



SIRIUS

---

SEQUENZIATORE

**Definizione di prodotto**

**Versione 1.0.0**

*Ingegneria Del Software AA 2013-2014*

## Informazioni documento

---

Titolo documento:	Definizione Di Prodotto
Data creazione:	2014-04-18
Versione attuale:	1.0.0
Utilizzo:	Esterno
Nome file:	<i>DefinizioneDiProdotto_v1.0.0.pdf</i>
Redazione:	Quaglio Davide
Approvazione:	Santangelo Davide
Distribuito da:	Sirius
Destinato a:	Prof. Vardanega Tullio Prof. Cardin Riccardo Zucchetti S.p.A.

## Sommario

Tale documento andrà a trattare in modo approfondito le componenti e la struttura del prodotto il *Sequenziatore* trattate nel documento *SpecificaTecnica\_v3.0.0.pdf*

## Diario delle modifiche

Versione	Data	Autore	Ruolo	Descrizione
----------	------	--------	-------	-------------

## Indice

<b>1</b>	<b>Introduzione</b>	<b>2</b>
1.1	Scopo del documento	2
1.2	Scopo del Prodotto	2
1.3	Glossario	2
1.4	Riferimenti	2
1.4.1	Normativi	2
1.4.2	Informativi	2
<b>2</b>	<b>Standard di progetto</b>	<b>4</b>
2.1	Standard di progettazione architetturale	4
2.2	Standard di documentazione del codice	4
2.3	Standard di denominazione di entità e relazioni	4
2.4	Standard di programmazione	4
2.5	Strumenti di lavoro	4
<b>3</b>	<b>Specifica della componente view</b>	<b>5</b>
3.1	Package com.sirius.sequenziatore.client.view	6
3.2	Package com.sirius.sequenziatore.client.view.user	6
3.3	Package com.sirius.sequenziatore.client.view.processowner	7
<b>4</b>	<b>Specifica della componente presenter</b>	<b>9</b>
4.1	Client	9
4.1.1	Package com.sirius.sequenziatore.client.presenter	10
4.1.2	Package com.sirius.sequenziatore.client.presenter.user	13
4.1.3	Package com.sirius.sequenziatore.client.presenter.processowner	30
4.2	Server	41
4.2.1	Package com.sirius.sequenziatore.server.presenter.common	42
4.2.2	Package com.sirius.sequenziatore.server.presenter.processowner	45
4.2.3	Package com.sirius.sequenziatore.server.presenter.user	48
<b>5</b>	<b>Specifica della componente model</b>	<b>52</b>
5.1	Client	52
5.1.1	Package com.sirius.sequenziatore.client.model	52
5.1.2	Package com.sirius.sequenziatore.client.model.collection	56
5.2	Server	60
5.2.1	Package com.sirius.sequenziatore.client.model	61

## Elenco delle tabelle

## Elenco delle figure

1	Diagramma classe <i>Router</i> . . . . .	10
2	Diagramma classe <i>Login</i> . . . . .	12
3	Diagramma classe <i>MainUser</i> . . . . .	14
4	Diagramma classe <i>Register</i> . . . . .	15
5	Diagramma classe <i>UserData</i> . . . . .	17
6	Diagramma classe <i>OpenProcess</i> . . . . .	19
7	Diagramma classe <i>ManagementProcess</i> . . . . .	20
8	Diagramma classe <i>PrintReport</i> . . . . .	22
9	Diagramma classe <i>SendData</i> . . . . .	23
10	Diagramma classe <i>SendText</i> . . . . .	25
11	Diagramma classe <i>SendNumb</i> . . . . .	26
12	Diagramma classe <i>SendImage</i> . . . . .	27
13	Diagramma classe <i>SendPosition</i> . . . . .	29
14	Diagramma classe <i>MainProcessOwner</i> . . . . .	30
15	Diagramma classe <i>OpenProcess</i> . . . . .	31
16	Diagramma classe <i>NewProcess</i> . . . . .	33
17	Diagramma classe <i>AddStep</i> . . . . .	35
18	Diagramma classe <i>ManageProcess</i> . . . . .	37
19	Diagramma classe <i>CheckStep</i> . . . . .	39
20	Diagramma package - <code>com.sirius.sequenziatore.server.presenter.common</code> 42	
21	Diagramma classe - <code>SignUpController</code> . . . . .	42
22	Diagramma classe - <code>LoginController</code> . . . . .	43
23	Diagramma classe - <code>StepInfoController</code> . . . . .	44
24	Diagramma classe - <code>ProcessInfoController</code> . . . . .	44
25	Diagramma package - <code>com.sirius.sequenziatore.server.presenter.processowner</code> 45	
26	Diagramma classe - <code>StepController</code> . . . . .	45
27	Diagramma classe - <code>ProcessController</code> . . . . .	46
28	Diagramma classe - <code>ApproveStepController</code> . . . . .	47
29	Diagramma package - <code>com.sirius.sequenziatore.server.presenter.user</code> 48	
30	Diagramma classe - <code>UserStepController</code> . . . . .	48
31	Diagramma classe - <code>UserProcessController</code> . . . . .	49
32	Diagramma classe - <code>AccountController</code> . . . . .	50
33	Diagramma classe - <code>ReportController</code> . . . . .	51
34	Diagramma classe <i>UserModel</i> . . . . .	52
35	Diagramma classe <i>ProcessModel</i> . . . . .	53

36	Diagramma classe <i>ProcessDataModel</i> . . . . .	54
37	Diagramma classe <i>StepModel</i> . . . . .	55
38	Diagramma classe <i>ProcessCollection</i> . . . . .	56
39	Diagramma classe <i>ProcessDataCollection</i> . . . . .	57
40	Diagramma classe <i>StepCollection</i> . . . . .	58

## 1 Introduzione

### 1.1 Scopo del documento

In questo documento si prefigge come obiettivo la definizione in modo approfondito della struttura e delle relazioni tra le componenti del prodotto *software Sequenziatore*, approfondendo quanto riportato nel documento *SpecificaTecnica\_v3.0.0.pdf*.

### 1.2 Scopo del Prodotto

Lo scopo del progetto *Sequenziatore*, è di fornire un servizio di gestione di processi definiti da una serie di passi da eseguirsi in sequenza o senza un ordine predefinito, utilizzabile da dispositivi mobili di tipo *smartphone* o *tablet*.

### 1.3 Glossario

Al fine di rendere più leggibili e comprensibili i documenti, i termini tecnici, di dominio, gli acronimi e le parole che necessitano di essere chiarite, sono riportate nel documento *Glossario\_v3.0.0.pdf*.

Ciascuna occorrenza dei vocaboli presenti nel *Glossario* è seguita da una “G” maiuscola in pedice.

### 1.4 Riferimenti

#### 1.4.1 Normativi

- Norme di Progetto: *NormeDiProgetto\_v3.0.0.pdf*;
- Analisi dei Requisiti: *AnalisiDeiRequisiti\_v3.0.0.pdf*;
- Specifica tecnica: *SpecificaTecnica\_v3.0.0.pdf*.

#### 1.4.2 Informativi

- Developing Backbone.js Applications, Addy Osmani  
<http://addyosmani.github.io/backbone-fundamentals>;
- BackboneJS  
<http://backbonejs.org/>;
- Documentazione Spring.io  
<http://spring.io/docs>;
- Regolamento dei documenti, prof. Vardanega Tullio:  
<http://www.math.unipd.it/~tullio/IS-1/2013/>;

- Dispense di ingegneria del software:
  - Programmazione: criteri e strategie, prof. Vardanega Tullio:  
<http://www.math.unipd.it/~rcardin/pdf/B02.pdf>;
  - Diagrammi delle classi e degli oggetti, prof. Cardin Riccardo:  
<http://www.math.unipd.it/~tullio/IS-1/2013/Dispense/E02a.pdf>;
  - Diagrammi di sequenza, prof. Cardin Riccardo:  
<http://www.math.unipd.it/~tullio/IS-1/2013/Dispense/E03a.pdf>;
  - Diagrammi dei package, prof. Cardin Riccardo:  
<http://www.math.unipd.it/~tullio/IS-1/2013/Dispense/E05.pdf>;



## 2 Standard di progetto

### 2.1 Standard di progettazione architettuale

Gli standard di progettazione architettuale sono definiti nel documento *Specifica Tecnica\_v3.0.0.pdf*.

### 2.2 Standard di documentazione del codice

Gli standard di documentazione del codice sono definiti nel documento *NormeDiProgetto\_v3.0.0.pdf*.

### 2.3 Standard di denominazione di entità e relazioni

Gli standard di denominazione di dei *package*, delle classi, degli attributi e dei metodi, sono definiti nel documento *NormeDiProgetto\_v3.0.0.pdf*.

### 2.4 Standard di programmazione

Gli standard di programmazione sono definiti nel documento *NormeDiProgetto\_v3.0.0.pdf*.

### 2.5 Strumenti di lavoro

Gli strumenti da utilizzare e le procedure da seguire durante lo sviluppo del prodotto *software Sequenziatore*, sono definiti nel documento *NormeDiProgetto\_v3.0.0.pdf*.

### 3 Specifica della componente view

La componente *view* è formata da *template HTML<sub>G</sub>* che possono contenere codice *javascript<sub>G</sub>* che, utilizzati dalle componenti del *presenter*, consentono di renderizzare l'interfaccia grafica dell'applicazione.

Le componenti del *presenter*, si interfacciano con la *view* utilizzando il metodo `template` della libreria *underscoreJS*, che consente di generare codice *HTML<sub>G</sub>* a seconda dei parametri del metodo. Per questo motivo, le interfacce presenti nel *package* `com.sirius.sequenziatore.client.view` definite nel documento *SpecificaTecnica\_v3.0.0.pdf*, non verranno né implementate né descritte nel presente documento.

La componente *view* è composta dai seguenti *template*:

- `com.sirius.sequenziatore.client.view.Login`;
- `com.sirius.sequenziatore.client.view.user.MainUser`;
- `com.sirius.sequenziatore.client.view.user.Register`;
- `com.sirius.sequenziatore.client.view.user.UserData`;
- `com.sirius.sequenziatore.client.view.user.OpenProcess`;
- `com.sirius.sequenziatore.client.view.user.ManagementProcess`;
- `com.sirius.sequenziatore.client.view.user.SendData`;
- `com.sirius.sequenziatore.client.view.user.SendText`;
- `com.sirius.sequenziatore.client.view.user.SendNumb`;
- `com.sirius.sequenziatore.client.view.user.SendPosition`;
- `com.sirius.sequenziatore.client.view.user.SendImage`;
- `com.sirius.sequenziatore.client.view.user.PrintProcess`;
- `com.sirius.sequenziatore.client.view.processowner.MainProcessOwner`;
- `com.sirius.sequenziatore.client.view.processowner.NewProcess`;
- `com.sirius.sequenziatore.client.view.processowner.AddStep`;
- `com.sirius.sequenziatore.client.view.processowner.OpenProcess`;
- `com.sirius.sequenziatore.client.view.processowner.ManageProcess`;
- `com.sirius.sequenziatore.client.view.processowner.CheckStep`;

### 3.1 Package `com.sirius.sequenziatore.client.view`

#### 3.1.0.1 Login

- **Descrizione:** *Template HTML* che permette di gestire l'interfaccia grafica relativa alle richieste di autenticazione al sistema.

### 3.2 Package `com.sirius.sequenziatore.client.view.user`

#### 3.2.0.2 MainUser

- **Descrizione:** Classe che permette la gestione delle principali componenti dell'interfaccia grafica dell'utente.

#### 3.2.0.3 Register

- **Descrizione:** *Template HTML* che permette di gestire dell'interfaccia grafica relativa alle richieste di registrazione da parte dell'utente.

#### 3.2.0.4 UserData

- **Descrizione:** *Template HTML* che permette la realizzazione dei *widget* che consentono visualizzazione e modifica dei dati dell'utente.

#### 3.2.0.5 OpenProcess

- **Descrizione:** *Template HTML* che permette di realizzare i *widget* per consentire l'apertura di un processo tramite ricerca o selezionandolo da una lista.

#### 3.2.0.6 ManagementProcess

- **Descrizione:** *Template HTML* che permette di realizzare i *widget* per consentire la visualizzazione dello stato del processo selezionato e i vincoli per concludere il passo in corso.

#### 3.2.0.7 SendData

- **Descrizione:** *Template HTML* che permette di realizzare i *widget* per consentire l'invio dei dati richiesti per la conclusione del passo in esecuzione.

#### 3.2.0.8 SendText

- **Descrizione:** *Template HTML* che permette di realizzare i *widget* che consentono di inserire il testo da inviare per concludere il passo in esecuzione.

