

# CMPS 312



# Supabase Cloud Services



**Database**



**Authentication**



**Storage**

**Dr. Abdelkarim Erradi**

**CSE@QU**

# Outline

1. Supabase Database
2. CRUD Operations
3. File Storage
4. Authentication
5. Access Image Gallery and Camera

# Backend-as-Service (BaaS)

- **Purpose:** Provide ready-made backend for web & mobile apps
- **Benefits:**
  - No need to build/manage servers, databases, or APIs
  - Speeds up development and reduces infrastructure complexity & cost
  - Allows developers focus on frontend and core business logic
- **Common Features:** User authentication, Managed databases, File storage, Serverless functions, Notifications & analytics
- **Examples:** Supabase Firebase AWS Amplify



# What is Supabase?

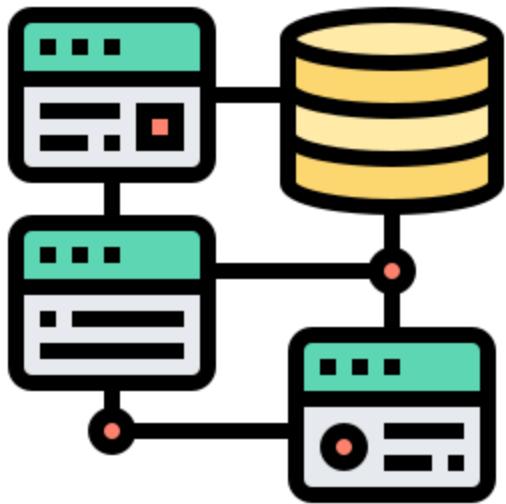
Supabase = Open-source Backend-as-a-Service (BaaS) built on PostgreSQL. Core Features:

- Database: Managed PostgreSQL with Row-Level Security (RLS)
- Authentication: Secure user sign-in via email/password & OAuth providers
- Storage: Scalable file storage with public/signed URLs
- Realtime: Listen to database changes in real time
- Edge Functions: Deploy serverless functions for custom logic
- Developer Tools: SDKs for **Flutter/Dart**, JavaScript, and more

# Getting Started

- Add `supabase_flutter` to `pubspec.yaml`
- Initialize Supabase in `main.dart`
  - This Enables database, authentication, and storage features in your Flutter app.

```
await Supabase.initialize(  
    url: 'https://your-project.supabase.co',  
    anonKey: 'your-anon-key',  
);
```



# Supabase Database

# Supabase Database

- Managed PostgreSQL: Includes SQL, views, triggers, policies, and functions
- Auto-Generated REST & GraphQL APIs for instant access
  - Use `.from('table')` to work with tables
- Schema First Design: Create tables via SQL or Supabase dashboard
- Row-Level Security (RLS) with customizable policies

```
// Fetching data
final response = await Supabase.instance.client
  .from('profiles')
  .select()
  .execute();
```



# Creating Supabase Database

- Design tables in SQL or the Supabase dashboard

-- Example table: todos

```
create table if not exists public.todos (
    id uuid primary key default gen_random_uuid(),
    description text not null,
    type text not null check (type in
        ('personal', 'work', 'family')),
    completed boolean not null default false,
    created_at timestampz not null default now(),
    user_id uuid references auth.users(id)
);
```

# Row Level Security (RLS)

- RLS: Ensures users can only access and modify their own data

-- *Enable RLS on the table*

```
alter table public.todos enable row level security;
```

-- *Policy: Read own rows*

```
create policy "read own" on public.todos  
for select using (auth.uid() = user_id);
```

-- *Policy: Modify own rows*

```
create policy "modify own" on public.todos  
for all using (auth.uid() = user_id);
```



# CRUD Operations



CREATE



READ



UPDATE



DELETE

---

C R U D

# CRUD

CRUD allows creating, reading, updating, and deleting database records

```
// Create
await Supabase.instance.client
  .from('tasks')
  .insert({ 'title': 'New Task' })
  .execute();

// Update
await Supabase.instance.client
  .from('tasks')
  .update({ 'done': true })
  .eq('id', 1)
  .execute();

// Delete
await Supabase.instance.client
  .from('tasks')
  .delete()
  .eq('id', 1)
  .execute();
```



