

LAPORAN DOKUMENTASI TUGAS TIGA

DATABASE APLIKASI TO DO LIST



Kelompok 3 :

Ketua : Erlynda Agustina (17)

Anggota : Chella Robiatul A (11)

Devina Putri A (13)

M Galih Satria (24)

REKAYASA PERANGKAT LUNAK & GIM
SMK BRANTAS KARANGKATES

A. Dokumentasi kode

1. Supabase (data base)



```
1 // Inisialisasi Supabase
2 Future<void> initialize() async { //ini
3     await Supabase.initialize(
4         url: 'https://townpxubyzawhrzhusf.supabase.co',
5         anonKey:
6             'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJsb2dpbi5nb29nb2dsZSIsInN1IiwiYmFzZSIsInZlZlI6InJvd25nNhd1YnIGZkdocrpod3NnIiwiMjcsInIiLCjpyYQIOjE3NTUwNDUWMyjctInIiCC0MjA3MDYyMTojOx0.XHuik8hLgab75v4pElpElCoC3aIxCMUihF20CKTLA',
7     );
8     client = Supabase.instance.client;
9 }
```

- ❖ Initialize ()
 - Ini adalah sebuah fungsi `async (asynchronous)`, artinya dia bisa menunggu proses yang memakan waktu (misalnya koneksi internet) tanpa membuat aplikasi nge-lag.
- ❖ `Supabase.initialize(...)`
 - Bagian ini adalah perintah untuk menyambungkan aplikasi ke proyek Supabase kamu.
- ❖ `url` → alamat (URL) proyek Supabase kamu. Itu unik, kayak alamat rumah database kamu.
- ❖ `anonKey` → semacam kunci akses publik. Dengan kunci ini, aplikasi kamu bisa ngobrol dengan Supabase (misalnya login, ambil data, simpan data).
- ❖ `await`
 - Karena butuh waktu untuk menyambungkan ke server Supabase, kita pakai `await` supaya program nunggu proses selesai dulu sebelum lanjut.
- ❖ `client = Supabase.instance.client;`
 - Setelah berhasil tersambung, kita simpan “`client`” (alat buat komunikasi dengan Supabase) ke dalam variabel `client`. Dengan `client` ini, kamu bisa melakukan operasi ke database, misalnya: insert data, ambil data, update data, delete data

```
1  Future<Map<String, dynamic>> registerUser({
2      required String email,
3      required String password,
4      required String name,
5      required String course,
6      DateTime? birthDate,
7      String? phoneNumber,
8  }) async {
9      try {
10          final response = await client.auth.signIn(
11              email: email,
12              password: password,
13          );
14
15          if (response.user != null) {
16              // Simpan data profil ke tabel profiles
17              final profileResult = await client
18                  .from('profiles')
19                  .insert({
20                      'id': response.user!.id,
21                      'email': email,
22                      'name': name,
23                      'course': course,
24                      'birth_date': birthDate?.toIso8601String(),
25                      'phone_number': phoneNumber,
26                      'updated_at': DateTime.now().toIso8601String(),
27                  })
28                  .select()
29                  .single();
30
31          print(
32              'Pengguna terdaftar dan profil disimpan: ${profileResult['email']}');
33          return {
34              'success': true,
35              'message': 'Registrasi berhasil! Silakan login.',
36          };
37      } else {
38          return {
39              'success': false,
40              'message': 'Gagal mendaftar: Tidak ada data pengguna.',
41          };
42      }
43  } catch (e) {
44      print('Kesalahan saat regi
```

- ❖ Bagian ini **mendaftarkan akun user ke Supabase Authentication** (jadi user bisa login pakai email + password).
- ❖ Dicek apakah proses sign up berhasil (user tidak null).
- ❖ Kalau berhasil:
Data tambahan (profil user) disimpan ke tabel **profiles** di database Supabase.
 - id → diisi dengan id user dari Supabase Auth (supaya nyambung antara akun dan profil).
 - email, name, course, birth_date, phone_number → diisi dengan data yang dikirim user.
 - updated_at → otomatis diisi waktu sekarang.
 - select().single() artinya kita minta data yang baru saja disimpan, dan ambil **satu baris saja**.
- ❖ Kalau semua berhasil, aplikasi akan:
 - Cetak pesan di console.

- Mengembalikan Map {'success': true, 'message': 'Registrasi berhasil! Silakan login.'}.
- ❖ Kalau gagal, kasih pesan error biar bisa ditampilkan ke user



```

1 Future<Map<String, dynamic>> loginUser({
2   required String email,
3   required String password,
4 } ) async {
5   try {
6     final response = await client.auth.signInWithEmailAndPassword(
7       email: email,
8       password: password,
9     );
10
11    if (response.user != null) {
12      await _saveUserToPrefs(response.user!);
13      print('Login berhasil: ${response.user!.email}');
14      return {
15        'success': true,
16        'message': 'Login berhasil.',
17      };
18    } else {
19      return {
20        'success': false,
21        'message': 'Gagal login: Tidak ada data pengguna.',
22      };
23    }
24  } catch (e) {
25    print('Kesalahan saat login: $e');
26    return {
27      'success': false,
28      'message': _getErrorMessage(e),
29    };
30  }
31 }

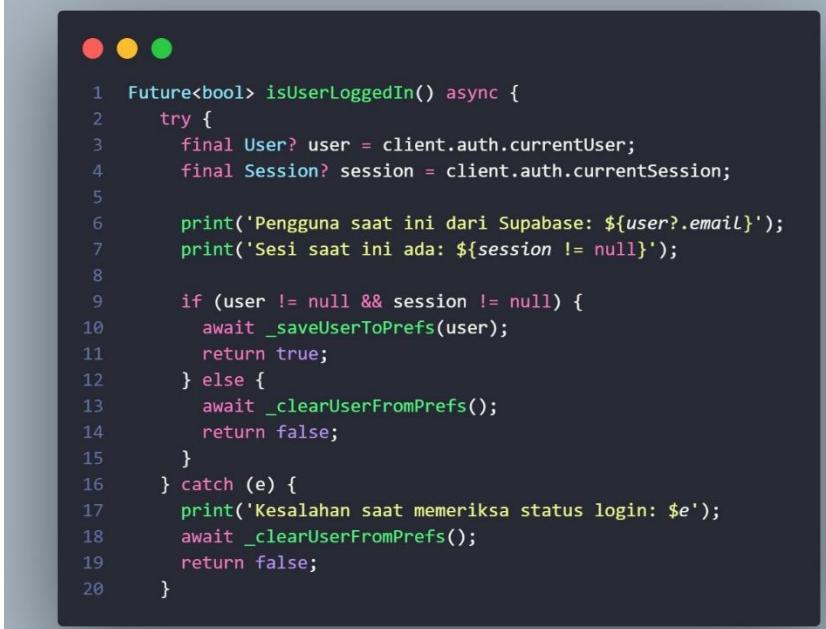
```