#### 3.2.0.9 SendNumb

- **Descrizione:** *Template HTML* che permette agli oggetti che la implementano di realizzare i *widget* che consentono di inserire i dati numerici da inviare per concludere il passo in esecuzione.

#### 3.2.0.10 SendPosition

- **Descrizione:** *Template HTML* che permette di realizzare i *widget* che consentono di inviare la posizione geografica richiesta per la conclusione del passo in esecuzione.

#### 3.2.0.11 SendImage

- **Descrizione:** *Template HTML* che permette di realizzare i *widget* che consentono di inserire le immagini richieste per concludere il passo in esecuzione.

#### 3.2.0.12 PrintProcess

- **Descrizione:** *Template HTML* che permette di realizzare i *widget* che consentono il salvataggio dei *report* sull'esecuzione del processo.

### 3.3 Package com.sirius.sequenziatore.client.view.processowner

#### 3.3.0.13 MainProcessOwner

- **Descrizione:** Componente che permette la gestione delle principali componenti dell'interfaccia grafica dell'utente *process owner*.

#### 3.3.0.14 NewProcess

- **Descrizione:** *Template HTML* che permette di gestire l'interfaccia grafica che consente di creare nuovi processi.

#### 3.3.0.15 AddStep

- **Descrizione:** *Template HTML* che permette di gestire l'interfaccia grafica che consente di definire un nuovo passo del processo in creazione.

#### 3.3.0.16 OpenProcess

- **Descrizione:** *Template HTML* che permette di realizzare i *widget* che consentono di aprire un processo tramite ricerca o selezionandolo da una lista.

### 3.3.0.17 ManageProcess

- **Descrizione:** *Template HTML* che permette di realizzare *iwidget* che consentono di gestire l'accesso ai dati inviati al *server<sub>G</sub>* dagli utenti.

### 3.3.0.18 CheckStep

- **Descrizione:** *Template HTML* che permette di realizzare *iwidget* che consentono di gestire l'approvazione dei passi che richiedono intervento umano.

## 4 Specifica della componente presenter

Questa componente consente la gestione della logica principale dell'applicazione *Sequenziatore* e viene suddivisa in due parti: *client* e *server*.

### 4.1 Client

Il *presenter* lato *client* consente di gestire la logica delle pagine dell'applicazione. La inizializzazione delle classi e la gestione degli eventi di cambio pagina, avviene tramite la classe principale **Router**, che estende la classe **Backbone.Router** fornita dal *framework<sub>G</sub> Backbone*. Le altre classi della componente, consentono di renderizzare le viste utilizzando i *template* della componente *view*, di gestire gli eventi generati dagli utenti, e di gestire la comunicazione con il server tramite le classi della componente *model*.

La componente è formata dalle seguenti *classi*:

- `com.sirius.sequenziatore.client.presenter.Router`;
- `com.sirius.sequenziatore.client.presenter.Login`;
- `com.sirius.sequenziatore.client.presenter.user.MainUser`;
- `com.sirius.sequenziatore.client.presenter.user.Register`;
- `com.sirius.sequenziatore.client.presenter.user.UserData`;
- `com.sirius.sequenziatore.client.presenter.user.OpenProcess`;
- `com.sirius.sequenziatore.client.presenter.user.ManagementProcess`;
- `com.sirius.sequenziatore.client.presenter.user.SendData`;
- `com.sirius.sequenziatore.client.presenter.user.SendText`;
- `com.sirius.sequenziatore.client.presenter.user.SendNumb`;
- `com.sirius.sequenziatore.client.presenter.user.SendPosition`;
- `com.sirius.sequenziatore.client.presenter.user.SendImage`;
- `com.sirius.sequenziatore.client.presenter.user.PrintReport`;
- `com.sirius.sequenziatore.client.presenter.processowner.MainProcessOwner`;
- `com.sirius.sequenziatore.client.presenter.processowner.NewProcess`;
- `com.sirius.sequenziatore.client.presenter.processowner.AddStep`;
- `com.sirius.sequenziatore.client.presenter.processowner.OpenProcess`;

- [com.sirius.sequenziatore.client.presenter.processowner.ManageProcess;](#)
- [com.sirius.sequenziatore.client.presenter.processowner.CheckStep.](#)

#### 4.1.1 Package com.sirius.sequenziatore.client.presenter

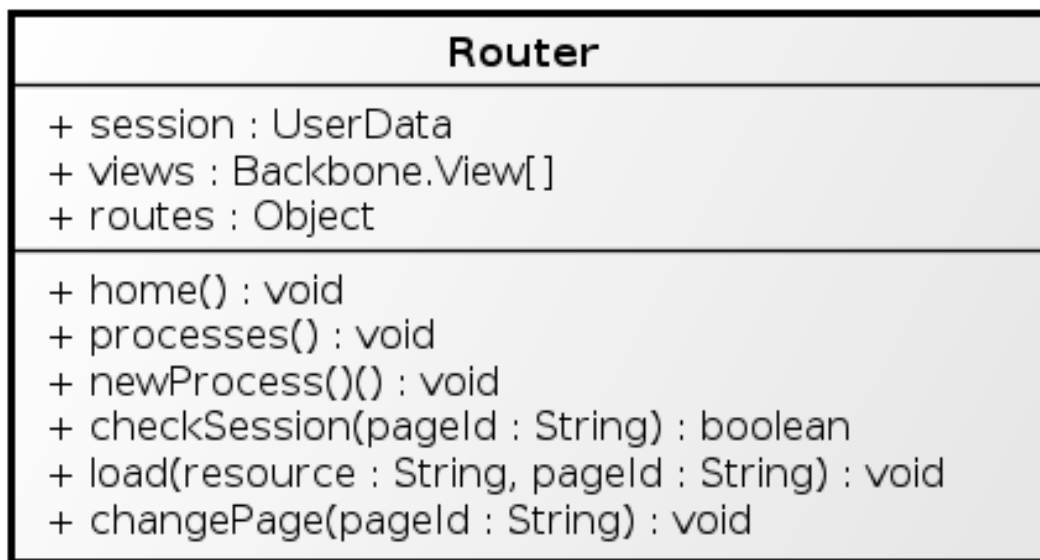


Figura 1: Diagramma classe *Router*

##### 4.1.1.1 Router

- **Descrizione:** Classe che permette di coordinare l’inizializzazione e la renderizzazione delle pagine, gestendo gli eventi e le azioni di cambio pagina;
- **Relazioni con altri componenti:**

La classe reperisce le informazioni di sessione dalla classe `com.sirius.sequenziatore.client.model::UserModel` e comunica con le seguenti classi se l’utente dispone dei diritti d’accesso necessari:

- `com.sirius.sequenziatore.client.presenter.Login;`
- `com.sirius.sequenziatore.client.presenter.user.Register;`
- `com.sirius.sequenziatore.client.presenter.user.MainUser;`
- `com.sirius.sequenziatore.client.presenter.user.UserData;`
- `com.sirius.sequenziatore.client.presenter.user.OpenProcessgic;`
- `com.sirius.sequenziatore.client.presenter.user.ManagmentProcess;`

- `com.sirius.sequenziatore.client.presenter.processowner.MainProcessOwner;`
- `com.sirius.sequenziatore.client.presenter.processowner.OpenProcess;`
- `com.sirius.sequenziatore.client.presenter.processowner.NewProcess;`
- `com.sirius.sequenziatore.client.presenter.processowner.CheckStep;`
- `com.sirius.sequenziatore.client.presenter.processowner.ManageProcess;`

- **Attributi:**

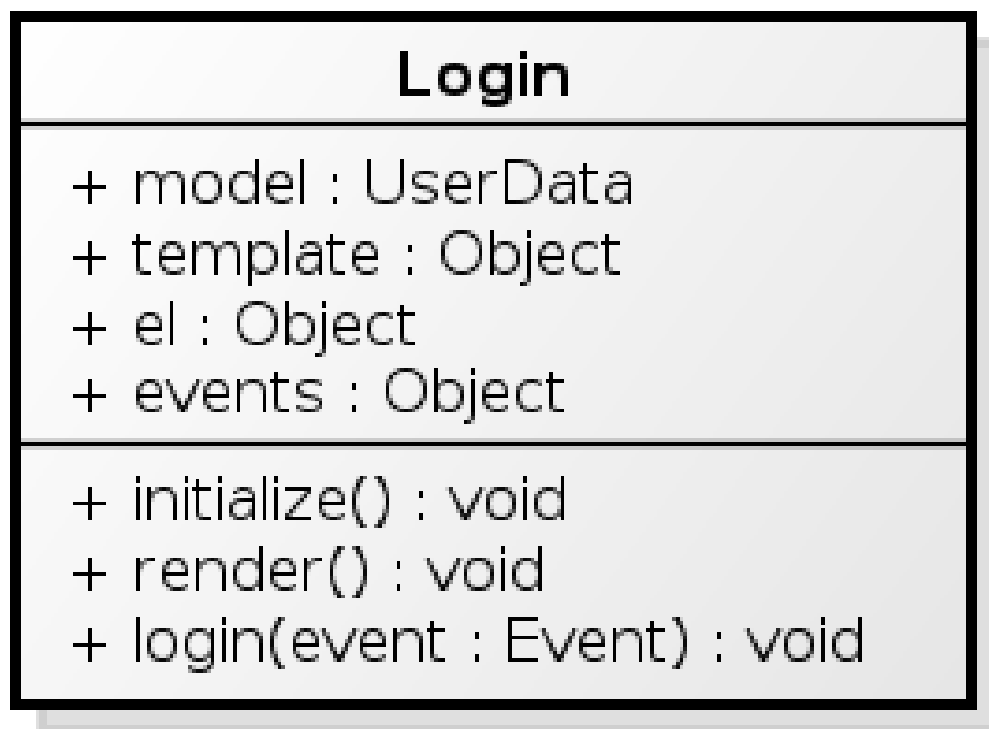
- `+ UserData session:`  
oggetto di tipo `com.sirius.sequenziatore.client.model.UserData`, che consente di gestire la sessione dell'utente;
- `+ Backbone.View[] views:`  
array che contiene le classi del presenter in esecuzione;
- `+ Object routes:`  
oggetto ridefinito da `Backbone.Router` che associa ad ogni evento di *routing<sub>G</sub>*, un metodo della classe;

- **Metodi:**

- `+ void home():`  
gestisce l'evento di *routing<sub>G</sub> home*;
- `+ void processes():`  
gestisce l'evento di *routing<sub>G</sub> processes*;
- `+ void newProcess():`  
gestisce l'evento di *routing<sub>G</sub> newProcess*;
- `+ void checkStep():`  
gestisce l'evento di *routing<sub>G</sub> checkStep*;
- `+ void process():`  
gestisce l'evento di *routing<sub>G</sub> process*;
- `+ void register():`  
gestisce l'evento di *routing<sub>G</sub> register*;
- `+ void user():`  
gestisce l'evento di *routing<sub>G</sub> user*;



- + `bool checkSession(String pageId):`  
ritorna `true` solo se l'utente è autenticato; in caso contrario crea e renderizza la pagina di *login*;
- + `void load(String resource, String pageId):`  
crea e aggiunge una vista di tipo *resource* al campo dati `this.views`, all'indice *pageId*;
- + `void changePage(String pageId):`  
imposta la pagina con id *pageId* come attiva, ed esegue la transizione di cambio pagina.


Figura 2: Diagramma classe *Login*

#### 4.1.1.2 Login

- **Descrizione:** Classe che ha il compito di gestire le richieste di autenticazione al sistema;
- **Relazioni con altri componenti:**

La classe gestisce i dati di sessione comunicando con la classe `com.sirius.sequenziatore.client.model.UserModel` e realizza l'interfaccia grafica utilizzando il *template* `com.sirius.sequenziatore.client.viewLogin`.

- **Attributi:**

- + `UserDataModel model`:  
campo dati di tipo `com.sirius.sequenziatore.client.model.UserModel` che contiene i dati di sessione dell'utente;
- + `Object template`:  
oggetto ridefinito da `Backbone.View`, che contiene il *template* `HTMLG` associato alla classe;
- + `Object el`:  
oggetto ridefinito da `Backbone.View` che rappresenta l'elemento `HTMLG` entro cui la classe ascolta eventi generati dagli utenti;
- + `Object events`:  
oggetto ridefinito da `Backbone.View` che associa ad ogni evento generato dagli utenti nella pagina `HTMLG`, un metodo della classe;

- **Metodi:**

- + `void initialize()`:  
metodo ridefinito da `Backbone.View`, invocato alla costruzione di ciascun oggetto della classe, che consente di aggiungere una pagina `HTMLG` associata al componente;
- + `void render()`:  
metodo ridefinito da `Backbone.View`, che consente di aggiungere alla pagina `HTMLG` il *template* campo dati della classe;
- + `void login(Event event)`:  
effettua una richiesta di *login*, utilizzando il campo dati `com.sirius.sequenziatore.client.model` per comunicare con il *serverG*.

