Paradigmi di Programmazione e Sviluppo
C.I. Sviluppo di Sistemi Software

Average Joe's
Gym Simulation

Andrea Campidelli (andrea.campidelli@studio.unibo.it)
Sokol Guri (sokol.guri@studio.unibo.it)
Elena Morelli (elena.morelli3@studio.unibo.it)

Anno accademico 2019-2020
Cesena

Indice

Capitolo 1

Introduzione

AverageJoe's è una piattaforma che permette di simulare l'interazione tra macchinari ginnici smart, ossia dotati di una sensoristica avanzata. La piattaforma consente di osservare l'utilizzo dei macchinari all'interno di una palestra, prenotabili rispetto agli esercizi presenti sulla scheda del cliente.

Il cliente potrà svolgere la sua sessione d'allenamento prenotando tramite smart devices i macchinari della palestra. Il macchinario verrà configurato in modo automatico, appena il smart device di un cliente caricherà la scheda d'allenamento. Il cliente potrà prenotare il macchinario del prossimo esercizio da eseguire tramite questo smart device, rendendo la sessione di workout il più confortevole possibile.

Il cliente sarà in grado di configurare il macchinari manualmente, nel caso in cui l'esercizio non sia presente all'interno della sua scheda d'allenamento.

Capitolo 2

Processo di Sviluppo

Per poter sviluppare AverageJoe's in modo efficiente e dinamico, il team ha deciso di utilizzare Scrum come metodologia per lo sviluppo.

2.1 Organizzazione degli sprint

Come delineato da Scrum è stato seguito lo schema degli sprint: per ognuno di essi, come prima operazione è stata effettuata uno come prima operazione è stata effettuata una sprint retrospective per individuare cosa migliorare nel nostro processo di sviluppo, seguita da uno sprint planning per pianificare cosa fare nello sprint da iniziare. Nella fase di retrospective sono stati valutate eventuali problematiche riguardanti il lavoro svolto, individuando i miglioramenti necessari e pianificando il modo in cui implementarli nello sprint successivo.

Nel planning invece oltre alla definizione dei task e alla suddivisione del lavoro, mantenendo sempre una ripartizione rispetto alle entità in gioco nel sistema, è stata scelta di volta in volta una deadline che fosse il più adatta possibile in base alla mole di lavoro da svolgere.

Durante la fase di sviluppo dello sprint inoltre un membro poteva assegnarsi uno o più nuovi task nell'eventualità che finisse tutto il lavoro a lui assegnatogli prima che la deadline fosse raggiunta. Alla fine di ciascuno sprint, nella fase di review, si è effettuata una valutazione complessiva del lavoro prodotto dove ciascun membro spiegava agli altri cosa avesse fatto.

2.2 Organizzazione del team

Per sfruttare al massimo le potenzialità di Scrum, si è deciso di utilizzare gli strumenti messi a disposizione dalla piattaforma Trello. Ogni **Product**

Backlog viene identificato attraverso una serie di schede che rappresentano i **task** che i membri del gruppo dovranno prendersi in carico. Le **schede** a loro volta sono suddivise in **liste** in base al loro stato di sviluppo.

Le liste individuate sono le seguenti: *To Do*, dove sono presenti i task da prendere in carico; *In Progress* composta dalle liste dei diversi Sprint con le relativa deadline; e infine *Done* dove sono presenti le schede completate.

2.3 Strumenti

2.3.1 Versioning

Si è utilizzato Git per effettuare il versioning del codice durante lo sviluppo, insieme alla piattaforma GitHub . E'stato creato un repository *principale* e da questo ogni membro del gruppo ha creato una fork personale. Il non dover lavorare su un unico repository ha permesso di non dover seguire rigidamente il *gitflow*, mantenendo nel principale l'uso di soli due branch quali il *master* e il *develop*. Sul proprio repository ciascun membro è stato libero di lavorare come meglio preferisse. Alla conclusione di ogni sprint per riunire il materiale prodotto ciascun membro ha fatto una *pull request*. Nella sprint review il materiale veniva ispezionato dagli altri membri e a seconda dei casi accettato o rifiutato. Avendo permessi di scrittura sul repository principale, questa operazione poteva essere effettuata da tutti.

2.3.2 Build

Per effettuare le build del codice prodotto si è fatto uso di SBT. La scelta iniziale era ricaduta su Gradle ma si è deciso di cambiare in corso d'opera. La preferenza è ricaduta su questo strumento per via della sua adeguatezza nel testing. In questo modo si può fare il run dei soli test falliti e rende anche possibile il lavorare con versioni di scala differenti.

2.3.3 Test

Una parte cruciale dello sviluppo rientra nel testing. Si è cercato di attenersi ad una metodologia di sviluppo orientata al TDD ma che non seguisse pedissequamente tutti i suoi canoni. Di volta in volta poi si è controllato se il lavoro svolto soddisfacesse qualche test fino a soddisfarli tutti. Una caratteristica molto importante riguarda le funzionalità che i test controllavano: sin dall'inizio si sono creati dei logging sulla base delle interazioni principali

fra gli elementi del sistema. Per quanto riguarda l'implementazione dei test, nel progetto si è fatto uso del framework di test di Akka.

2.3.4 Continuos Integration

Per automatizzare il building e il testing del sistema inizialmente si era optato per utilizzare Travis CI, ma data la recente uscita delle GitHub Actions si è deciso di virare in questa direzione per la semplicità con cui si possono scrivere i file di configurazione e perché si è ritenuto molto positivo poter effettuare tutte le operazioni su un'unica piattaforma.

Capitolo 3

Requisiti

3.1 Requisiti di business

Sono stati definiti i requisiti di alto livello che rappresentano le principali funzionalità che l'applicazione vuole offrire.

- Utilizzare il braccialetto come strumento per "accedere" a tutti i servizi della palestra.
- Utilizzo dati per "logiche aziendali" di manutenzione delle macchine e statistiche di utilizzo.
- Utilizzo delle macchine su prenotazione automatica.

3.2 Requisiti utente

Requisiti che si focalizzano su ciò che l'utente si aspetta dall'applicazione.

