



UNIVERSITÀ DEGLI STUDI DI  
NAPOLI FEDERICO II

Facoltà di Ingegneria Informatica Magistrale

Elaborato Software Testing

## **Software Testing**

### **Testing Blackbox, Whitebox, Lato Server e Client in una Web Application**

Anno Accademico 2021/2022

Candidato  
**Simone Bianco**  
matr. M63/000817

# Indice

<b>1</b>	<b>Introduzione</b>	<b>1</b>
1.1	Descrizione del Progetto . . . . .	1
1.2	Casi d’Uso . . . . .	2
1.3	Requisiti (Formato Breve) . . . . .	3
1.4	Requisiti (Formato Informale) . . . . .	3
1.5	Cenni all’Architettura dell’Applicazione . . . . .	5
1.5.1	Framework . . . . .	5
1.5.2	Design Pattern utilizzati . . . . .	6
1.5.3	Ulteriori Dettagli . . . . .	7
1.6	Considerazioni Importanti . . . . .	7
1.6.1	Setup Ambiente . . . . .	7
1.6.2	Server Built-In <sup>1</sup> . . . . .	9
1.6.3	Autenticazione a due Fattori di Google . . . . .	9
<b>2</b>	<b>Testing Backend - PHPUnit</b>	<b>12</b>
2.1	Organizzazione Directory . . . . .	13
2.2	Setup Iniziale . . . . .	13
2.3	Scrivere Test con PHPUnit <sup>2</sup> . . . . .	16
2.4	Testing Funzionale Blackbox . . . . .	16

---

<sup>1</sup>Laravel Built-in Server

<sup>2</sup>Fonte: [Documentazione PHPUnit](#)

---

2.4.1	Unit Test . . . . .	17
2.4.2	Integration Testing . . . . .	24
2.4.3	Test d'Integrazione per ReservationRepository . . . . .	24
2.4.4	Test Creazione Prenotazione da Parte del Paziente . . . . .	31
2.4.5	Test Relativi alla Sicurezza . . . . .	32
2.4.6	Test di Robustezza sulla Chiamata API della Prenotazione . . . . .	35
2.5	Testing Strutturale Whitebox . . . . .	37
2.5.1	Criterio di Terminazione . . . . .	37
2.5.2	XDebug . . . . .	38
2.5.3	Coverage di ReservationRepository . . . . .	44
2.5.4	Codice Coperto con i Test Whitebox . . . . .	58
2.5.5	Code Coverage Rimanente . . . . .	59
2.5.6	Conclusioni . . . . .	68
<b>3</b>	<b>Testing Frontend - Dusk</b>	<b>70</b>
3.1	Configurazione . . . . .	70
3.1.1	Setup File d'Ambiente . . . . .	71
3.1.2	Setup Database . . . . .	71
3.1.3	Creazione File di Base . . . . .	72
3.2	Scrittura ed Esecuzione dei Test . . . . .	73
3.2.1	First Login Test . . . . .	74
3.3	Valutazione della Test Suite Realizzata . . . . .	79
3.3.1	Efficacia . . . . .	79
3.3.2	Efficienza . . . . .	80
3.3.3	Conclusioni . . . . .	80
3.4	Differenze tra PHPUnit e JUnit . . . . .	80
3.5	Approcci Alternativi - Analisi Statica . . . . .	81

---

<b>4 Testing Frontend - Testim</b>	<b>82</b>
4.1 Katalon Recorder . . . . .	82
4.2 <b>Testim</b> . . . . .	83
4.2.1 First Login Test . . . . .	84

# Elenco delle figure

1.2.1 Diagramma dei casi d'uso . . . . .	2
1.6.1 2FA Google . . . . .	9
1.6.2 2FA nella Web Application . . . . .	10
1.6.3 2FA nell'Applicazione Google Authenticator . . . . .	11
2.0.1 PHPUnit . . . . .	12
2.2.1 phunit.xml . . . . .	14
2.4.1 Package Diagram Unit Test . . . . .	17
2.4.2 Package Diagram ReservationRepository . . . . .	18
2.4.3 Esempio Mockery . . . . .	18
2.4.4 Mock con Closure . . . . .	19
2.4.5 Package Test Blackbox . . . . .	24
2.4.6 Grafo delle Dipendenze ReservationRepository . . . . .	25
2.4.7 Test Paziente Crea Prenotazione . . . . .	32
2.5.1 Package Test Whitebox . . . . .	37
2.5.2 XDebug Logo . . . . .	38
2.5.3 Code Coverage . . . . .	40
2.5.4 Code Coverage con Path e Branches . . . . .	41
2.5.5 Complexity . . . . .	42
2.5.6 Coverage Report ReservationRepository . . . . .	43

---

2.5.7 Report ReservationController . . . . .	43
2.5.8 Coverage Report AndroidApiController -> reservationPost . . . . .	44
2.5.9 Coverage Report ReservationRepository nel dettaglio . . . . .	45
2.5.10CFG completeAndSave . . . . .	46
2.5.11CFG cancelAndStockIncrement . . . . .	48
2.5.12CFG createAndStockDecrement . . . . .	50
2.5.13CFG createRecallAndStockDecrement . . . . .	52
2.5.14CFG changeVaccineAndConfirm . . . . .	54
2.5.15CFG confirmAndSave . . . . .	56
2.5.16Coverage Report Finale . . . . .	58
2.5.17Coverage Report Finale con Branch e Path . . . . .	58
2.5.18Rimozione dalla Coverage delle Classi non usate . . . . .	59
2.5.19Copertura Raggiunta . . . . .	59
2.5.20Gerarchia Eccezioni . . . . .	60
2.5.21Input testRepositoryCoverage . . . . .	62
2.5.22decrementAndSave di StockRepository . . . . .	63
2.5.23decrementAndSave di StockRepository corretta . . . . .	64
2.5.24Coverage repositoryTest . . . . .	64
2.5.25Coverage testRouteCoverageKWay . . . . .	66
2.5.26Coverage testRouteCoverageRandom . . . . .	67
2.5.27Coverage testRouteCoverageKWay + testRouteCoverageRandom . . . . .	67
2.5.28Codice Morto . . . . .	68
2.5.29Copertura Finale . . . . .	68
3.2.1 Automa a Stati Finiti dell'Interfaccia . . . . .	73
3.2.2 Automa a Stati Finiti dell'Interfaccia . . . . .	74
3.2.3 Activity 2FA . . . . .	75

---

3.2.4 /login . . . . .	77
3.2.5 /qr/register . . . . .	77
3.2.6 /qr/authenticate . . . . .	78
3.2.7 /dashboard . . . . .	78
4.1.1 Katalon Recorder . . . . .	82
4.2.1 Testim . . . . .	83
4.2.2 Sequenza delle Operazioni in Testim.io . . . . .	85
4.2.3 Sequence Test First Login . . . . .	86
4.2.4 Fake Login . . . . .	87
4.2.5 Fake Login Input Inseriti . . . . .	88
4.2.6 Broken Login Input Inseriti . . . . .	89

# **Capitolo 1**

## **Introduzione**

Per l'esame di Software Design Architecture è stata sviluppata un'applicazione web tramite l'utilizzo del framework Laravel (per il backend) e di VueJs per il frontend (si riporta alla documentazione del progetto per tutti i dettagli sull'applicazione). Lo scopo del seguente elaborato è quello di progettare e sviluppare test sulla sopraccitata applicazione sia per il lato backend che per quello frontend, usando le tecniche viste durante il corso di Software Testing.

### **1.1 Descrizione del Progetto**

Il progetto consiste nella realizzazione di una web application per la gestione delle prenotazioni di vaccini anti-Covid. L'obiettivo del sistema è quello di facilitare i pazienti, nella prenotazione del vaccino tramite una comoda applicazione o semplicemente da un web browser.

Il servizio della prenotazione dovrà essere reso indipendente dalla tecnologia utilizzata dal client mettendo a disposizione endpoint contattabili da qualsiasi device, in modo da rendere l'applicazione facilmente estensibile in futuro. Per dimostrare il funzionamento del servizio, l'idea è quella di sviluppare un prototipo di app Android che consentirà ai pazienti di prenotarsi tramite i servizi offerti dall'applicazione centrale lato server scritta con il linguaggio PHP.

I pazienti non saranno gli unici a trarre vantaggio dalla piattaforma, i responsabili sanitari, infatti, potranno utilizzare la piattaforma (solo tramite web browser) per gestire le richieste di prenotazioni, ovvero confermarle, rifiutarle o modificarle. Potranno inoltre confermare la somministrazione del

vaccino e prenotare il richiamo. La piattaforma deve garantire anche la gestione delle scorte dei vaccini presenti in un centro vaccinale.

Di seguito sono riportati requisiti e casi d'uso, i quali sono stati usati come base per elaborare alcuni test, per informazioni aggiuntive riguardo architettura e struttura dell'applicazione, si riporta alla documentazione completa di progetto<sup>1</sup>.

## 1.2 Casi d'Uso

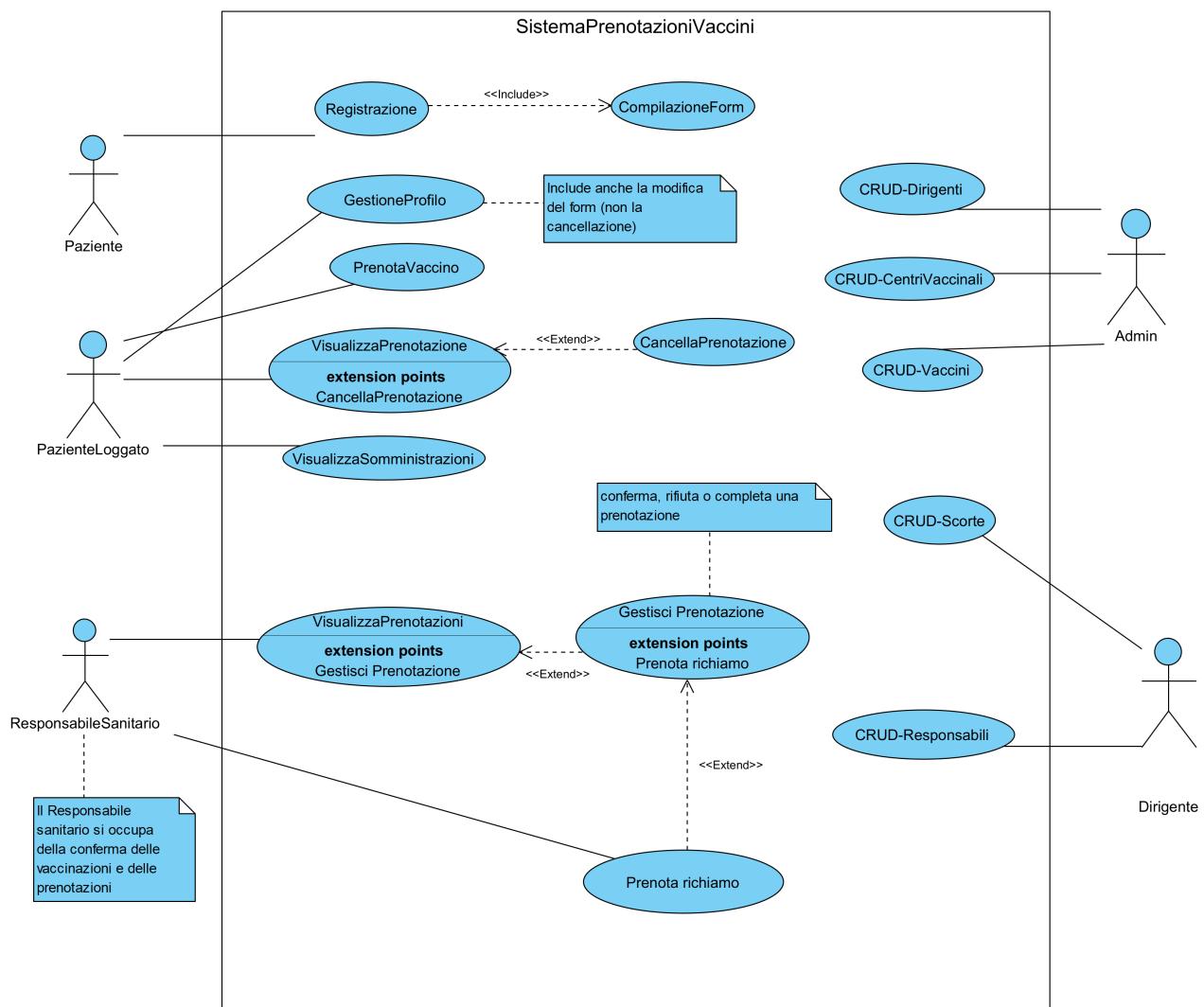


Figura 1.2.1: Diagramma dei casi d'uso

<sup>1</sup>Si trova in allegato nella cartella, non è stata messa qui per non rendere l'elaborato troppo dispersivo

## 1.3 Requisiti (Formato Breve)

Attore	Caso d'Uso	Descrizione
Paziente	Registrazione	Il paziente prima del login deve registrarsi alla piattaforma
	Compilazione Profilo	Il paziente per potersi registrare deve compilare il profilo con tutte le informazioni richieste
Paziente Loggato	Gestione Profilo	Il paziente deve poter modificare le proprie informazioni precedentemente immesse all'atto della registrazione
	Prenota Vaccino	Una volta completato il login, il paziente può effettuare la prenotazione di un vaccino richiedendola dall'app.
	Visualizza Prenotazione	Consente la visualizzazione della prenotazione che ha effettuato (se esiste)
	Cancella Prenotazione	Consente al paziente loggato di poter cancellare la prenotazione
	Visualizza Somministrazioni	Il paziente deve poter controllare le informazioni di tutti i vaccini che si è fatto somministrare
Responsabile Sanitario	Visualizza Prenotazioni	Il responsabile sanitario, tramite interfaccia web, può visualizzare tutte le informazioni relative alle prenotazioni della struttura sanitaria
	Gestisci Prenotazione	Il responsabile sanitario deve poter confermare, rifiutare o completare la prenotazione di un paziente
	Prenota Richiamo	Se previsto, il responsabile sanitario gestisce la prenotazione di una seconda dose di un determinato paziente

Tabella 1.1: Casi d'uso in formato breve

## 1.4 Requisiti (Formato Informale)

- **Registrazione:** Procedura di registrazione all'app da parte del paziente, obbligatoria quando accede per la prima volta alla piattaforma. Il paziente dovrà compilare correttamente tutti i campi richiesti (e-mail, password, nome, cognome, data, sesso, codice fiscale, città, cap, telefono e malattie) affinché la registrazione vada a buon fine.
- **Login:** Procedura di login da parte del paziente, del personale sanitario o del dirigente per accedere alla piattaforma.

- **Modifica profilo:** Dalla homepage consente al paziente di modificare le informazioni del profilo immessi all'atto della registrazione compreso il form per le eventuali malattie. Se il paziente immette dati non validi o non compila campi obbligatori il sistema non salva le modifiche e rindirizza il paziente sulla homepage. L'indirizzo e-mail non può essere modificato.
- **Prenota vaccino:** Permette di prenotare un vaccino scelto dal sistema. La regione e la struttura sanitaria viene scelta dal paziente (verranno mostrate tutte le strutture relative a quella regione). Il paziente può scegliere la data dell'appuntamento scegliendo tra quelle disponibili mostrate dal sistema, l'ora invece verrà assegnata in base al numero di prenotazioni totali della struttura in quella giornata. Il responsabile dovrà poi confermare o meno la prenotazione.
- **Visualizza prenotazione (Paziente):** Il paziente una volta che effettua il login entra automaticamente nella home dove gli verranno mostrate tutte le informazioni (struttura, data, ora, lotto-vaccino, stato) relative alla prenotazione, se esiste.
- **Cancella prenotazione:** Consente al paziente di cancellare la propria prenotazione. La cancellazione si può effettuare al massimo entro le 72 ore dalla data dell'appuntamento. In seguito, il paziente potrà effettuare una nuova prenotazione.
- **Visualizza somministrazioni:** Consente al paziente di visualizzare le vaccinazioni che ha già effettuato. Cliccando sulla pagina di visualizza somministrazioni, il sistema mostra al paziente le prenotazioni già completate (ovvero con la conferma della somministrazione). Se non esistono prenotazioni completate il sistema mostra un messaggio di somministrazioni non presenti. Se il paziente clicca su una somministrazione verrà rindirizzato sulla pagina di dettaglio della somministrazione in cui può decidere di scaricare un attestato di vaccinazione.
- **Visualizza Prenotazioni (Responsabile sanitario):** Permette all'utente di visualizzare tutte le prenotazioni oppure un gruppo di prenotazioni in base ad un filtro (da confermare, confermata, cancellata, completata) relative al centro vaccinale di cui fa parte. È possibile ricercare anche le prenotazioni relative ad un singolo paziente digitando il suo nome e cognome nella barra di ricerca.

- **Prenota richiamo:** Consente al responsabile sanitario di prenotare un eventuale richiamo per il paziente dopo la somministrazione della prima dose di un vaccino. Il responsabile dalla pagina di gestione prenotazioni può decidere di prenotare un richiamo cliccando su un pulsante dalla pagina di gestione prenotazione, solo se la prenotazione precedente risulta nello stato completata. A questo punto il responsabile viene rindirizzato sulla pagina di prenotazione di un richiamo relativo allo stesso paziente, in cui dovrà compilare i campi come data, orario, vaccino, e cliccare su conferma. La nuova prenotazione verrà salvata con lo stato confermata e sarà di tipo richiamo.
- **Gestici prenotazione:** Consente al responsabile sanitario di confermare, rifiutare, completare o modificare una prenotazione (ovvero confermare l'avvenuta somministrazione) dopo aver selezionato una prenotazione dalla pagina di visualizza prenotazioni. Se la prenotazione è nello stato “da confermare” allora saranno disponibili solo i pulsanti per confermarla o rifiutarla. Se la prenotazione è nello stato “confermata” o “cancellata” allora è disponibile solo il pulsante per completarla. Se il responsabile modifica una prenotazione attraverso i campi del form, questa verrà salvata solo in concomitanza della pressione del pulsante per confermare.

## 1.5 Cenni all’Architettura dell’Applicazione

Ricapitolando, l’applicazione risiede su di un server, gli operatori sanitari interagiscono tramite browser mentre i pazienti tramite applicazione Android.

### 1.5.1 Framework

#### Backend

Il backend dell’applicazione è stato sviluppato usando Laravel, un framework open source di tipo MVC scritto in PHP per lo sviluppo di applicazioni web. Distribuito con licenza MIT, mantiene tutto il codice disponibile su GitHub e viene indicato, in base al punteggio GitHub e StackOverflow, come il framework PHP più popolare. Alcune delle caratteristiche sono: un sistema di gestione

---

dei pacchetti modulare con un gestore delle dipendenze dedicato, differenti modalità di accesso ai database relazionali, strumenti che aiutano la distribuzione e la manutenzione dell'applicazione.

## Frontend

Il frontend è stato sviluppato con VueJs, un framework JavaScript open-source in configurazione Model–view–viewmodel per la creazione di interfacce utente e single-page applications.

## Interazione Frontend-Backend

Normalmente VueJs interagisce col server mediante chiamate API, per questa applicazione, invece, è stato utilizzato InertiaJs, un adapter frontend e backend. Costituisce un nuovo approccio rispetto allo sviluppo delle classiche server-driven web app permettendo di creare applicazioni single page completamente renderizzabili dal client senza la complessità delle moderne SPA facendo leva sul framework lato server. In altre parole il client comunicherà sempre tramite chiamate API, ma la loro gestione avviene in maniera trasparente dall'adapter come se fossero delle normali chiamate HTTP.

### 1.5.2 Design Pattern utilizzati

#### Repository e Validation<sup>2</sup>

I repository costituiscono un ulteriore livello di astrazione tra i dati usati dall'applicazione e quelli contenuti nel database, mentre le classi di validation encapsulano la logica di validazione dei dati che devono essere salvati. Le operazioni effettuate su una model, al momento della creazione o del salvataggio, potrebbero comprendere degli step aggiuntivi che seguono un determinato pattern comune che bisognerebbe ripetere in ogni punto del programma. Ad esempio, nel caso degli stock, deve essere generato un codice alfanumerico casuale ogni qualvolta ne viene creato uno nuovo, o più in generale tutti i salvataggi nel database, per salvaguardare la consistenza e la correttezza dei dati, dovrebbero essere validati prima di effettuare la richiesta al DB.

Invece di ripetere ogni volta queste operazioni, abbiamo creato un repository e una classe di validation per ciascun business object che encapsulano tali operazioni.

---

<sup>2</sup>Validation non è considerato proprio un design pattern, piuttosto una best practice

## Dependency Injection

Dependency injection (DI) è un design pattern della Programmazione orientata agli oggetti il cui scopo è quello di semplificare lo sviluppo e migliorare la testabilità di software di grandi dimensioni. Per utilizzare tale design pattern è sufficiente dichiarare le dipendenze di cui un componente necessita (dette anche interface contracts). Quando il componente verrà istanziato, un iniettore si prenderà carico di risolvere le dipendenze (attuando dunque l'inversione del controllo). Se è la prima volta che si tenta di risolvere una dipendenza l'injector istanzierà il componente dipendente, lo salverà in un contenitore di istanze e lo restituirà. Se non è la prima volta, allora restituirà la copia salvata nel contenitore. Una volta risolte tutte le dipendenze, il controllo può tornare al componente applicativo. Come descritto in precedenza, l'utilizzo di questo pattern è stato semplificato da Laravel.

### 1.5.3 Ulteriori Dettagli

Per ulteriori dettagli si riporta alla documentazione completa di progetto.

## 1.6 Considerazioni Importanti

Di seguito sono riportati alcuni punti fondamentali che sono stati presi in considerazione per testare l'applicazione.

### 1.6.1 Setup Ambiente

Sono riportati di seguito gli step necessari per avviare web application e test:

1. E' necessario avere PHP 7.4.24 nel PC con la cartella aggiunta alla variabile d'ambiente PATH, si consiglia di installarlo tramite [XAMPP](#) (in caso di windows è suggerito mettere come cartella di installazione C://xampp)
2. In allegato alla cartella consegnata, è presente una directory chiamata 'xampp', copiare e incollare il contenuto all'interno della cartella di xampp appena installato. Al suo interno, infatti, sono

presenti il file di configurazione php.ini e l'estensione di xdebug che andranno a sovrascrivere quelli di default nella cartella d'installazione.

3. Il progetto è reperibile al [repository di GitHub](#), tuttavia eseguire il setup richiederebbe di installare le librerie tramite Composer (l'equivalente di Gradle per PHP) e di compilare il codice JavaScript tramite NodeJs, ragion per cui è stato allegato alla consegna dell'elaborato un file .rar contenente la cartella dell'applicazione già configurata (software-testing-backend.rar)
4. La root contiene il file d'ambiente .env, il quale specifica il database usato dall'applicazione, nonché l'indirizzo e la porta utilizzati. Per il database è stato configurato un DB SQLite (contenuto nel file database/sqlite.development.data), per cui non è necessario effettuare ulteriori operazioni, mentre per quanto riguarda URL e porta di default sono impostati rispettivamente http://127.0.0.1 e 8000, per cui è possibile connettersi all'applicazione scrivendo nel browser http://127.0.0.1:8000 (dopo aver avviato run\_server.bat)

```
APP_NAME=Laravel
APP_ENV=local
APP_KEY=base64:/Wa44t6aDPMvASAPjs1NzSkGOTIPFkNivwgbll2NK6c=
APP_DEBUG=true
APP_URL=http://127.0.0.1:8000
```

Qualora si desiderasse cambiare tale impostazione, è possibile modificare il .env

5. All'interno della root sono presenti i seguenti file batch:
  - run\_server.bat -> lanciarlo per avviare il server
  - run\_tests.bat -> lanciarlo per avviare i test
  - run\_tests\_parallel.bat -> esegue i test in parallelo (consigliato vista la mole dei test)
  - run\_dusk\_tests.bat -> esegue i test del frontend (è necessario prima avviare il server tramite run\_server.bat)

6. E' stato predisposto il seguente account di default per l'accesso tramite interfaccia web:

- email: operator@email.it
- password: test

7. Nella cartella è presente anche l'applicazione android che si interfaccia all'applicazione, tuttavia non è oggetto di questo elaborato

### 1.6.2 Server Built-In<sup>3</sup>

Tipicamente converrebbe usare un web server come Apache o Nginx per hostare un applicazione Laravel, tuttavia è possibile utilizzare Laravel Homestead, una macchina virtuale compresa nel repository progettata per creare un ambiente di sviluppo virtuale, eseguibile tramite il comando:

```
php artisan serve
```

Oppure lanciando, come accennato, il file run\_server.bat

L'applicazione sarà raggiungibile tramite browser all'indirizzo 127.0.0.1:8000

### 1.6.3 Autenticazione a due Fattori di Google



Figura 1.6.1: 2FA Google

<sup>3</sup>[Laravel Built-in Server](#)

Solitamente, per effettuare l'accesso all'interno di un'applicazione, è sufficiente inserire username e password. Tale metodo, tuttavia, non è sufficiente al fine di rispettare i requisiti non funzionali di sicurezza imposti nella documentazione di progetto. Infatti, chiunque ottenga email e password di un operatore sanitario, potrebbe accedere al pannello di controllo e causare danni al sistema.

