4. IMPLEMENTASI SISTEM

4.1. Tools dan Library yang digunakan untuk implementasi

Aplikasi penjualan sepeda ini dibuat dengan menggunakan *framework* Flutter dengan bahasa pemrograman Dart. *Framework* Flutter digunakan untuk proses penyusunan tampilan aplikasi. Sedangkan Bahasa pemrograman Dart digunakan dalam pembuatan logika, penghubung antara front end dan back end serta pengiriman data ke dalam database. Database yang digunakan adalah Firebase FireStore.

4.2.Langkah Langkah Implementasi

Langkah-langkah untuk melakukan proses implementasi:

1.Install flutter dari https://docs.flutter.dev/get-started/install

2.Ekstrak Flutter SDK

3.Tambahkan Flutter ke Path

4.Instalasi Flutter pada aplikasi pengembangan

5.Membuat Project

Tabel 4. 1

Tabel Program

Program	Kegiatan	Segment Program
Sign Up	*generate otp*	Segment 4.2.1
	check otp expired	Segment 4.2.2
	add data to database	Segment 4.2.3
Login	*check login*	Segment 4.2.4
	check login status	Segment 4.2.5
Mencari *item*	*Show all item*	Segment 4.2.6

	searching	Segment 4.2.7
Menambahkan *item* ke favorit	*Check* favorit	Segment 4.2.8
	Mengubah favorit	Segment 4.2.9
Mengatur Lokasi	Menampilkan Peta	Segment 4.2.10
	Memperbarui *database*	Segment 4.2.11
Membeli *item*	Melihat gambar	Segment 4.2.12
	Mengecek ada tidaknya lokasi	Segment 4.2.13
	Membuat Transaksi Baru	Segment 4.2.14
Konfirmasi penerimaan barang	*Auto update status*	Segment 4.2.15
Melihat *history*	Transaksi *history*	Segment 4.2.16
Menambah jenis barang	Memilih gambar	Segment 4.2.17
	Menambahkan item ke *database*	Segment 4.2.18
Pembelian	Merubah sebagai harga	Segment 4.2.19
	Pengecekan dan perbarui *database*	Segment 4.2.20

4.2.1. Sign Up

Berikut ini adalah beberapa *coding* yang akan digunakan untuk melakukan *sign up*. Pada awalnya user akan diminta untuk melakukan pengisian data diri seperti nama, *email*, nomor telepon, *password* dan konfirmasi *password*. Setelah itu *coding* akan membuat *random number* untuk melakukan *generate otp*(Segment 4.2.1.1). Setelah *otp* berhasil di *generate, otp* akan dikirim menuju user dengan fungsi dari *flutter mailer*. Setelah otp dikirimkan *user* akan dibawah menuju halaman selanjutnya untuk melakukan verifikasi *otp* yang telah dikirimkan. *Otp* yang telah dikirim akan kadaluarsa dalam 5 menit dengan

menggunakan pengecekan(Segment 4.2.1.2).Jika seluruh proses berjalan dengan baik maka Coding akan melakukan penambahan data(Segment 4.2.1.3) sehingga user berhasil ditambahkan ke database. Beberapa data yang ditambahkan akan di enkripsi terlebih dahulu menggunakan libraries dari flutter demi menjaga keamanan user.

Segment 4.2.1 *generate otp*

```
int generateRandomNumber(int min, int max) {
    Random random = Random(DateTime.now().millisecondsSinceEpoch);
    int randomNumber = min + random.nextInt(max - min + 1);
    while (randomNumber < 100000 || randomNumber > 999999) {
      randomNumber = min + random.nextInt(max - min + 1);
    }
    return randomNumber;
  }
```

Segment 4.2.2 *check otp expired*

```
 bool isOtpExpired() {
    DateTime  otpExpiryTime  =  otpCreationTime.add(Duration(minutes:
5));
    print("Current Time: $now, OTP Expiry Time: $otpExpiryTime");
    Duration timeDifference = otpExpiryTime.difference(now);
    bool isExpired = timeDifference.isNegative;
    print("Time Difference: $timeDifference, Is Expired: $isExpired");
    return isExpired;
  }
```

Segment 4.2.3 *add data to database*

```
await FirebaseFirestore.instance.collection('User').add({
          'Email': Encrypt(widget.User.email),
          'Password':Encrypt(widget.User.password) ,
          'Nama': widget.User.nama,
          'PhoneNumber': widget.User.noTelepon,
```

```
          'Role' : 'User',
          'Status' :'Active',
          'TransactionList': [],
          'Photo':'',
          'Location':'',
          'WishlistId':'',
          'CartId':'',
          'Lat':0,
          'Long':0,
        });
```

