Relazione Snap4All

Alberto Del Buono Paolini - Federico Marra

Novembre 2022

Indice

1 Introduzione

Il punto di partenza di questo progetto è stata l'applicazione Snap4All già esistente che riscontrava vari problemi, non solo di usabilità ma anche di funzionalità di base. Queste sono state le conseguenze del mancato aggiornamento del pacchetto da alcuni anni:

- Installazione dei pacchetti `apt` necessari non funzionante.
- Mancanza di documentazione, sia nell'applicazione che nel codice stesso.
- Interfaccia grafica per l'utente obsoleta.

Per ogni capitolo che lo richieda saranno proposti blocchi di codice come esempio del lavoro descritto nello stesso. Ogni risorsa usata sarà inclusa sotto forma di nota a piè di pagina in corrispondenza del suo utilizzo. L'applicazione precedente è stata sviluppata nel 2018 da Steven Salazar[1], di cui alleghiamo la relazione qui[2].

2 Adattamento alla nuova versione di Termux

Per risolvere il primo, e anche più importante, dei problemi posti abbiamo dovuto aggiornare completamente l'applicazione ripartendo da una fork dell'ultima versione di Termux, disponibile su GitHub[3]. In quanto fork di Termux abbiamo anche ereditato[4] la loro licenza `GPLv3` per l'applicazione e le licenze `Apache 2.0` per i sottopacchetti `terminal-view` e `terminal-emulator`.

2.1 Merge tra Termux aggiornato e la vecchia applicazione Snap4All

Abbiamo riportato tutti i frammenti di codice segnalati con il commento `TERMUX MERGE` dalla codebase precedente alla nuova fork[5] appena creata. Purtroppo questo non è stato abbastanza dato che non tutti i cambiamenti al codice

[1] https://github.com/StevenSalazarM

[2] https://github.com/albbus-stack/snap4all/blob/master/old-relation/StevenSalazarOldRelation.pdf

[3] https://github.com/termux/termux-app

[4] https://github.com/termux/termux-app/blob/master/LICENSE.md

[5] https://github.com/albbus-stack/snap4all

di Termux originale erano stati segnalati con tale commento. Oltretutto alcuni file modificati non erano più presenti nella stessa forma dato che il codice relativo a Termux aveva subito aggiornamenti. Adesso tutti i cambiamenti alla repository Termux originale sono marcati con `TERMUX MERGE` e facilmente rintracciabili cercando `MERGE` su tutta la codebase.

TermuxActivity.java – Esempio di commento di merge

```
// START TERMUX MERGE

// Enables the console button on the MainActivity if there is already a
    session
MainActivity.activity.btnConsole.setEnabled(true);

// END TERMUX MERGE
```

2.2 Aggiornamento di `termux-api-package`

Chiaramente anche il sottopacchetto di Termux responsabile per le interazioni col sistema Android, cioè `termux-api-package`, era stato aggiornato con molta frequenza negli ultimi anni. Abbiamo quindi fatto una fork[6] di tale pacchetto, includendo quindi anche i cambiamenti necessari per il funzionamento dei comandi personalizzati `termux-enable-buttons`, `termux-bluetooth-scaninfo` e `termux-bluetooth-connect`. Questi ultimi chiamano le corrispondenti funzioni Java localizzate nel modulo `termux-api` che ne implementano le funzionalità.

termux-enable-buttons.sh

```
#!/data/data/com.termux/files/usr/bin/sh
set -e -u

SCRIPTNAME=termux-enable-buttons
show_usage () {
```

[6] https://github.com/albbus-stack/snap4all-termux-api-package

```
    echo "Usage: \$SCRIPTNAME"
    echo "Enables the node-red button."
    exit 0
}
...
/data/data/com.termux/files/usr/libexec/termux-api EnableButtons
```

2.3 Aggiornamento dei pacchetti di bootstrap

Nella versione precedente di Snap4All i pacchetti di bootstrap, cioè degli archivi compressi che contengono il filesystem di Termux, erano scaricati da un server esterno nel momento in cui veniva fatta la build dell'applicazione. Questo metodo non è solo lento ma preclude anche la modifica di tali archivi, essendo serviti da un server pubblico.