- ❖ `_saveUserToPrefs(response.user!)` → simpan data user ke local storage (biar nggak perlu login ulang setiap buka aplikasi).
- ❖ Cetak pesan di console: "Login berhasil: email_user".
- ❖ Kembalikan hasil {'success': true, 'message': 'Login berhasil.'}.
- ❖ Kalau ada **error** (misalnya internet mati, email/password salah format, atau server bermasalah):
 - ❖ Ditangkap di catch.
 - ❖ Cetak pesan error ke console.
 - ❖ Kembalikan hasil gagal dengan pesan error yang lebih jelas lewat `_getErrorMessage(e)`.



```
1 Future<Map<String, dynamic>> loginUser({
2   required String email,
3   required String password,
4 } ) async {
5   try {
6     final response = await client.auth.signInWithEmailAndPassword(
7       email: email,
8       password: password,
9     );
10    if (response.user != null) {
11      await _saveUserToPrefs(response.user!);
12      print('Login berhasil: ${response.user!.email}');
13      return {
14        'success': true,
15        'message': 'Login berhasil.',
16      };
17    } else {
18      return {
19        'success': false,
20        'message': 'Gagal login: Tidak ada data pengguna.',
21      };
22    }
23  } catch (e) {
24    print('Kesalahan saat login: $e');
25    return {
26      'success': false,
27      'message': _getErrorMessage(e),
28    };
29  }
30}
31 }
```

- ❖ Mencoba login ke server
- ❖ client.auth.signInWithEmailAndPassword(email, password) akan mengecek apakah email dan password cocok dengan data pengguna di server.
- ❖ Kalau login berhasil (user ditemukan)
- ❖ Data user (response.user) tidak kosong → berarti email & password benar.
- ❖ Fungsi _saveUserToPrefs dipanggil untuk menyimpan data user ke penyimpanan lokal (prefs) agar tidak perlu login ulang nanti.
- ❖ Menampilkan pesan di konsol: "Login berhasil: (email user)".
Kalau login gagal (user tidak ada)
- ❖ response.user kosong → berarti email atau password salah.
Kalau ada kesalahan lain (misalnya server error, internet mati, dll.)
- ❖ Bagian catch (e) menangkap error.
- ❖ Menampilkan pesan error di konsol.



```
1 Future<bool> isUserLoggedIn() async {
2     try {
3         final User? user = client.auth.currentUser;
4         final Session? session = client.auth.currentSession;
5
6         print('Pengguna saat ini dari Supabase: ${user?.email}');
7         print('Sesi saat ini ada: ${session != null}');
8
9         if (user != null && session != null) {
10             await _saveUserToPrefs(user);
11             return true;
12         } else {
13             await _clearUserFromPrefs();
14             return false;
15         }
16     } catch (e) {
17         print('Kesalahan saat memeriksa status login: $e');
18         await _clearUserFromPrefs();
19         return false;
20     }
}
```

- ❖ Ambil data user dan sesi saat ini
- ❖ client.auth.currentUser → cek apakah ada data pengguna yang sedang login.
- ❖ client.auth.currentSession → cek apakah ada sesi login aktif (token yang valid).
- ❖ Cetak informasi ke konsol (debugging)
- ❖ Menampilkan email user kalau ada (user?.email).
- ❖ Menampilkan apakah sesi ada atau tidak (session != null).
- ❖ Kalau user dan sesi valid (dua-duanya tidak null)
- ❖ Simpan lagi data user ke penyimpanan lokal (_saveUserToPrefs(user)) supaya aplikasi ingat pengguna.
- ❖ Kembalikan nilai true → artinya pengguna masih login.
Kalau user atau sesi tidak ada
- ❖ Bersihkan data user dari penyimpanan lokal (_clearUserFromPrefs()).
- ❖ Kembalikan nilai false → artinya pengguna tidak login atau sesi sudah habis.
- ❖ Kalau ada error (misalnya koneksi putus atau token rusak)
- ❖ Tampilkan pesan error di konsol.
- ❖ Hapus data user dari penyimpanan lokal.



```
1 Future<void> logoutUser() async {
2     try {
3         print('Mengeluarkan pengguna...');  
4         await client.auth.signOut();
5         await _clearUserFromPrefs();
6         print('Logout berhasil');
7     } catch (e) {
8         print('Kesalahan logout: $e');
9         await _clearUserFromPrefs();
10    }
11 }
```

- ❖ Perintah ini **nyuruh Supabase untuk mengeluarkan user** (logout dari server).
- ❖ Habis logout dari server, aplikasi juga **hapus data user yang tersimpan di HP** (biasanya di SharedPreferences)
 - ❖ Biar data user benar-benar bersih, dan kalau buka aplikasi lagi harus login ulang.
- ❖ print('Logout berhasil'); Kalau semua langkah berhasil, cetak pesan ke console bahwa logout sukses.
- ❖ catch (e) {
 print('Kesalahan logout: \$e');
 await _clearUserFromPrefs();
 }
 - ❖ Kalau ada error (misalnya koneksi internet mati atau server bermasalah):
 - ❖ Cetak pesan error ke console.
 - ❖ Tapi tetap jalankan _clearUserFromPrefs(); biar data user di HP tetap dihapus.



```
1 // Simpan data pengguna ke SharedPreferences
2 Future<void> _saveUserToPrefs(User user) async {
3     try {
4         final prefs = await SharedPreferences.getInstance();
5         await prefs.setString('user_id', user.id);
6         await prefs.setString('user_email', user.email ?? '');
7         print('Data pengguna disimpan ke preferensi: ${user.email}');
8     } catch (e) {
9         print('Kesalahan saat menyimpan preferensi pengguna: $e');
10    }
11 }
```

- ❖ **SharedPreferences.getInstance()**
→ minta akses ke “lemari penyimpanan” aplikasi.
- ❖ **prefs.setString('user_id', user.id)**
→ masukkan data id user ke dalam lemari dengan label user_id.

- ❖ **prefs.setString('user_email', user.email ?? '')**
→ simpan email user. Kalau email kosong/null, dia otomatis jadi string kosong ''.
- ❖ **print(...)**
→ cuma buat ngecek di console, apakah data berhasil disimpan.
- ❖ **catch (e)**
→ kalau ada masalah (misalnya gagal nyimpan), error-nya ditampilkan di console.



```

1 Future<void> _clearUserFromPrefs() async {
2   try {
3     final prefs = await SharedPreferences.getInstance();
4     await prefs.clear();
5     print('Data pengguna dihapus dari preferensi');
6   } catch (e) {
7     print('Kesalahan saat menghapus preferensi pengguna: $e');
8   }
9 }

```

(Bersihkan data pengguna dari SharedPreferences)

- ❖ **SharedPreferences.getInstance()**
→ buka akses ke “lemari penyimpanan” aplikasi.
- ❖ **prefs.clear()**
→ hapus semua data yang ada di dalamnya (id, email, atau apapun yang pernah disimpan).
- ❖ **Print ('Data pengguna dihapus...')**
→ hanya untuk info di console kalau data sudah berhasil dihapus.
- ❖ **catch (e)**
→ kalau ada error saat hapus data, error-nya ditampilkan.



```
1 Future<Map<String, dynamic>> getCurrentUser() async {
2     try {
3         final user = client.auth.currentUser;
4         if (user != null) {
5             return {
6                 'id': user.id,
7                 'email': user.email,
8             };
9         }
10        return {
11            'id': null,
12            'email': null,
13        };
14    } catch (e) {
15        print('Kesalahan saat mengambil pengguna saat ini: $e');
16        return {
17            'id': null,
18            'email': null,
19        };
20    }
21 }
```

❖ **client.auth.currentUser**

→ cek ke sistem autentikasi (Supabase) untuk tahu apakah ada user yang sedang login.