4.2.2. Login

Berikut ini adalah beberapa *coding* yang akan digunakan untuk melakukan *login*. Pada awalnya user dapat mengisi data diri yang diperlukan seperti *email* dan *password.* Setelah mengisi dan password user dapat menekan login untuk melakukan pengecekan(Segment 4.2.2.1). Setelah pengecekan dilakukan user akan diarahkan ke halaman *dashboard* tergantung *role* masing masing. User juga dapat menggunakan fitur *fingerprint* yang telah disediakan setelah mengaktifkan izin *fingerprint* dari *setting*. Jika diizinkan *login* dengan *fingerprint* maka untuk selanjutnya sistem akan melakukan pengecekan izin *fingerprint*(Segment 4.2.2.2), jika berhasil maka user dapat login dengan *fingerprint*.

Segment 4.2.4 *check login*

```
Future<void> _login() async {
    try {
      if              (_emailController.text.isEmpty              ||
_passwordController.text.isEmpty) {
        _showToast("Email and password are required.");
        return;
      }
      String email = encrypt(_emailController.text.toLowerCase());
      String password = encrypt(_passwordController.text);
      String role = "Owner";
```

```
      QuerySnapshot<Map<String, dynamic>> querySnapshot =
      await FirebaseFirestore.instance.collection('User').get();
      if (querySnapshot.docs.isEmpty) {
        _showToast("User not found");
        return;
      }
      bool userFound = false;
      for (QueryDocumentSnapshot<Map<String, dynamic>> document in
querySnapshot.docs) {
        String storedEmail = document.data()['Email'];
        String storedPassword = document.data()['Password'];
        String storedRole = document.data()['Role'];
        String status = document.data()['Status'];

        if (email == storedEmail && password == storedPassword) {
          userFound = true;
          if (status == 'Active') {
            final SharedPreferences sharedPreferences = await
SharedPreferences.getInstance();
            sharedPreferences.setString('email',
_emailController.text.toLowerCase());
            if (role == storedRole) {
              sharedPreferences.setString('role', "Owner");
              Navigator.pushReplacement(
                context,
                MaterialPageRoute(
                    builder: (context) => BottomNavS(email:
_emailController.text.toLowerCase(), index: 0)),
              );
            } else {
              sharedPreferences.setString('role', "User");
              Navigator.pushReplacement(
                context,
                MaterialPageRoute(
```

```
                    builder:     (context)     =>     BottomNav(email:
_emailController.text.toLowerCase(), index: 0)),
                );
              }
            } else {
              _showToast("Akun sedang tidak aktif");
            }
            break;
          }
        }
        if (!userFound) {
          _showToast("Email or Password is wrong");
          _emailController.clear();
          _passwordController.clear();
        }
      } catch (e) {
        print('Error: $e');
        _showToast("An error occurred during login.");
      }
    }
```

Segment 4.2.5 *check login status*

```
void checkLoginStatus() async {
    SharedPreferences prefs = await SharedPreferences.getInstance();
    String? userProfileString = prefs.getString('email');
    String? userrole = prefs.getString('role');
    bool? finger =prefs.getBool('fingerprintLogin');
    if (userProfileString != null) {
      setState(() {
        userEmail = userProfileString;
        userRole =userrole;
        _isFingerprintLoginEnabled=finger!;
      });
```

```
    }
  }
```

4.2.3. Mencari *item*

Berikut ini adalah beberapa *coding* yang akan digunakan untuk melakukan pencarian barang. User akan melakukan akses aplikasi. Saat akses aplikasi user memiliki 2 pilih yaitu untuk mencari *item* secara manual dari seluruh *item* yang ada(Segment 4.2.6), atau menggunakan *search bar* yang telah disediakan.pencarian menggunakan *search bar*(Segment 4.2.7) ini melakukan 3 *filter* dalam pencarian yang pertama yaitu menggunakan *cosine similarity* untuk melakukan pengecekan kesamaan . yang kedua menggunakan pencarian berdasarkan nama dan yang ketiga pencarian dilakukan berdasarkan *tag* dari *item*.Setelah 3 pencarian itu selesai maka hasil akan digabung menjadi satu.

