

Nama : Amarramitha Poodja Thantawi

NIM : H1D022064

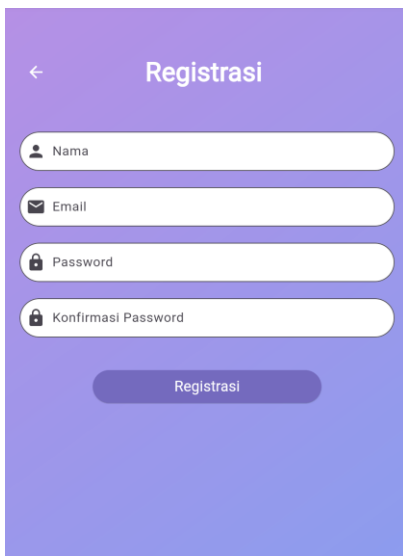
Shift Baru : C

Responsi 1 Praktikum Pemograman Mobile

Manajemen Buku

Tabel Bahasa

1. Proses Registrasi



Pengguna mengisi form registrasi dengan nama, email, dan password. Tiga `TextEditingController` digunakan untuk mengelola input pengguna: `_namaTextboxController` untuk nama, `_emailTextboxController` untuk email, `_passwordTextboxController` untuk password.

```
class _RegistrasiPageState extends State<RegistrasiPage> {  
  final _formKey = GlobalKey<FormState>();  
  bool _isLoading = false;  
  
  final _namaTextboxController = TextEditingController();  
  final _emailTextboxController = TextEditingController();  
  final _passwordTextboxController = TextEditingController();  
}
```

Setiap input memiliki validator untuk memastikan data yang dimasukkan sesuai dengan kriteria: Nama: Minimal 3 karakter. Email: Harus diisi dan valid. Password: Minimal 6 karakter. Konfirmasi Password: Harus sama dengan password yang dimasukkan.

```

Widget _namaTextField() {
  return TextFormField(
    controller: _namaTextboxController,
    decoration: InputDecoration(
      labelText: "Nama",
      prefixIcon: const Icon(Icons.person),
      filled: true,
      fillColor: Colors.white,
      border: OutlineInputBorder(borderRadius: BorderRadius.circular(30.0)),
    ), // InputDecoration
    keyboardType: TextInputType.text,
    validator: (value) {
      if (value!.length < 3) {
        return "Nama harus diisi minimal 3 karakter";
      }
      return null;
    },
    style: const TextStyle(fontFamily: 'Courier New'),
  ); // TextFormField
}

```

```

Widget _emailTextField() {
  return TextFormField(
    controller: _emailTextboxController,
    decoration: InputDecoration(
      labelText: "Email",
      prefixIcon: const Icon(Icons.email),
      filled: true,
      fillColor: Colors.white,
      border: OutlineInputBorder(borderRadius: BorderRadius.circular(30.0)),
    ), // InputDecoration
    keyboardType: TextInputType.emailAddress,
    validator: (value) {
      if (value.isEmpty) {
        return 'Email harus diisi';
      }
      Pattern pattern =
        r'^([^\s@(){}[\]\.\;\:\'"\,]+|\.?<[^\s@(){}[\]\.\;\:\'"\,]+>|(?!(?=[^\s@(){}[\]\.\;\:\'"\,]+>)[^\s@(){}[\]\.\;\:\'"\,]+))@([0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3})|((([a-zA-Z-0-9]+\.)+[a-zA-Z]{2,}))$';
      RegExp regex = RegExp(pattern.toString());
      if (!regex.hasMatch(value)) {
        return "Email tidak valid";
      }
      return null;
    },
    style: const TextStyle(fontFamily: 'Courier New'),
  ); // TextFormField
}

```

```

Widget _passwordTextField() {
  return TextFormField(
    controller: _passwordTextboxController,
    decoration: InputDecoration(
      labelText: "Password",
      prefixIcon: const Icon(Icons.lock),
      filled: true,
      fillColor: Colors.white,
      border: OutlineInputBorder(borderRadius: BorderRadius.circular(30.0)),
    ), // InputDecoration
    obscureText: true,
    validator: (value) {
      if (value!.length < 6) {
        return "Password harus diisi minimal 6 karakter";
      }
      return null;
    },
    style: const TextStyle(fontFamily: 'Courier New'),
  ); // TextFormField
}

