TP5 - Mutation testing

Objectifs

- Comprendre les limites de la couverture de code comme indicateur de qualite
- Installer et configurer Stryker.NET sur un projet existant
- Lire et interpreter un rapport de mutation testing HTML
- Identifier les tests faibles reveles par les mutants survivants
- Ecrire des tests cibles pour ameliorer le mutation score
- Comparer le mutation score des doublures manuelles vs les mocks NSubstitute

Prerequis

- TP2 complete (StringCalculator ou PasswordValidator avec sa suite de tests)
- TP3 complete (NotificationService avec doublures manuelles et NSubstitute)
- .NET 8 SDK installe

Mise en place

Installez Stryker.NET globalement :

```
dotnet tool install -g dotnet-stryker
```

Verifiez l'installation :

```
dotnet stryker --version
```

Placez-vous dans le dossier du projet de tests TP2 et lancez une premiere analyse :

```
dotnet stryker
```

Le rapport HTML est genere dans `StrykerOutput/reports/mutation-report.html`. Ouvrez-le dans votre navigateur.

Exercice 1 — Premier run et lecture du rapport (20 min)

Objectif

Lancer Stryker sur votre projet TP2 et prendre en main le rapport HTML.

Contraintes

- Travailler sur le projet TP2 tel qu'il a ete rendu (ne pas modifier les tests avant cette etape)
- Lire le rapport HTML — pas la sortie console

Verification

- [ ] Stryker s'execute sans erreur
- [ ] Vous pouvez afficher le rapport HTML dans le navigateur
- [ ] Vous connaissez le mutation score initial du projet
- [ ] Vous avez identifie les 3 fichiers avec le score le plus bas

Questions

1. Quel est le mutation score global de votre projet TP2 ?
2. Quelle classe a le plus de mutants survivants ?
3. Quelle est la difference entre un mutant "Survived" et un mutant "NoCoverage" ?

Indices

▶ Indice 1 — Stryker ne trouve pas le projet cible

▶ Indice 2 — Le rapport s'ouvre sur un ecran blanc

Exercice 2 — Autopsie des survivants (25 min)

Objectif

Identifier les 3 mutants survivants les plus significatifs et comprendre pourquoi ils survivent.

Contraintes

- Choisir des survivants dans des categories differentes (pas trois fois le meme type)
- Pour chaque survivant : copier le code mutant affiche dans le rapport, et expliquer en une phrase pourquoi le test passe malgre la modification

Livrable

Remplissez ce tableau (a remettre avec le TP) :

#	Fichier	Ligne	Mutant	Pourquoi il survit
1				
2				
3				

Verification

- ☐ 3 survivants identifies dans des categories differentes
- ☐ Explication claire pour chacun
- ☐ Vous pouvez dire quel test aurait du tuer ce mutant

Indices

▶ Indice 1 — Comment lire un mutant dans le rapport

▶ Indice 2 — Types de mutants frequents sur PasswordValidator

Exercice 3 — Tuer les survivants (30 min)

Objectif

Ecrire les tests manquants pour tuer les 3 survivants identifies a l'exercice 2. Verifier que le mutation score progresse.

Contraintes

- Un test par survivant minimum
- Chaque test doit avoir une assertion precise (pas de `Assert.NotNull` ou `Assert.True(true)`)
- Le nom du test doit documenter le cas couvert

Exemple de test cible

Si le survivant est `>= 8` mute en `> 8` dans `PasswordValidator` :

```
[Fact]
public void Validate_ExactlyEightCharacters_IsValid()
{
    // La borne exacte : 8 caracteres doivent etre valides
    var result = _validator.Validate("Abcdefg1"); // exactement 8 chars
    Assert.True(result);
}
```

Verification

- [ ] 3 tests ajoutes (ou plus)
- [ ] Chaque test a une assertion precise sur la valeur attendue
- [ ] Stryker relance — le mutation score a progresse
- [ ] Les 3 mutants cibles sont maintenant "Killed"

Indices

▶ Indice 1 — Le score n'a pas bouge apres ajout des tests
▶ Indice 2 — Tuer un mutant de type "operateur logique" (&&→||)

Exercice 4 — Doublures manuelles vs mocks NSubstitute (25 min)

Objectif

Comparer le mutation score du code teste avec des doublures manuelles vs des mocks NSubstitute avec `Arg.Any<>()` sur le projet TP3.

Contexte

En seance 3, vous avez ecrit deux versions des tests de `NotificationService` :

- Une version avec des **doublures manuelles** (`FakeEmailSender` , `SpyEmailSender`)
- Une version avec **NSubstitute** et `Arg.Any<string>()`

Instructions

1. Placez-vous dans le dossier TP3 et lancez Stryker
2. Notez le mutation score global
3. Dans le rapport, comparez le score sur `NotificationService.cs` entre les deux groupes de tests
4. Identifiez un mutant qui survive grace a `Arg.Any<>()` mais serait tue par une doublure manuelle

Verification

- [ ] Stryker lance sur le projet TP3
- [ ] Mutation score note pour les deux approches
- [ ] Au moins un exemple de survivant lie a `Arg.Any<>()` identifie
- [ ] Vous pouvez expliquer pourquoi la doublure manuelle tue ce mutant

Indices

▶ Indice 1 — Pourquoi Arg.Any<> laisse survivre des mutants
▶ Indice 2 — Configurer Stryker pour analyser TP3

Reflexion finale (10 min)

Questions

Repondez en quelques lignes a chacune des questions suivantes (a remettre avec le TP) :

1. A partir de quel mutation score votre projet TP2 vous semble-t-il suffisamment couvert ?

Justifiez votre reponse en tenant compte du type de projet (application critique ou utilitaire simple).

2. Peut-on raisonnablement viser 100% de mutation score ?

Expliquez le concept de mutant equivalent et son impact sur l'objectif.

3. Comment integreriez-vous Stryker dans votre workflow quotidien ?

Proposez une strategie (frequence d'execution, seuil CI, que faire des survivants).

Bareme indicatif

Critere	Points
Rapport Stryker produit et score initial documente	3 pts
3 survivants identifies avec explication	3 pts
Tests ajoutes cibles et pertinents	4 pts
Analyse comparative doublures manuelles vs NSubstitute	3 pts
Reflexion finale argumentee	2 pts

Total	**15 pts**