Politecnico di Torino

Corso di Laurea Magistrale in

Ingegneria Informatica

Tesi di Laurea Magistrale

Politecnico di Torino

Testing e Integrazione di un sistema Smart Home per la gestione degli avvisi di privacy di dispositivi IoT

Relatore:
Prof. Luca Ardito
Co-Relatore:
Prof. Luigi De Russis

Candidato:
Marco Russo

Anno accademico 2022/2023

Sommario

Il problema affrontato riguarda la realizzazione di un sistema Smart Home per la gestione degli avvisi di privacy di dispositivi IoT, sia da parte degli utenti finali che da parte di Data Protection Officers e Data Controllers. A questo fine è necessaria una piattaforma comune con funzionalità differenziate a seconda del ruolo dell'utente. Introducendomi all'interno del progetto con l'applicativo già sviluppato e funzionante, il mio contributo nei confronti del problema è stato quello di effettuare il testing e di realizzare il processo di Continuous Integration (CI).

Nell'elaborato viene prima illustrato il contesto dell'applicativo da esaminare, la Privacy Dashboard. Si discute, quindi, delle vulnerabilità di privacy associate a sistemi IoT, in particolare di quelli legati ai servizi di domotica, tipicamente conosciuti come Smart Home. Segue una spiegazione del framework SIFIS-Home, all'interno della quale si inserisce la Privacy Dashboard, e della struttura implementativa della Privacy Dashboard stessa.

Il testing dell'applicativo viene effettuato tramite test di unità, mirando a raggiungere una copertura del codice pari almeno al 60%. I test sono stati scritti con la metodologia White Box, includendo nella progettazione del test la struttura del codice da analizzare. Dove necessario vengono utilizzati degli oggetti mock, per emulare funzionalità di classi esterne a quella da esaminare. In alcuni punti in particolare, a causa della struttura della classe in esame, è stato necessario per l'utilizzo dei mock aggirare l'incapsulamento degli attributi (le limitazioni al loro accesso). Per la CI sono state implementate sia un flusso che genera una nuova versione dell'applicativo, sia un flusso che reagisce alle modifiche del codice testandolo e creando una immagine eseguibile.

I test hanno rilevato pochi errori all'interno del codice, principalmente dovuti a variabili male assegnate o valori scambiati. Sono emersi, grazie ad ispezioni per correggere gli errori, anche problematiche di logica di implementazioni in alcuni metodi. Infine, il processo stesso di testing si è imbattuto in problemi di manutenibilità, che hanno causato l'elusione dell'incapsulamento.

Indice

Elenco delle figure

Capitolo 1

Internet of Things, SIFIS-Home e Privacy Dashboard

1.1 Internet of Things

Il progetto rientra nell'ambito del Internet of Things[1] e, in particolare, del concetto di Smart Home. Con il termine Internet of Things ci si riferisce ad una serie di dispositivi che, grazie ad una loro interconnessione (tipicamente Internet), riescono a fornire dei servizi più complessi rispetto a quanto potrebbero singolarmente.
Una definizione univoca di Internet of Things non esiste, in quanto ogni contesto che ne fa uso ne possiede una propria; tuttavia, gli enti più autorevoli concordano su uno scenario in cui un certo numero di dispositivi (tendenzialmente elevato) comunicano tra di loro. Infatti, la ITU-T[2] definisce lo IoT come "Un'infrastruttura globale

[1] In italiano: Internet delle Cose

[2] International Telecommunication Union – Telecommunication Standardization Sector, in italiano: Unione Internazionale delle Telecomunicazioni – Settore Standardizzazione. La ITU è l'agenzia specializzata delle Nazioni Unite per le Tecnologie di Informazione e Comunicazione [1]. La ITU-T è responsabile dello sviluppo di standard internazionali, noti come Recommendations, che sono elementi cardine nell'infrastruttura globale delle Tecnologie di Informazione e Comunicazione [2].

per la società dell'informazione, che permette dei servizi avanzati interconnettendo oggetti (fisici e virtuali) basandosi su tecnologie di informazione e comunicazione" [3], mentre la IAB[3] lo definisce come "una tendenza dove un grande numero di dispositivi integrati usano servizi di comunicazione offerti da protocolli Internet" [5]. I dispositivi sopra citati variano da oggetti come elettrodomestici, lampadine, etc. . . , le cui nuove versioni vengono equipaggiate con funzionalità di comunicazione e interazione esterne, a oggetti appositamente sviluppati (dispositivi come Alexa di Amazon). In particolare, la ITU-T utilizza i seguenti termini per descrivere questi dispositivi [3]:

- **Device** (*Dispositivo*): Un'apparecchiatura con obbligatorie capacità di comunicazione e opzionali capacità di rilevamento, attuazione, cattura dati, conservazione dati ed elaborazione dati.

- **Thing** (*Cosa*): Un oggetto del mondo fisico (cosa fisica) o del mondo dell'informazione (cosa virtuale), capace di essere identificata e integrata nelle reti di comunicazione.

Ai fini dell'elaborato e del progetto Privacy Dashboard, con dispositivo faremo riferimento ad un oggetto fisico in grado di catturare, elaborare e trasmettere dati, sensibili o meno, degli utenti finali.
La struttura del Internet of Things può essere divisa su più livelli; lo IEEE[4] , in particolare, ne individua una struttura a tre livelli, come illustrato nella *Figura* 1.1.

Il layer più basso è quello responsabile di interagire con il mondo fisico e raccogliere i dati. Quello intermedio si occupa della trasmissione, l'elaborazione, classificazione e polimerizzazioni dei dati. Il layer superiore, infine, fornisce i servizi agli

[3]Internet Architecture Board, in italiano: Comitato per l'Architettura Internet. Il IAB fornisce direzioni tecniche a lungo termine per lo sviluppo di Internet [4].

[4]Institute of Electrical and Electronics Engineers, in italiano: Istituto di Ingegneri Elettrici e Elettronici. Lo IEEE la più grande organizzazione di professionisti tecnici del mondo, dedicata ad avanzare la tecnologia per il beneficio dell'umanità [6].

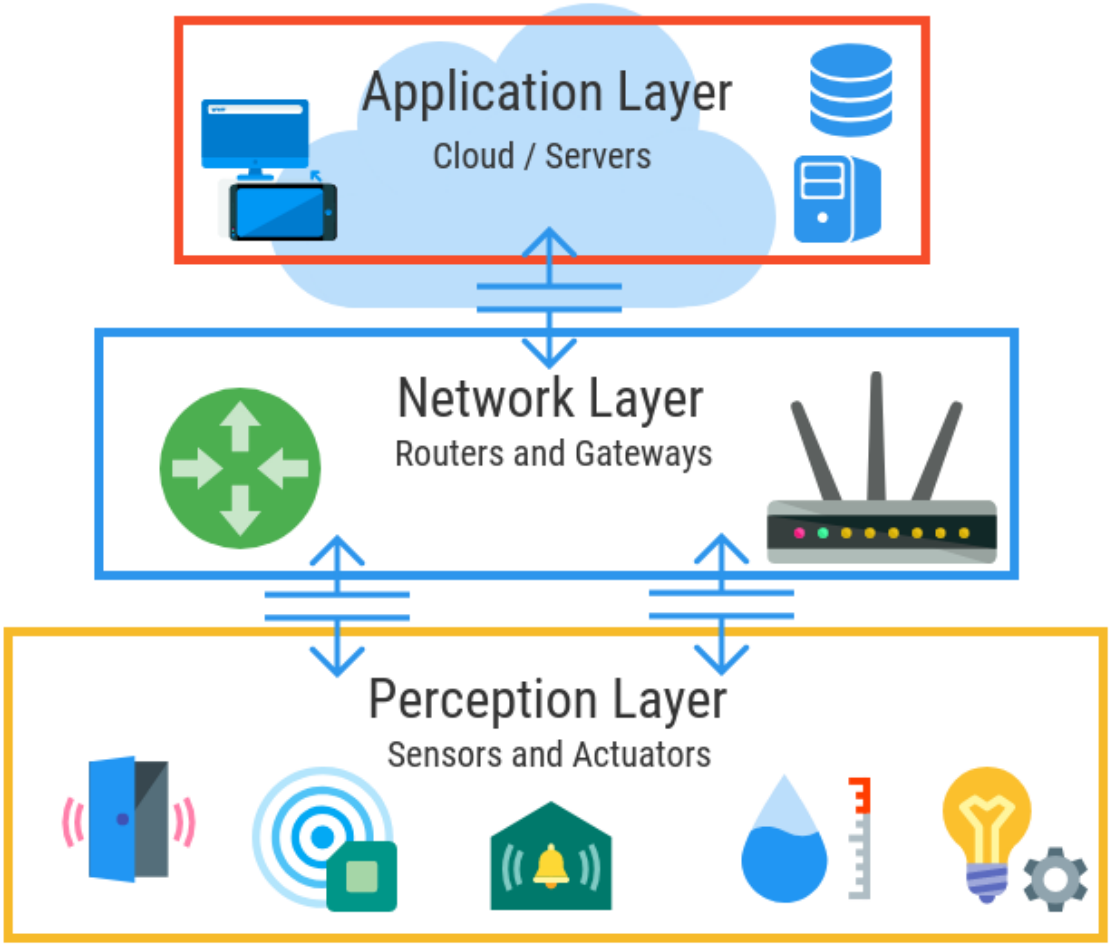


Figura 1.1: Tre livelli del Internet of Things

utenti finali [7].

La struttura del Internet of Things risulta essere molto sensibile agli attacchi informatici. Appoggiandosi all'infrastruttura di una rete (Internet o rete privata che sia) risulta essere di base sensibile ai tipici attacchi che possono colpire una rete. Tuttavia, se normalmente si ha uno scenario di comunicazione bidirezionale tra due utenti o monodirezionale uno a molti (broadcast/multicast), nel IoT si hanno molteplici dispositivi tutti in comunicazione tra di loro.

I singoli dispositivi IoT spesso non sono molto sicuri e, una volta che un attacco

ha successo, garantiscono accesso all'intera rete. Inoltre, il numero sempre in crescita di dispositivi (attualmente 15,14 miliardi, con una stima di 29,42 miliardi nel 2030 [8]) non fa che aumentare il rischio di subire un attacco da parte degli utenti.

