   }
1194
1195   Future<void> _showDataIntegrityDialog() {
1196     showDialog(
1197       context: context,
1198       builder: (BuildContext context) {
1199         return DataIntegrity();
1200       },
1201     );
1202   }
1203
1204   Future<void> _showDataAvailabilityDialog() {
1205     showDialog(
1206       context: context,
1207       builder: (BuildContext context) {
1208         return DataAvailability();
1209       },
1210     );
1211   }
1212
1213   Future<void> _showDataReliabilityDialog() {
1214     showDialog(
1215       context: context,
1216       builder: (BuildContext context) {
1217         return DataReliability();
1218       },
1219     );
1220   }
1221
1222   Future<void> _showDataAccuracyDialog() {
1223     showDialog(
1224       context: context,
1225       builder: (BuildContext context) {
1226         return DataAccuracy();
1227       },
1228     );
1229   }
1230
1231   Future<void> _showDataCompletenessDialog() {
1232     showDialog(
1233       context: context,
1234       builder: (BuildContext context) {
1235         return DataCompleteness();
1236       },
1237     );
1238   }
1239
1240   Future<void> _showDataConsistencyDialog() {
1241     showDialog(
1242       context: context,
1243       builder: (BuildContext context) {
1244         return DataConsistency();
1245       },
1246     );
1247   }
1248
1249   Future<void> _showDataValidityDialog() {
1250     showDialog(
1251       context: context,
1252       builder: (BuildContext context) {
1253         return DataValidity();
1254       },
1255     );
1256   }
1257
1258   Future<void> _showDataTrustworthinessDialog() {
1259     showDialog(
1260       context: context,
1261       builder: (BuildContext context) {
1262         return DataTrustworthiness();
1263       },
1264     );
1265   }
1266
1267   Future<void> _showDataConfidentialityDialog() {
1268     showDialog(
1269       context: context,
1270       builder: (BuildContext context) {
1271         return DataConfidentiality();
1272       },
1273     );
1274   }
1275
1276   Future<void> _showDataIntegrityDialog() {
1277     showDialog(
1278       context: context,
1279       builder: (BuildContext context) {
1280         return DataIntegrity();
1281       },
1282     );
1283   }
1284
1285   Future<void> _showDataAvailabilityDialog() {
1286     showDialog(
1287       context: context,
1288       builder: (BuildContext context) {
1289         return DataAvailability();
1290       },
1291     );
1292   }
1293
1294   Future<void> _showDataReliabilityDialog() {
1295     showDialog(
1296       context: context,
1297       builder: (BuildContext context) {
1298         return DataReliability();
1299       },
1300     );
1301   }
1302
1303   Future<void> _showDataAccuracyDialog() {
1304     showDialog(
1305       context: context,
1306       builder: (BuildContext context) {
1307         return DataAccuracy();
1308       },
1309     );
1310   }
1311
1312   Future<void
```

## E. detail.dart

### 3. Screenshoot Source Code Layanan

#### A. tampil.dart

```
import 'package:flutter/material.dart';
import 'package:ui_utils/ui_utils.dart';
import 'package:ui_utils/ui_utils.dart';
import 'package:ui_utils/ui_utils.dart';
import 'package:ui_utils/ui_utils.dart';
import 'package:ui_utils/ui_utils.dart';

class TampilLayanan extends StatelessWidget {
  const TampilLayanan({super.key});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: 'Tampil Layanan',
      ),
      body: ListView.builder(
        itemCount: 10,
        itemBuilder: (context, index) {
          return _buildLayanan(context, index);
        },
      ),
    );
  }

  Widget _buildLayanan(BuildContext context, int index) {
    return Card(
      child: ListTile(
        title: 'Layanan $index',
        subtitle: 'Detail Layanan $index',
        trailing: IconButton(
          icon: Icons.edit,
          onPressed: () {
            // Edit service logic
          },
        ),
      ),
    );
  }
}
```

#### B. tambah.dart

```
import 'package:flutter/material.dart';
import 'package:ui_utils/ui_utils.dart';
import 'package:ui_utils/ui_utils.dart';
import 'package:ui_utils/ui_utils.dart';
import 'package:ui_utils/ui_utils.dart';
import 'package:ui_utils/ui_utils.dart';

class TambahLayanan extends StatelessWidget {
  const TambahLayanan({super.key});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: 'Tambah Layanan',
      ),
      body: SingleChildScrollView(
        child: Column(
          children: [
            TextFormField(
              decoration: InputDecoration(
                labelText: 'Nama Layanan',
              ),
            ),
            TextFormField(
              decoration: InputDecoration(
                labelText: 'Detail Layanan',
              ),
            ),
            ElevatedButton(
              onPressed: _addService,
              child: Text('Tambah'),
            ),
          ],
        ),
      ),
    );
  }

  void _addService() {
    // Add service logic
  }
}
```

### C. detail.dart

```
1 import 'package:flutter/material.dart';
2 import 'package:uas_visual3_2210010428/layanan/layanan.dart';
3
4 class DetailLayanan extends StatelessWidget {
5   final Layanan Lyn;
6
7   const DetailLayanan({super.key, required this.Lyn});
8
9   @override
10  Widget build(BuildContext context) {
11    return Scaffold(
12      appBar: AppBar(
13        title: Text(Lyn.namalayanan),
14        centerTitle: true,
15        backgroundColor: Colors.blueGrey.shade100,
16      ),
17      body: Center(
18        child: Column(
19          children: [
20            const SizedBox(height: 20),
