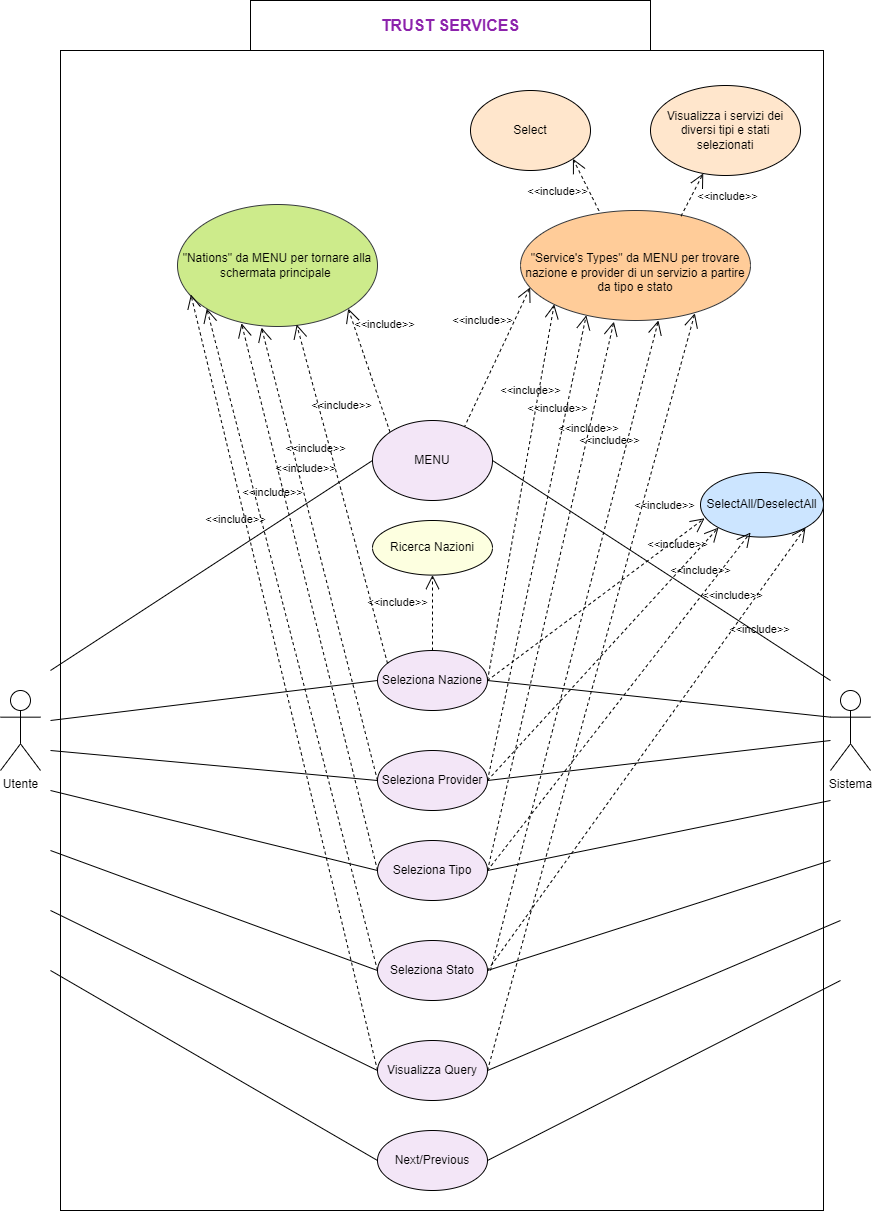
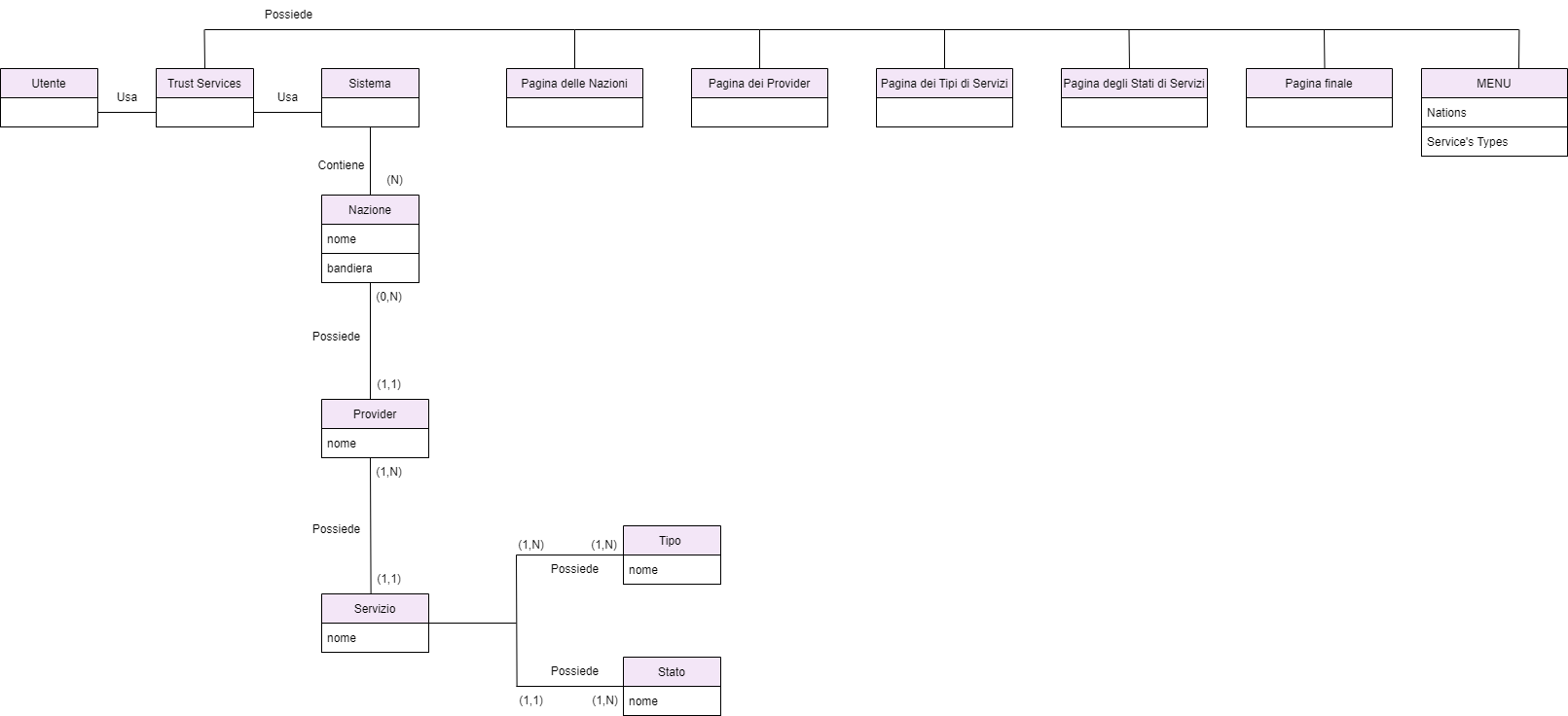
**Use cases: diagramma UML e descrizione**

