Università degli studi di Genova

Progetto di Transactional systems and data warehouse

Manuale: Drone and Mission Manager Bot

Versione 1.0

Alessio Gilardi, Alessio Bollea, Luca Defilippi

coordinato da:
Prof. Gianni Vercelli

Indice

1 Introduzione

1.1 Descrizione generale

Il *Drone and Mission Manager Bot* è stato creato per la *specialità droni* della Protezione Civile Nazionale Italiana, come progetto per il corso di *Transactional systems and data warehouse*, tenuto presso il Dipartimento di Informatica, Bioingegneria, Robotica e Ingegneria dei Sistemi (DIBRIS) dell'Università degli studi di Genova, dal Prof. Antonio Boccalatte.
Lo sviluppo è stato coordinato dal Prof. Gianni Vercelli.
Lo scopo del *bot* è quello di fornire un'interfaccia semplificata e integrata per gestire: emergenze, esercitazioni o attività nelle quali è richiesto l'uso di droni.

1.1.1 Bot telegram

I Bot Telegram sono applicazioni di terze parti che girano nell'ambito di Telegram stesso.
Il lettore interessato potrà trovare un approfondimento riguardante i bot telegram al seguente link (in inglese).

1.2 Motivazioni

La scelta della piattaforma Telegram si è resa necessaria per ragioni di portabilità, essendo essa disponibile per diversi sistemi operativi, oltre che ad una versione web, accessibile direttamente dal browser.

1.3 Pre-requisiti

Applicazione Telegram (Desktop, Android, iOS, web...).

1.4 Ottenere telegram

L'applicazione Telegram può essere scaricata gratuitamente. La si può trovare, a seconda della piattaforma:

- **Android**: Google Play Store;
- **iOS**: Apple App Store;
- **Desktop**: sito web;
- **Web**: link.

1.5 Tecnologie utilizzate

Per lo sviluppo ci si è avvalsi delle seguenti tecnologie:

- **NodeJS ®**: piattaforma *open-source* per l'esecuzione di codice Javascript lato server;
- **MongoDB ®**: *database administrator* (DBMS) non relazionale;
- **Mongoose**: *Object Data Modeling* (ODM) per MongoDB;
- **Telegraf**: *framework* per lo sviluppo di bot telegram;
- **Telegram**: servizio di messaggistica istantanea basato su *cloud*;
- **Visual Studio Code ®**: editor di testo avanzato, utilizzato per la stesura del codice.

2 Comandi e actions

2.1 Comandi

Di seguito, una lista dei comandi disponibili.

- ***/requestMission***: comando usato dall'**AM** per istanziare una nuova Missione.
  Vengono richieste:
  - data e ora (approssimative) della Missione;
  - Base alla quale viene affidata la Missione;
  - luogo della Missione (inseribile sia con coordinate GPS che UTM).
  - scenario e difficoltà della Missione

  Una volta richiesta una Missione il Responsabile di Base della Base specificata viene contattato: gli viene inviato un *Button* contenente l'*action* di *organizeMission*.
- ***/addLogbook***: consente ai piloti che hanno partecipato a una Missione per cui non hanno ancora aggiunto la documentazione di aggiungere il riferimento al loro *Logbook*;

- ***/addQtb***: consente di aggiungere il **QTB** generato da un Drone per una Missione.
  Una volta lanciato il comando, vengono presentati i droni ai quali risulta mancante almeno un **QTB**. Per ogni Drone viene inoltre mostrata una lista di missioni alle quali quel Drone ha partecipato e per le quali non è ancora stato inserito il **QTB**.
  In ogni Missione è presente un bottone, che l'utente può premere per iniziare la procedura di inserimento dei dati del **QTB** per quella specifica Missione e quel particolare Drone. Vengono richieste le seguenti informazioni:

  – Codice identificativo del documento;
  – Istanti di inizio e fine dei singoli voli effettuati dal Drone;
  – id dei piloti, uno per ogni volo precedentemente inserito.

  Una volta terminata l'operazione, il **QTB** viene aggiunto al database e ai piloti partecipanti vengono aggiornate le ore di volo effettuale in totale.

- ***/manageDrones***: permette al Responsabile di Base o ai manutentori di gestire i droni.
  Una volta lanciato il comando, viene chiesto all'utente di inserire un filtro per la ricerca; i filtri possibili sono:

  – Ricerca droni con una Missione programmata per il giorno stesso;
  – Ricerca droni disponibili;
  – Ricerca droni in manutenzione;

  In base al filtro scelto vengono mostrati i droni corrispondenti. Inoltre, nel caso in cui il filtro selezionato sia *per disponibilità*, viene mostrato all'utente -per ogni Drone- un pulsante per inviare quel Drone alla manutenzione; nel caso in cui, invece, il filtro selezionato sia *per droni in manutenzione*, verrà mostrato all'utente un pulsante per terminare la manutenzione di un Drone.