Abbiamo optato per hostarli su Github e quindi usare sempre gli stessi archivi localmente a buildtime. Questa modalità ci permette di generare archivi di bootstrap con qualsiasi pacchetto `apt` di cui necessitiamo, per poi farne l'upload su Github, grazie allo script `generate-bootstrap.sh`[7]. Tale script fa parte della repository `termux-packages`[8] e insieme a `run-docker.sh` ci permette di generare consistentemente gli archivi di bootstrap per ogni architettura. Nel nostro caso era necessario installare vari pacchetti:

- `git`: usato per scaricare da Github la fork di `termux-api-package`.
- `make, cmake, clang`: usati per installare `termux-api-package`.
- `node-js`: usato per installare `node-red`.
- `coreutils, openssh, nano`: tool di base per un sistema Unix, incluso l'editor testuale `nano` per comodità nella modifica di file di sistema dalla console.

Segue il nostro uso dei due script di `termux-packages`:

[7] https://github.com/termux/termux-packages/blob/master/scripts/generate-bootstraps.sh

[8] https://github.com/termux/termux-packages

shell

```
> ./run-docker.sh
> ./generate-bootstrap.sh -a git, make, cmake, clang, coreutils, nano,
nodejs, openssh
```

2.4 Fork di `node-red-contrib-termux-api`

Durante il testing dell'applicazione si sono verificati vari errori, tra cui uno relativo al pacchetto `node-red-contrib-termux-api`. Questo errore si è presentato utilizzando il nodo `Camera Photo` ed era dovuto alla chiamata incorretta della funzione `fs.unlink`, il cui compito è quello di cancellare la foto temporanea che è stata scattata dall'api di Termux. Abbiamo quindi fatto una fork[9] di tale pacchetto andando a cambiare la chiamata ad `unlink`, aggiungendo come secondo argomento un callback che viene eseguito al termine di questa funzione. Controllando gli altri nodi di questo pacchetto, non abbiamo trovato nessun' altra funzione che avesse una signature deprecata.

index.js

```
fs.readFile(photoFile, (err, data) => {
    if (err) throw err;
    msg.payload = data;
    node.send(msg);

    // This is where the live npm package fails, the signature of the
    unlink function used was incorrect, having no callback as the second
    argument
    fs.unlink(photoFile, (err) => {
        if (err) throw err;
    });
});
```

[9] https://github.com/albbus-stack/node-red-contrib-termux-api

2.5 Pulizia e documentazione del codice

Script d'installazione (parte di TermuxService)

Abbiamo modificato lo script d'installazione per essere leggibile in futuro, è stato infatti diviso in parti chiaramente distinte, rendendolo più facilmente estendibile. Inoltre le risorse esterne necessarie per installare le fork sopra discusse e inserire i flows Node-RED di esempio sono tutte reperite dalle rispettive repository Github, rendendo l'aggiornamento di tali molto più semplice e modulare.