❖ **if (user != null)**

→ kalau ada user yang login, ambil datanya.

❖ **return {'id': user.id, 'email': user.email}**

→ balikin data user dalam bentuk **map** (seperti kotak berisi pasangan kunci dan nilai).

❖ **atau tidak ada user**

→ balikin id: null, email: null biar jelas kalau belum ada yang login.

❖ **catch (e)**

→ kalau terjadi error pas ngambil data, dia juga balikin null untuk id dan email, lalu print error-nya di console.



```
1 Future<Map<String, dynamic>> saveUserProfile({
2   required String userId,
3   required String email,
4   required String name,
5   String? bio,
6   DateTime? birthDate,
7   String? phoneNumber,
8   String? course,
9 }) async {
10   try {
11     final response = await client
12       .from('profiles')
13       .upsert({
14         'id': userId,
15         'email': email,
16         'name': name,
17         'bio': bio,
18         'birth_date': birthDate?.toISOString(),
19         'phone_number': phoneNumber,
20         'course': course,
21         'updated_at': DateTime.now().toISOString(),
22       }, onConflict: 'id')
23       .select()
24       .single();
25
26     print('Profil disimpan: ${response['email']}');
27     return {
28       'success': true,
29       'message': 'Profil berhasil disimpan.',
30       'data': response,
31     };
32   } catch (e) {
33     print('Kesalahan saat menyimpan profil: $e');
34     return {
35       'success': false,
36       'message': 'Gagal menyimpan profil: ${e.toString()}',
37     };
38   }
39 }
```

- ❖ Parameter (data yang masuk ke fungsi):
 - EserId → id user (unik, biasanya dari Supabase Auth).
 - Email → email user.
 - Name → nama user.
 - Bio, birthDate, phoneNumber, course → data tambahan (opsional).
- ❖ **from('profiles')**
 - pilih tabel profiles di database Supabase.
- ❖ **upsert({...})** artinya
 - kalau **data dengan id itu belum ada**, maka **buat data baru**.
 - kalau **data dengan id itu sudah ada**, maka **update data lama**.
- ❖ **onConflict: 'id'**
 - aturan konflik: kalau ada id yang sama, maka update data yang sudah ada, bukan bikin baru.
- ❖ **.select().single()**
 - setelah simpan, ambil hasil data terbaru (satu baris saja).
- ❖ **print(...)**
 - hanya buat ngecek di console, email mana yang berhasil disimpan.



```
1 Future<Map<String, dynamic>> getUserProfile(String userId) async {
2     try {
3         final response =
4             await client.from('profiles').select().eq('id', userId).maybeSingle();
5
6         if (response != null) {
7             print('Profil ditemukan: ${response['email']}');
8             return {
9                 'success': true,
10                'data': {
11                    'id': response['id'],
12                    'email': response['email'],
13                    'name': response['name'] ?? '',
14                    'bio': response['bio'] ?? '',
15                    'birth_date': response['birth_date'] != null
16                        ? DateTime.parse(response['birth_date'])
17                        : null,
18                    'phone_number': response['phone_number'] ?? '',
19                    'course': response['course'] ?? '',
20                },
21            };
22        } else {
23            print('Profil tidak ditemukan untuk userId: $userId');
24            return {
25                'success': false,
26                'message': 'Profil tidak ditemukan.',
27            };
28        }
29    } catch (e) {
30        print('Terjadi kesalahan: $e');
31    }
32}
```

- ❖ **client.from('profiles').select().eq('id', userId)**
→ artinya: ambil data dari tabel profiles yang id-nya sama dengan userId yang kita kirim.
- ❖ **maybeSingle ()**
→ ambil **satu baris** saja. Kalau datanya nggak ada, hasilnya null.
- ❖ **if (response! = null)**
 - Kalau ketemu profil → kembalikan data profil (id, email, name, bio, tanggal lahir, nomor HP, course).
 - Kalau tidak ketemu → balikin pesan Profil tidak ditemukan.
- ❖ **DateTime.parse(response['birth_date'])**
→ kalau ada tanggal lahir, diubah dari string database ke tipe DateTime supaya gampang dipakai di Flutter.
- ❖ **catch (e)**
→ kalau ada error pas ambil data (misalnya koneksi putus), balikin pesan gagal.



```
1 String _getErrorMessage(dynamic e) {
2     if (e is AuthException) {
3         switch (e.message) {
4             case 'Invalid login credentials':
5                 return 'Email atau kata sandi salah.';
6             case 'User already registered':
7                 return 'Email sudah terdaftar.';
8             default:
9                 return 'Terjadi kesalahan: ${e.message}';
10        }
11    }
12    return 'Terjadi kesalahan: ${e.toString()}';
13 }
14
15 Future<Map<String, dynamic>> getUserTasks(String userId) async {
16     try {
17         final response =
18             await client.from('tasks').select().eq('user_id', userId);
19
20         return {
21             'success': true,
22             'data': response,
23         };
24     } catch (e) {
25         print('Kesalahan saat mengambil tasks: $e');
26         return {
27             'success': false,
28             'message': e.toString(),
29         };
30     }
31 }
```

- ❖ Kalau error-nya **Invalid login credentials** → kasih pesan ke user: "*Email atau kata sandi salah.*"
- ❖ Kalau error-nya **User already registered** → kasih pesan: "*Email sudah terdaftar.*"
- ❖ Kalau error lain → kasih pesan umum: "*Terjadi kesalahan: [isi error].*"
- ❖ Kalau bukan AuthException → tetap balikin error dalam bentuk teks.
- ❖ `_getErrorMessage` = buat nerjemahin error login supaya user ngerti.
- ❖ `getUserTasks` = buat ngambil daftar tugas user dari database.
- ❖ `client.from('tasks').select().eq('user_id', userId)`
→ ambil semua data dari tabel tasks yang punya user_id sesuai dengan user yang login.

2. Splash screen



```
1 class _SplashScreenState extends State<SplashScreen> {
2   final SupabaseService _supabaseService = SupabaseService();
3
4   // Controllers untuk form
5   final TextEditingController _registerEmailController =
6     TextEditingController();
7   final TextEditingController _registerNameController = TextEditingController();
8   final TextEditingController _registerCourseController =
9     TextEditingController();
10  final TextEditingController _registerPhoneController =
11    TextEditingController();
12  final TextEditingController _registerPasswordController =
13    TextEditingController();
14  final TextEditingController _registerConfirmPasswordController =
15    TextEditingController();
16  final TextEditingController _loginEmailController = TextEditingController();
17  final TextEditingController _loginPasswordController =
18    TextEditingController();
19
20  // State untuk tanggal lahir
21  DateTime? _registerBirthDate;
22
23  // State untuk password visibility
24  bool _isRegisterPasswordVisible = false;
25  bool _isRegisterConfirmPasswordVisible = false;
26  bool _isLoginPasswordVisible = false;
27
28  // State untuk loading
29  bool _isRegisterLoading = false;
30  bool _isLoginLoading = false;
31
32  String get formatTanggal {
33    if (_registerBirthDate == null) return 'Pilih tanggal lahir';
34    return DateFormat('dd MMMM yyyy').format(_registerBirthDate);
35  }
36
37  @override
38  void initState() {
39    super.initState();
40    _checkUserLogin();
41  }
42
43  // Check if user is already logged in
44  Future<void> _checkUserLogin() async {
45    final isLoggedIn = await supabaseService.isUserLoggedIn();
46    print('isLoggedIn: $isLoggedIn');
47    if (isLoggedIn && mounted) {
48      Navigator.pushReplacement(
49        context,
50        MaterialPageRoute(builder: (context) => const HomeScreen()),
51      );
52    }
53  }
54
55  // Register function
56  Future<void> register() async {
57    if (_registerPasswordController.text != _registerConfirmPasswordController.text) {
58      _showSnackBar('Kata sandi tidak cocok!', isError: true);
59      return;
60    }
61
62    if (_registerEmailController.text.isEmpty ||
63        _registerNameController.text.isEmpty ||
64        _registerCourseController.text.isEmpty ||
65        _registerPhoneController.text.isEmpty ||
66        _registerPasswordController.text.isEmpty ||
67        _registerBirthDate == null) {
68      _showSnackBar('Harap isi semua kolom!', isError: true);
69      return;
70    }
71
72    setState(() {
73      _isRegisterLoading = true;
74    });
75  }
```