****

| * **Use case**: Seleziona |
| --- |
| **Actor:** Utente, sistema  **Precondition:** La funzione è disponibile per ogni item: nazioni, provider, tipo di servizi, stato dei servizi  **Postcondition:** Gli item selezionati vengono evidenziati nel momento della selezione  **Actor action:**  **+** L’utente visualizza la lista degli item selezionabili  **+** L’utente può selezionare uno o più item  **System Responses:**  **+** Il sistema evidenzia di un altro colore gli item selezionati |
|  |
| * **Use case**: Deseleziona |
| **Actor:** Utente, sistema  **Precondition:** La funzione è disponibile per ogni item: nazioni, provider, tipo di servizi, stato dei servizi  **Postcondition:** Gli item deselezionati tornano al colore originale nel momento del click  **Actor action:**  **+** L’utente può deselezionare gli item, premendo sugli item già selezionati  **System Responses:**  **+** Il sistema fa tornare al colore originale gli item deselezionati |
|  |
| * **Use case**: Seleziona/Deseleziona Tutto |
| **Actor:** Utente, sistema  **Precondition:** La funzione è disponibile per ogni item: nazioni, provider, tipi di servizi, stato dei servizi  **Postcondition:** Vengono evidenziati tutti gli item se selezionati con “SelectAll”  **Actor action:**  **+**L’utente visualizza la lista degli item  **+** L’utente può selezionare/deselezionare tutti gli item attraverso il pulsante “Select All”  **System Responses:**  **+** Il sistema cambia colore a tutti gli item nel momento della selezione tramite “Select All”  **+** Il sistema fa tornare al colore originale tutti gli item nel momento in cui “Select All” venga utilizzato dopo aver già selezionato tutti gli item  **+** Se viene utilizzato il “Select All” dopo aver già selezionato degli item ma non tutti, vengono selezionati tutti gli item, compresi quelli già selezionati in precedenza |
|  |
| * **Use case:** Menu: “Nations” |
| **Actor:** Utente, sistema  **Precondition:** Il menu è presente in ogni schermata  **Postcondition:** L’utente tramite il menu, può tornare alla schermata iniziale, quella delle nazioni  **Actor action:**  **+** L’utente visualizza dal menu l’opzione “Nations” e può selezionarla  **System Responses:**  **+** Se l’utente utilizza l’opzione “Nations” del menu, allora il sistema riporta l’utente alla schermata iniziale |
|  |
| * **Use case**: Menu: “Service’s Types” |
| **Actor:** Utente, sistema  **Precondition:** Il menu è presente in ogni schermata  **Postcondition:** L’utente può effettuare delle ricerche combinate per tipo e stato di servizio  **Actor action:**  **+** L’utente seleziona dal menu l’opzione “Service’s Types”  **+** L’utente può selezionare uno o più tipi e uno o più stati di servizio  **System Responses:**  **+** Se l’utente seleziona un tipo o uno stato di servizio e preme su “**show service**” può visualizzare accanto la lista di servizi corrispondenti a quello stato o a quel tipo senza effettuare la ricerca combinata delle due. Se non è stato selezionato nulla e si usa comunque “**show service**” viene comunicato all’utente  **+**Se l’utente seleziona uno o piu’ tipi/stati di servizio con “**select**” vengono visualizzati a lato  **+** Se l’utente non seleziona nessun tipo e nessuno stato con il “**select**” ma esegue comunque una ricerca tramite il “**next**”, il sistema stampa a video un messaggio per comunicarlo all’utente  **+** Se l’utente seleziona solo lo stato oppure solo il tipo di servizio tramite “**select**” e preme “**next**”, il sistema stampa a video un messaggio per comunicarlo all’utente  **+** Se l’utente seleziona sia lo stato che il tipo e preme “**next**”, il sistema stampa a video una lista di servizi corrispondenti alla ricerca sia del tipo sia dello stato del servizio se esistono con quella combinazione. Se non esiste una combinazione, il sistema stampa un messaggio per comunicarlo all’utente  **+** Se l’utente ha completato tutti i passaggi correttamente fino a visualizzare la lista di servizi con quella combinazione tipo-stato selezionati, l’utente può selezionare uno o più servizi appartenete a questa lista premendoci sopra e il sistema stampa a video le nazioni e i provider corrispondenti ai servizi selezionati una volta utilizzato il pulsante “**find**”. L’utente può inoltre selezionare “**Show back**” per visualizzare il tipo e lo stato del servizio selezionato |
|  |
| * **Use case**: Next |
| **Actor:** Utente, sistema  **Precondition:** La funzione è disponibile per ogni interfaccia, eccetto per l’interfaccia finale  **Postcondition:** Tramite il pulsante next l’utente passa all’interfaccia successiva  **Actor action:**  **+** L’utente preme il pulsante next, presente in ogni pagina, tranne quella finale  **System Responses:**  **+** Il sistema stampa a video il risultato della selezione con tutti gli item corrispondenti alle selezioni fatte dall’utente fino a quel momento  **+** Se l’utente non ha selezionato nessun item, il sistema mostra un messaggio per comunicarlo all’utente |
|  |
| * **Use case**: Previous |
| **Actor:** Utente, sistema  **Precondition:** La funzione è disponibile per ogni interfaccia, eccetto per l’interfaccia iniziale  **Postcondition:** Tramite il pulsante previous l’utente torna all’interfaccia precedente con gli item che corrispondono alle precedenti selezioni effettuate  **Actor action:**  **+** L’utente preme il pulsante previous, presente in ogni pagina, tranne quella iniziale  **System Responses:**  **+** Il sistema stampa a video la pagina precedente, con tutti gli item che corrispondevano alle ricerche fatte fino a quel momento dall’utente |
|  |
| * **Use case**: Visualizza query |
| **Actor:** Utente, sistema  **Precondition:** L’utente deve aver terminato tutte le selezioni e aver premuto il pulsante “next” dell’interfaccia relativa agli stati dei servizi  **Postcondition:** Viene visualizzata nell’ultima interfaccia la query finale  **Actor action:** L’utente visualizza la query finale  **System Responses:**  **+** Il sistema stampa nell’ultima interfaccia la query finale contenente : nome nazione, provider, tipo di servizio, stato del servizio, nome del servizio, in base alle selezioni fatte dall’utente |

**\***L’UML dei casi d’uso è stato realizzato con .drawio, è possibile visualizzare il formato png all’interno della cartella “System\_Design ”-> ”Use\_cases” , con il nome di “Use\_cases.png”.

**Domain model e descrizione**

****

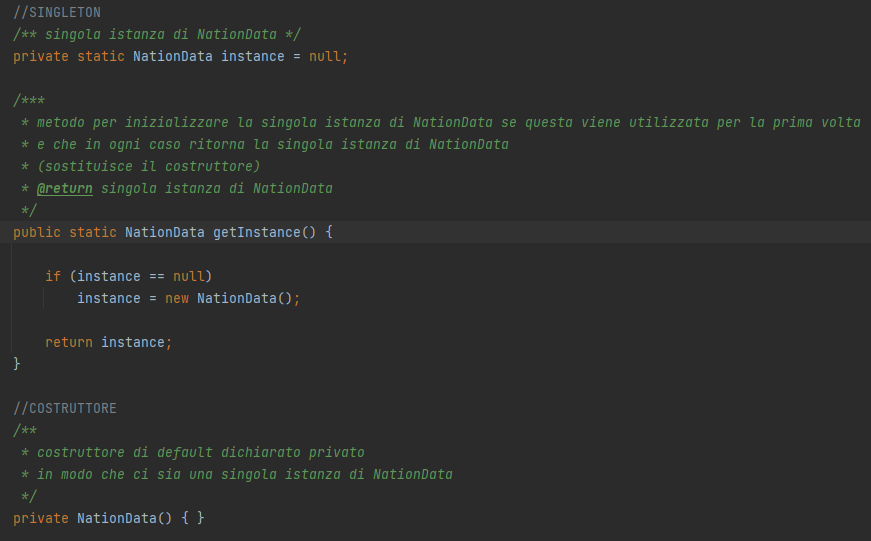
E’ possibile consultare il domain model nella cartella “System\_Design” -> “Domain\_model” in formato png denominato “Domain\_model.png”, realizzato con .drawio ed esportato.

Nel domain model, sono stati rappresentati gli elementi principali dell’applicazione con le relazioni che li collegano:

* Utente: rappresenta l’utente che utilizza l’applicazione Trust Services
* Trust Services è il nome dell’applicazione
* Trust services (l’applicazione) utilizza il sistema, il sistema è inteso come l’insieme di tutti i dati
* Il sistema contiene N nazioni, dove ogni nazione è rappresentata dal nome e dalla bandiera
* Una nazione possiede (0,N) provider, ovvero è possibile che una nazione non contenga nessun provider, e può averne massimo N, dove N è un numero naturale qualsiasi.
* I provider vengono descritti dal nome. Un provider può appartenere ad una nazione soltanto, e nel sistema non ci sono provider che non appartengono a nessuna nazione, questo è il motivo per cui la cardinalità è (1,1).
* Un provider può possedere (1,N) servizi, dove 1 significa che nel sistema sono presenti soltanto provider che hanno almeno 1 servizio, mentre possono esserci provider che possiedono fino ad N servizi, dove N è un numero naturale qualsiasi.
* I servizi sono rappresentati dal nome. Un servizio specifico può appartenere ad un provider soltanto, e non esistono nel sistema servizi che non appartengono a nessun provider (1,1).
* Un servizio può possedere (1,N) tipi, ovvero minimo 1 e massimo N. Non esistono servizi nel sistema che non appartengono a nessun provider.
* Un tipo viene descritto dal nome, e vale anche per esso la relazione (1,N) con servizio, ovvero un tipo appartiene minimo ad 1 servizio e massimo ad N. Non esiste nessun servizio che non appartenga a nessun provider.
* Un servizio può inoltre possedere (1,1) stati, minimo 1, massimo 1. Non esiste nel sistema un servizio senza nessuno stato, ed un servizio può avere più stati.
* Lo stato è descritto dal nome. Uno stato può appartenere a (1,N) servizi, minimo 1 servizio, massimo N, con N numero naturale. Non esistono nel sistema stati senza nessun servizio.
* Trust Services (l’applicazione), possiede le seguenti pagine:
  + Pagina delle nazioni: è la prima schermata dove è possibile vedere e selezionare le nazioni
  + Pagina dei provider: è la seconda pagina, dove vengono visualizzati i provider corrispondenti alle nazioni selezionate
  + Pagina dei tipi di servizi: è la terza pagina, dove vengono visualizzati i tipi di servizi relativi ai provider selezionati per ogni nazione selezionata
  + Pagina degli stati di servizi: è la quarta pagina, dove vengono visualizzati gli stati dei tipi di servizi selezionati, relativi ad ogni provider selezionato, per ogni nazione selezionata
  + Pagina finale: è la quinta pagina, dove viene stampata la query finale
  + MENU: è il menu a lato che permette ad ogni pagina di tornare alla schermata delle nazioni e che permette di effettuare ricerche combinate in base al tipo e allo stato di un servizio