21            Text("No Layanan : ${Lyn.no_layanan}"),
22            const SizedBox(
23              height: 5,
24            ),
25            Text("Nama Layanan : ${Lyn.namalayanan}"),
26            const SizedBox(
27              height: 5,
28            ),
29            Text("Deskripsi : ${Lyn.deskripsi}"),
30            const SizedBox(
31              height: 5,
32            ),
33            Text("Harga : ${Lyn.harga}"),
34            const SizedBox(
35              height: 5,
36            ),
37            Text("Durasi : ${Lyn.durasi}"),
38            const SizedBox(
39              height: 5,
40            ),
41            const SizedBox(
42              height: 5,
43            ),
44            Text("Kategori : ${Lyn.kategori}"),
45            const SizedBox(
46              height: 5,
47            ),
48          ],
49        ),
50      ),
51    );
52  }
53 }
54
```

### D. layanan.dart

```
1 class Layanan {
2   final String no_layanan;
3   final String namalayanan;
4   final String deskripsi;
5   final String harga;
6   final String durasi;
7   final String kategori;
8
9   Layanan({
10     required this.no_layanan,
11     required this.namalayanan,
12     required this.deskripsi,
13     required this.harga,
14     required this.durasi,
15     required this.kategori,
16   });
17
18   factory Layanan.fromMap(Map<String, dynamic> map) {
19     return Layanan(
20       no_layanan: map['no_layanan'],
21       namalayanan: map['namalayanan'],
22       deskripsi: map['deskripsi'],
23       harga: map['harga'],
24       durasi: map['durasi'],
25       kategori: map['kategori']);
26   }
27
28   Map<String, dynamic> toMap() {
29     return {
30       'no_layanan': no_layanan,
31       'namalayanan': namalayanan,
32       'deskripsi': deskripsi,
33       'harga': harga,
34       'durasi': durasi,
35       'kategori': kategori,
36     };
37   }
38 }
39
```

## E. edit.dart

```
1 import 'package:flutter/material.dart';
2 import 'package:http/http.dart' as http;
3 import 'package:uuid/uuid.dart';
4
5 final keystore = SharedPreferences.getInstance();
6 var _username = TextEditingController();
7 var _password = TextEditingController();
8 var _username2 = TextEditingController();
9 var _password2 = TextEditingController();
10 var _token = TextEditingController();
11 var _token2 = TextEditingController();
12
13 class _LoginPageState extends State<LoginPage> {
14   final _loginUrl = 'http://localhost:8080/api/login';
15   void _login() async {
16     try {
17       final response = await http.post(_loginUrl, body: {
18         'username': _username.text,
19         'password': _password.text,
20       });
21       if (response.statusCode == 200) {
22         final token = json.decode(response.body)['token'];
23         _token.text = token;
24         _token2.text = token;
25         _username.clear();
26         _password.clear();
27         _username2.clear();
28         _password2.clear();
29         _token2.clear();
30         _token.clear();
31         _token2.clear();
32       } else {
33         _token.clear();
34         _token2.clear();
35       }
36     } catch (e) {
37       _token.clear();
38       _token2.clear();
39     }
40   }
41
42   void _login2() async {
43     try {
44       final response = await http.post(_loginUrl, body: {
45         'username': _username2.text,
46         'password': _password2.text,
47       });
48       if (response.statusCode == 200) {
49         final token = json.decode(response.body)['token'];
50         _token2.text = token;
51         _token2.clear();
52         _username2.clear();
53         _password2.clear();
54         _token2.clear();
55       } else {
56         _token2.clear();
57       }
58     } catch (e) {
59       _token2.clear();
60     }
61   }
62
63   void _login3() async {
64     try {
65       final response = await http.post(_loginUrl, body: {
66         'username': _username2.text,
67         'password': _password2.text,
68       });
69       if (response.statusCode == 200) {
70         final token = json.decode(response.body)['token'];
71         _token2.text = token;
72         _token2.clear();
73         _username2.clear();
74         _password2.clear();
75         _token2.clear();
76       } else {
77         _token2.clear();
78       }
79     } catch (e) {
80       _token2.clear();
81     }
82   }
83
84   void _login4() async {
85     try {
86       final response = await http.post(_loginUrl, body: {
87         'username': _username2.text,
88         'password': _password2.text,
89       });
90       if (response.statusCode == 200) {
91         final token = json.decode(response.body)['token'];
92         _token2.text = token;
93         _token2.clear();
94         _username2.clear();
95         _password2.clear();
96         _token2.clear();
97       } else {
98         _token2.clear();
99       }
100     } catch (e) {
101       _token2.clear();
102     }
103   }
104
105   void _login5() async {
106     try {
107       final response = await http.post(_loginUrl, body: {
108         'username': _username2.text,
109         'password': _password2.text,
110       });
111       if (response.statusCode == 200) {
112         final token = json.decode(response.body)['token'];
113         _token2.text = token;
114         _token2.clear();
115         _username2.clear();
116         _password2.clear();
117         _token2.clear();
118       } else {
119         _token2.clear();
120       }
121     } catch (e) {
122       _token2.clear();
123     }
124   }
125
126   void _login6() async {
127     try {
128       final response = await http.post(_loginUrl, body: {