Segment 4.2.6 *show all item*

```
Stream<List<DocumentSnapshot>> fetchAndSortItemsStream() {
    return _firestore.collection('Item').snapshots().map((snapshot) {
      var items = snapshot.docs.where((doc) => doc.data()['Status'] ==
true && doc.data()['TotalStock'] > 0).toList();
      switch (sort) {
        case 'Termahal':
          items.sort((a,                    b)                    =>
parsePrice(b).compareTo(parsePrice(a)));
          break;
        case 'Termurah':
          items.sort((a,                    b)                    =>
parsePrice(a).compareTo(parsePrice(b)));
          break;
        case 'Terbanyak':
          items.sort((a,    b)    =>    (b.data()['TotalStock']    as
int).compareTo(a.data()['TotalStock'] as int));
          break;
        case 'Tersedikit':
```

```
          items.sort((a,    b)    =>    (a.data()['TotalStock']    as
int).compareTo(b.data()['TotalStock'] as int));
          break;
      }
      return items;
    });
  }
```

Segment 4.2.7 *searching*

```
  bool isTermContained(String searchTerm, String itemName) {
    var options = FuzzyOptions(
      findAllMatches: false,
      threshold: 0.6,
      isCaseSensitive: false,
    );

    var fuzzy = Fuzzy([itemName], options: options);
    var result = fuzzy.search(searchTerm);

    return result.isNotEmpty && result[0].score <= options.threshold;
  }

  Future<List<Map<String, dynamic>>> searchIncludingTags(String query)
async {
    final querySnapshot = await _firestore.collection('Item').get();

    List<Map<String, dynamic>> filteredResults = [];

    for (var doc in querySnapshot.docs) {
      var item = doc.data() as Map<String, dynamic>;
      item['id'] = doc.id; // Correctly capturing Firestore document
ID
```

```
      if (item['Status'] == false) {
        continue;
      }

      if (item['TotalStock'] != null && item['TotalStock'] <= 0) {
        continue;
      }

      List<String> tags = List<String>.from(item['Tags'] ?? []);
      bool matchesTag = tags.any((tag) =>
tag.toLowerCase().contains(query.toLowerCase()));

      if (isTermContained(query, item['Nama'] ?? '') || matchesTag) {
        filteredResults.add({
          'data': item,
          'similarity': 1.0,
        });
      }
    }

    return filteredResults;
  }

  double cosineSimilarity(Map<String, int> a, Map<String, int> b) {
    if (a.isEmpty || b.isEmpty) return 0.0;
    final aSet = a.keys.toSet();
    final bSet = b.keys.toSet();
    final intersection = aSet.intersection(bSet);
    double dotProduct = 0.0;
    for (var word in intersection) {
      dotProduct += a[word]! * b[word]!;
    }
    final magnitudeA = sqrt(a.values.fold(0, (sum, x) => sum + x * x));
    final magnitudeB = sqrt(b.values.fold(0, (sum, x) => sum + x * x));
```

```
    if (magnitudeA == 0 || magnitudeB == 0) {
      return 0.0; // Return 0 if either vector is zero
    }
    return dotProduct / (magnitudeA * magnitudeB);
  }

  Map<String, int> termFrequencyMap(String text) {
    var words = text.toLowerCase().split(RegExp(r'\W+'));
    var freqMap = words.where((w) => w.isNotEmpty).fold<Map<String,
int>>({}, (map, word) {
      map[word] = (map[word] ?? 0) + 1;
      return map;
    });
    print("Term Frequency Map for '$text': $freqMap");
    return freqMap;
  }

  Future<List<Map<String, dynamic>>> searchByName2(String query) async
{
    final querySnapshot = await _firestore.collection('Item').get();

    List<Map<String, dynamic>> filteredResults = [];

    for (var doc in querySnapshot.docs) {
      var item = doc.data() as Map<String, dynamic>;
      item['id'] = doc.id;  // Ensure to capture Firestore document ID


      if (item['Status'] == false) {
        continue;
      }

      if (item['TotalStock'] != null && item['TotalStock'] <= 0) {
        continue;
```

```
      }

      if (isTermContained(query, item['Nama'] ?? '')) {
        filteredResults.add({
          'data': item,
          'similarity': 1.0,
        });
      }
    }

    return filteredResults;
  }

  Future<List<Map<String, dynamic>>> searchByName(String query) async
{
    final queryVec = termFrequencyMap(query);
    final querySnapshot = await _firestore.collection('Item').get();

    List<Map<String, dynamic>> items = querySnapshot.docs.map((doc) {
      var itemData = doc.data() as Map<String, dynamic>;
      itemData['id'] = doc.id; // Here's where you assign the document
ID
      return itemData;
    }).toList();

    var options = FuzzyOptions(
      findAllMatches: true,
      threshold: 0.4,
      isCaseSensitive: false,
    );

    var names = items.map((e) => e['Nama'] as String).toList();
    var fuzzy = Fuzzy(names, options: options);
    var results = fuzzy.search(query);
```

```
    List<Map<String, dynamic>> filteredResults = [];
    for (var result in results) {
      try {
        // Find the item that matches the fuzzy search result
        var item = items.firstWhere((item) => item['Nama'] ==
result.item);

        if (item['Status'] == false) {
          continue;
        }

        if (item['TotalStock'] != null && item['TotalStock'] <= 0) {
          continue;
        }

        double similarity = cosineSimilarity(queryVec,
termFrequencyMap(item['Nama'] ?? ''));
        if (similarity >= 0.4) {
          filteredResults.add({
            'data': item,
            'similarity': similarity,
          });
        }
      } catch (e) {
        continue;
      }
    }
    filteredResults.sort((a, b) =>
b['similarity'].compareTo(a['similarity']));
    return filteredResults;
  }
```

```
  Future<List<Map<String,   dynamic>>>   searchAndMergeResults(String
query) async {
    List<Map<String, dynamic>> results1 = await searchByName(query);
    List<Map<String, dynamic>> results2 = await searchByName2(query);

    // Merge and remove duplicates
    Map<String, Map<String, dynamic>> uniqueResults = {};
    for (var result in [...results1, ...results2]) {
      uniqueResults[result['data']['id']] = result;
    }
    return uniqueResults.values.toList();
  }
```

