



Gestion des exceptions et des erreurs en Dart

Ndongo Tonux SAMB



Agenda

- 1 Dart/Flutter c'est quoi???
- 2 Les Exception en Dart
- 3 Gestion des exceptions avec try/catch
- 4 Création d'exceptions personnalisées
- 5 Gestion des erreurs dans Flutter
- 6 Best practices



Flutter



Dakar

About me



<http://tonuxcorp.dev>



```
return Scaffold(  
  body: Person(  
    child: Column(  
      children: [  
        Welcome(),  
        Informations(  
          name: 'Ndongo SAMB',  
          alias: 'Tonux',  
          title: 'IT Consultant',  
        ),  
        From(  
          country: 'Senegal', ,  
          capital: 'Dakar'),  
        Location(  
          country: 'Canada', ,  
          province: 'Quebec',  
          city: 'Montreal'  
        )  
        Work(  
          label: 'World Anti Doping Agency'  
        )  
      ]  
    )  
  );
```

Dart?

- Langage de programmation open-source développé par Google
- Conçu pour le développement d'applications: web , mobile et desktop

1

Orienté Object

Utilise des objets et des classes.

2

Compilé

Peut être compilé en code natif et JavaScript.

3

Asynchrone

Gestion facile des opérations non bloquantes.

4

Fortement typé

Système de types statiques et dynamiques.

5

Productivité des développeurs

Syntaxe simple, null safety, extensions, mixins.

* Dart est le langage principal de **Flutter**, permettant de créer des interfaces utilisateur modernes et réactives pour plusieurs plateformes avec une base de code unique.



Flutter

Dakar

Flutter?

- Framework open-source développé par Google.
- Développement multiplateforme : Android, iOS, web, et bureau.
- Utilise le langage de programmation Dart.

1

Lancé par Google
en mai 2017

2

Compilé

Peut être compilé en
code natif et
JavaScript.

3

Widgets

Éléments de base de
l'interface utilisateur.

4

Flutter Engine

Gère le rendu et les
animations

5

Dart Virtual
Machine

Permet l'exécution du
code Dart.

- Développement multiplateforme avec une seule base de code.
- Performances natives grâce à la compilation en code natif.
- Hot Reload : Modifications instantanées sans redémarrage.
- Bibliothèque de widgets riche pour des interfaces utilisateur attractives.



Flutter

Dakar

C'est quoi une exception ou Erreur?

Exception :

- Un événement anormal qui interrompt le flux normal d'un programme.
- Géré par des mécanismes de traitement des exceptions.
- Exemples : Division par zéro, accès à un index hors limites d'un tableau.

Erreur :

- Un problème grave souvent hors du contrôle du programmeur.
- Souvent lié à l'environnement d'exécution.
- Exemples : Manque de mémoire, erreurs de pile (stack overflow).

Pourquoi les Gérer ?

- **Robustesse** : Augmente la stabilité et la fiabilité des applications.
- **Expérience Utilisateur** : Prévenir les plantages et fournir des messages d'erreur clairs.
- **Maintenance** : Facilite le débogage et la maintenance du code.



Exceptions et Erreurs en Dart

Les exceptions et les erreurs sont deux types de problèmes qui peuvent survenir lorsque vous exécutez votre code Dart.

Les exceptions sont généralement causées par des opérations non valides, telles que la division par zéro, l'accès à une valeur NULL ou l'appel d'une méthode sur un objet incorrect.

Les erreurs sont plus graves et indiquent une défaillance de la logique du programme, telles que des erreurs de syntaxe, des erreurs de type ou des erreurs d'assertion.

* Les exceptions et les erreurs peuvent être levées et interceptées à l'aide des mots-clés `throw` et `catch` dans Dart.



Types d'exceptions dans Dart

Dart dispose de plusieurs classes d'exceptions intégrées que vous pouvez utiliser. Certaines des exceptions les plus courantes incluent :

- **FormatException** : levée lorsqu'une opération de conversion de chaîne échoue.
- **ArgumentError** : levé lorsqu'un argument de fonction n'est pas valide.
- **RangeError** : levée lorsqu'une valeur est en dehors de sa plage valide.
- **NoSuchMethodError** : levée lorsqu'une méthode inexistante est appelée.
- **TypeError** : levée lorsqu'une opération rencontre une valeur d'un type incompatible.



Comment lever des exceptions et des erreurs ?

Vous pouvez lever une exception ou une erreur à l'aide du mot-clé `throw`, suivi d'une expression qui renvoie à un objet qui implémente l'interface `Throwable`.

Cette interface est implémentée par les classes intégrées `Exception` et `Error`, ainsi que par leurs sous-classes.

Exemple: vous pouvez lever une exception `FormatException` si vous rencontrez un format de chaîne non valide, ou une `RangeError` si vous accédez à un index hors limites.