```

```

Widget _passwordKonfirmasiTextField() {
  return TextFormField(
    decoration: InputDecoration(
      labelText: "Konfirmasi Password",
      prefixIcon: const Icon(Icons.lock),
      filled: true,
      fillColor: Colors.white,
      border: OutlineInputBorder(borderRadius: BorderRadius.circular(30.0)),
    ), // InputDecoration
    obscureText: true,
    validator: (value) {
      if (value != _passwordTextboxController.text) {
        return "Konfirmasi Password tidak sama";
      }
      return null;
    },
    style: const TextStyle(fontFamily: 'Courier New'),
  ); // TextFormField
}

```

Ketika tombol "Registrasi" ditekan, fungsi `_submit()` dipanggil setelah validasi form berhasil. Jika validasi berhasil dan tombol tidak dalam status loading, proses registrasi dimulai.

```

Widget _buttonRegistrasi() {
  return _isLoading
    ? const CircularProgressIndicator()
    : ElevatedButton(
      style: ElevatedButton.styleFrom(
        padding:
          const EdgeInsets.symmetric(horizontal: 100, vertical: 15),
        shape: RoundedRectangleBorder(
          borderRadius: BorderRadius.circular(30),
        ), // RoundedRectangleBorder
        backgroundColor: const Color.fromARGB(255, 118, 106, 196),
      ),
      child: const Text(
        "Registrasi",
        style: TextStyle(fontSize: 18, color: Colors.white),
      ), // Text
      onPressed: () {
        var validate = _formKey.currentState!.validate();
        if (validate && !_isLoading) _submit();
      },
    ); // ElevatedButton
}

```

Pada proses validasi dan manajemen state, pertama-tama memanggil `save()` pada `_formKey.currentState` untuk menyimpan data input. Selanjutnya, mengubah `_isLoading` menjadi `true` agar indikator loading dapat ditampilkan. Kemudian, memanggil `RegistrasiBloc.registrasi`, yang menggunakan `RegistrasiBloc` untuk mengirimkan permintaan registrasi ke API dengan data yang telah dikumpulkan dari controller, yaitu nama, email, dan password. Setelah itu, kita menangani respons dari API; jika berhasil, kita menampilkan dialog sukses menggunakan `SuccessDialog`, yang memberi tahu pengguna bahwa registrasi telah berhasil dan mereka bisa login. Sebaliknya, jika registrasi gagal, kita menampilkan dialog peringatan menggunakan `WarningDialog` yang memberi tahu pengguna tentang kegagalan.

registrasi. Terakhir, kita mengatur ulang state dengan mengubah `_isLoading` kembali ke `false` setelah proses selesai, baik itu berhasil maupun gagal.

```
void _submit() {
  _formKey.currentState!.save();
  setState(() {
    _isLoading = true;
  });

  RegistrasiBloc.registrasi(
    nama: _namaTextboxController.text,
    email: _emailTextboxController.text,
    password: _passwordTextboxController.text,
  ).then((value) {
    showDialog(
      context: context,
      barrierDismissible: false,
      builder: (BuildContext context) => SuccessDialog(
        description: "Registrasi berhasil, silahkan login",
        onClick: () {
          Navigator.pop(context);
        },
      ),
    );
  }, onError: (error) {
    showDialog(
      context: context,
      barrierDismissible: false,
      builder: (BuildContext context) => const WarningDialog(
        description: "Registrasi gagal, silahkan coba lagi",
      ),
    );
  });

  setState(() {
    _isLoading = false;
  });
}
```

RegistrasiBloc mengelola logika bisnis untuk registrasi dengan fungsi registrasi yang menerima parameter nama, email, dan password. Dalam proses ini, RegistrasiBloc membuat `apiUrl` dari `ApiUrl.registrasi` untuk menentukan endpoint API yang tepat. Selanjutnya, ia menyusun body dari permintaan yang berisi data pengguna. Permintaan POST kemudian dikirimkan ke API menggunakan metode `Api().post()`. Setelah menerima respons, RegistrasiBloc mengonversi body respons yang diterima dalam format JSON menjadi objek Registrasi dengan menggunakan metode `fromJson`.

```

import 'dart:convert';
import 'package:responsi1/helpers/api.dart';
import 'package:responsi1/helpers/api_url.dart';
import 'package:responsi1/model/registrasi.dart';

class RegistrasiBloc {
  static Future<Registrasi> registrasi({
    String? nama,
    String? email,
    String? password,
  }) async {
    String apiUrl = ApiUrl.registrasi;
    var body = {"nama": nama, "email": email, "password": password};
    var response = await Api().post(apiUrl, body);
    var jsonObj = json.decode(response.body);
    return Registrasi.fromJson(jsonObj);
  }
}