129         'username': _username2.text,
130         'password': _password2.text,
131       });
132       if (response.statusCode == 200) {
133         final token = json.decode(response.body)['token'];
134         _token2.text = token;
135         _token2.clear();
136         _username2.clear();
137         _password2.clear();
138         _token2.clear();
139       } else {
140         _token2.clear();
141       }
142     } catch (e) {
143       _token2.clear();
144     }
145   }
146
147   void _login7() async {
148     try {
149       final response = await http.post(_loginUrl, body: {
150         'username': _username2.text,
151         'password': _password2.text,
152       });
153       if (response.statusCode == 200) {
154         final token = json.decode(response.body)['token'];
155         _token2.text = token;
156         _token2.clear();
157         _username2.clear();
158         _password2.clear();
159         _token2.clear();
160       } else {
161         _token2.clear();
162       }
163     } catch (e) {
164       _token2.clear();
165     }
166   }
167
168   void _login8() async {
169     try {
170       final response = await http.post(_loginUrl, body: {
171         'username': _username2.text,
172         'password': _password2.text,
173       });
174       if (response.statusCode == 200) {
175         final token = json.decode(response.body)['token'];
176         _token2.text = token;
177         _token2.clear();
178         _username2.clear();
179         _password2.clear();
180         _token2.clear();
181       } else {
182         _token2.clear();
183       }
184     } catch (e) {
185       _token2.clear();
186     }
187   }
188
189   void _login9() async {
190     try {
191       final response = await http.post(_loginUrl, body: {
192         'username': _username2.text,
193         'password': _password2.text,
194       });
195       if (response.statusCode == 200) {
196         final token = json.decode(response.body)['token'];
197         _token2.text = token;
198         _token2.clear();
199         _username2.clear();
200         _password2.clear();
201         _token2.clear();
202       } else {
203         _token2.clear();
204       }
205     } catch (e) {
206       _token2.clear();
207     }
208   }
209
210   void _login10() async {
211     try {
212       final response = await http.post(_loginUrl, body: {
213         'username': _username2.text,
214         'password': _password2.text,
215       });
216       if (response.statusCode == 200) {
217         final token = json.decode(response.body)['token'];
218         _token2.text = token;
219         _token2.clear();
220         _username2.clear();
221         _password2.clear();
222         _token2.clear();
223       } else {
224         _token2.clear();
225       }
226     } catch (e) {
227       _token2.clear();
228     }
229   }
230
231   void _login11() async {
232     try {
233       final response = await http.post(_loginUrl, body: {
234         'username': _username2.text,
235         'password': _password2.text,
236       });
237       if (response.statusCode == 200) {
238         final token = json.decode(response.body)['token'];
239         _token2.text = token;
240         _token2.clear();
241         _username2.clear();
242         _password2.clear();
243         _token2.clear();
244       } else {
245         _token2.clear();
246       }
247     } catch (e) {
248       _token2.clear();
249     }
250   }
251
252   void _login12() async {
253     try {
254       final response = await http.post(_loginUrl, body: {
255         'username': _username2.text,
256         'password': _password2.text,
257       });
258       if (response.statusCode == 200) {
259         final token = json.decode(response.body)['token'];
260         _token2.text = token;
261         _token2.clear();
262         _username2.clear();
263         _password2.clear();
264         _token2.clear();
265       } else {
266         _token2.clear();
267       }
268     } catch (e) {
269       _token2.clear();
270     }
271   }
272
273   void _login13() async {
274     try {
275       final response = await http.post(_loginUrl, body: {
276         'username': _username2.text,
277         'password': _password2.text,
278       });
279       if (response.statusCode == 200) {
280         final token = json.decode(response.body)['token'];
281         _token2.text = token;
282         _token2.clear();
283         _username2.clear();
284         _password2.clear();
285         _token2.clear();
286       } else {
287         _token2.clear();
288       }
289     } catch (e) {
290       _token2.clear();
291     }
292   }
293
294   void _login14() async {
295     try {
296       final response = await http.post(_loginUrl, body: {
297         'username': _username2.text,
298         'password': _password2.text,
299       });
300       if (response.statusCode == 200) {
301         final token = json.decode(response.body)['token'];
302         _token2.text = token;
303         _token2.clear();
304         _username2.clear();
305         _password2.clear();
306         _token2.clear();
307       } else {
308         _token2.clear();
309       }
310     } catch (e) {
311       _token2.clear();
312     }
313   }
314
315   void _login15() async {
316     try {
317       final response = await http.post(_loginUrl, body: {
318         'username': _username2.text,
319         'password': _password2.text,
320       });
321       if (response.statusCode == 200) {
322         final token = json.decode(response.body)['token'];