- ❖ Ini dipakai untuk menyimpan dan membaca input dari TextField (kolom teks).
Misalnya user ngetik email di form → datanya otomatis tersimpan di `_registerEmailController.text`.
- ❖ `DateTime? _registerBirthDate;`
Variabel ini dipakai buat nyimpen tanggal lahir user. null artinya user belum pilih tanggal.
- ❖ `bool _isRegisterPasswordVisible = false;`
`bool _isRegisterConfirmPasswordVisible = false;`
`bool _isLoginPasswordVisible = false;`
Variabel untuk atur apakah password ditampilkan atau disembunyikan (••••). Default = false (disembunyikan).
- ❖ Saat aplikasi pertama kali dibuka (initState), fungsi ini dijalankan:
 - ❖ Cek apakah user sudah login sebelumnya.
 - ❖ Kalau iya → langsung pindah ke HomeScreen.
 - ❖ Kalau tidak → tetap di halaman login/register.
 - ❖ Ini adalah logika daftar akun baru (register):

- ❖ **Cek password & konfirmasi** → kalau beda → tampilkan pesan error “*Kata sandi tidak cocok!*”.
- ❖ **Cek apakah semua kolom diisi** → kalau ada yang kosong → tampilkan pesan “*Harap isi semua kolom!*”.
- ❖ Kalau semua sudah benar → ubah state `_isRegisterLoading = true` → biasanya supaya UI nunjukin loading (“sedang proses daftar”).



- ❖ **Panggil fungsi registerUser di SupabaseService**
→ Ini artinya kita ngirim data yang sudah diisi di form (email, password, nama, dll) ke **Supabase** untuk dibuatkan akun baru.
- ❖ `email: _registerEmailController.text.trim()`
Ambil email dari kolom input, trim () buat hapus spasi depan & belakang.
- ❖ `password: _registerPasswordController.text`
Ambil password yang ditulis user.
- ❖ `name: _registerNameController.text.trim()`
Ambil nama user.
- ❖ `course: _registerCourseController.text.trim()`
Ambil jurusan/kursus yang dipilih.
- ❖ `birthDate: _registerBirthDate`
Ambil tanggal lahir yang dipilih user.
- ❖ `phoneNumber: _registerPhoneController.text.trim()`
Ambil nomor HP user.
- ❖ **await**
→ artinya tunggu proses pendaftaran selesai dulu (karena komunikasi ke server butuh waktu). Hasilnya nanti disimpan ke variabel result
- ❖ `setState(() { _isRegisterLoading = false; })`
→ Setelah proses daftar selesai (berhasil atau gagal), loading dihentikan. Jadi tombol "Daftar" bisa dipencet lagi, atau indikator loading hilang



```
1 if (result['success']) {
2     // Clear register form
3     _registerEmailController.clear();
4     _registerNameController.clear();
5     _registerCourseController.clear();
6     _registerPhoneController.clear();
7     _registerPasswordController.clear();
8     _registerConfirmPasswordController.clear();
9     _registerBirthDate = null;
10
```

❖ **if (result['success'])**

Mengecek apakah proses register berhasil.

- result adalah hasil dari registerUser.
- Kalau success = true → artinya akun berhasil dibuat.
- Kalau false → berarti gagal (misalnya email sudah dipakai).

❖ **Bersihin semua form input**

Kalau berhasil, maka semua kolom form didikosongkan lagi supaya siap dipakai user lain atau supaya form rapi.

- `_registerEmailController.clear();` → hapus teks di kolom email.
- `_registerNameController.clear();` → hapus teks di kolom nama.
- `_registerCourseController.clear();` → hapus teks di kolom jurusan.
- `_registerPhoneController.clear();` → hapus teks di kolom nomor HP.
- `_registerPasswordController.clear();` → hapus teks di kolom password.
- `_registerConfirmPasswordController.clear();` → hapus teks di kolom konfirmasi password.
- `_registerBirthDate = null;` → reset tanggal lahir ke kosong lagi.



```
1 Future<void> _login() async {
2     if (_loginEmailController.text.isEmpty ||
3         _loginPasswordController.text.isEmpty) {
4         _showSnackBar('Harap isi semua kolom!', isError: true);
5         return;
6     }
7
8     setState(() {
9         _isLoginLoading = true;
10    });
11
12    final result = await _supabaseService.loginUser(
13        email: _loginEmailController.text.trim(),
14        password: _loginPasswordController.text,
15    );
16
17    setState(() {
18        _isLoginLoading = false;
19    });
20
21    if (result['success']) {
22        // Clear login form
23        _loginEmailController.clear();
24        _loginPasswordController.clear();
25
26        _showSnackBar(result['message'], isError: false);
27        Navigator.pop(context); // Close modal
28        Navigator.pushAndRemoveUntil(
29            context,
30            MaterialPageRoute(builder: (context) => const HomeScreen()),
31            (route) => false,
32        );
33    } else {
34        _showSnackBar(result['message'], isError: true);
35    }
36}
```

- ❖ Membuat fungsi login yang jalan secara **asynchronous** (pakai Future dan async) karena ada proses ambil data dari server.
- ❖ Kalau kolom **email** atau **password** masih kosong → tampilkan pesan error ("Harap isi semua kolom!") lalu hentikan proses login.



```
1 isVisible: _isRegisterPasswordVisible,
2
3 onToggleVisibility: () {
4     setModalState(() {
5         _isRegisterPasswordVisible =
6             !_isRegisterPasswordVisible;
7     });
8 },
```

- ❖ **isVisible: _isRegisterPasswordVisible**
Ini adalah **status apakah password ditampilkan atau disembunyikan** (misalnya pakai simbol ●●● atau tulisan asli).
- ❖ **onToggleVisibility: () {**
Ini fungsi yang dipanggil saat user menekan ikon "mata" ☺ di kolom password.
- ❖ **setModalState(() { ... })** dipakai supaya tampilan **langsung berubah** setelah user menekan tombol.



```
1 isVisible: _isRegisterConfirmPasswordVisible,
2                               onToggleVisibility: () {
3                               setModalState(() {
4                               _isRegisterConfirmPasswordVisible =
5                               !_isRegisterConfirmPasswordVisible;
6                               });
7                               },
```

