HOMEWORK 3

Per lo svolgimento dell’homework 3 abbiamo deciso di usare lo stesso codice utilizzato per l’homework 1 .

Al netto del programma di validazione di una password, abbiamo indentificato 3 casi su cui sviluppare Property-based Testing :

1. Password valida
2. Password non valida
3. Password nulla

Tutti e 3 i casi sono stati gestiti da funzioni void che contengono all’interno del @Property un assertions per testare la classe.

Nel caso 1 e 2 abbiamo optato per la generazione di dati, sfruttando il @ForAll , di una tupla di 3 valori che sono :

* Boolean
* Boolean
* String

Contestualizzando :

1. Il primo valore rappresenta il booleano “requiresNum” e ci indica se la password debba o meno contenere un numero.
2. Il secondo valore rappresenta il booleano “requiresSpec” e ci indica se la password debba o meno contenere un carattere speciale.
3. La string è la password che viene generata.

La classe tiene conto anche di una lunghezza minima ed una massima per la password, queste sono inizializzate come costanti all’inizio della classe di test; è possibile modificarle senza intaccare l’integrità e coerenza del codice.

Entrando nel merito della definizione della password “valida” ed “invalida”, abbiamo utilizzato il concetto di arbitaries applicato al combine per generare i nostri valori.

Innanzitutto è stato usato l’arbitaries per generare il valore dei due booleani che saranno punto di riferimento per la presenza o meno di numeri o caratteri speciali nella password.

Usando un return sulla combinazione di quest’ultimi, come tupla di 2 elementi, abbiamo potuto usare il flatMap per generare la stringa in base al valore che il return ci da’ dei booleani. Grazie al flatMap è stato quindi possibile generare una tupla di ritorno finale che tenga conto di una password generata in base ai due valori booleani. Per fare ciò abbiamo semplicemente usato degli if per verificare i valori dei 2 booleani dati dal ritorno della combinations. Abbiamo identificato le diverse combinazioni, che sono commentate lato codice, ed effettuato il ritorn final che ci restituisce la tupla che sarà usata nel @Property per verificare l’assertions. Tutte le combinazioni tengono conto anche della lunghezza della stringa. In particolare, nel @Provide della stringa invalida, ci sono combinazioni di ritorno della str che tengono conto come discriminante della riuscita o meno del test, anche della lunghezza di quest’ultima. Se, per esempio, il booleano del numero è vero, vuol dire che per fallire non debbano esserci numeri; però il test può fallire anche se ci sono numeri ma la lunghezza è inferiore alla lunghezza minima. Abbiamo cercato di riprodurre tutti i casi possibili, per sviluppare dei test i migliori possibili.

Ci teniamo a dire che tutte le combinazioni usate per generare la password sono del tipo che viene generata una stringa di solo lettere a cui viene aggiunta alla fine il numero e/o il carattere speciale. Si sarebbe potuto usare un approccio di casualità di inserimento di questi valori, quindi non necessariamente alla fine. Non l’abbiamo fatto perché il programma valida la presenza o meno di questi valori, quindi non è rilevante dove siano all’interno della stringa.

Nel caso 3 (password nulla), abbiamo usato due distinti @ForAll per i due valori booleani e nessuna tupla. Questo perché, verificando il comportamento di una password nulla, non c’è una vera necessità di generare password. Abbiamo semplicemente definito testando il metodo con un valore nullo, infatti non c’è nessun @Provide.

Tutti i test , svolti come precedentemente esposto usando la libreria jqwik, danno tutti esito positivo, non ci sono test non riusciti.

Aggiunger informazioni sulla collection dei dati