Vous pouvez également créer vos propres exceptions et erreurs personnalisées en étendant la classe `Exception` ou `Error` et en remplaçant la méthode `toString`.



Comment détecter les exceptions et les erreurs ? 1 / 2

Vous pouvez intercepter une exception ou une erreur à l'aide du mot-clé `catch`, suivi d'un bloc de code qui gère le problème.

Vous pouvez éventuellement spécifier le type d'exception ou d'erreur que vous souhaitez intercepter, ainsi qu'un nom de variable pour accéder à l'objet levé. Par exemple, vous pouvez intercepter une exception `FormatException` et afficher son message à l'aide de la syntaxe suivante :

```
try {  
  // some code that might throw a FormatException  
} catch (e, s) { // e is the exception object, s is the stack trace  
  if (e is FormatException) {  
    print(e.message); // print the message of the FormatException  
  }  
}
```



Flutter

Dakar

Comment détecter les exceptions et les erreurs ? 2/2

Vous pouvez également utiliser le mot-clé `on` pour spécifier le type d'exception ou d'erreur que vous souhaitez intercepter, sans utiliser de nom de variable. Par exemple, vous pouvez intercepter une erreur `RangeError` et afficher son message à l'aide de la syntaxe suivante :

```
try {  
  // some code that might throw a RangeError  
} on RangeError catch (e) { // e is the exception object  
  print(e.message); // print the message of the RangeError  
}
```



Flutter

Dakar

Comment utiliser les blocs finally ?

Vous pouvez utiliser le mot-clé `finally` pour spécifier un bloc de code qui s'exécute toujours après les blocs `try` et `catch`, qu'une exception ou une erreur ait été levée ou non.

Ceci est utile pour nettoyer les ressources, fermer des fichiers ou effectuer toute autre action finale qui doit être effectuée. Par exemple, vous pouvez utiliser un bloc `finally` pour fermer un fichier après l'avoir lu, même si une exception ou une erreur s'est produite pendant le processus de lecture :

```
File file = File('some_file.txt');  
try {  
    // some code that reads from the file  
} catch (e) {  
    // some code that handles the exception or error  
} finally {  
    file.close(); // close the file in any case  
}
```



Flutter

Dakar



Pourquoi la gestion des erreurs est-elle essentielle ? 1 / 3

1

Amélioration de
l'expérience utilisateur

2

Empêcher les
applications de se
bloquer

3

Maintenir la stabilité de
l'application

4

Récupération
gracieuse des
défaillances

5

Journalisation et
rapports d'erreurs
(**Crashlytics**)



Flutter

Dakar

Pourquoi la gestion des erreurs est-elle essentielle ? 2/3

6

Sécurité des nullités et
stabilité (Null Safety)