- ❖ **isVisible: _isRegisterConfirmPasswordVisible**
Ini menyimpan **status apakah password konfirmasi terlihat atau tidak.**
- ❖ **onToggleVisibility**
Ini adalah fungsi yang jalan saat user menekan tombol ikon "mata"  di kolom konfirmasi password.
- ❖ Kode ini dipakai untuk **fitur show/hide di kolom konfirmasi password.**



```
1 isVisible: _isLoginPasswordVisible,
2                               onToggleVisibility: () {
3                               setModalState(() {
4                               _isLoginPasswordVisible =
5                               !_isLoginPasswordVisible;
6                               });
7                               },
```

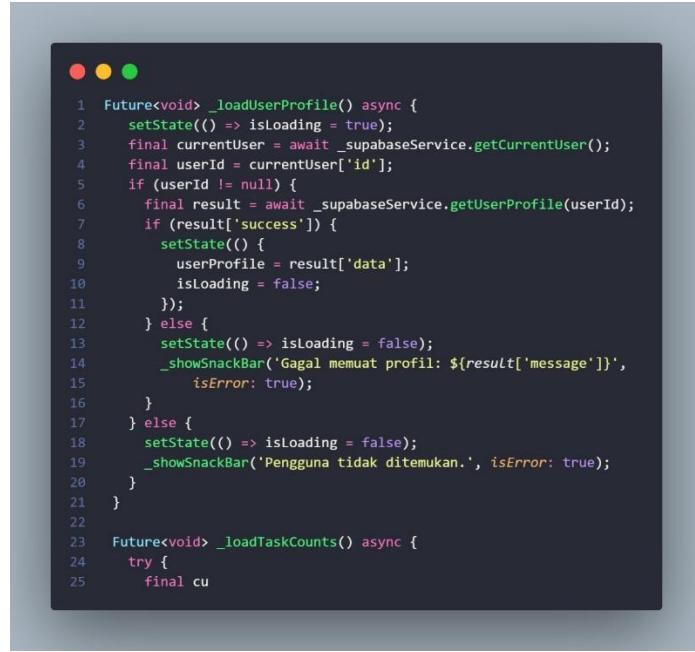
- ❖ Kode ini bikin **fitur show/hide password di form login.**



```
1 Widget _buildPasswordField({
2     required TextEditingController controller,
3     required String hintText,
4     required String labelText,
5     required bool isVisible,
6     required VoidCallback onToggleVisibility,
7 }) {
```

- ❖ **TextEditingController controller**
 - Ini adalah "alat penghubung" antara kolom input dan kode program.
 - Fungsinya buat **menyimpan/mengambil teks yang diketik user** di kolom password.
- ❖ buildPasswordField ini seperti **cetakan** untuk membuat kolom password yang:
 - ❖ ada teks petunjuk (hint)
 - ❖ ada label (judul kolom)
 - ❖ bisa show/hide password dengan tombol mata 

3. Home screen



```
1 Future<void> _loadUserProfile() async {
2     setState(() => isLoading = true);
3     final currentUser = await _supabaseService.getCurrentUser();
4     final userId = currentUser['id'];
5     if (userId != null) {
6         final result = await _supabaseService.getUserProfile(userId);
7         if (result['success']) {
8             setState(() {
9                 userProfile = result['data'];
10                isLoading = false;
11            });
12        } else {
13            setState(() => isLoading = false);
14            _showSnackBar('Gagal memuat profil: ${result['message']}', isError: true);
15        }
16    } else {
17        setState(() => isLoading = false);
18        _showSnackBar('Pengguna tidak ditemukan.', isError: true);
19    }
20 }
21
22 Future<void> _loadTaskCounts() async {
23     try {
24         final cu
```

- ❖ `_loadUserProfile()` bertugas untuk mengambil data profil user dari server (Supabase) lalu menampilkannya di aplikasi.
- ❖ Memanggil service untuk cek siapa user yang sedang login. Kalau tidak ada user yang login, userId akan null.
- ❖ Mengambil data lengkap user berdasarkan userId. Hasilnya disimpan di result.
- ❖ Kalau server bilang ada masalah (success == false), tampilkan snackbar error berisi pesan dari server. Kalau server bilang ada masalah (success == false), tampilkan snackbar error berisi pesan dari server.
- ❖ Kalau server bilang ada masalah (success == false), tampilkan snackbar error berisi pesan dari server.



```
1 Future<void> _loadTaskCounts() async {
2     try {
3         final currentUser = await _supabaseService.getCurrentUser();
4         final userId = currentUser['id'];
5         if (userId != null) {
6             final result = await _supabaseService.getUserTasks(userId);
7             if (result['success']) {
8                 final tasks = result['data'] as List<dynamic>;
9                 setState(() {
10                     todoCount = tasks.where((t) => t['done'] == false).length;
11                     doneCount = tasks.where((t) => t['done'] == true).length;
12                 });
13             }
14         } catch (e) {
15             _showSnackBar('Gagal load task count: $e', isError: true);
16         }
17     }
18 }
```

`_loadTaskCounts()` dipakai untuk menghitung jumlah tugas yang belum selesai dan yang sudah selesai milik user yang sedang login, lalu memperbarui tampilan aplikasi.

Mengambil informasi user saat ini dari Supabase.
Kalau user tidak login, userId akan null (dan kode berhenti di situ).

Memanggil service untuk ambil daftar tugas user tersebut.
Hasilnya berupa result dengan dua bagian:
`result['success']` → apakah pemanggilan berhasil
`result['data']` → daftar tugasnya

tasks adalah list dari semua tugas.

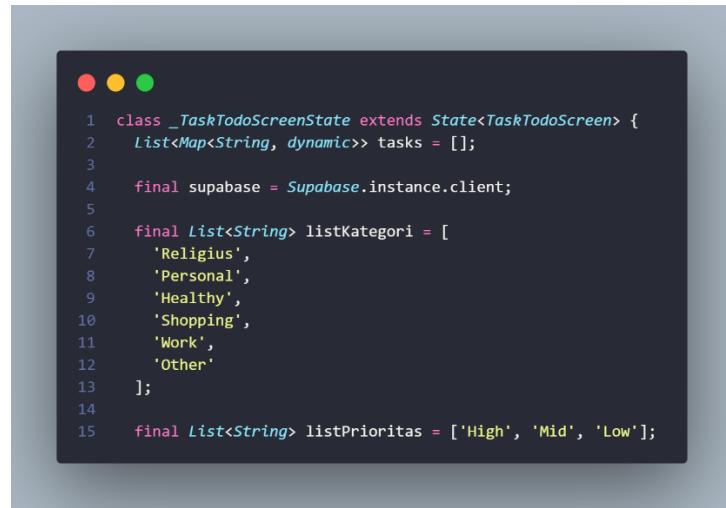
`todoCount` → jumlah tugas yang belum selesai (`done == false`).
`doneCount` → jumlah tugas yang sudah selesai (`done == true`).
`setState()` dipakai supaya tampilan aplikasi langsung ikut berubah.

Jika terjadi kesalahan saat ambil data, akan muncul snackbar berisi pesan error.