#### 4.1.2 Package `com.sirius.sequenziatore.client.presenter.user`

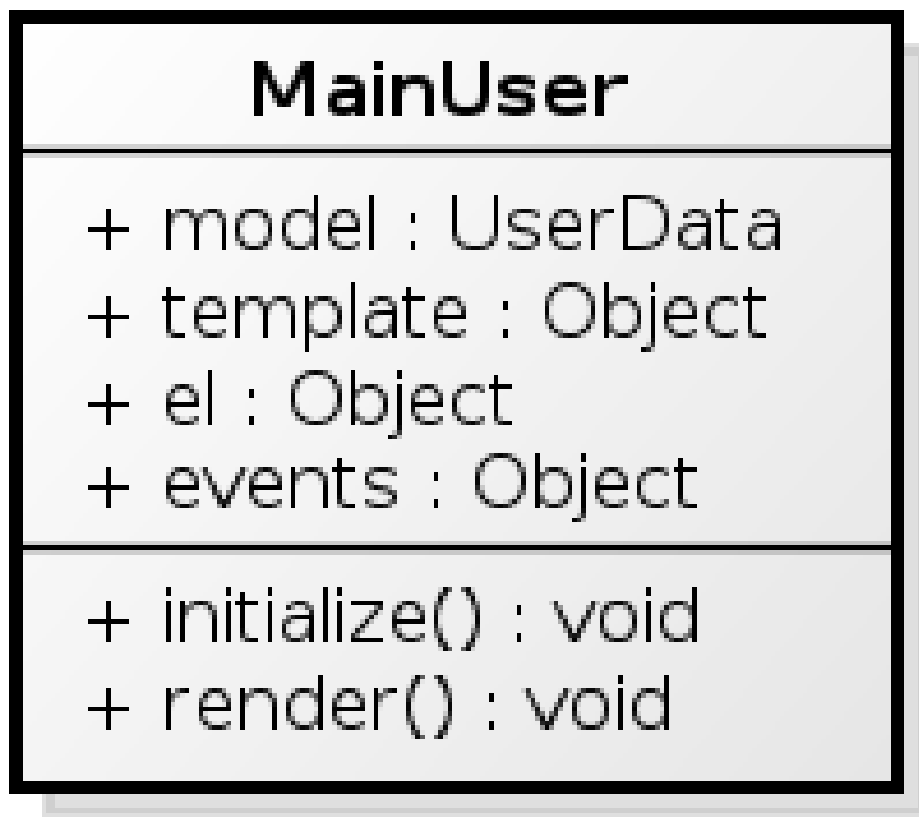


Figura 3: Diagramma classe *MainUser*

#### 4.1.2.1 MainUser

- **Descrizione:** Classe che ha il compito della gestione generale della logica delle funzionalità utente;

- **Relazioni con altri componenti:**

La classe comunica con l'interfaccia `com.sirius.sequenziatore.client.view.user.IMainUser` per la realizzazione dell'interfaccia grafica.

- **Attributi:**

– + `UserDataModel model`:  
campo dati di tipo  
`com.sirius.sequenziatore.client.model.UserModel` che contiene i dati di sessione dell'utente;

- + `Object template`:  
oggetto ridefinito da `Backbone.View`, che contiene il *template* `HTMLG` associato alla classe;
- + `Object el`:  
oggetto ridefinito da `Backbone.View` che rappresenta l'elemento `HTMLG` entro cui la classe ascolta eventi generati dagli utenti;
- + `Object events`:  
oggetto ridefinito da `Backbone.View` che associa ad ogni evento generato dagli utenti nella pagina `HTMLG`, un metodo della classe;

• **Metodi:**

- + `void initialize()`:  
metodo ridefinito da `Backbone.View`, invocato alla costruzione di ciascun oggetto della classe, che consente di aggiungere una pagina `HTMLG` associata al componente;
- + `void render()`:  
metodo ridefinito da `Backbone.View`, che consente di aggiungere alla pagina `HTMLG` il *template* campo dati della classe.

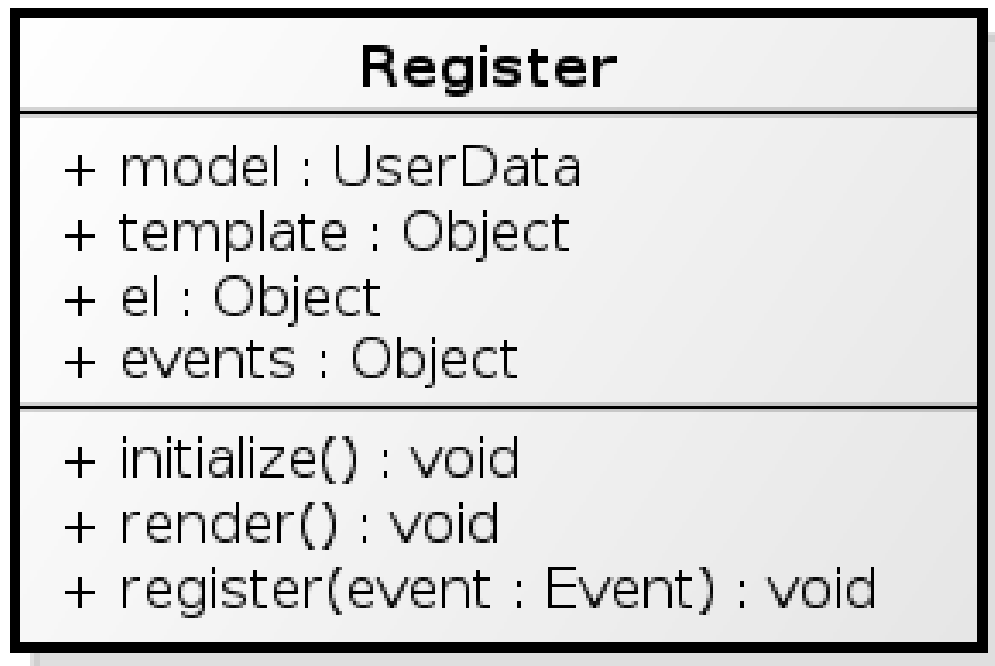


Figura 4: Diagramma classe *Register*

#### 4.1.2.2 Register

- **Descrizione:** Classe che ha il compito di gestire le richieste di registrazione da parte dell'utente;

- **Relazioni con altri componenti:**

La classe comunica con l'interfaccia `com.sirius.sequenziatore.client.view.user.IRegister` per la realizzazione dei *widget* per la registrazione, e con la classe `com.sirius.sequenziatore.client.model.UserModel` per comunicare col il *server*.

- **Attributi:**

- + `UserDataModel model`:  
campo dati di tipo `com.sirius.sequenziatore.client.model.UserModel` che contiene i dati utente e di sessione;
- + `Object template`:  
oggetto ridefinito da `Backbone.View`, che contiene il *template HTML<sub>G</sub>* associato alla classe;
- + `Object el`:  
oggetto ridefinito da `Backbone.View` che rappresenta l'elemento *HTML<sub>G</sub>* entro cui la classe ascolta eventi generati dagli utenti;
- + `Object events`:  
oggetto ridefinito da `Backbone.View` che associa ad ogni evento generato dagli utenti nella pagina *HTML<sub>G</sub>*, un metodo della classe;

- **Metodi:**

- + `void initialize()`:  
metodo ridefinito da `Backbone.View`, invocato alla costruzione di ciascun oggetto della classe, che consente di aggiungere una pagina *HTML<sub>G</sub>* associata al componente;
- + `void render()`:  
metodo ridefinito da `Backbone.View`, che consente di aggiungere alla pagina *HTML<sub>G</sub>* il *template* campo dati della classe;
- + `void register(Event event)`:  
effettua una richiesta di registrazione, utilizzando il campo dati `com.sirius.sequenziatore.client.model` per comunicare con il *server*.

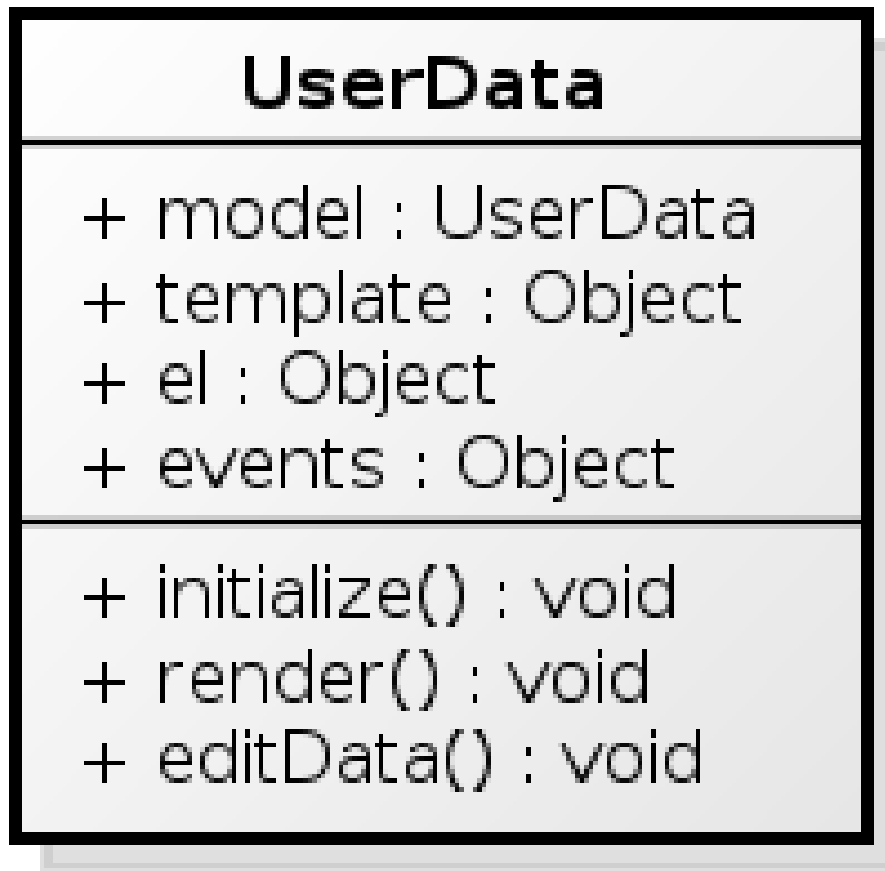


Figura 5: Diagramma classe *UserData*

#### 4.1.2.3 UserData

- **Descrizione:** Classe che ha il compito di gestire la visualizzazione e la modifica dei dati dell'utente;

- **Relazioni con altri componenti:**

La classe comunica con l'interfaccia `com.sirius.sequenziatore.client.view.user.IUserData` per realizzare il *widget* preposto alla visualizzazione e modifica dei dati dell'utente, e con la classe `com.sirius.sequenziatore.client.model.UserModel` per comunicare col il *server*<sub>G</sub>.