```

Model Registrasi mendefinisikan struktur data yang akan diterima setelah proses registrasi. Model ini memiliki beberapa atribut, yaitu code yang menyimpan kode respons dari server, status yang menyimpan status proses registrasi (apakah berhasil atau gagal), dan data yang menyimpan informasi tambahan yang mungkin disediakan oleh server. Selain itu, model ini juga dilengkapi dengan metode fromJson yang berfungsi untuk mengonversi data JSON menjadi objek Registrasi.

```

class Registrasi {
  int? code;
  bool? status;
  String? data;

  Registrasi({this.code, this.status, this.data});

  factory Registrasi.fromJson(Map<String, dynamic> obj) {
    return Registrasi(
      code: obj['code'],
      status: obj['status'],
      data: obj['data'],
    );
  }
}

```

2. Proses Login

Pengguna memasukkan email dan password di form yang disediakan. Input ini dikelola oleh `TextEditingController`, yaitu `_emailTextboxController` untuk email dan `_passwordTextboxController` untuk password. Validasi dilakukan untuk memastikan kedua field tidak kosong.

```
class _LoginPageState extends State<LoginPage> {
  final _formKey = GlobalKey<FormState>();
  bool _isLoading = false;

  final _emailTextboxController = TextEditingController();
  final _passwordTextboxController = TextEditingController();
```

Ketika tombol “Login” diklik, aplikasi akan memvalidasi formulir. Jika formulir valid, fungsi `_submit()` akan dipanggil.

```
    onPressed: () {
      var validate = _formKey.currentState!.validate();
      if (validate && !_isLoading) _submit();
    },
  ); // ElevatedButton
}
```