323         _token2.text = token;
324         _token2.clear();
325         _username2.clear();
326         _password2.clear();
327         _token2.clear();
328       } else {
329         _token2.clear();
330       }
331     } catch (e) {
332       _token2.clear();
333     }
334   }
335
336   void _login16() async {
337     try {
338       final response = await http.post(_loginUrl, body: {
339         'username': _username2.text,
340         'password': _password2.text,
341       });
342       if (response.statusCode == 200) {
343         final token = json.decode(response.body)['token'];
344         _token2.text = token;
345         _token2.clear();
346         _username2.clear();
347         _password2.clear();
348         _token2.clear();
349       } else {
350         _token2.clear();
351       }
352     } catch (e) {
353       _token2.clear();
354     }
355   }
356
357   void _login17() async {
358     try {
359       final response = await http.post(_loginUrl, body: {
360         'username': _username2.text,
361         'password': _password2.text,
362       });
363       if (response.statusCode == 200) {
364         final token = json.decode(response.body)['token'];
365         _token2.text = token;
366         _token2.clear();
367         _username2.clear();
368         _password2.clear();
369         _token2.clear();
370       } else {
371         _token2.clear();
372       }
373     } catch (e) {
374       _token2.clear();
375     }
376   }
377
378   void _login18() async {
379     try {
380       final response = await http.post(_loginUrl, body: {
381         'username': _username2.text,
382         'password': _password2.text,
383       });
384       if (response.statusCode == 200) {
385         final token = json.decode(response.body)['token'];
386         _token2.text = token;
387         _token2.clear();
388         _username2.clear();
389         _password2.clear();
390         _token2.clear();
391       } else {
392         _token2.clear();
393       }
394     } catch (e) {
395       _token2.clear();
396     }
397   }
398
399   void _login19() async {
400     try {
401       final response = await http.post(_loginUrl, body: {
402         'username': _username2.text,
403         'password': _password2.text,
404       });
405       if (response.statusCode == 200) {
406         final token = json.decode(response.body)['token'];
407         _token2.text = token;
408         _token2.clear();
409         _username2.clear();
410         _password2.clear();
411         _token2.clear();
412       } else {
413         _token2.clear();
414       }
415     } catch (e) {
416       _token2.clear();
417     }
418   }
419
420   void _login20() async {
421     try {
422       final response = await http.post(_loginUrl, body: {
423         'username': _username2.text,
424         'password': _password2.text,
425       });
426       if (response.statusCode == 200) {
427         final token = json.decode(response.body)['token'];
428         _token2.text = token;
429         _token2.clear();
430         _username2.clear();
431         _password2.clear();
432         _token2.clear();
433       } else {
434         _token2.clear();
435       }
436     } catch (e) {
437       _token2.clear();
438     }
439   }
440
441   void _login21() async {
442     try {
443       final response = await http.post(_loginUrl, body: {
444         'username': _username2.text,
445         'password': _password2.text,
446       });
447       if (response.statusCode == 200) {
448         final token = json.decode(response.body)['token'];
449         _token2.text = token;
450         _token2.clear();
451         _username2.clear();
452         _password2.clear();
453         _token2.clear();
454       } else {
455         _token2.clear();
456       }
457     } catch (e) {
458       _token2.clear();
459     }
460   }
461
462   void _login22() async {
463     try {
464       final response = await http.post(_loginUrl, body: {
465         'username': _username2.text,
466         'password': _password2.text,
467       });
468       if (response.statusCode == 200) {
469         final token = json.decode(response.body)['token'];
470         _token2.text = token;
471         _token2.clear();
472         _username2.clear();
473         _password2.clear();
474         _token2.clear();
475       } else {
476         _token2.clear();
477       }
478     } catch (e) {
479       _token2.clear();
480     }
481   }
482
483   void _login23() async {
484     try {
485       final response = await http.post(_loginUrl, body: {
486         'username': _username2.text,
487         'password': _password2.text,
488       });
489       if (response.statusCode == 200) {
490         final token = json.decode(response.body)['token'];
491         _token2.text = token;
492         _token2.clear();
493         _username2.clear();
494         _password2.clear();
495         _token2.clear();
496       } else {
497         _token2.clear();
498       }
499     } catch (e) {
500       _token2.clear();
501     }
502   }
503
504   void _login24() async {
505     try {
506       final response = await http.post(_loginUrl, body: {
507         'username': _username2.text,
508         'password': _password2.text,
509       });
510       if (response.statusCode == 200) {
511         final token = json.decode(response.body)['token'];
512         _token2.text = token;
513         _token2.clear();
514         _username2.clear();
515         _password2.clear();
516         _token2.clear();
517       } else {
518         _token2.clear();
519       }
520     } catch (e) {