TermuxService.java

```
String setupScript =
"#!/data/data/com.termux/files/usr/bin/sh\n" +
// Acquire wakelock
"termux-wake-lock\n"+
// Npm executable setup for termux
"[ -f "+isInstalled+" ] || chmod +x "+PREFIX_PATH+"/bin/npm\n"+
// Installs a custom termux-api package
"[ -d "+apiPackagePath+" ] || pkg install clang -o
    DPkg::Options::=\"--force-confold\" --no-upgrade -y\n"+
"[ -d "+apiPackagePath+" ] || git clone
    https://github.com/albbus-stack/snap4all-termux-api-package\n"+
"[ -f "+apiPackagePath+"/Makefile ] || ( cd snap4all-termux-api-package
    && cmake CMakeLists.txt )\n"+
"[ -f "+apiPackagePath+"/libtermux-api.so ] || make\n"+
"[ -f "+apiPackageInstallPath+" ] || make install\n"+
// Installs node-red
...
// Installs node-red-contrib-termux-api
...
// Changing index.js of node-red-contrib-termux-api to an updated one
    with the new node.js function signatures
"[ -f "+isInstalled+" ] || cd $HOME\n"+
```

```
"[ -f "+isInstalled+" ] || curl
  . https://raw.githubusercontent.com/albbus-stack/node-red-contrib-
  . termux-api/master/index.js >
  . index.js\n"+
"[ -f "+isInstalled+" ] || mv index.js ../usr/lib/node_modules/node-
  . red/node_modules/node-red-contrib-termux-api/\n"+
// Installs node-red-dashboard
...
// Installs node-red-contrib-snap4city-user
...
// Installs node-red-contrib-snap4city-developer
...
// Downloads and moves the node-red flows example to the node-red folder
"[ -f "+isInstalled+" ] || cd $HOME\n"+
"[ -f "+isInstalled+" ] || curl
  . https://raw.githubusercontent.com/albbus-
  . stack/snap4all/master/flows.json >
  . flows.json\n"+
"[ -f "+isInstalled+" ] || mv -f flows.json .node-red/\n"+
// Enables the buttons on the main page and starts the node-red server,
  . this part executes every time on boot since it has no modifiers
"[ -f "+isInstalled+" ] || touch $HOME/installed\n"+
"termux-toast \"starting node-red\" && "+vibration+"\n"+
"termux-enable-buttons\n"+
// Runs node-red and, if it terminates with an error, will run this
  . script again
"node "+nodeRedPath+" || $HOME/.termux/boot/start\n";
```

Documentazione del codice e rimozione dei log

Abbiamo documentato consistentemente tutte le parti del codice preesistente e del codice da noi aggiunto, conseguentemente abbiamo aggiornato parti di codice deprecate e rimosso tutti i log di debug superflui, mantenendo quelli relativi alla gestione di eccezioni.

Esempi di documentazione

```
// A boolean to store the installation status
public static boolean installed=false;

// An Intent used to launch and reference the MainActivity
public static Intent MainOptions;
...
// Changes the console ActionBar title
Objects.requireNonNull(getSupportActionBar()).setTitle("Console");

// Starts the MainActivity on creation
MainOptions = new Intent(this, MainActivity.class);
startActivity(MainOptions);
...
// Set pending intent, pointing to the MainActivity, to be launched when
    notification is clicked
TermuxActivity.MainOptions.setFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP);
PendingIntent contentIntent = PendingIntent.getActivity(this, 0,
    TermuxActivity.MainOptions, 0);
...
// Enables the console button on the MainActivity if the setup was
    successful
MainActivity.activity.btnConsole.setEnabled(true);
```

3 Nuove funzionalità

3.1 Riavvio non necessario

Nella versione precedente dei Snap4All era necessario riavviare il dispositivo in modo tale che Termux eseguisse lo script d'installazione tramite il modulo `Termux:Boot` dopo l'avvio. Questa modalità d'installazione risultava in un'attesa non necessaria da parte dell'utente (oltre a non essere molto affidabile su alcuni dispositivi), dato che possiamo chiamare lo script d'installazione ogni volta che una sessione di una console Termux viene inizializzata. Abbiamo questa possibilità grazie ad una funzionalità base di ogni sistema Unix che

utilizza bash come shell, cioè quella di chiamare il nostro script dal .bashrc. Ogni comando scritto in questo file viene quindi eseguito ad ogni startup di bash.