**Pattern utilizzati**

**Singleton:** rientra tra i pattern di tipo GOF, è stato utilizzato per avere un’unica istanza di alcune classi, come ad esempio per NationData. NationData è una classe che contiene tutti i dati delle nazioni, come ad esempio la lista con tutte le nazioni, la lista delle nazioni selezionate ecc. Dunque è necessaria soltanto una istanza di questo tipo per poter accedere ai dati in modo coerente e senza ambiguità. Singleton permette di avere un unico accesso globale all’istanza, tramite getInstance(). Mentre per la classe Nation, non è stato implementato il pattern singleton, perchè Nation contiene soltanto i dati di una nazione, dunque si hanno più istanze della classe Nation, in particolare tante quante il numero di nazioni.



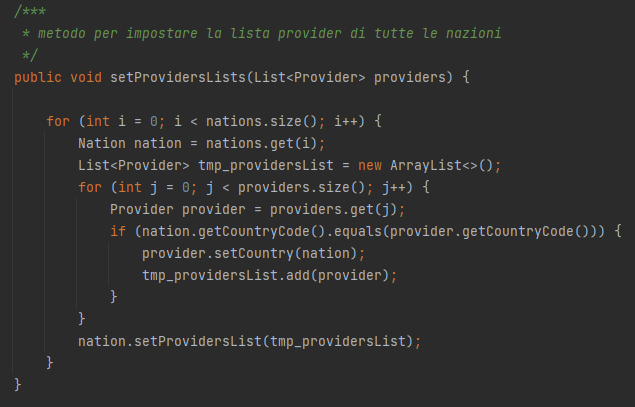
**Simple Factory:** è un pattern creazionale che rientra tra i pattern di tipo GOF. Esso ha lo scopo di creare degli oggetti senza specificarne la classe esatta. Nell’applicazione è stato utilizzato per il collegamento API, implementando la classe generica GetFromAPI, la quale permette la creazione di una lista di oggetti generici che vengono presi dal file json:



Il tipo di classe viene poi definito in DataInitializer, ad esempio per le nazioni:

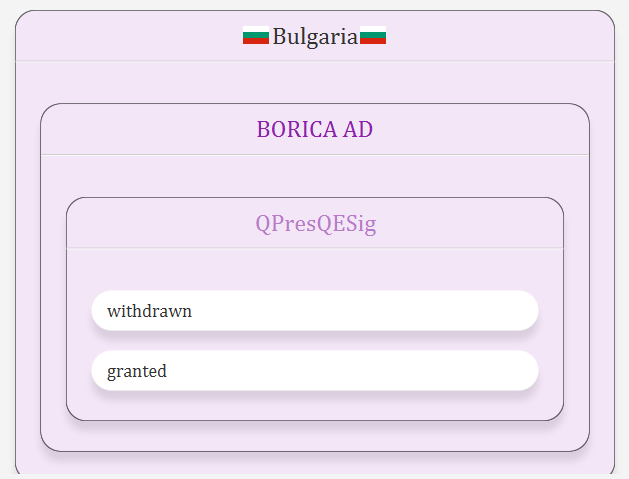


**Adaptee:** fa sempre parte dei pattern di tipo GOF, viene utilizzato quando interfacce diverse devono comunicare tra di loro. E’ stato utilizzato nel codice nella classe NationData con la funzione “setProvidersLists” che raggruppa i diversi provider per ogni nazione, la funzione è poi richiamata in DataInitializer:

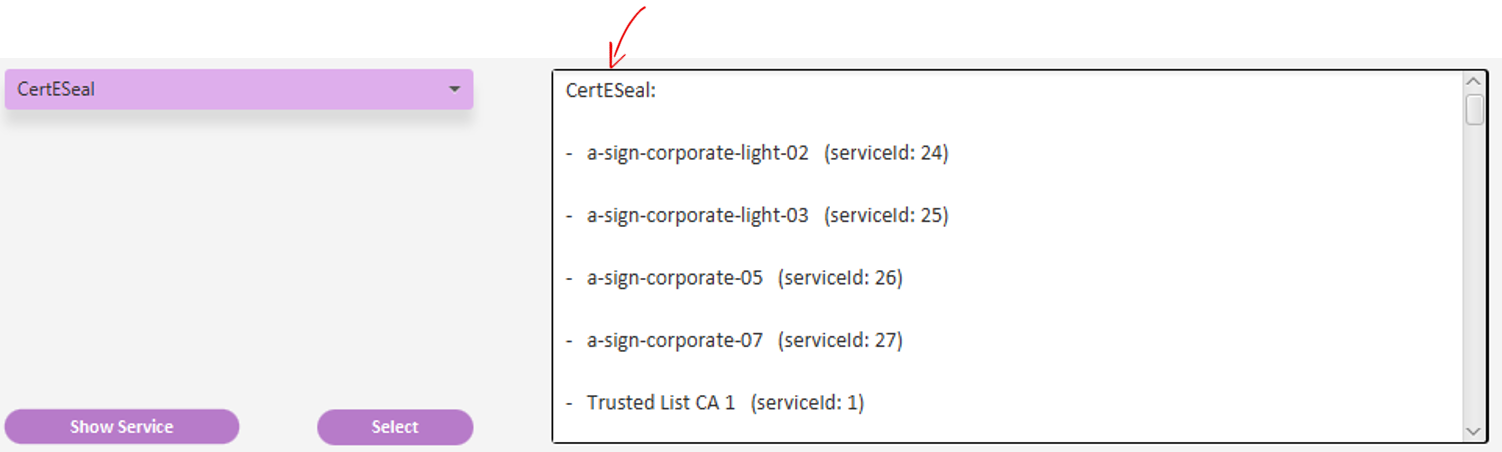




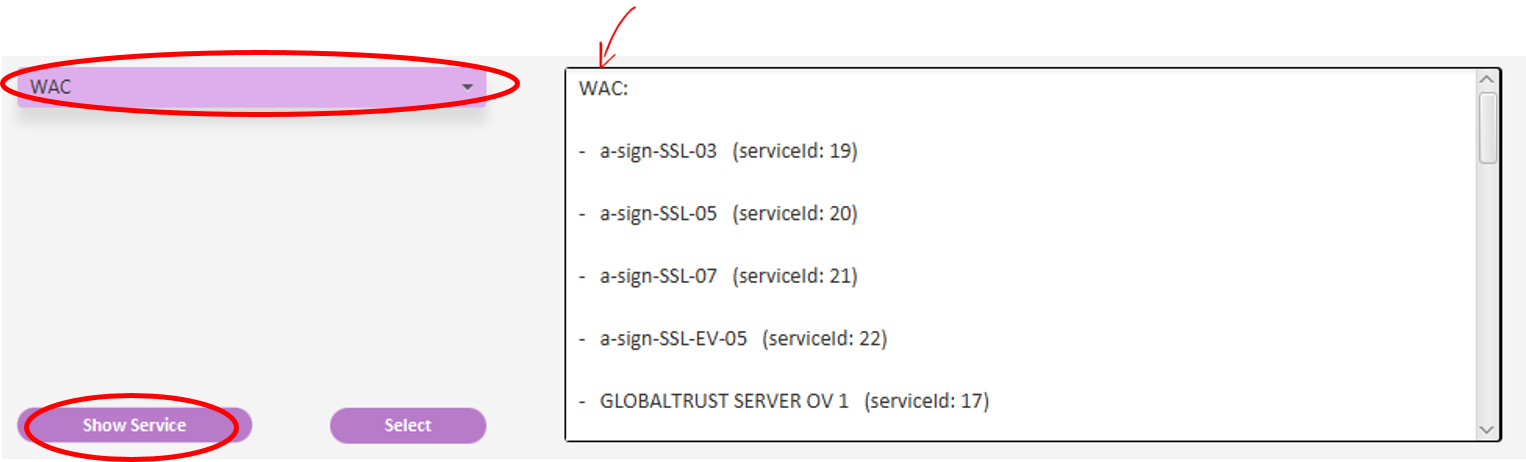
**Composite:** Composite pattern è un pattern della categoria GOF, che permette di riunire diversi oggetti in strutture più complesse. E’ stato utilizzato nell’applicazione per inserire all’interno di ogni interfaccia, i dati raggruppati all’interno di box controller annidati, come ad esempio:



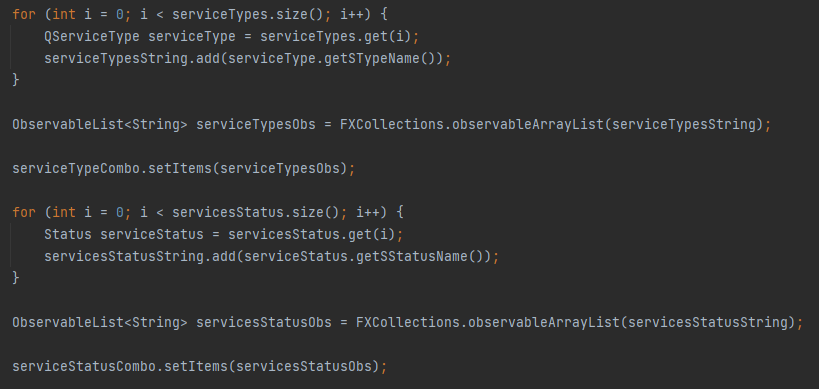
**Observable:** Il pattern Observable permette di aggiornare i dati nell’interfaccia grafica nel momento in cui essi cambiano il loro valore. E’ stato utilizzato nella parte del menu per quanto riguarda la sezione di ricerca tramite tipo e stato di un servizio:



Nel momento in cui si seleziona un altro tipo di servizio e si preme “**Show Service**”, di lato vengono aggiornati i servizi con quelli relativi al nuovo tipo selezionato



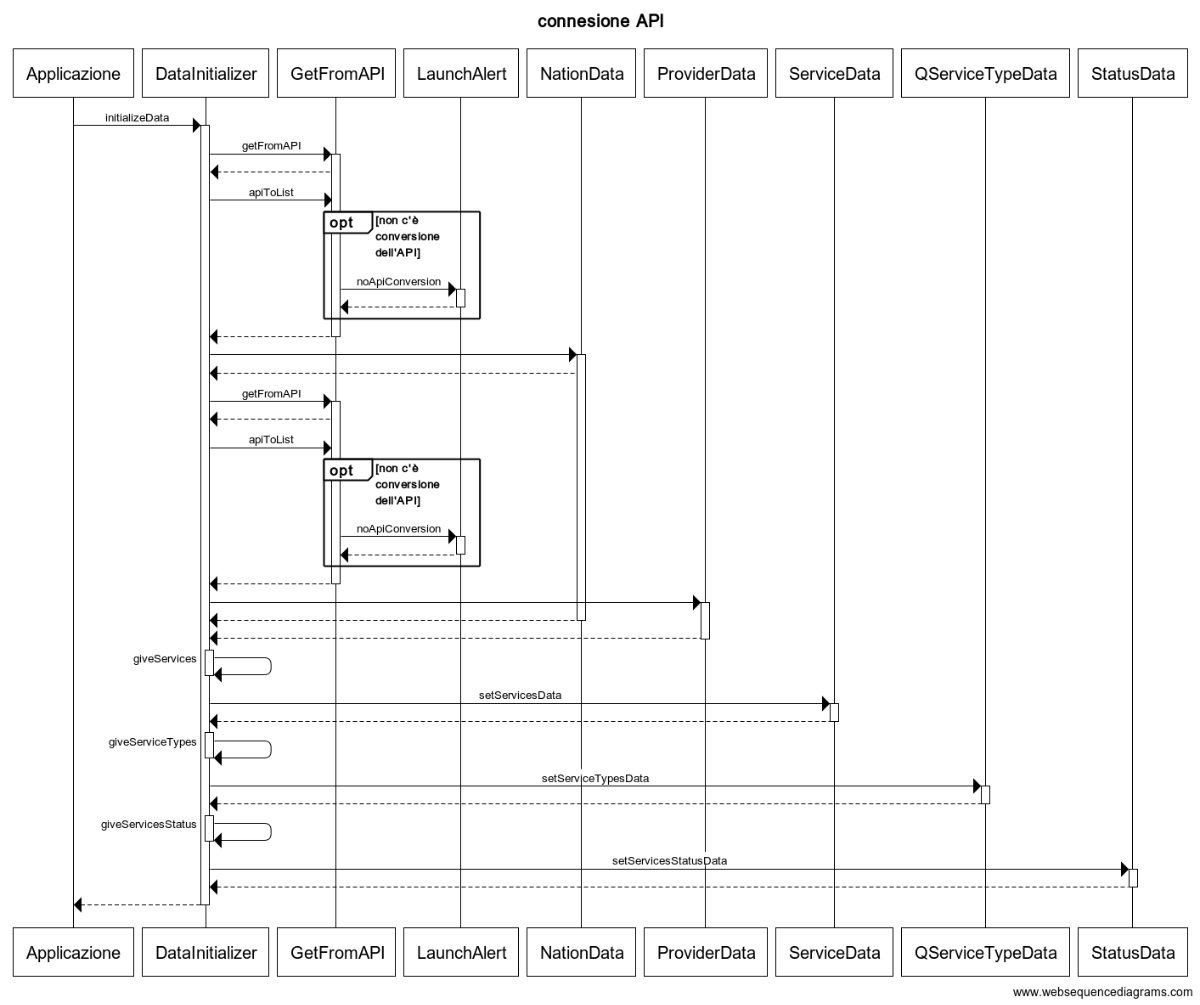
Per fare questo è stata utilizzata una ObservableList. Lo stesso procedimento è stato applicato anche allo stato dei servizi.



**Sequence Diagrams**

I sequence diagrams sono disponibili all’interno della cartella “System\_design” -> “Sequence\_diagrams”, sia in formato png, sia in formato di testo. Sono stati realizzati utilizzando [www.websequencediagrams.com](http://www.websequencediagrams.com) .

**Connessione API:**

****

L’applicazione richiama DataInitializer, la classe che si occupa del collegamento API e dell’inizializzazione dei dati. DataInitializer richiama la classe GetFromAPI per la conversione dei dati dall’API.

La **prima chiamata** a GetFromAPI viene fatta per risalire ai dati relativi alle nazioni:

**+**Se la conversione con l’API non viene effettuata, si lancia un errore noApiConversion attraverso LaunchAlert.

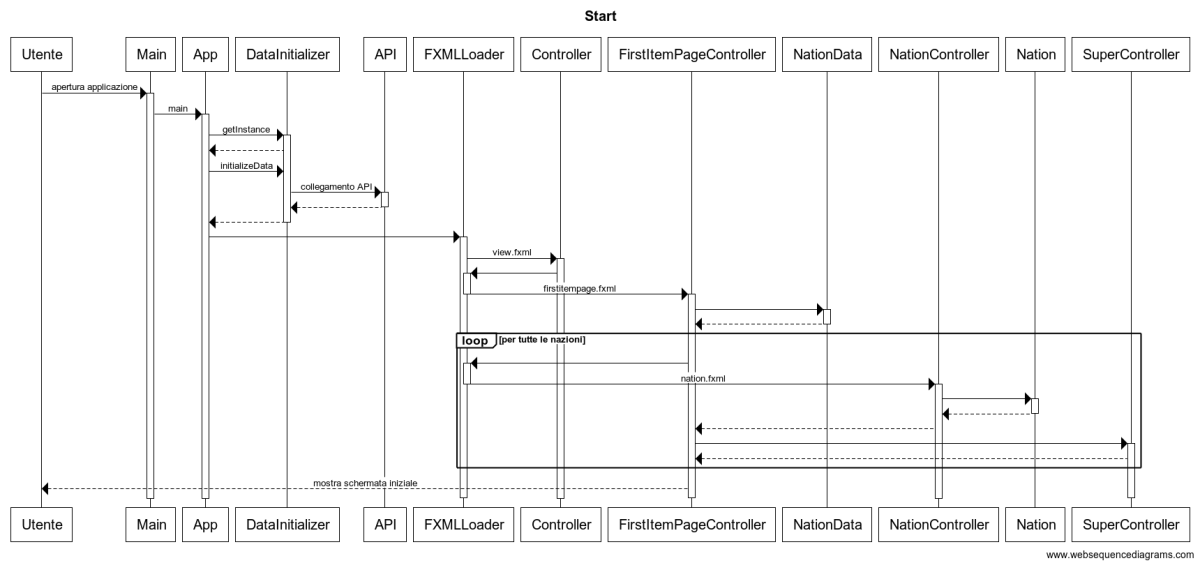
**+**Se è avvenuta la conversione, DataInitializer si occupa di inizializzare i dati in NationData.

