# Stratégie de tests

Nous avons établis une stratégie de tests à établir sur chaque Use Case de notre conception.

Pour chaque Use Case, on met en œuvre quatre tests différents, les tests unitaires, les tests d’intégration, les tests fonctionnels et les tests de validation.

Les deux premiers tests sont des tests à effectuer tout au long du développement pour chaque méthode utilisée dans le Use Case il faut créer un test unitaire, ensuite il faut lancer les tests d’intégration pour vérifier que cette nouvelle modification n’a pas altérer le résultat des autres méthodes.

Les deux autres tests sont des tests à effectuer à la fin, les tests fonctionnels permettent de vérifier que ce qu’on a développé fonctionne bien, fait ce qu’il est censé faire. Les tests de validation permettent quant à eux de vérifier que l’application finale correspond bien à la demande établie dans le Use Case au cours de la conception.

Figure Plan de tests pour chaque Use Case

# Tests unitaires

Les tests unitaires permettent de vérifier pour chaque méthode si elle se comporte comme elle doit. On vérifie donc les résultats renvoyés par chaque méthode pour garantir le bon fonctionnement de celle-ci.

Pour cela, on part du principe que nos données sont fixes, ce sont les données de test. Ensuite on vérifie pour des paramètres donnés, si toutes les méthodes d’un objet renvoient bien les données que l’on souhaite.

Pour réaliser ses tests, nous avons utilisé la librairie JUnit. Celle-ci, permet de créer des « TestCase » qui sont des classes qui permettent de tester d’autres classes. Ainsi, pour chaque classe on crée un TestCase qui teste toutes les fonctionnalités de la classe.

Nous avons effectué ces tests sur tous les Uses Cases que nous avons développés qui sont le login, la demande de réservation et la consultation de l’emploi du temps.

Dans chaque Testcase nous créons un objet de la classe à tester avec les valeurs que nous souhaitons. Par exemple, pour tester la classe Teacher on a choisi de prendre comme données celles d’Anne Laurent, donc on crée l’objet Teacher avec ses données puis on teste toutes les fonctions pour qu’elles retournent les données correspondantes.

# Tests d’intégration

Les tests d’intégration consistent lors de chaque modification à rejouer tous les tests unitaires pour vérifier que ces modifications n’ont pas altérer le comportement de toutes les autres fonctions du programme.

Pour mettre en place ces tests, nous avons utilisé Junit et ANT. La classe TestSuite de Junit nous permet de rassembler tous les TestCase et de définir une séquence de lancement de ceux-ci.

Par la suite avec ANT qui est l’équivalent du Makefile en Java, nous allons associer ce TestSuite à la compilation pour qu’à chaque compilation on effectue tous les tests unitaires. De cette manière à chaque modification du code, on relance la totalité des tests unitaires pour vérifier si ce nouveau code n’a pas altérer le reste du code.

# Tests fonctionnels

Pour effectuer les tests fonctionnels de notre application, nous avons établis pour chaque Use Case tous les scénarios d’utilisation possible.

# Tests de validation