TermuxService.java

```
String bashRcScript = "./.termux/boot/start";
```

3.2 Velocità d'installazione

I vecchi archivi di bootstrap che venivano usati nell'installazione di Termux non contenevano alcun pacchetto apt aggiuntivo, rendendo così il successivo processo d'installazione molto più lungo del necessario. Aggiungendo direttamente i pacchetti di cui abbiamo bisogno nel filesystem di Termux andiamo soltanto ad aumentare il peso dell'applicativo finale, riducendo drasticamente i tempi di attesa durante il setup di Snap4All.

3.2.1 Build con bootstrap completi tramite Linode

Per buildare l'applicativo completo con gli archivi di bootstrap aggiornati serve una macchina con almeno 8 GB di RAM da allocare esclusivamente al processo Java di build; questo è dovuto appunto alla grande dimensione degli .zip da decomprimere. Avendo quindi una limitazione fisica molto comune, abbiamo dovuto impostare un server dedito alla costruzione degli .apk completi, generalizzando questo processo il più possibile. Creando generate-and-build.sh[10] abbiamo reso replicabile la nostra configurazione del server. Questo script ha il compito di:

1. Generare gli archivi di bootstrap con inclusi i pacchetti specificati in input.
2. Installare Java e l'SDK Android.
3. Clonare o aggiornare la repository di Snap4All.
4. Compiere la build dell'applicazione, copiando gli .apk generati in una cartella immediatamente raggiungibile.

[10] https://github.com/albbus-stack/snap4all/blob/compile-branch/generate-and-build.sh

Abbiamo usato un server hostato da Linode per la velocità di connessione, l'ampia disponibilità di RAM e la facilità di replicazione di tale ambiente; chiaramente lo script è utilizzabile su qualsiasi altro server/sistema Unix. La macchina da noi usata è stata un `Linode 16 GB` con installato `Debian 11`.

3.3 Remake dell'interfaccia utente

Layout della pagina home (*MainActivity*)
Usando i componenti di `MaterialUI` disponibili tramite la libreria `material-components-android`[11], abbiamo aggiornato il design della pagina home. Adesso i quattro bottoni presenti sono `MaterialButtons` che implementano gli stati *"attivo"* e *"disabilitato"*, non solo in termine di colore del bottone ma anche dell'opacità dell'icona relativa. Abbiamo anche aggiunto, insieme al titolo dell'applicazione, un sottotitolo che descrive stringatamente le funzionalità di Snap4All. *Figura (a) a pagina 12*.

Layout della pagina informazioni e della console (*InfoActivity* e *MainActivity*)
Abbiamo modificato successivamente la vecchia pagina di informazione sull'applicazione, costituita da una `WebView` che mostra il file `info.html`[12]. Dunque abbiamo riscritto la documentazione che spiega il funzionamento di Snap4All e indirizza l'utente verso altre fonti per approfondire l'utilizzo da parte sua dei singoli pacchetti inclusi nell'applicazione. Per rendere il design consistente con la pagina home abbiamo inserito del `CSS` dentro al file sopra citato. *Figura (b) a pagina 12*.

Nella vecchia versione era presente un problema con la sidebar di Termux (dove sono elencate le varie sessioni attive della console): non venivano mostrate le icone e lo sfondo era completamente trasparente, rendendone così impossibile l'utilizzo. *Figura (c) a pagina 12*.

[11] https://github.com/material-components/material-components-android

[12] https://github.com/albbus-stack/snap4all/blob/master/app/src/main/assets/info.html

Asset vettoriali e azioni rapide

Per concludere la parte di interfaccia utente abbiamo ritracciato vettorialmente il logo di Snap4City[13] per essere usato come `.svg`, cioè rendendolo completamente scalabile a qualsiasi risoluzione. Ne abbiamo proposto due versioni, una colorata pari all'originale e una monocromatica più minimale per essere usata come icona della notifica. Queste sono incluse direttamente in una cartella[14] della repository Github. Abbiamo inoltre modificato per consistenza le icone relative ai tasti per le azioni rapide. *Figure (d) e (e) a pagina 12*.

