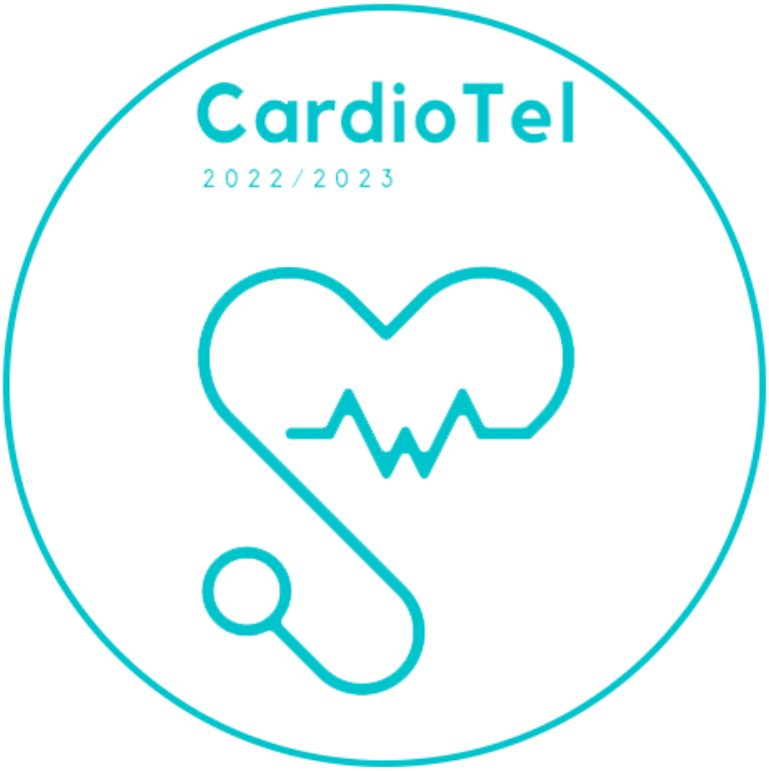


# Test Plan
# CardioTel

| | |
|---|---|
| **Riferimento** | C10_TP_1.0 |
| **Versione** | 1.0 |
| **Data** | 06/11/2022 |
| **Destinatario** | Top Management |
| **Presentato da** | Team C10 |
| **Approvato da** | |

Revision History

| Data | Versione | Descrizione | Autori |
|---|---|---|---|
| 06/12/2022 | 0.1 | Prima stesura | La Monica, Bacco |
| 08/12/2022 | 0.2 | Aggiunta Panoramica Sistema<br>Feature Da Testare, Pass Fail Criteria | La Monica, Bacco |
| 10/12/2022 | 0.3 | Aggiunti Approcci e Tipologie Testing | La Monica, Bacco |
| 12/12/2022 | 1.0 | Revisione Finale | La Monica, Bacco |

| | |
|---|---|
| **Intro** | L'obiettivo del progetto CardioTel ha l'obiettivo di fornire una piattaforma web per il tele monitoraggio delle attività cardiache di un paziente in modo da trattare e preventivare eventuali complicanze dovute alla variazione di determinati parametri vitali.<br>La piattaforma, in particolare, monitora tramite sistemi IOT certificati l'attività cardiaca e altri parametri relati in modo da individuare tempestivamente anomalie, offre inoltre un chatbot per fornire soluzioni immediate al trattamento dell'anomalia, consente di creare un report personalizzato da dare ai medici in caso di operazioni ed interventi futuri ove la mano del medico chirurgo risulti necessaria e, tramite un sistema di machine learning, fornisce al paziente e all'azienda un sistema per monitorare l'accuratezza delle stime sulle anomalie individuate.<br>Sono state individuate attività di testing per ogni endpoint del sistema, in particolare:<br>- Endpoint "Device"<br>- Endpoint "Predizione"<br>- Endpoint "Report"<br>- Endpoint "ChatBot" |
| **Riferimenti** | Si fanno riferimento ai seguenti documenti, in particolare<br>- **RAD**: I test case presenti nel seguente Test Plan sono elaborati considerando i requisiti funzionali e non funzionali presenti nel RAD<br>- **SDD**: I test case presenti nel seguente Test Plan devono sottostare alla suddivisione in sottosistemi individuata nell'SDD |
| **Panoramica Del Sistema** | Il sistema proposto presenta l'architettura three-tier, Per info più in dettaglio, si rimanda all'SDD, sotto la voce architettura di sistema, dove sono indicate anche le tecnologie utilizzate per ogni layer |
| **Feature Da Testare** | Le Feature da testare sono collegate ai main task di ogni endpoint, in particolare:<br>- Endpoint "Device" |