Dalam proses submit, `_isLoading` diatur menjadi `true` untuk menampilkan indikator loading, kemudian memanggil `LoginBloc.login` dengan email dan password yang dimasukkan oleh pengguna. Jika respons `code` dari API adalah 200, berarti login berhasil; dalam hal ini, token dan user ID akan disimpan menggunakan `UserInfo`, dan dialog sukses akan ditampilkan. Pengguna kemudian akan diarahkan ke halaman `BahasaPage`. Namun, jika login gagal (respons tidak 200 atau terjadi error), dialog peringatan akan ditampilkan dengan pesan bahwa login gagal.

```
void _submit() {
  setState(() {
    _isLoading = true;
  });

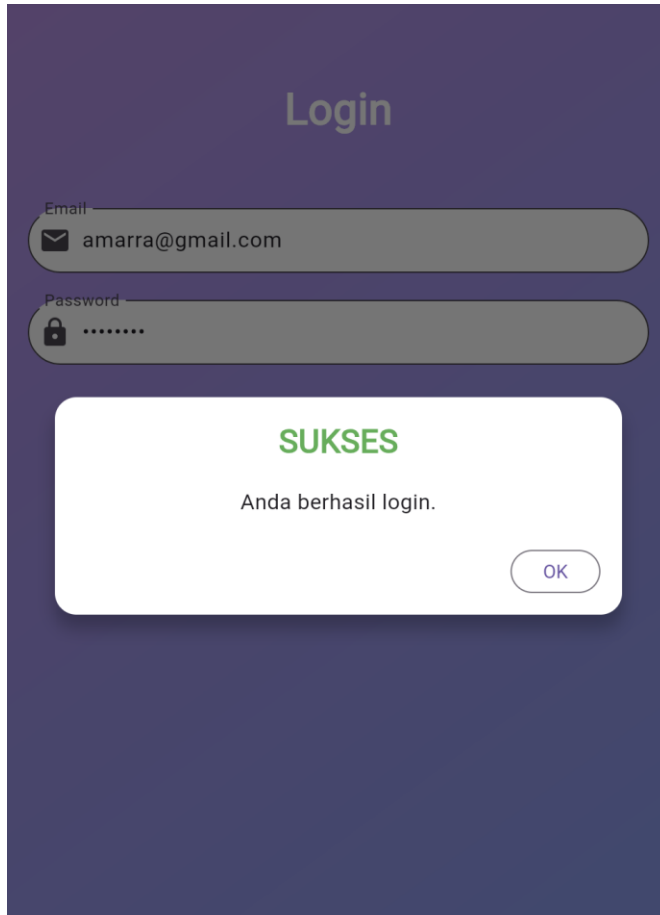
  LoginBloc.login(
    email: _emailTextboxController.text,
    password: _passwordTextboxController.text,
  ).then((value) async {
    if (value.code == 200) {
      await UserInfo().setToken(value.token.toString());
      await UserInfo().setUserID(int.parse(value.userID.toString()));

      // Show success dialog after login success
      showDialog(
        context: context,
        barrierDismissible: false,
        builder: (BuildContext context) => SuccessDialog(
          description: "Anda berhasil login.",
          onClick: () {
            Navigator.pushReplacement(
              context,
              MaterialPageRoute(builder: (context) => const BahasaPage()),
            );
          },
        ), // SuccessDialog
      );
    } else {
      _showWarningDialog("Login gagal, silahkan coba lagi");
    }
  }).catchError((error) {
    _showWarningDialog("Login gagal, silahkan coba lagi");
  });

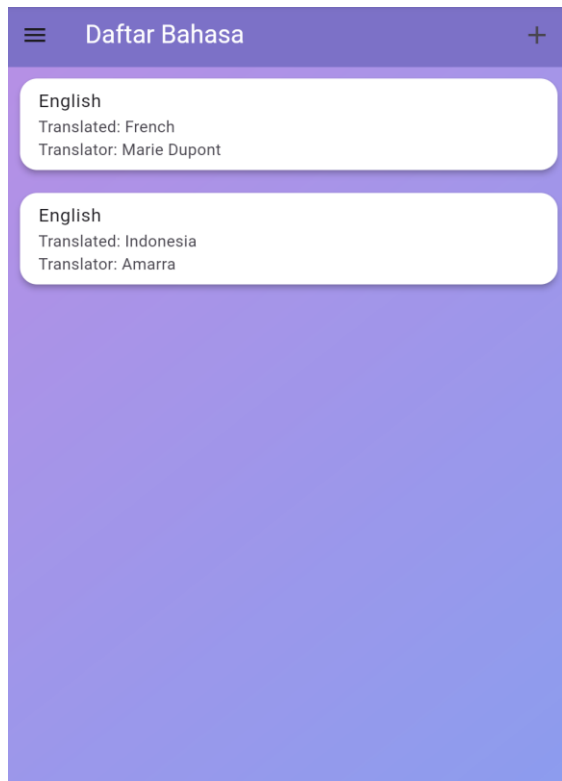
  setState(() {
    _isLoading = false;
  });
}
```

```
class LoginBloc {
  static Future<Login> login({String? email, String? password}) async {
    String apiUrl = ApiUrl.login;
    var body = {"email": email, "password": password};
    var response = await Api().post(apiUrl, body);
    var jsonObj = json.decode(response.body);
    return Login.fromJson(jsonObj);
  }
}
```

Jika login berhasil (dalam contoh ini, jika value.code sama dengan 200), kita menyimpan token dan ID pengguna menggunakan kelas UserInfo. Setelah menyimpan informasi, dialog sukses ditampilkan menggunakan SuccessDialog, yang memberi tahu pengguna bahwa login berhasil. Jika pengguna menekan tombol "OK" di dialog sukses, mereka akan diarahkan ke halaman produk (Bahasa_Page).



3. Create Data



Pada halaman Bahasa_Page, pengguna dapat melihat list data daftar bahasa buku. Ketika pengguna mengklik tanda tambah di pojok kanan atas halaman, maka akan masuk ke halaman tambah data.

Pengguna mengisi form di BahasaForm. Dalam BahasaForm, terdapat tiga input text yang digunakan untuk mengambil data dari pengguna, yaitu originalLanguage untuk memasukkan nama bahasa asli, translatedLanguage untuk memasukkan nama bahasa

terjemahan, dan translatorName untuk memasukkan nama penerjemah. Setiap field menggunakan TextEditingController untuk menangani input dari pengguna.

```
final _originalLanguageController = TextEditingController();
final _translatedLanguageController = TextEditingController();
final _translatorNameController = TextEditingController();
```