521       _token2.clear();
522     }
523   }
524
525   void _login25() async {
526     try {
527       final response = await http.post(_loginUrl, body: {
528         'username': _username2.text,
529         'password': _password2.text,
530       });
531       if (response.statusCode == 200) {
532         final token = json.decode(response.body)['token'];
533         _token2.text = token;
534         _token2.clear();
535         _username2.clear();
536         _password2.clear();
537         _token2.clear();
538       } else {
539         _token2.clear();
540       }
541     } catch (e) {
542       _token2.clear();
543     }
544   }
545
546   void _login26() async {
547     try {
548       final response = await http.post(_loginUrl, body: {
549         'username': _username2.text,
550         'password': _password2.text,
551       });
552       if (response.statusCode == 200) {
553         final token = json.decode(response.body)['token'];
554         _token2.text = token;
555         _token2.clear();
556         _username2.clear();
557         _password2.clear();
558         _token2.clear();
559       } else {
560         _token2.clear();
561       }
562     } catch (e) {
563       _token2.clear();
564     }
565   }
566
567   void _login27() async {
568     try {
569       final response = await http.post(_loginUrl, body: {
570         'username': _username2.text,
571         'password': _password2.text,
572       });
573       if (response.statusCode == 200) {
574         final token = json.decode(response.body)['token'];
575         _token2.text = token;
576         _token2.clear();
577         _username2.clear();
578         _password2.clear();
579         _token2.clear();
580       } else {
581         _token2.clear();
582       }
583     } catch (e) {
584       _token2.clear();
585     }
586   }
587
588   void _login28() async {
589     try {
590       final response = await http.post(_loginUrl, body: {
591         'username': _username2.text,
592         'password': _password2.text,
593       });
594       if (response.statusCode == 200) {
595         final token = json.decode(response.body)['token'];
596         _token2.text = token;
597         _token2.clear();
598         _username2.clear();
599         _password2.clear();
600         _token2.clear();
601       } else {
602         _token2.clear();
603       }
604     } catch (e) {
605       _token2.clear();
606     }
607   }
608
609   void _login29() async {
610     try {
611       final response = await http.post(_loginUrl, body: {
612         'username': _username2.text,
613         'password': _password2.text,
614       });
615       if (response.statusCode == 200) {
616         final token = json.decode(response.body)['token'];
617         _token2.text = token;
618         _token2.clear();
619         _username2.clear();
620         _password2.clear();
621         _token2.clear();
622       } else {
623         _token2.clear();
624       }
625     } catch (e) {
626       _token2.clear();
627     }
628   }
629
630   void _login30() async {
631     try {
632       final response = await http.post(_loginUrl, body: {
633         'username': _username2.text,
634         'password': _password2.text,
635       });
636       if (response.statusCode == 200) {
637         final token = json.decode(response.body)['token'];
638         _token2.text = token;
639         _token2.clear();
640         _username2.clear();
641         _password2.clear();
642         _token2.clear();
643       } else {
644         _token2.clear();
645       }
646     } catch (e) {
647       _token2.clear();
648     }
649   }
650
651   void _login31() async {
652     try {
653       final response = await http.post(_loginUrl, body: {
654         'username': _username2.text,
655         'password': _password2.text,
656       });
657       if (response.statusCode == 200) {
658         final token = json.decode(response.body)['token'];
659         _token2.text = token;
660         _token2.clear();
661         _username2.clear();
662         _password2.clear();
663         _token2.clear();
664       } else {
665         _token2.clear();
666       }
667     } catch (e) {
668       _token2.clear();
669     }
670   }
671
672   void _login32() async {
673     try {
674       final response = await http.post(_loginUrl, body: {
675         'username': _username2.text,
676         'password': _password2.text,
677       });
678       if (response.statusCode == 200) {
679         final token = json.decode(response.body)['token'];
680         _token2.text = token;
681         _token2.clear();
682         _username2.clear();
683         _password2.clear();
684         _token2.clear();
685       } else {
686         _token2.clear();
687       }
688     } catch (e) {
689       _token2.clear();
690     }
691   }
692
693   void _login33() async {
694     try {
695       final response = await http.post(_loginUrl, body: {
696         'username': _username2.text,
697         'password': _password2.text,
698       });
699       if (response.statusCode == 200) {
700         final token = json.decode(response.body)['token'];
701         _token2.text = token;
702         _token2.clear();
703         _username2.clear();
704         _password2.clear();
705         _token2.clear();
706       } else {
707         _token2.clear();
708       }
709     } catch (e) {
710       _token2.clear();
711     }
712   }
713
714   void _login34() async {
715     try {
716       final response = await http.post(_loginUrl, body: {
717         'username': _username2.text,
718         'password': _password2.text,
719       });