- **Attributi:**

– + `UserDataModel model`:

campo dati di tipo

`com.sirius.sequenziatore.client.model.UserModel` che contiene i dati utente e di sessione;

– + `Object template`:

oggetto ridefinito da `Backbone.View`, che contiene il *template* `HTMLG` associato alla classe;

– + `Object el`:

oggetto ridefinito da `Backbone.View` che rappresenta l'elemento `HTMLG` entro cui la classe ascolta eventi generati dagli utenti;

– + `Object events`:

oggetto ridefinito da `Backbone.View` che associa ad ogni evento generato dagli utenti nella pagina `HTMLG`, un metodo della classe;

• **Metodi:**

– + `void initialize()`:

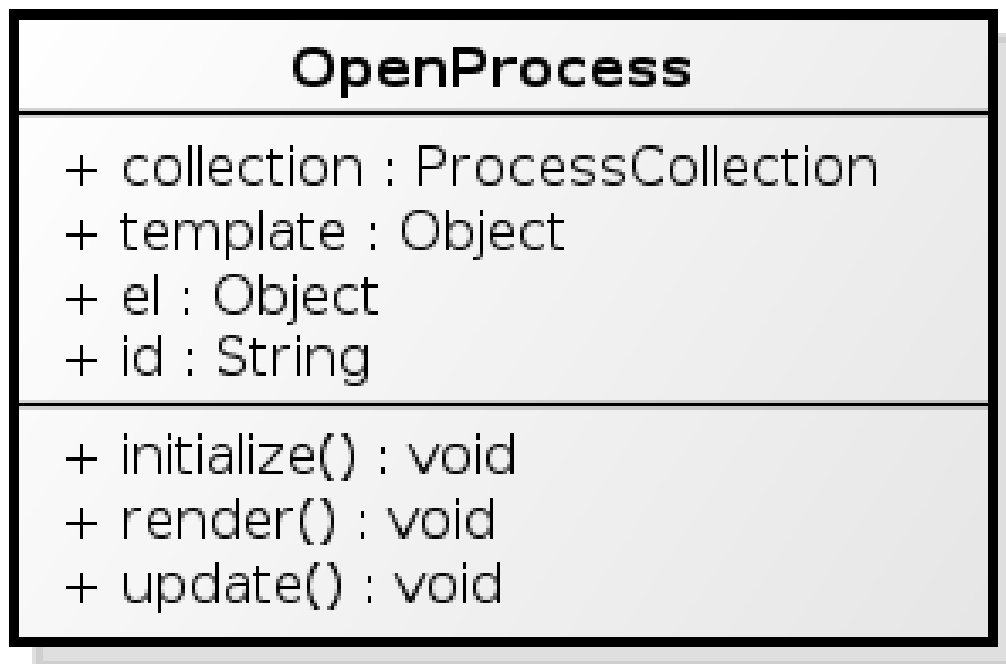
metodo ridefinito da `Backbone.View`, invocato alla costruzione di ciascun oggetto della classe, che consente di aggiungere una pagina `HTMLG` associata al componente;

– + `void render()`:

metodo ridefinito da `Backbone.View`, che consente di aggiungere alla pagina `HTMLG` il *template* campo dati della classe;

– + `void editData()`:

utilizza il campo dati `model` per salvare i dati modificati dall'utente nel `serverG`.


Figura 6: Diagramma classe *OpenProcess*

#### 4.1.2.4 OpenProcess

- **Descrizione:** Classe che ha il compito di selezionare, ricercare e aprire un processo fra quelli eseguibili;

- **Relazioni con altri componenti:**

La classe realizza e modifica l'opportuno *widget* mediante l'interfaccia `com.sirius.sequenziatore.client.view.user.IOpenProcess` e utilizza la classe

`com.sirius.sequenziatore.client.model.collection.ProcessCollection` per gestire e ottenere i dati dal *server<sub>G</sub>*.

- **Attributi:**

- + `ProcessCollection collection`:  
campo dati di tipo `com.sirius.sequenziatore.client.model.collection.ProcessCollection` che contiene la lista dei processi non terminati o non ancora eliminati dall'utente;
- + `Object template`:  
oggetto ridefinito da `Backbone.View`, che contiene il *template HTML<sub>G</sub>* associato alla classe;



- + `Object el`:  
oggetto ridefinito da `Backbone.View` che rappresenta l'elemento *HTML<sub>G</sub>* entro cui la classe ascolta eventi generati dagli utenti;
- + `String id`:  
campo dati ridefinito da `Backbone.View` contenente l'id della classe;

• **Metodi:**

- + `void initialize()`:  
metodo ridefinito da `Backbone.View`, invocato alla costruzione di ciascun oggetto della classe, che consente di aggiungere una pagina *HTML<sub>G</sub>* associata al componente;
- + `void render()`:  
metodo ridefinito da `Backbone.View`, che consente di aggiungere alla pagina *HTML<sub>G</sub>* il *template* campo dati della classe;
- + `void update()`:  
aggiorna il campo dati *collection* comunicando con il *server<sub>G</sub>*.

#### 4.1.2.5 ManagementProcess

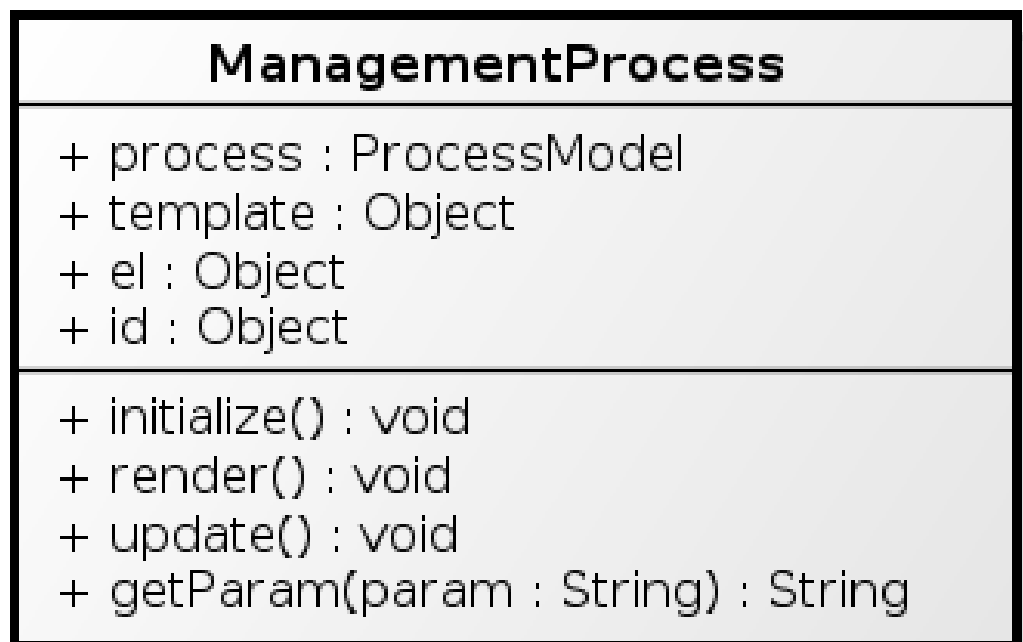


Figura 7: Diagramma classe *ManagementProcess*

- **Descrizione:** Classe che ha il compito di gestire e accedere alle informazioni relative allo stato del processo selezionato.;

- **Relazioni con altri componenti:**

La classe comunica con l'interfaccia `com.sirius.sequenziatore.client.view.user.IManagmentProcess` per realizzare il *widget* che permette la gestione del processo selezionato, utilizza la classe `com.sirius.sequenziatore.client.model.ProcessModel` per gestire e ottenere i dati dal *server<sub>G</sub>*, e provvede ad invocare le seguenti classi in base alle decisioni dell'utente:

- `com.sirius.sequenziatore.client.presenter.user.PrintReport;`
- `com.sirius.sequenziatore.client.presenter.user.SendData.`

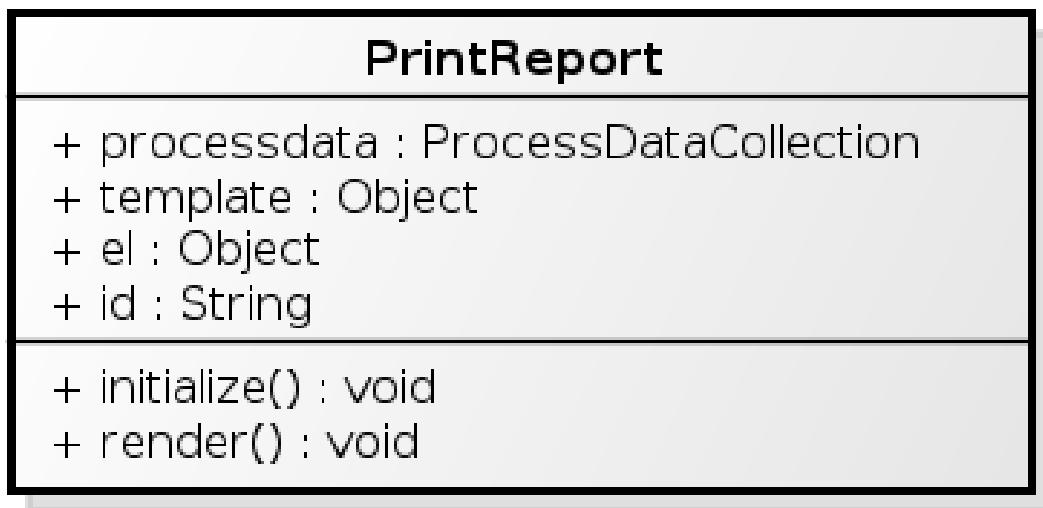
- **Attributi:**

- + `ProcessModel process:`  
campo dati di tipo `com.sirius.sequenziatore.client.model.ProcessModel` che contiene i dati del processo in gestione;
- + `Object template:`  
oggetto ridefinito da `Backbone.View`, che contiene il *template HTML<sub>G</sub>* associato alla classe;
- + `Object el:`  
oggetto ridefinito da `Backbone.View` che rappresenta l'elemento *HTML<sub>G</sub>* entro cui la classe ascolta eventi generati dagli utenti;
- + `String id:`  
campo dati ridefinito da `Backbone.View` contenente l'id della classe;

- **Metodi:**

- + `void initialize():`  
metodo ridefinito da `Backbone.View`, invocato alla costruzione di ciascun oggetto della classe, che consente di aggiungere una pagina *HTML<sub>G</sub>* associata al componente;
- + `void render():`  
metodo ridefinito da `Backbone.View`, che consente di aggiungere alla pagina *HTML<sub>G</sub>* il *template* campo dati della classe;
- + `void update():`  
aggiorna i campi dati `process` e `processData` comunicando con il *server<sub>G</sub>*;

- + String getParam(String param):  
ritorna il valore del parametro *param* se presente nella *URL<sub>G</sub>*.


Figura 8: Diagramma classe *PrintReport*

#### 4.1.2.6 PrintReport

- **Descrizione:** Classe che ha il compito di gestire la creazione del report di fine processo;

- **Relazioni con altri componenti:**

La classe comunica con l'interfaccia `com.sirius.sequenziatore.client.view.user.IPrintReport` per realizzare il *widget* per creare il report di fine processo, e utilizza la classe `com.sirius.sequenziatore.client.model.collection.ProcessDataCollection` per gestire e ottenere i dati dal *server<sub>G</sub>*.

- **Attributi:**

- + ProcessDataCollection processdata:  
campo dati di tipo `com.sirius.sequenziatore.client.model.collection.ProcessDataCollection` che contiene i dati inviati dall'utente relativi al processo in gestione;
- + Object template:  
oggetto ridefinito da `Backbone.View`, che contiene il *template HTML<sub>G</sub>* associato alla classe;

- + `Object el`:  
oggetto ridefinito da `Backbone.View` che rappresenta l'elemento *HTML<sub>G</sub>* entro cui la classe ascolta eventi generati dagli utenti;
- + `String id`:  
campo dati ridefinito da `Backbone.View` contenente l'id della classe;

• **Metodi:**

- + `void initialize()`:  
metodo ridefinito da `Backbone.View`, invocato alla costruzione di ciascun oggetto della classe, che consente di aggiungere una pagina *HTML<sub>G</sub>* associata al componente;
- + `void render()`:  
metodo ridefinito da `Backbone.View`, che consente di aggiungere alla pagina *HTML<sub>G</sub>* il *template* campo dati della classe.