4.2.4.Menambahkan *item* ke favorit

Berikut ini adalah beberapa *coding* yang akan digunakan untuk melakukan penambahan barang ke favorit. Pada awal *homepage*, *user* akan dilakukan pengecekan apakah *user* sudah memiliki *id* favorite atau belum(Segment 4.2.8), jika *user* belum memiliki id favorite maka sistem akan membuatkan id baru. *User* juga dapat untuk melakukan perubahan pada favorit dengan menekan *icon* yang telah disediakan(Segment 4.2.9). *User* juga dapat melihat *item* apa saja yang telah ditambahkan ke favorit

Segment 4.2.8 *Check Favorite*

```
  var userDoc = userSnapshot.docs.first;
_wishlistId = userDoc.get('WishlistId') ?? "";
if (_wishlistId.isEmpty) {
          // Create new Wishlist if it doesn't exist
          var             wishlistRef             =             await
_firestore.collection('Wishlist').add({
            'Item': []
          });
          _wishlistId = wishlistRef.id;
          // Update the user document with new wishlist ID
          await
_firestore.collection('User').doc(userDoc.id).update({
            'WishlistId': _wishlistId
```

```
          });
        }
```

Segment 4.2.9 Mengubah Favorit

```
  void _toggleFavorite(String documentId) {
    setState(() {
      if (favorites.contains(documentId)) {
        favorites.remove(documentId);
      } else {
        favorites.add(documentId);
      }
    });
    updateWishlist();
  }
  Future<void> updateWishlist() async {
    if (_wishlistId.isNotEmpty) {
      try {
        await
_firestore.collection('Wishlist').doc(_wishlistId).update({
          'Item': favorites.toList()
        });
        _showToast("Wishlist updated successfully");
      } catch (e) {
        _showToast("Error updating wishlist: $e");
      }
    }
  }
```

4.2.5.Mengatur lokasi

Berikut ini adalah beberapa *coding* yang akan digunakan untuk melakukan penambahan lokasi *user*. *User* setelah dari *homepage* dapat menuju halaman *profile*. Setelah berada di halaman *profile*, user dapat memilih *settings* dan *location*. Setelah itu user memiliki 3 opsi yaitu menulis lokasi secara manual,

menggunakan lokasi sekarang, dan menggunakan map untuk memilih lokasi yang diinginkan(Segment 4.2.10). Setelah melakukan pemilihan lokasi user dapat menyimpan lokasi dengan menekan tombol save lalu sistem akan memperbarui database.