720       if (response.statusCode == 200) {
721         final token = json.decode(response.body)['token'];
722         _token2.text = token;
723         _token2.clear();
724         _username2.clear();
725         _password2.clear();
726         _token2.clear();
727       } else {
728         _token2.clear();
729       }
730     } catch (e) {
731       _token2.clear();
732     }
733   }
734
735   void _login35() async {
736     try {
737       final response = await http.post(_loginUrl, body: {
738         'username': _username2.text,
739         'password': _password2.text,
740       });
741       if (response.statusCode == 200) {
742         final token = json.decode(response.body)['token'];
743         _token2.text = token;
744         _token2.clear();
745         _username2.clear();
746         _password2.clear();
747         _token2.clear();
748       } else {
749         _token2.clear();
750       }
751     } catch (e) {
752       _token2.clear();
753     }
754   }
755
756   void _login36() async {
757     try {
758       final response = await http.post(_loginUrl, body: {
759         'username': _username2.text,
760         'password': _password2.text,
761       });
762       if (response.statusCode == 200) {
763         final token = json.decode(response.body)['token'];
764         _token2.text = token;
765         _token2.clear();
766         _username2.clear();
767         _password2.clear();
768         _token2.clear();
769       } else {
770         _token2.clear();
771       }
772     } catch (e) {
773       _token2.clear();
774     }
775   }
776
777   void _login37() async {
778     try {
779       final response = await http.post(_loginUrl, body: {
780         'username': _username2.text,
781         'password': _password2.text,
782       });
783       if (response.statusCode == 200) {
784         final token = json.decode(response.body)['token'];
785         _token2.text = token;
786         _token2.clear();
787         _username2.clear();
788         _password2.clear();
789         _token2.clear();
790       } else {
791         _token2.clear();
792       }
793     } catch (e) {
794       _token2.clear();
795     }
796   }
797
798   void _login38() async {
799     try {
800       final response = await http.post(_loginUrl, body: {
801         'username': _username2.text,
802         'password': _password2.text,
803       });
804       if (response.statusCode == 200) {
805         final token = json.decode(response.body)['token'];
806         _token2.text = token;
807         _token2.clear();
808         _username2.clear();
809         _password2.clear();
810         _token2.clear();
811       } else {
812         _token2.clear();
813       }
814     } catch (e) {
815       _token2.clear();
816     }
817   }
818
819   void _login39() async {
820     try {
821       final response = await http.post(_loginUrl, body: {
822         'username': _username2.text,
823         'password': _password2.text,
824       });
825       if (response.statusCode == 200) {
826         final token = json.decode(response.body)['token'];
827         _token2.text = token;
828         _token2.clear();
829         _username2.clear();
830         _password2.clear();
831         _token2.clear();
832       } else {
833         _token2.clear();
834       }
835     } catch (e) {
836       _token2.clear();
837     }
838   }
839
840   void _login40() async {
841     try {
842       final response = await http.post(_loginUrl, body: {
843         'username': _username2.text,
844         'password': _password2.text,
845       });
846       if (response.statusCode == 200) {
847         final token = json.decode(response.body)['token'];
848         _token2.text = token;
849         _token2.clear();
850         _username2.clear();
851         _password2.clear();
852         _token2.clear();
853       } else {
854         _token2.clear();
855       }
856     } catch (e) {
857       _token2.clear();
858     }
859   }
860
861   void _login41() async {
862     try {
863       final response = await http.post(_loginUrl, body: {
864         'username': _username2.text,
865         'password': _password2.text,
866       });
867       if (response.statusCode == 200) {
868         final token = json.decode(response.body)['token'];
869         _token2.text = token;
870         _token2.clear();
871         _username2.clear();
872         _password2.clear();
873         _token2.clear();
874       } else {
875         _token2.clear();
876       }
877     } catch (e) {
878       _token2.clear();
879     }
880   }
881
882   void _login42() async {
883     try {
884       final response = await http.post(_loginUrl, body: {
885         'username': _username2.text,
886         'password': _password2.text,
887       });
888       if (response.statusCode == 200) {
889         final token = json.decode(response.body)['token'];
890         _token2.text = token;
891         _token2.clear();
892         _username2.clear();
893         _password2.clear();
894         _token2.clear();
895       } else {
896         _token2.clear();
897       }
898     } catch (e) {
899       _token2.clear();
900     }
901   }
902
903   void _login43() async {
904     try {
905       final response = await http.post(_loginUrl, body: {
906         'username': _username2.text,
907         'password': _password2.text,
908       });
909       if (response.statusCode == 200) {
910         final token = json.decode(response.body)['token'];
911         _token2.text = token;
912         _token2.clear();
913         _username2.clear();
914         _password2.clear();