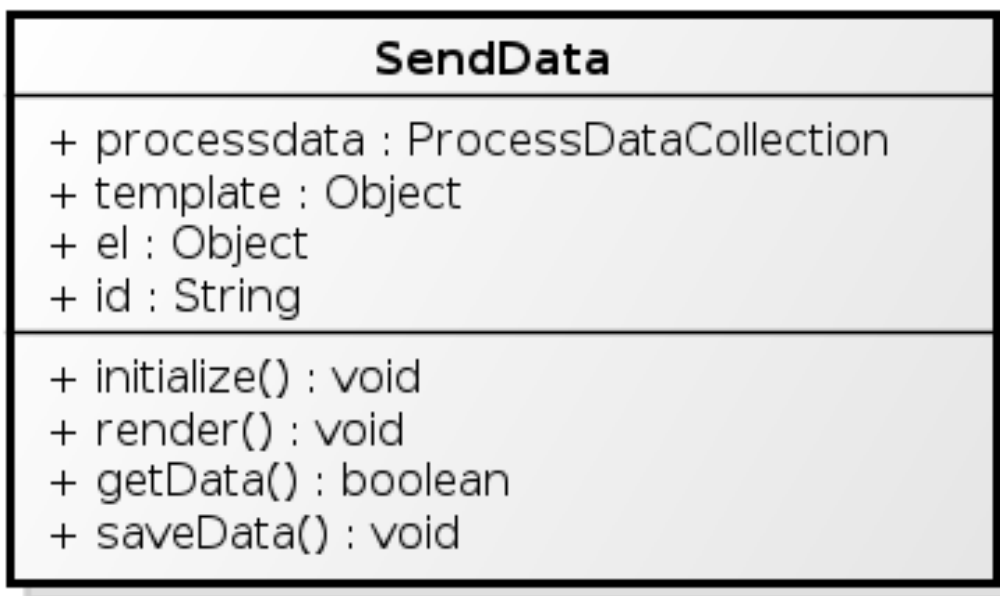


Figura 9: Diagramma classe *SendData*

#### 4.1.2.7 SendData

- **Descrizione:** Classe che ha il compito di gestire l'inserimento e l'invio di dati da parte degli utenti, per completare il passo corrente;
- **Relazioni con altri componenti:**

La classe comunica con l'interfaccia

`com.sirius.sequenziatore.client.view.user.ISendData` per creare il *widget* che consente di inviare i dati, utilizza la classe `com.sirius.sequenziatore.client.model.collection.ProcessDataCollection` per gestire e ottenere i dati dal *server<sub>G</sub>*, e infine invoca le seguenti classi che gestiscono l'invio di un tipo di dato specifico:

- `com.sirius.sequenziatore.client.presenter.user.SendText;`
- `com.sirius.sequenziatore.client.presenter.user.SendNumb;`
- `com.sirius.sequenziatore.client.presenter.user.SendImage;`
- `com.sirius.sequenziatore.client.presenter.user.SendPosition.`

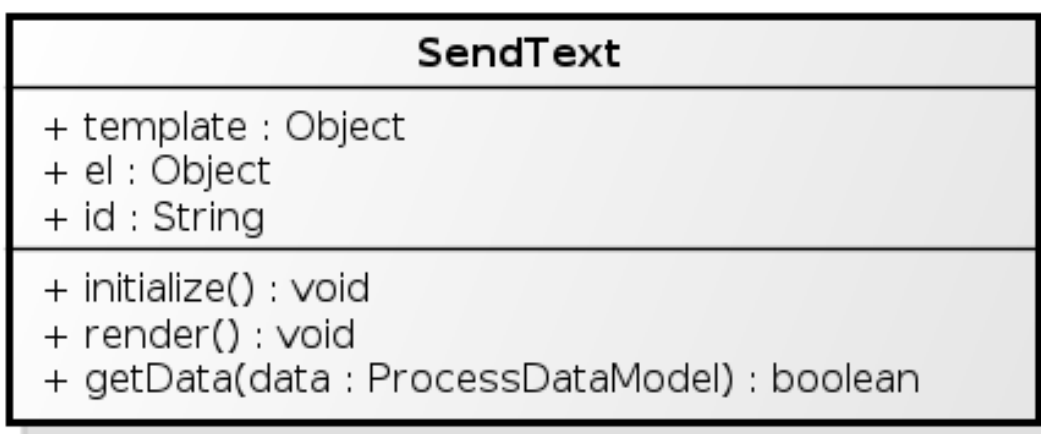
● **Attributi:**

- + `ProcessDataCollection processdata:`  
campo dati di tipo `com.sirius.sequenziatore.client.model.collection.ProcessDataCollection` che consente di interagire con la lista dei dati inviati dall'utente relativa al processo in gestione presente nel *server<sub>G</sub>*;
- + `Object template:`  
oggetto ridefinito da `Backbone.View`, che contiene il *template HTML<sub>G</sub>* associato alla classe;
- + `Object el:`  
oggetto ridefinito da `Backbone.View` che rappresenta l'elemento *HTML<sub>G</sub>* entro cui la classe ascolta eventi generati dagli utenti;
- + `String id:`  
campo dati ridefinito da `Backbone.View` contenente l'id della classe;

● **Metodi:**

- + `void initialize():`  
metodo ridefinito da `Backbone.View`, invocato alla costruzione di ciascun oggetto della classe, che consente di aggiungere una pagina *HTML<sub>G</sub>* associata al componente;
- + `void render():`  
metodo ridefinito da `Backbone.View`, che consente di aggiungere alla pagina *HTML<sub>G</sub>* il *template* campo dati della classe. Utilizza le classi `com.sirius.sequenziatore.client.presenter.user.SendText`, `com.sirius.sequenziatore.client.presenter.user.SendNumb`, `com.sirius.sequenziatore.client.presenter.user.SendImage` e `com.sirius.sequenziatore.client.presenter.user.SendPosition` per renderizzare l'interfaccia relativa all'inserimento dei diversi tipi di dato;

- + `bool` `getData()`:  
controlla se i dati inseriti dall'utente sono corretti: se lo sono ritorna `true` e li aggiunge alla collezione `processData`, altrimenti ritorna `false`;
- + `bool` `saveData()`:  
utilizza metodi del campo dati `processData`, per inviare i dati raccolti al *server*<sub>G</sub>.


Figura 10: Diagramma classe *SendText*

#### 4.1.2.8 SendText

- **Descrizione:** Classe che permette l'inserimento e il controllo di dati testuali inseriti dagli utenti;

- **Relazioni con altri componenti:**

La classe, mediante l'interfaccia `com.sirius.sequenziatore.client.view.user.ISendText`, realizza e aggiorna l'opportuno *widget*.

- **Attributi:**

- + `Object` `template`:  
oggetto ridefinito da `Backbone.View`, che contiene il *template* `HTMLG` associato alla classe;
- + `Object` `el`:  
oggetto ridefinito da `Backbone.View` che rappresenta l'elemento `HTMLG` entro cui la classe ascolta eventi generati dagli utenti;
- + `String` `id`:  
campo dati ridefinito da `Backbone.View` contenente l'id della classe;

- **Metodi:**

- + void `initialize()`:  
metodo ridefinito da `Backbone.View`, invocato alla costruzione di ciascun oggetto della classe, che consente di aggiungere una pagina *HTML<sub>G</sub>* associata al componente;
- + void `render()`:  
metodo ridefinito da `Backbone.View`, che consente di aggiungere alla pagina *HTML<sub>G</sub>* il *template* campo dati della classe;
- + bool `getData(ProcessDataModel data)`:  
controlla se i dati inseriti dall'utente sono corretti: se lo sono ritorna `true` e li aggiunge al riferimento `data`, altrimenti ritorna `false`.

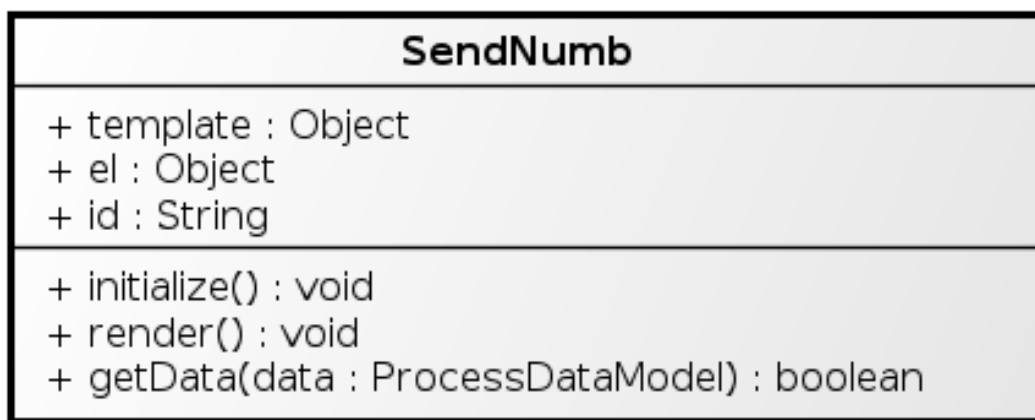


Figura 11: Diagramma classe *SendNumb*

#### 4.1.2.9 SendNumb

- **Descrizione:** Classe che ha il compito di permettere l'inserimento e il controllo di dati numerici inseriti dagli utenti;
- **Relazioni con altri componenti:**  
La classe, mediante l'interfaccia `com.sirius.sequenziatore.client.view.user.ISendNumb`, realizza e aggiorna l'opportuno *widget*.
- **Attributi:**
  - + Object `template`:

oggetto ridefinito da `Backbone.View`, che contiene il *template* `HTMLG` associato alla classe;

- + `Object el`:  
oggetto ridefinito da `Backbone.View` che rappresenta l'elemento `HTMLG` entro cui la classe ascolta eventi generati dagli utenti;
- + `String id`:  
campo dati ridefinito da `Backbone.View` contenente l'id della classe;

#### • Metodi:

- + `void initialize()`:  
metodo ridefinito da `Backbone.View`, invocato alla costruzione di ciascun oggetto della classe, che consente di aggiungere una pagina `HTMLG` associata al componente;
- + `void render()`:  
metodo ridefinito da `Backbone.View`, che consente di aggiungere alla pagina `HTMLG` il *template* campo dati della classe;
- + `bool getData(ProcessDataModel data)`:  
controlla se i dati inseriti dall'utente sono corretti: se lo sono ritorna `true` e li aggiunge al riferimento `data`, altrimenti ritorna `false`.

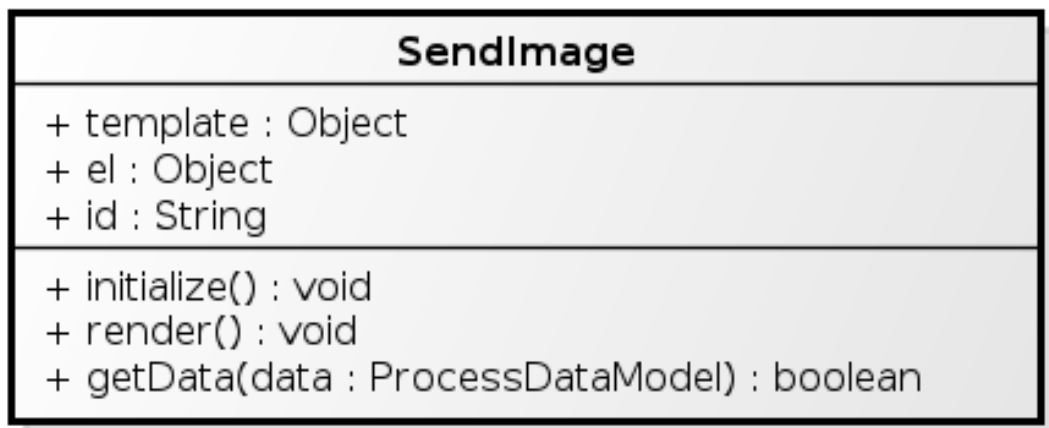


Figura 12: Diagramma classe *SendImage*

#### 4.1.2.10 SendImage

- **Descrizione:** Classe che gestisce l'inserimento e il controllo di immagini inserite dagli utenti;



- **Relazioni con altri componenti:**

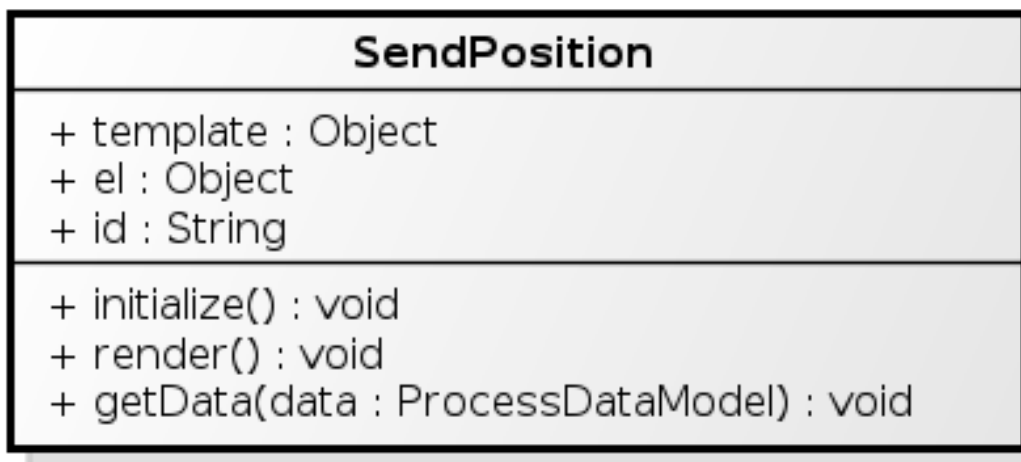
La classe, mediante l'interfaccia `com.sirius.sequenziatore.client.view.user.ISendImage`, realizza e aggiorna l'opportuno *widget*.

- **Attributi:**

- + `Object template`:  
oggetto ridefinito da `Backbone.View`, che contiene il *template HTML<sub>G</sub>* associato alla classe;
- + `Object el`:  
oggetto ridefinito da `Backbone.View` che rappresenta l'elemento *HTML<sub>G</sub>* entro cui la classe ascolta eventi generati dagli utenti;
- + `String id`:  
campo dati ridefinito da `Backbone.View` contenente l'id della classe;

- **Metodi:**

- + `void initialize()`:  
metodo ridefinito da `Backbone.View`, invocato alla costruzione di ciascun oggetto della classe, che consente di aggiungere una pagina *HTML<sub>G</sub>* associata al componente;
- + `void render()`:  
metodo ridefinito da `Backbone.View`, che consente di aggiungere alla pagina *HTML<sub>G</sub>* il *template* campo dati della classe;
- + `bool getData(ProcessDataModel data)`:  
controlla se i dati inseriti dall'utente sono corretti: se lo sono ritorna `true` e li aggiunge al riferimento `data`, altrimenti ritorna `false`.