4 Flows Node-RED Snap4City di esempio

Abbiamo successivamente creato due flow di esempio che fanno uso delle librerie `node-red-contrib-snap4city-user`[15] e `node-red-contrib-snap4city-developer`[16], facenti parte della suite di nodi Snap4City. Questi sono aggiunti automaticamente durante l'installazione, sempre nello script di setup, andando a copiare dentro la cartella `.node-red` il file `flows.json`[17] in modo tale che Node-RED carichi questi flows quando inizializza il server locale. Per utilizzare correttamente i primi due flows di esempio è necessario configurare le credenziali a Snap4City seguendo il tutorial presente nella pagina di informazione.

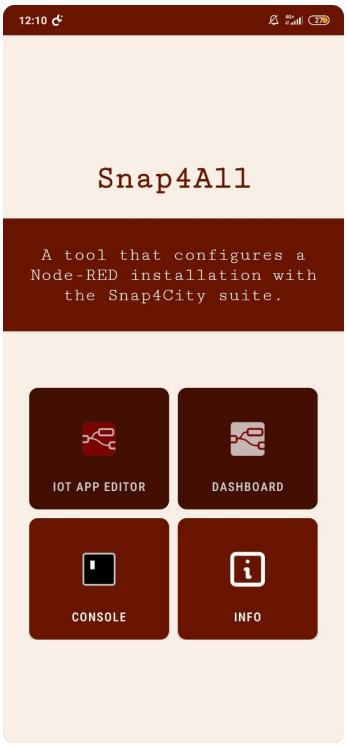
[13] https://www.snap4city.org/

[14] https://github.com/albbus-stack/snap4all/tree/master/logos

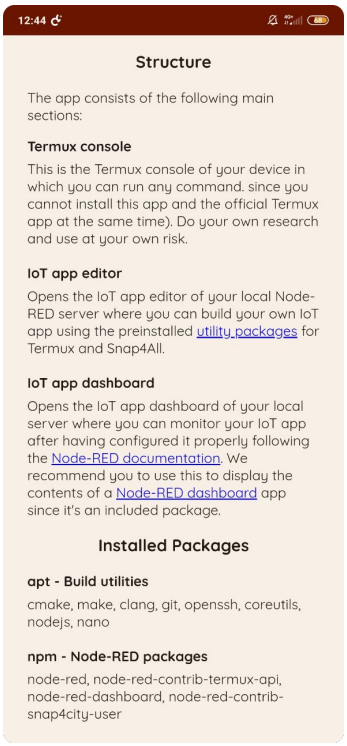
[15] https://github.com/disit/node-red-contrib-snap4city-user

[16] https://github.com/disit/node-red-contrib-snap4city-developer

[17] https://github.com/albbus-stack/snap4all/blob/master/app/src/main/java/com/termux/app/flows-example.json