La **seconda chiamata** a GetFromAPI viene fatta per accedere ai dati dei provider e si procede come fatto per le nazioni.

Successivamente, il DataInitializer, utilizzerà delle funzioni interne per inizializzare i dati in ServiceData, QServiceTypeData e StatusData.

Finito di inizializzare i dati, DataInitilizer può tornare all’applicazione.

**Start:**

****

Il Main richiama la classe App quando l’applicazione viene aperta.

La classe App richiama DataInitializer creando una singola istanza con il metodo getInstance() [*Singleton*].

DataInitializer si occupa di inizializzare i dati, facendo il collegamento con l’API [*diagramma precedente*]. Una volta inizializzati i dati, la classe App carica le componenti grafiche tramite FXMLLoader.

La prima componente grafica che viene caricata è **view.fxml** con il corrispettivo controller, chiamato Controller. Il view.fxml è la “base” grafica che conterrà successivamente tutte le altre.

Controller tramite FXMLLoader carica una nuova componente grafica: **firstitempage.fxml** con il controller FirstItemPageController, essa rappresenta lo spazio della pagina che è destinato a contenere le nazioni.

FirstItemPageController richiama la classe NationData per ottenere i dati relativi a tutte le nazioni.

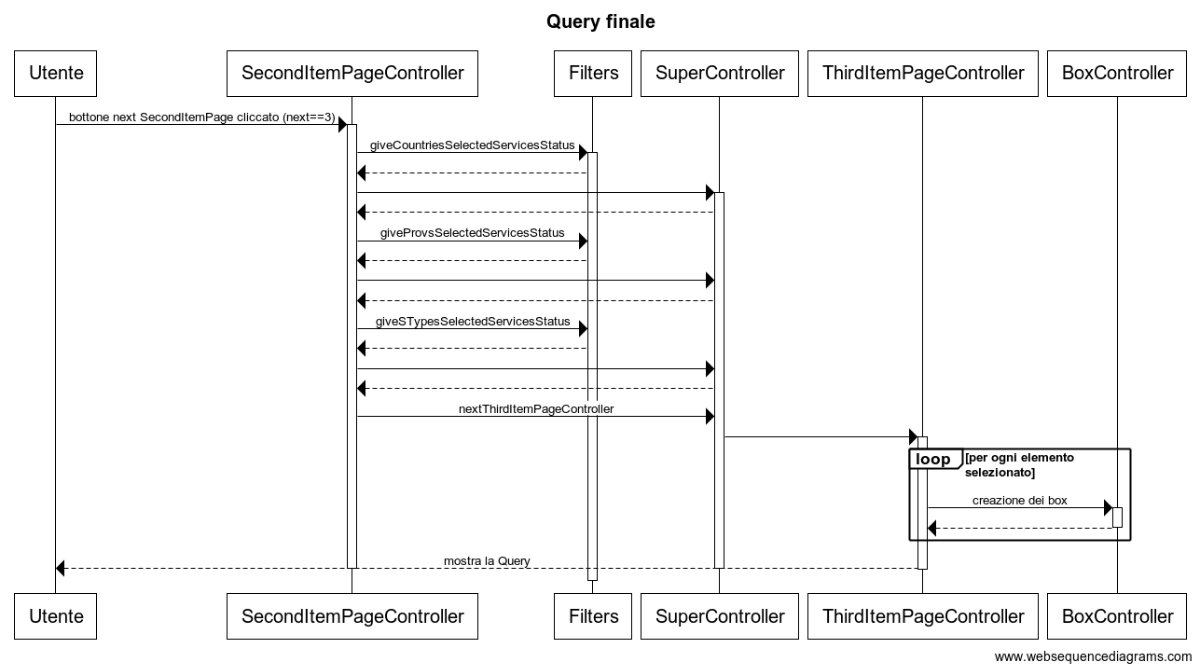
Successivamente, si entra in un loop, nel quale vengono stampate tutte le nazioni nei corrispettivi box:

FirstItemPageController tramite FXMLLoader carica **nation.fxml** con il proprio controller chiamato NationController, il quale richiama l’oggetto Nation contenente i dati della singola nazione.

Viene utilizzata anche la classe SuperController che è la superclasse di tutti i controller e permette a tutte le interfacce di accedere alle liste di item, una volta create.

Per il caricamento delle interfacce grafiche, il procedimento rimane lo stesso anche per le altre pagine.

**Query finale:** (nazioni -> provider -> tipi servizi -> stato servizi -> query)

****

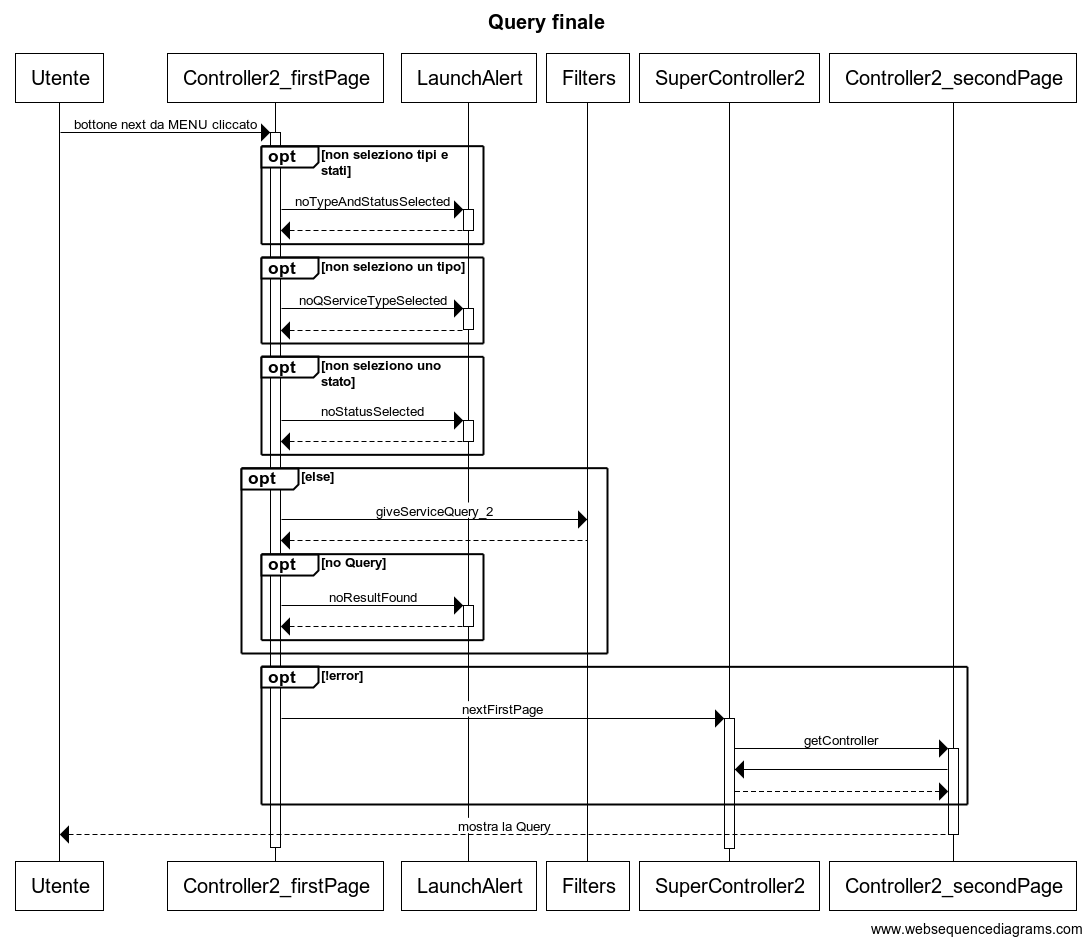
**+** Dopo aver premuto next sull’ultima interfaccia (next==3), avviene una chiamata a **SecondItemPageController**.

SecondItemPageController è il controller dell’interfaccia e richiama la classe Filters attraverso le funzioni give che restituiscono la lista di nazioni, provider e tipi di servizio per ogni stato di servizio selezionato. Dopo ogni chiamata give() i dati vengono aggiornati anche in SuperController.

Successivamente, SecondItemPageController tramite SuperController, richiama ThirdItemPageController, cioè il controller relativo all’ultima interfaccia.

All’interno di **ThirdItemPageController** avviene un loop, nel quale viene creato un box per ogni item.

A questo punto è possibile mostrare la query finale all’utente.

**Query finale:** (da menu QService’s Type)

**+** L’utente dopo aver premuto next dal menu si trova in **Controller2\_firstPage**, relativo alla prima pagina accessibile dal menu.