Figura 13: Diagramma classe *SendPosition*

#### 4.1.2.11 SendPosition

- **Descrizione:** Classe che ha il compito di gestire il calcolo e il controllo della posizione geografica dell'utente;

- **Relazioni con altri componenti:**

La classe, mediante l'interfaccia `com.sirius.sequenziatore.client.view.user.ISendPosition`, realizza e aggiorna l'opportuno *widget*.

- **Attributi:**

- + Object template:  
oggetto ridefinito da `Backbone.View`, che contiene il *template HTML<sub>G</sub>* associato alla classe;
- + Object el:  
oggetto ridefinito da `Backbone.View` che rappresenta l'elemento *HTML<sub>G</sub>* entro cui la classe ascolta eventi generati dagli utenti;
- + String id:  
campo dati ridefinito da `Backbone.View` contenente l'id della classe;

- **Metodi:**

- + void initialize():  
metodo ridefinito da `Backbone.View`, invocato alla costruzione di ciascun oggetto della classe, che consente di aggiungere una pagina *HTML<sub>G</sub>* associata al componente;

- + void render():  
metodo ridefinito da `Backbone.View`, che consente di aggiungere alla pagina *HTML<sub>G</sub>* il *template* campo dati della classe;
- + bool getData(`ProcessDataModel data`):  
controlla se i dati inseriti dall'utente sono corretti: se lo sono ritorna `true` e li aggiunge al riferimento `data`, altrimenti ritorna `false`.

#### 4.1.3 Package `com.sirius.sequenziatore.client.presenter.processowner`

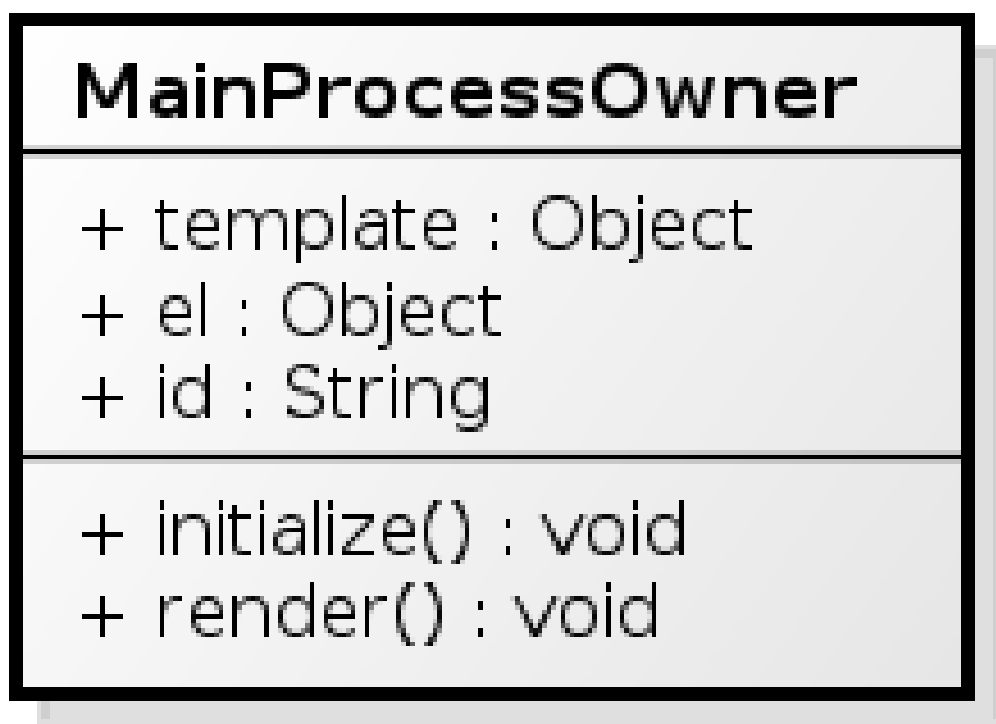


Figura 14: Diagramma classe *MainProcessOwner*

##### 4.1.3.1 MainProcessOwner

- **Descrizione:** Classe che ha il compito della gestione generale della logica delle funzionalità *Process Owner<sub>G</sub>*;

- **Relazioni con altri componenti:**

La classe comunica con il *template* `com.sirius.sequenziatore.client.view.processowner.IMainProcessOwner` per la realizzazione dell'interfaccia grafica.

- **Attributi:**

- + `Object` `template`:  
oggetto ridefinito da `Backbone.View`, che contiene il *template* `HTMLG` associato alla classe;
- + `Object` `el`:  
oggetto ridefinito da `Backbone.View` che rappresenta l'elemento `HTMLG` entro cui la classe ascolta eventi generati dagli utenti;
- + `String` `id`:  
campo dati ridefinito da `Backbone.View` contenente l'id della classe;

• **Metodi:**

- + `void` `initialize()`:  
metodo ridefinito da `Backbone.View`, invocato alla costruzione di ciascun oggetto della classe, che consente di aggiungere una pagina `HTMLG` associata al componente;
- + `void` `render()`:  
metodo ridefinito da `Backbone.View`, che consente di aggiungere alla pagina `HTMLG` il *template* campo dati della classe.

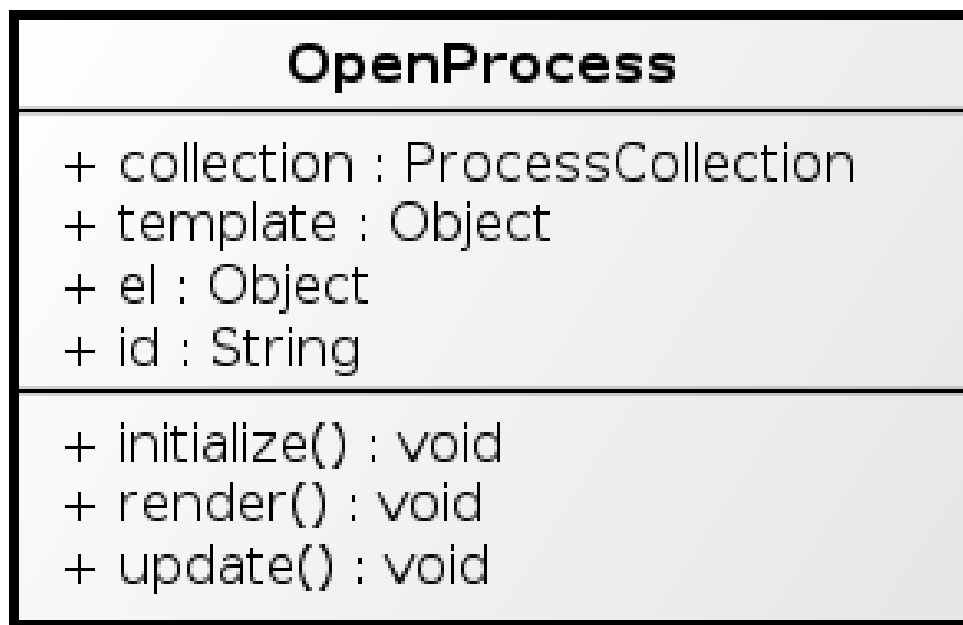


Figura 15: Diagramma classe *OpenProcess*

#### 4.1.3.2 OpenProcess

- **Descrizione:** Classe che ha il compito di gestire la ricerca e la selezione di un processo;

- **Relazioni con altri componenti:**

La classe comunica con il *template*

`com.sirius.sequenziatore.client.view.processowner.IOpenProcess` per la realizzazione dell'interfaccia grafica, e con la classe

`com.sirius.sequenziatore.client.model.collectionProcessCollection` per gestire e ottenere i dati dal *server<sub>G</sub>*.

- **Attributi:**

- + `ProcessCollection collection`:

campo dati di tipo `com.sirius.sequenziatore.client.model.collection.ProcessCollection` che contiene la lista dei processi non eliminati dal *process owner<sub>G</sub>*;

- + `Object template`:

oggetto ridefinito da `Backbone.View`, che contiene il *template HTML<sub>G</sub>* associato alla classe;

- + `Object el`:

oggetto ridefinito da `Backbone.View` che rappresenta l'elemento *HTML<sub>G</sub>* entro cui la classe ascolta eventi generati dagli utenti;

- + `String id`:

campo dati ridefinito da `Backbone.View` contenente l'id della classe;

- **Metodi:**

- + `void initialize()`:

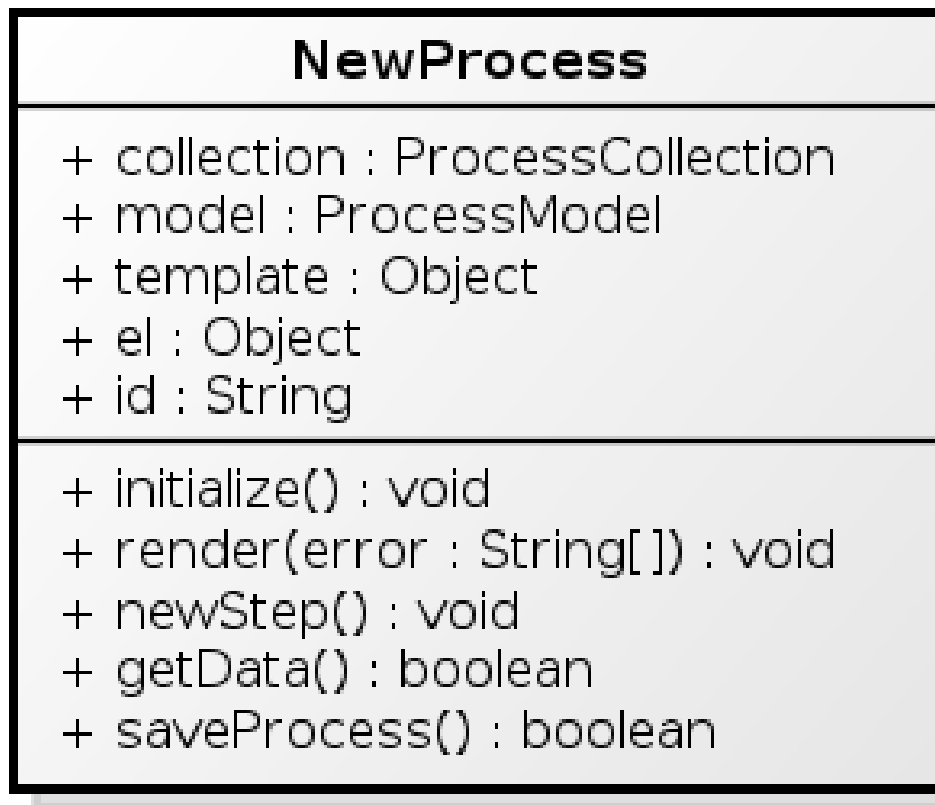
metodo ridefinito da `Backbone.View`, invocato alla costruzione di ciascun oggetto della classe, che consente di aggiungere una pagina *HTML<sub>G</sub>* associata al componente;

- + `void render()`:

metodo ridefinito da `Backbone.View`, che consente di aggiungere alla pagina *HTML<sub>G</sub>* il *template* campo dati della classe;

- + `void update()`:

aggiorna il campo dati `collection` comunicando con il *server<sub>G</sub>*.