Segment 4.2.10 Menampilkan peta

```
void _showManualMapInput() {
    showModalBottomSheet(
      context: context,
      builder: (ctx) {
        return StatefulBuilder(
          builder: (BuildContext context, StateSetter
bottomSheetSetState) {
            return Container(
              height: MediaQuery.of(context).size.height * 0.75,
              padding: EdgeInsets.all(8),
              child: Column(
                children: [
                  Padding(
                    padding: const EdgeInsets.all(8.0),
                    child: TextField(
                      decoration: InputDecoration(
                        labelText: 'Search for a location',
                        suffixIcon: Icon(Icons.search),
                        border: OutlineInputBorder(),
                      ),
                      onSubmitted: (value) async {
                         try {
                          List<Location>    locations    =    await
locationFromAddress(value);
                          if (locations.isNotEmpty) {
                            Location location = locations.first;
                            latLng2.LatLng        newLatLng        =
latLng2.LatLng(location.latitude, location.longitude);
                            bottomSheetSetState(() {
```

```
                                _currentLatLng = newLatLng;
                                _updateMarkerPosition(newLatLng);
                              });
                            } else {
                              _showToast('No locations found.');
                            }
                          } catch (e) {
                            _showToast('Failed   to   find   location:
${e.toString()}');
                          }
                        },
                      ),
                    ),
                    Expanded(
                      child: FlutterMap(
                        mapController: _mapController,
                        options: MapOptions(
                          center:           _currentLatLng           ??
latLng2.LatLng(37.77483, -122.41942),
                          zoom: 15.0,
                          onTap: (_, latlng) {
                            bottomSheetSetState(() {
                              _updateMarkerPosition(latlng);
                            });
                          },
                        ),
                        children: [
                          TileLayer(
                            urlTemplate:
"https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png",
                            subdomains: ['a', 'b', 'c'],
                          ),
                          MarkerLayer(markers: _selectedMarker != null ?
[_selectedMarker!] : []),
```

```
                    ],
                  ),
                ),
                Text("Please zoom in and ensure the address is
correct."),
                ElevatedButton(
                  onPressed: () {

_getFullAddressFromLatLng(_currentLatLng!.latitude,
_currentLatLng!.longitude);
                    _showToast("Location successfully selected.");
                    Navigator.pop(context);
                  },
                  child: Text('Save'),
                  style: ElevatedButton.styleFrom(
                    foregroundColor: Colors.white, backgroundColor:
_iconcolor,
                    textStyle: TextStyle(fontSize: 20),
                    padding:  EdgeInsets.symmetric(horizontal:  50,
vertical: 20),
                  ),
                ),
              ],
            ),
          );
        },
      );
    },
  );
 }

 void _updateMarkerPosition(latLng2.LatLng newLatLng) {
   setState(() {
     _currentLatLng = newLatLng;
```

```
      _selectedMarker = Marker(
        point: newLatLng,
        child: Icon(
          Icons.location_pin,
          color: _iconcolor,
          size: 50.0,
        ),
      );
      _mapController.move(newLatLng, _mapController.zoom);
      _mapController.move(newLatLng, _mapController.zoom);
    });
  }
```

Segment 4.2.11 Memperbarui database

```
void _updateFirestoreLocation() async {
    try {
      await FirebaseFirestore.instance
          .collection('User')
          .doc(widget.ID)
          .update({
            'Location': Currentlocation,
            'Lat':_currentLatLng!.latitude,
            'Long':_currentLatLng!.longitude
          });
      _showToast('Location saved successfully: $Currentlocation');
    } catch (e) {
      _showToast('Failed to save location: ${e.toString()}');
    }
  }
```

4.2.6.Membeli Item

Berikut ini adalah beberapa *coding* yang akan digunakan untuk melakukan pembelian barang. User pada awalnya dapat memilih *item* yang ingin untuk dibeli. User akan diarahkan ke halaman detail Item dimana

user dapat melihat gambar secara *full screen,* melihat spesifikasi *item,* dan menambahkan barang ke favorit. Jika user ingin membeli item user dapat menekan tambahkan barang ke keranjang maka akan muncul pop up untuk *user* melakukan pembelian barang. Setelah user mengisi data pembelian barang dan menambahkan ke keranjang, maka sistem akan memperbarui database. Saat user ingin melakukan *check out* user akan melakukan pengecekan terhadap lokasi *user*(Segment 4.2.13). Jika uset tidak memiliki lokasi *user* akan diminta untuk mengisi lokasi, jika *user* telah memiliki lokasi user akan dibawa ke halaman checkout untuk melakukan pembayaran.

