***UNIBite – Testing***

Università degli studi di Bergamo  
Ingegneria Informatica

Beccarelli Raissa Matr. 1086785  
Locatelli Giacomo Matr. 1086262  
Valceschini Marco Matr. 1086356

***Immagine che contiene testo, Carattere, Elementi grafici, simbolo

Descrizione generata automaticamente***

**1 Testing**

Durante il ciclo di vita del nostro progetto ci siamo occupati, oltre allo sviluppo del codice, di eseguire dei test per verificare la correttezza e l’affidabilità del nostro sistema. I modelli su cui ci siamo basati per eseguire i test sono stati, per quanto riguarda quello di fase, il tipo “Dimostrazione”, ovvero ci siamo occupati di verificare che il software soddisfacesse le nostre aspettative e, per quanto riguarda il modello del ciclo di vita, il tipo “Valutazione” che ci ha permesso di rilevare i difetti di progettazione e di implementazione.

Il nostro processo di testing è stato suddiviso in due parti: una prima fase in cui abbiamo utilizzato test manuali e una seconda in cui abbiamo utilizzato dei test automatici.  
Durante la prima fase, ci siamo occupati di verificare la correttezza del codice man mano che questo veniva scritto. Partendo dal documento dei requisiti, abbiamo scelto volta per volta una funzione da implementare e, dopo aver scritto il codice, ci siamo occupati di controllare se la funzione svolgeva il suo compito adeguatamente, producendo il risultato aspettato.  
Durante la seconda fase, ovvero al termine della scrittura del codice e dell’implementazione di tutti i requisiti, abbiamo proseguito con la fase di test creando all’interno del package di test delle classi JUnit Test Case. All’interno del package abbiamo quindi scritto dei test suddividendoli in base alla classe a cui il metodo testato si riferiva.

**2 Casi di test**

Per quanto riguarda quindi la seconda fase abbiamo creato cinque classi di test all’interno del src/test/java:

* CarrelloTest;
* ConnessioneTest;
* PersonaleTest;
* PiattoTest;
* StudenteDocenteTest.

Queste classi corrispondono esattamente alle classi contenenti all’interno del package controller, nel src/main/java.  
Nella prima classe, CarrelloTest, ci siamo occupati di testare che:

* Quando un carrello viene creato la matricola che gli viene associata sia corretta;
* Quando un carrello viene creato, viene generata una lista di piatti associata vuota;
* Si possa aggiungere un piatto desiderato alla lista di piatti in modo corretto nel carrello (stesso test nel caso in cui il piatto desiderato sia un primo e si abbia scelto di aggiungere il formaggio);
* Nel momento in cui un piatto è aggiunto alla lista di piatti, nel database il numero di porzioni disponibili per quel piatto diminuisca veramente (e stessa cosa al contrario, nel caso in cui questo venga poi tolto dal carrello);
* Il prezzo dei piatti aggiunti nel carrello sia effettivamente quello mostrato a schermo;
* Il prezzo dei piatti mostrati a schermo, nel caso in cui uno StudenteDocente appartenga ad una fascia con diritto di sconto, sia effettivamente scontato.

Per la seconda classe, ConnessioneTest, abbiamo testato che:

* La connessione venga aperta correttamente;
* La connessione venga chiusa correttamente;
* Si possa aprire solo e soltanto una connessione.

Per la classe PersonaleTest, i test sono stati:

* La verifica del login del personale, verificando che lo username e la password siano correttamente associati ed esistano all’interno del database;
* La verifica che la creazione di un nuovo piatto da parte del personale venga correttamente inserita all’interno del database.

Per la classe PiattoTest:

* Abbiamo verificato che ad ogni piatto venisse associato il prezzo corretto rispetto al suo tipo di portata (primo, secondo, contorno, dolce e frutta);
* Abbiamo verificato che venissero settate il numero di porzioni desiderate, inserite in input, e fossero correttamente inserite nel database.

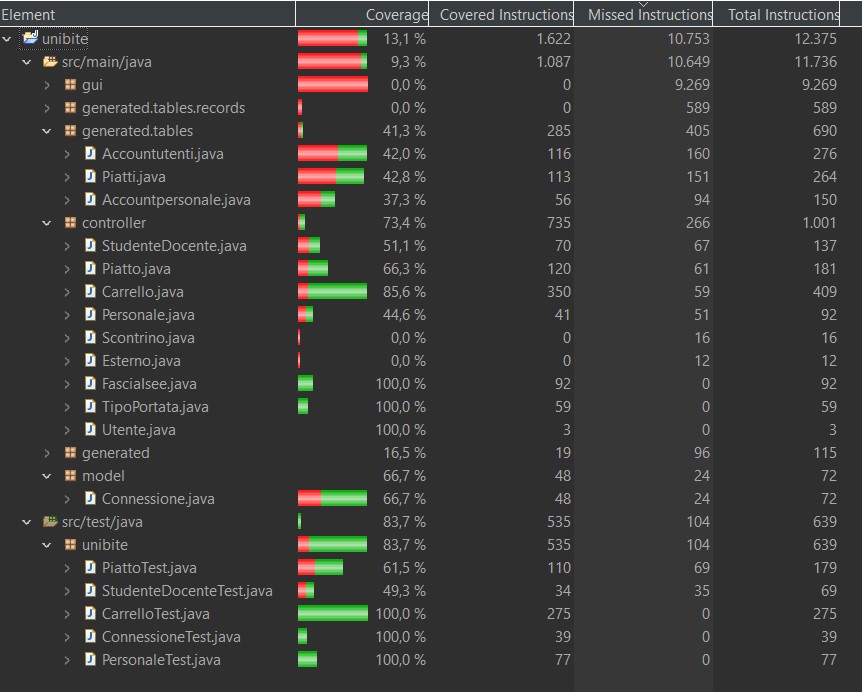
Per l’ultima classe, StudenteDocenteTest, abbiamo testato che:

* Come per il personale, il login funzionasse correttamente;
* Quando uno StudenteDocente viene creato, il conto virtuale sia settato a 0€.

Ci siamo quindi occupati di testare le funzioni più importanti, non andando a testare funzioni meno rilevanti, come ad esempio i getters e setters.

**3 Copertura**

Per quanto riguarda la copertura dei test, eseguendo i casi di test sopra elencati abbiamo misurato la copertura del codice:



Abbiamo quindi notato che sull’interfaccia grafica GUI, come ci aspettavamo, la copertura è dello 0%, in quanto non trovavamo rilevante verificare la correttezza della grafica.  
Per quanto riguarda le tabelle generate da JOOQ, abbiamo notato una copertura del 41%, dovuto al fatto che verificavamo la correttezza dell’inserimento dei dati all’interno del database.  
Per il package più rilevante, il controller, abbiamo una copertura del 73,4% che riteniamo abbastanza buona. Questa percentuale è dovuta al fatto che, come detto prima, le funzioni più rilevanti delle classi al suo interno sono state testate con i test automatici.  
Infine anche per la classe Connessione, riteniamo che la copertura del 67% sia buona.