Figura 16: Diagramma classe *NewProcess*

#### 4.1.3.3 NewProcess

- **Descrizione:** Classe che ha il compito di gestire la logica della definizione di un nuovo processo;

- **Relazioni con altri componenti:**

La classe comunica con il *template*

`com.sirius.sequenziatore.client.view.processowner.INewprocess` per la realizzazione dell'interfaccia grafica, con la classe

`com.sirius.sequenziatore.client.model.collection.ProcessCollection`

comunicare con il *server<sub>G</sub>* e con la classe

`com.sirius.sequenziatore.client.presenter.processowner.AddStep;`

- **Attributi:**

– + `ProcessCollection collection;`

campo dati di tipo `com.sirius.sequenziatore.client.model.collection.ProcessCollection` che consente di interagire con la lista dei processi non eliminati dal *process owner<sub>G</sub>*, presente nel *server<sub>G</sub>*;

– + `ProcessModel model`:

campo dati di tipo

`com.sirius.sequenziatore.client.model.ProcessModel` che contiene i dati del processo in definizione;

– + `Object template`:

oggetto ridefinito da `Backbone.View`, che contiene il *template HTML<sub>G</sub>* associato alla classe;

– + `Object el`:

oggetto ridefinito da `Backbone.View` che rappresenta l'elemento *HTML<sub>G</sub>* entro cui la classe ascolta eventi generati dagli utenti;

– + `String id`:

campo dati ridefinito da `Backbone.View` contenente l'id della classe;

• **Metodi:**

– + `void initialize()`:

metodo ridefinito da `Backbone.View`, invocato alla costruzione di ciascun oggetto della classe, che consente di aggiungere una pagina *HTML<sub>G</sub>* associata al componente;

– + `void render(String[] errors)`:

metodo ridefinito da `Backbone.View`, che consente di aggiungere alla pagina *HTML<sub>G</sub>* il *template* campo dati della classe, compilato con gli eventuali errori `errors`;

– + `void newStep()`:

utilizza la classe

`com.sirius.sequenziatore.client.presenter.processowner.AddStep` per definire e aggiungere un nuovo passo al processo `model`;

– + `bool getData()`:

controlla se i dati inseriti dal *process owner<sub>G</sub>* sono corretti: se lo sono ritorna `true` e li aggiunge al processo `model`, altrimenti ritorna `false`;

– + `bool saveProcess()`:

utilizza metodi del campo dati `collection`, per inviare il processo `model` al *server<sub>G</sub>*.

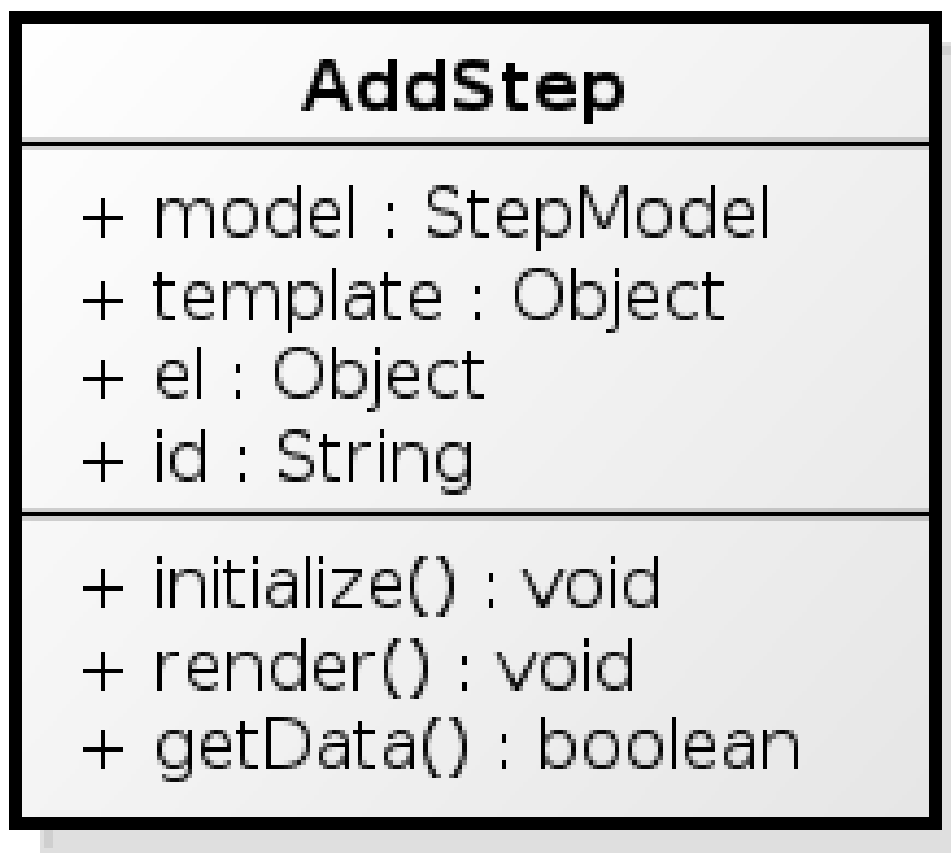


Figura 17: Diagramma classe *AddStep*

#### 4.1.3.4 AddStep

- **Descrizione:** Classe che ha il compito di gestire la logica di definizione dei passi di un processo;

- **Relazioni con altri componenti:**

La classe comunica con il *template*

`com.sirius.sequenziatore.client.view.processowner.IAddStep` per la realizzazione dell'interfaccia grafica e utilizza la classe

`com.sirius.sequenziatore.client.model.Step` per salvare i dati del passo in creazione.

- **Attributi:**

– + `StepModel model`:



campo dati di tipo

`com.sirius.sequenziatore.client.model.StepModel` che contiene i dati del passo in definizione;

– + `Object template`:

oggetto ridefinito da `Backbone.View`, che contiene il *template* `HTMLG` associato alla classe;

– + `Object el`:

oggetto ridefinito da `Backbone.View` che rappresenta l'elemento `HTMLG` entro cui la classe ascolta eventi generati dagli utenti;

– + `String id`:

campo dati ridefinito da `Backbone.View` contenente l'id della classe;

- **Metodi:**

– + `void initialize()`:

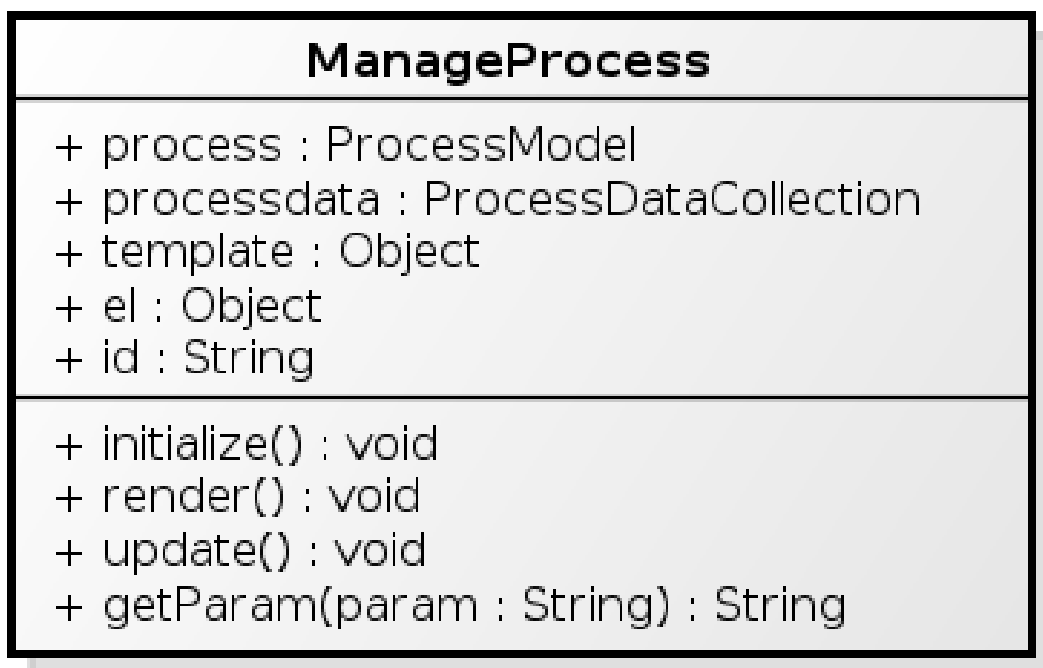
metodo ridefinito da `Backbone.View`, invocato alla costruzione di ciascun oggetto della classe, che consente di aggiungere una pagina `HTMLG` associata al componente;

– + `void render(String[] errors)`:

metodo ridefinito da `Backbone.View`, che consente di aggiungere alla pagina `HTMLG` il *template* campo dati della classe, compilato con gli eventuali errori `errors`;

– + `bool getData()`:

controlla se i dati inseriti dal *process owner<sub>G</sub>* sono corretti: se lo sono ritorna `true` e li aggiunge al passo `model`, altrimenti ritorna `false`.


Figura 18: Diagramma classe *ManageProcess*

#### 4.1.3.5 ManageProcess

- **Descrizione:** Classe che ha il compito di gestire e accedere alle informazioni relative allo stato dei processi e ai dati inviati dagli utenti. Le operazioni di gestione dello stato comprendono la terminazione e l'eliminazione di un processo;

- **Relazioni con altri componenti:**

La classe comunica con il *template*

`com.sirius.sequenziatore.client.view.processowner.IManageProcess` per la realizzazione dell'interfaccia grafica, e con le classi `com.sirius.sequenziatore.client.model.collection.ProcessDataCollection` e `com.sirius.sequenziatore.client.model.ProcessModel` per gestire e ottenere i dati dal *serverG*.

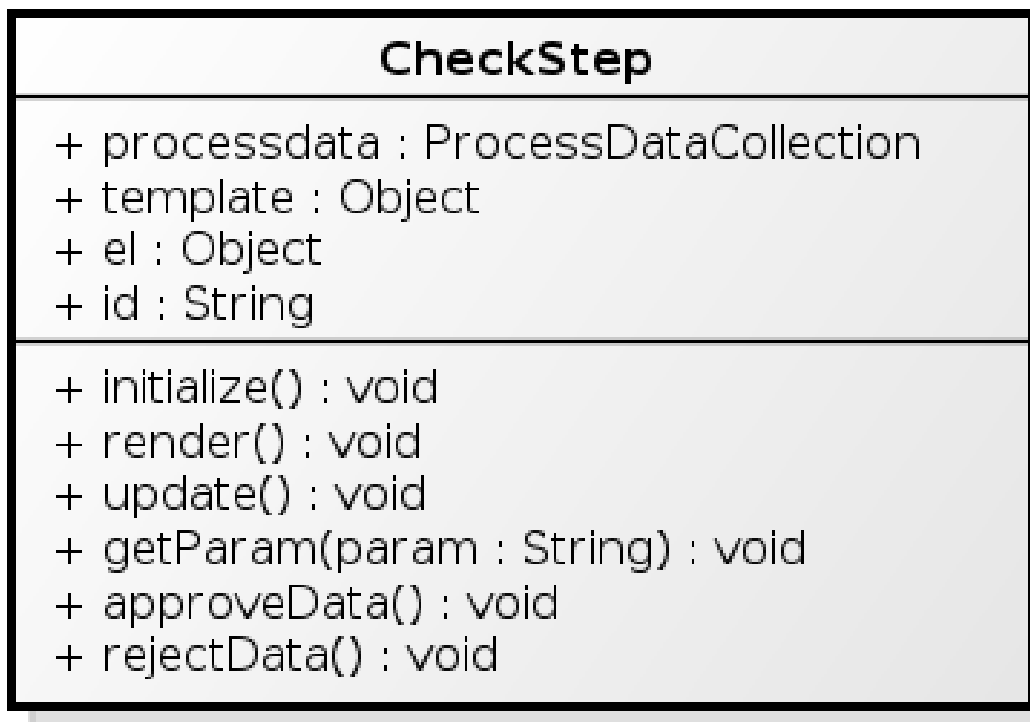
- **Attributi:**

- + `ProcessModel process`:  
campo dati di tipo  
`com.sirius.sequenziatore.client.model.ProcessModel` che contiene i  
dati del processo in gestione;

- + `ProcessDataCollection processdata`:  
campo dati di tipo `com.sirius.sequenziatore.client.model.collection.ProcessDataCollection` che contiene i dati inviati dagli utenti relativi al processo in gestione;
- + `Object template`:  
oggetto ridefinito da `Backbone.View`, che contiene il *template* `HTMLG` associato alla classe;
- + `Object el`:  
oggetto ridefinito da `Backbone.View` che rappresenta l'elemento `HTMLG` entro cui la classe ascolta eventi generati dagli utenti;
- + `String id`:  
campo dati ridefinito da `Backbone.View` contenente l'id della classe;

- **Metodi:**

- + `void initialize()`:  
metodo ridefinito da `Backbone.View`, invocato alla costruzione di ciascun oggetto della classe, che consente di aggiungere una pagina `HTMLG` associata al componente;
- + `void render()`:  
metodo ridefinito da `Backbone.View`, che consente di aggiungere alla pagina `HTMLG` il *template* campo dati della classe;
- + `void update()`:  
aggiorna i campi dati `process` e `processData` comunicando con il *serverG*;
- + `String getParam(String param)`:  
ritorna il valore del parametro *param* se presente nella *URLG*;


Figura 19: Diagramma classe *CheckStep*

#### 4.1.3.6 CheckStep

- **Descrizione:** Classe che ha il compito di definire la logica del controllo di un passo che richiede intervento umano per essere approvato;

- **Relazioni con altri componenti:**

La classe comunica con il *template*

`com.sirius.sequenziatore.client.view.processowner.ICheckStep` per la realizzazione dell'interfaccia grafica, e con le classi `com.sirius.sequenziatore.client.model.collection.ProcessDataCollection` e

`com.sirius.sequenziatore.client.model.ProcessModel` per gestire e ottenere i dati dal *serverG*.