In qualsiasi momento è possibile visualizzare una lista dei comandi disponibili digitando */help*.

2.2 Actions

Le *actions* sono generalmente azioni scatenate dalla pressione di un *Button*. Di seguito, una lista delle *actions* implementate:

- ***organizeMission***: premendo il pulsante corrispondente, l'**AM** viene notificato della presa in carico della Missione e al Responsabile di Base viene mostrata una lista di droni disponibili, tra cui può scegliere quali usare.
  Fatto ciò viene inviata una notifica al Personale di Base: viene inviato un messaggio con due pulsanti: *Accept* o *Decline*.
  Il ruolo che la particolare persona ricoprirà verrà specificato in seguito dal Responsabile di Base.

- ***acceptMission***: permette di accettare una Missione.
  Ogniqualvolta un membro del personale accetta una Missione, viene eseguito un controllo per verificare se il numero complessivo di coloro che hanno accettato è tale da poter formare un *Team*. In caso affermativo viene notificato il Responsabile di Base, indicando nello specifico i membri che hanno accettato (il messaggio si aggiorna automaticamente ogni volta che nuove persone accettano).
  Al suddetto messaggio è allegato un pulsante *createTeam*.

- ***declineMission***: la Missione viene rifiutata e non si viene più notificati a riguardo.

- ***abortMission***: azione utilizzabile dal Responsabile di Base per bloccare una Missione per la quale non si riesce a trovare personale sufficiente.
  In caso di *abort* tutto il Personale coinvolto è informato a riguardo:

  - Responsabile di Base della base principale;
  - **AM**;
  - Responsabile di Base delle basi secondarie;
  - Personale che aveva accettato o che non aveva risposto.

- ***createTeam***: consente al Responsabile di Base di creare un team scegliendo tra le persone che hanno accettato.

- ***extendToBase***: premendo sul pulsante con il nome della base, la richiesta di personale per una Missione viene estesa alla Base specifica. Il Responsabile di Base della nuova base è notificato e il personale disponibile è contattato: se accetta sarà mostrato al Responsabile di Base della base principale.

3 Timeouts

Nell'applicazione vengono implementati 3 *timeout*.

- ***organize***;
- ***extend***;
- ***global***.

3.1 Organize

Timeout per il Responsabile di Base: se il Responsabile di Base non inizia ad organizzare una Missione entro un tempo limite viene contattato l'**AM** che ha richiesto la Missione.

3.2 Extend

Se entro un tempo limite il *team* della Missione non viene creato, il Responsabile di Base viene contattato: gli vengono mostrate le persone che hanno accettato, rifiutato o che non hanno risposto. Gli viene poi domandato se vuole estendere la ricerca di personale ad altre Basi e, in tal caso, premendo sul pulsante relativo alla base si innesca la procedura di estensione della richiesta. Altrimenti, gli viene domandato se vuole abortire la Missione.

NOTA: se il Personale notificato inizialmente dovesse essere insufficiente per affrontare la Missione, questo *timeout* sarà portato a 0 e quindi il Responsabile di Base riceverà la possibilità di estendere la ricerca immediatamente.

3.3 Global

Timeout maggiore: quando viene superato, senza che sia stato creato un *Team* viene chiesto al Responsabile di Base se vuole abortire la Missione perché è passato troppo tempo.

4 Logica di implementazione

L'applicazione è stata implementata secondo una *logica procedurale*: non è stata utilizzata logica ad oggetti poiché, da subito, non è parso necessario. In ogni caso, ***Javascript*** consente una notevole modularità nello sviluppo. Quindi, l'applicazione è stata suddivisa in diversi moduli, cercando di separare il più possibile le varie logiche implementative, con l'intento di rendere più semplici future modifiche.
L'applicazione si presenta così strutturata: nella cartella principale sono presenti 3 file di configurazione, 2 file ***Javascript*** ed altre sottocartelle.
I file di configurazione sono:

1. **.env**: file di configurazione di default per applicazioni *Node.js* utilizzando il modulo *dotenv*, in questo file sono presenti dati di configurazione sull'ambiente, quindi i dati di connessione al database MongoDB e il *token* di accesso al *bot* Telegram.
2. **risk-matrix.txt**: file che contiene la matrice di rischio, al fine di indicare la difficoltà e lo scenario della Missione.
3. **timeouts.json**: file che contiene i valori (in minuti) dei vari *timeout* (*organize*, *extend* e *global*).