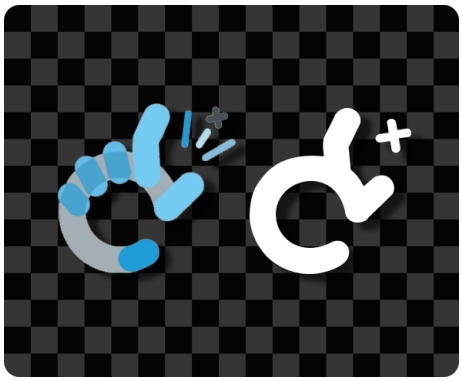


(a) Home

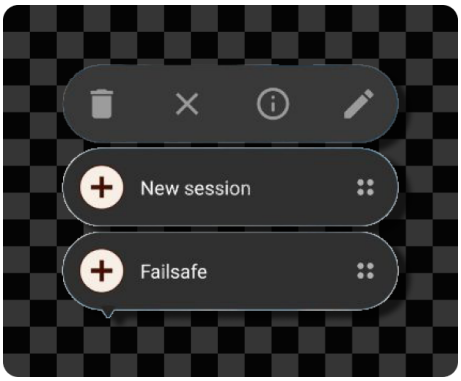


(b) Info

(c) Console

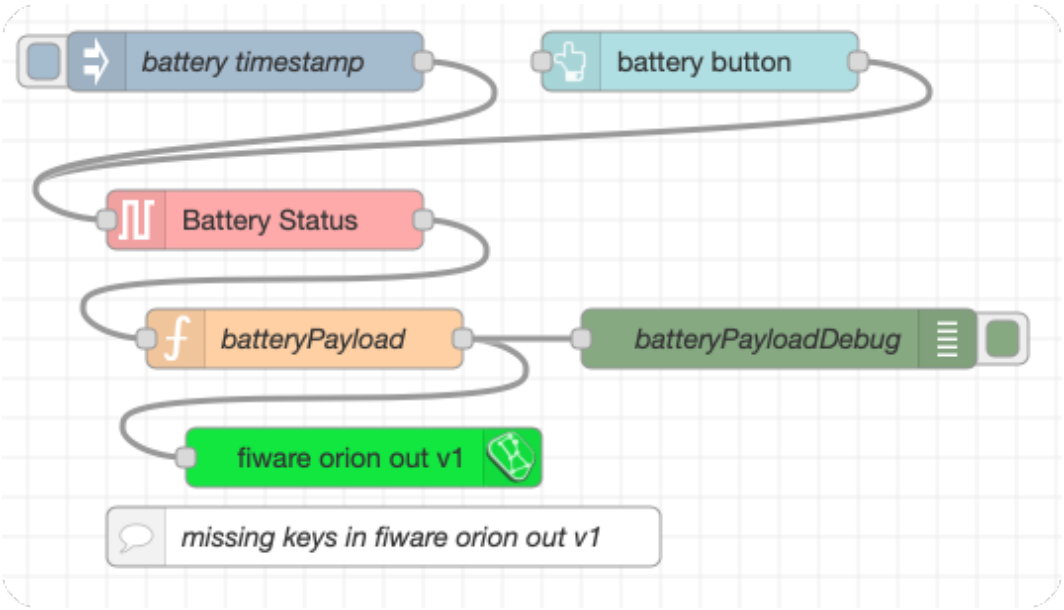


(d) Loghi vettoriali



(e) Azioni rapide

4.1 Primo flow di esempio - Batteria



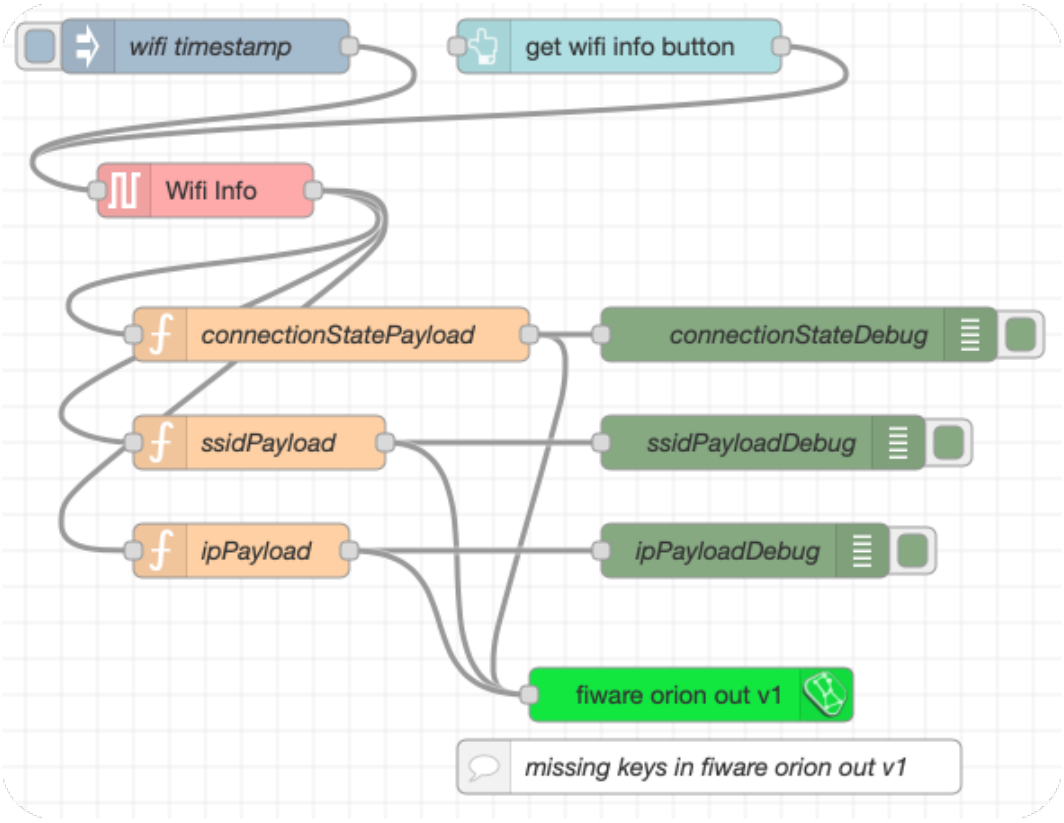
Il flow estrae la percentuale della batteria del dispositivo tramite il nodo `termux-battery-status`[18]. Questa stringa viene poi convertita ad un `float` nel nodo funzione. Successivamente il `msg.payload` viene trasformato in stringa e passato al nodo `fiware orion out` che provvederà a mandarlo al broker di Snap4City, contenente un `IOT device` con un certo nome corrispondente ai due campi `key_1` e `key_2` compilati durante il setup del nodo.

Nodo batteryPayload