Sebelum menyimpan data, form melakukan validasi untuk memastikan bahwa semua field terisi dengan benar. Jika ada field yang kosong, pengguna akan diberi tahu dengan pesan yang sesuai.

```
simpan() {
  setState(() {
    _isLoading = true;
  });
  Bahasa createBahasa = Bahasa(id: null);
  createBahasa.originalLanguage = _originalLanguageController.text;
  createBahasa.translatedLanguage = _translatedLanguageController.text;
  createBahasa.translatorName = _translatorNameController.text;
  BahasaBloc.addBahasa(bahasa: createBahasa).then((value) {
    Navigator.of(context).push(MaterialPageRoute(
      builder: (BuildContext context) => const BahasaPage(), // MaterialPageRoute
    ), onError: (error) {
      showDialog(
        context: context,
        builder: (BuildContext context) => const WarningDialog(
          description: "Simpan gagal, silahkan coba lagi",
        ));
    });
  });
  setState(() {
    _isLoading = false;
  });
}
```

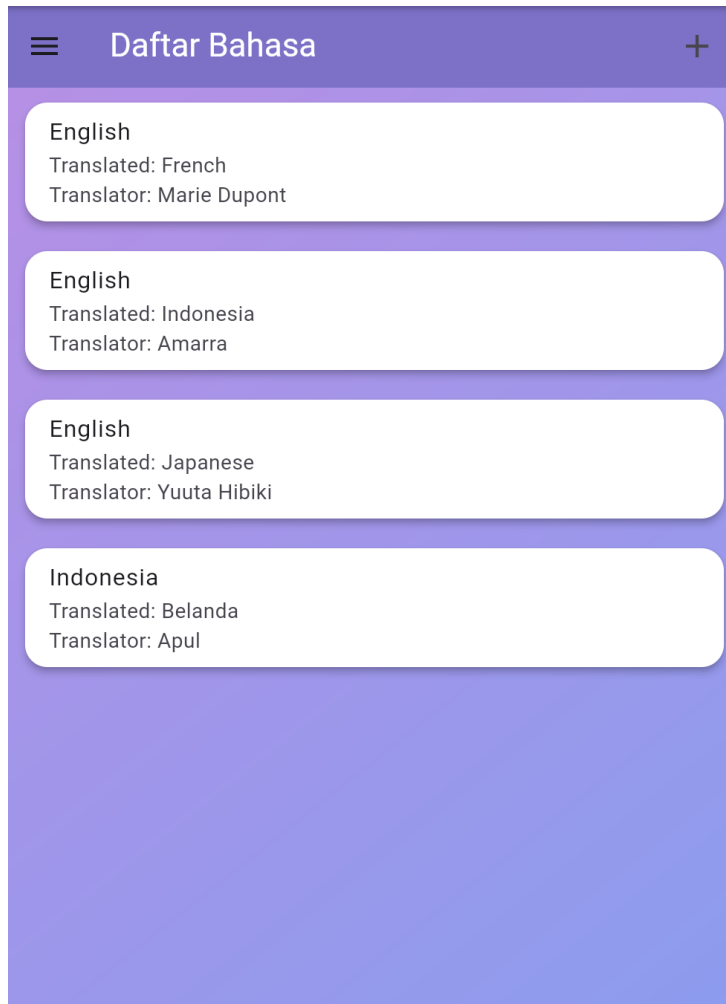
Ketika pengguna menekan tombol "Simpan", fungsi simpan() dipanggil. Di dalam fungsi ini, pertama-tama dilakukan validasi form. Jika validasi berhasil, data diambil dari TextEditingController dan dimasukkan ke dalam model Bahasa.

```
static Future<bool> addBahasa({required Bahasa bahasa}) async {
  String apiUrl = ApiUrl.createBahasa;
  var body = {
    "original_language": bahasa.originalLanguage,
    "translated_language": bahasa.translatedLanguage,
    "translator_name": bahasa.translatorName,
  };

  var response = await Api().post(apiUrl, body);

  if (response.statusCode == 200) {
    var jsonObj = json.decode(response.body);
    return jsonObj['status'];
  } else {
    throw Exception("Failed to add bahasa");
  }
}
```