# Database CRUD Operations

```
final client = Supabase.instance.client;

// CREATE
Future<void> addTodo(Todo todo) async {
  await client.from('todos').insert(todo.toJson());
}

// READ (list)
Future<List<Todo>> getTodos() async {
  final data = await client.from('todos').select().order('created_at', ascending: false);
  return (data as List).map((j) => Todo.fromJson(j)).toList();
}

// READ (single)
Future<Todo?> getTodoById(String id) async {
  final json = await client.from('todos').select().eq('id', id).maybeSingle();
  return json == null ? null : Todo.fromJson(json);
}

// UPDATE
Future<void> updateTodo(Todo todo) async {
  await client.from('todos').update(todo.toJson()).eq('id', todo.id);
}

// DELETE
Future<void> deleteTodo(String id) async {
  await client.from('todos').delete().eq('id', id);
}

// COUNT
Future<int> getTodosCount() async {
  final res = await client.from('todos').select().count(CountOption.exact);
  return res.count; // supabase_flutter 2.x
}
```

# Best Practices

- Implement data access in repositories
- Explore SupabaseClient as a provider
- Paginate with range() for large lists
- Use select() projections to limit payload
- Always handle errors (try/catch) and show user-friendly messages



# Real-Time Updates

- *Live Data Syncing:* Supabase Realtime streams table changes immediately to clients
  - Use cases: Live chat, dashboards, multiplayer games
  - Listen only to relevant tables/rows to optimize performance
  - Combine with state management for better user experience

```
final stream = Supabase.instance.client
    .from('messages')
    .stream(primaryKey: ['id']);

stream.listen((changes) {
    // Handle new/updated message objects
});
```



# Listen to Database Realtime Updates

Postgres Stream:

```
.from('table').stream(primaryKey: ['id'])
```

Stream for ToDo ist

```
Stream<List<Todo>> observeTodos() {  
    final client = Supabase.instance.client;  
    return client  
        .from('todos')  
        .stream(primaryKey: ['id'])  
        .order('created_at', ascending: false)  
        .map((rows) =>  
            rows.map(Todo.fromJson).toList());  
}
```

# File Storage





# File Storage

- Supabase Storage uploads, manages, and serves files securely
  - Upload/download user files or images
  - Use cases: Profile pictures, documents, content feeds
  - Create buckets via Supabase dashboard
  - Manage file access rules

```
Future<void> uploadFile(String filePath) async {  
    final file = File(filePath);  
    final response = await Supabase.instance.client.storage  
        .from('uploads')  
        .upload('filename.png', file);  
}
```



# File Storage

S3-like buckets with public/private access and signed URLs.

- Create buckets in the Supabase dashboard
- Store images, documents, or media; serve via public or signed URLs
- Secure with RLS-like bucket policies
- Best Practices
  - Use hashed/UUID file names; avoid collisions
  - Keep buckets private and use signed URLs where possible



# File Storage

```
final storage = Supabase.instance.client.storage;
// Upload bytes (e.g., from picker)
Future<String> uploadAvatar(Uint8List bytes, String userId) async {
  final path = 'avatars/$userId-${DateTime.now().millisecondsSinceEpoch}.png';
  await storage.from('avatars').uploadBinary(
    path,
    bytes,
    fileOptions: const FileOptions(contentType: 'image/png'),
  );
  // Return public URL (if bucket is public) or create a signed URL
  return storage.from('avatars').getPublicUrl(path);
}

// Signed URL (private bucket)
Future<Uri> getSignedUrl(String path,
  {Duration ttl = const Duration(minutes: 5)}) async {
  final url = await storage.from('avatars').createSignedUrl(path,
    ttl.inSeconds);
  return Uri.parse(url);
}
```