Segment 4.2.12 Melihat Gambar

```
void _showFullImage(BuildContext context, List<String> imageUrls, int
initialIndex) {
    int currentIndex = initialIndex;

    showDialog(
      context: context,
      builder: (BuildContext context) {
        return StatefulBuilder(
          builder: (context, setState) {
            return Dialog(
              backgroundColor: Colors.black,
              insetPadding: EdgeInsets.zero,
              child: Stack(
                children: [
                  InteractiveViewer(
                    boundaryMargin: EdgeInsets.zero,
                    minScale: 0.5,
                    maxScale: 3,
                    child: Center(
                      child: Image.network(
                        imageUrls[currentIndex],
                        fit: BoxFit.contain,
                      ),
                    ),
```

```
                    ),
                    Positioned(
                      bottom: 20,
                      left: 0,
                      right: 0,
                      child: Row(
                        mainAxisAlignment:
MainAxisAlignment.spaceEvenly,
                        children: <Widget>[
                          if (currentIndex > 0)
                            TextButton(
                              onPressed: () {
                                setState(() {
                                  if (currentIndex > 0) {
                                    currentIndex--;
                                  }
                                });
                              },
                              child: Text('Previous'),
                              style: TextButton.styleFrom(
                                foregroundColor: Colors.white,
                                backgroundColor: Color(0xFF763377),
                              ),
                            ),
                          if (currentIndex < imageUrls.length - 1)
                            TextButton(
                              onPressed: () {
                                setState(() {
                                  if (currentIndex < imageUrls.length -
1) {
                                    currentIndex++;
                                  }
                                });
                              },
```

```
                            child: Text('Next'),
                            style: TextButton.styleFrom(
                              foregroundColor: Colors.white,
                              backgroundColor: Color(0xFF763377),
                            ),
                          ),
                        ],
                      ),
                    ),
                    Positioned(
                      top: 20,
                      right: 20,
                      child: IconButton(
                        icon: Icon(Icons.close, color: Colors.white),
                        onPressed: () => Navigator.of(context).pop(),
                      ),
                    ),
                  ],
                ),
              );
            },
          );
        },
      );
}
```

Segment 4.2.13 Mengecek ada tidaknya lokasi

```
Future<void> CheckLocation() async {
    try {
      var userSnapshot = await _firestore
          .collection('User')
          .where('Email', isEqualTo: Encrypt(widget.Email))
          .get();
      if (userSnapshot.docs.isNotEmpty) {
        var userDoc = userSnapshot.docs.first;
        Location = userDoc.get('Location') ?? "";
      } else {
        print('User not found with email ${widget.Email}');
      }
    } catch (e) {
      print('Error updating ListTransaksi: $e');
    }
    if(Location==""){
      showToast("Silahkan mengisi lokasi pada setting");
      Navigator.push(
        context,
        MaterialPageRoute(builder:   (context)   =>   BottomNav(email:
widget.Email, index: 4,)),
      );
    }else{
      Navigator.push(
        context,
        MaterialPageRoute(builder:            (context)            =>
CheckoutPage(Email:widget.Email, ItemId: _selectedItems,)),
      );
    }
  }
```

Segment 4.2.14 Membuat Transaksi baru

```
Future<String?> addTransactionDocument() async {
      DateTime now = DateTime.now();
      String       formattedDate       =       DateFormat('yyyy-MM-dd
HH:mm:ss').format(now);
      try {
        CollectionReference               transactions               =
FirebaseFirestore.instance.collection('Transaction');
        DocumentReference transactionRef = await transactions.add({
          'listCart': listCart,
          'listQuantity': listQuantity,
          'totalPrice': formatAsCurrency(HargaTerakhir),
          'location': Location,
          'status': "Pending",
          'email':widget.Email,
          'deliveryDate': "deliveryDate",
          'BuktiPembayaran':Linkbukti,
          'itemPrice':hargaperpiece,
          'TransactionDate':formattedDate,
          'Ongkir':formatAsCurrency(ongkirs),
          'Note':savedNote,
          'BuktiPengiriman':"",
        });
        String transactionId = transactionRef.id;
        updateListTransaksi(widget.Email, transactionId);
        print('New    transaction    document    added    with    ID:
$transactionId');
        return transactionId; // Return the document ID
      } catch (e) {
        print('Error adding transaction document: $e');
        return null; // Return null if there was an error
      }
    }
```

4.2.7.Konfirmasi penerimaan barang

Berikut ini adalah beberapa *coding* yang akan digunakan untuk melakukan pembelian barang. Pada awalnya user dapat menuju ke halaman history. User akan dapat melakukan 2 cara untuk mengkonfirmasi penerimaan barang. Pertama user dapat membiarkan transaksi lebih dari 3 hari (Segment 4.2.15) untuk membiarkan sistem otomatis merubah status atau user dapat menuju ke halaman detail transaksi dan mengubah status secara manual.