- Interfacciarsi con i macchinari
- Tracciamento dell'allenamento

3.3 Requisiti Funzionali

Qui sono state definite le funzionalità che l'applicazione vuole fornire in modo più specifico. Si è cercato di dare anche una visione di ciò che l'applicazione è in grado di fare. Il sistema dovrà:

- Ingresso in palestra: Deve essere possibile accedere alla palestra passando il braccialetto su un apposito scanner. Il sistema dovrà fare un controllo sulla validità dell'utente e del suo abbonamento e, se necessario mostrare all'utente il relativo avviso di negazione dell'accesso. Data la necessità di un controllo contingentato sugli accessi, il visitatore deve poter essere fermato in questo punto, prima di accedere ai locali della palestra.

- Accesso alle macchine: Dovrà essere possibile loggarsi alle macchine e impostare i dati di configurazione tramite il solo utilizzo del braccialetto, senza bisogno di altre operazioni da parte dell'utente.

- Il sistema dovrà avvertire lo staff, attraverso opportuni messaggi, della necessità di sostituzione pezzi / manutenzione delle macchine, in relazione allo stato di utilizzo della macchina.

- Il sistema deve gestire la prenotazione delle macchine in relazione alla scheda di allenamento e ai tempi medi di utilizzo delle macchine, prenotando il posto in tempo utile per la fine dell'esercizio corrente e mantenendo la prenotazione non oltre un determinato lasso di tempo per non rallentare eccessivamente gli altri utenti. Un utente si può loggare ad una macchina solo se è libera o se è scaduto il termine della prenotazione.

- Visualizzazione dei valori: Il display della macchina dovrà mostrare i valori configurati per l'allenamento e i parametri vitali dell'utente

- Modifiche al piano di allenamento: l'utente deve poter modificare, attraverso il display della macchina, i parametri dell'allenamento definiti dallo staff

- Il sistema deve registrare, per future consultazioni da parte dell'utente, i seguenti dati: configurazioni macchine, ripetizioni, tempi, parametri vitali, ecc...

3.4 Requisiti non funzionali

I seguenti requisiti sono volti ad assicurare un livello di qualità del sistema e del suo comportamento. Inoltre sono stati definiti alcuni vincoli che il sistema dovrà rispettare.

3.4.1 Scalabilità

- Il sistema dovrà gestire fino ad un numero prefissato di utenti contemporaneamente all'interno della palestra. Il sistema non sarà scalabile per gestire più utenti di quelli prefissati.
- Nel caso sia stato raggiunto il limite di utenti consentiti, tramite l'accesso alla palestra, il sistema non permetterà più l'accesso ad altri utenti fino a che non si sia liberato un posto e quindi un altro utente abbia effettuato l'uscita.

3.4.2 Resistenza ai guasti

- Nel caso in cui un utente perda la connessione al macchinario, tramite il braccialetto, il sistema dovrà continuare a funzionare mantenendo l'integrità per gli altri utenti e le componenti del sistema.

3.4.3 Usabilità

- Il sistema dovrà fornire all'utente un'interfaccia utente chiara e che permetta all'utente di poter usufruire di tutte le funzionalità messe a disposizione.

3.4.4 Riusabilità

- Il codice dovrà essere per quanto possibile riutilizzabile per evitare ripetizioni.
- Il sistema sarà organizzato in modo modulare in modo che i suoi componenti possano essere riutilizzabili.

3.4.5 Requisiti di implementazione

Per quanto riguarda l'implementazione sono stati definiti alcuni obbiettivi e vincoli da rispettare al fine di produrre del software di qualità.

- Utilizzare un approccio per il più possibile funzionale.
- Mantenere uno stato immutabile nei vari componenti del sistema così da poter facilitare ulteriormente la realizzazione del punto precedente.
- Strutturare le prenotazioni dei macchinari al fine di evitare possibili deadlock.

Capitolo 4

Design architetturale

Il sistema è stato pensato facendo riferimento al pattern architetturale Model-View-Controller. L'utente può utilizzare la piattaforma tramite l'interfaccia grafica messa a disposizione dalla View, mentre la logica del programma è contenuta nel Model. Il Controller si pone come intermezzo per la comunicazione tra le diverse componenti del Model e come entità di gestione della View.

4.1 Model

All'interno del model vi sono tutte le entità fondamentali del sistema con le eventuali logiche di comunicazione. I componenti individuati sono: le Physical Machine, i Customer, i Wristband.

I **Physical Machine** rappresentano i macchinari fisici, cioè le attrezzature da palestra che vengono messe a disposizione ai customer.

I **Customer** sono gli iscritti alla palestra. Ad ognuno di essi è stato assegnato Training Program dove sono presenti gli esercizi da svolgere su ogni macchinario con i rispettivi parametri di utilizzo. Hanno anche il compito di prenotare un macchinario presente all'interno del prioprio Training Program e di inviare i relativi parametri alla macchina.

I **Wristband** sono i braccialetti che i customer hanno a disposizione per interagire con i macchinari. Si occupano del login sulle macchine e di inviare i parametri (battito cardiaco) dell'utilizzatore.

4.2 View

La view permette la visualizzazione dei parametri degli esercizi che i customer stanno svolgendo oltre a quelli vitali dei customer stessi. L'utente tramite la view simula il collegamento RFID tra braccialetto e macchinario scegliendo con quale collegarsi attraverso appositi bottoni.

4.3 Controller

Entrambi i controller si occupano della creazione dei vari attori che compongono il modello. Per quanto riguarda la parte Hardware, gli attori alla creazione si integrano con la view, e attraveso quest'ultima interagiranno fra di loro durante lo svolgimento degli esercizi. Il GymController invece, oltre alla creazione, si occupa anche di alcune interazioni tra gli attori del modello che non sono direttamente legati: restituisce i customer alle macchine e viceversa quando hanno bisogno di relazionarsi fra di loro.

4.4 Interazione principale

Qui di seguito si riportano le fasi cruciali del sistema, spiegando come le entità principali interagiscono fra di loro in queste circostanze.