# List

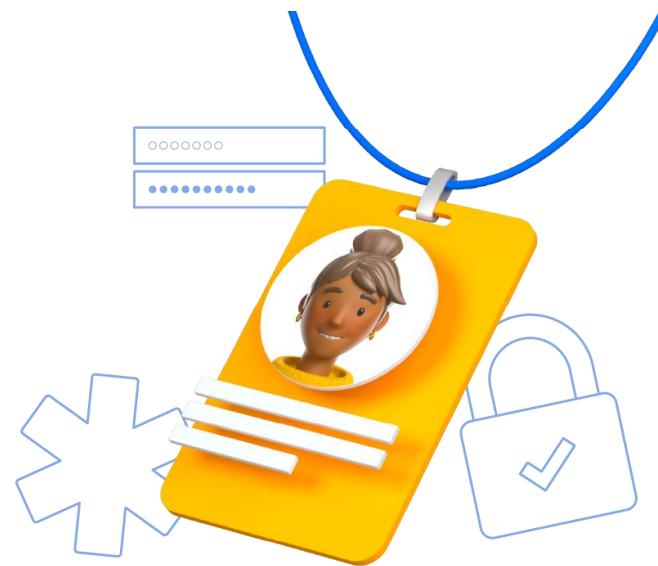
- Get URLs of files in particular subfolder

```
Future<List<String>> getImageUrls() async {
    final storage = Supabase.instance.client.storage;
    final files = await storage.from('images').list(path: '');
    return files.map((f) =>
        storage.from('images').getPublicUrl(f.name)).toList();
}
```

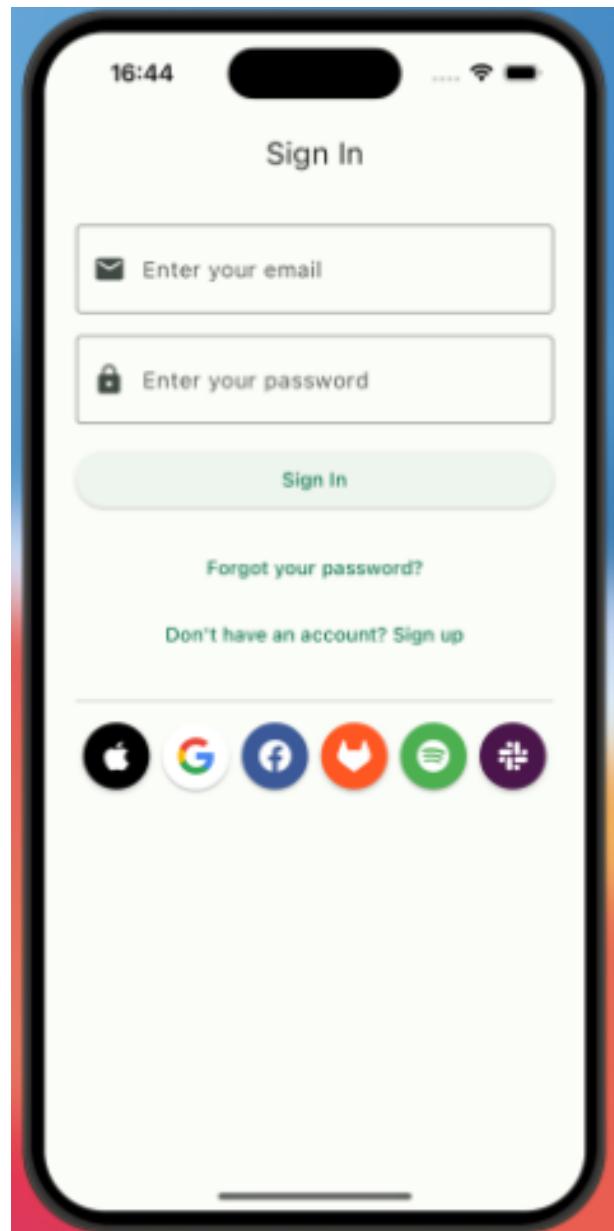
 If the bucket is **private**, use signed URLs instead:

```
Future<List<String>> getImageUrls() async {
    final storage = Supabase.instance.client.storage;
    final files = await storage.from('images').list(path: '');
    return Future.wait(files.map((f) =>
        storage.from('images').createSignedUrl(f.name, 3600)));
}
```