1.2 Smart Home

Con Smart Home si indica una particolare applicazione delle tecnologie IoT ai fini di ottimizzare la realtà domestica di una persona. Essa si inserisce nel più ampio contesto della domotica, ovvero lo studio multidisciplinare e l'applicazione di tecnologie volte a migliorare la qualità della vita, il comfort e l'efficienza nei luoghi frequentati dall'uomo [9].
Possiamo identificare alcune principali aree di interesse della Smart Home [9]:

- **Termoregolazione**: Un sistema di climatizzazione intelligente in grado di accendersi, spegnersi e regolarsi in base ad una serie di fattori (ad esempio, la presenza o meno di persone nelle mura).

- **Illuminazione**: Come per il punto superiore, il sistema può accendersi e spegnersi in base alla presenza delle persone nelle singole stanze.

- **Impianto Elettrico**: Il sistema può monitorare il consumo di energia e regolare i carichi di corrente in modo da prevenire i blackout e ridurre i consumi.

- **Sicurezza e Videosorveglianza**: Un impianto di sicurezza smart è in grado di rilevare tempestivamente i pericoli, far scattare gli allarmi e chiamare i servizi di emergenza.

- **Automazione**: Il funzionamento dei vari sistemi sopra elencati può essere in base alle condizioni esterne e alle proprie esigenze.

- **Controllo remoto**: I sistemi possono essere anche controllati in remoto dall'utente.

Si evincono quindi una serie di vantaggi derivanti dall'impiego di un sistema Smart Home, tra cui in primis il risparmio energetico dovuto alla personalizzazione del funzionamento dei vari sistemi domestici, permettendone l'utilizzo solo nei momenti di effettivo bisogno. Un altro vantaggio è la comodità di controllo dei vari elementi sia in remoto che tramite l'installazione di assistenti vocali. Un ultimo vantaggio lo si individua nella sicurezza, con la possibilità di avere sotto controllo l'abitazione 24 ore su 24 [9].
D'altra parte, come già accennato, un'infrastruttura IoT risulta suscettibile ad attacchi informatici a causa del grande numero di dispositivi interconnessi. Ciò risulta essere un pericolo sia per la privacy dei residenti che per la sicurezza fisica degli stessi (possibilità di aprire porte e/o finestre, disattivare allarmi, etc...).
Ai fini della privacy, esaminando i vari servizi di home assistant[5] in circolazione (Alexa di Amazon, Siri di Apple e Google Home) possiamo identificare una serie di problematiche evidenti. Prima di tutto, data la loro natura di assistenti vocali, essi hanno bisogno di registrare i comandi pronunciati dall'utente. Ciò significa che essi sono sempre in attesa della cosiddetta wake word o wake phrase[6] la quale, una volta ricevuta, fa partire la registrazione del comando vocale. Queste registrazioni vengono gestite in modo diverso a seconda del servizio utilizzato [10].
I dispositivi Alexa presentano deboli o nulle capacità elaborative locali, dovendo quindi necessariamente inviare le trascrizioni nel cloud. Questi dati, una volta criptati, vengono associati al proprio account e mantenuti per un tempo indeterminato, a meno di un intervento manuale dell'utente. La loro cancellazione, tuttavia, può

[5]In italiano: Assistente domestico.

[6]In italiano: Parola o Frase d'attivazione.

compromettere il corretto riconoscimento della propria voce o dei comandi impartiti. Per i dispositivi dotati di video camera, Amazon afferma che le elaborazioni avvengono sempre sul dispositivo, eliminando la necessità di salvare i dati. Si è verificato in diverse occasioni però che le registrazioni dei dispositivi Ring di Amazon (videocitofoni smart), venissero fornite alla polizia senza mandato, sebbene solo in caso di emergenza. Inoltre, Amazon farà uso delle vostre abitudini recepite dai propri dispositivi per generare del marketing mirato, come già succede nei confini delle sue app mobile [10].

I servizi offerti da Apple risultano invece essere molto più sicuri e meno invasivi. Come Alexa, anche Siri invia le registrazioni al cloud per essere elaborate, tuttavia i dispositivi più recenti sono in grado gestire alcuni comandi in locale, o persino offline. Le registrazioni inviate al cloud, inoltre, non vengono mantenute per più di 6 mesi e servono per esaminare e migliorare le prestazioni del servizio, e l'utente ha sempre la libertà di chiederne la cancellazione e di non mantenerle in primo luogo. A differenza di Amazon, Apple non usa i dati raccolti dalle proprie app per operazioni di profiling[7] ed è più restia a cedere i dati dei propri utenti, anche alle forze dell'ordine [10].

Google Home e il suo Assistant sono collegati a tutto l'ecosistema Google (telefoni Andorid, Google Chrome, etc...) permettendo il controllo delle varie funzionalità da qualsiasi dispositivo si voglia. Le registrazioni dei dispositivi Google, al contrario degli altri, non vengono salvate come impostazione base, ma si può attivare il loro mantenimento per fini di revisioni. Tuttavia, in merito alla profilazione per gli annunci pubblicitari, Google risulta essere il servizio peggiore, data la sua posizione come colosso della pubblicità online [10].

[7] In italiano: Profilazione. Tecnica di marketing che consiste nell'esaminare le abitudini virtuali e ricerche di un utente per fornire marketing mirato.

1.3 SIFIS-Home

Il progetto SIFIS-Home, o Secure Interoperable Full-Stack Internet of Things for Smart Home, è un progetto co-finanziato tramite il programma Horizon 2020 della Commissione Europea. Lo scopo di questo progetto è la realizzazione di un framework secure-by-design[8] che migliori la sicurezza dei sistemi IoT delle Smart Homes [11].
SIFIS-Home si basa sull'utilizzo di tecnologie e risorse open-source[9] , essendo anch'esso open-source. Questo principio è importante ai fini del progetto per aumentare la fiducia degli utenti nelle capacità del framework [11]. In particolare, un programma è definibile open-source se presenta le seguenti proprietà [12]:

- Libertà di eseguire il programma come si desidera, per qualsiasi scopo.
- Libertà di studiare come funziona il programma e di modificarlo in modo da adattarlo alle proprie necessità (l'acceso al codice sorgente ne è un prerequisito).
- Libertà di ridistribuire copie in modo da aiutare agli altri.
- Libertà di migliorare il programma e distribuirne pubblicamente i miglioramenti apportati in modo che tutta la comunità ne tragga beneficio (l'acceso al codice sorgente ne è un prerequisito).

Un programma che presenta queste proprietà è definibile open-source, o software libero.
L'architettura SIFIS-Home è composta da due parti:

[8] In italiano: Sicuro in progettazione. Tecnica di progettazione software che prevede che la sicurezza informatica dell'applicativo faccia parte del design base, rendendola più solida.

[9] In italiano: Sorgente Aperta. Distribuire un software sotto questa licenza ne permette lo studio, modifica e distribuzione libera.

- Un insieme di strumenti di sviluppo che permette agli sviluppatori di realizzare applicazioni in grado di sfruttare il framework SIFIS-Home per implementare funzioni di sicurezza.

- Il framework che utilizza funzionalità di comunicazione ed elaborazioni dati sicure e rispettose della privacy.

1.3.1 Conformità con il GDPR

Dato l'obiettivo del framework SIFIS-Home di migliorare la sicurezza delle applicazioni IoT, esso si trova anche a dover gestire i dati degli utenti finali. Per questo motivo, è necessario rispettare i principi di privacy esposti dagli articoli del GDPR[10] [11]. Il GDPR è un regolamento europeo che stabilisce i diritti dei cittadini UE in merito al trattamento dei propri dati, in particolare in merito alla raccolta, circolazione e uso di questi dati [13].
Identifichiamo subito due definizioni importanti: i data subjects e i loro personal data. Un data subject è una persona identificata o identificabile, direttamente o indirettamente, tramite un identificativo, posizioni o più elementi caratteristici della sua persona (cultura, posizione socioeconomica, etc...). Con personal data si indica una qualsiasi informazione riguardante un data subject [14]. Il data subject deve fornire il proprio consenso per il trattamento dei suoi dati, ed è libero di revocarlo in un qualsiasi momento [15].
Quando si sviluppa un software, gli sviluppatori non sono direttamente obbligati a seguire le regole del GDPR, in quanto è il trattamento dei dati personali a ricadere sotto le regolamentazioni europee [11]. Con trattamento si intende una qualsiasi operazione, o insieme di esse, applicate su dati personali, o un insieme di essi [14]. La figura che si occupa del trattamento dei dati personali è detta data controller [14].

[10]General Data Protection Regulation, in italiano: Regolamento Generale sulla Protezione Dati.

Essa è responsabile e deve dimostrare la conformità del trattamento rispetto all'articolo 5 del GDPR, il quale afferma che i dati personali devono essere trattati in modo lecito, corretto e trasparente nei confronti del data subject e che essi non siano trattenuti per più tempo di quanto non sia necessario [16].
Per assicurarsi che il data subject sia correttamente informato del trattamento dei suoi dati, all'interno del framework SIFIS-Home un data controller può creare un apposito documento, accessibile dal data subject, chiamato privacy notice [11]. La creazione di un privacy notice è basata sulle indicazioni degli articoli 13 e 14 del GDPR, i quali stabiliscono che il data subject debba essere informato quando dati personali vengono acquisiti direttamente (art. 13) o indirettamente, da terze parti (art. 14) [11, 17, 18]. Il privacy notice è parte integrante del funzionamento del framework SIFIS-Home e dell'applicativo Privacy Dashboard come verrà esposto a breve.
Un'ultima figura integrale è quella del Data Protection Officer[11], il cui ruolo è quello di assicurarsi che un'organizzazione, e per estensione i data controller, trattino i dati personali dei data subject in conformità con le leggi in vigore [11, 19–21].

1.4 Privacy Dashboard

La Privacy Dashboard è un applicativo web che ha come scopo il facilitare la gestione del GDPR nelle applicazioni IoT per Smart Home. Essa è stata pensata come un gestore unificato di informazioni di privacy e strumenti di comunicazione, permettendo sia ai data subjects che ai data controllers di avere un unico spazio digitale dove poter visionare tutte le informazioni di cui necessitano o per esercitare

[11] In seguito si userà l'abbreviazione DPO.

i propri diritti o per assicurare il rispetto del GDPR [11].