4.4.1 Login nella palestra

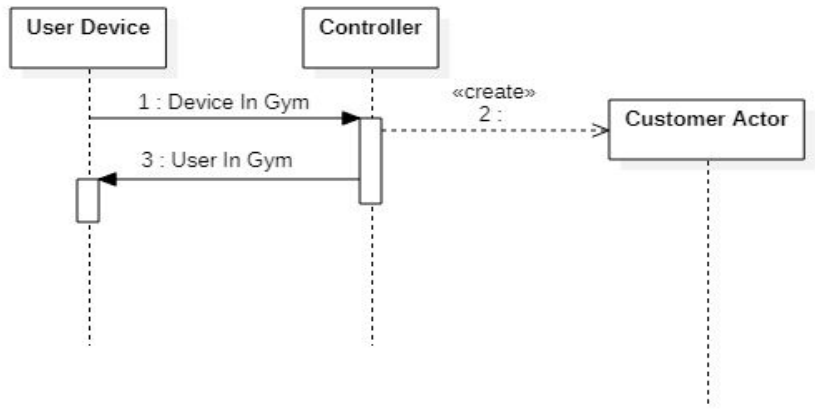

Figura 4.1: Log-in palestra

Il login alla palestra avviene come indicato in Figura 6.1. Ogni utente iscritto alla palestra riceve in dotazione un braccialetto smart (UserDevice), che tramite riconoscimento RFID, effettuerà l'ingresso all'interno della palestra comunicandolo al controller. Una volta che il controller avrà ricevuto questo messaggio creerà un nuovo attore per l'utente entrato.

4.4.2 Creazione dei Macchinari

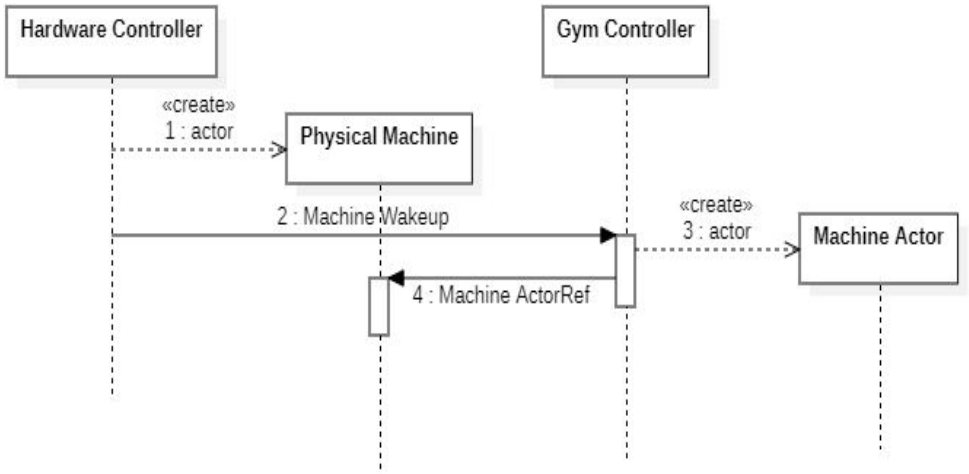

Figura 4.2: Creazione PhysicalMachine

Per la gestione dei macchinari (PhysicalMachine) e dei braccialetti è stato creato un HardwareController, che ha il compito di inizializzare queste tipologie di componenti. Una volta creata una PhysicalMachine l'HardwareController avviserà il GymController, il quale darà origine ad un MachineActor.

Portato a termine questo compito il GymController manderà al PhysicalMachine il riferimento al MachineActor appena creato, così da attribuire al macchinario la sua entità logica.

Capitolo 5

Design di dettaglio

A valle di un'analisi del sistema si è giunti alla conclusione che ogni suo componente fosse completamente reattivo con un comportamento orientato all'interazione tramite messaggi. Inoltre, l'obiettivo è quello di costruire un sistema scalabile a fronte dell'aumentare del numero di utenti e dei macchinari. Tutti questi fattori hanno fatto capire come fosse necessaria un'astrazione maggiore del paradigma ad oggetti classico. Per questo motivo si è utilizzato il paradigma ad attori, perfetto per realizzare sistemi distribuiti al cui interno sono presenti entità reattive. Grazie a questa scelta in un prossimo futuro il passaggio da un server concentrato, come quello realizzato per il progetto, ad uno distribuito, non richiederà grandi modifiche e/o grandi sforzi. La realizzazione dei componenti permette di ottenere una grande indipendenza reciproca tra essi, caratteristica che permette loro di risiedere su macchine diverse senza alcun tipo di problema. Nel progetto si è scelto di implementare gli attori con la tecnologia di Akka in quanto è uno standard affermato per il linguaggio di programmazione da noi usato, supportato e con feedback molteplici e frequenti da parte della community.

5.1 Model

Il model è suddiviso in diverse entità oltre ai tre attori principali: PhysicalMachine, Device, Customer.

5.1.1 Machine

E' l'entità che ha la conoscenza della logica di funzionamento di un macchinario fisico. Il suo compito è quello di configurare l'attrezzatura rispetto alla tipologia di macchinario assegnata. Ogni PhysicalMachine si avvale di

un *MachineActor* per la logica applicativa del macchinario. Questa entità è ciò che trasforma la macchina fisica in un'entità smart, rendendo possibile l'interazione con le componenti del sistema. Il MachineActor è colui che si occupa di gestire lo stato del macchinario rispetto ad un possibile booking da parte del braccialetto oltre al salvataggio dei parametri degli esercizi svolti. Il *FileWriterActor* è l'attore figlio del MachineActor che si occupa della mansione spiegata precedentemente.

5.1.2 Device

E' l'entità che ha la conoscenza della logica di funzionamento del braccialetto fisico. Il device ha il compito di permettere all'utente l'ingresso all'interno della palestra oltre a mostrargli lo stato dei sui parametri vitali.