| | |
|---|---|
| | - Testing corretto saving dei dati sul db per relative CRUD.<br>- Endpoint "Predizione"<br>- Testing accuratezza pipeline learner<br>- Endpoint "Report"<br>- Testing corretta generazione file di report per i medici<br>- Endpoint "ChatBot"<br>- Testing corretta automazione delle risposte in base ai sintomi selezionati<br>Altri requisiti funzionali definiti a bassa/media priorità verranno testati solo se necessario e con budget di effort ancora disponibile. |
| **Pass/Fail Criteria** | Per ogni test, verrà identificato un oracolo, con lo scopo di individuare eventuali comportamenti non attesi/sbagliati da parte dell'implementazione del relativo requisito funzionale.<br>Un test ha successo (pass) se, dato un input al sistema, l'output ottenuto è diverso dall'output atteso dall'oracolo. Un test fallisce (fail) se, dato un input al sistema, l'output ottenuto è uguale all'output atteso dall'oracolo.<br>L'attività di Testing verrà considerata valida se i seguenti vincoli saranno rispettati:<br>- Vengono testati tutti i RF ad alta priorità;<br>- Viene effettuato test di regressione ogni volta che si introducono nuove caratteristiche al sistema che impattano la main feature dell'endpoint o se vengono modificate quelle presenti;<br>- Raggiungimento di branch coverage pari ad almeno il 75% |
| **Approccio** | Il testing dell'intero sistema si compone di tre fasi: testing di sistema, testing di integrazione e testing di unità. Verranno progettati nell'ordine appena definito, ma verranno eseguiti in ordine inverso.<br>Prima della fase di implementazione del sistema, avverrà la progettazione dei casi di test di sistema, perfezionati in seguito nella loro fase di esecuzione; durante la fase implementativa avverrà la progettazione dei casi di test di unità. |

| | |
|---|---|
| | Durante lo sviluppo saranno eseguite periodiche attività di revisione sul codice prodotto.<br>Poiché la progettazione è organizzata seguendo un modello simile al modello a V, il testing di sistema è stato pianificato in seguito alla stesura del documento Requirements Analysis Document, mentre la pianificazione del testing di integrazione avverrà dopo la stesura del System Design Document |
| **Testing Di Sistema** | Functional testing<br>- Il functional testing ha il fine di validare i requisiti funzionali. Consiste nell'individuare i possibili faults generati dagli input degli utenti.<br>Performance Testing<br>- Grazie a Quarkus, si possono testare le perfomance del singolo endpoint<br><br>Acceptance Testing<br>- L'acceptance testing verrà effettuato solo sul functional testing<br>Deployment Testing<br>- Per questione di budget e carenza skill, il sistema non potrà essere deployato su sistemi kubernetes-like. Si prevede ulteriore budget per il training ed il suo successivo deployment |
| **Testing Di Integrazione** | Verrà utilizzato un approccio bottom-up, metodo ritenuto più adatto per un software basato sul paradigma Object Oriented.<br>La definizione dei test case avverrà tramite il framework Quarkus.<br>Verrà valutato l'utilizzo di Github Actions per realizzare la Continuous Integration, in alternativa a Travis CI. L'automatizzazione del run dei test sarà gestita da Maven, ed infine come tool di misurazione e report coverage sarà utilizzato JaCoCo |
| **Testing Di Unità** | Per il testing di unità la strategia prevista consiste nel testare ogni metodo delle classi del sistema esclusi i getter/setter. I casi di test saranno definiti attraverso un approccio black-box e saranno documentati direttamente nel codice, |

| | |
|---|---|
| | attraverso l'uso del framework Quarkus, che permette di testare gli endpoint attraverso funzioni built-in ad hoc.<br>Per ogni Production Class sarà definita una Test Class che rispetterà il formato NomeProductionClassTest. Tali classi saranno scritte in parallelo alle Production class, per garantire una più facile copertura del codice. Le stesse classi saranno poi revisionate e modificate da sviluppatori differenti |
| **Ispezione Del Codice** | Per aumentare la qualità del codice ci si affiderà principalmente ai controlli e della CI/CD con Maven. Sebbene vi sia l'intenzione di fare ispezione del codice, a causa del basso budget non si può assicurare tale pratica. |
| **Criteri Di Sospensione E Ripristino** | In questa sezione verranno specificati i criteri di sospensione del test e le attività di test che dovranno essere ripetute quando si riprende il test.<br>**Sospensione**<br>- Il testing non verrà sospeso fino alla sua terminazione, anche in caso di rilevazione di una failure. Il testing potrà essere momentaneamente sospeso nel caso venga restituito, al momento dell'esecuzione, un errore nella definizione di uno dei test stessi<br>**Ripristino**<br>- Il testing verrà ripreso dopo aver risolto i fault individuati |
| **Materiale Per Il Testing** | L'hardware necessario per l'attività di test è un semplice computer con una connessione ad Internet con installato docker |
| **Test Cases** | L'approccio per la definizione dei test frame sarà il category partition. Al fine di minimizzare il numero di test case, gli input saranno partizionati in classi di equivalenza. Per definire l'output atteso si userà un oracolo umano, per via dell'assenza di specifiche formali/semi-formali |