I file ***Javascript*** sono:

- ***app.js***: è il modulo principale dell'applicazione, ha il compito di caricare gli altri moduli e lanciare funzionalità specifiche in base all'interazione con gli utenti.
- ***utils.js***: contiene funzioni di carattere generale, non strettamente legate al progetto.

Le altre cartelle presenti sono:

- **node_modules**: contiene i moduli di terze parti utilizzati e le loro dipendenze.
- **bot**: contiene i file:
  - *bot-functions.js* che ha al suo interno alcune funzioni specifiche per il *bot*, come il caricamento dei comandi utilizzabili dall'utente
  - *router.js*, modulo che ha il compito di instradare la pressione di un bottone verso l'*handler* corretto.

e le cartelle:

  - ***actions***: azioni di risposta alla pressione di bottoni che non hanno bisogno di essere gestite con una logica a *step* successivi.
  - ***scenes***: contiene i moduli che gestiscono comandi (es: "/requestMission") oppure risposte a pressione di bottoni che però hanno bisogno di una logica a *step* successivi.
  - ***middlewares***: i *middlewares* sono piccole porzioni di *software* con un compito molto specifico, vengono chiamati ogni volta che è stata compiuta un'azione da parte dell'utente. Di seguito sono presenti due middleware:
    * **data-loader.js**: controlla se i dati dell'utente sono stati caricati in sessione e in caso contrario li carica mediante una query al DB (utile per non dover ricaricare i dati di volta in volta, ma solo alla scadenza della sessione o della validità dei dati).
    * **command-permission-check.js**: verifica, nel caso sia stato ricevuto un comando (es: "/requestMission"), che l'utente abbia i diritti necessari per utilizzarlo (es: l'utente deve essere **AM**).
  - ***timeouts***: contiene l'implementazione della logica dei tre *timeout* (*organize*, *extend* e *global*).

- **db**: contiene i moduli di connessione e disconnessione dal **DB** ("*db-connect.js*"), i modelli e gli schemi delle *Collections* del ***DB noSQL*** ("*models.js*", "*schemas.js*") e il modulo per effettuare le query alle spefiche *Collections* ("*queries.js*").

- **events**: contiene i moduli che gestiscono eventi scatenati in seguito al completamento di un'azione (es: "*onMissionOrganized*" è eseguito in cascata alla *scene* "*organizeMission*" e si occupa, tra l'altro, di contattare il personale di base). Sono presenti due moduli (*event-emitters.js* e *event-register.js*), i quali permettono di creare e registrare i vari eventi sull'*handler* giusto, poi nella *sub folder bot* sono presenti i vari *handlder*, ognuno relativo ad un evento:

  - ***on-mission-accepted.js***: Ogni volta che una persona accetta una Missione, si controlla se è pronto un *team* e, nel caso, viene notificato il Responsabile di Base e viene inserito l'evento *missionAccepted.* nell'*EventLogbook.*

- ***on-mission-organized.js***: quando la Missione è organizzata viene contattato il personale di base.
- ***on-mission-requested.js***: Modulo che gestisce l'evento *request-Mission*.
  Il modulo si occupa di gestire gli eventi conseguenti alla richiesta di una Missione:
  1. Invia una notifica al Responsabile di Base della Missione richiedendo di prenderla in carico;
  2. Viene inserito l'evento nell'*EventLog*.
- ***on-team-created.js***: *event handler* che gestisce le operazioni successive alla creazione di un *team*:
  1. Si notifica chi è stato aggiunto alla Missione;
  2. Si aggiunge la Missione alle *waitingForLogbook* (implementato in funzione periodica: *checkTodayMissions*);
  3. Si avvisa chi non è stato scelto;
  4. Si rimuove la Missione da quelle accettate a chi non è stato scelto, in modo da renderlo nuovamente disponibile;
  5. Si inserisce l'evento nell'*Event Log*.

5 Possibili miglioramenti

- Migliorare interazione con l'utente.
- Migliorare la lettura dell'*input*, rendere il sistema più robusto ad *input* inaspettati (es. bottoni precedenti non ancora premuti) e aumentare la possibilità di annullamento dell'operazione corrente, ritornando allo stato precedente.
- Migliorare la gestione delle *scenes*, ad esempio eliminando i messaggi scambiati.
- Migliorare l'interfaccia con il DB, riorganizzare le *query*, sfruttando meglio la libreria *Mongoose*.

6 Definizioni e acronimi

- **QTB**: Quaderno Tecnico di Bordo;
- **AM**: *Accountable Manager*;

- **DB**: *DataBase*.

Genova, 28 marzo 2019