1.4.1 Utenti e Ruoli

Come già accennato in precedenza, le tre figure, o come sono note nell'applicativo "ruoli", coinvolte dal framework SIFIS-Home (e quindi dalla Privacy Dashboard) sono data subject, data controller e data protection officer, ognuna delle quali, in aggiunta alle funzionalità comuni, ne presenta alcune riservate. Le vere differenze, però, si presentano tra i data subject e gli altri due ruoli.
I data subject possono, come accennato in precedenza, esercitare i propri diritti di accesso e cancellazione dei dati, tramite l'invio di richieste di diritti, conosciute nell'app come right request [11]. Ciò è supportato da un sistema di messaggistica interna all'applicativo, che mette in comunicazione i data subject con i data controller e DPO delle App dei propri dispositivi IoT. Inoltre, essi potranno anche gestire l'aggiunta o rimozione di App direttamente all'interno dell'applicativo.
La funzione più importante dei data controller e dei DPO è quella di creare il privacy notice. Essi sono anche in grado di rispondere ai messaggi dei data subjects e gestire le loro right request in merito alle App di cui sono responsabili. Infine, possono compilare un questionario per le proprie App che mira a valutarne la conformità al GDPR [11].

1.4.2 App

Sia i data subject che i data controller e i DPO possono visionare tutte le app a loro associate. I primi vedono le App a cui hanno dato il consenso per la raccolta dati, i secondi vedono le app di cui sono responsabili [11]. Le app, inoltre, presentano una valutazione su tre livelli (sistema a semaforo) che ne esprime il rispetto per le

regole del GDPR e la bontà del trattamento dei dati [11].
I data controller e i DPO sono in grado di compilare un questionario per effettuare la valutazione di un app. Le domande riguardano obblighi del GDPR, standard d'industria sulla privacy e aderenza alle licenze di software open source [11].
I data subject hanno la possibilità di visionare anche un elenco di App disponibili per il monitoraggio all'interno della Privacy Dashboard.

1.4.3 Privacy Notice

Il privacy notice, come accennato prima, è un documento che comunica al data subject tutte le informazioni in merito alla raccolta, uso e mantenimento dei suoi dati personali da parte di una sua app. Esso viene compilato dal data controller o DPO assicurandosi che esso sia scritto in una forma di facile comprensione e che sia consegnato al data subject quando i suoi dati vengono trattati [11, 17, 22].
All'interno della Privacy Dashboard si rende possibile per i data controller e DPO una compilazione guidata del documento, seguendo un template fornito dal GDPR stesso [23]. In questo modo ci si assicura che tutti le informazioni necessarie per l'esercizio dei diritti del data subject siano raccolte all'interno del privacy notice, permettendo al data controller e DPO di svolgere un lavoro più rapido e completo [11]. Questo tipo di soluzione è utile anche per i data subject, in quanto consente loro di accedere a queste informazioni in modo semplice e chiaro. Si rende anche possibile per i data subject l'esercizio dei propri diritti di revoca [11, 15].

1.4.4 Implementazione

La Privacy Dashboard è stata realizzata usando come linguaggio base Java, sfruttando il framework open source full-stack[12] Vaadin Flow [24] e il framework Spring Security [25]. Il primo mette a disposizione una serie di componenti UI[13] proprietari sotto forma di classi, così che possano essere personalizzati e integrati secondo le esigenze del programmatore, risultando più pratico rispetto ad una realizzazione più classica con HTML5; il secondo, invece, permette di gestire in modo semplice l'autorizzazione e autenticazione degli utenti, rendendo disponibili le funzionalità dell'applicativo in base al ruolo dell'utente che ha fatto l'accesso.
La Privacy Dashboard può essere suddivisa in 3 package:

- **Data**: In questo package sono presenti sia le classi rappresentative delle entità della Privacy Dashboard sia tutte le funzionalità di gestione di questi, a partire dalle interazioni con il database a più basso livello, fino alle API[14] che vengono chiamate dalle pagine del front-end.

- **Security**: Questo package contiene le classi che gestiscono le funzionalità di autenticazione e autorizzazione dell'utente. Esse permettono di effettuare l'autenticazione sia tramite le classiche funzionalità di log in, sia con l'uso di un token di autenticazione inserito nella richiesta http.

- **Views**: In questo package troviamo le classi responsabili del front-end e di tutte le sue interazioni. Esse interagiscono tra di loro secondo un paradigma

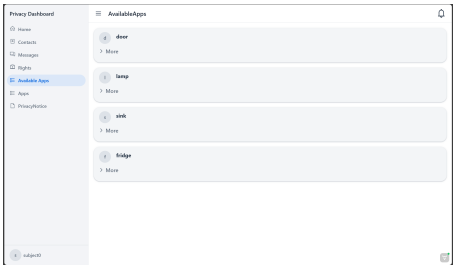
[12]Con full-stack si intende che il framework è in grado di realizzare sia il front-end che il back-end.

[13]User Interface, in italiano: Interfaccia Utente.

[14]Application Programming Interface, in italiano: Interfaccia di Programmazione dell'Applicazione. Procedure che permettono la comunicazione tra diversi software o componenti hardware. Nello sviluppo web, indicano le funzioni che mettono in comunicazione il front-end con il back-end.

di design per cui una cornice esterna, in questo caso una barra di navigazione laterale verticale, della pagina è costante, mentre uno spazio interno[15] presenta il contenuto caratteristico della pagina corrente.

(a) Vista della pagina Home



(b) Vista della pagina Available Apps

Figura 1.2: Un paio di viste della Privacy Dashboard

[15] Detto outlet

Capitolo 2

Testing e Integrazione Continua

2.1 Ingegneria del Software

L'ingegneria del software è una disciplina che riguarda tutti gli aspetti della produzione del software: dalle prime fasi della specifica di un sistema fino alla sua manutenzione dopo il rilascio [26]. Un'altra definizione, proposta dalla IEEE, la descrive come l'approccio sistematico allo sviluppo, uso, manutenzione e smaltimento (ritiro) del software [27]. Questo approccio sistematico è anche detto processo software. L'ingegneria del software nel lungo termine è solitamente più conveniente a livello economico, in quanto un mancato utilizzo dei suoi metodi comporta costi più elevati per il test, la garanzia di qualità e la manutenzione del software [26].
Un processo software è un insieme di attività che porta alla creazione di un prodotto software. Possiamo identificare quattro attività fondamentali che fanno parte di un processo software [26]:

1. **Specifiche del software**: Si definiscono le funzionalità e i vincoli operativi del software da produrre.

2. **Sviluppo del software**: Viene progettato e pianificato il software.

3. **Convalida del software**: Si convalida il software per garantire che rispetti le specifiche definite al punto 1).

4. **Evoluzione del software**: Il software viene modificato per soddisfare eventuali cambiamenti dei requisiti (del cliente o del mercato) e correggere errori.

2.1.1 Convalida del software

Poniamo maggiore enfasi sull'attività di convalida del software, all'interno della quale si colloca l'elaborato. Il processo di convalida del software è costituito da una serie di attività di supporto:

- **Verifica**: Controlla che il software sia costruito correttamente tramite l'uso di ispezioni e testing.

- **Validazione**: Controlla che il software rispetti le specifiche.

- **Gestione della configurazione**: Definisce gli elementi del sistema, il loro rilascio e i loro cambiamenti.

- **Gestione del progetto**: Gestisce le risorse per rispettare budget e scadenze.

- **Controllo qualità**: Si assicura che il software rispetti determinati standard di qualità.

I test sono una tecnica di verifica definita dinamica, poiché richiede l'esecuzione del codice per essere svolta. I test possono rivelare la presenza di errori, ma non confermarne l'assenza [28]. Individuiamo due varianti di test:

- **Black box testing** (*a scatola nera*): Si ignorano i meccanismi interni di un sistema o componente e si osserva l'output generato in risposta a input e condizioni di esecuzione selezionate.

- **White box testing** (*a scatola chiara/trasparente*): Si prendono in considerazione i meccanismi interni di un sistema o componente, assicurandosi che tutte le parti del codice siano raggiungibili ed eseguibili.

Le ispezioni sono una tecnica di verifica statica, che si basa sull'esaminazione visiva dello sviluppo del prodotto per trovare errori, violazioni degli standard e altre problematiche. È definita statica poiché non richiede l'esecuzione del codice, a differenza dei test, che invece sono dinamici. Nelle ispezioni, gli ispettori esaminano i requisiti del sistema, gli schemi di progettazione, il codice sorgente dei programmi e perfino i test proposti per il sistema [26].
L'ispezione del software ha 3 vantaggi rispetto al test [26]:

- Durante i test, gli errori possono mascherare altri errori. Quando un errore determina output inaspettati, non si può sapere se altri risultati anomali siano dipendenti o meno dall'errore originario.

- Le versioni incomplete di un sistema possono essere ispezionate senza costi aggiuntivi.

- Oltre a cercare i difetti di un programma, un'ispezione può anche valutare attributi di qualità più generali.

Le ispezioni, tuttavia, non possono rimpiazzare i test del software. Esse non sono adatte a scoprire i difetti che si presentano a causa di interazioni inaspettate tra varie parti di un programma, i problemi di tempistica o quelli relativi alle prestazioni del sistema.
La *Figura* 2.1 rappresenta un'astrazione del tradizionale processo di prova del software; sono evidenziate in giallo le fasi del processo, mentre in blu sono evidenziati i documenti in input e output alle varie fasi. Con test case indichiamo le specifiche degli input dei test e degli output previsti dal sistema, più una descrizione di ciò che si sta provando. I dati di test sono gli input che sono stati previsti per la verifica

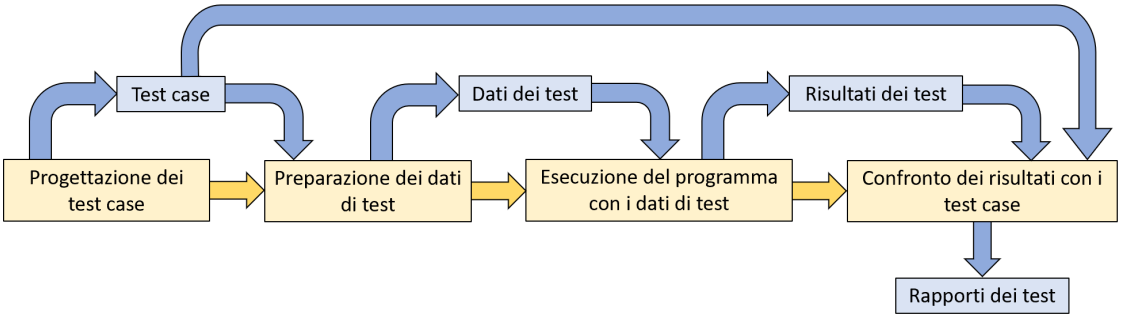


Figura 2.1: Modello di processo di test del software

del sistema. I risultati vengono automaticamente confrontati con quelli previsti, tramite l'uso di apposite funzionalità del linguaggio o dell'ambiente di sviluppo. Tipicamente, un sistema software commerciale è soggetto a tre stadi di test [26]:

1. **Test dello sviluppo**: Il sistema viene provato durante lo sviluppo per scoprire bug e difetti.

