

# Specifiche dell'applicazione Vigifarmaco-Mobile

Emilio Gambone, Fabio Scapini, Jacopo Secondo

Version 1.0.0 1 Ottobre 2017

# Panoramica

VigiFarmaco-Mobile è la versione mobile dell'applicazione VigiFarmaco, per la gestione online di segnalazioni di reazioni avverse da farmaci e vaccini. E' disponibile online all'indirizzo <https://www.vigifarmaco.it> e utilizzabile con i più popolari browser da dispositivi mobili.

In sintesi il funzionamento è il seguente:

I segnalatori inviano la scheda di reazione avversa attraverso l'interfaccia web. I Responsabili di Farmacovigilanza visualizzano e revisionano tutte le segnalazioni, trasferendole alla rete nazionale di Farmacovigilanza.

Nella versione mobile si è pensato di alleggerire il numero di campi da compilare per i segnalatori in modo da rendere la segnalazione il più facile e veloce possibile.

Il segnalatore dovrà compilare un semplice form, con le sezioni:

- Paziente
- Reazione Avversa
- Farmaci
- Segnalatore

Le informazioni da inserire sono state diminuite rispetto alla versione Desktop, e sono solo quelle obbligatorie secondo la rete di Farmacovigilanza.

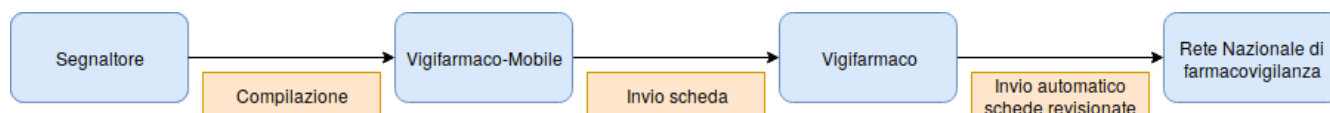


Figure 1. Schema Funzionamento applicazione

## Ambiente di Lavoro

L'ambiente di lavoro prescelto è stato EmberJS, un framework Javascript open-source basato sul pattern MVVM ( Model-View-ViewModel ).