Per questo motivo è stata implementata l'autenticazione a 2 passaggi d Google<sup>4</sup>.

Quando viene effettuato l'accesso, viene generato un codice QR e una secret key di 64 caratteri.

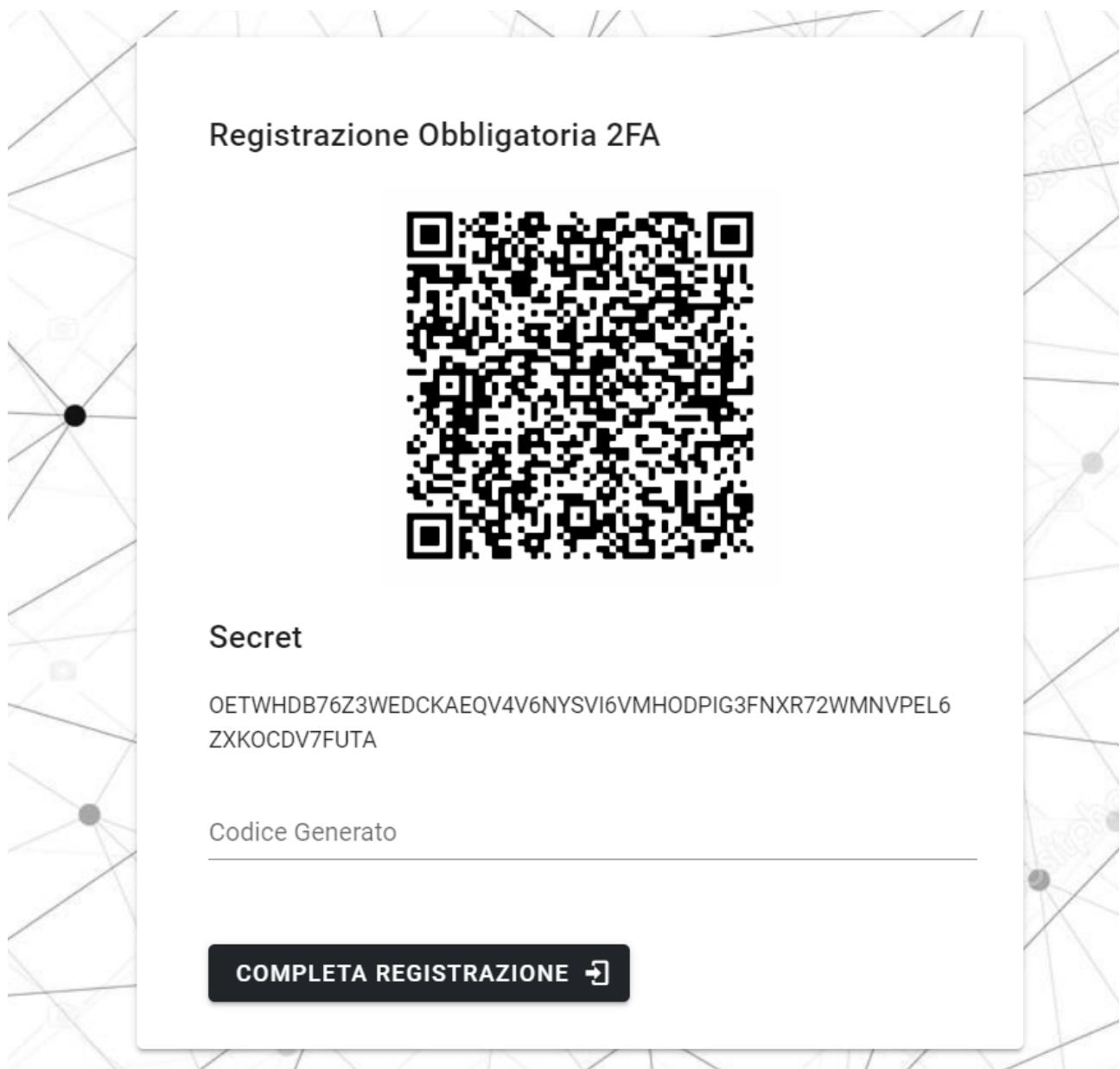


Figura 1.6.2: 2FA nella Web Application

<sup>4</sup>[Come funziona](#)

All’utente viene chiesto di usare l’applicazione “Google Authenticator” per scansionare il codice QR. Una volta scansionato il codice, l’applicazione fornirà un codice composto da 6 numeri (OTP) generato ogni 30 secondi, da inserire per effettuare l’accesso.

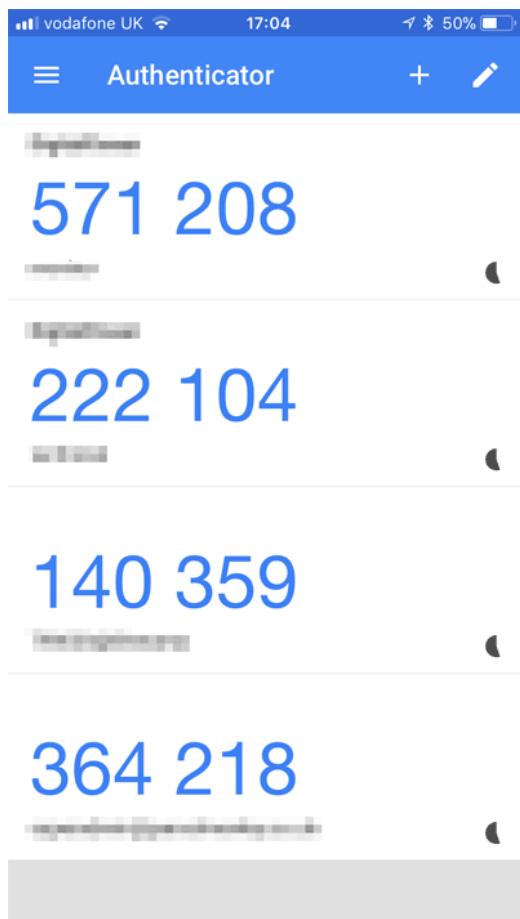


Figura 1.6.3: 2FA nell’Applicazione Google Authenticator

Tale codice verrà chiesto ad ogni accesso, mentre codice QR e secret verranno mostrati solo e solamente al primo accesso. E’ molto importante per l’utente conservare la secret key, in quanto in caso di perdita del cellulare o dell’applicazione mobile, senza la secret non sarà più possibile generare codici per il login.

Tale funzionalità è stata implementata in Laravel tramite la libreria [pragmarx/google2fa](#)

# Capitolo 2

## Testing Backend - PHPUnit



Figura 2.0.1: PHPUnit

I primi test realizzati sono stati quelli dedicati al lato backend dell'applicazione. Come menzionato, il backend è stato sviluppato usando Laravel, un framework basato su PHP che integra PHPUnit. PHPUnit è un framework di testing sviluppato per il linguaggio PHP. È un'istanza dell'architettura xUnit per i framework di unit testing originati con SUnit ed è diventato popolare con JUnit. PHPUnit è stato creato da Sebastian Bergmann e il suo sviluppo è ospitato su GitHub. Come altri framework di testing, anche PHPUnit usa le asserzioni per verificare che il comportamento del software coincida con quello previsto.

## 2.1 Organizzazione Directory

Di default, la directory “tests” contiene 2 subdirectories: Feature e Unit. Basandoci sulla documentazione di Laravel, Unit dovrebbe contenere i test più semplici, quelli destinati al testing di piccole porzioni isolate di codice (es. dei singoli metodi o classi) che non accedono a risorse e/o servizi esterni come chiamate API, mentre quelli in Feature testano porzioni più grandi di codice dove interagiscono diversi oggetti e/o endpoint esterni.<sup>1</sup> I test dedicati al frontend, invece, si trovano in una directory a parte chiamata Browser che non è definita all'interno di phpunit.xml

## 2.2 Setup Iniziale

All'interno della root è incluso un file di configurazione chiamato “phpunit.xml”, il quale consente di impostare le impostazioni di base di PHPUnit.

---

<sup>1</sup>Fonte: [Testing Laravel](#)

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <phpunit xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3      xsi:noNamespaceSchemaLocation=".vendor/phpunit/phpunit/phpunit.xsd"
4      bootstrap="vendor/autoload.php"
5      colors="true">
6      <testsuites>
7          <testsuite name="Unit">
8              <directory suffix="Test.php">./tests/Unit</directory>
9          </testsuite>
10         <testsuite name="Feature">
11             <directory suffix="Test.php">./tests/Feature</directory>
12         </testsuite>
13     </testsuites>
14     <coverage processUncoveredFiles="true">
15         <include>
16             <directory suffix=".php">./app</directory>
17         </include>
18         <report>
19             <html outputDirectory="tests/reports/coverage"/>
20         </report>
21     </coverage>
22     <php>
23         <server name="APP_ENV" value="testing"/>
24         <server name="BCRYPT_ROUNDS" value="4"/>
25         <server name="CACHE_DRIVER" value="array"/>
26         <server name="DB_CONNECTION" value="sqlite"/>
27         <server name="MAIL_MAILER" value="array"/>
28         <server name="QUEUE_CONNECTION" value="sync"/>
29         <server name="SESSION_DRIVER" value="array"/>
30         <server name="TELESCOPE_ENABLED" value="false"/>
31     </php>
32 </phpunit>

```

Figura 2.2.1: phpunit.xml

E' stata aggiunta, sotto il tag <php>, la seguente riga:

```
<server name="DB_DATABASE" value=":memory:"/>
```

In questa maniera, quando verranno effettuate delle operazioni che coinvolgono il DB, i test useranno un database SQLite temporaneo creato in memoria centrale<sup>2</sup> dedicato esclusivamente ai test. Al fine di rendere indipendente ciascun test da tutti gli altri, è stato imposto il refresh di tutti i dati all'inizio di ciascun test. Ciò è stato reso possibile includendo il tratto RefreshDatabase<sup>3</sup> all'interno delle varie classi di test, il quale fa in modo che ad ogni esecuzione il DB venga ricreato lanciando le migration (database\migrations).

Per quanto riguarda il riempimento del DB, sono state valutate 2 alternative:

<sup>2</sup>SQLite in-memory DB

<sup>3</sup>DatabaseTesting

1. Riempire le tabelle con dati casuali

2. Inserire dati scelti manualmente

Per quanto la prima opzione possa essere più conveniente dal punto di vista dell'effort (dopotutto le classi per generare dati casuali sono state già create per la presentazione dell'applicazione), potrebbero compromettere la **Ripetibilità**, infatti non è detto che le fuzioni che generano dati casuali non siano affette da bug o che comunque la randomicità dei dati prodotti non comprometta il corretto funzionamento del software, o che il risultato dei test risulti differente ad ogni esecuzione.

Rimuovendo completamente la componente di randomicità dai test siamo sicuri di ottenere, dati gli stessi input e le stesse precondizioni, gli stessi risultati. Tuttavia, inserire tutti i dati manualmente è altrettanto problematico, ad esempio che succede se volessimo fare un test su 100 pazienti? O un test di performance con 20 strutture e migliaia di prenotazioni caricate? Sarebbe impensabile scrivere tutto a mano. Per ovviare a questo problema, sono state progettate le classi di seeder dei test, in maniera tale da creare record i cui attributi siano prevedibili a priori in base a un certo numero di fattori. Ad esempio, se si vogliono creare 10 pazienti, basta passare al seeder dei pazienti 10 e avremo:

Paziente 1:

- nome: nome paziente 1
- cognome: cognome paziente 1
- email: email\_paziente\_1@emai.it

Paziente 2:

- nome: nome paziente 2
- cognome: cognome paziente 2
- email: email\_paziente\_2@emai.it



etc...

Qualora avessimo bisogno di creare dati con valori custom, sarà sempre possibile aggiungerli.

## 2.3 Scrivere Test con PHPUnit<sup>4</sup>

La struttura dei test in PHPUnit è molto simile a quella dei test scritti in JUnit, in particolare l'esecuzione di ciascun test è preceduta dall'esecuzione di un metodo di setUp e seguita da uno di tearDown. Tutti i metodi di test devono cominciare col termine test e quando viene eseguita una classe contenente più test, il **flusso di esecuzione** è il seguente:

1. setUpBeforeClass: metodo statico eseguito al setup della classe (prima del primo setUp)
2. setUp: metodo eseguito prima di ciascun test definito nella classe
3. assertPreConditions: metodo eseguito prima dell'inizio di ogni test definito nella classe, usato per verificare determinate pre condizioni
4. assertPostConditions: metodo eseguito alla fine di ogni test definito nella classe, usato per verificare determinate post condizioni
5. tearDown: metodo eseguito alla fine di ciascun test
6. tearDownAfterClass: metodo statico eseguito dopo tutti i test della classe (dopo l'ultimo tearDown)

Le classi di test devono estendere la classe TestCase. Anche qui è possibile utilizzare delle notazioni per definire delle proprietà delle classi e dei metodi di test.

## 2.4 Testing Funzionale Blackbox

I test di seguito riportati sono funzionali, perché verificano il funzionamento del software, e blackbox perché non andiamo a ragionare o a prendere in considerazione la struttura del codice.

<sup>4</sup>Fonete: [Documentazione PHPUnit](#)

## 2.4.1 Unit Test

Gli unit test vengono effettuati su componenti isolate del sistema, per verificare che il loro comportamento coincida con quello aspettato.

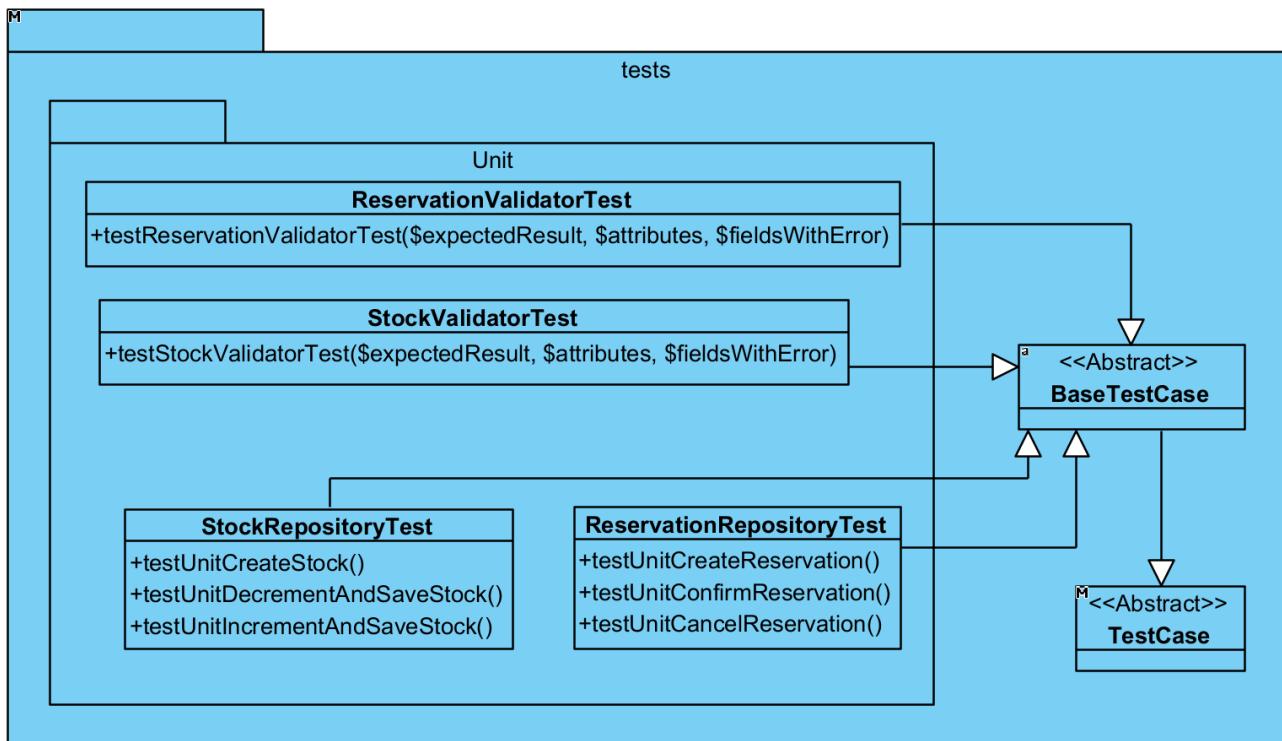


Figura 2.4.1: Package Diagram Unit Test

E' stata scritta una classe astratta, tests\BaseTestCase.php, che estende TestCase e fa in maniera tale che il database venga riempito con un set di dati di base.

Supponiamo di non avere ancora i repository pronti per essere usati, pertanto il salvataggio dei dati verrà effettuato direttamente tramite le model, in particolare utilizzando le factory (database\factories). Laravel, infatti, mette a disposizione queste classi<sup>5</sup> che consentono di definire una logica di creazione per le model (le quali, in Laravel, astraggono sia l'oggetto con i suoi attributi che la tabella nel DB), in maniera tale che quando vengono istanziate tramite factory, queste si occupano di settare in maniera trasparente alcune delle sue proprietà (ad esempio degli attributi che si vogliono impostare di default).

<sup>5</sup>Sono state pensate per il testing e per riempire il DB con dati **fittizzi**, ma in realtà possono essere sfruttate anche per altri scopi

In questo caso, BaseTestCase istanzia 1 struttura, 1 paziente, 1 vaccino, 1 lotto e 1 stock i cui dati sono stati scelti manualmente.

## ReservationRepositoryTest

Unità sotto test: app\Repositories\ReservationRepository.php

Classe: tests/Unit/ReservationRepositoryTest

Descrizione: ReservationRepository è il repository per il quale passano tutte le operazioni riguardanti le prenotazioni. Analizziamo il seguente package diagram:

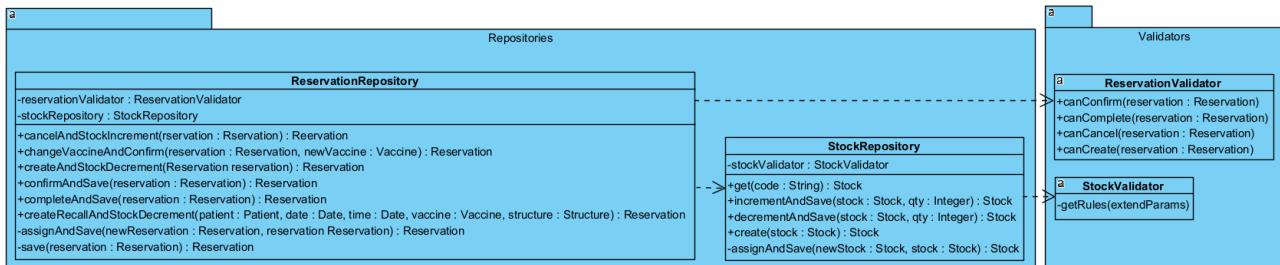


Figura 2.4.2: Package Diagram ReservationRepository

ReservationRepository è dipendente da StockRepository e ReservationRepository, le quali sono associate alla classe tramite dependency injection. Supponiamo di non avere un'implementazione per le classi StockRepository e ReservationValidator, è comunque possibile scrivere test per verificare il funzionamento del repository, in quanto Laravel integra la classe Mockery<sup>6</sup>, che consente di creare dei mock per le classi non ancora implementate. Ad esempio:

```

20     $this->instance(
21         abstract StockRepository::class,
22         Mockery::mock( ...args: StockRepository::class, function (Mockery\MockInterface $mock) {
23             $mock->shouldReceive( ...methodNames: 'decrementAndSave')
24                 ->once()
25                 ->withArgs(fn($stock) => true);
26         })
27     );

```

Figura 2.4.3: Esempio Mockery

<sup>6</sup>Mockery

fa in maniera tale che durante l'esecuzione del test, quando il service container deve injectare tra le dipendenze di una classe StockRepository, passa invece la classe di mock definita da tale istruzione. Sempre qui, è possibile definire chiamate ai metodi fintizie, in questo caso decrementAndSave che può essere richiamata una sola volta, accetta un singolo parametro (\$stock) e ritorna sempre true. E' anche possibile passare come argomenti una closure, così da simulare parte del comportamento del metodo da implementare:



```
85     $this->instance(
86         abstract: StockRepository::class,
87         Mockery::mock( ...args: StockRepository::class, function (Mockery\MockInterface $mock) {
88             $mock->shouldReceive( ...methodNames: 'incrementAndSave')
89                 ->once()
90                 ->withArgs(function ($stock) {
91                     $stock->quantity++;
92                     $stock->save();
93                     return true;
94                 });
95         });
96     );
```

Figura 2.4.4: Mock con Closure

Precondizioni:

- Esiste una struttura
- Esiste un paziente
- Esiste un vaccino
- Esiste un lotto
- Esiste uno stock



Test Case ID		createAndStockDecrement		
Test Case Description		Chiamata alla funzione createAndStockDecrement di ReservationRepository, che crea una prenotazione e decrementa lo stock del vaccino scelto dal sistema		
Step #	Test Data	Step #	Test Data	
1	date = oggetto carbon con data di oggi più 5 giorni	1	stock_id = id dell'unico stock memorizzato nel DB	
1	time = '12:00'	1	notes = 'test notes'	
1	patient_id = id dell'unico paziente memorizzato nel DB			
Step #	Step Details	Output Atteso	Actual Results	Result
1	Chiamata al metodo createAndStockDecrement di ReservationRepository usando i mock per ReservationValidator e StockRepository	Un'istanza della model Reservation contenente i dati della prenotazione appena creata	As Expected	Pass
Postcondizioni:				
E' presente il record della prenotazione nel database				
Lo stato della prenotazione è 'pending'				
Le note della prenotazione sono 'test notes'				

Test Case ID		testUnitConfirmReservation		
Test Case Description		Chiamata alla funzione confirmAndSave di ReservationRepository, che conferma una prenotazione esistente		
Step #	Test Data	Precondizioni		
1	reservation = oggetto Reservation contenenti i dati dell'unica prenotazione esistente	Esiste una prenotazione		
1	notes = 'test notes 2'			
Step #	Step Details	Output Atteso	Actual Results	Result
1	Chiamata al metodo confirmAndSave di ReservationRepository usando i mock per ReservationValidator e StockRepository	Un'istanza della model Reservation contenente i dati della prenotazione esistente	As Expected	Pass
Postcondizioni:				
Le note della prenotazione sono 'test notes 2'				
Lo stato della prenotazione è 'pending'				

Test Case ID		testUnitCancelReservation		
Test Case Description		Chiamata alla funzione cancelAndStockIncrement di ReservationRepository, che annulla una prenotazione esistente e decremente lo stock del vaccino che era stato assegnato		
Step #	Test Data	Precondizioni		
1	reservation = oggetto Reservation contenenti i dati dell'unica prenotazione esistente	Esiste una prenotazione		
1	notes = 'test notes 3'			
Step #	Step Details	Output Atteso	Actual Results	Result
1	Chiamata al metodo confirmAndSave di ReservationRepository usando i mock per ReservationValidator e StockRepository	Un'istanza della model Reservation contenente i dati della prenotazione esistente	As Expected	Pass
Postcondizioni:				
Le note della prenotazione sono 'test notes 3'				
Lo stato della prenotazione è 'cancelled'				



## ReservationRepositoryTest

Unità sotto test: app\Repositories\StockRepository.php

Classe: tests\Unit\StockRepositoryTest.php

Descrizione: StockRepository è il repository per il quale passano tutte le operazioni riguardanti creazione, incremento e decremento degli stock.



## Reservation Validator Testing

Unità sotto test: app\Validators\ReservationValidator.php

Classe di test: tests\Unit\ReservationValidatorTest.php

Descrizione: E' la classe responsabile della validation degli attributi della prenotazione, è richiamata dal repository delle prenotazioni prima dei salvataggi ed è una delle classi per le quali è stato creato un mock nel test precedente.

E' stato usato un approccio data driven per testare le varie combinazioni di dati, mentre è stata adottata la tecnica delle classi equivalenti per scegliere i dati di input, testando prima la combinazione valida, poi facendo variare tutte le classi di equivalenza insieme (usando quindi la strategia della copertura minima). Il validator non si fermerà al primo errore, ma restituirà un array contenente la lista di attributi che hanno dato errore, quindi possiamo verificare per ogni test che gli errori ottenuti siano quelli che ci aspettavamo e quindi il test non perderà di precisione. Le classi di equivalenza scelte sono le seguenti:

- date: { null, stringa vuota, data futura in formato Y-m-d (valido), data passata }
- state: { null, stringa vuota, stato valido (pending, completed, cancelled, confirmed), "INVAILID\_STATE" }
- time: { null, stringa vuota, orario valido, orario non valido (> 24:00 o < 00:00), stringa alfanumerica }
- note: { null, stringa di massimo 255 caratteri (valido), stringa più grande di 255 caratteri }



- code: { null, stringa vuota, stringa massimo di 32 caratteri (valido), stringa maggiore di 255 caratteri, stringa contenente caratteri non consentiti (AA22##\_\_..) }
- patient\_id: { null, stringa vuota, id <= 0, id > 1 ma di un record non esistente, id di un record presente nel database (valido) }
- stock\_id: { null, stringa vuota, id <= 0, id > 1 ma di un record non esistente, id di un record presente nel database (valido) }

Per un totale di 5 test.

Precondizioni:

- Esiste una struttura
- Esiste un paziente
- Esiste un vaccino
- Esiste un lotto
- Esiste uno stock

Input: reservationProvider definito in tests\Unit\ReservationValidatorTest.php

Output Aspettato:

- il primo dataset (l'unico valido) non provoca eccezione di validazione, tutti gli altri si
- i campi che hanno fallito la validazione coincidono con quelli specificati nel data provider

Output Ottenuto: As Expected

Risultato del Test: Ok

## Stock Validator Testing

Classe sotto test: app\Validators\StockValidator.php

Classe di test: tests\Unit\StockValidatorTest.php

Descrizione: E' la classe responsabile della validation degli attributi dello stock, è richiamata dal repository degli stock ed è una delle classi per le quali è stato creato un mock nel test ReservationValidatorTest.

E' stato utilizzato il medesimo approccio del test precedente, con le seguenti classi di equivalenza:

- quantity: {null, -1, int overflow, stringa vuota, valore compreso tra 0 e max int (valido) }
- code: { null, stringa vuota, stringa diversa da 32 e minore di 255, stringa pari a 32 (valido), stringa maggiore di 255 caratteri }
- structure\_id: { null, stringa vuota, < 0, >1 non presente nel DB, id di un record presente nel db (valido) }
- batch\_id: { null, stringa vuota, < 0, >1 non presente nel DB, id di un record presente nel db (valido) }

Per un totale di 6 test.

Precondizioni:

- Esiste una struttura
- Esiste un vaccino
- Esiste un lotto

Input: data provider definito in tests\Unit\StockValidatorTest.php

Output Aspettato:

- il primo dataset (l'unico valido) non provoca eccezione di validazione, tutti gli altri si
- i campi che hanno fallito la validazione coincidono con quelli specificati nel data provider

Output Ottenuto: As Expected

Risultato del Test: Ok

## 2.4.2 Integration Testing

A differenza degli unit test, nei test di integrazione non testiamo più le componenti isolate, ma andiamo man mano a sostituire classi e metodi, simulati tramite stub e mock, con le reali implementazioni.

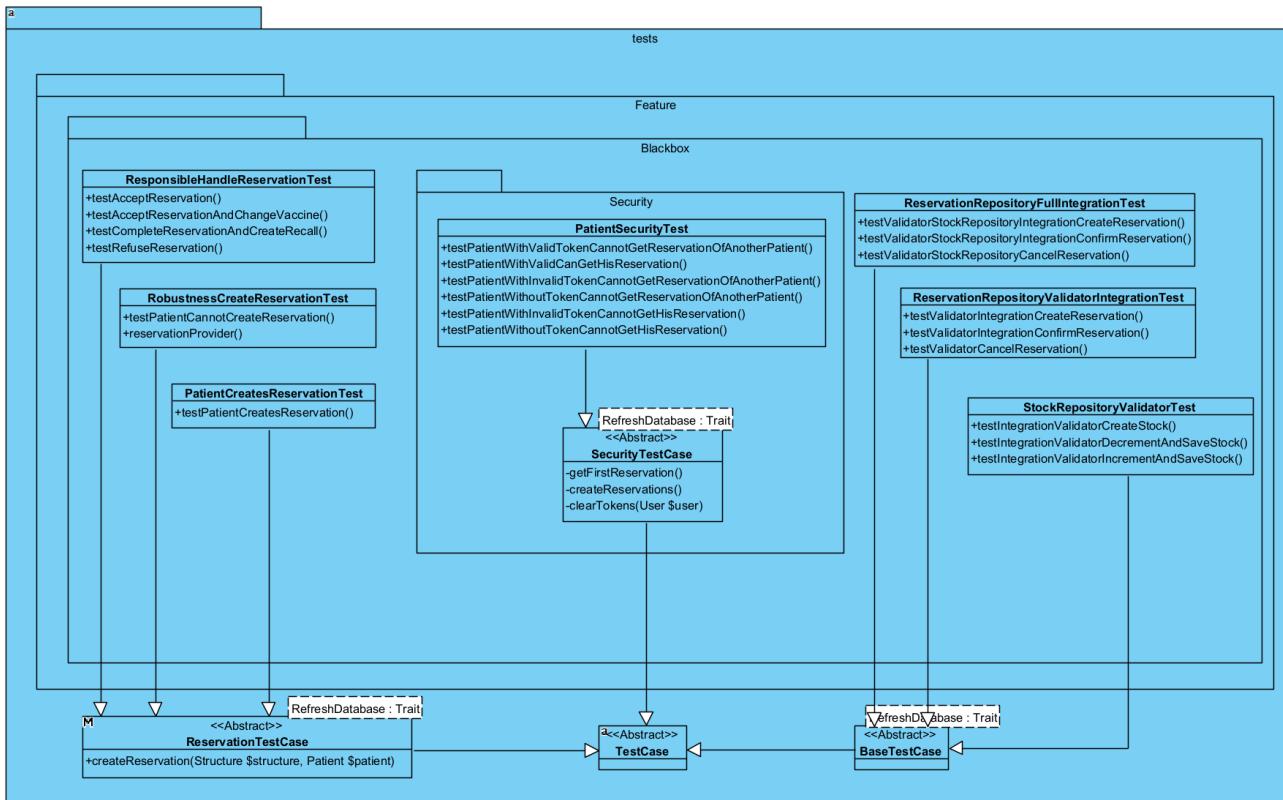


Figura 2.4.5: Package Test Blackbox

## 2.4.3 Test d'Integrazione per ReservationRepository

Esaminiamo il grafo delle dipendenze:

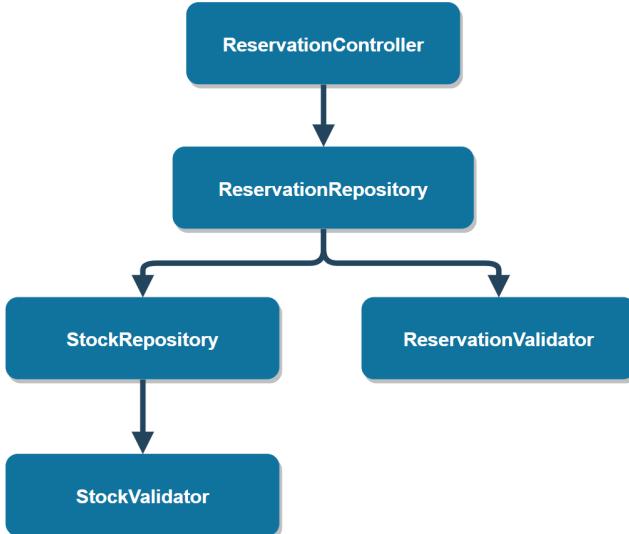


Figura 2.4.6: Grafo delle Dipendenze ReservationRepository

Una volta che si hanno a disposizione le classi implementate, bisogna scrivere dei test d'integrazione per verificare che funzionino correttamente insieme alla classe padre. A tale scopo, è stato adottato un approccio di integrazione bottom-up, integrando secondo questo ordine:

1. StockValidator + StockRepository
2. ReservationRepository + ReservationValidator
3. StockValidator + StockRepository + ReservationRepository + ReservationValidator
4. ReservationController + StockValidator + StockRepository + ReservationRepository + ReservationValidator

### **StockValidator + StockRepository**

Classe: tests\Feature\Blackbox\StockRepositoryValidatorIntegration.php

Descrizione: i test sono praticamente identici a quelli di StockRepositoryTest, con la differenza che non viene usato un mock per StockValidator



## ReservationRepository + ReservationValidator

Classe: tests\Feature\Blackbox\ReservationRepositoryValidatorIntegration.php

Descrizione: i test sono praticamente identici a quelli di ReservationRepositoryTest, con la differenza che non viene usato un mock per ReservationValidator

## StockValidator + StockRepository + ReservationRepository + ReservationValidator

Classe: tests\Feature\Blackbox\ReservationRepositoryFullIntegration.php

Descrizione: i test sono praticamente identici a quelli di ReservationRepositoryTest, con la differenza che non vengono usati mock

## Integrazione con ReservationController

Ora che abbiamo testato ReservationRepository individualmente, possiamo scrivere un test che simuli le chiamate HTTP alle quali sono associate i metodi del controller della prenotazione.

Prima di scrivere i test, dobbiamo tenere in considerazione i seguenti fattori:

- Simulare il login dell'utente: tutte le operazioni effettuate richiedono che l'utente sia loggato.  
Il login non si traduce altro che nella presenza nella tabella “sessions” nel database di un record contenente le informazioni della sessione, come l'ID dell'utente, il browser dal quale si è effettuato il login, etc... E' possibile simulare l'autenticazione nei test di PHPUnit usando l'istruzione “be”, specificando l'utente (memorizzato in “users”) del quale si vuole simulare l'autenticazione
- Simulare l'autenticazione 2FA: per i responsabili sanitari è obbligatorio, per accedere al sito, inserire il codice a 6 cifre della 2FA di google. Quando l'utente inserisce il codice, viene settato a “true” il flag “2fa” associato alla sessione e l'utente ottiene il permesso di accedere all'applicazione.
- Per quanto riguarda le date, una prenotazione non può essere creata in data precedente a quella attuale, quindi i test potrebbero fallire se eseguiti in futuro, ragion per cui verrà sempre scelta

come base la data attuale e verrà usata la classe di Laravel [Carbon](#) per aggiungere un offset (ad esempio data attuale + 5 giorni), così da aumentare la robustezza dei test

- Sempre al fine di migliorare la robustezza dei test, quando dobbiamo prelevare un record dal database, invece di inserire l'ID diretto facciamo riferimento alla sua posizione tramite [Eloquent](#)<sup>7</sup>. In questo caso, dato che esistono solo un paziente, un responsabile ed una struttura, è sufficiente prendere sempre il primo record del database e siamo sicuri di ottenere sempre lo stesso risultato. La motivazione principale è che in questo momento stiamo usando SQLite come database e siamo sicuri che l'ID del primo elemento inserito sia sempre 1, tuttavia non è detto che nell'implementazione finale non si decida di usare un altro database (in altre parole, in questo modo il test è più robusto)

I seguenti test estendono `ReservationTestCase` (come mostrato nel class diagram), il quale lancia all'inizio di ciascun test un seeder che riempie il database con una struttura, un paziente, 4 vaccini, un lotto e uno stock per ogni vaccino e infine una prenotazione da parte del paziente, oltre che d includere il tratto `RefreshDatabase`, che ripristina il database allo stato originale prima e dopo ogni test<sup>8</sup>.

Per la progettazione dei casi di test, si sono tenuti in considerazione i requisiti, ovvero le possibili operazioni che il responsabile può effettuare sulla prenotazione.

A differenza di `BaseTestCase`, `ReservationTestCase` usa i repository e i seeder per effettuare il riempimento del database.

I test simulano la sessione dell'operatore sanitario creato dal seeder, ciascuno effettua una chiamata al metodo `'update'` del controller `'app/http/Controllers/ReservationController.php'`.

Classe: `tests/Feature/Blackbox/ResponsibleHandleReservationTest`

Precondizioni:

- E' presente una sola struttura sanitaria nel database
- Sono presenti 4 vaccini
- Sono presenti 4 lotti (1 per ogni vaccino)

<sup>7</sup>Eloquent è un object-relational mapper grazie al quale è possibile accedere al database in maniera semplice ed intuitiva

<sup>8</sup>Perché refreshare il database prima e dopo ciascun test? Così i test sono indipendenti ed possibile eseguirli in parallelo

- E' presente uno stock per ogni lotto
- E' presente un solo responsabile
- E' presente un solo paziente
- Esiste una prenotazione in stato di attesa
- L'utente è loggato come responsabile
- L'utente ha effettuato l'autenticazione a 2 Fattori



Test Case ID		testAcceptReservation
Test Case Description		Test per verificare che un responsabile può accettare una prenotazione in stato di attesa
Step #	Test Data	
1	vaccine = Pfizer	
1	state = confirmed	
1	reservation_id = ID della prenotazione creata dal seeder	
Step #	Step Details	Output Atteso
1	Effettua richiesta PUT all'URL /prenotazione/reservation_id/update	Redirect alla route 'reservations.index' (/prenotazioni)
Postcondizioni:		
Lo stato della prenotazione è "confirmed"		
La data della prenotazione è quella odierna + 1 giorno alle 12:00		
Vaccino e stock sono gli stessi che erano stati assegnati prima del test		

Test Case ID		testAcceptReservationAndChangeVaccine
Test Case Description		Test per verificare che un responsabile può accettare una prenotazione cambiando il vaccino scelto dal sistema
Precondizioni		Step #
Esiste lo stock di un altro vaccino disponibile		1 vaccine = Moderna
		1 state = confirmed
		1 reservation_id = ID della prenotazione creata dal seeder
Step #	Step Details	Output Atteso
1	Effettua richiesta PUT all'URL /prenotazione/reservation_id/update	Redirect alla route 'reservations.index' (/prenotazioni)
Postcondizioni:		
Lo stato della prenotazione è "confirmed"		
La data della prenotazione è quella odierna + 1 giorno alle 12:00		
Lo stock della prenotazione è cambiato ed è uno stock riferito al vaccino scelto per il cambio		
Lo stock che era stato inizialmente assegnato è stato incrementato di 1		
Lo stock assegnato è stato decrementato di 1		

Test Case ID		testCompleteReservationAndCreateRecall
Test Case Description		Test per verificare che un responsabile possa completare una prenotazione (lo stato "completed" indica che è stata effettuata la somministrazione del vaccino e che quindi la prenotazione può essere archiviata). Il test effettua 3 step, parte dalla prenotazione creata, la conferma, la completa e infine crea il richiamo
Step #	Test Data	Step #
1	vaccine = Pfizer	2 vaccine = Pfizer
1	state = confirmed	2 state = completed
1	reservation_id = ID della prenotazione creata dal seeder	2 reservation_id = ID della prenotazione creata dal seeder
Step #	Test Data	Step #
3	vaccine = Pfizer	3 patient = ID dell'unico paziente esistente
3	time = 12:00	3 structure = nome dell'unica struttura esistente
3	date = tra 100 giorni	
Step #	Step Details	Output Atteso
1	Effettua richiesta PUT all'URL /prenotazione/reservation_id/update	Redirect alla route 'reservations.index' (/prenotazioni)
2	Effettua richiesta PUT all'URL /prenotazione/reservation_id/update	Redirect alla route 'reservations.index' (/prenotazioni)

3	Effettua richiesta POST all'URL /prenotazione/salva	Redirect alla route 'reservations.index' (/prenotazioni)	As Expected	Pass
<b>Postcondizioni:</b>				
Lo stato del richiamo è "confirmed"				
La data della prenotazione è quella odierna + 1 giorno alle 12:00				
Lo stock della prenotazione è cambiato ed è uno stock riferito al vaccino scelto per il cambio				
Lo stock che era stato inizialmente assegnato è stato incrementato di 1				
Lo stock assegnato è stato decrementato di 1				
E' stata creata una nuova prenotazione in data corrispondente a quella attuale +100 giorni alle 12:00				
Lo stock del vaccino assegnato al richiamo è stato decrementato di uno				

Test Case ID	testAcceptReservationAndChangeVaccine					
Test Case Description	Test per verificare che un responsabile può accettare una prenotazione cambiando il vaccino scelto dal sistema					
Precondizioni			Step #	Test Data		
			1	state = cancelled		
			1	reservation_id = ID della prenotazione creata dal seeder		
Step #	Step Details	Output Atteso		Actual Results		
1	Effettua richiesta PUT all'URL /prenotazione/reservation_id/update	Redirect alla route 'reservations.index' (/prenotazioni)		As Expected		
<b>Postcondizioni:</b>						
Lo stato della prenotazione è "cancelled"						
Lo stock che era stato inizialmente assegnato è stato incrementato di 1						



## Conclusioni

La separazione tra test di unità e di integrazione, aumenta la Fault Localization della test suite, infatti al fallimento di un test, è più semplice individuare la porzione di software responsabile del problema. Ad esempio se fallisce il test d'integrazione al controller ma gli altri passano, è probabile che il problema si trovi all'interno del controller.

### 2.4.4 Test Creazione Prenotazione da Parte del Paziente

Il paziente crea la propria prenotazione tramite una chiamata API POST effettuata al metodo reservation del controller app\http\Controllers\AndroidApiController.php. In teoria tale chiamata è effettuata dall'applicazione Android allegata all'elaborato, nella pratica potrebbe essere utilizzata per effettuare l'operazione da qualsiasi device. Tale test simula la chiamata da parte del paziente all'endpoint registrato, al fine di verificare che la prenotazione venga creata correttamente.

Anche questo è da considerare un test di integrazione, in quanto utilizza le classi precedentemente testate tramite unit test.

Il controller utilizza il metodo createAndStockDecrement di ReservationRepository che è stato già testato nei test di integrazione precedentemente descritti.

Classe: tests/Feature/Blackbox/PatientCreatesReservationTest

Precondizioni:

- E' presente una sola struttura sanitaria nel database
- Sono presenti 4 vaccini
- Sono presenti 4 lotti (1 per ogni vaccino)
- E' presente uno stock per ogni lotto
- E' presente un solo responsabile
- E' presente un solo paziente
- Il paziente possiede un token valido per effettuare chiamate API



<b>Test Case ID</b>		testPatientCreatesReservation					
<b>Test Case Description</b>		Test per verificare che un responsabile può accettare una prenotazione in stato di attesa					
Step #	Test Data	Step #	Test Data				
1	date = data odierna + 5 giorni	1	header[0] = ['Accept' => 'application/json']				
1	patient_id = ID del paziente che possiede il token	1	header[1] = ['Authorization' => token generato]				
1	structure_id = ID dell'unica struttura presente nel DB						
Step #	Step Details	Output Atteso		Actual Results	Result		
1	Chiamata API POST all'URL /api/reservation	Stato 200 (OK)		As Expected	Pass		
<b>Postcondizioni:</b>							
E' presente una nuova prenotazione nel DB							
La nuova prenotazione ha data odierna + 5 giorni							
L'orario della prenotazione è 08:00							
La nuova prenotazione è assegnata al paziente scelto							
La nuova prenotazione è associata alla struttura							

Figura 2.4.7: Test Paziente Crea Prenotazione

## 2.4.5 Test Relativi alla Sicurezza

Un aspetto molto importante dell'applicazione è la sicurezza, in particolare è importante che utenti non autorizzati non riescano ad accedere risorse per le quali non possiedono i privilegi

### Test Sicurezza Paziente

Come accennato, il paziente crea la prenotazione tramite applicazione Android, questa a sua volta interagisce con il server tramite delle chiamate API. Sono stati aggiunti dei controlli di sicurezza sugli endpoint delle API per evitare che utenti non autorizzati accedessero alle risorse per le quali non posseggono i permessi, in particolare sono state imposte delle regole fondamentali:

1. Tutte le richieste devono avere nell'header un bearer token valido (es. non scaduto)
2. A tutte le chiamate legate al paziente deve essere specificato nel body l'indirizzo email del paziente che effettua la chiamata
3. Il token passato nell'header deve essere associato all'email specificata nel body

L'unico modo per un attaccante di bypassare il controllo, è conoscere email + bearer token della sessione di un paziente. I seguenti test verificano che i controlli funzionino correttamente e che solo i pazienti autorizzati siano in grado di accedere agli endpoint del server.

Classe: tests/Feature/Blackbox/Security/PatientSecurityTest

Precondizioni:

- E' presente una sola struttura sanitaria nel database
- Sono presenti 4 vaccini
- Sono presenti 4 lotti (1 per ogni vaccino)
- E' presente uno stock per ogni lotto
- E' presente un solo responsabile
- Esistono 2 pazienti
- Ciascun paziente possiede una prenotazione

Una qualsiasi chiamata API da parte di un paziente necessita di un token di autenticazione che viene assegnato dal server quando il paziente effettua il login, è necessario per effettuare qualsiasi operazione.

### **API GET get-last-reservation-by-patient-email**

Tale chiamata richiede l'email del paziente come parametro dell'url e restituisce l'ultima prenotazione creata. Considerando il token, possiamo suddividere gli input in classi di equivalenza:

- token = { assente, valido, non valido }
- email = { email del paziente che possiede il token, email non appartenente al paziente }

Possiamo quindi scrivere un totale di 6 combinazioni.



Test Case ID		testPatientWithValidTokenCannotGetReservationOfAnotherPatient		
Test Case Description		Test per verificare che un paziente che possiede un token valido non può ottenere la prenotazione di un altro paziente		
Step #	Test Data		Step #	Test Data
1	email = email del secondo paziente		2	header[0] = ['Accept' => 'application/json']
2	email = email del secondo paziente		2	header[1] = ['Authorization' => token generato]
Step #	Step Details		Output Atteso	Actual Results
1	Genera token per il primo paziente		token generato = Token d'Autenticazione Valido	As Expected Pass
2	Chiamata API GET all'URL /api/get-last-reservation-by-patient: email/{email}		Stato 401 (UNAUTHORIZED)	As Expected Pass

Test Case ID		testPatientWithInvalidTokenCannotGetReservationOfAnotherPatient		
Test Case Description		Un paziente che possiede un token non valido non può ottenere la prenotazione di un altro paziente		
Step #	Test Data		Step #	Test Data
1	email = email del secondo paziente		2	header[0] = ['Accept' => 'application/json']
			2	header[1] = ['Authorization' => invalid_token]
Step #	Step Details		Output Atteso	Actual Results
2	Chiamata API GET all'URL /api/get-last-reservation-by-patient: email/{email}		Stato 401 (UNAUTHORIZED)	As Expected Pass

Test Case ID		testPatientWithoutTokenCannotGetReservationOfAnotherPatient		
Test Case Description		Un paziente senza un token non valido non può ottenere la prenotazione di un altro paziente		
Step #	Test Data		Step #	Test Data
1	email = email del secondo paziente		2	header[0] = ['Accept' => 'application/json']
			2	header[1] = ['Authorization' => invalid_token]
Step #	Step Details		Output Atteso	Actual Results
2	Chiamata API GET all'URL /api/get-last-reservation-by-patient: email/{email}		Stato 401 (UNAUTHORIZED)	As Expected Pass

Test Case ID		testPatientWithInvalidTokenCannotGetHisReservation		
Test Case Description		Un paziente senza un token non valido non può ottenere la prenotazione di un altro paziente		
Step #	Test Data		Step #	Test Data
1	email = email del secondo paziente		2	header[0] = ['Accept' => 'application/json']
			2	header[1] = ['Authorization' => invalid_token]
Step #	Step Details		Output Atteso	Actual Results
2	Chiamata API GET all'URL /api/get-last-reservation-by-patient: email/{email}		Stato 401 (UNAUTHORIZED)	As Expected Pass

## 2.4.6 Test di Robustezza sulla Chiamata API della Prenotazione

E' stato scritto un test ad hoc per testare la robustezza della chiamata API per la creazione di una prenotazione da parte del paziente, al fine di verificare che i controlli sulle validation siano funzionanti e non consentano di effettuare operazioni sul DB. In particolare, la chiamata prevede 3 campi:

- patient\_id: è obbligatorio, deve essere un intero > 1 e appartenente ad un paziente esistente 
- structure\_id: è obbligatorio, deve essere un intero > 1 e appartenere ad una struttura esistente
- date: è obbligatorio, deve essere una data valida in formato Y-m-d superiore a quella odierna

Sono state quindi definite le seguenti classi di equivalenza:

- patient\_id => { null, stringa non numerica, intero < 1, intero > 0, ID di un record non presente nel database (valido) }
- structure\_id => { null, stringa non numerica, intero < 1, intero > 0, ID di un record non presente nel database (valido) }
- date => { null, data in formato Y-m-d successiva alla data odierna (valido), data valida ma antecedente alla data odierna, data valida oggi, stringa non corrispondente ad una data, data con mese non valido, data con giorno non valido, data con anno non valido, data in un formato di verso, es. d/m/Y }

Sono stati scelti i seguenti valori da provare:

- patient\_id => { null, "wrong ID", 0, 1, 100 }
- structure\_id => { null, "wrong ID", 0, 1, 100 }
- date => { null, today + 5 days in format Y-m-d, today - 5 days in format Y-m-d, today in format Y-m-d, "wrong date", "2021-22-01", "2021-11-32", "XXXX-11-32", today + 5 in format d/m/Y }

Dal momento che input e CdE non sono in numero particolarmente elevato, è stato scelto il metodo delle classi adiacenti, impostando come input base quello con il set di valori validi, procedendo a far variare una classe d'equivalenza alla volta, verificando che in tutti i casi la request desse come risposta codice d'errore 500.

Classe: tests/Feature/Blackbox/RobustnessCreateReservationTest

Precondizioni:

- E' presente una sola struttura sanitaria nel database
- Sono presenti 4 vaccini
- Sono presenti 4 lotti (1 per ogni vaccino)
- E' presente uno stock per ogni lotto
- E' presente un solo responsabile
- E' presente un solo paziente

Test:

- ID: testPatientCannotCreateReservation
- Precondizioni: il paziente possiede un token valido
- Input:
  - Bearer token
  - Data provider definito in tests/Feature/Blackbox/RobustnessCreateReservationTest
- Postcondizioni: nessuna prenotazione è stata registrata
- Output Atteso: 500 -> server internal error per ogni chiamata
- Output Ottenuto: eccezioni non gestite le #4, #5, #6, #7, 200 (ok) per #9 e 500 (server internal error) per le altre
- Risultato: Fallito



E' stato provveduto a correggere le validation degli input nel controller che gestisce la chiamata API e, rilanciando il test, si è ottenuto il risultato aspettato per tutti gli input.

## 2.5 Testing Strutturale Whitebox

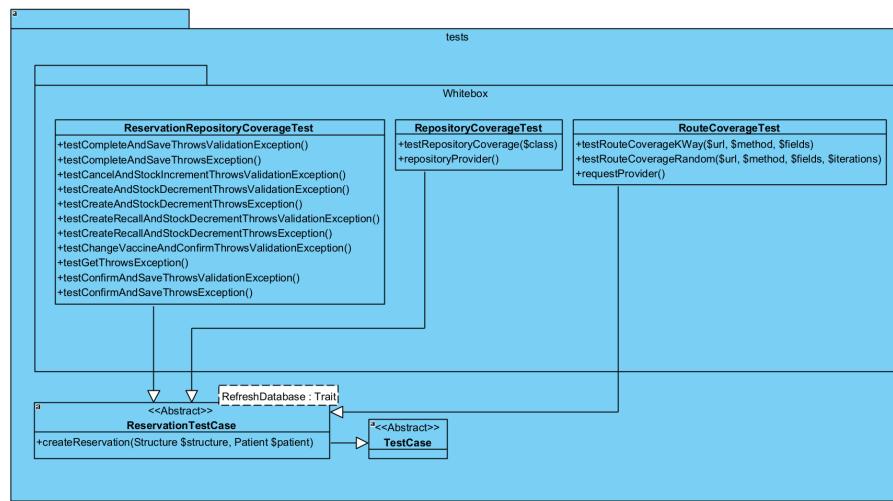


Figura 2.5.1: Package Test Whitebox

Fin ora abbiamo scritto test ragionando esclusivamente in termini di input inseriti - output aspettato, senza prendere minimamente in considerazione quali porzioni di codice andassimo a coprire. Ci chiediamo, adesso, quanto codice abbiamo coperto tramite i test scritti. PHPUnit supporta la code coverage<sup>9</sup>, tuttavia richiede XDebug.

### 2.5.1 Criterio di Terminazione

Quando terminare il testing? Poniamo come metrica la code coverage, in particolare, in base alle fonti consultate<sup>10</sup>, una buona percentuale di coverage risulta essere tra il 70 e l'80%, pertanto dunque il nostro obiettivo sarà quello di raggiungere tale risultato.



<sup>9</sup>[Code Coverage in PHPUnit](#)

<sup>10</sup>Atlassian, Bullseye

## 2.5.2 XDebug



Figura 2.5.2: XDebug Logo

E' un'estensione aggiuntiva per PHP che fornisce le seguenti funzionalità:

- Monitoraggio memoria allocata
- Possibilità di mettere dei breakpoint nel codice
- Monitoraggio di tutte le variabili allocate sul server e il relativo valore
- Analisi di code coverage tramite instrumentation<sup>11</sup>

Per aggiungere XDebug, è stato necessario (se è stata già copiata la cartella xampp allegata sovrascrivendo php.ini, è possibile saltare questo passaggio):

1. Scaricare ed inserire l'estensione manualmente (è possibile trovarla nella cartella allegata xampp/php/ext), spostando il file php\_xdebug-2.9.7-7.4-vc15-x86\_64.dll in c:/xampp/php/ext/
2. Editare il file php.ini nella cartella xampp/php/php.ini e aggiungere le seguenti righe alla fine del file (è possibile trovare la versione già modificata del file, ma con le righe di xdebug com-

<sup>11</sup>Fonte: [Documentazione XDebug](#)

mentate, nella cartella allegata in xampp/php):