2. **Test della release**: Un apposito team prova una versione completa del sistema prima che sia rilasciata agli utenti.

3. **Test degli utenti**: Gli utenti, o i potenziali utenti, di un sistema provano il sistema nei loro ambienti.

Il processo di test di solito consiste in un mix di test manuali e automatici. Nei test manuali, l'esecutore dei test, chiamato tester, avvia un programma con alcuni dati di prova e confronta i risultati con quelli previsti, per poi evidenziare le discrepanze per gli sviluppatori del programma. I test automatici vengono scritti e raccolti all'interno di una test suite, che viene eseguita ogni volta che il software deve essere provato. Questo procedimento è più veloce di quello manuale, specialmente quando è necessario effettuare dei test di regressione . I test, tuttavia, non possono mai essere completamente automatici, in quanto test automatizzati non possono verificare gli elementi del sistema dal punto di vista visivo [26].

2.1.2 Test delle Unità

Ai fini di illustrare il lavoro svolto nell'elaborato, esaminiamo i test di sviluppo. Al loro interno individuiamo tre categorie di testing [26]:

- **Test delle unità**: Vengono provate singole unità di programma o classi di oggetti, per verificarne le funzionalità. Vengono eseguiti con una metodologia white box.

- **Test di integrazione**: Vengono integrati più unità di programma per creare componenti più complessi per verificare le funzionalità di un sottosistema. Questi test sono eseguiti possono essere eseguiti sia in modalità white box che black box, a seconda della fase di sviluppo.

- **Test di sistema**: I componenti del sistema vengono integrati e il sistema viene provato a scatola chiusa (black box) per verificare le interazioni tra i componenti.

I test svolti all'interno dell'elaborato sono test delle unità, costruiti secondo la metodologia white box. Come accennato sopra, questi test verificano i componenti di un programma, come i metodi o le classi di oggetti. I tipi più semplici di componenti sono le singole funzioni o i singoli metodi di un programma. I test dovrebbero eseguire queste routine con differenti parametri di input. [26]. Per realizzare ciò, vengono utilizzate una serie di metodi che creano oggetti da poter dare in input ai metodi e costruttori da testare.
I test vanno progettati in modo da verificare tutte le funzioni di ciascun oggetto. Ciò vuol dire testare tutte le operazioni associate ad esso: impostare e controllare il valore di tutti i suoi attributi e porre l'oggetto in tutti i possibili stati. Occorre quindi simulare tutti gli eventi che provocano un cambiamento di stato dell'oggetto [26].
Un test automatico è composto da tre parti [26]:

1. **Parte di impostazione**: Viene inizializzato il sistema con il test case, ovvero gli input e output previsti.

2. **Parte della chiamata**: Viene chiamato l'oggetto o il metodo da testare.

3. **Parte dell'asserzione**: Viene confrontato il risultato della chiamata con il risultato previsto. Se l'asserzione è vera, il test ha avuto successo; se è falsa, il test è fallito.

Per automatizzare i test si usano dei framework di automazione. All'interno dell'elaborato è stato utilizzato il framework JUnit5, e in particolare il modulo JUnit Jupiter [29].

2.1.3 Oggetti Mock e Riflessione

A volte, l'oggetto che si sta testando ha delle dipendenze con altri oggetti che potrebbero non essere stati implementati, il cui uso potrebbe rallentare il processo di test o, nel caso di test di unità, di cui non interessa testarne le funzionalità. In questi casi è opportuno utilizzare oggetti mock (*finti*) [26].
Gli oggetti mock presentano la stessa interfaccia degli oggetti esterni realmente utilizzati e che simulano le loro funzionalità. I framework che permettono l'uso di oggetti mock permettono di "personalizzare" il loro comportamento affinché essi possano essere utilizzati. Si devono specificare i metodi che verranno chiamati (a volte anche l'ordine), i parametri che essi riceveranno e l'output che dovranno produrre.
Nell'elaborato, per sfruttare le funzionalità degli oggetti mock durante il testing è stato utilizzato il framework Mockito, e in particolare i seguenti metodi [30]:

- **mock**: Permette la creazione di un oggetto mock.

- **when**: Permette di specificare cosa accade alla chiamata di un metodo dell'oggetto mock. Deve essere seguito da uno dei metodi "then" per specificare l'esito della chiamata.

- **any**: Permette all'oggetto mock di eseguire il comportamento specificato dal metodo "when" a prescindere dal valore dei parametri in input.

```
@Test
public void createPrivacyNoticeCardSubjectTest(){
    dataBaseService = mock(classToMock: DataBaseService.class);
    authenticatedUser = mock(classToMock: AuthenticatedUser.class);
    communicationService = mock(classToMock: CommunicationService.class);

    when(authenticatedUser.getUser()).thenReturn(createUser(Role.SUBJECT));
    when(dataBaseService.getControllersFromApp(any())).thenReturn(List.of(createUser(Role.CONTROLLER)));
    when(dataBaseService.getDPOsFromApp(any())).thenReturn(List.of(createUser(Role.DPO)));

    view = new PrivacyNoticeView(dataBaseService, authenticatedUser, communicationService);
```

Figura 2.2: Esempio d'uso del framework Mockito

La riflessione è l'abilità del codice di esaminare e modificare parti del proprio comportamento e struttura. Essa viene spesso usata durante il processo di testing, con modalità affini a quelle degli oggetti mock. In particolare nel linguaggio Java, è possibile usare la riflessione per aggirare l'incapsulamento[1] degli attributi e dei metodi di una classe. Questo risulta utile per testare metodi privati singolarmente e verificare che attributi privati possiedano valori coerenti con il flusso dell'esecuzione. All'interno dell'elaborato, per implementare le funzionalità di riflessione è stata usata la libreria di Java "java.lang.reflect" [31].

2.1.4 Code Coverage

Un'altra metrica importante, legata ai test delle unità è la code coverage . Essa aiuta a capire quanto del source code viene testato, utile per valutare la qualità

[1]Paradigma dei linguaggi orientati agli oggetti che limita l'accesso agli attributi a seconda del luogo in cui avviene la richesta d'accesso.

della test suite [32]. In teoria, ogni segmento di codice dovrebbe avere almeno un test associato. Quindi, si può essere sicuri che tutto il codice nel sistema è stato effettivamente eseguito [26].
Non esiste, tuttavia, una soluzione ottimale nella code coverage e una percentuale elevata di coverage potrebbe essere problematica se le parti critiche dell'applicazione non venissero testate o se i test esistenti non fossero efficaci nel rilevare correttamente gli errori [32]. Generalmente, si punta ad una coverage dell'80%.
Esistono diverse tipologie di coverage [32]:

- **Copertura delle funzioni**: Quante delle funzioni definite sono state richiamate.

- **Copertura delle istruzioni**: Quante istruzioni del programma sono state eseguite.

- **Copertura dei branch**: Quanti branch delle strutture di controllo sono state eseguiti.

- **Copertura delle condizioni**: Quante sotto espressioni booleane sono state testate per un valore vero e uno falso.

- **Copertura delle righe**: Quante righe di codice sorgente sono state testate.

Come obiettivo dell'elaborato, è stato deciso di raggiungere una copertura delle istruzioni di almeno il 60%. Per visionare la copertura raggiunta ho usato la libreria open source JaCoCo [33].

Privacy Dashboard

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.privacydashboard.application.data.generator		0%		0%	46	46	253	253	3	3	1	1
com.privacydashboard.application.data.service		0%		0%	129	129	296	296	92	92	3	3
com.privacydashboard.application.views.privacyNotice		47%		22%	68	82	197	329	44	51	1	3
com.privacydashboard.application.views.questionnaire		80%		22%	53	76	137	467	21	36	0	2
com.privacydashboard.application.views.apps		62%		49%	75	118	166	439	33	53	0	3
com.privacydashboard.application.views.mainLayout		0%		0%	49	49	152	152	31	31	2	2
com.privacydashboard.application.views.rights		69%		52%	68	123	162	460	37	63	2	6
com.privacydashboard.application.security		9%		18%	44	51	148	163	35	40	5	6
com.privacydashboard.application.views		0%		0%	18	18	78	78	13	13	3	3
com.privacydashboard.application.data.apiController		93%		89%	55	402	56	1.001	13	114	1	9
com.privacydashboard.application.views.login		0%		0%	14	14	49	49	10	10	2	2
com.privacydashboard.application.views.contacts		73%		22%	17	24	19	68	9	15	0	1
com.privacydashboard.application.views.messages		84%		37%	17	31	20	113	10	23	0	2
com.privacydashboard.application.views.usefulComponents		87%		n/a	5	18	6	44	5	18	1	3
com.privacydashboard.application.views.home		94%		83%	3	9	3	43	2	6	0	1
com.privacydashboard.application		0%		n/a	2	2	3	3	2	2	1	1
com.privacydashboard.application.data		98%		n/a	1	6	1	16	1	6	0	5
com.privacydashboard.application.data.entity		100%		100%	0	87	0	129	0	84	0	8
Total	9.084 of 23.093	60%	538 of 1.226	56%	664	1.285	1.746	4.103	361	660	22	61

Figura 2.3: Report della code coverage

2.1.5 Integrazione Continua

L'integrazione continua (Continuous Integration, o CI) è un processo di sviluppo software in cui gli sviluppatori integrano il nuovo codice che hanno scritto più frequentemente durante tutto il ciclo di sviluppo, anche più volte al giorno. Ad ogni iterazione della build, vengono eseguiti dei test automatici per identificare eventuali problemi di integrazione prima che si accumulino [34].