5.1.3 Customer

L'entità *Customer* rappresenta un cliente della palestra a cui è stato assegnato un *TrainingProgram* (*scheda d'allenamento*) con una possibile lista di *"smart" Exercise* da poter eseguire all'interno della palestra. Le schede d'allenamento sono memorizzate nel data storage della palestra, dal cui vengono estratte.

Gestione dei customer

Durante lo scambio dei messaggi, l'attore del customer verrebbe sovraccaricato e la gestione della comunicazione sarebbe impossibile. Per avere più fluidità nella comunicazione si è utilizzato una organizzazione ad albero con tre livelli. In questo modo, si facilita lo smistamento delle richieste e la gestione dei comportamenti degli attori. Ogni attore, parte di un certo livello dell'albero, avrà a carico solo le responsabilità correlate a lui.

- **CustomerManager**: Tale attore svolge il ruolo di un entry point per l'istanziamento di CustomerActor e la sua autenticazione in un macchinario. Questo attore effettua un filtraggio delle richieste e le inoltra al attore che si sta chiedendo la comunicazione.
- **CustomerGroup**: Tale attore viene istanziato da CustomerManager. CustomerGroup viene utilizzato per eseguire la logica dietro le richieste che CustomerManager riceve. Principalmente dovrebbe:
  - eseguire query sul database e verificare la correttezza delle richieste ricevute,
  - istanziare un CustomerActor e caricare il suo TrainingProgram.

per eseguire query sul database e verificare la correttezza delle richieste.

- **CustomerActor**: Tale attore rappresenta un cliente dentro la palestra e verrà istanziato dal CustomerGroup, dopo le verifiche effettuate. Il suo ruolo principale è lo scambio dei messaggi con i *smart device* e le *smart machine*, per rendere la sessione d'allenamento sempre più smart. In particolare dovrebbe:
  - ricevere e verificare una richiesta di autenticazione da parte di un macchinario.
  - prenotare un macchinario per il prossimo esercizio della scheda d'allenamento creando un nuovo attore.
  - tenere aggiornati i smart device e i machinari riguardo l'andamento dell'esercizio e della prenotazione.

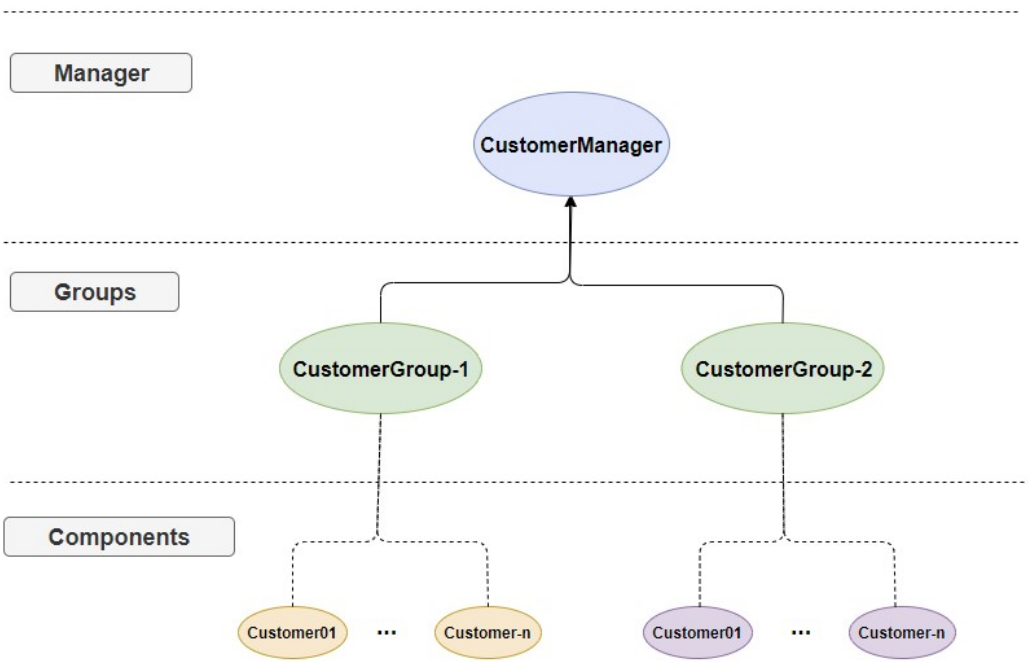

Figura 5.1: Organizzazione dei customer

5.1.4 Training

L'allenamento di un cliente si basa principalmente sulla sua scheda d'allenamento **TrainingProgram** è un insieme di esercizi assegnato a lui. Gli

esercizi sono ordinati in base alla priorità definita per ogni esercizio. Sulla entità TrainingProgram si effettuano operazioni di aggiunzione e rimozione degli esercizi, per ritornare una scheda d'allenamento aggiornata.

L'entità **Exercise** viene creata definendo un ordine dell'esercizio e una lista di parametri utilizzabili per la configurazione dei macchinari. Gli esercizi sono salvati in database nel formato di un *workout*, che però verranno implicitamente convertiti.

5.2 Controller

Il controller si compone di due entità per mantenere separata, coerentemente con le altre parti del sistema, la parte relativa all'hardware. Così da permettere un passaggio ai dispositivi fisici senza alterare la parte centrale del sistema. L'HardwareController si occupa quindi esclusivamente della creazione degli attori che, come la loro controparte fisica, interagiranno direttamente tra loro attraverso RFID e Bluetooth (in questo caso simulati da messaggi). Il GymController invece, oltre a creare gli attori del modello logico, gestisce le loro interazioni preliminari:

- quando un braccialetto si avvicina ad una macchina, il controller attraverso vari passaggi gestisce l'associazione tra il MachineActor e il CustomerActor;
- quando un Customer cerca una tipologia di macchina da prenotare, il GymController restituisce la lista delle macchine di quel tipo presenti in palestra;
- si occupa inoltre di fornire ai modelli hardware la loro controparte logica dopo averla creata.

5.3 Database

L'entità Database consiste in due *Storage*. Uno contenente i dati dei customer registrati all'interno del sistema e l'altro la lista degli esercizi da completare con i rispettivi parametri, sempre identificati dal customer di appartenenza. I dati si trovano in due file JSON distinti e vengono caricati attraverso l'utilizzo di una serie di metodi adibiti al parsing.