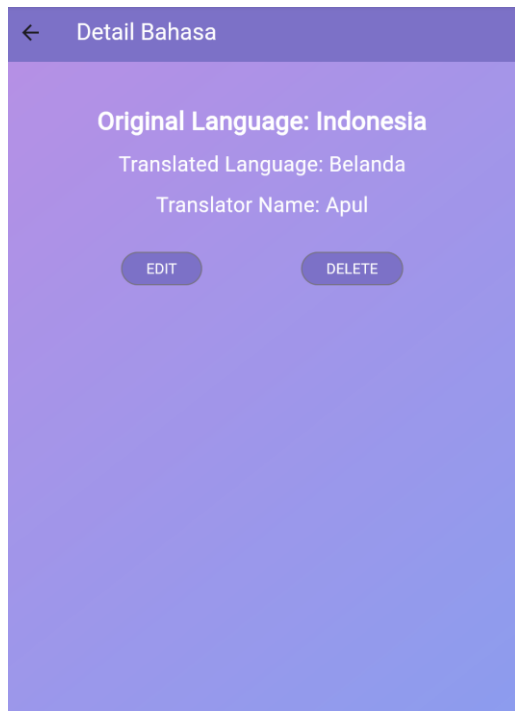
Selanjutnya, fungsi `addBahasa()` dari `BahasaBloc` dipanggil dengan objek `Bahasa` yang baru dibuat sebagai parameter. Dalam `addBahasa()` pada `BahasaBloc`, data kemudian dikirim ke server melalui API menggunakan HTTP POST request. Jika permintaan berhasil, API akan mengembalikan respons yang menunjukkan bahwa data telah berhasil ditambahkan.



Setelah data berhasil disimpan, pengguna dapat dinavigasikan kembali ke halaman `BahasaPage`, yang akan memperbarui daftar bahasa dengan data terbaru.

4. Edit Data

Ketika pengguna memilih bahasa yang ditampilkan di halaman `BahasaPage`, aplikasi akan mengarahkan ke halaman `Detail`.



Ketika pengguna mengklik tombol “edit”, data yang relevan diambil dan dimuat ke dalam field input di BahasaForm menggunakan TextEditingController. Field yang diisi sebelumnya adalah originalLanguage, translatedLanguage, dan translatorName.

```
isUpdate() {  
  if (widget.bahasa != null) {  
    setState(() {  
      judul = "Ubah Bahasa";  
      tombolSubmit = "UBAH";  
      _originalLanguageController.text = widget.bahasa!.originalLanguage!;  
      _translatedLanguageController.text = widget.bahasa!.translatedLanguage!;  
      _translatorNameController.text = widget.bahasa!.translatorName!;  
    });  
  }  
}
```

Pada halaman edit data, pengguna dapat mengubah data seperti original language, translated language, maupun translator languagenya.

← Ubah Bahasa

Original Language
Indonesia

Translated Language
Belanda

Translator Name
Van Der Hoven

UBAH

Ketika pengguna melakukan perubahan dan menekan tombol "Ubah", fungsi `simpan()` dipanggil. Di dalam fungsi ini, validasi form dilakukan untuk memastikan bahwa semua input valid. Jika validasi berhasil, data terbaru diambil dari `TextEditingController` dan dimasukkan ke dalam objek `Bahasa` yang ada.

```
ubah() {  
  setState(() {  
    _isLoading = true;  
  });  
  Bahasa updateBahasa = Bahasa(id: widget.bahasa!.id!);  
  updateBahasa.originalLanguage = _originalLanguageController.text;  
  updateBahasa.translatedLanguage = _translatedLanguageController.text;  
  updateBahasa.translatorName = _translatorNameController.text;  
  BahasaBloc.updateBahasa(bahasa: updateBahasa).then((value) {  
    Navigator.of(context).push(MaterialPageRoute(  
      builder: (BuildContext context) => const BahasaPage(),  
    ));  
  }, onError: (error) {  
    showDialog(  
      context: context,  
      builder: (BuildContext context) => const WarningDialog(  
        description: "Permintaan ubah data gagal, silahkan coba lagi",  
      ));  
  });  
  setState(() {  
    _isLoading = false;  
  });  
}
```