915         _token2.clear();
916       } else {
917         _token2.clear();
918       }
919     } catch (e) {
920       _token2.clear();
921     }
922   }
923
924   void _login44() async {
925     try {
926       final response = await http.post(_loginUrl, body: {
927         'username': _username2.text,
928         'password': _password2.text,
929       });
930       if (response.statusCode == 200) {
931         final token = json.decode(response.body)['token'];
932         _token2.text = token;
933         _token2.clear();
934         _username2.clear();
935         _password2.clear();
936         _token2.clear();
937       } else {
938         _token2.clear();
939       }
940     } catch (e) {
941       _token2.clear();
942     }
943   }
944
945   void _login45() async {
946     try {
947       final response = await http.post(_loginUrl, body: {
948         'username': _username2.text,
949         'password': _password2.text,
950       });
951       if (response.statusCode == 200) {
952         final token = json.decode(response.body)['token'];
953         _token2.text = token;
954         _token2.clear();
955         _username2.clear();
956         _password2.clear();
957         _token2.clear();
958       } else {
959         _token2.clear();
960       }
961     } catch (e) {
962       _token2.clear();
963     }
964   }
965
966   void _login46() async {
967     try {
968       final response = await http.post(_loginUrl, body: {
969         'username': _username2.text,
970         'password': _password2.text,
971       });
972       if (response.statusCode == 200) {
973         final token = json.decode(response.body)['token'];
974         _token2.text = token;
975         _token2.clear();
976         _username2.clear();
977         _password2.clear();
978         _token2.clear();
979       } else {
980         _token2.clear();
981       }
982     } catch (e) {
983       _token2.clear();
984     }
985   }
986
987   void _login47() async {
988     try {
989       final response = await http.post(_loginUrl, body: {
990         'username': _username2.text,
991         'password': _password2.text,
992       });
993       if (response.statusCode == 200) {
994         final token = json.decode(response.body)['token'];
995         _token2.text = token;
996         _token2.clear();
997         _username2.clear();
998         _password2.clear();
999         _token2.clear();
1000       } else {
1001         _token2.clear();
1002       }
1003     } catch (e) {
1004       _token2.clear();
1005     }
1006   }
1007
1008   void _login48() async {
1009     try {
1010       final response = await http.post(_loginUrl, body: {
1011         'username': _username2.text,
1012         'password': _password2.text,
1013       });
1014       if (response.statusCode == 200) {
1015         final token = json.decode(response.body)['token'];
1016         _token2.text = token;
1017         _token2.clear();
1018         _username2.clear();
1019         _password2.clear();
1020         _token2.clear();
1021       } else {
1022         _token2.clear();
1023       }
1024     } catch (e) {
1025       _token2.clear();
1026     }
1027   }
1028
1029   void _login49() async {
1030     try {
1031       final response = await http.post(_loginUrl, body: {
1032         'username': _username2.text,
1033         'password': _password2.text,
1034       });
1035       if (response.statusCode == 200) {
1036         final token = json.decode(response.body)['token'];
1037         _token2.text = token;
1038         _token2.clear();
1039         _username2.clear();
1040         _password2.clear();
1041         _token2.clear();
1042       } else {
1043         _token2.clear();
1044       }
1045     } catch (e) {
1046       _token2.clear();
1047     }
1048   }
1049
1050   void _login50() async {
1051     try {
1052       final response = await http.post(_loginUrl, body: {
1053         'username': _username2.text,
1054         'password': _password2.text,
1055       });
1056       if (response.statusCode == 200) {
1057         final token = json.decode(response.body)['token'];
1058         _token2.text = token;
1059         _token2.clear();
1060         _username2.clear();
1061         _password2.clear();
1062         _token2.clear();
1063       } else {
1064         _token2.clear();
1065       }
1066     } catch (e) {
1067       _token2.clear();
1068     }
1069   }
1070
1071   void _login51() async {
1072     try {
1073       final response = await http.post(_loginUrl, body: {
1074         'username': _username2.text,
1075         'password': _password2.text,
1076       });
1077       if (response.statusCode == 200) {
1078         final token = json.decode(response.body)['token'];
1079         _token2.text = token;
1080         _token2.clear();
1081         _username2.clear();
1082         _password2.clear();
1083         _token2.clear();
1084       } else {
1085         _token2.clear();
1086       }
1087     } catch (e) {
1088       _token2.clear();
1089     }
1090   }
1091
1092   void _login52() async {
1093     try {
1094       final response = await http.post(_loginUrl, body: {
1095         'username': _username2.text,
1096         'password': _password2.text,
1097       });
1098       if (response.statusCode == 200) {
1099         final token = json.decode(response.body)['token'];
1100         _token2.text = token;
1101         _token2.clear();
1102         _username2.clear();
1103         _password2.clear();
1104         _token2.clear();
1105       } else {
1106         _token2.clear();
1107       }
1108     } catch (e) {
1109       _token2.clear();
