TID:

Test Integration

Document



Transport Efficiency Manager

|  |  |
| --- | --- |
| Riferimento |  |
| Versione | 0.2 |
| Data | 09/04/2021 |
| Destinatario | Prof.ssa F. Ferrucci |
| Presentato da | Team NC08 |
| Approvato da |  |

RevisionHistory

|  |  |  |  |
| --- | --- | --- | --- |
| **Data** | **Versione** | **Descrizione** | **Autori** |
| 03/04/2021 | 0.1 | Versione iniziale | Francesca Moschella, Federica Attianese, Federica Pica |
| 09/04/2021 | 0.2 | Ultimazione | Francesca Moschella, Federica Attianese  Federica Pica |

Indice dei contenuti

[RevisionHistory 2](#_Toc75453573)

[1. Introduzione 4](#_Toc75453574)

[2. Riferimenti 4](#_Toc75453575)

[3. Integration testing 4](#_Toc75453576)

[**3.1 Strategia selezionata** 4](#_Toc75453577)

[**3.2 Strumenti utilizzati** 4](#_Toc75453578)

[4. Componenti testate 4](#_Toc75453579)

# Introduzione

La fase che segue quella del testing di unità è quella del testing di integrazione, che consiste nella verifica della correttezza delle interazioni tra due o più componenti del sistema, nel rispetto delle Specifiche di Integrazione.

Il testing di integrazione rappresenta una delle fasi di testing più importanti, e questo documento ha il compito di identificare e descrivere la strategia di testing scelta per questa fase e ritenuta la più adatta al nostro sistema.

# Riferimenti

Il documento di riferimento è: NC08\_SDD\_ver.1.1.

# Integration testing

**3.1 Strategia selezionata**

La strategia ritenuta più adatta alle esigenze di sviluppo del sistema per la fase di integrazione del testing è quella di tipo **“Bottom-up”**, questa tipologia di approccio permette di effettuare prima il testing sul funzionamento delle componenti base posizionate al livello più basso della gerarchia (specificate come Entity e Service) e poi, successivamente, di controllare come esse si integrano con le componenti di livello superiore che le utilizzano per fornire un servizio.

**3.2 Strumenti utilizzati**

Oltre all’utilizzo del framework JUnit, per i test di unità, si è fatto ricorso all’utilizzo di un altro framework open source, Mockito, che permette di simulare diversi scenari utili al testing.

# Componenti testate

In questa sezione sono specificate le componenti da testare selezionate tenendo conto della strategia scelta, Bottom-up.

Per quanto riguarda il layer **Model**, le componenti testate sono:

* Profilo;
* Utente;
* DettaglioUtente;
* Corsa;
* DatiCorsa;
* Conducente;
* Linea;
* Mezzo;
* ProgrammaCorse;
* AccountService;
* ProgrammaCorseService;
* DatiCorsaService;
* RisorseService;
* ProgrammaAutomaticoMaker;

Per quanto riguarda il layer **Controller**, le componenti da testare sono:

* AccountController;
* CorsaController;
* DatiCorsaController;
* ProgrammaCorseController;
* RisorseController;
* TemErrorController.