Camille Raymond – Christophe Gire
Damien Sendner – Thibaut Rouquette

Projet URM: Tests

University Resources Management

Rapport de tests présentant la stratégie de test de l'application URM.

21/03/2012

Contenu

I. Stratégie de tests 2
II. Tests unitaires 2
III. Tests d'intégration 3
IV. Tests fonctionnels 3
V. Tests de validation 3
Annexes 4
1) Scénarios Authentification + Menu 4
2) Scénarios Consultation de l'emploi du temps 4
3) Scénarios Demande de réservation 5

I. Stratégie de tests

Nous avons établi une stratégie de tests à mettre en place 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'ait pas altéré 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 1 Plan de tests pour chaque Use Case

II. 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 ces 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.

III. 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 altéré 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 altéré le reste du code.

IV. Tests fonctionnels

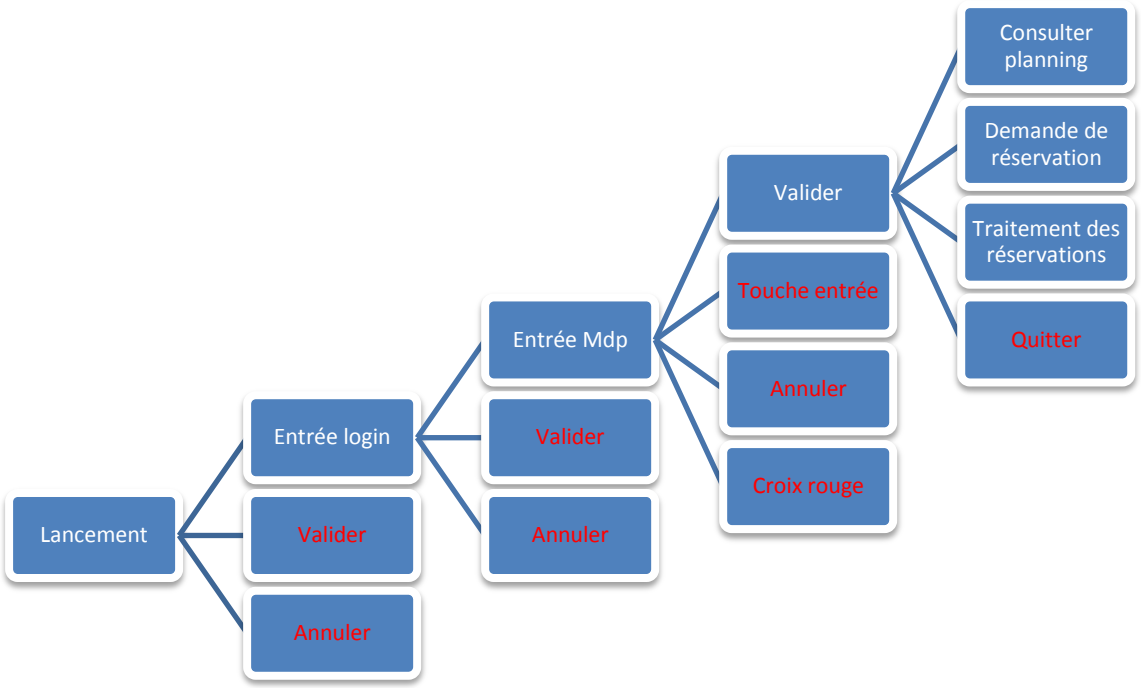
Pour effectuer les tests fonctionnels de notre application, nous avons établi pour chaque Use Case tous les scénarios d'utilisation possible.
Vous pourrez trouver ces scénarios en annexe.
Après avoir établi ces scénarios, on les joue un par un pour savoir si toutes les fonctionnalités de l'application fonctionnent bien. S'il y a un problème on le note.
Ici les scénarios du cas Demande de réservation ne sont pas exhaustifs car il est très difficile, quand l'application commence à devenir un peu compliquée, de déterminer tous les cas.
Il est possible d'automatiser ce genre de tests avec des logiciels tel que « TestComplete ». Cependant, nous ne disposons pas d'assez de temps pour cela, nous ferons donc nos tests à la main.
Dans un deuxième temps, nous avons également établi une liste de possibilités sur les données saisies par l'utilisateur (par exemple : chiffres, lettres, bon couple pseudo login, mauvais couple, etc.).