 Guida ufficiale EmberJS : <https://guides.emberjs.com/v2.17.0/>

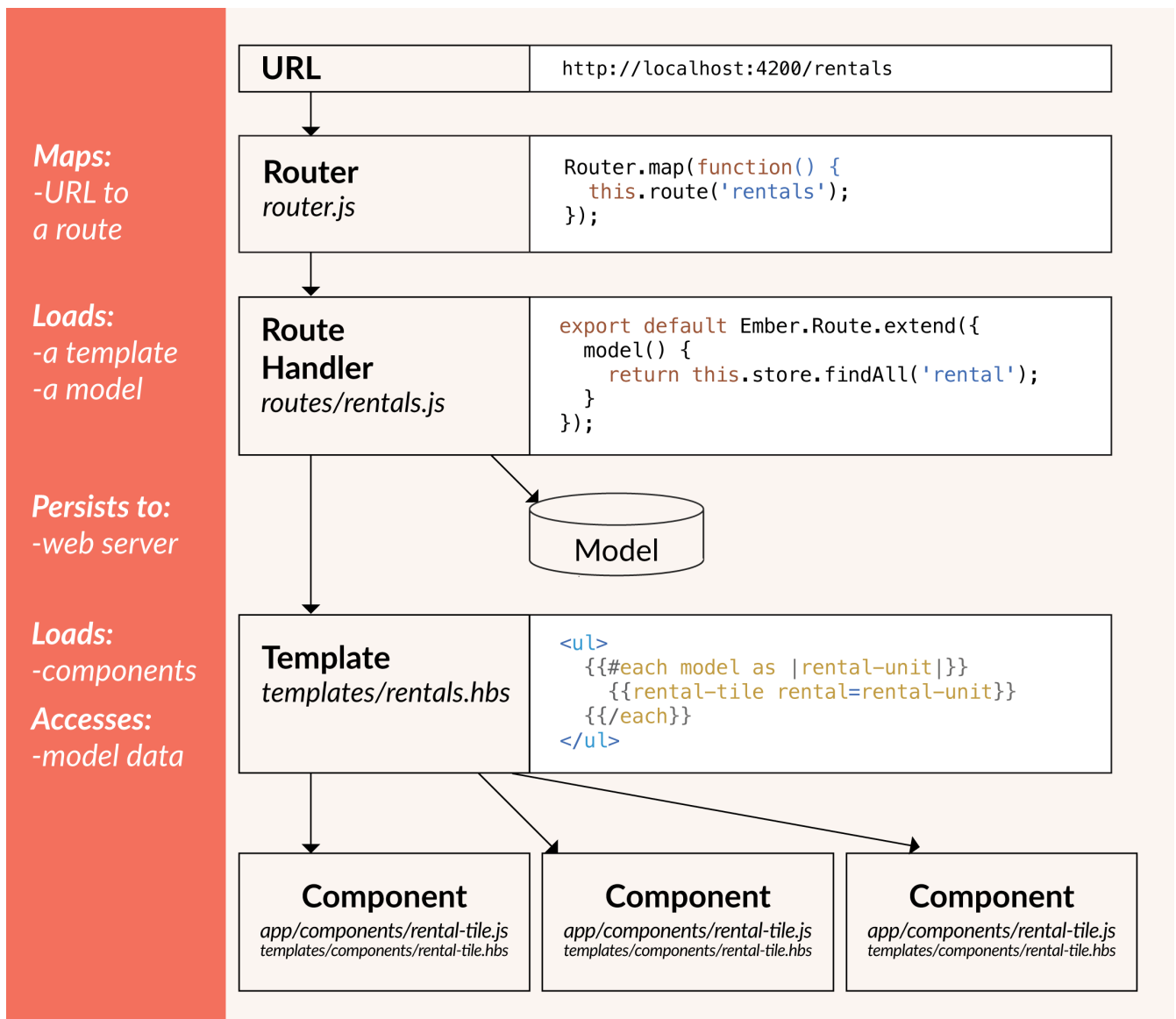


Figure 2. Schema semplificato del funzionamento di Ember

## Installazione e configurazione Ember

1. Installazione npm : **sudo apt install npm**
2. Installazione ember : **sudo npm install -g ember-cli@3.00**
3. Installazione bower ( per gestire dipendenze aggiuntive ) : **npm install -g bower**
4. Installazione dipendenze (all'interno della directory con il progetto) : **npm install**

Per lanciare il server locale, digitare **ember s** all'interno della directory con il progetto. Sarà consultabile alla poi alla pagina <http://localhost:4200>.

## Models

Sono stati definiti i seguenti modelli:

- **Report** : contenente tutti i valori scelti dall'utente, che poi saranno inviati al server.
- **Treatment** : contenente tutti i farmaci di un determinato report.

- **Outcome** : contenente tutti i possibili esiti delle reazioni avverse. I valori sono stati scaricati dal server.
- **Seriousness** : contenente tutti i criteri di gravità delle reazioni avverse. I valori sono stati scaricati dal server.
- **Region** : contenente tutte le regioni italiane. I valori sono stati scaricati dal server.
- **Structure** : contenente tutte le aziende sanitarie del territorio italiano. I valori sono stati scaricati dal server.

## Relazioni tra i modelli

Report è il modello principale, tutti gli altri modelli sono collegati a questo modello, e solo a questo modello.

- report - treatment → uno a molti
- outcome - report → uno a molti
- seriousness - report → uno a molti
- region - report → uno a molti
- structure - report → uno a molti

## Routes

In Ember, lo stato dell'applicazione è definito da un URL. Ogni URL ha il suo oggetto di tipo Route corrispondente.

Ogni route ha il proprio modello associato contenente i dati associati allo stato corrente dell'applicazione. Nel metodo **model()** del file **app/routes/report.js** vengono ritornati tutti i modelli tramite una promessa di promesse.

