CMPS 312

Supabase Storage & Authentication

Dr. Abdelkarim Erradi

CSE@QU

Outline

1. Storage
2. Authentication
3. Access Image Gallery and Camera

Storage

📁 Storage

What It Does:

- Upload, manage, and serve files securely

Key Features:

- Has a simple bucket/folder/file structure
- Upload/download user files or images
- Create storage buckets via Supabase dashboard
- Define file access rules (public, private, signed URLs)

Common Use Cases:

- Store Profile pictures, Documents, Images

Best Practices

- Use UUID file names to avoid collisions
- Keep buckets private and give users temporary access with signed URLs

File Upload

```dart
final storage = Supabase.instance.client.storage;
/// Upload an avatar using a file path
Future<String> uploadAvatarFromPath(String filePath, String userId) async {
  final file = File(filePath);
  final fileName =
      'avatars/$userId-${DateTime.now().millisecondsSinceEpoch}.png';
  await storage.from('avatars').upload(fileName, file,
        fileOptions: const FileOptions(contentType: 'image/png'),
      );
  // If bucket is public → returns public URL
  return storage.from('avatars').getPublicUrl(fileName);
}
```

```dart
// Signed URL for private buckets – by default valid for 5 minutes
Future<Uri> getSignedUrl(String path,
    {Duration ttl = const Duration(minutes: 5)}) async {
  final signedUrl = await storage
      .from('avatars')
      .createSignedUrl(path, ttl.inSeconds);
  return Uri.parse(signedUrl);
}
```

Show image in Flutter app

```dart
Image.network(signedUrl);
```

List files in a bucket

- 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();
}
```

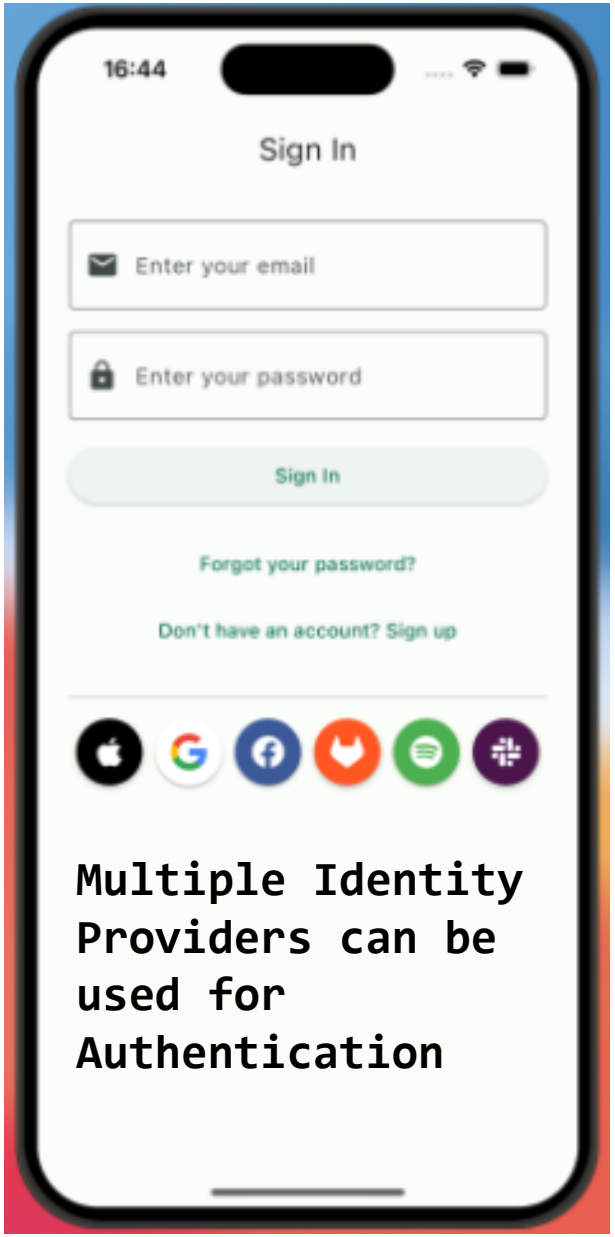
🔒 If the bucket is private, use **signed URLs** — temporary, secure links that automatically expire after a set duration to prevent misuse