- ❖ `_refreshHome()` dipakai untuk memperbarui tampilan halaman utama (Home).
- ❖ `await _loadUserProfile()`
Memanggil fungsi untuk mengambil data profil user terbaru dari server.
Misalnya: nama, email, foto profil, dll.
- ❖ `await _loadTaskCounts()`
Memanggil fungsi untuk mengambil jumlah tugas terbaru milik user. Menghitung berapa yang sudah selesai dan berapa yang belum selesai. Karena pakai await, Baris kedua hanya dijalankan setelah baris pertama selesai.
Jadi data profil dimuat dulu, lalu data tugas dihitung.

4. TO DO



```
1 class _TaskTodoScreenState extends State<TaskTodoScreen> {
2     List<Map<String, dynamic>> tasks = [];
3 
4     final supabase = Supabase.instance.client;
5 
6     final List<String> listKategori = [
7         'Religious',
8         'Personal',
9         'Healthy',
10        'Shopping',
11        'Work',
12        'Other'
13    ];
14 
15     final List<String> listPrioritas = ['High', 'Mid', 'Low'];
```

- ❖ Ambil ID user yang sedang login
- ❖ Ambil semua task milik user itu dari Supabase
- ❖ Urutkan task berdasarkan prioritas (High → Mid → Low)
- ❖ Format data (tanggal, kategori, dll) supaya rapi
- ❖ Buat list hanya untuk task yang belum selesai
- ❖ Tampilkan ke aplikasi
- ❖ Kalau gagal, kasih notifikasi error



```
1 void showEditBottomSheet(Map<String, dynamic> task, int index) {
2     final titleController = TextEditingController(text: task['title']);
3     final notesController = TextEditingController(text: task['description']);
4     String kategoriTerpilih = task['category'];
5     String prioritasTerpilih = task['priority'];
```

- ❖ Mempersiapkan form edit untuk sebuah task.
- ❖ Mengisi form dengan data lama (judul, catatan, kategori, prioritas) supaya tidak kosong.
- ❖ Supaya user bisa edit data lama dengan mudah tanpa harus ketik ulang dari nol.
- ❖ Nyimpen prioritas task yang lama ke variabel prioritasTerpilih.
 - Contoh: "High", "Mid", atau "Low"
- ❖ Bikin controller untuk TextField judul.
 - TextEditingController dipakai supaya input teks bisa dikontrol (misalnya ngambil isinya, atau mengganti isinya).
 - Di sini langsung diisi dengan task['title'], jadi TextField otomatis terisi dengan judul tugas lama, dan bisa diedit.

```
● ● ●
1  await supabase.from('tasks').update({
2      'title': titleController.text,
3      'category': kategoriTerpilih,
4      'priority': prioritasTerpilih,
5      'notes': notesController.text,
6      'date': formattedDate,
7      'time': formattedTime,
8    }).on('id', task['id']);
9
10   setState(() {
11     tasks[index]['title'] = titleController.text;
12     tasks[index]['category'] = kategoriTerpilih;
13     tasks[index]['priority'] = prioritasTerpilih;
14     tasks[index]['description'] = notesController.text;
15     tasks[index]['icon'] = getCategoryIcon(kategoriTerpilih);
16     tasks[index]['date'] = formattedDate;
17     tasks[index]['time'] = formattedTime;
18     tasks[index]['subtitle'] =
19       "$prioritasTerpilih • $formattedDate";
20   });
21
22   Navigator.pop(context);
23 },
24 child: const Text('Update Task'),
25 ),
26 ],
27 ),
```

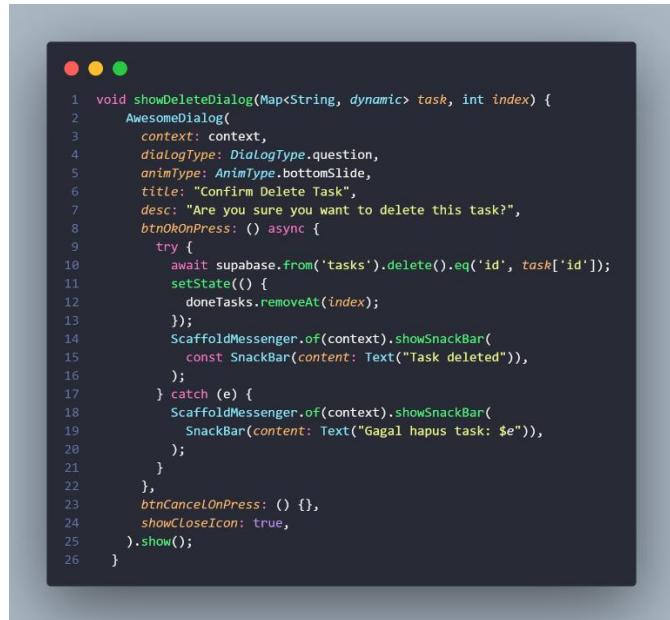
- ❖ `setState(() {...})` → biar tampilan langsung berubah (refresh otomatis).
- ❖ `tasks[index]` → ambil task yang diedit sesuai urutan (index-nya).
- ❖ Terus semua nilai lama diganti dengan yang baru.
- ❖ `getCategoryIcon(kategoriTerpilih)` → ambil icon sesuai kategori barunya.
- ❖ `subtitle` → diubah jadi "Prioritas • Tanggal" biar kelihatan rapi
- ❖ `supabase.from('tasks').update({...})` → ngasih tau ke database: "update data di tabel tasks dengan isi yang baru".

5. Task DONE



```
1 class _TaskDoneScreenState extends State<TaskDoneScreen> {
2     final supabase = supabase.instance.client;
3     List<Map<String, dynamic>> doneTasks = [];
4
5     @override
6     void initState() {
7         super.initState();
8         fetchDoneTasks();
9     }
10
11 Future<void> fetchDoneTasks() async {
12     try {
13         final userId = supabase.auth.currentUser!.id;
14
15         final response = await supabase
16             .from('tasks')
17             .select()
18             .eq('user_id', userId)
19             .eq('done', true);
20
21         setState(() {
22             doneTasks = List<Map<String, dynamic>>.from(response);
23         });
24     } catch (e) {
25         ScaffoldMessenger.of(context).showSnackBar(
26             SnackBar(content: Text('Gagal ambil task done: $e')),
27         );
28     }
29 }
30
31 IconData getCategoryIcon(String category) {
32     switch (category.toLowerCase()) {
33         case "religius":
34             return Icons.self_improvement;
35         case "personal":
36             return Icons.person;
37         case "healthy":
38             return Icons.fitness_center;
39         case "shopping":
40             return Icons.shopping_cart;
41         case "work":
42             return Icons.work;
43         case "other":
44             default:
45                 return Icons.task;
46     }
47 }
```