```
msg.payload = {
    "name": "battery",
    "value": parseFloat(msg.payload.percentage),
    "type": "float"
};
msg.payload = JSON.stringify(msg.payload);
return msg;
```

[18] https://wiki.termux.com/wiki/Termux-battery-status

4.2 Secondo flow di esempio - Wifi



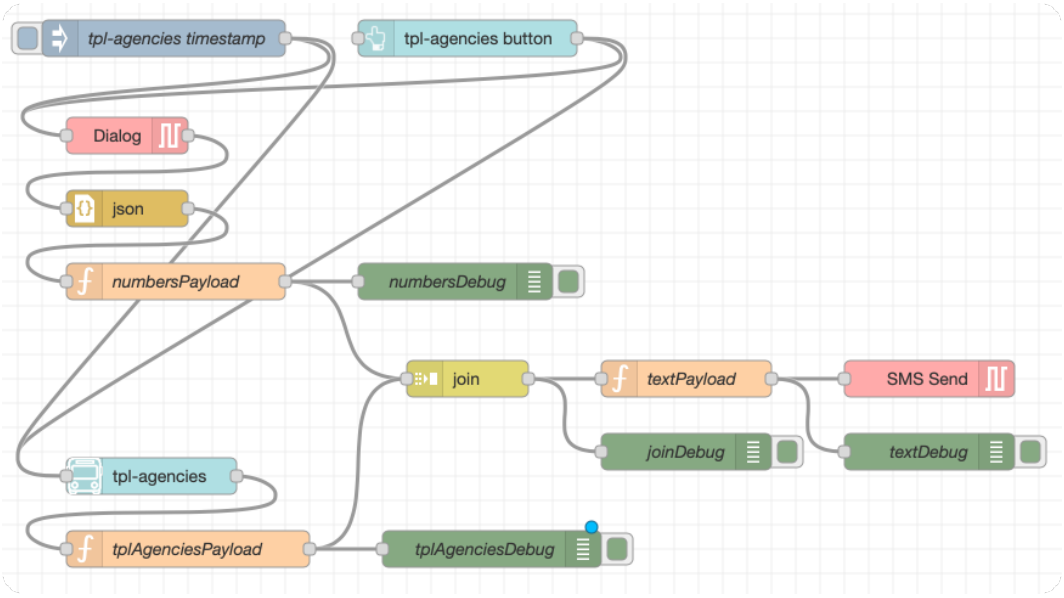
Quando il flow parte il nodo `Wifi Info` chiama l'api `termux-wifi-connectioninfo`[19] che recupera lo stato della connessione, SSID e indirizzo IP. Queste informazioni entrano nei tre nodi funzione `conncectionStatePayload`, `ssidPayload` e `ipPayload`, i quali manipolano l'informazione ricevuta e convertono l'oggetto risultante in stringa. Come nel primo flow di esempio verranno mandate le informazioni tramite il nodo `fiware orion out`.