5.4 View

E' composta dall'oggetto *View* che ha il compito di creare lo scheletro della GUI. Lo scheletro è composto da due pannelli adibiti rispettivamente a mostrare i parametri dei macchinari e quelli dei device. Ogni device creato all'interno del panello mostra, tramite una lista, i possibili macchinari con cui simulare la connessione tramite RFID. L'aggiornamento dei display di ogni elemento della view è ha carico del *ViewToolActor*. La GUI permette anche il settaggio dei parametri tramite la presenza di text field. Nel caso in cui vengano inseriti valori non consoni per il sistema questo mostrerà una dialog di errore.

5.5 Interazione tra attori

5.5.1 Login al macchinario

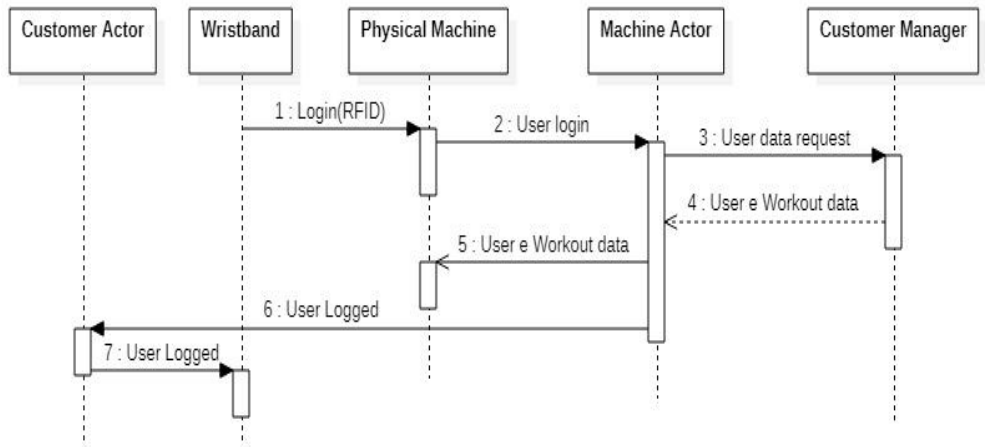

Figura 5.2: Operazione di login al macchinario

La Figura 6.1 mostra l'interazione fra gli attori presenti nel sistema nel momento in cui il Wristband si avvicina al macchinario. La PhysicalMachine viene notificata tramite un messaggio, che simula l'RFID, della richiesta di connessione, mettendo al corrente anche il suo MachineActor. Il MachineActor richiede al CustomerManager i parametri di settaggio dei macchinari per poi passarli alla PhisicalMachine. Il Wristband sarà successivamente aggiornato così da mostrare i parametri vitali dell'utente. Durante l'esecuzione dell'esercizio il CustumerActor si occuperà di prenotare il macchinario successivo presente all'interno dell TrainingProgram dell'utente. Una volta

completato l'esercizio il MachineActor provvederà alla scrittura dei parametri dell'esercizio eseguito.

5.5.2 Prenotazione di un macchinario

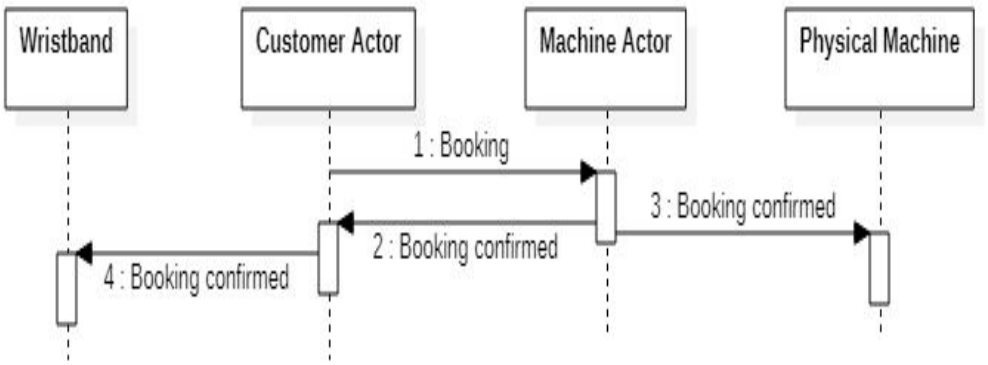

Figura 5.3: Prenotazione di un macchinario

Come già menzionato in precedenza il CustomerActor, gestito dal CustomerBooker, ha il compito di prenotare un macchinario nel caso in cui ci siano degli esercizi presenti all'interno del TrainingProgram. Nel caso non sia vuoto il CustomerActor provvederà ad estrarre un esercizio e a prenotare il macchinario necessario per svolgerlo. La prenotazione viene mandata al MachinActor il quale entrerà in uno stato di prenotazione in un lasso di tempo scandito da un timer. Nel caso in cui il timer scada prima che l'utente abbia effettuato la connessione con il macchinario prenotato, allora la prenotazione verrà cancellata e l'utente potrà loggarsi solo nel caso in cui il macchinario risulti ancora libero. La figura 6.1 mostra la richiesta di prenotazione su un macchinario libero.