# Authentication



# Authentication

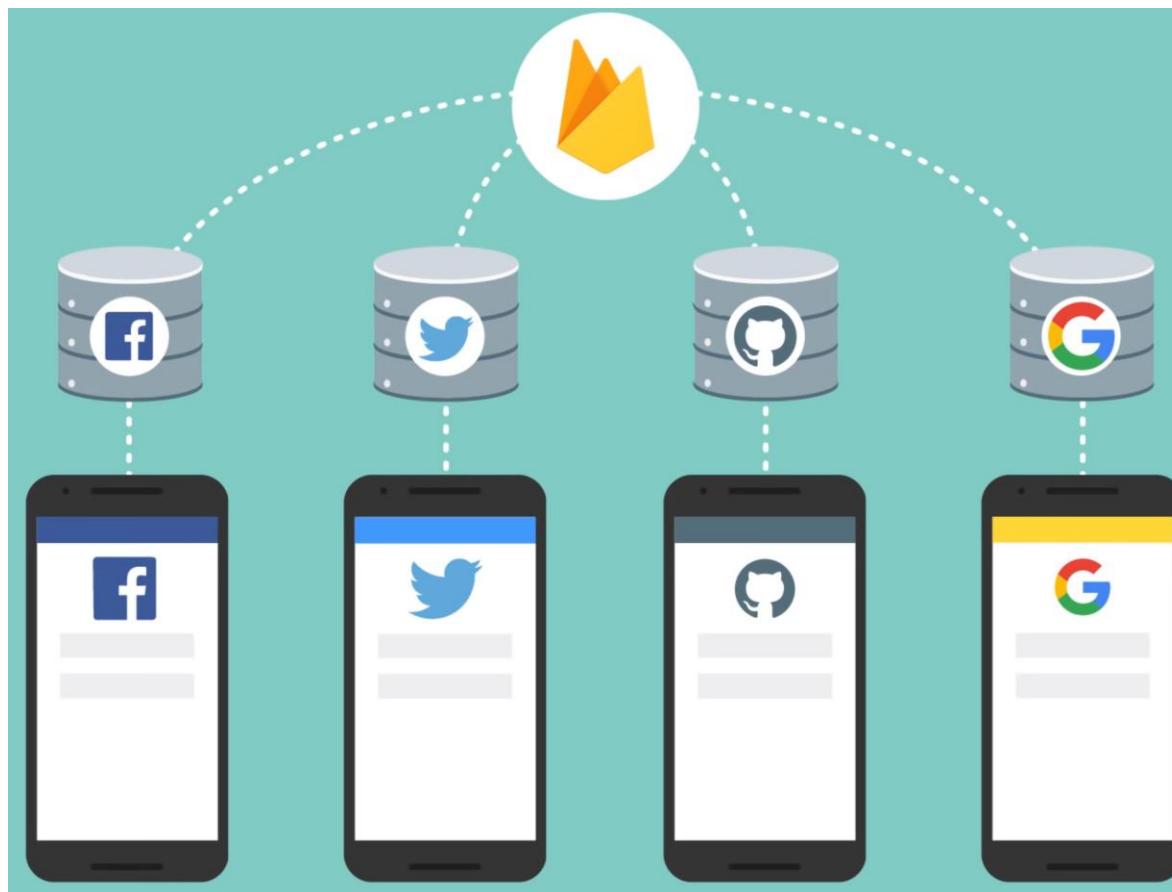


- **Authentication = Identity verification:**
  - Verify the identity of the user given the credentials received
  - Making sure the user is who he claims to be
- Every user gets a unique ID
- Restrict who can read and write what data



# Authentication

- Email/password, OTP/magic links, and OAuth providers (Google, Apple, etc.)
  - User sign-up, login, session management, password reset
  - Session-based auth with refresh tokens



Multiple Identity Providers can be used for Authentication



# Authentication

```
final auth = Supabase.instance.client.auth;

// Sign up
Future<void> signUp(String email, String password) async {
  await auth.signUp(email: email, password: password);
}

// Sign in
Future<void> signIn(String email, String password) async {
  await auth.signInWithEmailAndPassword(email: email, password:
password);
}

// Sign out
Future<void> signOut() async {
  await auth.signOut();
}
```