Come accennato, uno dei vantaggi della CI è il rilevamento precoce di bug. Il rilevamento di un bug a breve distanza dalla sua introduzione rende anche più semplice la sua correzione, in quanto le modifiche apportate sono poche e fresche nella memoria dello sviluppatore. Di conseguenza, progetti che vengono sviluppati secondo un processo di CI, in media, tendono ad avere meno bug. L'efficacia di questo processo, tuttavia, dipende dall bontà della test suite [35].

Un processo che segue l'integrazione è la fornitura continua (Continuous Delivery, o CD), che si concentra sulla fornitura agli utenti di eventuali aggiornamenti, correzioni di errori e nuove funzioni nel modo più rapido e sicuro possibile [34].

L'uso congiunto di CI e CD aiuta a rimuovere una grossa a barriera alla tecnica del rilascio frequente (frequent delivery). Esso è importante perché permette agli utenti di ricevere nuove funzionalità più rapidamente, fornendo così un feedback

immediato su di esse e diventando più coinvolti nel processo di sviluppo [35].
Per poter sfruttare i processi di CI e CD è necessario l'uso di un framework o applicativo apposito. All'interno dell'elaborato è stata usata la funzionalità di GitHub delle Actions. Essa è una piattaforma per la CI e la CD che permette di creare dei workflow che testano e creano la build di un progetto, o ne rilasciano una versione, all'avvenimento di determinati eventi [36].
Una Action può essere configurata affinché si avvii al verificarsi di un dato evento nella repository[2], come una richiesta di pull o l'apertura di una issue. Il workflow di una action può contenere uno o più jobs, che possono essere eseguiti in parallelo o in sequenza. Ognuno di questi job sarà eseguito all'interno di una macchina virtuale e sarà composta da una serie di step (almeno uno), i quali eseguiranno uno script o una azione, un'estensione che esegue una serie di istruzioni quando chiamata [36].

2.2 Test del Package Views

Come già accennato nel capitolo 1.4.4, il package Views contiene le classi che gestiscono la parte estestica, le chiamate alle API e le funzionalità di navigazione delle pagine. Le classi di questo package sono state testate verificando che:

- I costruttori creino il layout della pagina in modo corretto a seconda dei parametri inseriti in input.

- Le altre funzionalità presenti nella classe funzionino correttamente.

La totalità dei test del package Views può essere visionata al seguente indirizzo: *https://github.com/sifis-home/ privacydashboard/tree/master/ src/test/java/com/ privacydashboard/application/views.*

[2]In italiano: Repositorio. Luogo virtuale in cui viene conservato il codice in modo sicuro.

2.2.1 Test di HomeView.java

La classe HomeView.java si occupa di creare la pagina iniziale dell'applicativo. Essa presenta due principali funzionalità: la creazione di bottoni (chiamati section nel codice) e la loro disposizione all'interno del layout della pagina. Le section create differiscono in parte a seconda del ruolo dell'utente attuale: se l'utente è un subject, esso è in grado di accedere alle pagine per richiedere l'esercizio dei propri diritti e visualizzare lo stato delle proprie richieste; se l'utente è un controller o un DPO, esso è in grado di accedere alle pagine per rispondere alle richieste dei subject e per compilare il questionario per una delle proprie app.
A tale scopo sono stati realizzati due test, *HomeViewSubjectConstructorTest* e *HomeViewNotSubjectConstructorTest*, che verificano sia la costruzione totale della pagina che le differenze dovute al ruolo.

2.2.2 Test di AppsView.java e AvailableAppsView.java

La classe AppsView.java si occupa di creare la pagina all'interno della quale gli utenti possono visualizzare le proprie App. La principale funzionalità è proprio quella di creare delle card per la visualizzazione delle App e delle loro caratteristiche. Per realizzare ciò vengono utilizzate una serie di funzioni, ognuna delle quali si occupa di costruire una parte specifica della card.
Dato che alcuni di questi metodi sono dichiarati come private, nella fase di testing si è deciso di testare, oltre al costruttore, il metodo privato più esterno, responsabilie dell'effettiva creazione della card nella sua totalità, ovvero *createApp*. Inoltre, parti dell'interfaccia e delle card variano a seconda delle valutazioni singole o complessive delle App.
A tale scopo sono stati creati dei test (*redAppViewTest*, *greenAppViewTest*, *createAppRedTest* e *createAppOrangeTest*) il cui unico obiettivo è quello di verificare che la specifica sezione che varia in base alla valutazione cambi coerentemente.

Simile alla classe AppsView.java è AvailableAppsView.java. Essa, sfruttando la stessa struttura, visualizza le app che il subject può installare. Questa presenta alcune differenze nei dati esibiti, tuttavia i test base della classe AppsViewTest.java erano sufficienti per coprire le novità introdotte da questo codice.

2.2.3 Test di ContactsView.java

La classe ContactsView.java è responsabile della creazione della pagina all'interno della quale l'utente può visionare l'elenco dei propri contatti. La principale funzionalità di questa pagina è quella di creare delle card dei contatti dell'utente loggato. Come per AppsViews.java, per realizzare la card vengono utilizzate una serie di funzioni per costruire la totalità della card.
Per testare le funzionalità dell'app sono stati creati due test, *ContactsViewConstructorTest* e *createContactTest*, che verificano rispettivamente la corretta creazione del layout della pagina e la corretta creazione della card del contatto.

2.2.4 Test di MessagesView.java

La classe MessagesView.java è responsabile di mostrare le varie conversazioni che l'utente sta intrattenendo con altri utenti. In questa pagina vengono mostrati solo i contatti degli utenti con cui si sta messaggiando e cliccando su di essi si passerà alla pagina effettiva della convesazione. E'presente anche un bottone per la creazione di una nuova conversazione con un utente.
Per testare questa classe sono stati realizzati tre test: uno che verifica il layout della pagina, *MessagesViewConstructorTest*, uno che verifica la creazione della card di un contatto di un utente, *showContactTest*, e uno che verifica la finestra di nuova conversazione.

2.2.5 Test di SingleConversationView.java

La classe SingleConversationView.java è responsabile per la visualizzazione di una conversazione fra due utenti. In questa pagina vengono visualizzati tutti i messaggi inviati dai due utenti ed è presente la possibilità di inviare nuovi messaggi.
Sono stati creati quindi una serie di test per verificare che la pagina venisse costruita correttamente, *SingleConversationViewConstructorTest.java*, che recuperasse i dati del contatto correttamente, *beforeEnterTest*, che visualizzi i messaggi coerentemente a seconda del mittente, *showMessageSenderIsAuthenticatedTest* e *showMessageSenderIsNotAuthenticatedTest*, e che invii i messaggi correttamente, *sendMessageTest*.

2.2.6 Test di PrivacyNoticeView.java

La classe PrivacyNoticeView.java è responsabile della creazione della pagina dove gli utenti possono accedere alle privacy notice delle proprie app. Nel caso in cui l'utente sia un data controller o un DPO, esso potrà anche creare nuovi privacy notice o apportare modifiche a quelli esistenti.
Per questo motivo sono stati creati una serie di test che verificano la corretta costruzione della pagina, a seconda del ruolo dell'utente, ad es. *createPrivacyNoticeCardSubjectTest* e *createPrivacyNoticeCardNotASubjectTest*.

2.2.7 Test di FormPrivacyNotice.java

La classe FormPrivacyNotice.java è responsabile della pagina in cui data controller e DPO possono creare un privacy notice, rispondendo alle domande del template del GDPR [23].
A tale scopo è stato realizzato un test che verifica la corretta costruzione della pagina e delle domande.

2.2.8 Test di Questionnaire.java

La classe Questionnaire.java è responsabile della creazione della pagina nella quale i data controller e i DPO possono visualizzare le app per le quali hanno compilato, o meno, il questionario di valutazione della pagina. Le card delle app la cui valutazione è già stata effettuata, inoltre, vengono distinte dal colore della valutazione. Per effettuare queste verifiche sono stati creati una serie di test che verificano la corretta creazione delle card delle app a seconda delle valutazioni presenti e due test che verificano il corretto layout della pagina e la corretta creazione del pulsante per la compilazione di un nuovo questionario.

2.2.9 Test di SingleQuestionnaire.java

La classe SingleQuestionnaire.java è responsabile della creazione della pagina dove viene visualizzato il questionario di valutazione da compilare. La principale funzionalità della pagina è, quindi, quella di combinare le varie domande e risposte e presentarle in modo ordinato.
I test di questa classe verificano dunque che le varie stringhe di testo vengano assegnate correttamente.

2.2.10 Test di ControllerDPORightsView.java

La classe ControllerDPORightsView.java è responsabile della creazione della pagina dove i data controller e DPO possono visualizzare le richieste di esercizio dei diritti e rispondere ad esse, visualizzandole sotto forma di card (realizzate grazie alla classe GridComponenetRightView.java descritta in 2.2.12).
Data l'elevata dipendenza dalle interazioni con il database, è stato realizzato solo un test di unità che verifica la struttura del layout della pagina, *controllerDPO-ConstructorTest*.

2.2.11 Test di SubjectsRightView.java

La classe SubjectRightView.java è responsabile della creazione della pagina dove i data subject possono esercitare i propri diritti mandando delle richieste ai data controller e/o DPO delle App interessate, visualizzandole sotto forma di card (realizzate grazie alla classe GridComponenetRightView.java descritta in 2.2.12).
Data l'elevata dipendenza dalle interazioni con il database, è stato realizzato solo un test di unità che verifica la struttura del layout della pagina, *subjectRightsViewConstructorTest*.

2.2.12 Test di GridComponentRightView.java

La classe GridComponentRightView.java è responsabile della creazione di card riassuntive sulle richieste di diritti effettuate, permettendo anche ai data controller e DPO di rispondere ad esse.
A tale scopo, sono stati creati una serie di test che verificano la corretta costruzione delle card, verificando anche che la classe risponda correttamente ad input errati o non completi.

2.2.13 Test di DialogRight.java

La classe DialogRight.java è responsabile della creazione di box di dialogo che mostrano informazioni sull'utente e la richiesta di diritti effettuata. La principale funzionalità della pagina è quella di costruire un oggetto MyDialog (descritto nel 2.2.14) con le informazioni relative al diritto che si vuole esercitare, l'utente che lo richiede e l'app interessata.
A tale scopo, sono stati realizzati una serie di test che verificano la corretta costruzione dell'oggetto, verificando anche che la classe risponda coerentemente ad input errati o non completi.