Nodo ipPayload

```
msg.payload = {
    "name": "ip",
    "value": msg.payload.ip,
    "type": "string"
};
msg.payload = JSON.stringify(msg.payload);
return msg;
```

[19] https://wiki.termux.com/wiki/Termux-wifi-connectioninfo

4.3 Terzo flow di esempio - Agenzie di trasporto



Partito il flow, vengono eseguiti due subflows in parallelo:

1. Subflow `numbers`

Il nodo `Dialog` chiama l'api `termux-dialog`[20] che mostra una finestra dove viene chiesto all'utente di immettere un numero di telefono. Per processare questo input, il payload viene passato da un nodo `json` che lo converte da una stringa ad un oggetto. Il messaggio entra nel nodo funzione `numbersPayload` che estrae tale input e gli aggiunge come topic `numbers`.

2. Subflow `tplagencies`

Il nodo `tpl-agencies`[21] fornisce un elenco delle agenzie di trasporto pubblico disponibili su Snap4City. Il JSON contenente questa lista entra nel nodo funzione `tplAgenciesPayload` che lo converte a stringa e gli aggiunge come topic `tplagencies`.

Nodi numbersPayload e tplAgenciesPayload

```
msg.payload = msg.payload.text
msg.topic = "numbers"
return msg;
```

[20] https://wiki.termux.com/wiki/Termux-dialog

[21] https://github.com/disit/node-red-contrib-snap4city-user/blob/master/tpl-agencies.js

```
---
msg.payload = JSON.stringify(msg.payload)
msg.topic = "tplagencies"
return msg;
```

4.3.1 Join subflows

Nel nodo `join` confluiscono i due subflows in un unico `msg.payload`, indicizzato secondo i nomi dei topic, che aspetta l'arrivo di entrambi i messaggi prima di far proseguire l'esecuzione del flow (attraverso l'impostazione `Send the message after 2 message parts and every subsequent message`).
L'informazione entra nel nodo funzione `textPayload` il quale:

1. Assegna a `msg.numbers` il `msg.payload` del primo subflow che contiene l'input dell'utente.
2. Cicla su tutti gli oggetti JSON contenuti nella lista `msg.payload.tplagencies` (corrispondente al secondo subflow) e costruisce una stringa con tutti i nomi delle agenzie di trasporto che poi verrà assegnata al messaggio finale.

Infine, il nodo `SMS Send` chiama l'api `termux-sms-send`[22] che provvede a mandare un SMS al numero corrispondente a `msg.numbers` con contenuto la stringa appena costruita.

Nodo textPayload

```
msg.numbers = msg.payload.numbers
let s = "All Snap4City TPL Agencies:\n"
for (let x of (JSON.parse(msg.payload.tplagencies))) {
    s += x.name + "\n";
}
msg.payload = s
return msg;
```

[22] https://wiki.termux.com/wiki/Termux-sms-send