# Sign up

- Sign up and the user details to authentication

```
Future<User?> signUp(User user) async {
  try {
    final auth = Supabase.instance.client.auth;
    // Create a new user with email and password
    User authResult = await auth.signUp(
      email: user.email,
      password: user.password,
    );

    // If the user was created successfully, update their profile
    if (authResult.user != null) {
      await authResult.user!.updateDisplayName('${user.firstName} ${user.lastName}');
      await authResult.user!.updatePhotoURL('http://test.com/spongebob.png');
      // Refresh the user to apply changes
      await authResult.user!.reload();
    }

    return authResult;
  } catch (e) {
    print('Error during sign up: $e');
    return null;
  }
}
```

# Get current user details

- Anywhere in the app you can access the details of current user

```
void getCurrentUser() {  
    final auth = Supabase.instance.client.auth;  
    User? user = auth.currentUser;  
    if (user != null) {  
        print('User is signed in! Email: ${user.email}');  
    } else {  
        print('No user is signed in.');  
    }  
}
```

# Listen to auth state

- Listen to auth state changes

```
// Listen to auth state
void listenAuth() {
  auth.onAuthStateChanged.listen((event) {
    final session = event.session;
    // navigate or rebuild providers
  });
}
```

# Route Auth Guard

- Auth Guard (GoRouter + Riverpod)
- Use guards tied to auth state

```
final authStateProvider = StreamProvider((ref) {  
  return Supabase.instance.client.auth.onAuthStateChanged  
    .map((e) => e.session);  
});  
  
final authGuard = GoRoute(  
  path: '/account',  
  builder: (context, state) => const AccountScreen(),  
  redirect: (context, state) {  
    final session = context.read(authStateProvider).maybeWhen(  
      data: (s) => s,  
      orElse: () => null,  
    );  
    return session == null ? '/signin' : null;  
  },  
);
```



# Architecture & Patterns

- Use Riverpod providers to expose repositories

```
final supabaseClientProvider = Provider((ref) =>  
  Supabase.instance.client);
```

```
final todoRepositoryProvider = Provider((ref) {  
  final client = ref.watch(supabaseClientProvider);  
  return TodoRepository(client);  
});
```



# Best Practices & Recommendations

## 1) Security

- Enable RLS on every table; write least-privilege policies
- Use signed URLs for private assets; short TTLs for sensitive files

## 2) Performance

- Use projections and pagination; avoid select('\*') in production
- Prefer streams for live lists; batch UI updates

## 3) Keep migration SQL in version control

## 4) Use Edge Functions for server-side logic

# Access Image Gallery and Camera



# Access Image Gallery and Camera

- Using `image_picker` package for picking images from the image gallery or taking new pictures with the camera

```
Future<File?> pickImage(ImageSource source) async {
    final imagePicker = ImagePicker();
    final pickedImage = await imagePicker.pickImage(
        source: source, // camera or gallery
        maxWidth: double.infinity,
    );
    if (pickedImage == null) return null;
    return File(pickedImage.path);
}
```

# image\_picker methods

```
final ImagePicker picker = ImagePicker();
// Pick an image
final XFile? image = await picker.pickImage(source: ImageSource.gallery);
// Capture a photo
final XFile? photo = await picker.pickImage(source: ImageSource.camera);
// Pick a video
final XFile? galleryVideo =
    await picker.pickVideo(source: ImageSource.gallery);
// Capture a video
final XFile? cameraVideo = await picker.pickVideo(source: ImageSource.camera);
// Pick multiple images
final List<XFile> images = await picker.pickMultiImage();
// Pick singe image or video
final XFile? media = await picker.pickMedia();
// Pick multiple images and videos
final List<XFile> medias = await picker.pickMultipleMedia();
```

# Summary

- **Supabase** database store/query app's data
  - Data model consists tables having relationships
- File **Storage** is used to store and retrieve files
- **Authentication** provides built-in backend services to ease user authentication
  - email/password authentication allows users to register and log in to the app
  - Secure user's authentication using various identity providers (e.g., email/password, Google Auth)



# References

- Supabase Flutter Docs:

<https://supabase.com/docs/guides/getting-started/quickstarts/flutter>

- Realtime:

<https://supabase.com/docs/guides/realtime>

- Storage:

<https://supabase.com/docs/guides/storage>

- Auth: <https://supabase.com/docs/guides/auth>

- RLS Policies:

<https://supabase.com/docs/guides/auth/row-level-security>