2.2.14 Test di MyDialog.java

La classe MyDialog.java è una classe di servizio grazie alle quale è possibile richiamare dei box di dialogo nell'applicativo.
Per testarne le funzionalità sono state realizzate una serie di test che verificano la corretta inizializzazione delle varie parti dell'oggetto.

2.2.15 Test di ToggleButton.java

La classe ToggleButton.java è una classe di servizio che permette la creazione rapida di bottoni nell'applicativo.
Sono stati creati una serie di test per verificare la corretta inizializzazione degli attributti dell'oggetto.

2.3 Test del Package Data

Come già accennato nel capitolo 1.4.4, il package Data contiene le classi che rappresentano le entità contenute nel database e le API che realizzano la comunicazione tra il database e il front-end, implementando le operazioni CRUD[3]. Le classi di questo package sono state testate verificando che:

- I metodi restituiscano un output coerente con la documentazione del metodo.
- I metodi rispondano in modo coerente con la documentazione ad input errati o non completi.

La totalità dei test del package Data può essere visionata al seguente indirizzo: *https://github.com/sifis-home/ privacydashboard/tree/master/ src/test/java/com/ privacydashboard/application/data.*

[3]Create, Read, Update and Remove, in italiano: Crea, leggi, Modifica e Rimuovi. Molte delle entità, tuttavia, non presentano la possibilità di essere modificate, o la presentano parzialmente.

2.3.1 Test delle Entity

Le classi del gruppo entity sono una raccolta di classi che astraggono gli oggetti contenuti nel database e sono alla base delle varie funzionalità dell'applicativo. Esse presentano esclusivamente delle funzionalità di assegnazione e lettura degli attributi. Di conseguenza la suite di test è limitata e estremamente diversa dagli altri test del package Data, verificando esclusivamente la corretta assegnazione degli attributi.

2.3.2 Test di ApiGeneralController.java

La classe ApiGeneralController.java è una classe di servizio che raccoglie una serie di API di manipolazione delle entità o di accesso atomico al database. Nella prima categoria riscontriamo una serie di metodi a coppia[4] il cui compito è quello di costruire una stringa JSON da una entità e viceversa, risultando così metodi inversi tra loro[5]; la seconda categoria presenta dei metodi il cui scopo è recuperare un oggetto dal database tramite il loro identificativo unico. Nella classe sono anche presenti alcuni metodi che eseguono dei controlli che sono ripetuti tra vari metodi della classe (ad esempio, il controllo del ruolo dell'utente attuale, tramite il metodo *isControllerOrDpo*).

La prima categoria di metodi è stata testata verificando che gli output fossero costruiti correttamente, ispezionando anche che i metodi operassero in modo tale da essere inversi. Si è testato anche che i metodi funzionassero correttamente secondo la documentazione riportata, quindi verificando che funzionassero con il minimo di parametri necessari e che lanciassero le corrette eccezioni in risposta agli errori indicati.

[4]Tecnicamente i metodi sarebbero un ternario, tuttavia uno di essi è nascosto dietro un altro, rendendoli a tutti gli effetti un solo metodo.

[5]Eseguire l'uno sull'output dell'altro dovrebbe restituire l'input iniziale.

La seconda categoria di metodi è stata testata con tre tipologie di test, illustrati nella *Figura* 2.4: due per verificare la corretta esecuzione del metodo (se l'oggetto esiste o meno) e uno per verificare la risposta ad un ID non valido.

```
@Test
public void getNotificationFromIdIDNotValidTest(){
    IllegalArgumentException e = assertThrows(expectedType: IllegalArgumentException.class, () -> {
        apiMock.getNotificationFromId(notificationId: "1");
    });
    assertEquals(e.getMessage(), actual: "invalid ID");
}

@Test
public void getNotificationFromIdEmptyTest(){
    Optional<Notification> note = Optional.empty();

    when(dataBaseService.getNotificationFromId(new UUID(mostSigBits: 0, leastSigBits: 0))).thenReturn(note);

    IllegalArgumentException e = assertThrows(expectedType: IllegalArgumentException.class, () -> {
        apiMock.getNotificationFromId(new UUID(mostSigBits: 0, leastSigBits: 0).toString());
    });
    assertEquals(e.getMessage(), actual: "notification does not exist");
}

@Test
public void getNotificationFromIdMatchTest(){
    Notification note = createNotification();
    Optional<Notification> test = Optional.of(note);

    when(dataBaseService.getNotificationFromId(note.getId())).thenReturn(test);

    assertEquals(apiMock.getNotificationFromId(note.getId().toString()), note);
}
```

Figura 2.4: Esempio di test dei metodi *getXFromId*.

2.3.3 Test di ApiAppController.java

La classe ApiAppController.java contiene le API delle operazioni CRUD per le App. Tra i metodi della classe troviamo anche dei metodi per recuperare gli utenti associati all'app in base al ruolo (*getSubjects*, *getDPOs* e *getControllers*). In alcuni punti, inoltre, effettua un controllo sul ruolo dell'utente che ha chiamato il metodo per assicurarsi che non sia un data subject.

A tale scopo, i test, oltre a verificare la corretta risposta agli input, errati o meno, verificano anche che i metodi non proseguano in caso l'utente non sia un data controller o un DPO.

2.3.4 Test di ApiMessageController.java

La classe ApiMessageController.java contiene dei metodi che permettono di recuperare e aggiungere messaggi nel database. Oltre a poter recuperare un singolo messaggio, la classe permette di recuperare la totalità dei messaggi associati ad un utente (*getAllMessagesFromUser*) e una conversazione tra due utenti (*getConversation*). La classe, inoltre, effettua dei controlli per verificare che l'utente sia una delle due parti coinvolte (nel caso della lettura di un messaggio) o che esso sia il mittente e sia lui che il ricevente abbiano l'app su cui il messaggio verte (nel caso di aggiunta di un messaggio).
A tale scopo, i test, oltre a verificare la corretta risposta agli input, errati o meno, verificano anche che l'utente che ha chiamato la API soddisfi i requisiti elencati sopra.

2.3.5 Test di ApiNotificationController.java

La classe ApiNotificationController.java contiene i metodi per la gestione delle entità Notification, il cui scopo è quello di informare gli utenti dell'arrivo di un messaggio o del cambiamento di una richiesta di diritto o privacy notice. Difatti, le Notification portano al loro interno un attributo che identifica a quale dei tre oggetti sopra indicati si riferisce. Inoltre, Notification presenta un attributo che indica se la notifca è stata aperta o meno (*isRead*), giustificando così la presenza di API che recuperano le notifiche lette o meno (*getReadFromUser* e *getNotReadFromUser*) e che ne cambiano il valore (*changeIsRead*).
I test si sono occupati di verificare che i metodi rispondano correttamente ad input errati, siano in grado di filtrare per l'attributo *isRead* e che siano in grado di recuperare l'oggetto interessato dalla notifica.

2.3.6 Test di ApiPrivacyNoticeController.java

La classe ApiPrivacyNoticeController.java contiene i metodi per la gestione delle privacy notice. Oltre alla possibilità di aggiugere e rimuovere una singola privacy notice, la classe mette a disposizione dei metodi perl'aggiunta multipla e la copia di una privacy notice ad una o più App. E' anche in grado di recuperare una privacy notice di una specifica App, o tutte quelle associate ad uno specifico data subject. La maggior parte delle funzionalità della classe è accessibile solo a data controller e DPO.
A tale scopo, i test, oltre a verificare la corretta risposta agli input, errati o meno, verificano anche che l'utente che ha chiamato la API abbia un ruolo autorizzato.

2.3.7 Test di ApiRightRequestController.java

La classe ApiRightRequestController.java contiene i metodi per la gestione delle richieste di diritto. L'entità RightRequest presenta un attributo che indica se la richiesta sia stata gestita o meno, *handled*, giustificando la presenza di metodi che recuperano le richieste gestite o meno (*getHandledFromUser* e *getNotHandledFromUser*) e ne cambiano il valore (*changeIsHandled)*. La funzionalità di modifica delle richieste della classe è accessibile solo a data controller e DPO.
A tale scopo, i test, oltre a verificare la corretta risposta agli input, errati o meno, verificano anche che l'utente che ha chiamato la API abbia un ruolo autorizzato.

2.3.8 Test di ApiUserAppRelationController.java

La classe ApiUserAppRelationController.java contiene i metodi per la gestione delle entità UserAppRelation, che rappresentano il possesso di un utente di una data App e ne contengono gli eventuali consensi. La classe, quindi, oltre a permettere di aggiungere e rimuovere un entità UserAppRelation, permette anche di aggiungere e rimuovere i consensi di un utente ad una data App. Le funzionalità che gestiscono i

consensi di un utente possono essere usate esclusivamente dai data controller e dai DPO.
A tale scopo, i test, oltre a verificare la corretta risposta agli input, errati o meno, verificano anche che l'utente che ha chiamato la API abbia un ruolo autorizzato.

2.3.9 Test di ApiUserController.java

La classe ApiUserController.java contiene i metodi per le operazioni CRUD per gli utenti. Tra i metodi della classe troviamo anche i metodi per recuperare le App (*getApps*) e i contatti (*getAllContacts*) associati ad un dato utente. Alcune di queste funzionalità possono essere eseguite esclusivamente dall'utente direttamente interessato (*delete*, *update*).
A tale scopo, i test, oltre a verificare la corretta risposta agli input, errati o meno, verificano anche che l'utente che ha chiamato la API abbia un ruolo autorizzato.

2.4 Test del Package Security

Come già accennato nel capitolo 1.4.4, il package Security contiene le classi che realizzano le funzioni di autenticazione e autorizzazione degli utenti all'interno dell'app. Questo package è stato meno soggetto al processo di testing poiché è stato in continua lavorazione per la durata dell'elaborato.
La totalità dei test del package Security può essere visionata al seguente indirizzo: *https://github.com/sifis-home/ privacydashboard/tree/master/ src/test/java/com/ privacydashboard/application/security.*

2.4.1 Test di UserDetailsServiceImpl.java

La classe UserDetailsServiceImpl.java presenta dei metodi che permettono di manipolare i campi di un'entità Utente all'interno del database. Permette anche di

fornire l'autorizzazione ad un utente sulla base del suo ruolo.
La maggior parte delle funzioni non si presta a testing delle unità in quanto, data la necessità di usare oggetti mock, sarebbero risultate delle tautologie, privando di senso i test. Sono state testate, dunque, le funzionalità di assegnazione dell'autorizzazione e un metodo che verifica la presenza o meno dell'utente (*isAlreadyPresent*), testando la sua risposta ai possibili esiti del controllo.