**+**Se non vengono selezionati tipi di servizio e/o stati di servizio viene lanciato un errore con LaunchAlert.

**+**Se invece è stato selezionato tutto correttamente, viene richiamata dal Controller2\_firstPage, la classe Filters tramite un give() che restituisce i dati risultanti dalla selezione.

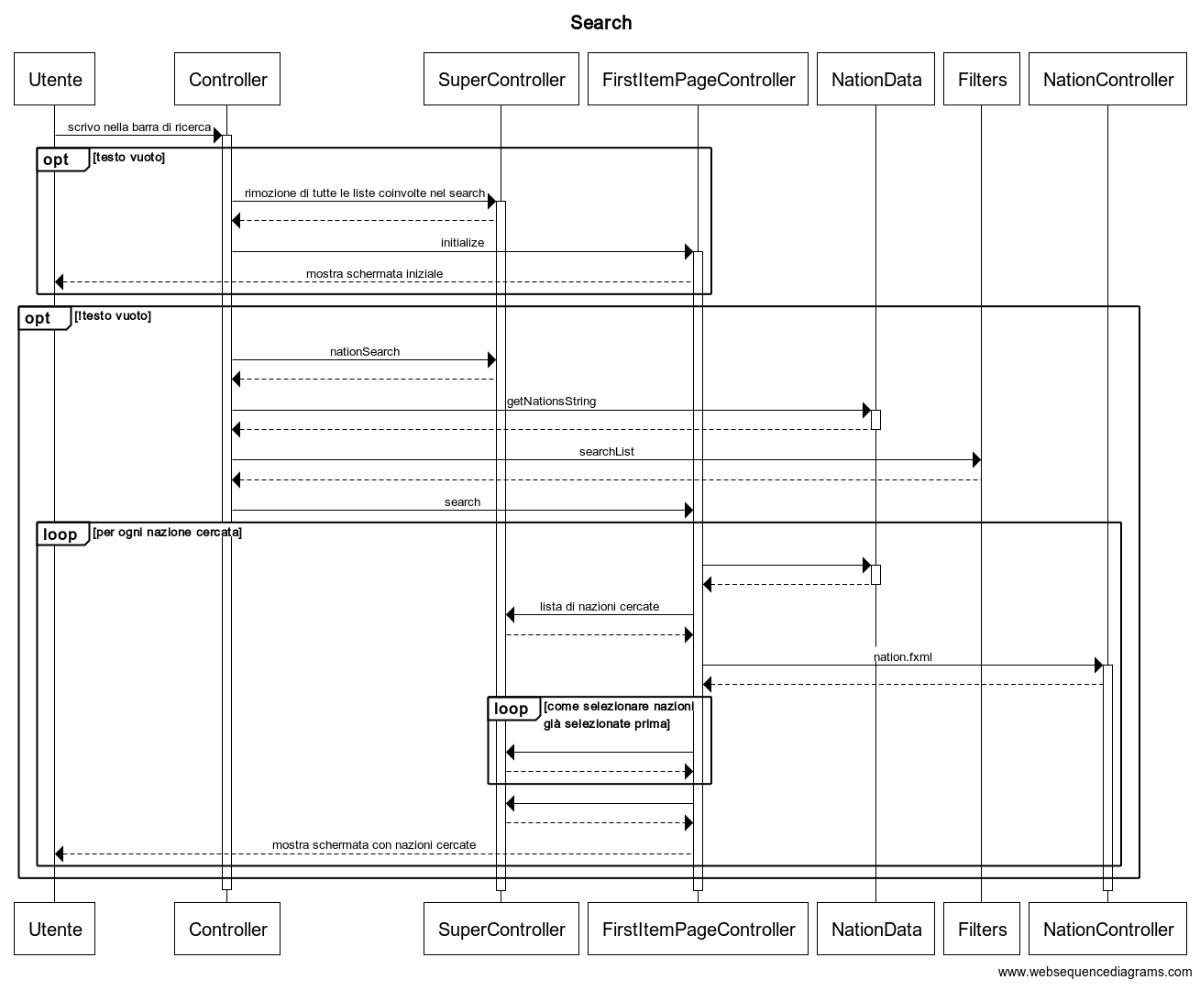
**+**Se non c’è nessun risultato corrispondente, viene lanciato un errore sempre tramite LaunchAlert.

**+**Se non c’è stato nessun errore, si procede alla pagina successiva, viene richiamato **SuperController2** (la superclasse dei controller relativi al menu). Supercontroller2 chiama **Controller2\_secondPage**, il controller dell’ultima pagina accessibile da menu, con i risultati delle selezioni.

Successivamente la query viene mostrata all’utente.

[Per il funzionamento in dettaglio del next, vedere *diagramma next pagina 13*]

**Search:**

****

L’utente scrivendo sulla barra di ricerca, richiama il Controller.

**+**Se il testo digitato è vuoto, Controller rimuove tutte le liste coinvolte nel search tramite SuperController, poi inizializza FirstItemPageController che mostra la *schermata iniziale* con tutte le nazioni.

**+**Se il testo non è vuoto, Controller richiama SuperController che restituisce una lista con i nomi che corrispondono alla ricerca. Successivamente, Controller richiama NationData per ottenere una lista con tutti i nomi di tutte le nazioni. Controller richiama Filters tramite la funzione SearchList che fornisce una lista di stringhe come risultato della ricerca.

Controller chiama FirstItemPageController.

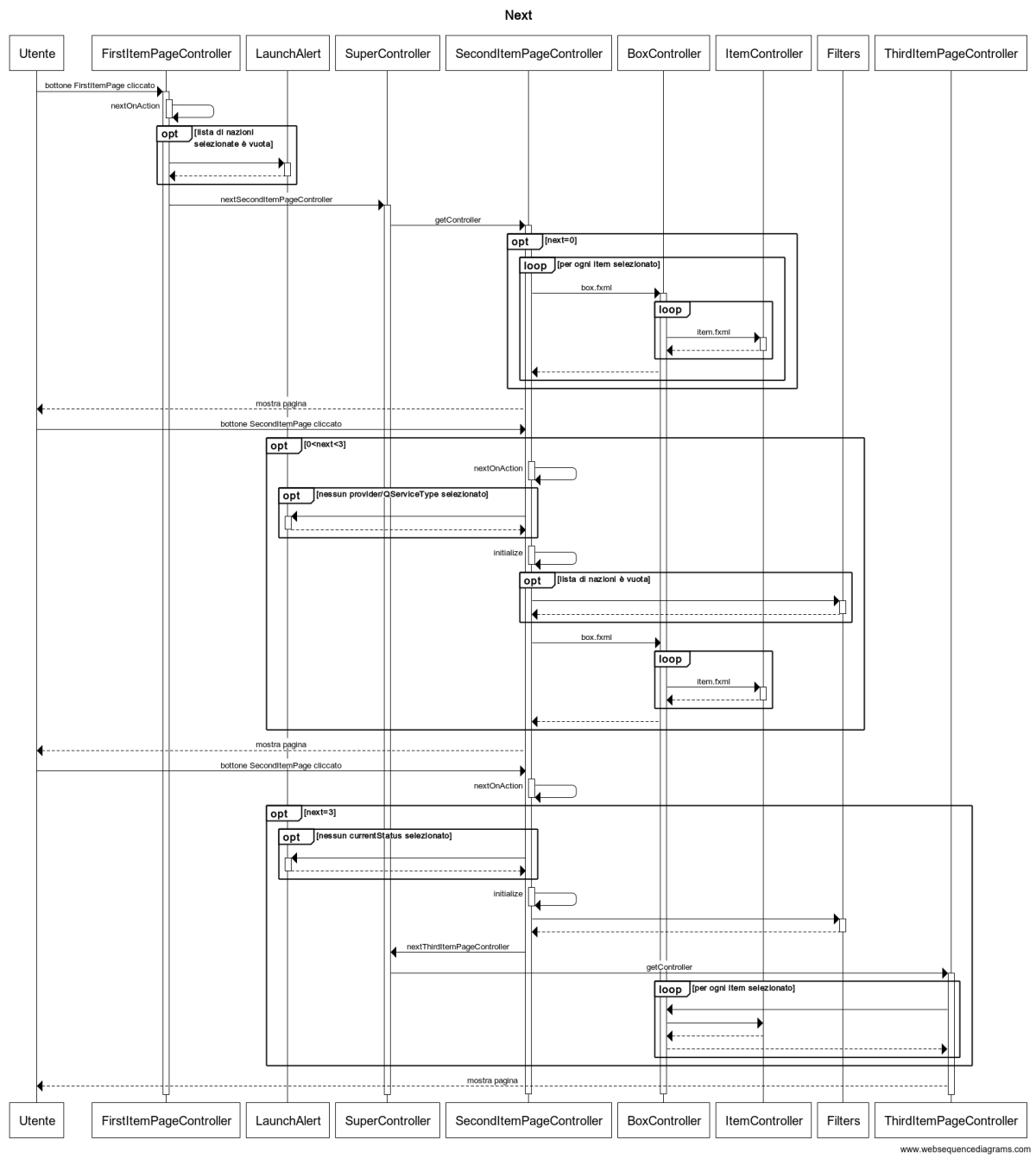
A questo punto si entra in un loop, FirstItemPageController richiama NationData contenente tutte le liste di nazioni. FirstItemPageController aggiorna in SuperController la lista di nazioni cercate.