5.6 Organizzazione del codice

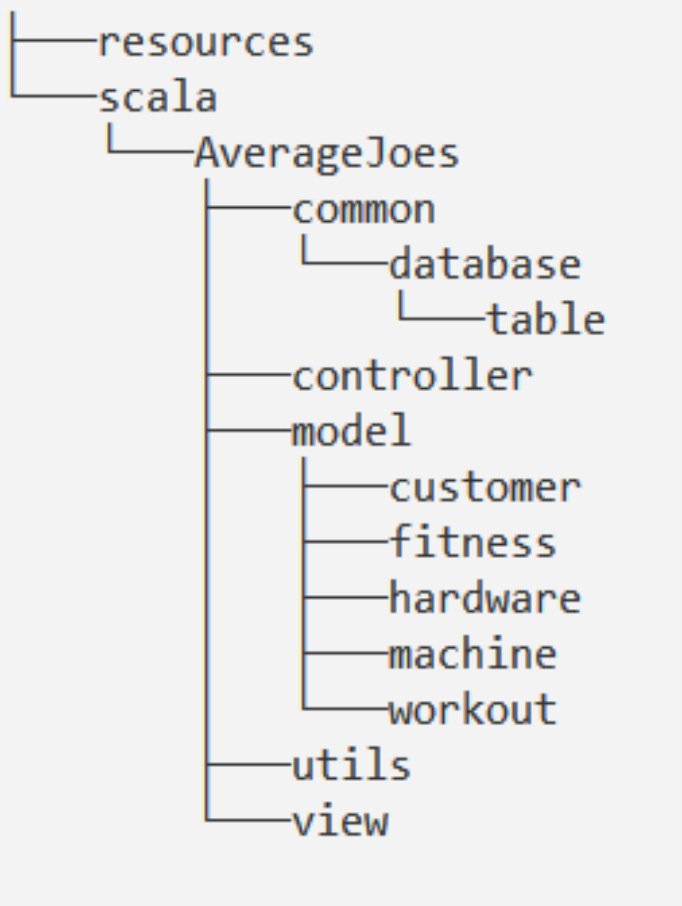

Figura 5.4: Struttura dei package

- resources: contiene i file relativi al salvataggio dei dati.
- common: contiene le classi relative alla struttura di un database interno
- controller: contiene il controller del sistema
- model: contiene le classi relative alle entità del sistema
  - customer: classi relative all'utente compresa la gestione della prenotazione di un macchinario
  - fitness: classi relative al settaggio dei parametri per gli esercizi relative al customer

- hardware: classi relative ai macchinari fisici
- machine: classi relative alla logica dei macchinari
- workout: classi relative al settaggio dei parametri relative alla tipologia di macchinari
- utils: classi per di utilità
- view: classi relative all'intergìfaccia grafica e al suo aggiornamento

Capitolo 6

Implementazione

6.1 Interfaccia utente

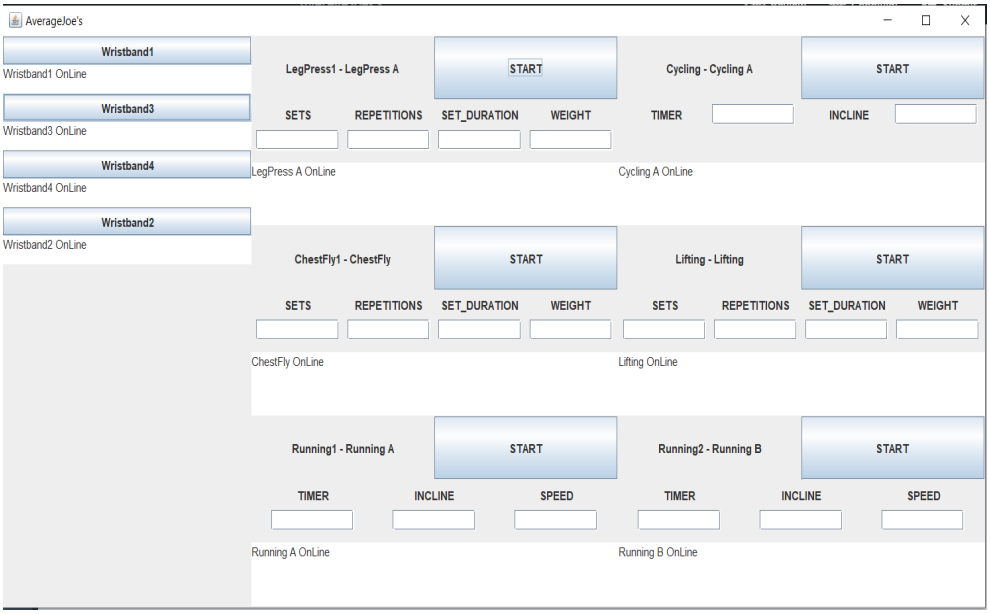

Figura 6.1: Interfaccia grafica

Per interfaccia grafica mostra come l'utente possa utilizzare il simulatore cliccando sul pulsante Wristband e selezionando il macchinario con cui vole connettersi. Una volta che l'operazione sarà andata a buon fine, verranno mostrati i parametri relativi al Customer che possiede il Wristband in utilizzo. L'utente potrà decidere se mantenere i parametri presenti nel TrainingProgram del Customer o cambiarli. Premendo il pulsante start la

simulazione sarà avviata e verranno mostrati i parametri vitali dell'utilizzatore sia sul display della macchina che su quello del Wristband. Durante l'esecuzione verrà anche mostrato se una macchina è stata prenotata e da quale Wristband e la stessa cosa, al contrario, sarà presente sul display del Wristband.

6.2 Suddivisione compiti

6.2.1 Andrea Campidelli

Essendo stato designato come product owner, inizialmente mi sono concentrato sulla composizione delle sequenze di scambio di messaggi tra i vari attori e sulla definizione, insieme con Elena per la parte Machine, della separazione tra la parte hardware simulata e il sistema stesso.

Ho poi cercato, nel più breve tempo possibile per non bloccare gli altri sviluppi, di abbozzare gli attori dei controller e dei componenti della parte fisica, per poi affinare il comportamento e il codice in un secondo momento. In questa fase, il cambiamento più impegnativo, e più soddisfacente, è stato il passaggio alla soluzione *Typed* di *Akka*. Questo ci ha permesso di avere una più chiara definizione dei comportamenti degli attori e del controllo della correttezza anche durante la scrittura del codice, più pulita anche grazie alla sua intrinseca modellabilità in *FSM*.

Rimanendo come riferimento per il confronto sullo scambio di messaggi tra attori diversi e sui processi di massima, ho proseguito creando alcune classi di supporto, tra cui un semplice logger utilizzato per verificare il funzionamento senza la view e per effettuare alcuni test. Insieme a Sokol, mi sono confrontato e abbiamo provato alcune opzioni per la gestione dei parametri di esercizio per le diverse macchine da utilizzare per il Training Program.