```
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, 300))); //valid for 5 mins
}
```

Authentication

Back

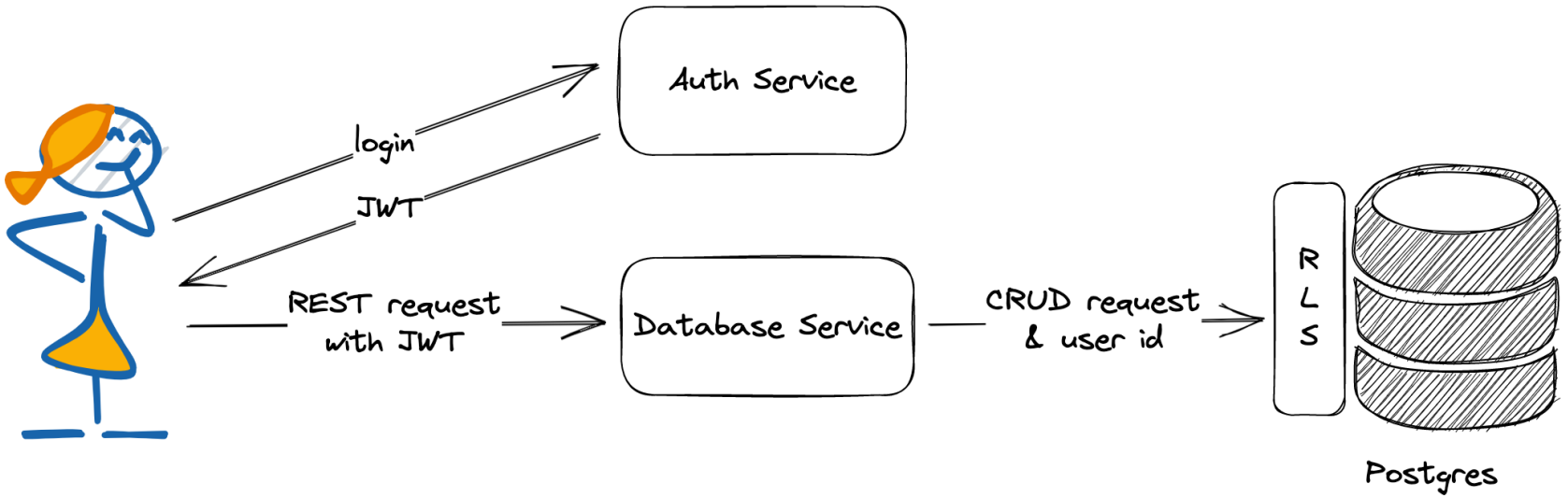
Authentication



- **Definition:** Verifying a user's identity before granting access
- **Key Points:**
  - Confirms the user is who they claim to be
  - Uses credentials (e.g., email/password, Auth providers such as Google)
  - Assigns each user a **unique ID** for tracking and security
  - Enforces **access control**: restricts who can read or write data

Authentication and Authorization flow

- **Login**: Client sends credentials → Auth Service returns a **JWT (JSON Web Token)**
- **Requests**: Client includes JWT in **REST API calls** to Database Service
- **Database**: Service validates JWT, extracts **user ID**, performs **CRUD** on PostgreSQL
- **RLS**: PostgreSQL applies **Row-Level Security** using user ID to restrict access to authorized rows



🔐 Authentication

```dart
final auth = Supabase.instance.client.auth;
// Sign up
Future<void> signUp(String email, String password)
async {
  await auth.signUp(email, password);
}

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

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

Sign Up and Store User Metadata

```
Future<User?> signUp(User user) async {
  try {

    final auth = supabase.auth;
    // ------------------- Sign up --------------------
    final response = await auth.signUp(
      email: user.email,
      password: user.password,
      data: {
        // Optional metadata stored inside auth.users
        'firstName': user.firstName,
        'lastName': user.lastName,
        'avatar_url': user.avatarUrl
      },

     return response.user;
  } 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() {
  User? user = supabase.auth.currentUser;
  if (user != null) {
    print('User is signed in! Id: ${user.id}');
    print('User is signed in! Email: ${user.email}');
    print('Metadata: ${user.userMetadata}');
  } else {
    print('No user is signed in.');
  }
}
```

Listen to auth state

- Real-time Updates: If you need to react to authentication state changes (e.g., a user logs in or out), you should listen to the **onAuthStateChange** stream provided by Supabase Auth

```
final authStateProvider = StreamProvider((ref) {
  return Supabase.instance.client.auth.onAuthStateChange
    .map((e) => e.session);
});
```

Route Auth Guard

- **Purpose**: Use Auth Guards to protect routes based on authentication state
  - Prevent unauthorized access to sensitive screens
  - Redirect unauthenticated users to sign-in
  - Simplify navigation logic using state-based routing

```
// Guard: Redirect if not signed in
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;
  },
);
```

GitHub Authentication

- **Purpose:** Enable users to sign in using their GitHub account
- **Create GitHub OAuth App**
  - GitHub → Settings → Developer settings → OAuth Apps → *New App*
  - Ste Homepage URL: e.g., `myapp://auth/callback`
  - Callback URL: `https://<project>.supabase.co/auth/v1/callback`
  - Copy **Client ID** and **Client Secret**
- **Enable GitHub in Supabase**
  - Dashboard → **Auth → Providers → GitHub**
  - Paste credentials and enable provider
- **Trigger Login in Your App**

```
Supabase.instance.client.auth.signInWithOAuth(
  Provider.github,
  redirectTo: 'myapp://auth/callback' );
```

Access Image Gallery and Camera

Back

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

```dart
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

- **Storage**: Securely upload, store, and retrieve files
- **Authentication**: Built-in backend services for user sign-up and login
  - Supports **email/password** and **Auth providers** (e.g., Google)
- **Security**: Protect user data with authentication and authorization policies

References

- **Storage:** https://supabase.com/docs/guides/storage
- **Realtime:** https://supabase.com/docs/guides/realtime
- **Auth:** https://supabase.com/docs/guides/auth