*route/report.js*

```
return Ember.RSVP.hash({
  reports: temp == undefined ? store.createRecord('report') : temp,
  outcomes: store.peekAll('outcome').get('length') == 0 ?
store.findAll('outcome') : store.peekAll('outcome'),
  seriousness: store.peekAll('seriousness').get('length') == 0 ?
store.findAll('seriousness') : store.peekAll('seriousness'),
  regions: store.peekAll('region').get('length') == 0 ?
store.findAll('region') : store.peekAll('region'),
  healthcare_structures: store.peekAll('structure').get('length') == 0 ?
store.findAll('structure') : store.peekAll('structure')
})
```

La promessa ritornata viene risolta solo quando tutte le promesse sono risolte.

Le URL dell'applicazione sono generate con i nomi delle route. L'URL iniziale è <http://localhost:4200/report/homepage>

# Templates

I templates sono usati per costruire l' HTML delle pagine, e sono scritti in Handlebars. Dai template è possibile chiamare azioni del controller corrispondente.

Esempio della chiamata di una action:

*template/report/reaction.hbs*

```
<form {{action "setElement" 'seriousness_criterium_id' this on="change"}}>
```

Viene chiamata l'azione **setElement(name)** definita nel rispettivo controller.

Viene passato il parametro **seriousness\_criterium\_id**, che verrà visto come stringa dal metodo.

## Component

Un componente è un tag HTML customizzato, scritto in JavaScript e definito in Handlebars. L'unico componente definito è **app/components/submitButton.js**

Il component è utilizzato per formare i bottoni in ogni pagina. Nella vista vengono passati i parametri al component.

Questo ha permesso di risparmiare le righe di codice per scrivere ogni bottone nelle viste.

## Helpers

Gli Helper sono porzioni di codice in JavaScript, che hanno lo scopo di generare, decorare e gestire la vista dei template in maniera personalizzata.

Sono stati utilizzati per riusare lo stesso codice per visualizzare i Radio-Button e le Select di HTML, ovvero passando gli opportuni parametri dalla vista, vengono create stringhe HTML rappresentante il tipo desiderato.

## Controllers

Nei Controller sono definite le azioni che avranno effetto sul Model. Ogni route ha associato un suo controller.

Le azioni vengono chiamate dalla vista passando anche eventuali paramentri.

Esempio:

```
setElement(name){
    let select = document.getElementById(name);

    if(name == 'seriousness_criterium_id' &&
select.options[select.selectedIndex].text == 'decesso'){
        this.get('model').reports.set(name, select.value);
        var outcomes = this.get('store')._identityMap._map.outcome._models;
        for (var j=0; j<outcomes.length; j++){
            if (outcomes[j]._data.name == 'decesso'){
                this.get('model').reports.set('outcome_id', outcomes[j]._data.aifa);
            }
        }
    }
    this.get('model').reports.set(name, select.value);
}
```

Nell'esempio l'azione setta un valore nel modello passato come parametro dalla vista.

## Adapters

Gli adapter sono oggetti che traducono le chiamate di rete dall'applicazione al server.


Ember rende disponibili solo adapters per servizi REST, le API di vigifarmaco sono SOAP, quindi si è dovuto customizzare sia adapter che serializer.

I dati ritornano dalle API in XML e c'è bisogno di fare parsing in JSON.

## Recupero dei dati dal server

I valori per i seguenti campi, saranno scaricati dal server tramite le API di VigiFarmaco.

- Esito (outcome) della reazione avversa
- Gravità (seriousness) della reazione avversa
- Codice identificativo della regione di appartenenza
- Codice identificativo della struttura sanitaria

 Per utilizzare le API di vigifarmaco è necessario avere un API key generabile dopo la registrazione [https://www.vigifarmaco.it/users/sign\\_up](https://www.vigifarmaco.it/users/sign_up)

## Richiesta GET

Per le richieste GET è stato necessario fare override del metodo **findaAll()** dell'adapter.