Successivamente, abbiamo affinato e armonizzato le comunicazioni tra i vari attori, ricercando nel contempo di sfruttare all'interno del codice le potenzialità di *Scala*. Infine, nel poco tempo rimasto, mi sono concentrato nella ricerca di un metodo che permettesse di testare un intero scenario di funzionamento senza dover verificare ogni singolo passaggio. La soluzione adottata è stata quella di verificare che avvenissero tutti i cambi di *Behaviour* nell'ordine prestabilito e questo implicitamente dovrebbe assicurare il preventivo corretto scambio di messaggi, dato che questi ultimi sono necessari all'atteso cambio di comportamento.

6.2.2 Sokol Guri

Il macro-task di cui mi sono occupato sin dall'inizio del progetto è la progettazione del *customer*. Inizialmente mi sono focalizzato sull'implementazione di una semplice versione di un customer actor, in grado di essere istanziato e cambiare *behaviour*. La gestione di un customer actor diventava più difficile quando nuove funzionalità si dovevano aggiungere, perchè doveva essere disponibile per comunicare con diversi componenti del sistema. Tenendo conto dei vantaggi che *TypedActors* offrono sulla definizione del protocollo di comunicazione con l'insieme dei messaggi accettabili da un certo attore, ho deciso di optare una organizzazione a livelli. Ogni livello rappresenta un tipo di attore, con CustomerManager che svolge il ruolo del supervizor. La progettazione a livelli facilitava lo smistamento dei messaggi e l'intervento quando si scontravano dei messaggi o comportamenti non attendibili. Ogni livello aveva il suo compito ben definito e così un messaggio doveva usare l'entry point, in questo caso CustomerManager, per arrivare alla destinazione. Durante questo scorrimento, il messaggio subiva dei controlli, per confermare che il destinatario sarebbe pronto a ricevere tale messaggio.

Parallelamente alla progettazione del *customer*, mi sono focalizzato alla progettazione dell'allenamento, in particolare alle schede d'allenamento e agli esercizi. Inizialmente si è pensato di gestire anche gli esercizi che non veniva eseguiti tramite un macchinario *smart* e per questo motivo mi sono confrontato con Andrea su come definire la metrica d'esecuzione e i parametri di configurazione. Per la creazione di un esercizio ho implementato un convertitore implicito che estrae dal database un workout e lo converte nel formato di un *Exercise*.

Continuando il lavoro sulle schede d'allenamento, mi sono reso conto che era necessario la gestione dei parametri di configurazione in modo corretto e sicuro. Per questo motivo ho sviluppato *SafePropertyValue* che implicitamente convertiva i parametri in dei valori non negativi.

Insieme ad Elena abbiamo implementato lo storage della palestra, dove lei aveva esteso le funzionalità di un basic storage che avevo implementato all'inizio del progetto per testing. La mia versione iniziale offriva operazioni *CRUD* sullo storage di diversi componenti, considerati entità.

6.2.3 Elena Morelli

Durante tutta la durata del progetto ci siamo divisi sprint per sprint i task da svolgere cercando di mantenere coerenza con i vari ambiti per essere più veloci. Nella stesura del Codice ho cercato di rispettare il più possibile lo stile della programmazione funzionale, cercando di mantenere uno stato immutabile nei punti in cui fosse necessario e di astrarre quando possibile per modularizzare bene tutti gli elementi del sistema.

Una delle prime cose di cui mi sono occupata è dell'implementazione del MachineActor, dove inizialmente avevamo optato per una soluzione untyped. Abbiamo cercato di mantenere la struttura dei messaggi il più simile possibile in modo che fosse semplice richiamarli al momento dell'invio. Per l'implementazione di questo attore mi sono sentita più volte con Andrea per definire bene la suddivisione dei compiti/messaggi tra questo attore e quello della PhysicalMachine.

Una volta conclusa questa parte mi sono focalizzata sulla view. Ho studiato gli elementi base di scala.swing in modo da realizzare una GUI molto semplice che simulasse il display dei macchinari e quello dei braccialetti. Successivamente ho aggiunto una dialog che permettesse al braccialetto di simulare la connessione RFID. La view permette anche all'utente di modificare i parametri degli esercizi che vuole svolgere nel caso siano già fissati all'interno del training set, in caso contrario può aggiungerli a piacimento. Per l'aggiornamento della GUI ho implementato il ViewToolActor creandolo in modo che potesse essere utilizzato da entrambi gli attori che avessero bisogno di aggiornare i propri display.

Altra cosa di cui mi sono occupata è stata la creazione di un database per avere i dati degli esercizi per ciascun customer. Inizialmente avevamo considerato di utilizzare una struttura di salvataggio dati più complessa e cercando avevo trovato interessante la libreria Slick. Una volta notato la difficoltà nella configurazione, causata dalla presenza della libreria Akka, e la scarsità del tempo rimasto ho optato per un semplice salvataggio su file. Per il database ho sfruttato in parte delle classi implementate da Sokol, modificandole rispetto alle nuove necessità.

Ho aggiunto un file di utilities per l'estrazione e l'inserimento di dati da file di tipo JSON. Ho infine creato il WriteOnFileActor per la scrittura su file dei parametri di un esercizio concluso.

Capitolo 7

Retrospettiva

7.1 Andamento del processo di sviluppo

Come già anticipato nel Capitolo 2 il processo di sviluppo si è articolato in sprint.

Sprint 0 Durante questo primo sprint ci si è concentrati sulla definizione del Product Backlog, dei requisiti e del design architetturale. Questo è servito per organizzare al meglio lo sviluppo che sarebbe partito dallo sprint successivo dando un'organizzazione generale del progetto. Si è inoltre creato il repository principale e le fork relative ai membri, così come si configurato tutto l'ambiente di sviluppo.

Sprint 1 In questo sprint si è implementata una comunicazione base tra le entità.

- Creazione degli attori Customer, MachineActor, PhysicalMachine e Device.
- Implementazione di una struttura comune di messaggistica: per poter permettere una maggiore facilità nella creazione di un messaggio.

