

Class has
single
responsibility



Class has
only one
reason to
change

Single
Responsibility
Principle

الفكرة الأساسية



كل كلاس أو وظيفة في برمجة Flutter يجب أن:

- يقوم بوظيفة واحدة فقط
- يكون له سبب واحد للتغيير

مثل شخص في مطعم له وظيفة واحدة محددة!

تشبيه عملي



تخيل تطبيق مطعم:

- الطباخ: يصنع الطعام فقط
- النادل: يقدم الطلبات فقط
- المحاسب: يحصل المال فقط

إذا خلطنا المهام ستصبح الفوضى!

مثال من Flutter



(SRP خطأ ينتهي)

```
1. class UserManager {  
2.   إدارة بيانات المستخدم //  
3.   User? user;  
4.  
5.   حفظ في قاعدة البيانات //  
6.   void saveToDatabase() {  
7.     // ...  
8.   }  
9.  
10.  إرسال إشعار //  
11.  void sendNotification(String message) {  
12.    // ...  
13.  }  
14. }
```

المشكلة: هذا الكلاس يقوم بـ 3 مهام مختلفة!

مثال صحيح (يطبق SRP)

```
1. // 1. كلاس لإدارة بيانات المستخدم فقط
2. class User {
3.     final String name;
4.     final String email;
5.
6.     User(this.name, this.email);
7. }
8.
9. // 2. كلاس للتعامل مع قاعدة البيانات فقط
10. class UserRepository {
11.     void saveUser(User user) {
12.         // ...
13.     }
14. }
15.
16. // 3. كلاس لإرسال الإشعارات فقط
17. class NotificationService {
18.     void sendNotification(User user, String message) {
19.         // ...
20.     }
21. }
```

الحل: تقسيم المهام إلى 3 كلاسات مستقلة!

لماذا SRP مهم في Flutter

- سهولة التعديل: لو تغيرت قاعدة البيانات، نعدل فقط `UserRepository`
- أقل أخطاء: كل جزء متخصص في شيء واحد
- إعادة استخدام الكود: يمكن استخدام `NotificationService` في أي مكان

كيف تعرف أنك تنتهك SRP?

أسأل نفسك:

- هل يمكن تقسيم هذا الكلاس لأجزاء أصغر؟
- هل له أكثر من سبب للتغيير؟
- هل الوظائف غير مرتبطة ببعضها؟

إذا كانت الإجابة "نعم"، فأنت تحتاج لإعادة هيكلة الكود!

مثال عملي في Flutter Widget

SRP ينتهك Widget 

```
1. class UserProfile extends StatelessWidget {  
2.   // يحفظها + يعرضها + يحمل البيانات  
3.   @override  
4.   Widget build(BuildContext context) {  
5.     // ...  
6.   }  
7. }
```

تطبيق SRP 

```
1. // 1. Widget للعرض فقط  
2. class UserProfileUI extends StatelessWidget {  
3.   final User user;  
4.  
5.   UserProfileUI(this.user);  
6.  
7.   @override  
8.   Widget build(BuildContext context) {  
9.     // ...  
10.  }  
11. }  
12. // 2. كلاس منفصل للبيانات  
13. class UserProfileData {  
14.   Future<User> fetchUser() {  
15.     // ...  
16.   }  
17. }
```

نصائح للمبتدئين

1. ابدأ بكتابة كلاس لوظيفة واحدة
2. إذا وجدت كلمة "و" في وصف الكلاس (مثل "يحفظ البيانات ويرسل إشعارات")، فهذا **تحذير!**
3. **تذكر:** كلما صغرت المسؤولية، كبرت الفائدة!

الخلاصة

SRP يعني: "كل شيء له وظيفة واحدة، وكل وظيفة لها شيء واحد"

هذا يجعل تطبيقك:

- أسهل في الفهم
- أسهل في الصيانة
- أقل عرضة للأخطاء