E' richiesto nell'header l'APIkey da sviluppatore.

Il metodo ritorna una promessa che in caso di risoluzione popolerà i modelli, e in caso di rifiuto segnalerà l'errore in console.

```

findAll(){

    return new Promise(function(resolve,reject){
        let xhr = new XMLHttpRequest();
        var method = 'GET';
        var url = 'https://test.vigifarmaco.it/api/v1/infos/outcomes';

        if ("withCredentials" in xhr) {
            // XHR for Chrome/Firefox/Opera/Safari.
            xhr.open(method, url, true);
        } else if (typeof XDomainRequest != "undefined") {
            // XDomainRequest for IE.
            xhr = new XDomainRequest();
            xhr.open(method, url);
        } else {
            // CORS not supported.
            xhr = null;
        }

        if (!xhr) {
            alert('CORS not supported');
            return;
        }

        // ogni volta che lo stato della connessione cambia viene chiamato il metodo
        alertContents()
        xhr.onreadystatechange = alertContents;
        xhr.setRequestHeader("Vigifarmaco-API-Key", "");
        xhr.send();

        function alertContents() {
            if (xhr.readyState == 4){
                if(xhr.status == 200){
                    resolve(this.responseXML);
                }
                else {
                    reject(new Error('getOutcomes: \'' + url + '\'' failed with status: [' +
this.status + ']' + 'failed with readyState:' + this.readyState+'));
                }
            }
        }
    })
}

```

## Richiesta POST

Per la richiesta POST è stato necessario fare override del metodo **createRecord()** del adapter. Il metodo **createRecord()** viene chiamato dal metodo **save()** nel file **controller/finalCheck.js** Sono richiesti nell'header l'APIkey da sviluppatore e il Content-type: text/xml.

Il metodo ritorna una promessa che in caso di risoluzione ritornerà un XML con il codice della segnalazione, e in caso di errore un XML con l'errore.

## Serializers

I serializer sono oggetti usati dall'applicazione per formattare i dati da ricevere e inviare al server. Nelle richieste GET i dati ricevuti in XML devono essere tradotti in JSON.

Nella richiesta POST i dati devono prima essere strutturati in un XML.

Per i valori scaricati dal server nel metodo **normalizeResponse()** del serializer viene creato un oggetto che rispetta le specifiche di JSON API e vengono inseriti i valori contenuti nelle foglie dell'XML.

 JSON API specifications : <http://jsonapi.org/>

Per i valori inviati al server nel metodo **serialize()** dell'adapter viene creato un oggetto XML manualmente e vengono inseriti i valori contenuti nello store di Ember.

## Invio di una nuova segnalazione

Per poter inviare più di una scheda durante la stessa sessione è stato necessario inserire il seguente controllo nel metodo **model()** del file **route/report.js**

*route/report.js*

```
model(){
  var store = get(this, 'store');

  var records = store.peekAll('report');

  var state;
  var temp = undefined;

  /* per ogni report in memoria controllo se le è stato eliminato (cioè già inviato
  al server)
  se tutti i report presenti sono stati eliminati ne creo uno nuovo altrimenti
  verrà usato quello non eliminato
  */
  records.forEach(function(item){
    state = records.objectAt(records.indexOf(item)).get('isDeleted');
    if (state == false){
      temp = records.objectAt(records.indexOf(item));
    }
  })
}
```

In Ember quando un record viene inviato al server, viene automaticamente impostato come eliminato. Su un record eliminato non è più possibile compiere alcuna azione. Quindi nel caso il



record risulta eliminato ne viene creato un altro.

## Validazioni

Per le validazioni dei campi dei modelli secondo gli standard dell'ente nazionale di Farmacovigilanza, è stato installato il pacchetto Ember CP Validations.

 Manuale di utilizzo : <http://offirgolan.github.io/ember-cp-validations/docs/modules/Usage.html>

Tutti i pacchetti installati e le relative versioni sono consultabili nel file **package.json**