I problemi principali rilevati alla fine dello sprint possono essere racchiusi nei seguenti punti:

- Sottostima del carico: i task risultavano necessitare dell'implementazione di più entità rispetto a quelle tenute in considerazione.
- Scelte diverse di implementazione: non si era assunto l'utilizzo di attori non-typed senza specificarlo, ciò a fatto si che si creassero incongruenze.

Sprint 2 In questa fase dello sviluppo si è ritenuto opportuno provvedere all'implementazione di alcuni componenti secondari necessari per la creazione degli attori oltre alla sistemazione delle incongruenze rispetto al codice e al protocollo di comunicazione.

- Convertire il MachineActor, PhisicalMachine, Device e in un attori typed con aggiunta dei messaggi mancanti

Durante la sprint review sono stati notati i seguenti problemi:

- Incongruenza sui test: si è virato sull'utilizzo di test SBT nonostante si fosse deciso di utilizzare Gradle in un primo momento.

Sprint 3 Fase di creazione della parte di interazione con l'utente e della gestione dei parametri di esecuzione

- Inizio implementazione della view e relativi attori di update
- Creazione di classi volte alla conversione dei parametri rispetto alla tipologia del macchinario

Durante la sprint review sono stati notati i seguenti problemi:

- Gestione dei parametri di settaggio dei macchinari complicata e poco efficace

Sprint 4 Fase di creazione di una struttura dati persistente, aggiunta della simulazione sui parametri vitali. Creazione dell'entità volta a effettuare il booking dei macchinari.

- Si è creata un CustumerBooking che si occupasse di prenotare un macchinario prima della fine dell'esercizio.
- Creazione di una struttura dati basata su file, con le relative utility di estrazione e inserimento.
- Studio rispetto ad un meccanismo di mantenimento della persistenza dei dati.
- Aggiunta di un timer: gestione della possibilità di rendere un macchinario in stato di booking perenne

Durante la sprint review sono stati notati i seguenti problemi:

- Presenza di meccanismi simili per la gestione dei parametri dei macchinari

Sprint 5 Fase di sviluppo adibita al testing dopo i notevoli cambiamenti aggiunti dall'implementazione della parte del booking e all'aggiunta della possibilità di utilizzo dei macchinari non presenti all'interno del TrainingProgram

- Risoluzione della problematica relativa alla possibilità di avere parametri vuoti.
- Aggiunta del deterioramento del macchinario
- Aggiunta della visualizzazione dei parametri eseguiti dall'utente

7.2 Commenti finali

7.2.1 Andrea Campidelli

Scala è stato una scoperta molto interessante: in altri linguaggi (es. *c*), mi è più artificioso lavorare con parti funzionali, perché risultano meno comprensibili e più difficilmente debuggabili rispetto al resto del codice. Con *Scala*, invece, la scrittura è molto più naturale e "le cose stanno al loro posto", seguendo la fluidità del linguaggio. Anche *Akka* si è rivelato molto stimolante, soprattutto nella sua versione *Typed*.

Trello è uno strumento con cui ho già grande familiarità, mentre *Scrum* mi era noto poiché l'avevo utilizzato, seppur in versione "light", in contesti aziendali. Lo stesso vale per il *TDD*, anch'esso mai abbracciato appieno in ambito lavorativo. Per quanto siano entrambe metodologie che conosco e apprezzo, purtroppo non sono riuscito a sfruttare al meglio questa opportunità per approfondirne la conoscenza, dato che Scala e *Akka* hanno richiesto un notevole impegno in termini di tempo.

In conclusione, dispongo ora di due nuovi strumenti da utilizzare e di un rinnovato impeto per l'approfondimento di due vecchi strumenti da cui non riesco sempre a trarre il massimo, oltre all'esperienza di un progetto gestito completamente da remoto con due colleghi con i quali è nata una proficua collaborazione.

7.2.2 Sokol Guri

Alla fine di questo progetto, posso affermare di avere più familiarità con gli attori di *Akka*, specialmente quelli *Typed*, che prima erano un "mondo" ambiguo e inesplorato per me. Di sicuro progetti basati sull'architettura ad

attori faranno parte del mio futuro. Sperimentare il mondo degli attori con Scala mi ha fatto migliorare la mia comprensione e l'utilizzo di un linguaggio funzionale per lo sviluppo di un progetto.

L'organizzazione del lavoro basandosi sulla tipologia *Scrum*, utilizzando *Trello* è stato un esperienza produttiva, però sicuramente da migliorare per usufruire tutti i vantaggi che offre. All'inizio è stato un po difficile lavorare seguendo questa modalità, però svolgendo dei meeting costantemente, ha facilitato il confronto e lo sviluppo.

Usando i social networks è stato utile ad portare alla conclusione questo progetto, partecipando ad un team mai incontrata prima. Con i tempi che stano arrivando, questo progetto sicuramente mi aiuterà ad affrontare meglio i futuri lavori in modalità *smart-working*.

7.2.3 Elena Morelli

Il partecipare a questo progetto mi ha permesso di acquisire esperienza in tante cose. Una delle principali è stato sperimentare Scrum che ritengo che sia uno strumento molto utile se sfruttato nella giusta maniera. Nel nostro progetto avremmo sicuramente potuto farne un uso migliore, in modo da avere una pianificazione più rigorosa che ci permettesse di rispettare i tempi di consegna. Comprendo solo ora quanto avrebbe potuto aiutarci una buona pianificazione essendo un gruppo formato da persone che non si sono mai viste.

L'utilizzo di scala nel progetto mi ha permesso di fissare meglio alcuni concetti e di capirne meglio le potenzialità, soprattutto per quanto riguarda l'espressività. Durante lo sviluppo ho cercato di fare uso della tecnologia TDD anche se è un aspetto che potrei migliorare. Avrei potuto farne un utilizzo maggiore sopratutto nel testing dell'interazioni con gli altri attori. In conclusione ritengo che il progetto sia andato bene nonostante le molte problematiche iniziali nello stabilire come affrontare lo sviluppo.

Elenco delle figure