- ❖ Ini bikin state/penyimpanan data untuk halaman "Task Done".
- ❖ supabase → koneksi ke database Supabase.
- ❖ doneTasks → tempat menyimpan daftar tugas yang statusnya sudah selesai (done = true).
- ❖ Fungsi ini untuk mengambil daftar tugas yang selesai dari Supabase.
- ❖ supabase.auth.currentUser!.id → ambil id user yang sedang login, supaya cuma ambil data milik user itu, bukan semua orang.
- ❖ .from ('tasks') → ambil data dari tabel tasks.
- ❖ .select () → ambil semua kolom data.
- ❖ .eq('user_id', userId) → hanya ambil task milik user login.
- ❖ .eq('done', true) → hanya ambil task yang sudah selesai.



```
1 void showDeleteDialog(Map<String, dynamic> task, int index) {
2     AwesomeDialog(
3         context: context,
4         dialogType: DialogType.question,
5         animType: AnimType.bottomSlide,
6         title: "Confirm Delete Task",
7         desc: "Are you sure you want to delete this task?",
8         btnOKOnPress: () async {
9             try {
10                 await supabase.from('tasks').delete().eq('id', task['id']);
11                 setState(() {
12                     doneTasks.removeAt(index);
13                 });
14                 ScaffoldMessenger.of(context).showSnackBar(
15                     const SnackBar(content: Text("Task deleted")),
16                 );
17             } catch (e) {
18                 ScaffoldMessenger.of(context).showSnackBar(
19                     SnackBar(content: Text("Gagal hapus task: $e")),
20                 );
21             }
22         },
23         btnCancelOnPress: () {},
24         showCloseIcon: true,
25     ).show();
26 }
```

- ❖ Ini adalah fungsi bernama showDeleteDialog.
- ❖ task → data task yang mau dihapus.
- ❖ index → posisi task itu di dalam daftar doneTasks.
- ❖ dialogType: DialogType.question → tampilkan icon tanda tanya ? .
- ❖ animType: AnimType.bottomSlide → dialog muncul dengan animasi dari bawah ke atas.
- ❖ title → judul dialog: "Confirm Delete Task".
- ❖ desc → deskripsi/pertanyaan konfirmasi: "Are you sure you want to delete this task?".
- ❖ Fungsi ini dipanggil waktu user mau hapus task.
- ❖ Muncul dialog konfirmasi dengan pilihan OK atau Cancel.
- ❖ Kalau user pilih OK → task dihapus dari Supabase + daftar lokal (doneTasks).
- ❖ Kalau Cancel → dialog ditutup, tidak ada yang berubah.



```
1 Future<void> uncheckTask(Map<String, dynamic> task, int index) async {
2     try {
3         await supabase.from('tasks').update({'done': false}).eq('id', task['id']);
4         setState(() {
5             doneTasks.removeAt(index);
6         });
7     } catch (e) {
8         ScaffoldMessenger.of(context).showSnackBar(
9             SnackBar(content: Text("Gagal update task: $e")),
10        );
11    }
12 }
```

- ❖ Fungsi ini dipanggil ketika user mau batalkan task selesai (uncheck).
- ❖ Task itu diubah di database Supabase → status done = false.
- ❖ Task langsung dihapus dari daftar doneTasks di aplikasi (biar layar update).
- ❖ Kalau gagal, munculin pesan error.
- ❖ doneTasks.removeAt(index); → hapus task itu dari daftar doneTasks (task selesai).

6. Create



```
1 final titleController = TextEditingController();
2 final notesController = TextEditingController();
3
4 final List<String> listKategori = [
5   'Religius',
6   'Personal',
7   'Healthy',
8   'Shopping',
9   'Work',
10  'Other'
11 ];
12 final List<String> listPrioritas = ['High', 'Mid', 'Low'];
13
14 String? kategoriTerpilih;
15 String? prioritasTerpilih;
16 DateTime? tanggalTerpilih;
17 TimeOfDay? waktuTerpilih;
18
19 final supabase = Supabase.instance.client;
20
21 String get formatTanggal {
22   if (tanggalTerpilih == null) return '';
23   return DateFormat('MMMM d, yyyy').format(tanggalTerpilih);
24 }
25
26 String get formatWaktu {
27   if (waktuTerpilih == null) return '';
28   final now = DateTime.now();
29   final dt = DateTime(now.year, now.month, now.day, waktuTerpilih!.hour,
30     waktuTerpilih!.minute);
31   return DateFormat('HH:mm').format(dt);
32 }
33
34 Future<void> simpanData() async {
35   try {
36     final session = supabase.auth.currentSession;
37     if (session == null) {
38       throw Exception('User belum login!');
39     }
40
41     final userId = supabase.auth.currentUser!.id;
42
43     // Insert data ke Supabase
44     await supabase.from('tasks').insert({
45       'title': titleController.text,
46       'category': kategoriTerpilih,
47       'priority': prioritasTerpilih,
48       'date': formatTanggal,
49       'time': formatWaktu,
50       'notes': notesController.text,
51       'user_id': userId,
52     });
53
54     titleController.clear();
55     notesController.clear();
56     setState(() {
57       kategoriTerpilih = null;
58       prioritasTerpilih = null;
59       tanggalTerpilih = null;
60       waktuTerpilih = null;
61     });
62
63     showTopSnackBar(context, '✓ Task Saved Successfully');
64   } catch (e) {
65     showTopSnackBar(context, '✗ Error: $e');
66   }
67 }
```

- ❖ Buat **controller** untuk menampung teks dari input (form)
- ❖ titleController → buat nyimpen judul task.
- ❖ notesController → buat nyimpen catatan/notes tambahan.
- ❖ User isi judul, notes, pilih kategori, prioritas, tanggal, dan waktu.
- ❖ Klik tombol "Save".
- ❖ Fungsi simpanData () dijalankan → data dikirim ke Supabase.
- ❖ Kalau sukses → form dikosongkan + muncul pesan sukses.
- ❖ Kalau gagal → muncul pesan error.

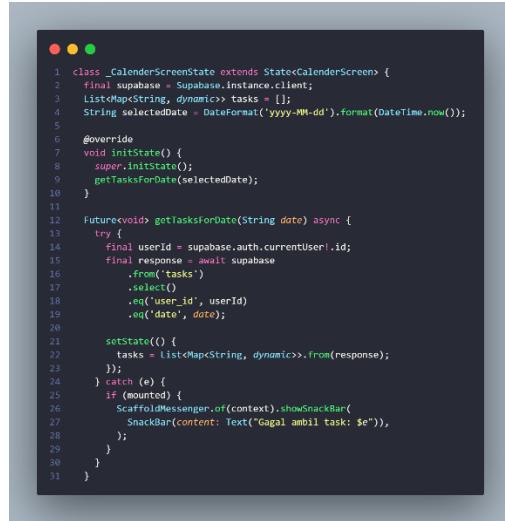
```
1 onPressed: () async {
2     await simpanData();
3 },
4 style: ElevatedButton.styleFrom(
5     foregroundColor: const Color(0xFF584A4A),
6     backgroundColor: const Color(0xFFA0D7C8),
7     padding: const EdgeInsets.symmetric(
8         horizontal: 50, vertical: 10),
9     shape: RoundedRectangleBorder(
10        borderRadius: BorderRadius.circular(50),
11    ),
12),
13 child: Text("Save",
14     style: GoogleFonts.poppins(
15         fontSize: 22,
16         fontWeight: FontWeight.w600,
17         color: const Color(0xFF584A4A))),
18 ),
19 ),
20 ],
21 ),
22 ),
23 ),
24 ],
25 ],
26 ),
27 );
28 );
```

- ❖ `onPressed` → fungsi yang dipanggil saat tombol ditekan.
- ❖ `async` → supaya bisa menjalankan perintah asinkron (misalnya tunggu database selesai).
- ❖ `await simpanData();` → memanggil fungsi `simpanData()` yang kamu buat sebelumnya untuk menyimpan data ke Supabase.

```
1 overlay.insert(overlayEntry);
2
3 Future.delayed(const Duration(seconds: 2), () {
4     overlayEntry.remove();
5 });
6 }
```

- ❖ `overlay` = lapisan paling atas layar (di atas semua widget lain).
- ❖ `overlayEntry` = widget khusus yang sudah kamu buat sebelumnya (misalnya pesan kecil).
- ❖ `Future.delayed` → jalankan sesuatu **setelah menunggu beberapa waktu**.
- ❖ `Duration (seconds: 2)` → tunggu **2 detik**.
- ❖ `overlayEntry.remove();` → setelah 2 detik, hapus `overlayEntry` dari layar.