Fungsi `updateBahasa()` dari `BahasaBloc` dipanggil dengan objek `Bahasa` yang telah diperbarui sebagai parameter. Proses pengiriman data ke API dilakukan di dalam fungsi `updateBahasa()` di `BahasaBloc`, di mana data dikirim menggunakan HTTP PUT request. Jika permintaan berhasil, API akan mengembalikan respons yang menunjukkan bahwa data telah berhasil diperbarui.

```

static Future<bool> updateBahasa({required Bahasa bahasa}) async {
  // Pastikan id bahasa tidak null sebelum memanggil int.parse
  if (bahasa.id == null) {
    throw Exception("ID bahasa tidak boleh null");
  }

  String apiUrl = ApiUrl.updateBahasa(
    | bahasa.id!); // Jika id adalah int, tidak perlu parse
  print(apiUrl);

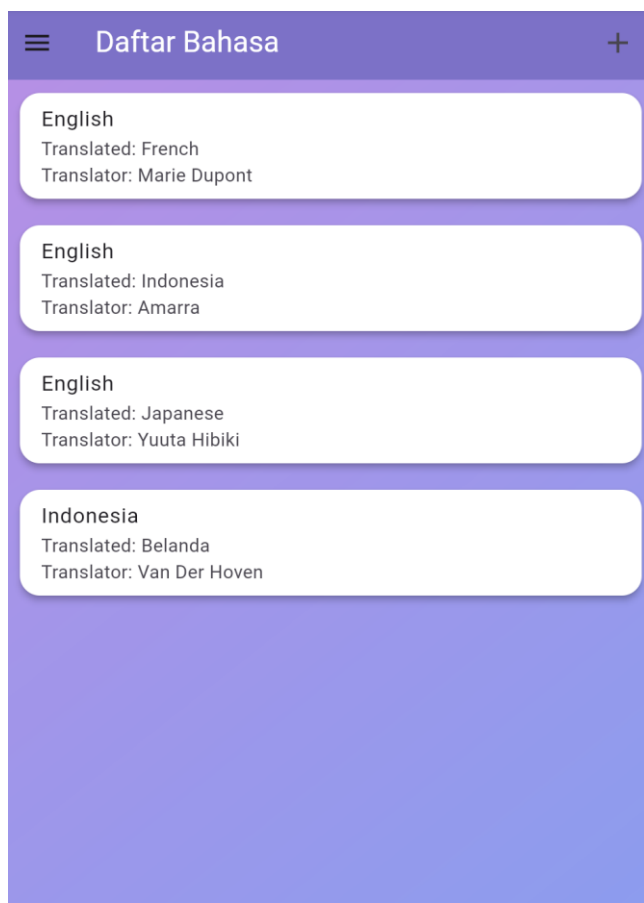
  var body = {
    "original_language": bahasa.originalLanguage,
    "translated_language": bahasa.translatedLanguage,
    "translator_name": bahasa.translatorName,
  };

  print("Body : $body");
  var response = await Api().put(apiUrl, jsonEncode(body));

  if (response.statusCode == 200) {
    var jsonObj = json.decode(response.body);
    return jsonObj['status'];
  } else {
    throw Exception("Failed to update bahasa");
  }
}

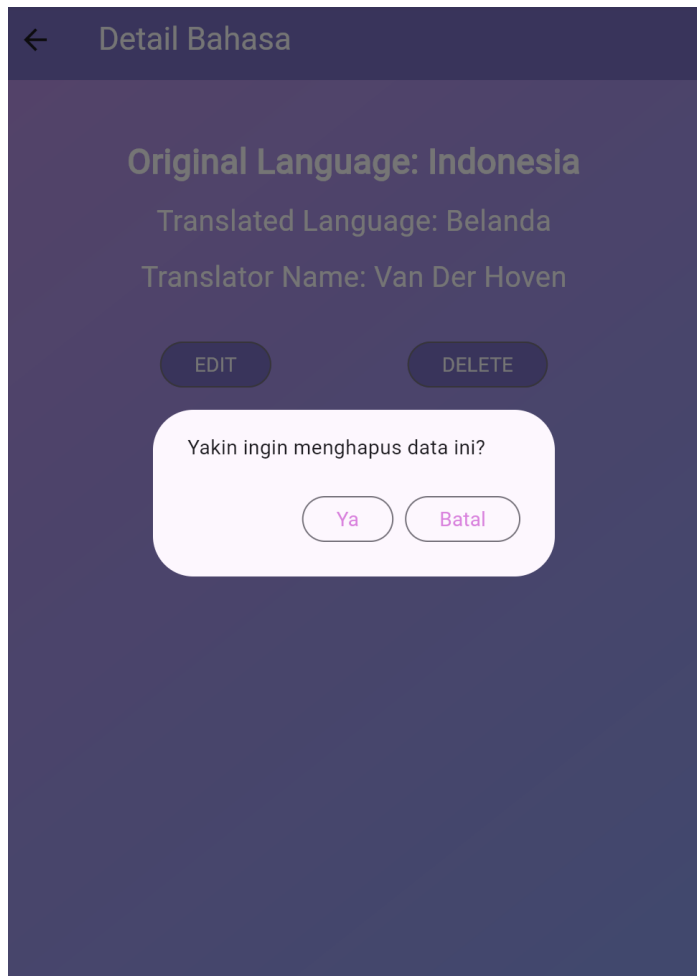
```

Setelah data berhasil diperbarui, pengguna akan diarahkan kembali ke halaman BahasaPage, di mana daftar bahasa akan diperbarui untuk mencerminkan perubahan yang baru saja dilakukan. Tidak ada notifikasi yang ditampilkan setelah proses edit selesai.



