ASTRA - Smart Shock Room

Prototipo V0.0.2

Per il funzionamento del primo prototipo del sistema è necessario avere alcuni framework preinstallati nel pc :

- *JaCaMo e relativo plugin per Eclipse* per l'esecuzione dell'ambiente ad agenti
- *Node.js* per l'esecuzione dei servizi REST
- *RabbitMQ* per la gestione delle comunicazioni Publish/Subscribe
- *MongoDB* per la gestione dei database

Il prototipo è composto da più parti da avviare separatamente :

N.B. al primo avvio dopo aver scaricato il repo serve installare le dipendenze dei vari componenti REST, manualmente usando *npm install* in ognuno oppure eseguendo lo script *install_dependencies.sh* che automatizza tale procedura.

L'esecuzione può allo stesso modo essere avviata manualmente per ogni singolo componente o sfruttando lo script *launcher.sh,* il quale avvia tutti i servizi utili contemporaneamente, in finestre separate, sfruttando il pacchetto *nodemon,* installabile tramite *npm.*

N.B. : Lo script presumibilmente funziona solo in ambiente windows

N.B. Ricordarsi di avviare mongoDB prima di avviare l'esecuzione del sistema e controllare che il server RabbitMQ sia attivo.

Conviene avviare prima la parte relativa ai servizi web, che impiega un po' di tempo per compilazione ed avvio e successivamente la parte relativa agli agenti.

---- Componenti del Repo ---

1 - AlexaSkill

Questa cartella contiene tutto il codice relativo alla Skill di Alexa, lo zip contenente gli stessi file e il file audio necessario per la gestione del timeout. Per l'esecuzione è necessario caricare i file all'interno della piattaforma *Alexa developer console,* gestibile tramite browser, tramite la quale è anche possibile fare il deploy nel dispositivo fisico collegato al proprio account.

2 - Astra Room Assistant

Questa cartella rappresenta il progetto JaCaMo che gestisce la parte ad agenti del sistema. Al momento non è stato esportato un jar dato che non si ha una versione stabile del sistema, per cui per l'esecuzione dovrà essere importato il progetto in eclipse ed eseguito tramite l'IDE, oppure eseguito manualmente da linea di comando.
N.B. tale componente non è eseguito automaticamente dallo script, va lanciato separatamente

3 - Command Monitor

Questo componente ha funzioni di test e realizza le statistiche sul tempo di esecuzione dei comandi. È stato realizzato usando Angular.
Va lanciato tramite *nodemon* o *ng serve --port 3011.*

Il servizio è attivo sulla porta 3011.

N.B. tale componente non è eseguito automaticamente dallo script, va lanciato separatamente

4a - DisplayClient

Questo componente rappresenta il display fisico sulla quale vengono mostrati i dati, gestendo la parte di visualizzazione e gestione del layout. È stato realizzato usando Angular.

Va lanciato tramite *nodemon* o *ng serve --port 3000.*

Il servizio è attivo sulla porta 3000 e comunica con il relativo servizio tramite webSocket.

4b - DisplayService

Questo componente rappresenta il servizio web con cui richiedere la visualizzazione di dati sul display. È stato realizzato usando NodeJs, Express e MongoDB.

Va lanciato tramite *nodemon* o *npm start.*

Il servizio è attivo sulla porta 3001.

5 - Room Command Manager

Questo componente rappresenta il servizio web con cui vengono gestiti i comandi inviati al sistema . È stato realizzato usando NodeJs, Express e MongoDB.

Va lanciato tramite *nodemon* o *npm start.*

Il servizio è attivo sulla porta 3010.

6a - Active Trauma Service

Questo componente rappresenta il servizio web con cui vengono gestite le informazioni relative al trauma originate da TraumaTracker e i parametri vitali, al momento soltanto simulati casualmente . È stato realizzato usando NodeJs, Express e MongoDB.

Va lanciato tramite *nodemon* o *npm start.*

Il servizio è attivo sulla porta 3005.

6b - MockSource Service

Questo componente rappresenta il servizio web a cui è possibile richiedere dati mock, sia testuali che immagini, per testare altri componenti . È stato realizzato usando NodeJs, Express e MongoDB.

Va lanciato tramite *nodemon* o *npm start.*

Il servizio è attivo sulla porta 3007.

6c - TAC Source

Questo componente rappresenta il servizio di gestione del macchinario della TAC, tramite cui è possibile gestire l'aggiunta di nuovi esami e modificare lo stato della stessa. È stato realizzato con Angular.

Va lanciato tramite *nodemon* o *ng serve --port 3002 .*

Il servizio è attivo sulla porta 3002 e comunica tramite Websocket con il relativo servizio

6c - TAC Source Service

Questo componente rappresenta il servizio web relativo alla TAC.

Va lanciato tramite *nodemon* o *npm start .*

Il servizio è attivo sulla porta 3003.