V. Tests de validation

Les derniers tests sont les tests de validation, qui consistent à vérifier point par point si chaque Use Case correspond à la demande établie dans la première phase de conception. Pour cela, on vérifie si les fonctions demandées sont disponibles et si l'ergonomie mise en place dans les maquettes correspond à celle que l'on avait défini.

Annexes

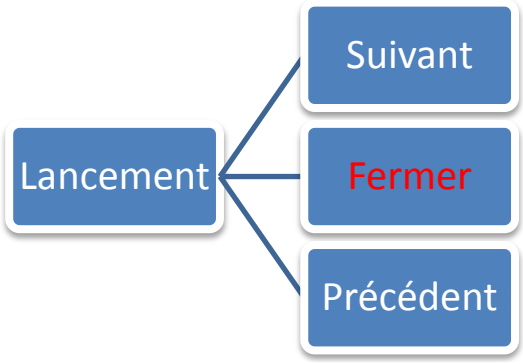
1) Scénarios Authentification + Menu



Possibilités de données :

- Bon couple login/mdp (qui existe dans la base)
- Mauvais couple login/mdp
- Super-user

2) Scénarios Consultation de l'emploi du temps

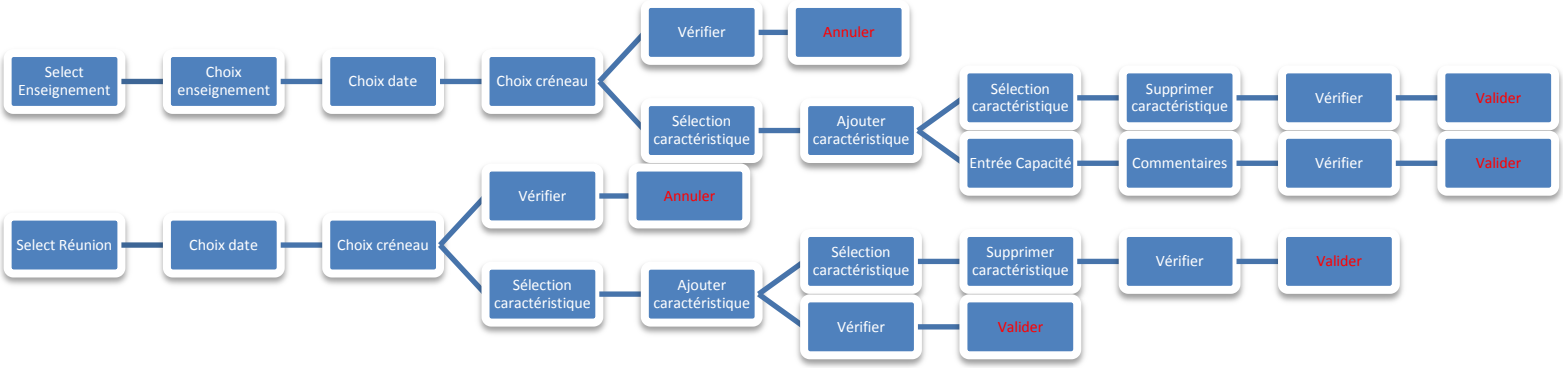


Après le lancement du menu ConsultaterPlanning : les trois choix qui s'offrent à l'utilisateur sont :

- La fermeture de la fenêtre via le bouton « Fermer »
- L'affichage la semaine suivante via le bouton « > »
- L'affichage la semaine précédente via le bouton « < »

Ces deux derniers cas entrainent un retour au lancement.

3) Scénarios Demande de réservation



Possibilités de données :

- Chiffres dans capacité.