7. Calender



```
1 class _CalenderScreenState extends State<CalenderScreen> {
2   final supabase = Supabase.instance.client;
3   List<Map<String, dynamic>> tasks = [];
4   String selectedDate = DateFormat('yyyy-MM-dd').format(DateTime.now());
5
6   @override
7   void initState() {
8     super.initState();
9     getTasksForDate(selectedDate);
10  }
11
12 Future<void> getTasksForDate(String date) async {
13   try {
14     final userId = supabase.auth.currentUser!.id;
15     final response = await supabase
16       .from('tasks')
17       .select()
18       .eq('user_id', userId)
19       .eq('date', date);
20
21     setState(() {
22       tasks = listMap<String, dynamic>.from(response);
23     });
24   } catch (e) {
25     if (mounted) {
26       ScaffoldMessenger.of(context).showSnackBar(
27         SnackBar(content: Text("Gagal ambil task: $e")),
28       );
29     }
30   }
31 }
```

- ❖ ser_id → biar hanya task milik user yang login.
- ❖ date → biar hanya task untuk tanggal tertentu (misalnya "2025-08-26").
- ❖ Saat halaman dibuka, langsung cek tanggal hari ini.
- ❖ Ambil semua task dari database Supabase sesuai tanggal itu dan sesuai user yang login.
- ❖ Simpan hasilnya ke tasks biar bisa ditampilkan di layar.
- ❖ Kalau error → munculin notifikasi error.



```
1 final days = List.generate(7, (i) {
2   final date = DateTime.now().add(Duration(days: i - 2));
3   return DateFormat('yyyy-MM-dd').format(date);
4 });
```

- ❖ **DateTime.now()**
Ambil tanggal & waktu **sekarang**.
- ❖ **DateFormat('yyy-MM-dd').format (date)**
Ubah format tanggal jadi string (teks) dengan bentuk "**tahun-bulan-hari**".
Contoh: "2025-08-26".

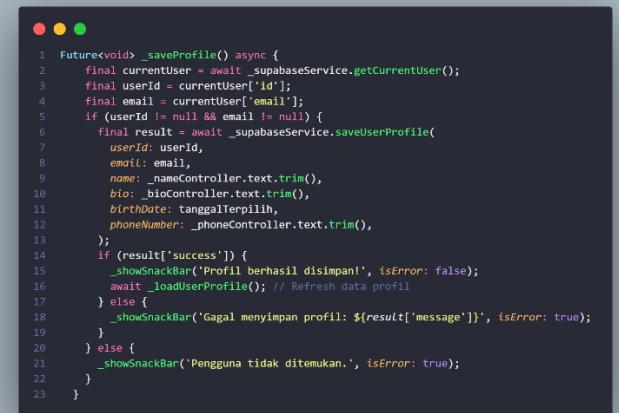
8. My Profil



- ❖ `_nameController` → menyimpan teks dari input **nama**.
- ❖ `_bioController` → menyimpan teks dari input **bio/cerita singkat**.
- ❖ `_phoneController` → menyimpan teks dari input **nomor HP**.
- ❖ koneksi ke database (`_supabaseService`),
- ❖ kontrol input (nama, bio, nomor HP),
- ❖ menyimpan tanggal yang dipilih,
- ❖ menyimpan data profil,
- ❖ dan cek apakah halaman sedang loading.



- ❖ Fungsi `_loadUserProfile()` itu kayak **mesin ambil data profil**:
- ❖ Simpan hasilnya ke `userProfile`.
- ❖ Masukkan data ke masing-masing controller:
- ❖ Nama → `_nameController.text`
- ❖ Bio → `_bioController.text`
- ❖ Nomor HP → `_phoneController.text`
- ❖ Simpan juga tanggal lahir ke `tanggalTerkilih`.
- ❖ Terakhir set `isLoading = false` supaya loading selesai.



```
1 Future<void> _saveProfile() async {
2     final currentUser = await _supabaseService.getCurrentUser();
3     final userId = currentUser['id'];
4     final email = currentUser['email'];
5     if (userId != null && email != null) {
6         final result = await _supabaseService.saveUserProfile(
7             userId: userId,
8             email: email,
9             name: _nameController.text.trim(),
10            bio: _bioController.text.trim(),
11            birthDate: tanggalTerpilih,
12            phoneNumber: _phoneController.text.trim(),
13        );
14        if (result['success']) {
15            _showSnackBar('Profil berhasil disimpan!', isError: false);
16            await _loadUserProfile(); // Refresh data profil
17        } else {
18            _showSnackBar('Gagal menyimpan profil: ${result['message']}', isError: true);
19        }
20    } else {
21        _showSnackBar('Pengguna tidak ditemukan.', isError: true);
22    }
23 }
```

- ❖ `_saveProfile()` dipakai untuk **menyimpan atau update profil pengguna** ke database Supabase.
- ❖ `userId` → id pengguna (penting untuk tahu profil siapa yang diubah)
- ❖ `email` → email user
- ❖ `name` → isi dari input nama (`_nameController`)
- ❖ `bio` → isi dari input bio (`_bioController`)
- ❖ `birthDate` → tanggal lahir (`tanggalTerpilih`)
- ❖ `phoneNumber` → isi dari input nomor HP (`_phoneController`)
- ❖ Tampilkan snackbar: "**Profil berhasil disimpan!**"



```
1 Future<void> _logout() async {
2     try {
3         await _supabaseService.logoutUser();
4         if (mounted) {
5             _showSnackBar('Logout berhasil.', isError: false);
6             Navigator.of(context).pushAndRemoveUntil(
7                 MaterialPageRoute(builder: (context) => SplashScreen()),
8                 (Route<dynamic> route) => false,
9             );
10        }
11    } catch (e) {
12        if (mounted) {
13            _showSnackBar('Kesalahan saat logout: $e', isError: true);
14        }
15    }
16 }
```

- ❖ Fungsi asynchronous, karena logout butuh komunikasi dengan server (Supabase).
- ❖ Logout dari Supabase (hapus sesi login).
- ❖ Kasih notifikasi kalau berhasil.
- ❖ Arahkan user ke SplashScreen (halaman awal).
- ❖ Kalau ada error → kasih notifikasi error.



```
1  @override
2  void dispose() {
3      _nameController.dispose();
4      _bioController.dispose();
5      _phoneController.dispose();
6      super.dispose();
7  }
```

- ❖ Semua controller (name, bio, phone) dibuang.
- ❖ Supaya memori tetap lega dan aplikasi lancar.