Segment 4.2.15 Auto update status

```
  Future<void> checkAndUpdateStatus() async {
    try {
      QuerySnapshot transactionSnapshot = await
FirebaseFirestore.instance.collection('Transaction').get();

      for (QueryDocumentSnapshot doc in transactionSnapshot.docs) {
        Map<String, dynamic> transactionData = doc.data() as
Map<String, dynamic>;
        if (transactionData['status'] == 'Process') {
          String deliveryDateString = transactionData['deliveryDate'];
          Timestamp deliveryDate =
Timestamp.fromDate(DateTime.parse(deliveryDateString));
          DateTime currentDate = DateTime.now();
          int differenceInDays =
currentDate.difference(deliveryDate.toDate()).inDays;
          if (differenceInDays > 3) {
            await FirebaseFirestore.instance
                .collection('Transaction')
                .doc(doc.id)
                .update({'status': 'Delivered'});
          }
        }
      }
    } catch (e) {
      print('Error checking and updating status: $e');
```

```
    }
  }
```

4.2.8 Melihat *history*

Berikut ini adalah beberapa *coding* yang akan digunakan untuk melihat *history*. *User* dan *owner* dapat melihat *history* dengan menuju pada halaman *history*. Halaman *history* akan terbagi menjadi 4 yaitu "*Process*", "*Pending*", "*Delivered*", dan "*Cancel*". Untuk user transaksi yang ditampilkan akan secara langsung ter*filter* ke *history user* sedangkan untuk *owner* semua transaksi akan muncul sehingga *owner* dapat mengatur proses transaksi dengan lebih mudah. Semua transaksi akan secara otomatis ter*filter* dengan status masing masing baik dari sisi *user* dan *owner*.

Segment 4.2.16 Transaksi *history*

```
if(role=='User'){
          return ListView.builder(
            itemCount: listTransaksi.length,
            itemBuilder: (context, index) {
              return FutureBuilder<DocumentSnapshot>(
                future:
FirebaseFirestore.instance.collection('Transaction').doc(listTransaks
i[index]).get(),
                builder: (context, transactionSnapshot) {
                  if      (transactionSnapshot.connectionState      ==
ConnectionState.waiting) {
                    return CircularProgressIndicator();
                  }    else    if    (transactionSnapshot.hasData    &&
transactionSnapshot.data!.data() != null) {
                    var               transactionData               =
transactionSnapshot.data!.data() as Map<String, dynamic>;
                    if (transactionData['status'] == statusFilter) {
                      return            buildTransactionCard(context,
listTransaksi[index]);
                    } else {
```

```
                    return Container(); // Return an empty container
for non-matching status
                  }
                } else {
                  return   ListTile(title:   Text('Transaction   not
found'));
                }
              },
            );
          },
        );
      } else {
        return StreamBuilder(
          stream:
FirebaseFirestore.instance.collection('Transaction').snapshots(),
          builder: (context, snapshot) {
            if (snapshot.connectionState == ConnectionState.waiting)
{
              return Center(child: CircularProgressIndicator());
            } else if (snapshot.hasError) {
              return           Center(child:           Text('Error:
${snapshot.error}'));
            } else if (snapshot.hasData) {
              var transactionDocs = snapshot.data!.docs;
              return ListView.builder(
                itemCount: transactionDocs.length,
                itemBuilder: (context, index) {
                  var               transactionData               =
transactionDocs[index].data() as Map<String, dynamic>;
                  if (transactionData['status'] == statusFilter) {
                    return            buildTransactionCard(context,
transactionDocs[index].id);
                  } else {
```

```
                        return Container(); // Return an empty container
for non-matching status
                      }
                    },
                  );
                }
```