5. Hapus Data

Ketika pengguna memilih bahasa yang ditampilkan di halaman BahasaPage, aplikasi akan mengarahkan ke halaman Detail.

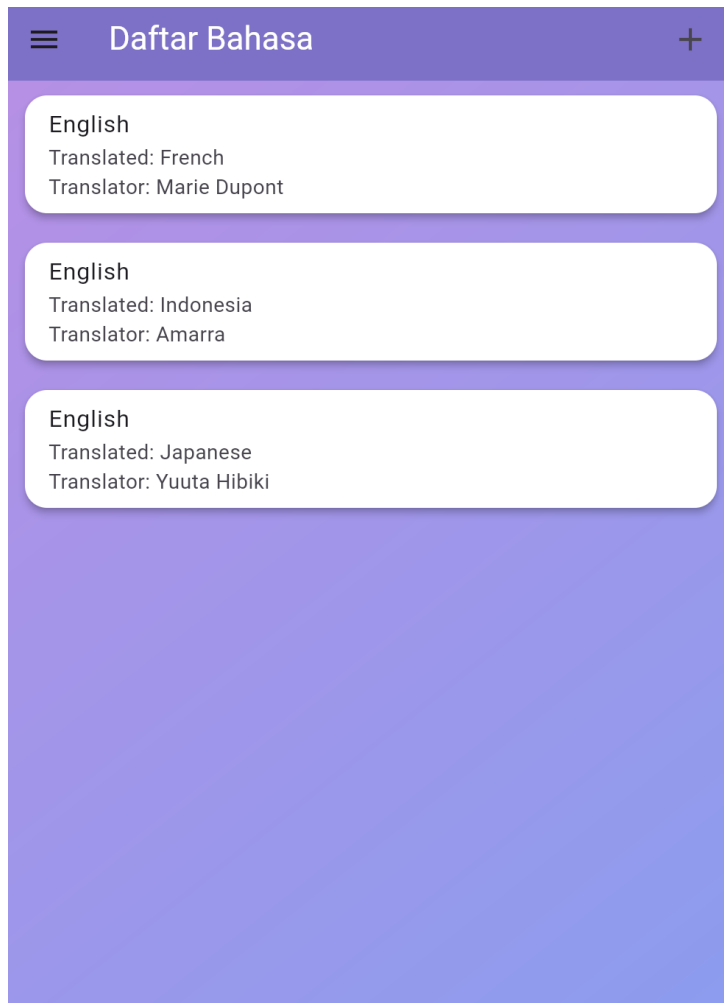


Setelah pengguna mengklik tombol “hapus”, aplikasi akan meminta konfirmasi untuk memastikan bahwa pengguna benar-benar ingin menghapus data tersebut. Konfirmasi ini bertujuan untuk menghindari penghapusan data secara tidak sengaja.

```
static Future<bool> deleteBahasa({required int id}) async {  
  String apiUrl = ApiUrl.deleteBahasa(id);  
  var response = await Api().delete(apiUrl);  
  
  if (response.statusCode == 200) {  
    var jsonObj = json.decode(response.body);  
    return (jsonObj as Map<String, dynamic>)['data'];  
  } else {  
    throw Exception("Failed to delete bahasa");  
  }  
}
```

Setelah konfirmasi diberikan, fungsi `deleteBahasa()` pada *BahasaBloc* dipanggil. Fungsi ini bertanggung jawab untuk menghapus data bahasa yang telah dipilih oleh pengguna.

Di dalam fungsi `deleteBahasa()`, permintaan HTTP DELETE dikirim ke server dengan menggunakan API. Permintaan ini berisi ID bahasa yang ingin dihapus. Server kemudian memproses permintaan dan menghapus data bahasa dari basis data. Jika permintaan penghapusan berhasil, API akan mengembalikan respons yang menyatakan bahwa data bahasa telah berhasil dihapus.



Setelah data bahasa berhasil dihapus, pengguna akan diarahkan kembali ke halaman *BahasaPage*, di mana daftar bahasa akan diperbarui secara otomatis untuk menghapus bahasa yang baru saja dihapus. Tidak ada notifikasi yang ditampilkan setelah proses penghapusan berhasil.