7

Sécurité et conformité

8

Rendre facile les
changements

9

Confiance de l'équipe
sur le code

10

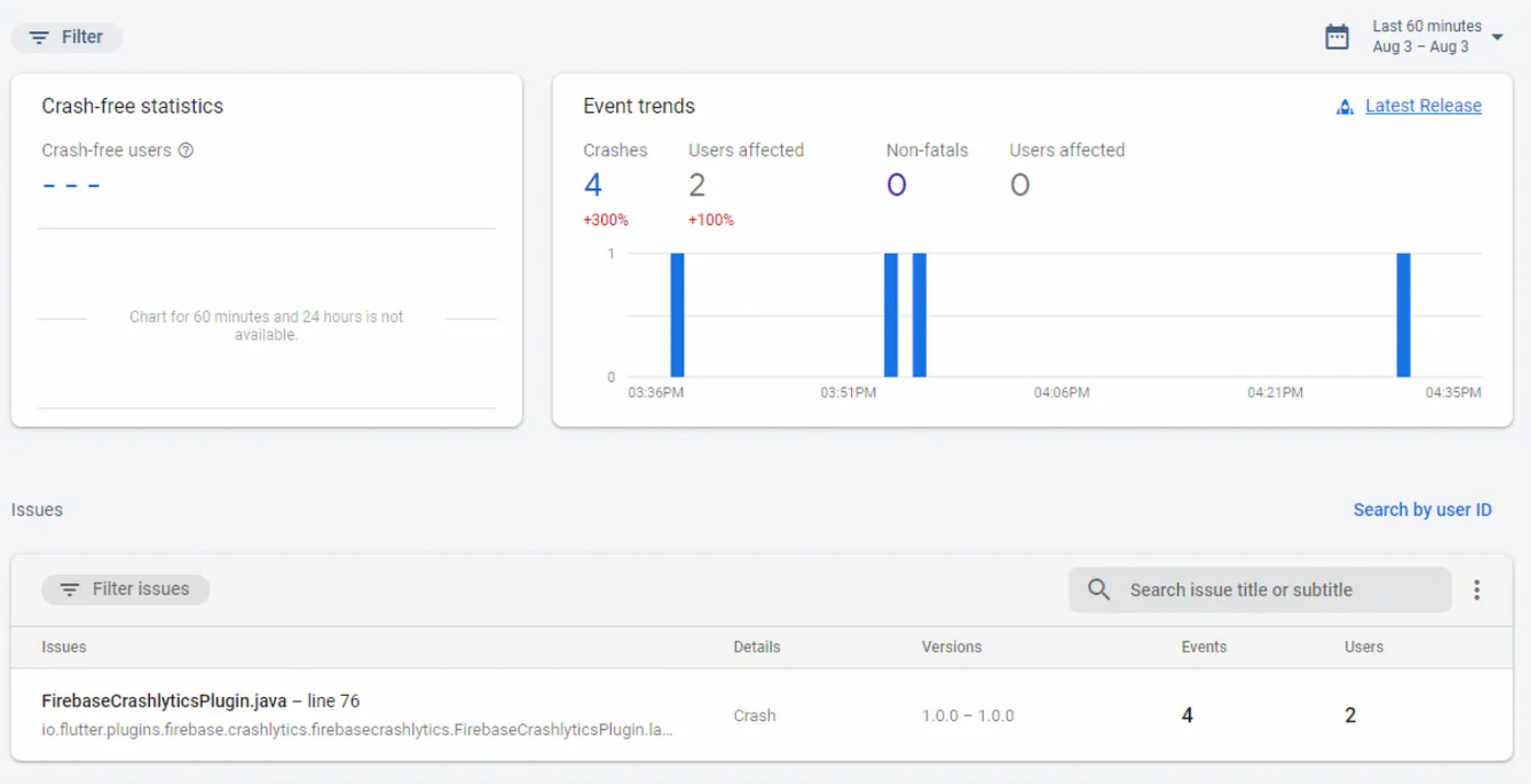
Facile pour faire les
évolution



Flutter

Dakar

Pourquoi la gestion des erreurs est-elle essentielle ? 3/3



Comment utiliser les blocs finally ?

Vous pouvez utiliser le mot-clé `finally` pour spécifier un bloc de code qui s'exécute toujours après les blocs `try` et `catch`, qu'une exception ou une erreur ait été levée ou non.

Ceci est utile pour nettoyer les ressources, fermer des fichiers ou effectuer toute autre action finale qui doit être effectuée. Par exemple, vous pouvez utiliser un bloc `finally` pour fermer un fichier après l'avoir lu, même si une exception ou une erreur s'est produite pendant le processus de lecture :

```
File file = File('some_file.txt');
try {
    // some code that reads from the file
} catch (e) {
    // some code that handles the exception or error
} finally {
    file.close(); // close the file in any case
}
```

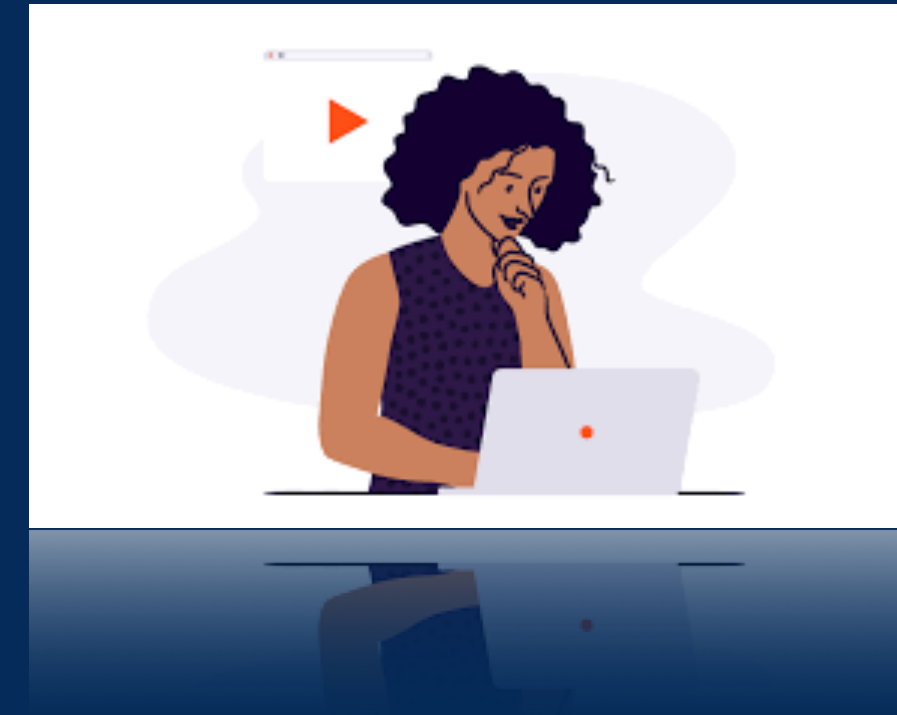


Flutter

Dakar



Demo



THANK YOU!

<http://tonuxcorp.dev>