FirstItemPageController crea l’elemento grafico nation.fxml con NationController per ogni nazione selezionata [*Maggiori dettagli riguardo la creazione di elementi grafici fxml sono sul diagramma start a pagina 9*].

**+**Si effettua un loop interno al primo in cui vengono selezionate le nazioni che erano selezionate prima della ricerca.

Infine si mostra la schermata con le nazioni cercate.

**Next:** (nazioni -> provider -> tipi servizi -> stato servizi -> query)

****

L’utente si trova nella prima **pagina FirstItemPageController** (next = -1), una volta premuto il pulsante next, la variabile next si incrementa di uno. Se non sono state selezionate nazioni, viene inviato un messaggio d’errore tramite LaunchAlert. Altrimenti viene creato il **SecondItemPageController** attraverso una chiamata al SuperController.

**+**Se **next == 0** si entra in un loop, il SecondItemPageController crea un BoxController (uno per ogni nazione selezionata), all’interno del BoxController avviene un ulteriore loop in cui vengono caricati gli elementi grafici item.fxml con i rispettivi ItemController (per ogni provider relativo ad ogni nazione selezionata).

Ora l’utente si trova in SecondItemPageController, e preme il bottone next.

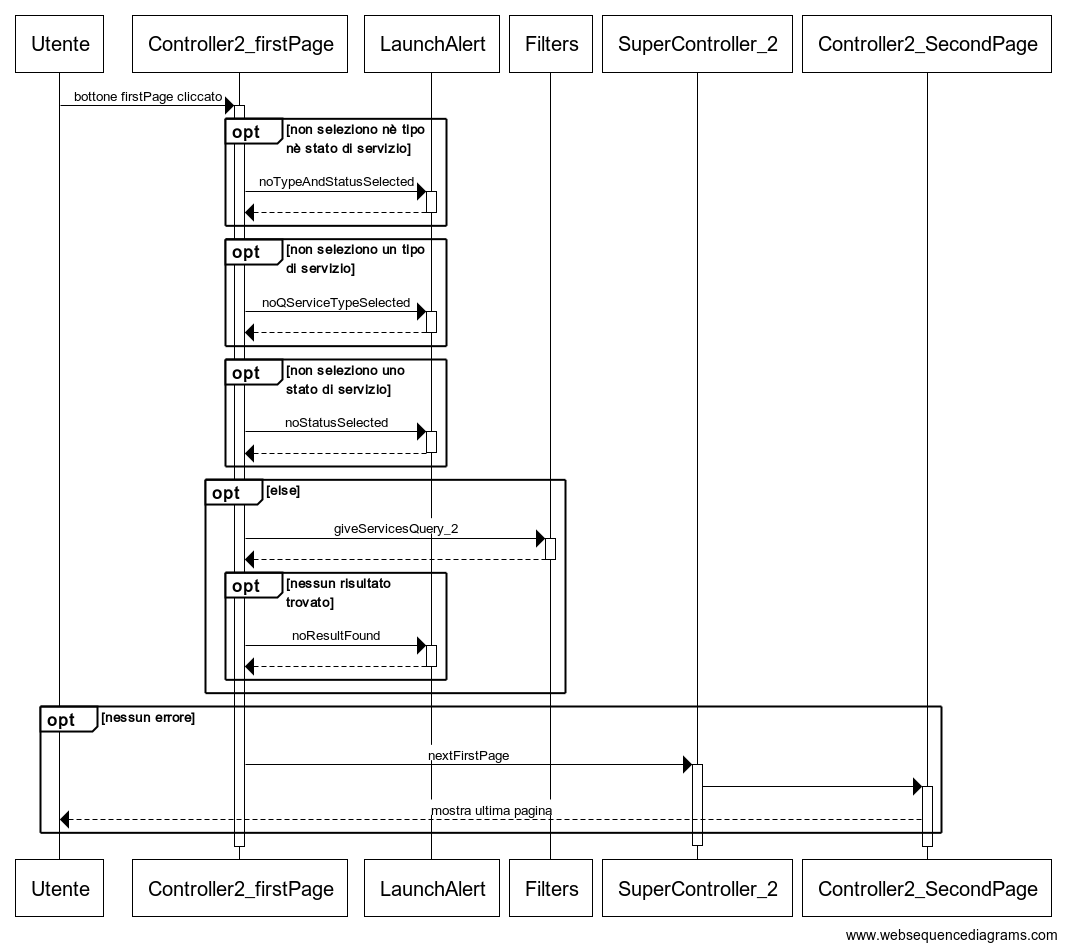
**+**Se **0 < next < 3**, il SecondItemPageController utilizza la funzione interna nextOnAction e richiama se stesso. Se non c’è nessun provider/nessun tipo di servizio selezionato, viene lanciato un errore tramite LaunchAlert. Se la lista delle nazioni è vuota, viene richiamato Filters. A questo punto, SecondItemPageController carica un box.fxml con il proprio BoxController e all’interno di BoxController si entra in un loop nel quale vengono caricati gli item.fxml con i propri ItemController. E’ possibile mostrare la pagina all’utente. Il procedimento è simile a quello precedente, soltanto che ci sono più box uno all’interno dell’altro.

**+**Se **next == 3**, se non è stato selezionato nessuno stato di servizio viene mandato un messaggio tramite LaunchAlert. Altrimenti SecondItemPageController, richiama se stesso con initialize(), viene chiamato Filters e vengono aggiornati i dati relativi alle selezioni. Successivamente viene creato ThirdItemPageController tramite il SuperController. All’interno di ThirdItemPageController, si segue lo stesso procedimento dei precedenti, in un loop vengono creati dei BoxController, e all’interno dei box vengono inseriti gli item.

Ogni volta che l’utente preme il pulsante next, la variabile next viene incrementata di 1.

[*Maggiori dettagli riguardo la creazione di elementi grafici fxml sono sul diagramma start a pagina 9*].

**Next:** (da menu QService’s Type)



Se l’utente si trova nella pagina del menu e preme il bottone next, viene chiamato **Controller2\_firstPage**:

**+**Se non è selezionato nessuno stato e/o nessun tipo di servizio, viene segnalato tramite LaunchAlert.

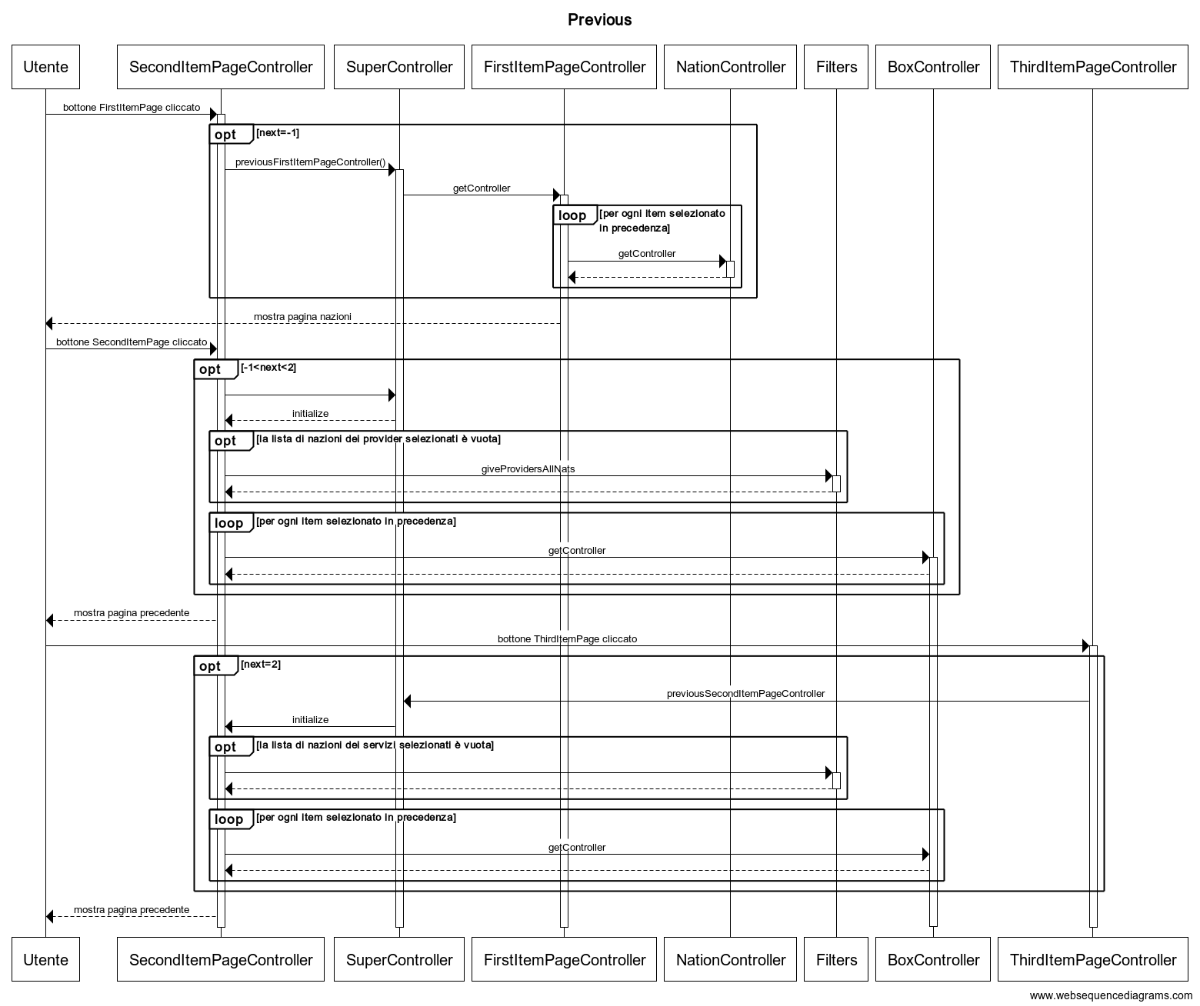
**+**Se è stato selezionato almeno uno stato e almeno un tipo di servizio, Controller2\_firstPage chiama Filters attraverso una funzione give per ottenere i dati della query.