2.5 Action per la Integrazione Continua

Ai fini dell'elaborato, è stata richiesta l'implementazione di due file: uno che realizzasse la CI e un'altro che si occupasse della CD. Il primo di questi (github.yml), che realizza la CI, presenta un workflow chiamato "test and docker", il quale è composto da tre jobs, illustrati nella *Figura* 2.5. La Action viene attivata ogni qualvolta viene effettuata una push sulla repository principale.

```yaml
name: test and docker

on:
  push:
    branches:
      - master

jobs:

  # Build and test the applications with multiple Java versions
  build-and-test: ···

  # Build files necessary for building Docker Images (Dockerfiles and Contexts)
  build-for-docker: ···

  # Build multiarch Docker Images and push as GitHub Packages
  docker-image-push: ···
```

Figura 2.5: Workflow test and docker

Il primo di questi job, "build-and-test", fa partire la suite di test sul codice e, in caso di esito positivo, prosegue a creare una build del codice. Parte di questo job è anche una serie di step che calcolano la percentuale di code coverage e la inviano

a GitHub sotto forma di badge[6].

Gli altri due job, "build-for-docker" e "docker-image-push", si occupano di creare un'immagine docker e di caricarla su GitHub per poterla recuperare in un secondo momento. Un'immagine di docker, per semplicità di fruizione, può essere vista come un indice del programma, un file che contiene tutte le librerie e le dipendenze (come e dove le librerie vengono usate) di esso, di piccole dimensioni e dal quale si può costruire la totalità del programma senza considerare la macchina sottostante.

Il secondo file (deploy.yml), che realizza la CD, presenta un workflow chiamato "deploy", il quale è composto da un solo job, come illustrato nella *Figura* 2.6. Questa Action viene attivata ogni volta che viene effettuata una push con un nuovo tag, quindi ad ogni nuova versione del programma.

```yaml
name: deploy

on:
  push:
    tags:
      - '*'

jobs:

  # Build executable Jar files and package as a Release
  deploy-release: ···
```

Figura 2.6: Workflow deploy

Il job "deploy-release" si occupa di creare un eseguibile del programma, recuperare il tag della versione corrente, creare un zip con tutti i file necessari per la corretta esecuzione del codice e di caricarlo su GitHub per poter essere poi distribuito.

[6]In italiano: medaglia. Piccola immagine contente la percentuale e colorata in base al valore di questa.

Capitolo 3

Risultati e Analisi

La quasi totalità dei test effettuati ha dato esiti positivi. Vengono riportati, di coneguenza, esclusivamente le classi all'interno delle quali sono stati riscontrati degli errori, spiegandone la causa e la soluzione al problema. Le immagini riportate sono state prese dalla schermata di GitHub che elenca i cambiamenti del codice in seguito ad un modifca. Il colore rosso indica righe o parole rimosse, mentre con il verde si indica l'aaggiunta di parole o righe.

3.1 Risultati dei test del Package Views

3.1.1 Risultati dei test di Questionnaire.java

Dei test relativi a questa classe fallivano *createQuestionnaireAppOrangeVoteTest* e *createQuestionnaireAppRedVoteTest*, in quanto la classe che segnalava la votazione dell'app *orangeName*/*redName* veniva assegnata al'elemento sbagliato (card invece che name). La correzione è riportata nella *Figura* 3.1.

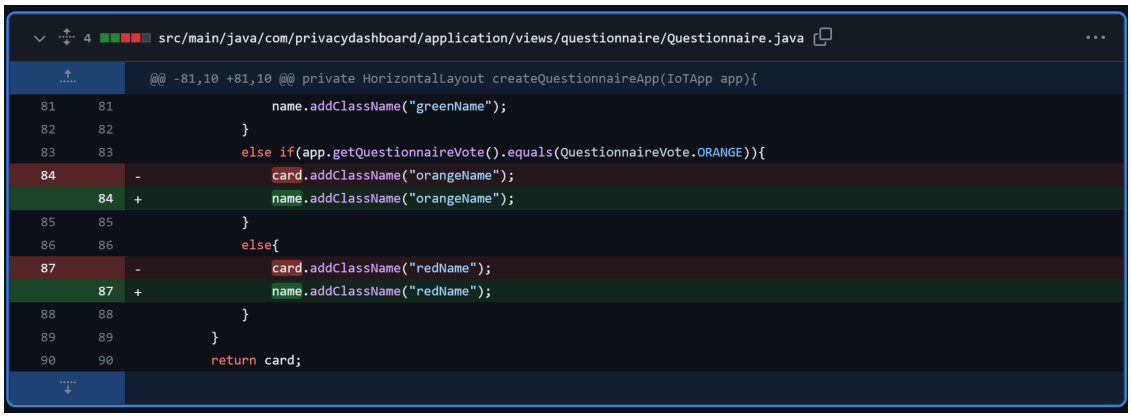


Figura 3.1: Correzione errori nella classe Questionnaire.java

3.2 Risultati dei test del Package Data

3.2.1 Risultati test delle Entity

La classe UserAppRelation.java, all'assegnazione dell'utente, salva in un attributo apposito il ruolo dell'utente, senza però poterlo mai leggere, a causa dell'assenza di un metodo get, o usarlo in altri metodi.
Questo è un errore:

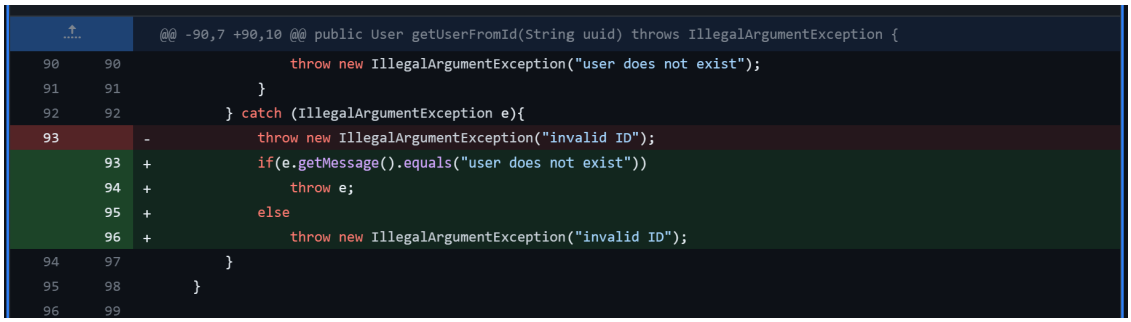
- Di implementazione, perché avere un attributo che non si può leggere e he non viene usato dalla classe è soltanto uno spreco di memoria.

- Di logica, perché l'attributo è lo stesso dell'attributo User e, dato che il ruolo non può mai essere letto se non dall'attributo User, la sua presenza come un attributo esterno non è giustificata da un possbile uso nel codice.

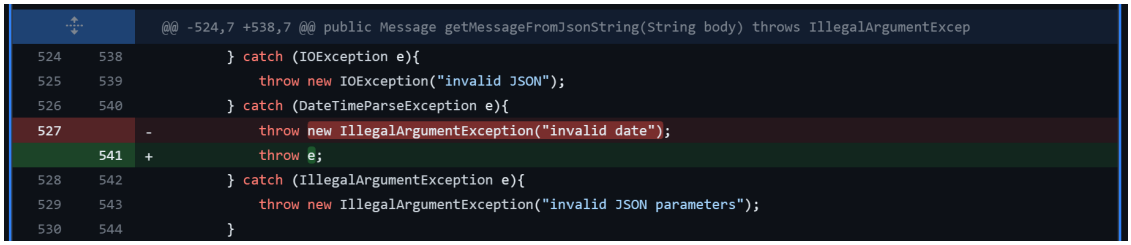
L'attributo andrebbe rimosso, però in mancanza di documentazione non è chiaro l'intento originale per cui il ruolo viene salvato in un campo apposito.

3.2.2 Risultati dei test di ApiGeneralController.java

Un errore comune ad una serie di metodi all'interno di questa classe riguarda il lancio delle eccezioni. In particolare, alcune eccezioni venivano lanciate con un determinato messaggio (corretto) in risposta ad un errore, trovandosi però all'interno di una clausola try-catch che controlla un altro tipo di errore: di conseguenza, l'eccezione originale viene mascherata dal try-catch, rendendo difficile al codice chiamante rispondere in modo coerente ad essa. Due esempi di questi errori vengono illustrati nelle *Figure* 3.2. I metodi coinvolti sono quelli che recuperano le entità a partire dal loro Id (*getObjectFromId*) e *getMessageFromJsonString*



(a) Metodo *getUserFromId*



(b) Metodo *getMessageFromJsonString*

Figura 3.2: Correzione errori nel lancio delle eccezioni

Il metodo *getAppFromJsonNode* effettuava un'operazione illecita per Java, assegnando il valore nullo (null) ad una HashTable, come illustrato nella *Figura* 3.3. Il metodo *createJsonFromUserAppRelation* falliva i test in quanto scambiava i valori da assegnare al nome dell'app e del'utente, come illustrato nella *Figura* 3.4.

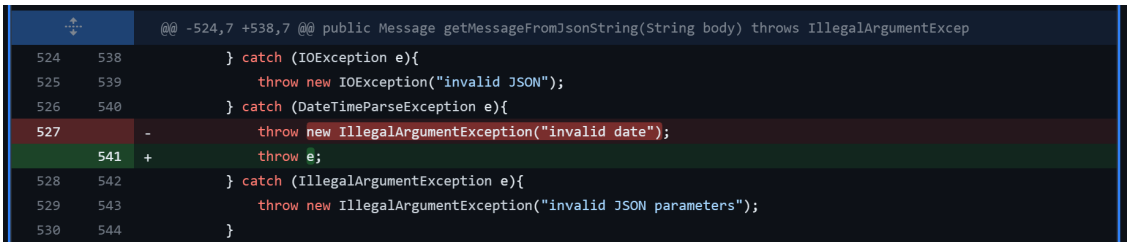


Figura 3.3: Correzione errore nel metodo *getAppFromJsonNode*

Figura 3.4: Correzione errore nel metodo *createJsonFromUserAppRelation*

Infine, il metodo *getMessageFromJsonNode* falliva i test in quanto sbagliava l'assegnazione dell'attributo Response, assegnandolo invece a Details, come illustrato nella *Figura* 3.5.