```
[XDebug]
zend_extension = "c:\xampp\php\ext\php_xdebug-2.9.7-7.4-vc15-x86_64.dll"
xdebug.remote_autostart = 1
xdebug.profiler_append = 0
xdebug.profiler_enable = 0
xdebug.profiler_enable_trigger = 0
xdebug.profiler_output_dir = "c:\xampp\tmp"
xdebug.remote_enable = 1
xdebug.remote_handler = "dbgp"
xdebug.remote_host = "127.0.0.1"
xdebug.remote_log = "c:\xampp\tmp\xdebug.txt"
xdebug.remote_port = 9000
xdebug.trace_output_dir = "c:\xampp\tmp"
xdebug.remote_cookie_expire_time = 36000
```

### 3. Lanciare il file run\_tests.bat

Una volta lanciati i test, i report sulla coverage verranno generati in formato html nella cartella tests/-Report. Proviamo ad esaminare la code coverage ottenuta con i test fin ora:

	Code Coverage							
	Lines		Functions and Methods			Classes and Traits		
Total		39.81%	559 / 1404		38.10%	80 / 210		26.47% 18 / 68
Actions		0.00%	0 / 52		0.00%	0 / 7		0.00% 0 / 6
Console		75.00%	3 / 4		50.00%	1 / 2		0.00% 0 / 1
Exceptions		52.63%	30 / 57		25.00%	1 / 4		0.00% 0 / 1
Helper		0.00%	0 / 117		0.00%	0 / 17		0.00% 0 / 6
Http		31.20%	107 / 343		22.50%	9 / 40		7.69% 1 / 13
Models		37.37%	71 / 190		51.11%	23 / 45		33.33% 3 / 9
Observers		100.00%	8 / 8		100.00%	3 / 3		100.00% 1 / 1
Policies		37.50%	6 / 16		0.00%	0 / 6		0.00% 0 / 1
Providers		86.00%	43 / 50		75.00%	9 / 12		57.14% 4 / 7
Repositories		44.96%	205 / 456		33.33%	17 / 51		0.00% 0 / 9
Rules		76.00%	19 / 25		50.00%	3 / 6		0.00% 0 / 2
Validators		77.91%	67 / 86		82.35%	14 / 17		75.00% 9 / 12

Figura 2.5.3: Code Coverage

Come si può osservare, è attiva la coverage delle linee di codice, delle funzioni e delle classi. Se si vuole anche la coverage dei path e dei branch, bisogna specificare nel tag <coverage> in phpunit.xml (dovrebbe già contenere questa configurazione):

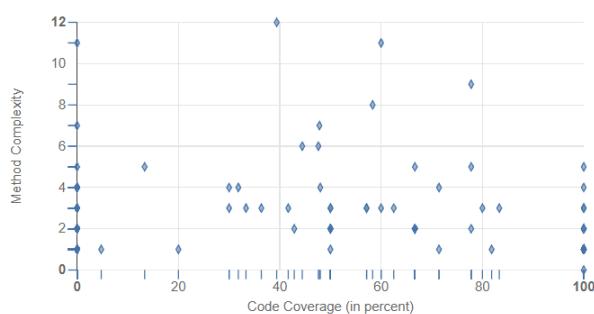
```
<coverage processUncoveredFiles="true" pathCoverage="true">
```

	Code Coverage														
	Lines			Branches			Paths			Functions and Methods			Classes and Traits		
Total	39.63%	596 / 1504		38.26%	251 / 656		12.51%	141 / 1127		37.56%	83 / 221		23.61%	17 / 72	
Actions	0.00%	0 / 52		0.00%	0 / 17		0.00%	0 / 15		0.00%	0 / 7		0.00%	0 / 6	
Console	75.00%	3 / 4		50.00%	1 / 2		50.00%	1 / 2		50.00%	1 / 2		0.00%	0 / 1	
Exceptions	43.86%	25 / 57		37.25%	19 / 51		3.68%	6 / 163		25.00%	1 / 4		0.00%	0 / 1	
Helper	0.00%	0 / 154		0.00%	0 / 54		0.00%	0 / 36		0.00%	0 / 19		0.00%	0 / 7	
Http	27.04%	106 / 392		32.09%	43 / 134		18.63%	19 / 102		15.56%	7 / 45		0.00%	0 / 14	
Models	39.58%	76 / 192		50.45%	56 / 111		4.93%	30 / 608		48.89%	22 / 45		22.22%	2 / 9	
Observers	100.00%	8 / 8		100.00%	3 / 3		100.00%	3 / 3		100.00%	3 / 3		100.00%	1 / 1	
Policies	25.00%	4 / 16		25.00%	4 / 16		18.18%	2 / 11		0.00%	0 / 6		0.00%	0 / 1	
Providers	86.00%	43 / 50		91.67%	11 / 12		91.67%	11 / 12		91.67%	11 / 12		85.71%	6 / 7	
Repositories	53.51%	244 / 456		40.39%	82 / 203		35.38%	46 / 130		41.18%	21 / 51		0.00%	0 / 9	
Rules	84.00%	21 / 25		76.19%	16 / 21		45.00%	9 / 20		66.67%	4 / 6		0.00%	0 / 2	
Testing	0.00%	0 / 15		0.00%	0 / 12		0.00%	0 / 7		0.00%	0 / 4		0.00%	0 / 2	
Validators	79.52%	66 / 83		80.00%	16 / 20		77.78%	14 / 18		76.47%	13 / 17		66.67%	8 / 12	

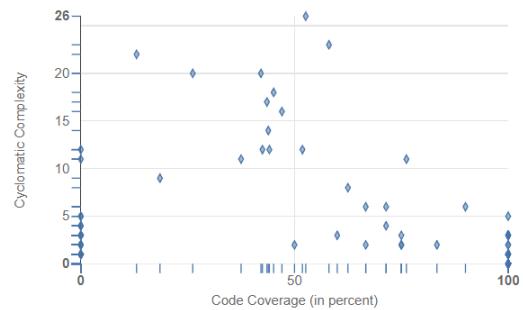
Figura 2.5.4: Code Coverage con Path e Branches

Il tool genera anche un report relativo alla complessità del progetto, ovvero riguardo le classi e i metodi che presentano complessità maggiore:

Complexity



Complexity



Project Risks

Method	CRAP
scopeFilter	132
saveOrCreate	56
customApiResponse	44
getBusyTimes	30
reservationsChanged	21
save	20

Project Risks

Class	CRAP
App\Models\Reservation	340
App\Http\Controllers\ReservationController	180
App\Helper\CreateTestData	156
App\Repositories\BatchRepository	132
App\Exceptions\Handler	97
App\Http\Controllers\AndroidApiController	97

Figura 2.5.5: Complexity

Tornando alla coverage, come prevedibile, la % di codice coperta dai test è molto bassa. Concentriamoci sulla coverage di quelle che sono le parti più critiche dell'applicazione:

- Repository della prenotazione (ReservationRepository)
- Validator della prenotazione (ReservationValidator)

Analizziamo i singoli report:

	Code Coverage								
	Classes and Traits			Functions and Methods			Lines		
	Total	0.00%	0 / 1	0.00%	30.00%	3 / 10	CRAP	58.06%	72 / 124
ReservationRepository	0.00%	0 / 1	0.00%	30.00%	3 / 10	62.01		58.06%	72 / 124
<u>construct</u>				100.00%	1 / 1	1		100.00%	3 / 3
<u>get</u>				0.00%	0 / 1	2.50		50.00%	2 / 4
<u>confirmAndSave</u>				0.00%	0 / 1	4.79		41.67%	5 / 12
<u>completeAndSave</u>				0.00%	0 / 1	5.32		36.36%	4 / 11
<u>cancelAndStockIncrement</u>				0.00%	0 / 1	4.12		50.00%	9 / 18
<u>createAndStockDecrement</u>				0.00%	0 / 1	4.12		50.00%	9 / 18
<u>createRecallAndStockDecrement</u>				0.00%	0 / 1	3.47		62.50%	15 / 24
<u>changeVaccineAndConfirm</u>				0.00%	0 / 1	3.71		57.14%	12 / 21
<u>assignAndSave</u>				100.00%	1 / 1	1		100.00%	10 / 10
<u>save</u>				100.00%	1 / 1	1		100.00%	3 / 3

Figura 2.5.6: Coverage Report ReservationRepository

	Code Coverage								
	Classes and Traits			Functions and Methods			Lines		
	Total	0.00%	0 / 1	0.00%	37.50%	3 / 8	CRAP	26.19%	33 / 126
ReservationController	0.00%	0 / 1	0.00%	37.50%	3 / 8	180.84		26.19%	33 / 126
<u>construct</u>				100.00%	1 / 1	1		100.00%	4 / 4
<u>index</u>				0.00%	0 / 1	6		0.00%	0 / 27
<u>reservationsChanged</u>				0.00%	0 / 1	21.27		13.33%	2 / 15
<u>create</u>				0.00%	0 / 1	2		0.00%	0 / 19
<u>store</u>				100.00%	1 / 1	1		100.00%	8 / 8
<u>getBusyTimes</u>				0.00%	0 / 1	30		0.00%	0 / 14
<u>edit</u>				0.00%	0 / 1	1.86		4.76%	1 / 21
<u>update</u>				100.00%	1 / 1	4		100.00%	18 / 18

Figura 2.5.7: Report ReservationController

	Code Coverage							
	Classes and Traits		Functions and Methods			Lines		
Total	0.00%	0 / 1	30.77%	4 / 13	CRAP:	42.15%	51 / 121	
AndroidApiController	0.00%	0 / 1	30.77%	4 / 13	97.45	42.15%	51 / 121	
__construct			100.00%	1 / 1	1	100.00%	5 / 5	
getStructuresByRegion			0.00%	0 / 1	1.12	50.00%	1 / 2	
loginPost			0.00%	0 / 1	9.07	31.82%	7 / 22	
getLastReservationByPatientEmail			0.00%	0 / 1	1.51	20.00%	3 / 15	
registerPost			0.00%	0 / 1	2	0.00%	0 / 21	
reservationPostValidation			100.00%	1 / 1	1	100.00%	9 / 9	
reservationPost			100.00%	1 / 1	1	100.00%	15 / 15	
clearTokens			0.00%	0 / 1	2	0.00%	0 / 2	
getTokenByPatientOrFail			0.00%	0 / 1	3.04	83.33%	5 / 6	
tokenValidation			0.00%	0 / 1	2.15	66.67%	2 / 3	
getPatientData			0.00%	0 / 1	2	0.00%	0 / 7	
getPatientLastReservation			0.00%	0 / 1	2	0.00%	0 / 10	
validateUser			100.00%	1 / 1	2	100.00%	4 / 4	

Figura 2.5.8: Coverage Report AndroidApiController -> reservationPost

### 2.5.3 Coverage di ReservationRepository

Il report ci dice nello specifico quali righe di codice non sono state coperte, ad esempio:

```
/*
 * @param Reservation $reservation
 * @param string $notes
 * @return Reservation
 * @throws MaxCapacityExceededException
 * @throws ValidationException
 */
public function confirmAndSave(Reservation $reservation, string $notes = ""): Reservation
{
    try {
        $reservation->notes = $notes;
        $this->reservationValidator->canConfirm($reservation);