**+**Se non è stato trovato nessun risultato dalla funzione give, Controller2\_firstPage lancia un messaggio tramite LaunchAlert.

**+**Altrimenti, se non c’è stato nessun errore ed è stato trovato almeno un risultato, viene creato **Controller2\_secondPage** tramite il SuperController\_2. E viene mostrata la query all’utente.

Ogni volta che l’utente preme il pulsante next, la variabile next viene incrementata di 1.

**Previous:** (nazioni -> provider -> tipi servizi -> stato servizi -> query)

****

L’utente preme il bottone previous, next viene decrementato:

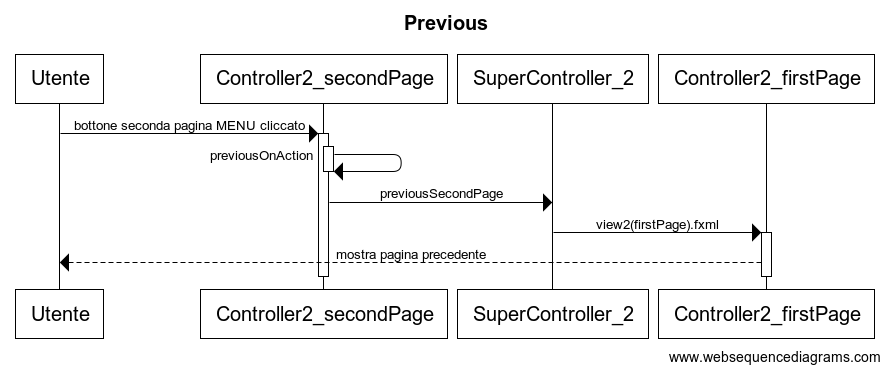
**+**Se **next == -1**, significa che l’utente è nella seconda pagina e si vuole tornare alla pagina iniziale. SecondItemPageController tramite SuperController richiama **FirstItemPageController**, nel quale avviene un loop e per ogni item viene richiamato NationController. In questo modo FirstItemPageController renderà disponibile la pagina iniziale delle nazioni all’utente.

**+**Se **-1 < next < 2,** SecondItemPageController, richiama SuperController che inizializza nuovamente **SecondItemPageController**. Se la lista di provider selezionati è vuota, viene richiamata Filters tramite un give, per ottenere la lista di provider per nazioni selezionate. Successivamente si entra in un loop nel quale viene creato un BoxController per ogni item selezionato in precedenza. Così viene visualizzata la pagina precedente tramite SecondItemPageController.

**+**Se **next == 2**, significa che l’utente ha premuto previous nell’ultima pagina in **ThirdItemPageController**. ThirdItemPageController richiama **SecondItemPageController** tramite SuperController. Se la lista di nazioni dei servizi selezionati è vuota, viene richiamato Filters. Successivamente si utilizza un loop, dove per ogni item selezionato in precedenza viene chiamato un BoxController.

A questo punto si mostra la pagina all’utente.

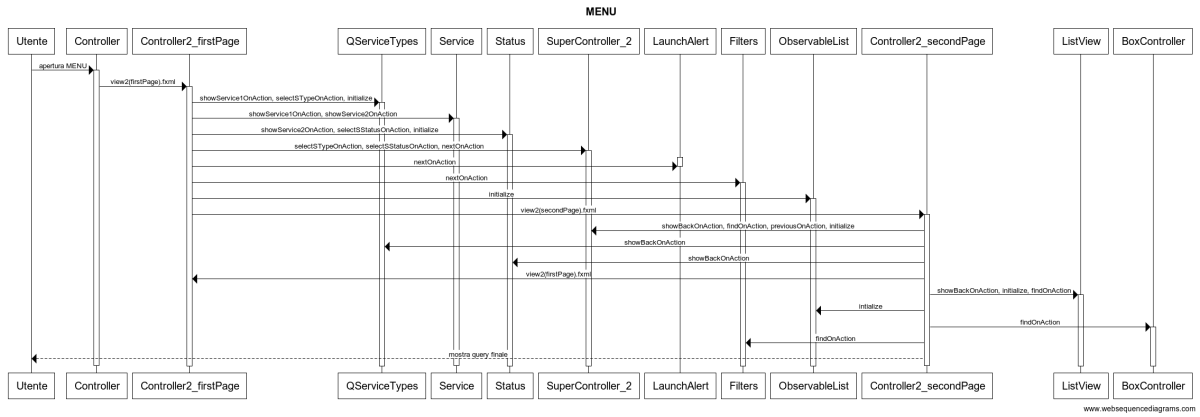
**Previous:** (da menu QService’s Type)



L’utente preme il bottone previous dall’ultima schermata del menu: **Controller2\_secondPage** richiama se stessa con previousOnAction. Successivamente, richiama SuperController\_2 che carica **view2(firstPage).fxml** con il relativo Controller2\_firstPage, e così viene mostrata all’utente la pagina precedente.

[*Maggiori dettagli riguardo la creazione di elementi grafici fxml sono sul diagramma start a pagina 9*].

**Menu:**

****

L’utente apre il menu, viene caricata la pagina **view2(firstPage).fxml** con il relativo **Controller2\_firstPage**. Tramite i metodi Show onAction e select onAction il controller gestisce la selezione di QServiceType, e Status, e permette di visualizzare il risultato della selezione effettuata.

Tutti gli item selezionati (QServiceType e Status) vengono inseriti all’interno di alcune liste nel SuperController\_2, per permettere alle interfacce di accedervi.

Quando l’utente seleziona next viene lanciato un errore tramite LaunchAlert se non è stato selezionato nulla. Quando si hanno degli elementi selezionati, Controller2\_firstPage richiama Filters che si occupa delle ricerche.

Controller\_firstPage, inizializza una lista di tipo *Observable* per visualizzare il tipo e lo stato selezionati.

Dopo il next, viene creata la seconda **pagina view2(secondPage).fxml** con il controller **Controller2\_secondPage** che comunica con SuperController\_2 tramite i metodi shoBackOnAction, finOnAction, previousOnAcion e initialize.

Controller2\_secondPage richiama QServiceTypes tramite showBackOnAction e richiama anche la ListView e l’ObservableList per aggiornarne i dati. Tramite findOnAction, Controller2\_secondPage richiama BoxController e Filters per ottenere il risultato finale, e poi viene mostrato all’utente.

[*Per il funzionamento in dettaglio del next, vedere* *diagramma next pagina 13-14*]

[*Maggiori dettagli riguardo la creazione di elementi grafici fxml sono sul diagramma start a pagina 9*].

**Design Model**

Il design model rappresenta le classi e le interfacce dell’applicazione e in che modo sono collegate tra di loro. E’ stato realizzato utilizzando [www.plantuml.com](http://www.plantuml.com) . E’ possibile visualizzare il design model all’interno della cartella “System\_design” -> “Design\_model” in formato png e in formato di testo.