1110     }
1111   }
1112
1113   void _login53() async {
1114     try {
1115       final response = await http.post(_loginUrl, body: {
1116         'username': _username2.text,
1117         'password': _password2.text,
1118       });
1119       if (response.statusCode == 200) {
1120         final token = json.decode(response.body)['token'];
1121         _token2.text = token;
1122         _token2.clear();
1123         _username2.clear();
1124         _password2.clear();
1125         _token2.clear();
1126       } else {
1127         _token2.clear();
1128       }
1129     } catch (e) {
1130       _token2.clear();
1131     }
1132   }
1133
1134   void _login54() async {
1135     try {
1136       final response = await http.post(_loginUrl, body: {
1137         'username': _username2.text,
1138         'password': _password2.text,
1139       });
1140       if (response.statusCode == 200) {
1141         final token = json.decode(response.body)['token'];
1142         _token2.text = token;
1143         _token2.clear();
1144         _username2.clear();
1145         _password2.clear();
1146         _token2.clear();
1147       } else {
1148         _token2.clear();
1149       }
1150     } catch (e) {
1151       _token2.clear();
1152     }
1153   }
1154
1155   void _login55() async {
1156     try {
1157       final response = await http.post(_loginUrl, body: {
1158         'username': _username2.text,
1159         'password': _password2.text,
1160       });
1161       if (response.statusCode == 200) {
1162         final token = json.decode(response.body)['token'];
1163         _token2.text = token;
1164         _token2.clear();
1165         _username2.clear();
1166         _password2.clear();
1167         _token2.clear();
1168       } else {
1169         _token2.clear();
1170       }
1171     } catch (e) {
1172       _token2.clear();
1173     }
1174   }
1175
1176   void _login56() async {
1177     try {
1178       final response = await http.post(_loginUrl, body: {
1179         'username': _username2.text,
1180         'password': _password2.text,
1181       });
1182       if (response.statusCode == 200) {
1183         final token = json.decode(response.body)['token'];
1184         _token2.text = token;
1185         _token2.clear();
1186         _username2.clear();
1187         _password2.clear();
1188         _token2.clear();
1189       } else {
1190         _token2.clear();
1191       }
1192     } catch (e) {
1193       _token2.clear();
1194     }
1195   }
1196
1197   void _login57() async {
1198     try {
1199       final response = await http.post(_loginUrl, body: {
1
```



#### 4. Screenshoot Hasil Running

##### A. Login

Login Page

Username  
operator

Password  
.....

Login

Login Page

Username  
admin

Password  
.....

Login

Aplikasi Bengkel Putra

Pelanggan

Layanan

##### B. Pelanggan

Data Pelanggan

01  
Gina putri  
081234  
Jl kastuti Banjarbaru  
Ban bocor  
18-01-2025

02  
Muhammad azam  
0812367  
Sungai Ulin  
Ganti Oli  
19-01-2025

03  
Muhammad Davinci  
0857321  
Jln Perambaan 3  
Perbaikan mesin  
18-01-2025

+ person

Tambah Pelanggan Servis

Silahkan isi data berikut :

Input No Antrian Pelanggan  
04

Input Nama Lengkap  
Salsabilla

Input No Telepon  
0823612

Input Alamat Pelanggan  
Jln Kacang Panjang

Input Deskripsi Layanan  
Ganti aki

Tanggal Servis  
20-01-2025

Submit

Reset

Ubah Data Pelanggan Servis

Silahkan isi data berikut :

Input No Antrian Pelanggan  
04

Input No Telepon  
0823612

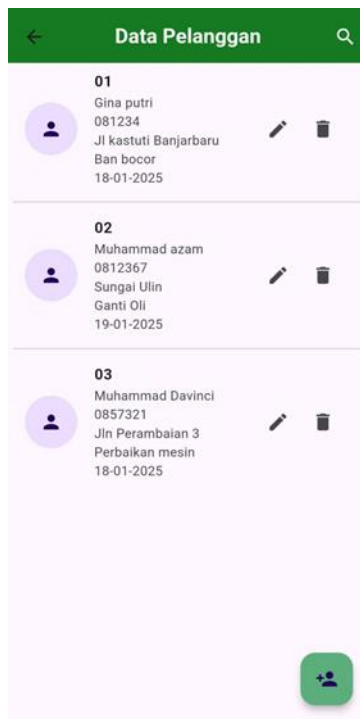
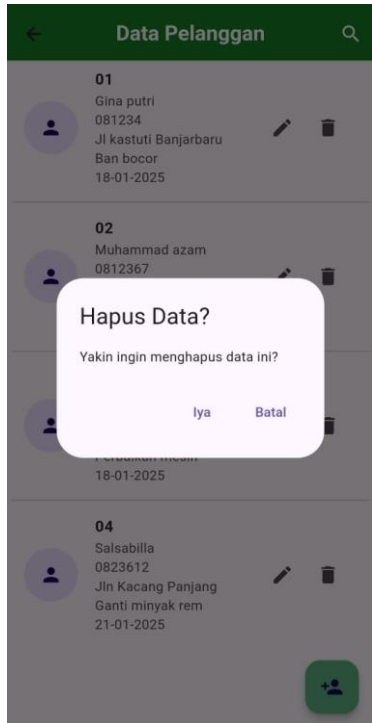
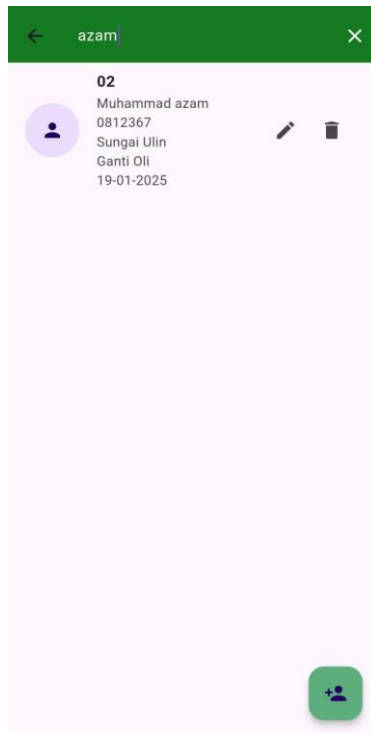
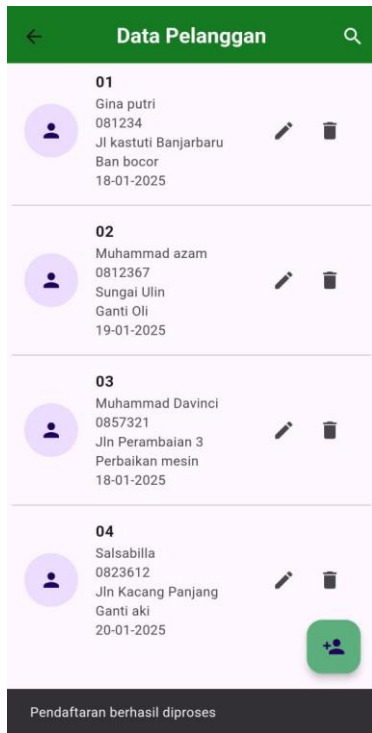
Input Alamat Pelanggan  
Jln Kacang Panjang

Input Deskripsi Layanan  
Ganti aki

Tanggal Servis  
20-01-2025

Submit

Reset



## C. Bengkel



←

Data Layanan Servis

🔍



01

Ganti oli  
Pergantian oli dengan  
kualitas bagus  
50000  
20 menit  
Mesin





02


Bongkar mesin  
Bongkar mesin  
menyeluruh untuk  
perbaikan  
120000  
60 menit  
Mesin



03

Tambal ban  
Perbaikan ban dan  
pengganrian vleg  
20000  
10 menit  
Perbaikan ban dan vleg





←

Tambah Layanan Servis

Silahkan isi data berikut :

Input No Layanan Servis

04

Input Nama Layanan

Ganti aki

Input Deskripsi Servis

Perbaikan aki pada mesin

Input Harga Servis

30000

Input Durasi Servis

40 menit

Input Kategori Servis

mesin

Submit

Reset

←

Data Layanan Servis

🔍


01

Ganti oli  
Pergantian oli dengan  
kualitas bagus  
50000  
20 menit  
Mesin



02

Bongkar mesin  
Bongkar mesin  
menyeluruh untuk  
perbaikan  
120000  
60 menit  
Mesin



03

Tambal ban  
Perbaikan ban dan  
pengganrian vleg  
20000  
10 menit  
Perbaikan ban dan vleg



04

Ganti aki  
Perbaikan aki pada  
mesin  
30000  
40 menit  
mesin



←

Ubah Data Layanan Servis

Silahkan isi data berikut :

Input No Layanan Servis

05

Input Nama Layanan

Ganti aki

Input Deskripsi Servis

Perbaikan aki pada mesin

Input Harga Servis

35000

Input Durasi Servis

45 menit

Input Kategori Servis

mesin

Submit

Reset



←

Data Layanan Servis

🔍



01

Ganti oli  
Pergantian oli dengan  
kualitas bagus  
50000  
20 menit  
Mesin





02

Bongkar mesin  
Bongkar mesin  
menyeluruh untuk  
perbaikan  
120000  
60 menit  
Mesin





03

Tambal ban  
Perbaikan ban dan  
pengganrian vleg  
20000  
10 menit  
Perbaikan ban dan vleg



05

Ganti aki  
Perbaikan aki pada  
mesin  
35000  
45 menit  
mesin



←

Bongkar mesin

No Layanan : 02

Nama Layanan : Bongkar mesin

Deskripsi : Bongkar mesin menyeluruh untuk perbaikan

Harga : 120000

Durasi : 60 menit

Kategori : Mesin