        $reservation->state = Reservation::CONFIRMED_STATE;
        return $this->assignAndSave(
            Reservation::whereId($reservation->id)->firstOrFail(),
            $reservation
        );
    } catch (ValidationException $validationException) {
        Log::error("Reservation confirmAndSave Validation:\n" . $validationException->getMessage());
        Log::error(print_r($validationException->errors(), true));
        throw $validationException;
    } catch (Exception $exception) {
        Log::error("Reservation confirmAndSave:\n" . $exception->getMessage());
        throw $exception;
    }
}
```

Figura 2.5.9: Coverage Report ReservationRepository nel dettaglio

Ragionando sulla struttura del codice, scriviamo dei test whitebox che vadano a coprire le parti scoperte.

Classe: tests/Feature/Whitebox/ReservationRepositoryCoverageTest

Precondizioni:

- E' presente una sola struttura sanitaria nel database
- Sono presenti 4 vaccini
- Sono presenti 4 lotti (1 per ogni vaccino)
- E' presente uno stock per ogni lotto
- E' presente un solo responsabile
- E' presente un solo paziente

- E' presente una sola prenotazione nel database

### Coverage Funzione completeAndSave

Di seguito è riportato il CFG<sup>12</sup> della funzione con la coverage segnata in verde (coperto) e in rosso (non coperto).

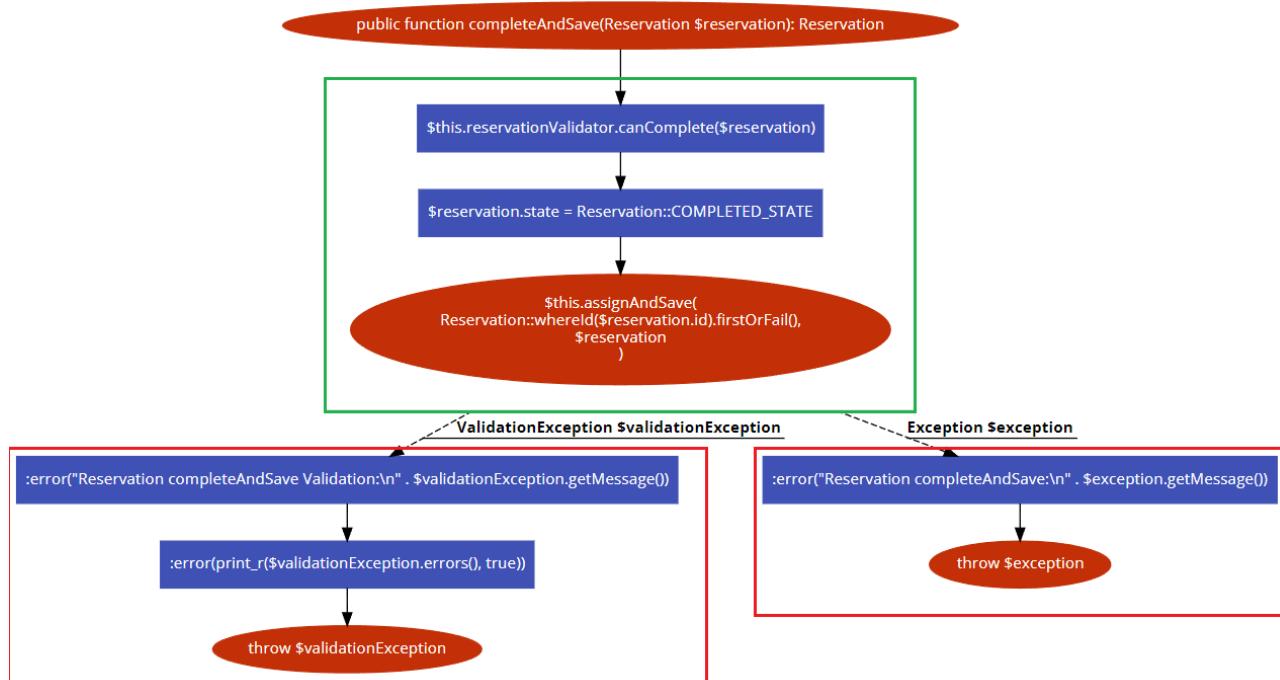


Figura 2.5.10: CFG completeAndSave

Abbiamo un cammino LI coperto e 2 da coprire.

<sup>12</sup>E' stato utilizzato [code2flow](#) per generare il CFG

<b>Test Case ID</b>		testCompleteAndSaveThrowsValidationException		
<b>Test Case Description</b>		Test per coprire il catch della ValidationException del metodo completeAndSave che segna la prenotazione come completa. Settando lo stato della prenotazione a 'NOT_VALID_STATE', la funzione di validazione, che effettua i controlli sui campi della prenotazione prima di salvare il record nel DB, individua l'errore e lancia un'eccezione di validazione (ValidationException)		
<b>Step #</b>	<b>Test Data</b>	<b>Precondizioni</b>		
1	reservation = prenotazione esistente	Esiste una prenotazione in stato "pending"		
1	state = 'NOT_VALID_STATE'			
<b>Step #</b>	<b>Step Details</b>	<b>Output Atteso</b>	<b>Actual Results</b>	<b>Result</b>
1	Chiamo la funzione 'completeAndSave' del repository	Viene lanciata l'eccezione 'ValidationException'	As Expected	Pass
<b>Postcondizioni</b>				
La prenotazione non è stata completata				



<b>Test Case ID</b>		testCompleteAndSaveThrowsException		
<b>Test Case Description</b>		Test per coprire il catch dell'eccezione generica Exception del metodo completeAndSave che segna la prenotazione come completa. Settando la capacità giornaliera della struttura a 0 (numero di prenotazioni che può accettare/gestire in un singolo giorno), quando si va a salvare la prenotazione, la funzione che assegna l'orario della prenotazione lancia eccezione.		
<b>Step #</b>	<b>Test Data</b>	<b>Step #</b>	<b>Test Data</b>	
1	reservation = prenotazione esistente	2	capacity = 0	
2	structure = struttura alla quale è assegnata la prenotazione	3	reservation = prenotazione esistente	
<b>Precondizioni</b>				
Esiste una prenotazione in stato "pending"				
<b>Step #</b>	<b>Step Details</b>	<b>Output Atteso</b>	<b>Actual Results</b>	<b>Result</b>
1	Chiamo la funzione 'confirmAndSave' del repository	Reservation Confermata	As Expected	Pass
2	Setto a 0 la capacità giornaliera della struttura e salvo	True	As Expected	Pass
3	Chiamo la funzione 'completeAndSave' del repository	Viene lanciata l'eccezione generica 'Exception'	As Expected	Pass
<b>Postcondizioni</b>				
La prenotazione non è stata completata				

## Coverage Funzione cancelAndStockIncrement

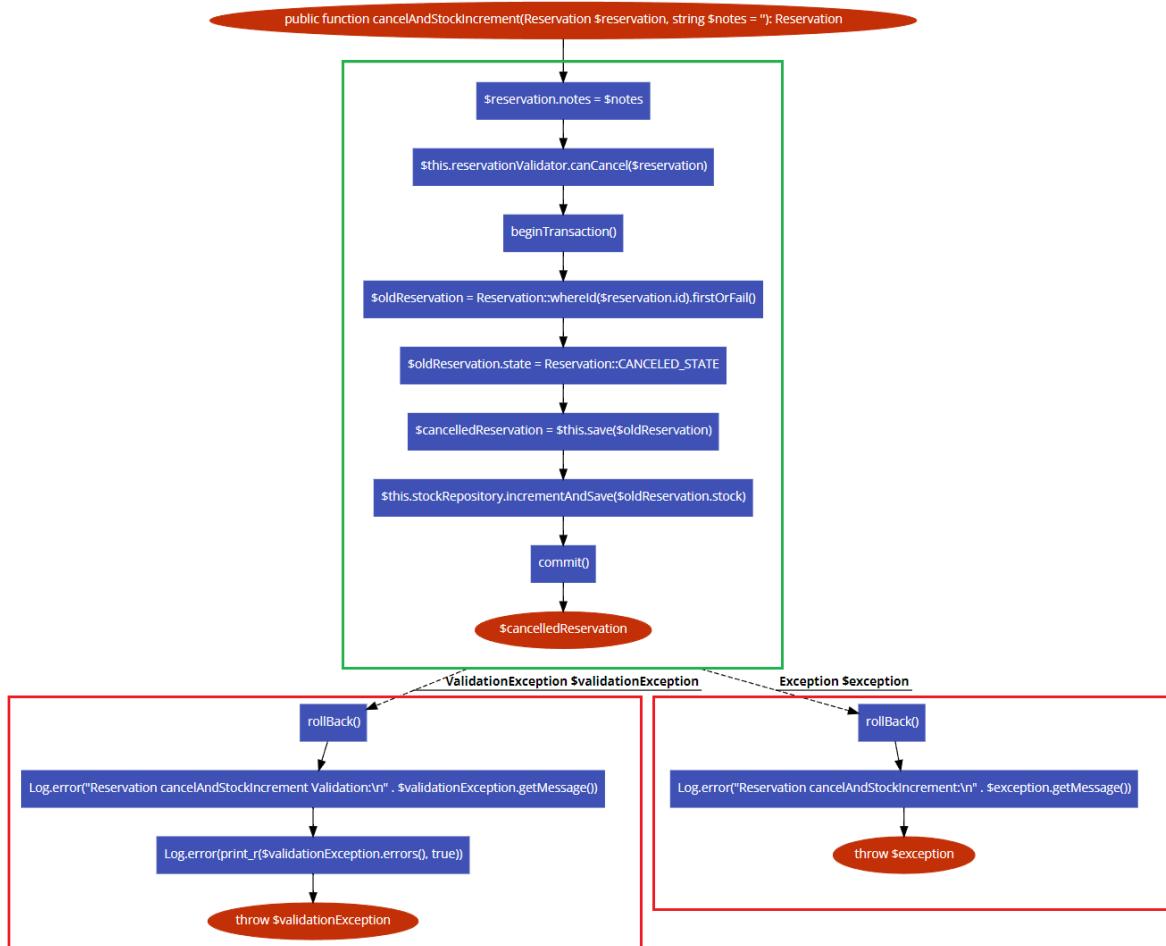


Figura 2.5.11: CFG cancelAndStockIncrement

Abbiamo un cammino LI coperto e 2 da coprire.

<b>Test Case ID</b>		testCancelAndStockIncrementThrowsValidationException		
<b>Test Case Description</b>		Test per coprire il catch della ValidationException del metodo cancelAndStockIncrement che annulla la prenotazione e incrementa lo stock che era stato assegnato. Quando viene passata la prenotazione, viene settato il suo stato a 'NOT_VALID_STATE', che chiaramente non esiste e non viene supportato dal campo enum impostato nella migration del database. Tuttavia, tale controllo è effettuato prima di salvare la prenotazione, all'interno della funzione di validazione, inerentemente viene sollevata l'eccezione 'ValidationException'		
<b>Step #</b>	<b>Test Data</b>	<b>Precondizioni</b>		
1	reservation = prenotazione esistente	Esiste una prenotazione in stato "pending"		
1	state = 'NOT_VALID_STATE'			
<b>Step #</b>	<b>Step Details</b>	<b>Output Atteso</b>	<b>Actual Results</b>	<b>Result</b>
1	Chiamo la funzione 'cancelAndStockIncrement' del repository	Viene lanciata l'eccezione 'ValidationException'	As Expected	Pass
<b>Postcondizioni</b>				
La prenotazione non è stata cancellata				
Lo stock non è stato incrementato				

## Coverage Funzione createAndStockDecrement

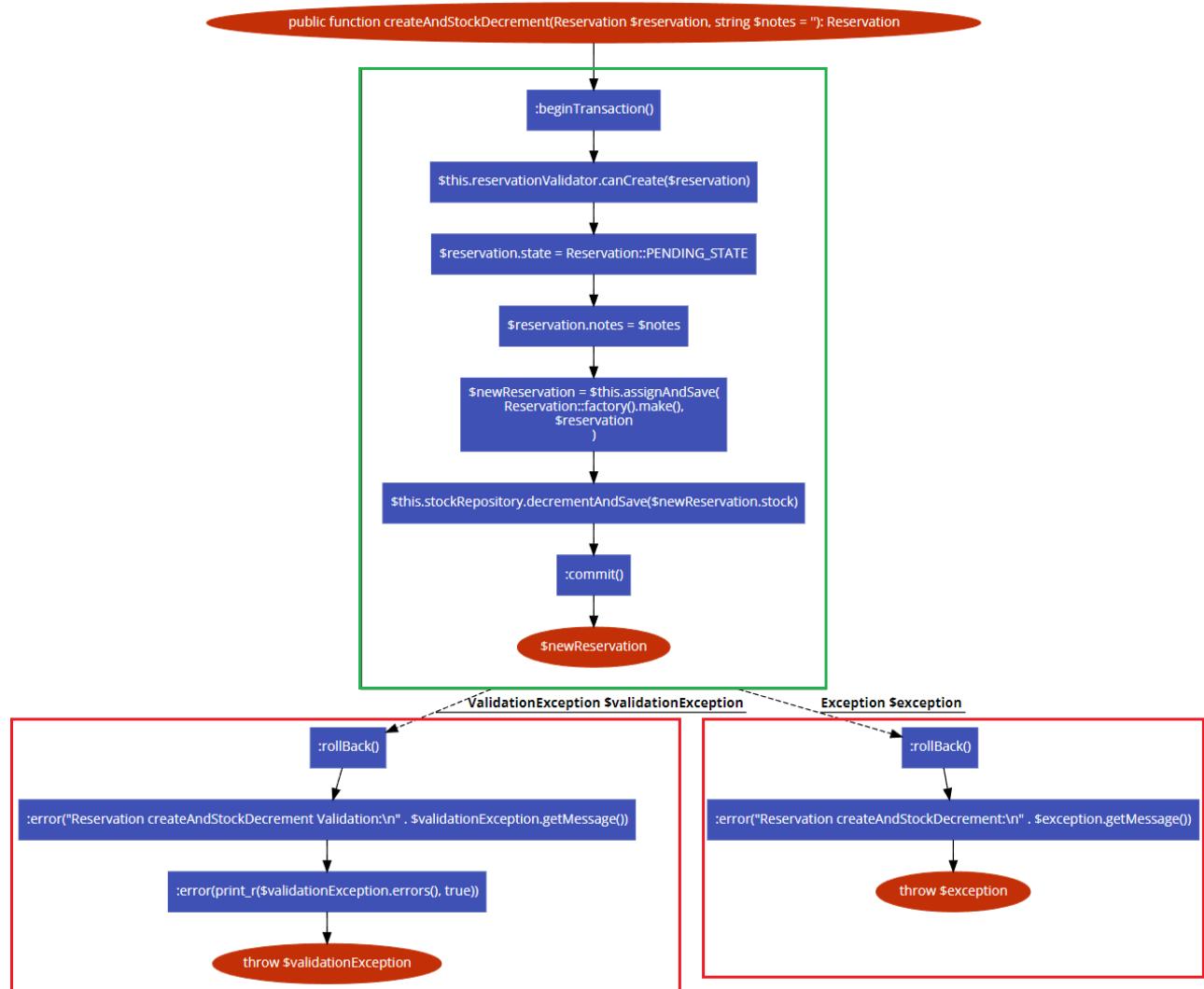


Figura 2.5.12: CFG createAndStockDecrement

Abbiamo un cammino LI coperto e 2 da coprire.

<b>Test Case ID</b>		testCreateAndStockDecrementThrowsValidationException		
<b>Test Case Description</b>		Test per coprire il catch della ValidationException del metodo createAndStockDecrement che crea la prenotazione e decrementa lo stock assegnato. Non esistendo il paziente di ID 100, il validator rileva questo errore e lancia eccezione.		
<b>Step #</b>	<b>Test Data</b>		<b>Precondizioni</b>	
1	patient_id = 100		Non esiste alcuna prenotazione	
1	stock_id = 1		Non esiste il paziente con ID pari a 100	
1	date = data odierna in formato Y-m-d			
<b>Step #</b>	<b>Step Details</b>		<b>Output Atteso</b>	<b>Actual Results</b>
1	Chiamo la funzione 'createAndStockDecrement' del repository		Viene lanciata l'eccezione	As Expected
<b>Postcondizioni</b>				
La prenotazione non viene creata				
Lo stock non viene decrementato				

<b>Test Case ID</b>		testCreateAndStockDecrementThrowsException		
<b>Test Case Description</b>		Test per coprire il catch della ValidationException del metodo createAndStockDecrement che crea la prenotazione e decrementa lo stock assegnato. A provocare l'eccezione generica è l'ID dello stock che non è interno, ma viene passato come stringa		
<b>Step #</b>	<b>Test Data</b>		<b>Precondizioni</b>	
1	patient_id = 100		Non esiste alcuna prenotazione	
1	stock_id = '100'		Non esiste lo stock con ID pari a 100	
1	date = data odierna in formato Y-m-d		Non esiste il paziente con ID pari a 100	
<b>Step #</b>	<b>Step Details</b>		<b>Output Atteso</b>	<b>Actual Results</b>
1	Chiamo la funzione 'createAndStockDecrement' del repository		Viene lanciata l'eccezione 'Exception'	As Expected
<b>Postcondizioni</b>				
La prenotazione non viene creata				
Lo stock non viene decrementato				

## Coverage Funzione createRecallAndStockDecrement

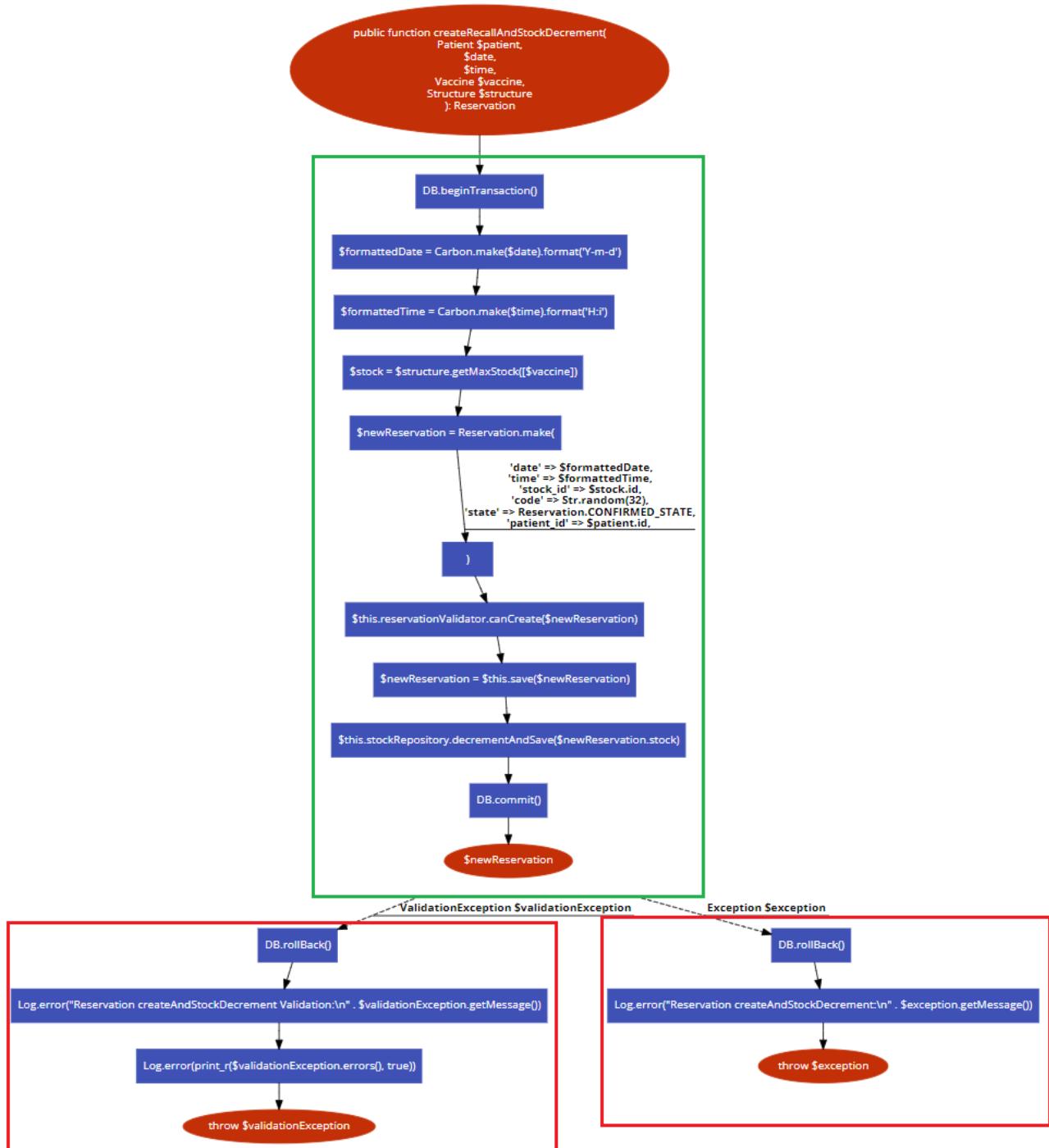


Figura 2.5.13: CFG createRecallAndStockDecrement

Abbiamo un cammino LI coperto e 2 da coprire.

<b>Test Case ID</b>		testCreateRecallAndStockDecrementThrowsValidationException		
<b>Test Case Description</b>		Test per coprire il catch della ValidationException del metodo createRecallAndStockDecrement che crea un richiamo e decrementa lo stock assegnato		
<b>Step #</b>	<b>Test Data</b>	<b>Precondizioni</b>		
1	date = data odierna meno 2 giorni	La quantità di tutti gli stock è settata a 10		
1	time = '12:00'	Esiste una prenotazione in stato 'completata'		
1	vaccine = 'Pfizer'			
1	structure = unica struttura presente nel database			
<b>Step #</b>	<b>Step Details</b>	<b>Output Atteso</b>	<b>Actual Results</b>	<b>Result</b>
1	Chiamo la funzione 'completeAndSave' del repository	Viene lanciata l'eccezione 'ValidationException'	As Expected	Pass
<b>Postcondizioni</b>				
Il richiamo non è stato creato				
Lo stock non è stato decrementato				

<b>Test Case ID</b>		testCreateRecallAndStockDecrementThrowsException		
<b>Test Case Description</b>		Test per coprire il catch dell'eccezione generica Exception del metodo createRecallAndStockDecrement che segna la prenotazione come completa		
<b>Step #</b>	<b>Test Data</b>	<b>Precondizioni</b>		
1	date = data odierna meno 2 giorni	La struttura non ha dosi sufficienti per il vaccino 'Pfizer'		
1	time = '12:00'	Esiste una prenotazione in stato 'completata'		
1	vaccine = 'Pfizer'			
1	structure = unica struttura presente nel database			
<b>Step #</b>	<b>Step Details</b>	<b>Output Atteso</b>	<b>Actual Results</b>	<b>Result</b>
1	Chiamo la funzione 'completeAndSave' del repository	Viene lanciata l'eccezione 'Exception'	As Expected	Pass
<b>Postcondizioni</b>				
Il richiamo non è stato creato				
Lo stock non è stato decrementato				

## Coverage Funzione changeVaccineAndConfirm

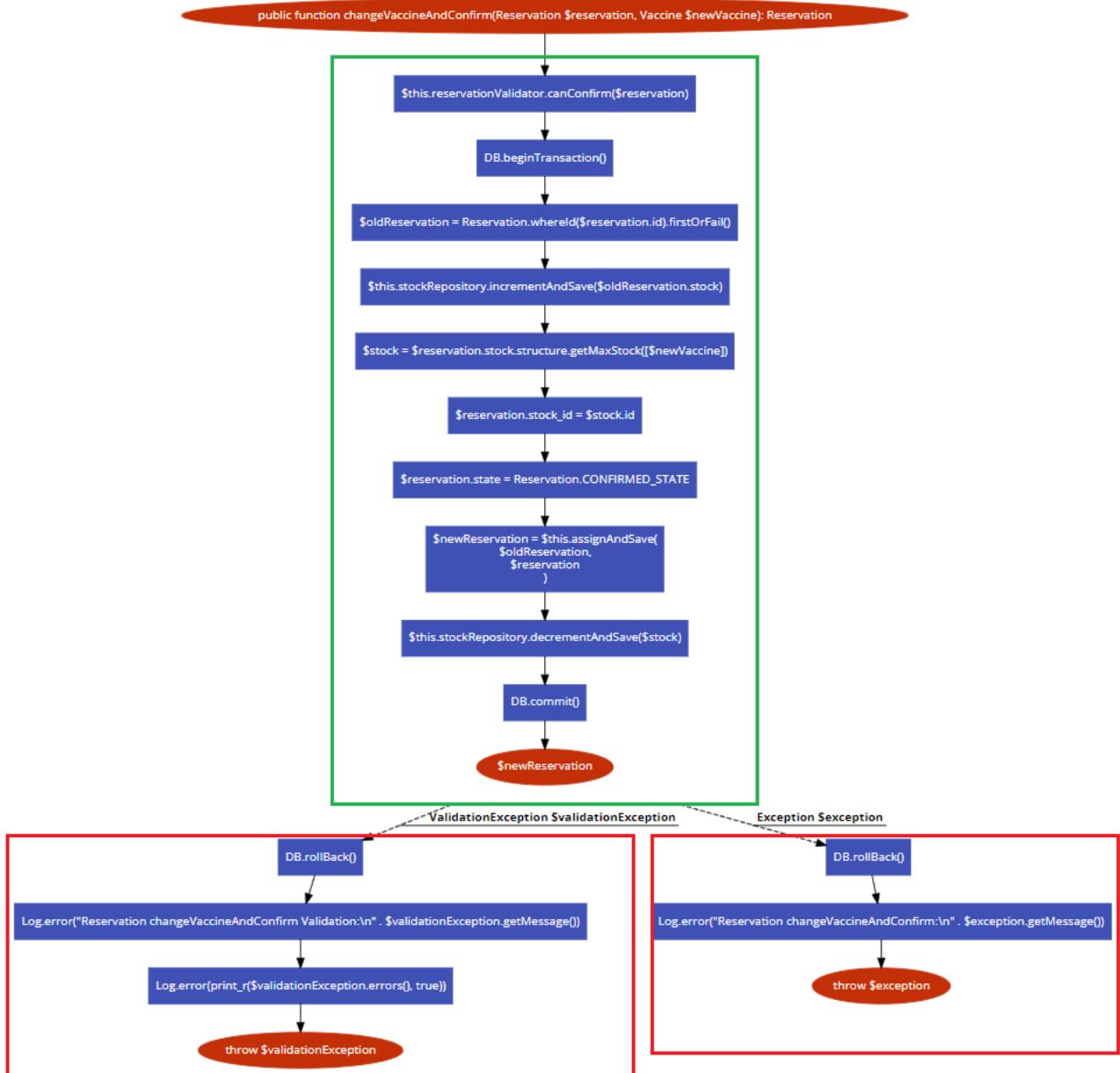


Figura 2.5.14: CFG changeVaccineAndConfirm

Abbiamo un cammino LI coperto e 2 da coprire.

<b>Test Case ID</b>		testChangeVaccineAndConfirmThrowsValidationException		
<b>Test Case Description</b>		Test per coprire il catch della ValidationException del metodo changeVaccineAndConfirm che cambia il vaccino assegnato alla prenotazione e la conferma. La prenotazione è già confermata, per cui non può essere cambiato il vaccino. Tale controllo è effettuato all'interno della validation, ragion per cui viene lanciata l'eccezione ValidationException.		
<b>Step #</b>	<b>Test Data</b>	<b>Precondizioni</b>		
1	vaccine = 'Moderna'	Esiste una prenotazione in stato 'confermata'		
1	reservation = unica reservation presente			
<b>Step #</b>	<b>Step Details</b>	<b>Output Atteso</b>	<b>Actual Results</b>	<b>Result</b>
1	Chiamo la funzione 'completeAndSave' del repository	Viene lanciata l'eccezione 'ValidationException'	As Expected	Pass
<b>Postcondizioni</b>				
Il vaccino non è stato cambiato				
La prenotazione rimane confermata				

<b>Test Case ID</b>		testChangeVaccineAndConfirmThrowsValidationException		
<b>Test Case Description</b>		Test per coprire il catch dell'eccezione generica Exception del metodo changeVaccineAndConfirm che cambia il vaccino e segna la prenotazione come completa. La prenotazione non è completa, quindi il vaccino può essere cambiato, tuttavia la struttura non possiede una dose di Moderna. Tale controllo non è effettuato nella validation, pertanto lancia un'eccezione generica.		
<b>Step #</b>	<b>Test Data</b>	<b>Precondizioni</b>		
1	vaccine = 'Moderna'	La struttura non ha dosi per il vaccino 'Moderna'		
1	reservation = unica prenotazione esistente	Esiste una prenotazione in stato 'pending'		
<b>Step #</b>	<b>Step Details</b>	<b>Output Atteso</b>	<b>Actual Results</b>	<b>Result</b>
1	Chiamo la funzione 'completeAndSave' del repository	Viene lanciata l'eccezione 'Exception'	As Expected	Pass
<b>Postcondizioni</b>				
Il vaccino non è stato cambiato				
La prenotazione rimane in stato 'pending'				

## Coverage Funzione confirmAndSave

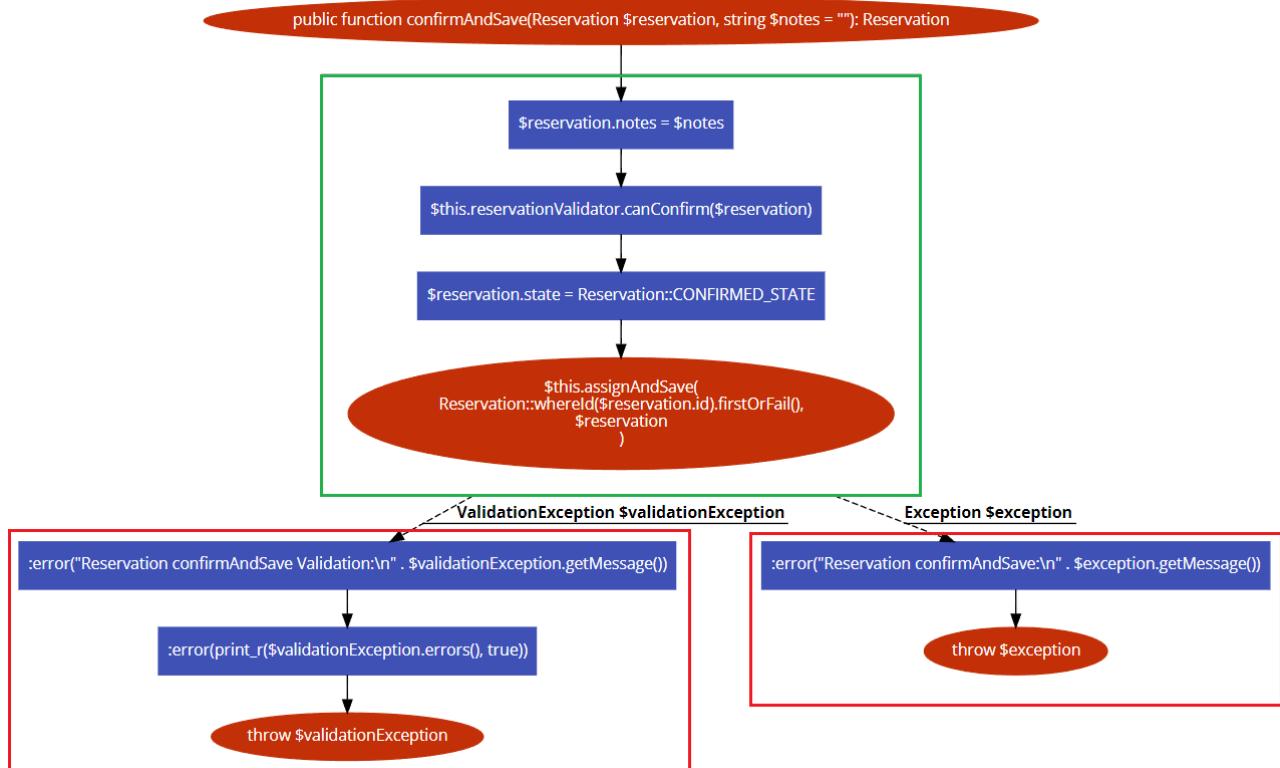


Figura 2.5.15: CFG confirmAndSave

Abbiamo un cammino LI coperto e 2 da coprire.

<b>Test Case ID</b>		testConfirmAndSaveThrowsValidationException		
<b>Test Case Description</b>		Test per coprire il catch della ValidationException del metodo confirmAndSave che conferma la prenotazione. Passando come note della prenotazione una stringa di 300 caratteri, viene sollevata eccezione nel validatore, dato che il massimo consentito è 255.		
<b>Step #</b>	<b>Test Data</b>	<b>Precondizioni</b>		
1	reservation = prenotazione esistente	Esiste una prenotazione in stato "pending"		
1	notes = stringa randomica di 300 caratteri			
<b>Step #</b>	<b>Step Details</b>	<b>Output Atteso</b>	<b>Actual Results</b>	<b>Result</b>
1	Chiamo la funzione 'confirmAndSave' del repository	Viene lanciata l'eccezione 'ValidationException'	As Expected	Pass
<b>Postcondizioni</b>				
La prenotazione non è stata confermata				

<b>Test Case ID</b>		testConfirmAndSaveThrowsException		
<b>Test Case Description</b>		Test per coprire il catch dell'eccezione Exception del metodo confirmAndSave che conferma la prenotazione. Settando uno stock_id di uno stock non esistente, viene sollevata tale Eccezione.		
<b>Step #</b>	<b>Test Data</b>	<b>Precondizioni</b>		
1	reservation = prenotazione esistente	Esiste una prenotazione in stato "pending"		
1	stock_id = 100	Non esiste uno stock con ID 100		
<b>Step #</b>	<b>Step Details</b>	<b>Output Atteso</b>	<b>Actual Results</b>	<b>Result</b>
1	Chiamo la funzione 'confirmAndSave' del repository	Viene lanciata l'eccezione 'Exception'	As Expected	Pass
<b>Postcondizioni</b>				
La prenotazione non è stata confermata				

## 2.5.4 Codice Coperto con i Test Whitebox

Sono stati rieseguiti tutti i test (comando php artisan test o i file batch run\_tests.bat e run\_tests\_parallel.bat) e generando un nuovo report, risulta la seguente copertura:

	Code Coverage									
	Classes and Traits			Functions and Methods				Lines		
Total	0.00%	0 / 1		70.00%	7 / 10	CRAP		91.13%	113 / 124	
ReservationRepository	0.00%	0 / 1		70.00%	7 / 10	23.37		91.13%	113 / 124	
__construct				100.00%	1 / 1	1		100.00%	3 / 3	
get				100.00%	1 / 1	2		100.00%	4 / 4	
confirmAndSave				100.00%	1 / 1	3		100.00%	12 / 12	
completeAndSave				0.00%	0 / 1	3.18		72.73%	8 / 11	
cancelAndStockIncrement				0.00%	0 / 1	3.10		77.78%	14 / 18	
createAndStockDecrement				100.00%	1 / 1	3		100.00%	18 / 18	
createRecallAndStockDecrement				100.00%	1 / 1	3		100.00%	24 / 24	
changeVaccineAndConfirm				0.00%	0 / 1	3.06		80.95%	17 / 21	
assignAndSave				100.00%	1 / 1	1		100.00%	10 / 10	
save				100.00%	1 / 1	1		100.00%	3 / 3	

Figura 2.5.16: Coverage Report Finale

	Code Coverage											
	Classes and Traits		Functions and Methods				Paths		Branches		Lines	
Total	0.00%	0 / 1		70.00%	7 / 10	CRAP		86.96%	20 / 23		83.78%	31 / 37
ReservationRepository	0.00%	0 / 1		70.00%	7 / 10	24.17		86.96%	20 / 23		83.78%	31 / 37
__construct				100.00%	1 / 1	1		100.00%	1 / 1		100.00%	1 / 1
get				100.00%	1 / 1	2		100.00%	2 / 2		100.00%	4 / 4
confirmAndSave				100.00%	1 / 1	3		100.00%	3 / 3		100.00%	12 / 12
completeAndSave				0.00%	0 / 1	3.33		66.67%	2 / 3		60.00%	3 / 5
cancelAndStockIncrement				0.00%	0 / 1	3.33		66.67%	2 / 3		60.00%	3 / 5
createAndStockDecrement				100.00%	1 / 1	3		100.00%	3 / 3		100.00%	5 / 5
createRecallAndStockDecrement				100.00%	1 / 1	3		100.00%	3 / 3		100.00%	5 / 5
changeVaccineAndConfirm				0.00%	0 / 1	3.33		66.67%	2 / 3		60.00%	3 / 5
assignAndSave				100.00%	1 / 1	1		100.00%	1 / 1		100.00%	1 / 1
save				100.00%	1 / 1	1		100.00%	1 / 1		100.00%	3 / 3

Figura 2.5.17: Coverage Report Finale con Branch e Path

Non si è riusciti a raggiungere il 100% di copertura a causa dell'impossibilità di trovare una combinazione di input che triggerebbe l'eccezione generica per alcuni metodi.

## 2.5.5 Code Coverage Rimanente

Sono state rimosse dalla coverage le classi del framework e/o non rilevanti, ad esempio quelle usate per generare dati casuali:

```

14      <coverage processUncoveredFiles="true" pathCoverage="true">
15          <include>
16              <directory suffix=".php">./app/Models</directory>
17              <directory suffix=".php">./app/Repositories</directory>
18              <directory suffix=".php">./app/Validators</directory>
19              <directory suffix=".php">./app/Policies</directory>
20              <directory suffix=".php">./app/Http/Controllers</directory>
21              <directory suffix=".php">./app/Http/Observers</directory>
22              <directory suffix=".php">./app/Http/Middleware</directory>
23              <file>./app/routes/web.php</file>
24              <directory suffix=".php">./database/migrations</directory>
25      </include>
26      <exclude>
27          <file>./app/Http/Controllers/Controller.php</file>
28          <file>./app/Http/Controllers/ResponsibleController.php</file>
29          <file>./app/Http/Controllers/TestApiController.php</file>
30          <file>./app/Http/Middleware/HandleInertiaRequests.php</file>
31          <file>./app/Http/Middleware/Authenticate.php</file>
32          <file>./app/Http/Middleware/EncryptCookies.php</file>
33          <file>./app/Http/Middleware/PreventRequestsDuringMaintenance.php</file>
34          <file>./app/Http/Middleware/TrustHosts.php</file>
35          <file>./app/Http/Middleware/TrimStrings.php</file>
36          <file>./app/Http/Middleware/TrustProxies.php</file>
37          <file>./app/Http/Middleware/VerifyCsrfToken.php</file>
38          <file>./app/Http/Models/Director.php</file>
39      </exclude>

```

Figura 2.5.18: Rimozione dalla Coverage delle Classi non usate

Dopo gli ultimi test, abbiamo raggiunto la seguente copertura sul totale dell'applicazione:

	Code Coverage											
	Lines		Branches		Paths		Functions and Methods		Classes and Traits			
Total	45.93%	480 / 1045	41.75%	177 / 424	16.34%	109 / 667	42.38%	64 / 151	25.00%	10 / 40		
Http	30.45%	95 / 312	34.74%	33 / 95	24.64%	17 / 69	21.21%	7 / 33	0.00%	0 / 9		
Models	39.89%	71 / 178	46.67%	42 / 90	6.83%	30 / 439	52.27%	23 / 44	22.22%	2 / 9		
Policies	25.00%	4 / 16	25.00%	4 / 16	18.18%	2 / 11	0.00%	0 / 6	0.00%	0 / 1		
Repositories	53.51%	244 / 456	40.39%	82 / 203	35.38%	46 / 130	41.18%	21 / 51	0.00%	0 / 9		
Validators	79.52%	66 / 83	80.00%	16 / 20	77.78%	14 / 18	76.47%	13 / 17	66.67%	8 / 12		

Figura 2.5.19: Copertura Raggiunta

Anche se vogliamo considerare solo righe di codice e branches, il 40-45% risulta insufficiente per porre termine al testing (ci eravamo prefissati di raggiungere almeno il 70%). Il nuovo obiettivo è quello di creare dei test meno specifici per le aree di codice non ancora coperte al fine di individuare eccezioni non gestite dal sistema.

## Coverage dei Repository

La gerarchia delle eccezioni è la seguente:

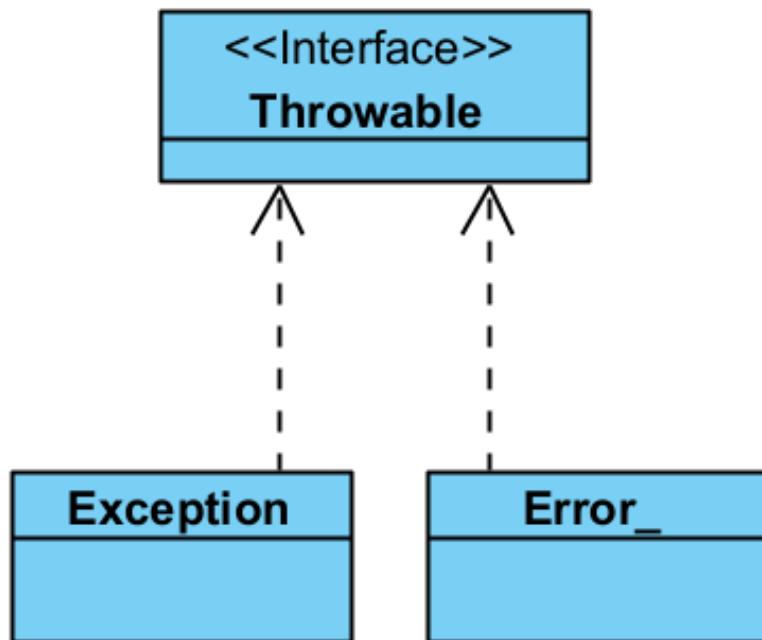


Figura 2.5.20: Gerarchia Eccezioni

Throwable è l'interfaccia che viene implementata sia da Exception che da Error. Entrambe vengono lanciate quando vi sono comportamenti inaspettati del software, ma mentre le Eccezioni possono essere previste in base al contesto (ad esempio se c'è una divisione, anche se non dovrebbe accadere, posso comunque prevedere che possa esserci come denominatore 0), gli Errori si verificano a causa di inaccortezze da parte del programmatore, ad esempio il caso in cui si abbia una variabile nulla e si provi a usare un metodo a partire da tale variabile, o se una funzione richiede valori interi e si prova a passare una stringa. Per tale ragione i repository gestiscono Exception, Throwable non deve mai verificarsi.

Scriviamo dunque dei test per i repository che vadano a coprire quanto più codice possibile, individuando il verificarsi di eventuali errori.

Classe: tests/Feature/Whitebox/RepositoryCoverageTest.php

Precondizioni:

- E' presente una sola struttura sanitaria nel database
- Sono presenti 4 vaccini
- Sono presenti 4 lotti (1 per ogni vaccino)
- E' presente uno stock per ogni lotto
- E' presente un solo responsabile
- E' presente un solo paziente

Metodo di test: testRepositoryCoverage

Descrizione: Questo test usa la reflection per esplorare la struttura del repository, individuarne le classi e generare dei dati di test da assegnare ai parametri. In particolare:

1. Con un approccio data driven viene passata la classe del repository che si deve testare
2. Il test usa la reflection per istanziare tale classe e individuare i metodi
3. Per ogni metodo vengono individuati tutti i parametri
4. Per ogni parametro viene generato un dizionario di valori da provare, in particolare se il tipo è una model, questa viene istanziata e il dizionario corrispondente conterrà la model vuota e la model generata tramite factory, la quale conterrà dei valori randomici definiti nelle factory
5. Viene chiamato il metodo più volte, facendo variare in maniera casuale i valori di input e facendo il catch sia di Exception che di Throwable. Se viene lanciato Exception continuiamo, mentre se viene lanciato Throwable significa che si è verificato un errore e il test fallisce



Input:

```
public function repositoryProvider(): array
{
    return [
        [UserRepository::class],
        [StockRepository::class],
        [VaccineRepository::class],
        [AccountRepository::class],
        [ResponsibleRepository::class],
        [StructureRepository::class],
        [PatientRepository::class],
        [BatchRepository::class],
    ];
}
```

Figura 2.5.21: Input testRepositoryCoverage

Output Atteso: Non si sono verificati errori

Output Ottentuto: Si è verificato un errore per il metodo decrementAndSave di StockRepository:

```
Tests\Feature\Whitebox\RepositoryCoverageTest > repository coverage
with data set #1    Lanciata eccezione non gestita con il seguente
messaggio: Call to a member function toArray() on int     e il
seguente body
```

```
Array      (      [ stock ] => -1      [ qty ] => -1      )
```

Risultato del Test: Fallito 

Analizziamo il repository:

```

71     public function decrementAndSave($stock, int $qty = 1): Stock
72     {
73         try {
74             $stockAttributes = $stock->toArray();
75             $stock->quantity -= $qty;
76
77             return $this->assignAndSave(
78                 Stock::whereId($stock->id)->first(),
79                 $stock
80             );
81         } catch (ValidationException $validationException) {
82             Log::error("Stock decrementAndSave Validation:\n" . $validationException->getMessage());
83             Log::error(print_r($validationException->errors(), true));
84             throw $validationException;
85         }
86     }

```

Figura 2.5.22: decrementAndSave di StockRepository

L'errore è provocato dalla combinazione di 2 fattori:

1. Il tipo di \$stock non è specificato, quindi possiamo passare qualsiasi cosa alla funzione
2. Chiamiamo un metodo dalla variabile (toArray()) assumendo che sia un'istanza della model Stock

Correggendo, il test è fallito nuovamente:

Lanciata eccezione non gestita con il seguente messaggio: Argument 1 passed to App\Repositories\StockRepository::assignAndSave() must be an instance of App\Models\Stock, null given, called in C:\laravel\università\software-testing-backend \app\Repositories\StockRepository.php on line 80 e il seguente body

```

Array  (
    [ stock ] => Array
        (
            [ quantity ] => -1
        )
)

```

```
[ qty ] => 1
)
```

Il problema, questa volta, è dovuto alla riga 78. `first()`, infatti, se non trova il record cercato ritorna null, ma la funzione che viene chiamata, ossia `assignAndSave`, non accetta valori nulli. Di seguito è riportata la funzione corretta:

```

71     public function decrementAndSave(Stock $stock, int $qty = 1): Stock
72     {
73         try {
74             $stock->quantity -= $qty;
75
76             return $this->assignAndSave(
77                 Stock::whereId($stock->id)->firstOrFail(),
78                 $stock
79             );
80         } catch (ValidationException $validationException) {
81             Log::error( message: "Stock decrementAndSave Validation:\n" . $validationException->getMessage());
82             Log::error(print_r($validationException->errors(), true));
83             throw $validationException;
84         }
85     }

```

Figura 2.5.23: decrementAndSave di StockRepository corretta

Analizziamo la coverage ottenuta aggiungendo questo test:

	Code Coverage											
	Lines		Branches		Paths		Functions and Methods		Classes and Traits			
Total	58.76%	614 / 1045	60.61%	257 / 424	22.19%	148 / 667	51.66%	78 / 151	27.50%	11 / 40		
Http	30.45%	95 / 312	34.74%	33 / 95	24.64%	17 / 69	21.21%	7 / 33	0.00%	0 / 9		
Models	39.89%	71 / 178	46.67%	42 / 90	6.83%	30 / 439	52.27%	23 / 44	22.22%	2 / 9		
Policies	25.00%	4 / 16	25.00%	4 / 16	18.18%	2 / 11	0.00%	0 / 6	0.00%	0 / 1		
Repositories	82.24%	375 / 456	79.31%	161 / 203	64.62%	84 / 130	66.67%	34 / 51	0.00%	0 / 9		
Validators	83.13%	69 / 83	85.00%	17 / 20	83.33%	15 / 18	82.35%	14 / 17	75.00%	9 / 12		

Figura 2.5.24: Coverage repositoryTest

## Coverage dei Controller

Classe: tests/Feature/Whitebox/RouteCoverageTest

Descrizione: In maniera simile a quanto fatto per i repository, è stato impostato un data provider dove ciascun record contiene:

- L'URL collegato al controller
- Il tipo di richiesta (GET, POST, PUT, etc...)
- Un array con i nomi dei parametri che vengono estratti dal body della richiesta
- Il numero di iterazioni

Dal momento che il data provider non consente di istanziare dati dinamici, è stato predisposto un metodo che dato il campo in input genera un dizionario di possibili valori da provare. Ad esempio se il campo è “date”, vengono generate delle date. Di base tutti i dizionari conterranno i seguenti valori:

- -1
- 1
- 0
- Stringa di 50 caratteri
- Stringa di 1000 caratteri
- Stringa vuota
- null

In base ai singoli campi:

- date: data attuale come oggetto Carbon, data attuale - 5 giorni come oggetto Carbon, data attuale + 5 giorni come oggetto Carbon, data attuale in formato Y-m-d, data attuale - 5 giorni in formato Y-m-d, data attuale + 5 giorni in formato Y-m-d

- time: 00:00, 12:00, 17:00, 25:00
- vaccine: pfizer, astrazeneca, moderna, johnson&johnson
- state: pending, accepted, cancelled, completed
- structure: nome dell'unica struttura memorizzata

A differenza dei repository, qui non è stato possibile utilizzare la reflection per 'indovinare' il tipo dei parametri, in quanto vengono scoperti a tempo di esecuzione interrogando la request.

I metodi di test sono 2:

1. `testRouteCoverageKWay`: con un approccio 3-way, facciamo variare 3 valori alla volta 
2. `testRouteCoverageRandom`: facciamo variare tutti i parametri della richiesta in maniera casuale

Eseguendo solo `testRouteCoverageKWay` si ottiene la seguente coverage:

	Code Coverage														
	Lines			Branches			Paths			Functions and Methods			Classes and Traits		
Total		62.97%	658 / 1045		54.48%	231 / 424		20.09%	134 / 667		54.30%	82 / 151		35.00%	14 / 40
Http		67.31%	210 / 312		63.16%	60 / 95		46.38%	32 / 69		57.58%	19 / 33		44.44%	4 / 9
Models		73.60%	131 / 178		72.22%	65 / 90		8.66%	38 / 439		63.64%	28 / 44		22.22%	2 / 9
Policies		43.75%	7 / 16		50.00%	8 / 16		36.36%	4 / 11		16.67%	1 / 6		0.00%	0 / 1
Repositories		53.51%	244 / 456		40.39%	82 / 203		35.38%	46 / 130		41.18%	21 / 51		0.00%	0 / 9
Validators		79.52%	66 / 83		80.00%	16 / 20		77.78%	14 / 18		76.47%	13 / 17		66.67%	8 / 12

Figura 2.5.25: Coverage `testRouteCoverageKWay`

Eseguendo solo `testRouteCoverageRandom`:

	Code Coverage														
	Lines		Branches		Paths		Functions and Methods		Classes and Traits						
Total		65.84%	688 / 1045		56.13%	238 / 424		20.69%	138 / 667		55.63%	84 / 151		32.50%	13 / 40
Http		70.83%	221 / 312		66.32%	63 / 95		49.28%	34 / 69		54.55%	18 / 33		33.33%	3 / 9
Models		83.15%	148 / 178		75.56%	68 / 90		8.88%	39 / 439		70.45%	31 / 44		22.22%	2 / 9
Policies		56.25%	9 / 16		56.25%	9 / 16		45.45%	5 / 11		16.67%	1 / 6		0.00%	0 / 1
Repositories		53.51%	244 / 456		40.39%	82 / 203		35.38%	46 / 130		41.18%	21 / 51		0.00%	0 / 9
Validators		79.52%	66 / 83		80.00%	16 / 20		77.78%	14 / 18		76.47%	13 / 17		66.67%	8 / 12

Figura 2.5.26: Coverage testRouteCoverageRandom

Eseguendoli entrambi:

	Code Coverage														
	Lines		Branches		Paths		Functions and Methods		Classes and Traits						
Total		66.03%	690 / 1045		56.84%	241 / 424		21.44%	143 / 667		56.29%	85 / 151		35.00%	14 / 40
Http		71.47%	223 / 312		69.47%	66 / 95		52.17%	36 / 69		57.58%	19 / 33		44.44%	4 / 9
Models		83.15%	148 / 178		75.56%	68 / 90		9.57%	42 / 439		70.45%	31 / 44		22.22%	2 / 9
Policies		56.25%	9 / 16		56.25%	9 / 16		45.45%	5 / 11		16.67%	1 / 6		0.00%	0 / 1
Repositories		53.51%	244 / 456		40.39%	82 / 203		35.38%	46 / 130		41.18%	21 / 51		0.00%	0 / 9
Validators		79.52%	66 / 83		80.00%	16 / 20		77.78%	14 / 18		76.47%	13 / 17		66.67%	8 / 12

Figura 2.5.27: Coverage testRouteCoverageKWay + testRouteCoverageRandom

Il test rileva, come nel caso precedente, eventuali errori che si verificano nella gestione della richiesta HTTP. Se infatti questa ritorna stato 500 (errore), significa che non è stata gestita correttamente dall'applicazione. Analizzando le aree di codice non coperte, è stato possibile individuare del codice morto all'interno dell'applicazione, come il seguente:



```

public function reservationsChanged(Request $request): int
{
    $structure = Structure::whereId($request->get('structure_id'))->first();

    $this->authorize('poll', [Reservation::class, $structure]);

    if (!$structure) {
        return 0;
    }

    $lastUpdate = $request->get('last_update');

    $lastStructureUpdate = $structure->last_reservation_update;

    if (!$lastStructureUpdate) {
        return 0;
    }
}

```

Figura 2.5.28: Codice Morto

Appare evidente che il primo if non sarà mai true: se la struttura non esiste, infatti, l'autorizzazione viene negata e quindi la condizione !\$structure non sarà mai true.

## 2.5.6 Conclusioni

Dopo gli ultimi test è stata raggiunta la seguente copertura:

	Code Coverage											
	Lines		Branches		Paths		Functions and Methods		Classes and Traits			
Total	76.84%	803 / 1045	74.53%	316 / 424	26.99%	180 / 667	63.58%	96 / 151	37.50%	15 / 40		
Http	70.19%	219 / 312	67.37%	64 / 95	49.28%	34 / 69	57.58%	19 / 33	44.44%	4 / 9		
Models	73.60%	131 / 178	72.22%	65 / 90	9.57%	42 / 439	63.64%	28 / 44	22.22%	2 / 9		
Policies	56.25%	9 / 16	56.25%	9 / 16	45.45%	5 / 11	16.67%	1 / 6	0.00%	0 / 1		
Repositories	82.24%	375 / 456	79.31%	161 / 203	64.62%	84 / 130	66.67%	34 / 51	0.00%	0 / 9		
Validators	83.13%	69 / 83	85.00%	17 / 20	83.33%	15 / 18	82.35%	14 / 17	75.00%	9 / 12		

Figura 2.5.29: Copertura Finale

Sia per le righe di codice che per i branch la copertura risulta tra il 70% e l'80%, che è la condizione che ci eravamo prefissati, mentre la copertura dei path risulta ancora debole (27%). Gran parte del

---

codice non coperto è rappresentato dalla funzione che gestisce i filtri della prenotazione scopeFilter della model Reservation, la quale era la classe indicata con maggiore complessità nel report di XDebug.

La code coverage, comunque, non ci da garanzia sulla mancanza di difetti. Abbiamo trovato casi in cui l'applicazione non riesce a gestire determinate combinazioni di input e corretto tali difetti, tuttavia potrebbero esistere altre combinazioni più mirate che non sono state provate le quali potrebbero sollevare errori, per non parlare poi di eventuali difetti che non provocano fallimenti/eccezioni, ma comportamenti inaspettati del software.

# Capitolo 3

## Testing Frontend - Dusk

Abbiamo verificato il funzionamento dell'applicazione lato backend. Questo, tuttavia, non ci dà alcuna garanzia che le funzionalità testate interagiscano in maniera corretta con le view, ad esempio queste potrebbero inviare dati al backend che non sono previsti nei test.

Esiste un'estensione ufficiale di Laravel, chiamata Dusk<sup>1</sup>, basata sempre su PHPUnit che consente di scrivere test espressivi per effettuare test sul frontend. Tale testing, dal momento che interagisce col l'applicazione nel suo complesso, può essere considerato di sistema.

### 3.1 Configurazione

Una volta installata l'estensione usando composer, bisogna considerare che Dusk usa Selenium e richiede il driver di Chrome per simulare la navigazione sul browser, è necessario, quindi, avere il driver di Chrome nel PC con il path salvato nelle variabili d'ambiente (è allegato nella cartella della consegna).

Prima di scrivere test, è stato necessario risolvere un ulteriore problema: mentre nel caso del backend abbiamo usato un in-memory database, Dusk utilizza un processo completamente separato dagli altri test, questo significa che dobbiamo definire un database di test dedicato esclusivamente a Dusk.

---

<sup>1</sup>[Documentazione Ufficiale](#)

### 3.1.1 Setup File d'Ambiente

Se non viene definito un file d'ambiente ad hoc per Dusk, questo userà il file d'ambiente globale (.env), andando quindi a usare per i test il database definito per l'applicazione. Per ovviare a questo problema, è stato definito, seguendo lo standard definito nella documentazione, un file d'ambiente chiamato .env.dusk.local

Quando vengono eseguiti i test di dusk, il framework effettua un backup del .env dell'applicazione, lo sostituisce con quello di dusk e terminata l'esecuzione ripristina i file.

### 3.1.2 Setup Database

Invece di usare un database hostato su un servizio esterno, Laravel permette di definire dei database SQLite tramite i file di configurazione e che viene gestito dal suo server built-in. SQLite, rispetto a un database Mysql, comporta i seguenti vantaggi:

- Tutte le tabelle sono gestite in un unico file
- È molto più veloce

Tuttavia, presenta anche numerosi svantaggi come impossibilità di gestire la concorrenza e di definire stored procedures.

Per effettuare il setup del database, sono stati eseguiti i seguenti step (non è necessario riprodurli):

1. Creazione del file database/sqlite.testing.database (conterrà la definizione delle tabelle)
2. Definizione del database in config/database.php