4.2.9 Menambah jenis barang

Berikut ini adalah beberapa *coding* yang akan digunakan untuk melihat menambah jenis barang. Pada halaman awal owner dapat menekan *add item.* User akan diminta untuk mengisi beberapa hal seperti nama, deskripsi, harga, grosir, harga grosir, dan gambar. *User* dapat melakukan pemilihan gambar dari galeri. Tipe *file* yang dapat dipilih adalah *jpg, jpeg,* dan *png*(Segment 4.2.17).Setelah mengisi data data, *item* akan ditambahkan ke *database*(Segment 4.2.18). Owner akan diberi pilihan untuk menambahkan video atau melihat item. Jika user memilih untuk melihat item user dapat menambahkan *tag* dari item

Segment 4.2.17 Memilih gambar

```
  Future<void> _pickerFile() async {
   _showToast("Please choose the images");
   late Map<Permission, PermissionStatus> statuss;
   final androidInfo = await DeviceInfoPlugin().androidInfo;

   if (androidInfo.version.sdkInt <= 32) {
     statuss = await [Permission.storage].request();
   } else {
     statuss = await [Permission.photos].request();
   }

   var allAccepted = true;
   statuss.forEach((permission, status) {
     if (status != PermissionStatus.granted) {
       allAccepted = false;
     }
```

```
    });

    if (allAccepted) {
      try {
        FilePickerResult?          result          =          await
FilePicker.platform.pickFiles(
          allowMultiple: true,
          type: FileType.custom,
          allowedExtensions: ['jpg', 'jpeg', 'png'],
        );

        if (result != null) {
          setState(() {
            // Add the selected files to the media list
            _mediaList.addAll(result.paths.map((path)              =>
File(path!)).toList());
            _currentIndex = 0;  // Point to the latest added media
          });
        }
      } catch (e) {
        print('Error picking images or videos: $e');
      }
    }
  }
```

Segment 4.2.18 Menambahkan *item* ke *database*

```
if (_mediaList.isNotEmpty) {
      try {
        for (var media in _mediaList) {
          String                       mediaName                      =
DateTime.now().millisecondsSinceEpoch.toString();
          firebase_storage.Reference              ref              =
firebase_storage.FirebaseStorage.instance
```

```
              .ref('${widget.judul}/$mediaName.jpg');

          if (media is File && media.existsSync()) {
            await ref.putFile(media);
            String mediaUrl = await ref.getDownloadURL();
            mediaUrls.add(mediaUrl);
          }
        }

        var                docRef                =                await
FirebaseFirestore.instance.collection('Item').add({
          'Nama': widget.judul,
          'Deskripsi': widget.deskripsi,
          'Harga': widget.harga,
          'HargaBeli': [],
          'Grossir': widget.grossir.toInt(),
          'HargaGrossir': widget.hargagrossir,
          'Stock': [],
          'TotalStock':0,
          'Terjual':0,
          'Status': true,
          'MediaUrls': mediaUrls,
          'Tags':[],
          'Tanggal':[],
        });

        documentId = docRef.id; // Save the document ID
      } catch (error) {
        print('Error uploading media: $error');
        _showToast('Failed to upload media');
        return;
      }
    }
```

4.2.10.Pembelian

berikut ini adalah beberapa *coding* yang akan digunakan untuk melihat pembelian barang. Pada awalnya owner dapat memilih edit item. Setelah itu *owner* akan dibawa ke halaman untuk mengubah item. User dapat memilih pembelian. Lalu user dapat memasukan angka yang akan otomatis berubah menjadi harga dan stok barang yang ditambahkan. Sistem akan melakukan pengecekan untuk memastikan tidak ada data yang kosong setelah itu data pada *database* akan diperbarui.

Segment 4.2.19 Merubah sebagai harga

```
void _formatAsCurrency(TextEditingController controller) {
    String       newText       =       controller.text.replaceAll('.',
'').replaceAll(',', '');
    if (newText.isNotEmpty) {
      double value = double.parse(newText);
      controller.value = TextEditingValue(
        text:
value.toInt().toString().replaceAllMapped(RegExp(r'(\d{1,3})(?=(\d{3}
)+(?!\d))'), (Match m) => '${m[1]}.'),
        selection:                    TextSelection.collapsed(offset:
controller.text.length),
      );
    }
  }
```

Segment 4.2.20 Pengecekan dan perbarui *database*

```
if (newStock != null && newBuyPrice != null && newStock > 0) {
                          stockbaru = totalStock + newStock;
                          stockList.add(newStock);
                          Tanggal.add(formattedDate);
                          HargaBeli.add(newBuyPrice);
                          if (newStock != null && newBuyPrice != null)
{
                            DocumentReference         docRef         =
FirebaseFirestore.instance.collection('Item').doc(widget.judul);
```