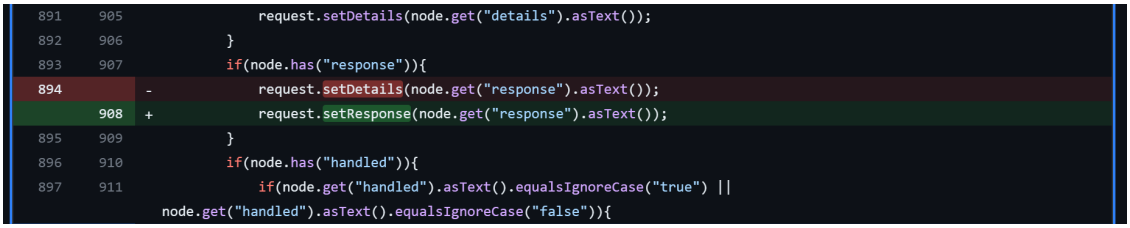


Figura 3.5: Correzione errore nel metodo *getMessageFromJsonNode*

3.2.3 Risultati dei test di ApiUserAppRelationController.java

All'interno della classe ApiUserAppRelationController.java, i metodi *addConsenses*, *removeConsenses* e *removeAllConsenses*, fallivano i test che verificavano la

corretta risposta ad input errati in quanto non restituivano l'oggetto previsto, data la mancanza della keyword *return* come illustrato nella *Figura* 3.6.

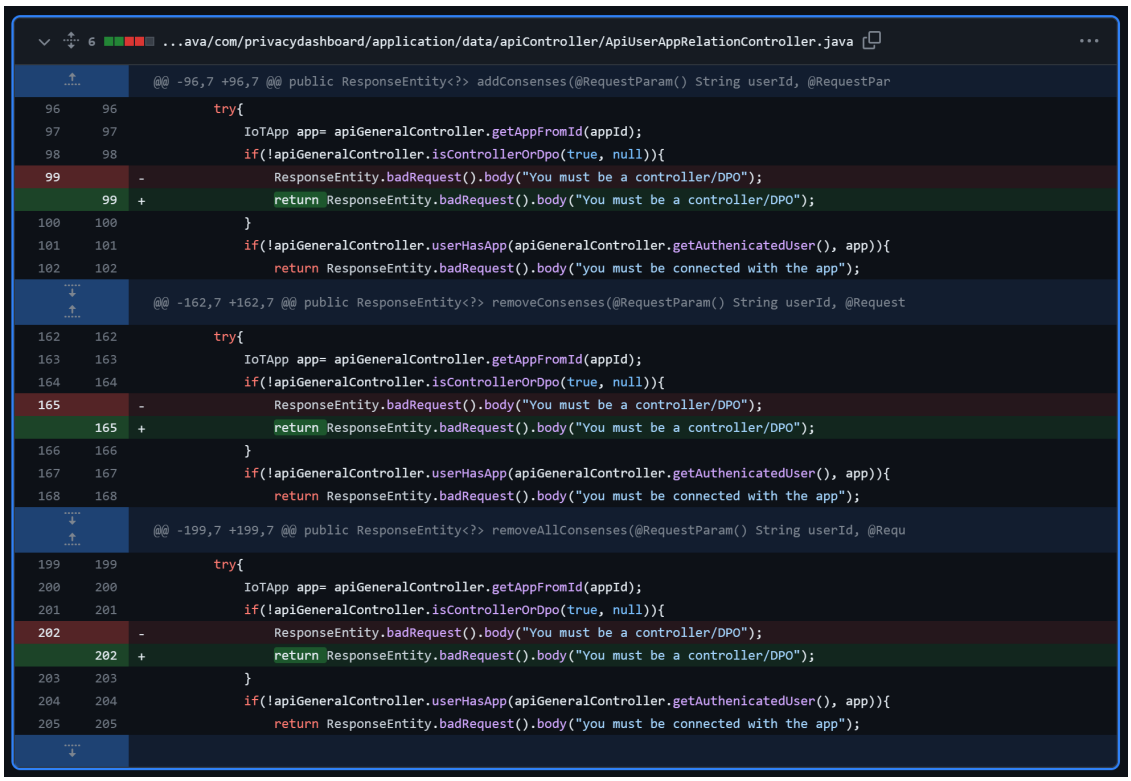


Figura 3.6: Correzione errori nella classe ApiUserAppRelation.java

Capitolo 4

Conclusioni

Come si è potuto evincere dai risultati dei test, all'interno del codice non erano presenti grossi errori nell'implementazione del codice. Sono emersi alcuni errori di logica, come le eccezioni lanciate dentro un try-catch (3.2.2) o un attributo inizializzato ma mai utilizzato (3.2.1).
Un'altra problematica emersa durante la fase di testing riguarda la manutenibilità del codice in merito ad alcune classi che sfruttano il tag di Spring *@Autowired*: se da una parte esso semplifica la scrittura del codice, è risultato una complicazione durante la creazione dei test. Infatti, esso, prevenendo la necessità di assegnare gli attributi *dataBaseService*, *authenticatedUser* e *userDetailsServiceImpl* durante l'esecuzione normale dell'applicativo, li rende nulli durante l'esecuzione controllata dei test. Ciò ha comportato la necessità di aggirare l'incapsulamento dei dati per poter assegnare loro un oggetto mock per l'esecuzione dei test. Oltre ad essere in contrasto con i paradigmi di programmazione di Java, risulta problematico anche per la futura realizzazione di test d'integrazione e test di sistema.
Un'altra situazione che potrebbe essere migliorata è quella della ripetizione del codice tra le classi AppsView.java, AvailableAppsView.java e le loro rispettive classi di test. Sarebbe corente, infatti, creare una classe da cui le altre due ricavino i metodi comuni, per una serie di vantaggi:

- Ridurre la ripetizione del codice permette di apportare modifiche estetiche alle due pagine senza dover riportare il codice da una classe all'altra.
- La suite di test diventa più snella e verificare le nuove funzionalità non richiede la duplicazione dei test.

Ancora, migliorerebbe la manutenibilità e leggibilità del codice estrapolare dalle classi FormPrivacyNotice.java e SingleQuestionnaire.java le varie stringhe di testo di domande e risposte. Salvandole in file di testo sarebbe più semplice modificare loro, qualora vi fosse necessità, e il codice delle classi. Con piccole modifiche ai test, inoltre, sarebbe possibile anche non dover cambiare i test al semplice cambiamento del testo delle domande. Ciò renderebbe i test e il codice anche più resistenti all'aggiunta di altre lingue, riducendo di molto lo sforzo richiesto per implementarle. Per migliorare la manutenibilità del codice si potrebbe anche realizzare una classe di servizio contenente tutte le funzioni di creazione di input sfruttate nelle varie classi di test.
Infine, migliorare la documentazione del codice, in particolare esiti e usi desiderati di metodi e attributi, renderebbe la realizzazione dei test più efficace e rapida. Aiuterebbe, anche, lo sviluppo del codice, offrendo una fonte immediatamente leggibile delle funzionalità delle varie classi, per evitare ripetizione del codice.

Bibliografia

[1] I. T. Union, "About internation telecommunication union (itu)."

[2] I. T. U. T. S. Sector, "Itu-t in brief."

[3] "Recommendation itu-t y.2060," tech. rep., International Telecommunication Union - Telecommunication Standardization Sector, 2012.

[4] I. A. Board, "Internet architecture board."

[5] "Rfc 7452," tech. rep., Internet Architecture Board, 2015.

[6] I. of Electrical and E. Engineers, "Ieee at a glance," 2022.

[7] I. of Electrical and E. Engineers, "Internet of things security analysis," 2011.

[8] "Number of internet of things (iot) connected devices worldwide from 2019 to 2023, with forecasts from 2022 to 2030," *Statista*, 2023.

[9] S. Domotica, "Domotica: cos'è e come funziona," 2023.

[10] A. Authority, "Smart home privacy: What data is collected, and how it is used?," 2022.

[11] G. C. Marco Ciurcina, "Final report on legal and ethical aspects," tech. rep., 2023.

[12] F. S. Foundation, "Cos'è il software libero?."

[13] "Gdpr articolo 1: Obiettivo e finalità," tech. rep., Commissione Europea.

[14] "Gdpr articolo 4: Definizioni," tech. rep., Commissione Europea.

[15] "Gdpr articolo 7: Condizioni per il consenso," tech. rep., Commissione Europea.

[16] "Gdpr articolo 5: Principi applicabili al trattamento di dati personali," tech. rep., Commissione Europea.

[17] "Gdpr articolo 13: Informazioni da fornire qualora i dati personali siano raccolti presso l'interessato," tech. rep., Commissione Europea.

[18] "Gdpr articolo 14: Informazioni da fornire qualora i dati personali non siano stati ottenuti presso l'interessato," tech. rep., Commissione Europea.

[19] "Gdpr articolo 37: Designazione del responsabile della protezione dei dati," tech. rep., Commissione Europea.

[20] "Gdpr articolo 38: Posizione del responsabile della protezione dei dati," tech. rep., Commissione Europea.

[21] "Gdpr articolo 39: Compiti del responsabile della protezione dei dati," tech. rep., Commissione Europea.

[22] "Gdpr articolo 12: Informazioni, comunicazioni e modalità trasparenti per l'esercizio dei diritti dell'interessato," tech. rep., Commissione Europea.

[23] C. Europea, "Writing a gdpr-compliant privacy notice (template included),"

[24] Vaadin, "Vaadin flow."

[25] V. Tanzu, "Spring security."

[26] I. Sommerville, *Ingegneria del software, Decima edizione*. Pearson Italia, 2017.

[27] I. of Electrical and E. Engineers, *Guide to the Software Engineering Body of Knowledge v3.0*. IEEE, 2014.

[28] E. W. Dijkstra, "The humble programmer," 1972.

[29] JUnit, "Junit 5 user guide," 2023.

[30] S. Faber, "Mockito."

[31] Oracle, "Package java.lang.reflect."

[32] Atlassian, "Cos'è la code coverage?."

[33] EclEmma, "Jacoco java code coverage."

[34] IBM, "Cos'è l'integrazione continua?."

[35] M. Fowler, "Continuous integration," 2006.

[36] GitHub, "Understanding github actions."