```
'sqlite_testing' => [  
    'driver' => 'sqlite',  
    'url' => env('DATABASE_URL'),  
    'database' => database_path('sqlite.testing.database'),  
    'prefix' => '',
```

```
        'foreign_key_constraints' => env('DB_FOREIGN_KEYS', true)  
    ]
```

### 3. Assegnazione database in .env.dusk.local

```
DB_CONNECTION=sqlite_testing  
DB_HOST=127.0.0.1  
DB_PORT=3306  
DB_DATABASE=  
DB_USERNAME=root  
DB_PASSWORD=
```

#### 3.1.3 Creazione File di Base

Lanciando il comando tramite terminale

```
php artisan dusk
```

Oppure

lanciando il file run\_dusk\_tests.bat. **E' importante avviare prima il server (run\_server.bat) prima di lanciare i test sul frontend.**

Vengono lanciati i test di Dusk definiti in tests/Browser e, se non esiste, viene creata la classe astratta tests/DuskTestCase che estende BaseTestCase. Tutti i test di Dusk estenderanno tale classe, la quale contiene il setup iniziale.

## 3.2 Scrrittura ed Esecuzione dei Test

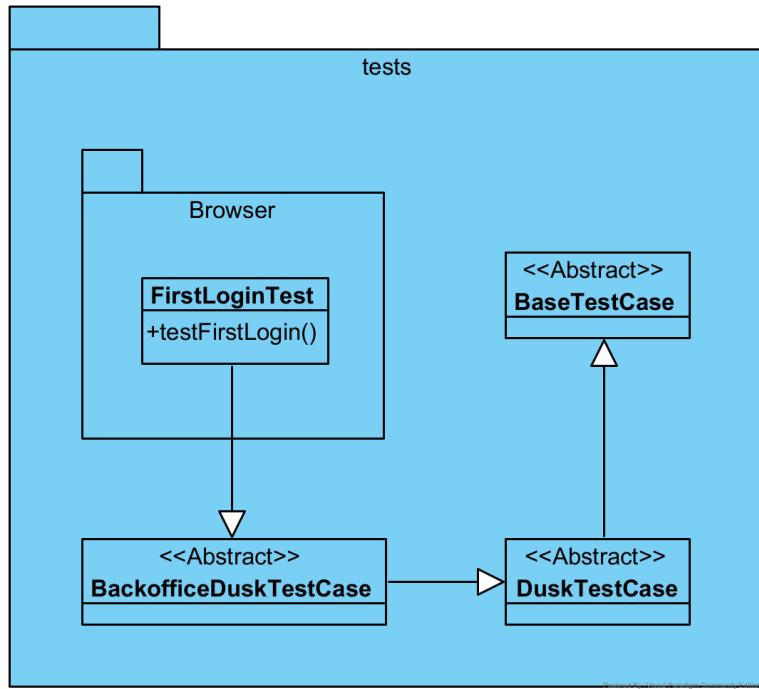


Figura 3.2.1: Automa a Stati Finiti dell’Interfaccia

Invece di estendere direttamente DuskTestCase, abbiamo anche qui la necessità di creare e popolare il database con dei dati comuni per ogni test, ragion per cui, come fatto per il backend, è stata creata una classe chiamata tests/BackofficeDuskTestCase.php che si occupa di effettuare il setup dei test, e i test di Dusk estenderanno tale classe.

Prima di procedere, di seguito è riportato l’automa a stati finiti dell’interfaccia:

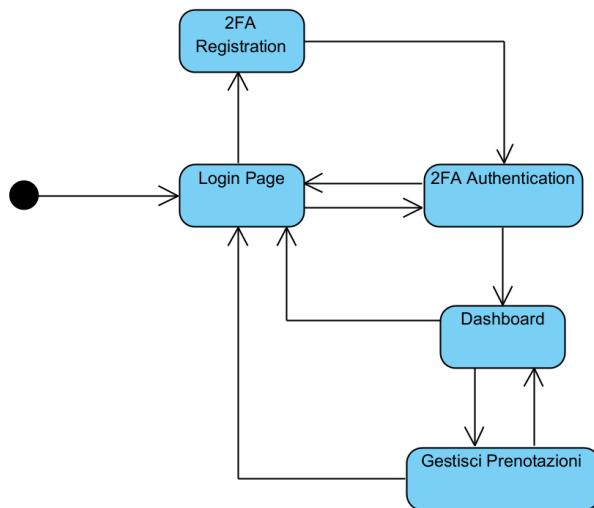


Figura 3.2.2: Automa a Stati Finiti dell'Interfaccia

### 3.2.1 First Login Test

L'obiettivo è verificare che l'autenticazione a due fattori funzioni correttamente al primo login. In particolare, il flusso di esecuzione è il seguente:

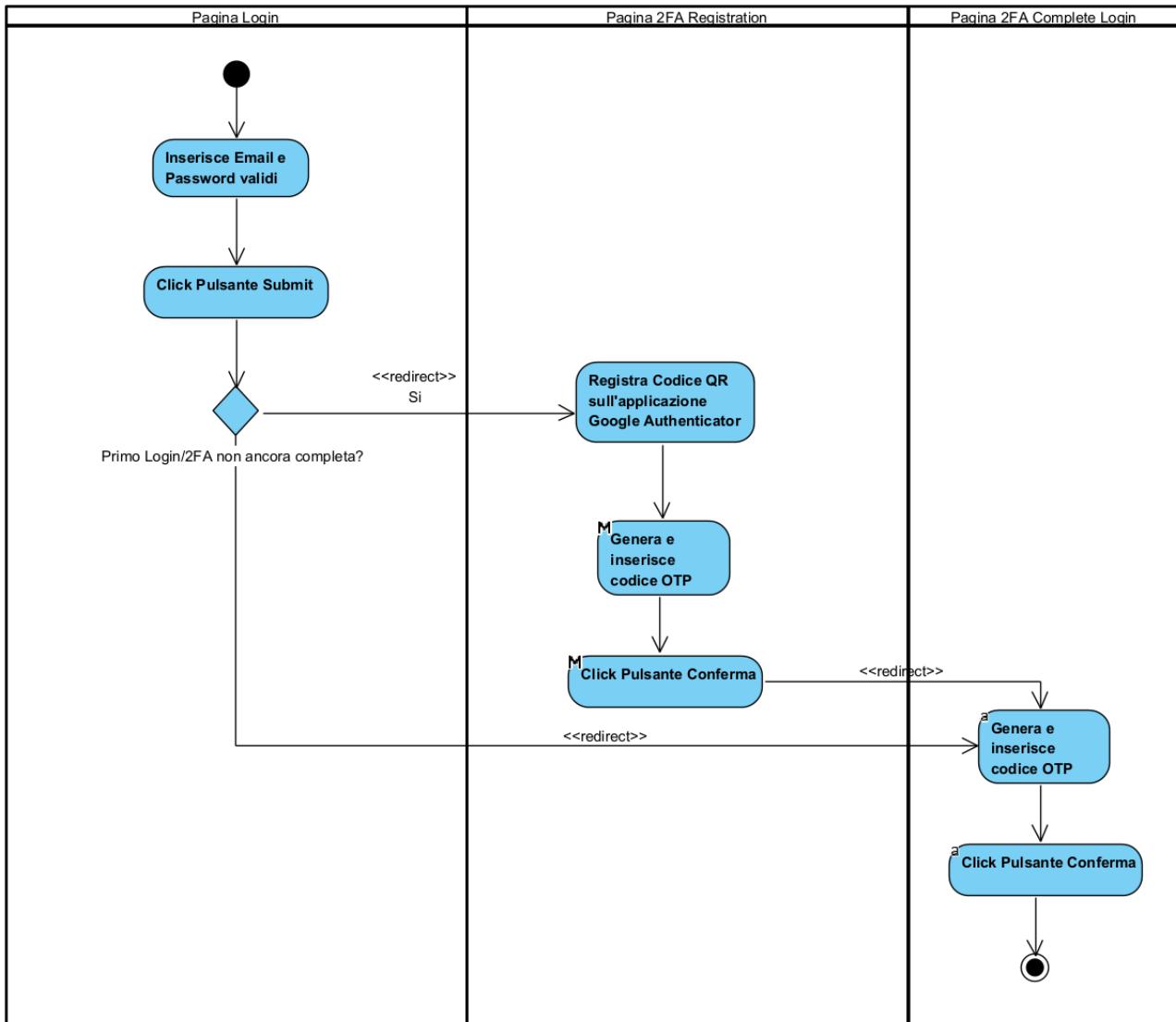


Figura 3.2.3: Activity 2FA

Dobbiamo verificare che il funzionamento dell'applicazione rispecchi l'activity riportato in figura.

Classe: tests/Feature/Whitebox/FirstLoginTest.php

Precondizioni:

- Nel database è presente una struttura sanitaria
- Nel database è presente un responsabile
- Il responsabile non ha ancora effettuato il primo login

<b>Test Case ID</b>		firstLoginTest		
<b>Test Case Description</b>		Test per verificare che l'applicazione gestisca correttamente il primo login dell'operatore sanitario. Quando questi effettua il login per la prima volta, gli viene mostrato il codice QR per effettuare il collegamento con l'applicazione Google Authenticator. Una volta ottenuto il codice temporaneo, inserendolo all'interno del form e confermando, attiva il suo account. Da quel momento in poi, ad ogni login, gli verrà chiesto di inserire il codice OTP generato da Google Authenticator.		
Step #	Test Data	Step #	Test Data	
2	email = 'operator@email.it'		/qr/register submit button xpath = 3 '/html/body/div/div/main/div/div/div/form/button'	
2	password = 'test'		/qr/authenticate submit button xpath = 4 '/html/body/div/div/main/div/div/div/div/form/div[3]/button[1]'	
2	login button xpath = '/html/body/div/div/main/div/div/div/div/form/div[5]/button'			
Step #	Step Details	Output Atteso	Actual Results	Result
1	Visito la pagina /login	La pagina viene caricata, mostrando tutti i widget previsti	As Expected	Pass
2	Inserimento email e password e click sul pulsante login	Viene caricata la pagina /qr/register, mostrando tutti i widget previsti	As Expected	Pass
3	Inserimento OTP generato tramite secret nel form e click sul pulsante di conferma	Viene caricata la pagina /qr/authenticate, mostrando tutti i widget previsti	As Expected	Pass
4	Inserimento del codice OTP temporaneo e click su conferma	Viene caricata la pagina /dashboard, mostrando tutti i widget previsti	As Expected	Pass



Il test è stato scritto per salvare degli screenshot ad ogni step, sono riportati di seguito quelli principali:

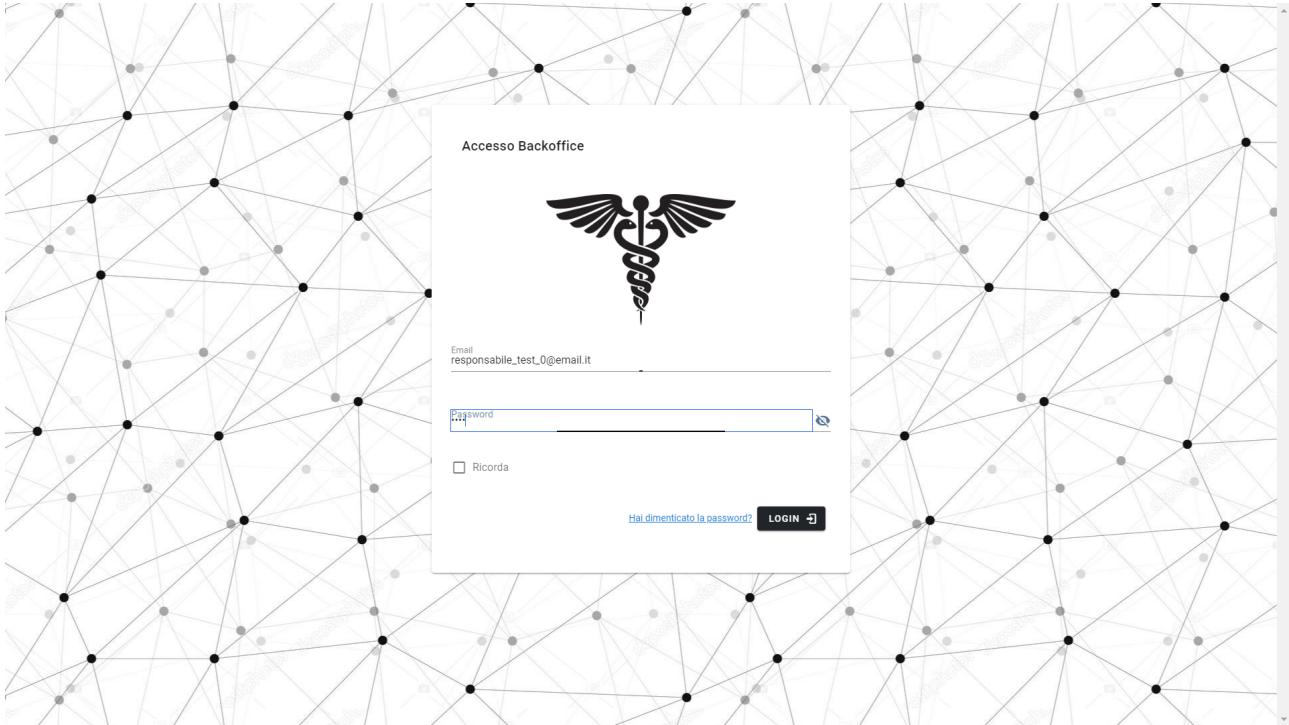


Figura 3.2.4: /login

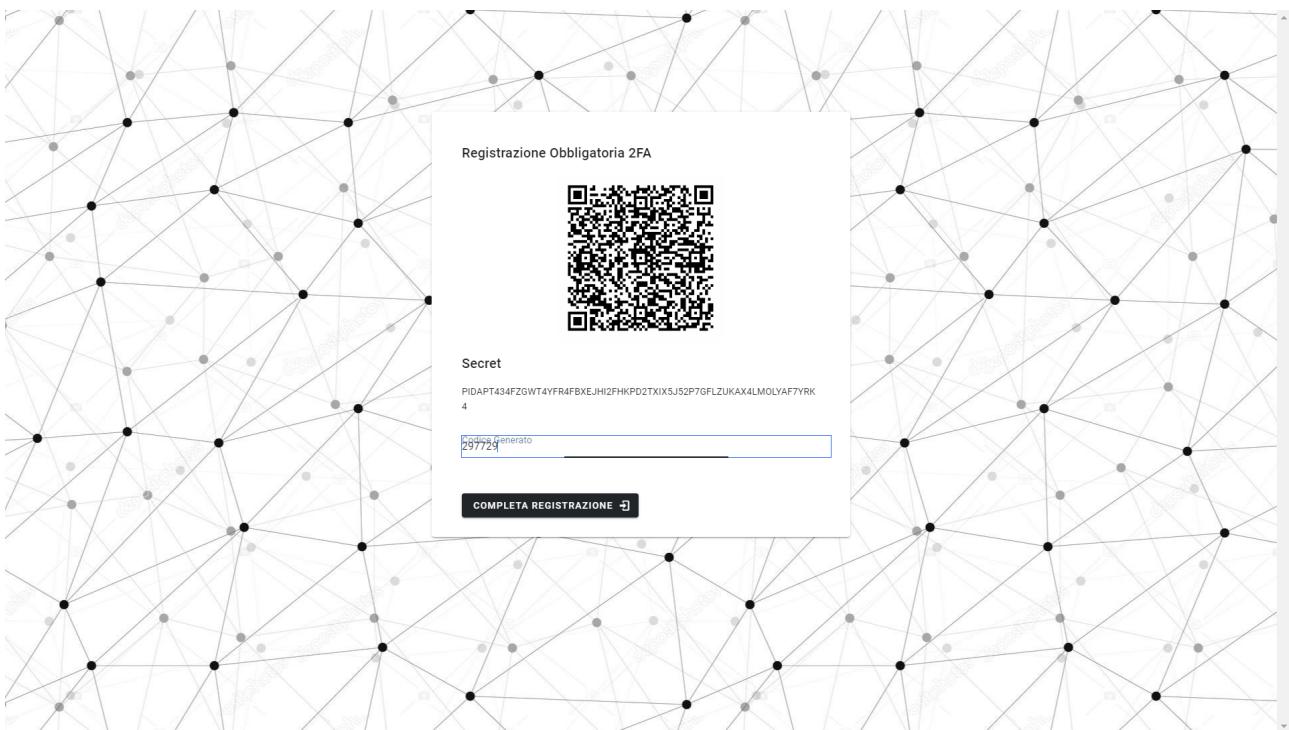


Figura 3.2.5: /qr/register

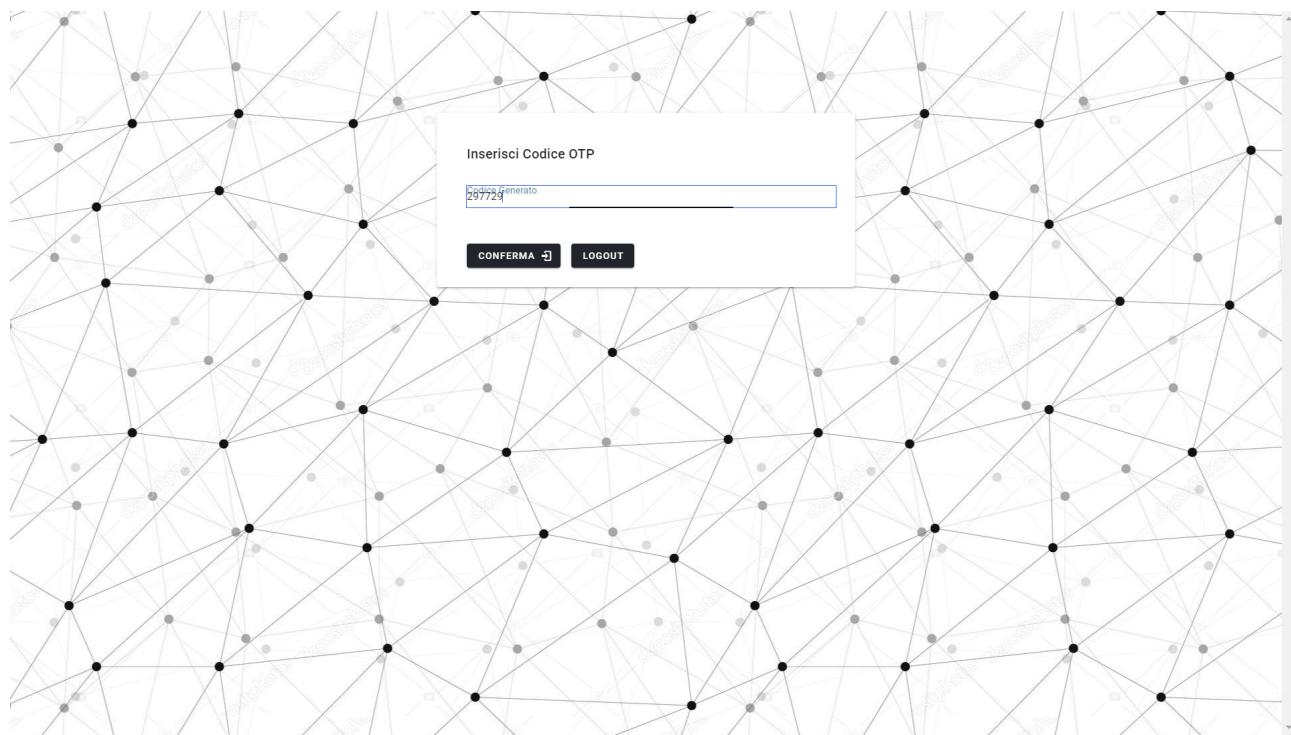


Figura 3.2.6: /qr/authenticate

Figura 3.2.7: /dashboard

## Conclusioni

I test effettuati tramite Dusk sono molto utili e relativamente semplici da scrivere, tuttavia presentano alcune limitazioni:

- Si è cercato di scriverli quanto più robusti possibile, ad esempio per individuare i campi del login (email e password), invece di individuarli tramite posizione assoluta, stili o xpath (tutti elementi che potrebbero cambiare alla minima modifica dell'interfaccia), si è presa la lista degli input testuali e si sono riempiti i primi 2. Cosa potrebbe accadere, però, qualora ad esempio si decidesse di modificare la pagina di login per comprendere anche un form dedicato alla registrazione? O se inserissimo un campo testuale per inserire direttamente l'OTP senza passare per una schermata intermedia? Il test verrebbe probabilmente invalidato.

Per i pulsanti, inoltre, è stato utilizzato l>xpath assoluto, vale a dire che modificando la pagina anche di poco il test verrebbe invalidato

- Possono essere scritti esclusivamente da chi ha accesso al codice
- Richiedono comunque certe skill di programmazione

## 3.3 Valutazione della Test Suite Realizzata

Abbiamo usato la code coverage come metrica di valutazione, tuttavia possiamo anche considerare l'efficacia e l'efficienza.

### 3.3.1 Efficacia

L'efficacia, in teoria, è pari a

$$\text{Efficacia} = \frac{\text{Fault Trovati}}{\text{Fault Esistenti}}$$

tuttavia è impossibile da valutare, dal momento che il numero di fault esistenti non può essere noto.

In alternativa, possiamo valutarla come numero di fallimenti trovati o come numero di difetti trovati.

La test suite realizzata ha permesso di individuare 47 difetti e 216 fallimenti.

### 3.3.2 Efficienza

Un'alternativa all'efficacia è l'efficienza, anche questa valutabile secondo diversi punti di vista:

$$Efficienza_{Fallimenti-Test} = \frac{Fallimenti\ Trovati}{Test\ Eseguiti} = \frac{216}{98} = 2.2 \quad \text{💡}$$

$$Efficienza_{Difetti-Test} = \frac{Difetti\ Trovati}{Test\ Eseguiti} = \frac{47}{98} = 0.48$$

$$Efficienza_{Fallimenti-Effort} = \frac{Fallimenti\ Trovati}{Ore\ Testing} = \frac{216}{8*15} = 1.8$$

$$Efficienza_{Difetti-Effort} = \frac{Difetti\ Trovati}{Ore\ Testing} = \frac{47}{8*15} = 0.39$$

### 3.3.3 Conclusioni

Efficacia ed efficienza possono costituire una metrica per misurare la test suite, tuttavia da sole non danno molte indicazioni sulla bontà della test suite. Ad esempio abbiamo trovato 47 difetti, ma è difficile stabilire se i difetti rimanenti si avvicinino a questo numero o se siano in misura nettamente maggiore. L'ideale sarebbe avere Efficacia ed Efficienza relative, ma non avendo una seconda test suite da prendere come riferimento bisogna accontentarsi.

## 3.4 Differenze tra PHPUnit e JUnit

I due framework presentano diverse similitudini e differenze:

- JUnit consente di scrivere test direttamente nell'implementazione delle classi usando la notazione `@test`, in PHPUnit ciò non è possibile
- Le notazioni di JUnit consentono di effettuare una più ampia gamma di operazioni
- JUnit consente di creare test data driven direttamente da csv, PHPUnit non supporta il caricamento diretto di file
- JUnit supporta nativamente la code coverage, PHPUnit ha bisogno del plugin XDebug
- JUnit non supporta nativamente l'uso di Mock (ma è comunque possibile usare una libreria esterna), mentre PHPUnit implementa il supporto nativo ai Mock

### 3.5 Approcci Alternativi - Analisi Statica

Fin ora è stato descritto l'approccio sperimentale adottato per la verifica del software. A questo, è stata affiancata l'analisi statica del codice tramite l'IDE<sup>2</sup>, che fornisce un supporto in tal senso, e un plugin della stessa, PHP Stan.

---

<sup>2</sup>PHPStorm

# Capitolo 4

## Testing Frontend - Testim

Un'alternativa rispetto a scrivere test con Dusk, è quella di utilizzare un tool di ~~coding~~&replay. La strategia è semplice: si registrano le operazioni effettuate sul sito e il tool è in grado di ripeterle passo dopo passo, consentendo di specificare anche delle asserzioni (ad esempio se si vede una certa scritta).

### 4.1 Katalon Recorder

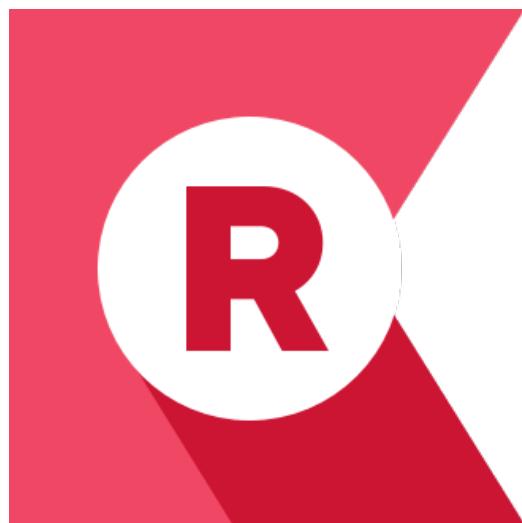


Figura 4.1.1: Katalon Recorder

E' un plugin per Chrome open source, permette, come accennato, di registrare delle operazioni effettuate su di un sito e ripetere la sequenza. Tra i pro ritroviamo sicuramente che è possibile creare rapidamente test, persino da chi non è un programmatore, tuttavia presenta alcune limitazioni:



- Per individuare gli elementi è sempre necessario utilizzare dei selector o gli xpath, rendendo i test poco robusti
- Non è possibile specificare logica esterna, ad esempio nel caso dell'autenticazione a 2 fattori non è possibile scrivere una funzione che vada a calcolare il codice generato
- Non è possibile effettuare asserzioni sul backend (se ad esempio confermo una prenotazione tramite interfaccia, come faccio a sapere se è stata correttamente aggiornata nel database?)
- I test non sempre sono ripetibili, ad esempio se creo una prenotazione tramite test, non sarà possibile ricreare la stessa prenotazione dato che sarà già inserita nel database

Per ovviare a queste limitazioni, è stato utilizzato un altro tool chiamato Testim.

## 4.2 Testim

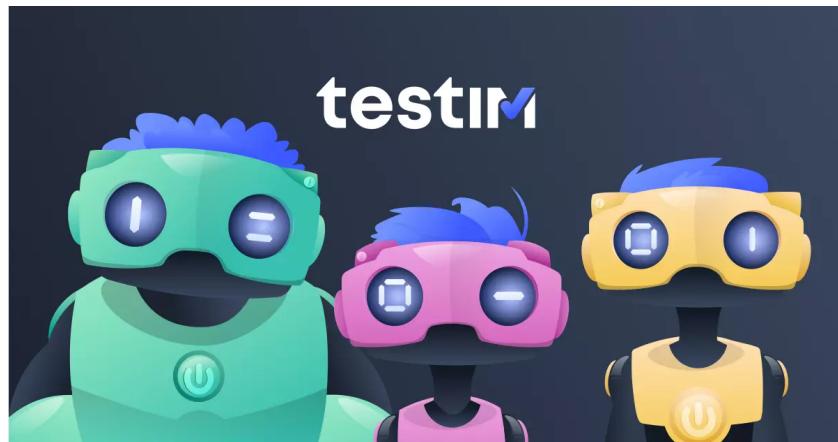


Figura 4.2.1: Testim

Basato su Selenium, è un Framework per l'automated testing che consente di creare test basati sull'intelligenza artificiale più robusti. Esso permette infatti di:

- Selezionare i widget da testare tramite Catch and Replay
- Individuare gli elementi di una pagina usando modelli di IA per riconoscerli anche dopo visibili cambiamenti (seppur entro certi limiti)

- Scrivere delle funzioni e definire variabili custom
- Estrarre informazioni testuali dai widget e assegnarli a delle variabili
- Definire validation complesse
- Comunicare con il backend dell'applicazione tramite chiamate API, ad esempio si potrebbero scrivere delle API di setup e teardown dei test, in maniera tale da impostare certe precondizioni all'inizio di un test e ristabilire le condizioni iniziali dopo la sua esecuzione

Andando su [testim.io](#) è possibile registrarsi (sono permesse solo email istituzionali, come quella unina), una volta effettuato accesso all'account, il sito consiglia di scaricare il plugin Testim Editor, il quale consente di registrare test tramite browser, in maniera molto simile a come fa Katalon Recorder. La differenza sta nel fatto che il test non viene registrato in locale, ma sulla piattaforma di [testim.io](#) ed è possibile personalizzarlo da lì.

#### 4.2.1 First Login Test

Similmente a quanto fatto con Dusk, testiamo che il primo login funzioni in maniera corretta. Come accennato, si presentano i seguenti problemi:

1. Dusk avviava un processo separato dall'applicazione, il quale creava un database temporaneo con dati temporanei che venivano cancellati alla fine del test, in questo caso i test interagiscono direttamente con il database dell'applicazione, di conseguenza le modifiche che effetuiamo sono permanenti
2. Conseguenza della 1, quando il responsabile scelto avrà effettuato il primo accesso, non sarà più possibile ripetere il test, perché gli sarà già stata assegnata la secret. Questo sembra un problema specifico, il realtà si presenta anche per tutte le altre modifiche permanenti
3. Come generiamo il codice OTP corretto? Dusk era in grado di accedere al backend dell'applicazione e di usare la libreria Google2FA per eseguire l'algoritmo e calcolare l'OTP dalla secret, qui possiamo scrivere delle funzioni custom ma il calcolo dell'otp è abbastanza complesso

## Soluzioni:

1. E' stato aperto un endpoint firstLoginTestSetup, accessibile tramite chiamata API, che crea un nuovo account che ancora deve effettuare il primo accesso e gli assegna una chiave alfanumerica unica, restituendo a testim email e password per l'accesso, più tale identificativo
2. Alla fine del test verrà chiamato un secondo endpoint, firstLoginTestTeardown, al quale viene passato l'ID generato dal primo endpoint e tramite questo individua l'account creato e lo elimina, ripristinando il database nello stato originario
3. E' stato aperto un terzo endpoint che prende in ingresso la secret e restituisce il codice OTP, per semplicità è stato creato all'interno dell'applicazione Laravel, ma volendo era possibile crearlo usando una qualsiasi altra tecnologia

Dal momento che l'applicazione genera dinamicamente un nuovo account per ogni test, adesso è possibile anche eseguire un numero indefinito di test in parallelo che operano sullo stesso database.

Per creare il test, è stato sufficiente utilizzare la funzionalità “record” per fare il catch delle operazioni, successivamente sono state aggiunte le chiamate API e sono stati sostituiti gli input usati (email, password, etc...) con i valori forniti dalle API. Infine, è stato eseguito il replay delle operazioni.

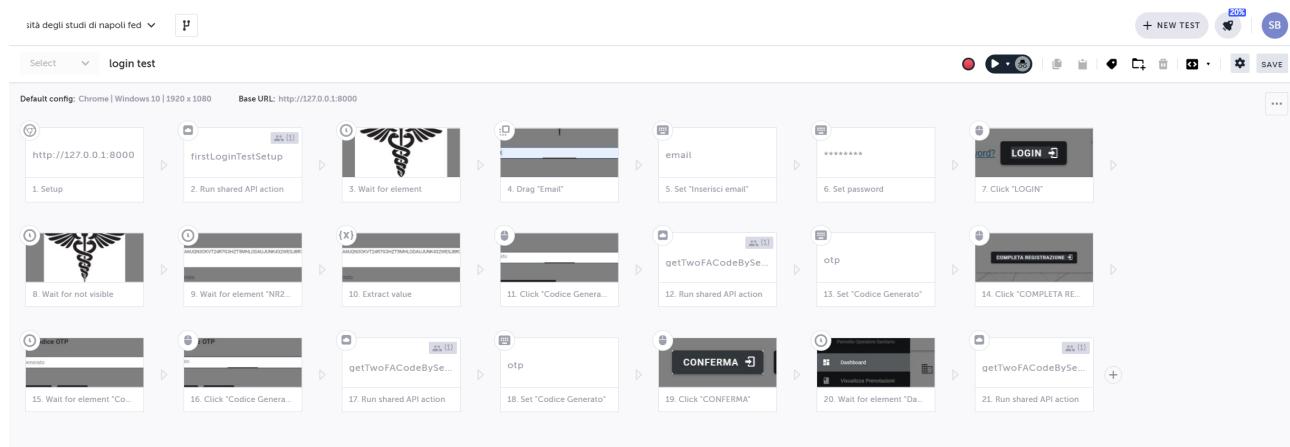


Figura 4.2.2: Sequenza delle Operazioni in Testim.io

Di seguito è riportato il sequence diagram del test (le chiamate API sono definite in app/http/Controllers/ApiTestController.php):

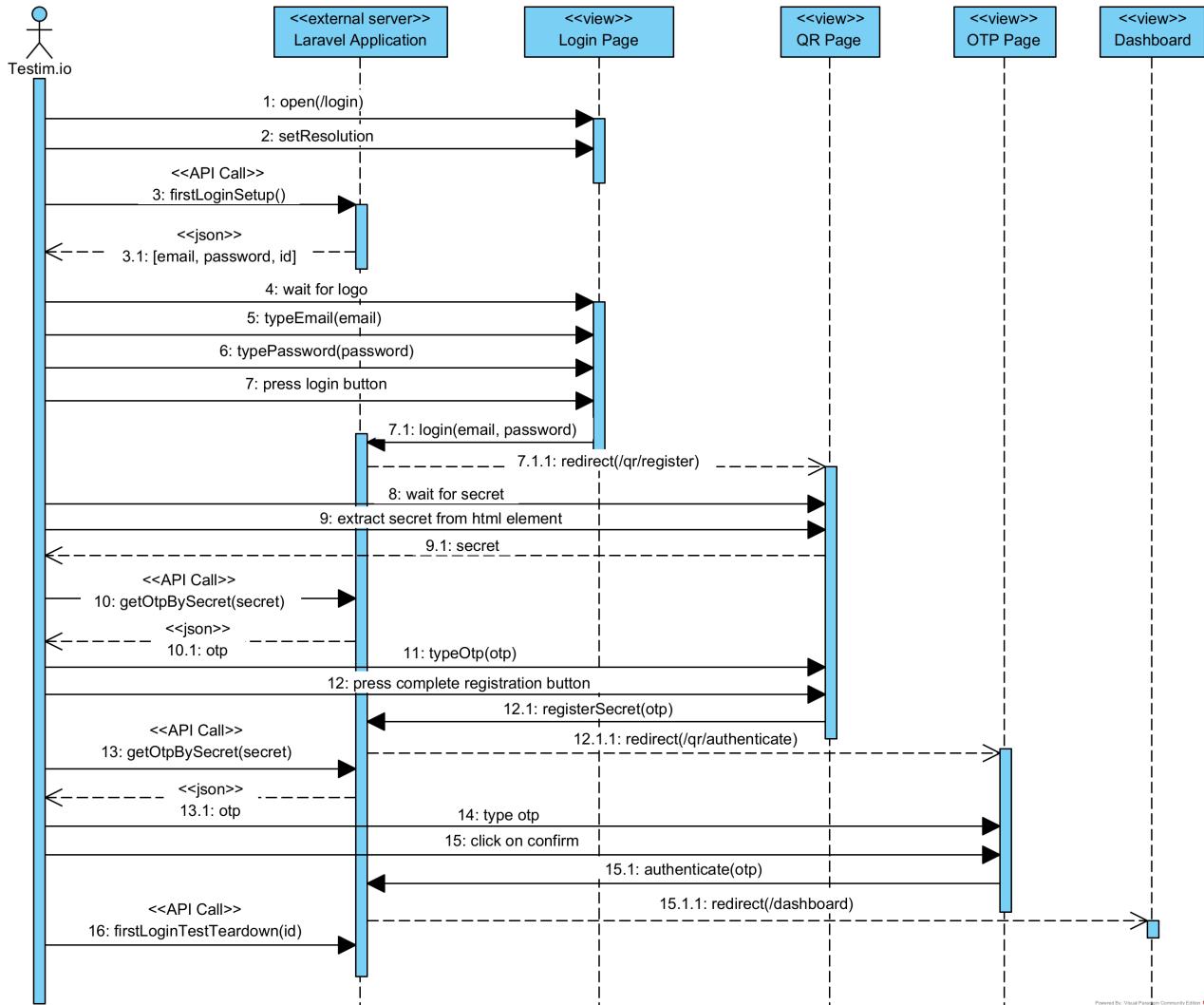


Figura 4.2.3: Sequence Test First Login

### Verifica della Robustezza del Test

Come accennato, a differenza degli altri strumenti di catch&replay come Katalon Recorder, che usano selector e xpath, Testim.io usa modelli di IA per individuare gli elementi selezionati. Mettiamo alla prova la robustezza del test, supponendo che vengano effettuate modifiche alla pagina di login e vediamo se continua a funzionare. E' stata creata la seguente pagina di fake login:



Figura 4.2.4: Fake Login

Sono state apportate le seguenti modifiche:

- I campi username e password sono stati invertiti
- E' stato aggiunto un nuovo campo testuale
- I pulsanti sono stati invertiti
- E' stato aggiunto un nuovo pulsante
- Il logo è stato spostato in basso

Eseguendo lo stesso identico test (cambiando l'URL di base in <http://127.0.0.1:8000/fake-login>), te-stim.io riesce comunque a individuare gli elementi all'interno della pagina, ad inserire gli input correttamente e a trovare il pulsante giusto.

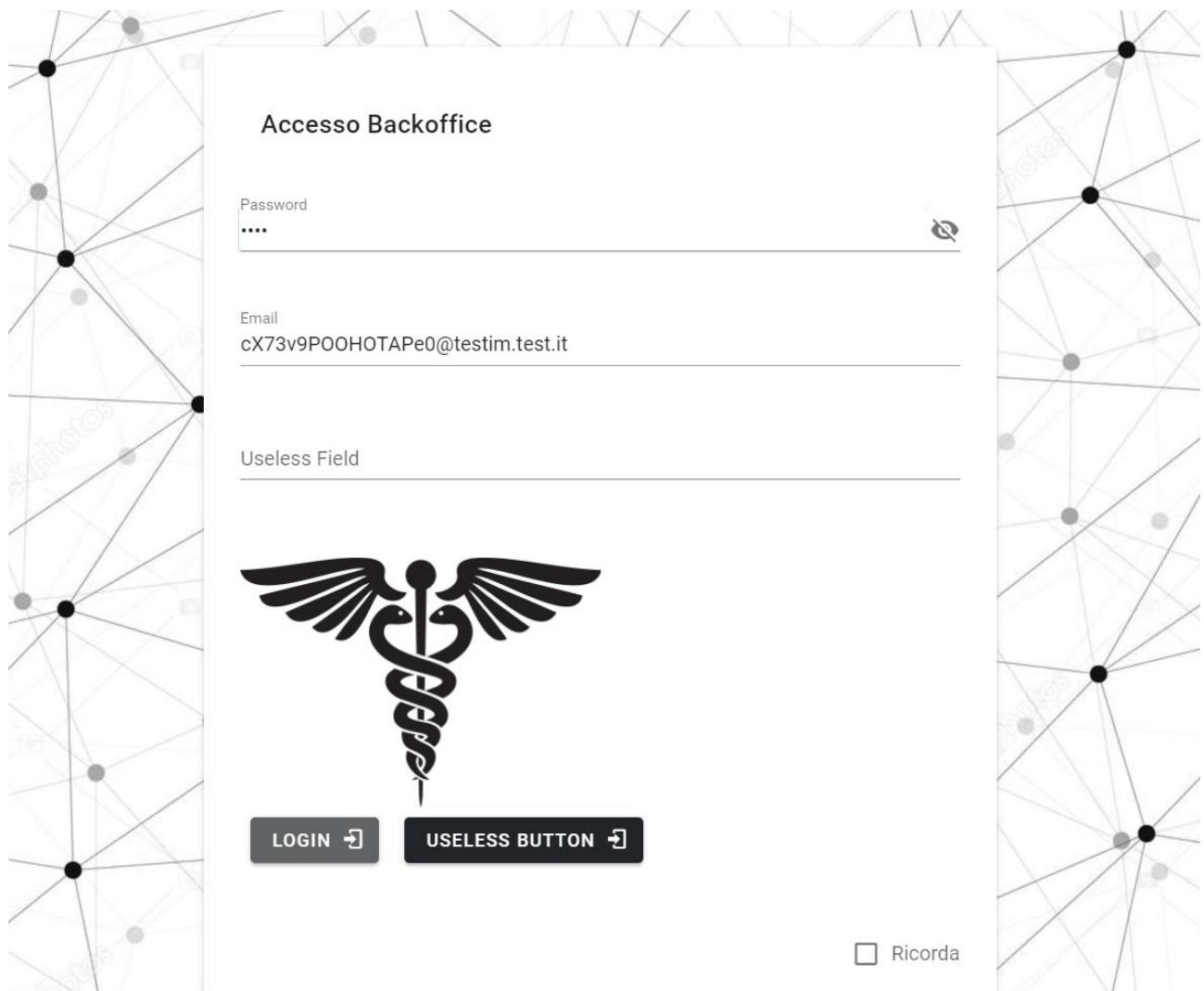


Figura 4.2.5: Fake Login Input Inseriti

Sono state eseguite in sequenza le seguenti ulteriori modifiche per provare a far fallire il test:

- Rimozione di alcuni attributi dai tag dei componenti (ad esempio l'ID) -> il test regge
- Cambio del testo del pulsante di login da 'Login' a 'Lowgain' -> il test regge
- Cambio del colore del pulsante da nero a blu -> il test fallisce perché non riesce ad individuare il pulsante corretto

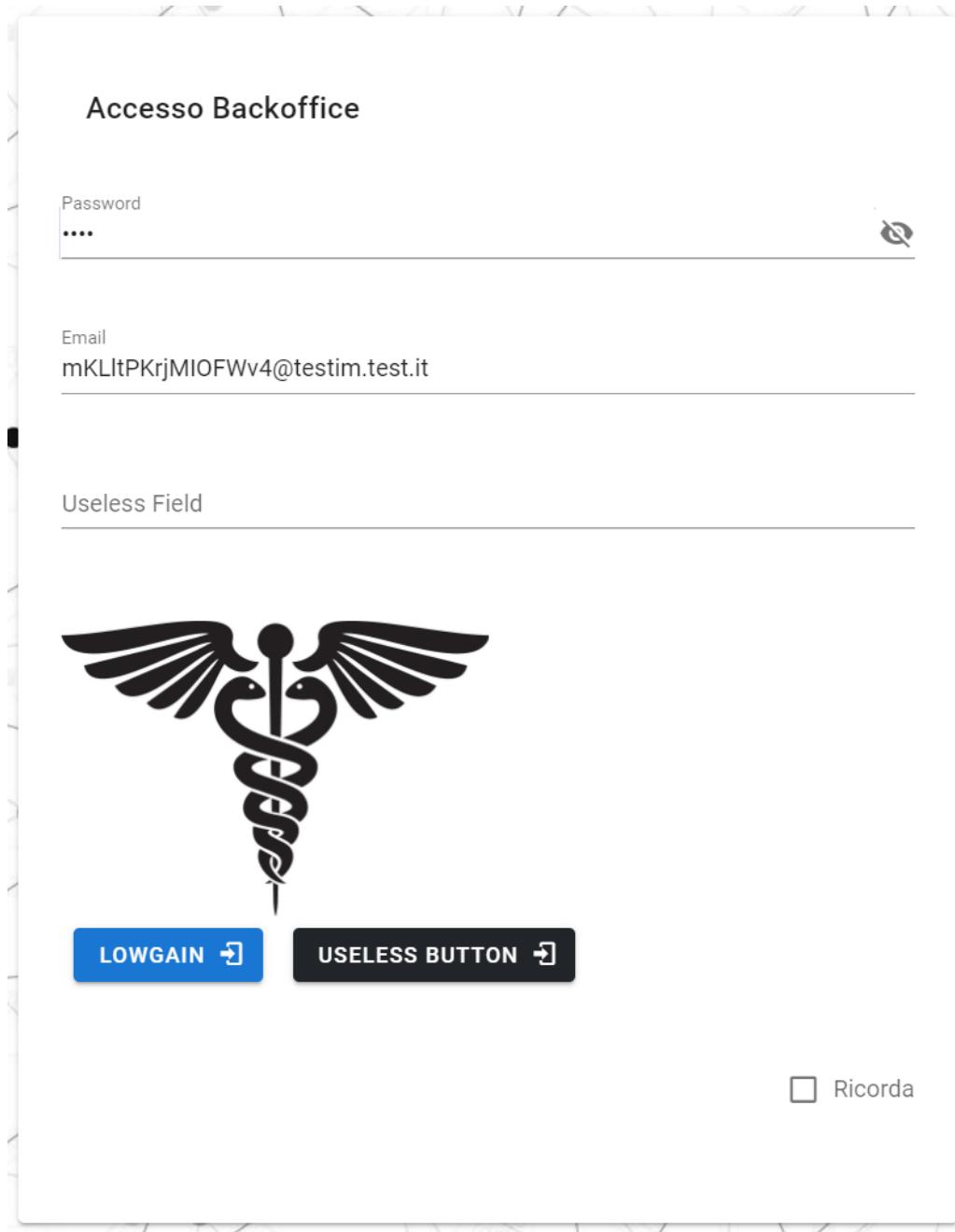


Figura 4.2.6: Broken Login Input Inseriti

E' curioso osservare che se si cambia solo il colore o solo il testo il test regge, ma se si cambiano testo e colore il test individuerà come pulsante di login "Useless Button", reputandolo più simile al pulsante del quale era stato fatto il catch.

## Conclusioni

Testim.io si presta molto bene a scrivere test robusti che possono continuare a funzionare anche dopo considerevoli modifiche al codice, tuttavia bisogna effettuare le seguenti considerazioni:

1. La robustezza può essere una lama a doppio taglio, infatti consideriamo i seguenti casi:
  - Il layout della pagina cambia e bisogna spostare il logo a destra invece che a sinistra
  - Uno sviluppatore modifica/cancella degli stili css e il layout di alcune pagine viene sfondato, ma non in maniera abbastanza grave da far fallire il test

Nel primo caso la robustezza può risultare utile, perché non siamo costretti a modificare il test ad ogni minima modifica, nel secondo, invece, il test non si accorgerebbe del problema.

Una possibile soluzione potrebbe essere quella di scrivere test poco robusti per tutti quegli elementi che siamo sicuri che non siano soggetti a modifiche, mentre robusti per quelli che potrebbero variare.
2. I test scritti necessitano comunque che i tester abbiano accesso all'applicazione, infatti sono stati creati degli endpoint che fanno il setup e il teardown del test. Tuttavia, non sempre i tester coincidono con gli sviluppatori e non sempre possono mettere mano al codice sorgente. Questo ostacolo, nel caso preso in esame, può essere comunque evitato in diversi modi:
  - Nel caso della 2FA, non è necessario connettersi direttamente al web server, è anche possibile scrivere direttamente l'algoritmo su testim.io o effettuare chiamate a un servizio completamente separato
  - Nel caso delle credenziali, gli sviluppatori dovrebbero fornire ai tester i dati da immettere (per il login, ad esempio, degli account di prova)

Si risolve, così, il problema di dover mettere mano al codice sorgente.

Se si desidera accedere al test tramite testim.io, è possibile usare le seguenti credenziali:

- Email: sim.bianco@studenti.unina.it
- Password: #SoftwareTesting2021