- **Attributi:**

- + `ProcessDataCollection processdata:`  
campo dati di tipo `com.sirius.sequenziatore.client.model.collection.ProcessDataCollection` che contiene i dati inviati dagli utenti in attesa di approvazione;
- + `Object template:`  
oggetto ridefinito da `Backbone.View`, che contiene il *template HTMLG* associato alla classe;

- + `Object el`:  
oggetto ridefinito da `Backbone.View` che rappresenta l'elemento *HTML<sub>G</sub>* entro cui la classe ascolta eventi generati dagli utenti;
- + `String id`:  
campo dati ridefinito da `Backbone.View` contenente l'id della classe;

- **Metodi:**

- + `void initialize()`:  
metodo ridefinito da `Backbone.View`, invocato alla costruzione di ciascun oggetto della classe, che consente di aggiungere una pagina *HTML<sub>G</sub>* associata al componente;
- + `void render()`:  
metodo ridefinito da `Backbone.View`, che consente di aggiungere alla pagina *HTML<sub>G</sub>* il *template* campo dati della classe;
- + `void update()`:  
aggiorna il campo dati `processData` comunicando con il *server<sub>G</sub>*;
- + `String getParam(String param)`:  
ritorna il valore del parametro *param* se presente nella *URL<sub>G</sub>*;
- + `void approveData()`:  
salva nel *server* lo stato approvato ai dati della collezione *processData* dei quali il *process owner<sub>G</sub>* ha richiesto l'approvazione;
- + `void rejectData()`:  
salva nel *server* lo stato approvato ai dati della collezione *processData* che il *process owner<sub>G</sub>* ha respinto;

## 4.2 Server

Questa componente è incaricata di gestire la comunicazione con il client e di elaborarne le richieste restituendo i dati richiesti e quando necessario interroga la componente model per ottenere i dati dal database. Tale componente è composta dalle classi:

- `com.sirius.sequenziatore.server.presenter.common.SignUpController`
- `com.sirius.sequenziatore.server.presenter.common.LoginController`
- `com.sirius.sequenziatore.server.presenter.common.StepInfoController`
- `com.sirius.sequenziatore.server.presenter.common.ProcessInfoController`
- `com.sirius.sequenziatore.server.presenter.processowner.StepController`
- `com.sirius.sequenziatore.server.presenter.processowner.ProcessController`
- `com.sirius.sequenziatore.server.presenter.processowner.ApproveStepController`
- `com.sirius.sequenziatore.server.presenter.user.AccountController`
- `com.sirius.sequenziatore.server.presenter.user.UserStepController`
- `com.sirius.sequenziatore.server.presenter.user.UserProcessController`
- `com.sirius.sequenziatore.server.presenter.user.ReportController`

Nella prossime sessioni verranno trattate in dettaglio le seguenti classi dividendo l'esposizione per *package*, si evidenzia come la voce mappatura base sia l'estensione della mappatura su cui si programma il sistema che sarà `localhost:8080/sequenziatore/`, quindi tutte le mappature base saranno da considerarsi come aggiunte a seguito di `/sequenziatore/` e successivamente le varie varianti dei metodi. Tutte le classi *controller* del *presenter* dovranno essere marcate come `@Controller` per essere riconosciute in modo corretto da *Spring*.

#### 4.2.1 Package com.sirius.sequenziatore.server.presenter.common

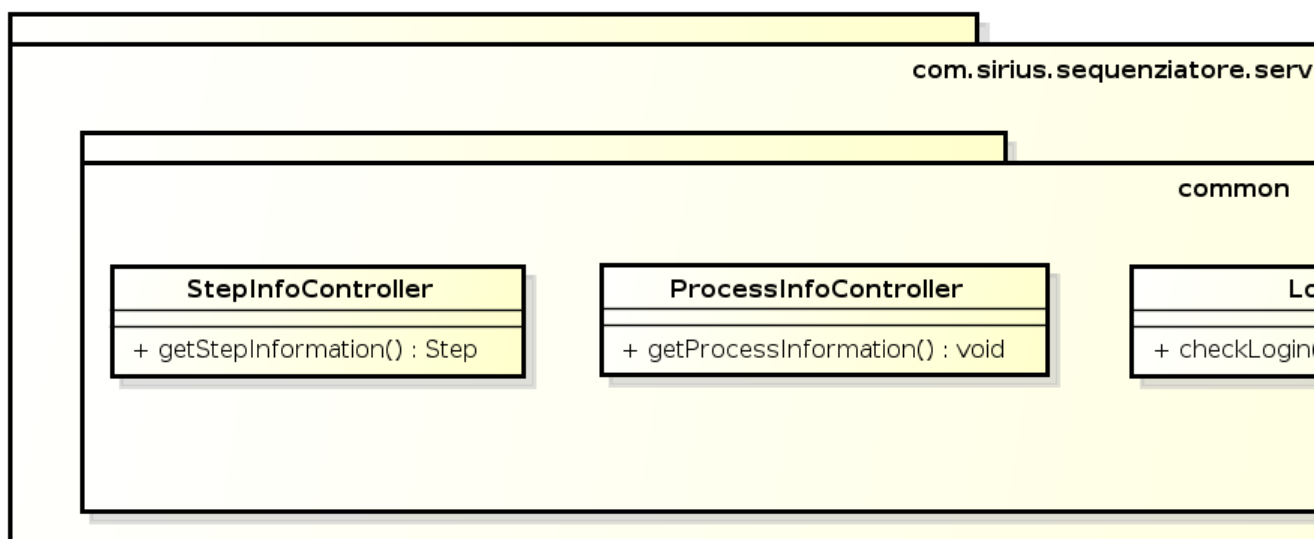


Figura 20: Diagramma package - `com.sirius.sequenziatore.server.presenter.common`

All' interno di questa sezione verranno trattate tutte le classi contenute nel package *common*.

##### 4.2.1.1 SignUpController



Figura 21: Diagramma classe - `SignUpController`

- **Descrizione:** Questa classe dovrà gestire tutte le richieste di registrazione al sistema, sarà incaricata di inserire i dati nel database e di avvertire il client della riuscita della registrazione.
- **Mappatura base:** `\signup`
- **Relazioni con altri componenti:** La classe utilizzerà le seguenti classi:
  - `com.sirius.sequenziatore.server.model.User`;
  - `com.sirius.sequenziatore.server.model.UserDao`;
 tramite le interfacce:
  - `com.sirius.sequenziatore.server.model.ITransferObject`;

```
– com.sirius.sequenziatore.server.model.IDataAccessObject;
```

- **Metodi:**

```
– +registerUser(User toBeRegistered):
```

questo metodo gestirà un metodo **POST** e restituirà dovrà lanciare un' eccezione di tipo **HttpError** qual' ora ci siano stati problemi nella registrazione;

#### 4.2.1.2 LoginController



Figura 22: Diagramma classe - LoginController

- **Descrizione:** Questa classe gestirà le richieste di *log in*, dovrà controllare se l' utente esiste nel sistema e se le credenziali d' accesso sono corrette restituendo il tipo di utente altrimenti un errore se l' utente non esiste nel sistema;

- **Mappatura base:** *\login*

- **Relazioni con altri componenti:** La classe utilizzerà le seguenti classi:

```
– com.sirius.sequenziatore.server.model.User;
– com.sirius.sequenziatore.server.model.UserDao;
```

tramite le interfacce:

```
– com.sirius.sequenziatore.server.model.ITransferObject;
– com.sirius.sequenziatore.server.model.IDataAccessObject;
```

- **Metodi:**

```
– +checkLogin(User toBeLogged):
```

questo metodo gestirà un metodo di tipo **POST** , controllerà le credenziali di accesso e dovrà lanciare un' eccezione di tipo **HttpError** qualora ci siano stati problemi nella login;





Figura 23: Diagramma classe - StepInfoController

#### 4.2.1.3 StepInfoController

- **Descrizione:** Questa classe restituirà lo scheletro, quindi la composizione del passo richiesto;
- **Mappatura base:** `\step\{id}`
- **Relazioni con altri componenti:** La classe utilizzerà le seguenti classi:
  - `com.sirius.sequenziatore.server.model.Step;`
  - `com.sirius.sequenziatore.server.model.StepDao;`
 tramite le interfacce:
  - `com.sirius.sequenziatore.server.model.ITransferObject;`
  - `com.sirius.sequenziatore.server.model.IDataAccessObject;`
- **Metodi:**
  - `+Step getStepInformation():`  
il metodo gestisce una richiesta di tipo **GET** restituendo la struttura del passo con id uguale all' id fornito dopo averla recuperata dal *database*;



Figura 24: Diagramma classe - ProcessInfoController

#### 4.2.1.4 ProcessInfoController

- **Descrizione:** Questa classe dovrà restituire a chi lo richiede un processo dato l' *id* con i suoi dati;
- **Mappatura base:** `\process\{id}`
- **Relazioni con altri componenti:** La classe utilizzerà le seguenti classi:

- `com.sirius.sequenziatore.server.model.Process;`
- `com.sirius.sequenziatore.server.model.ProcessDao;`

tramite le interfacce:

- `com.sirius.sequenziatore.server.model.ITransferObject;`
- `com.sirius.sequenziatore.server.model.IDataAccessObject;`

#### • Metodi:

- `+Process getProcessInformation():`  
il metodo gestisce una richiesta di tipo **GET** e restituisce la struttura di un processo con id processo richiesto;

### 4.2.2 Package `com.sirius.sequenziatore.server.presenter.processowner`

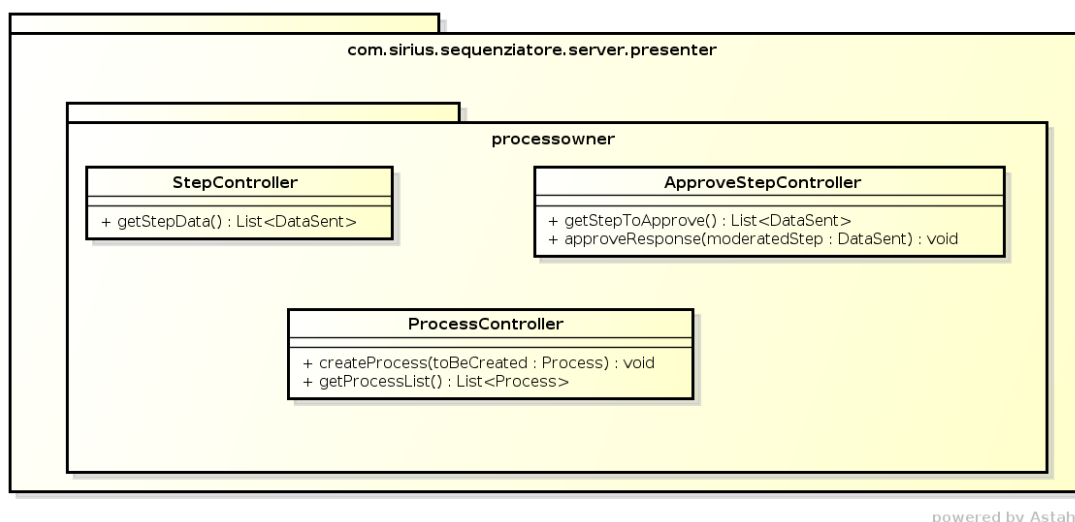


Figura 25: Diagramma package - `com.sirius.sequenziatore.server.presenter.processowner`

#### 4.2.2.1 StepController



Figura 26: Diagramma classe - `StepController`

- **Descrizione:** Questa classe dovrà fornire al *process owner* tutti i dati inseriti dagli utenti per un dato passo, quindi dovrà restituire una collezione di dati al process owner il quale potrà visionarli;

- **Mappatura base:** `\stepdata\{idstep}\processowner`
- **Relazioni con altri componenti:** La classe utilizzerà le seguenti classi:

- `com.sirius.sequenziatore.server.model.DataSent;`
- `com.sirius.sequenziatore.server.model.StepDao;`

tramite le interfacce:

- `com.sirius.sequenziatore.server.model.ITransferObject;`
- `com.sirius.sequenziatore.server.model.IDataAccessObject;`

- **Metodi:**

- `+List<StepData> getStepData():`  
questo metodo gestisce una richiesta di tipo **GET** che fornisce al *process owner* tutti i dati inviati dagli utenti per un certo passo;

#### 4.2.2.2 ProcessController

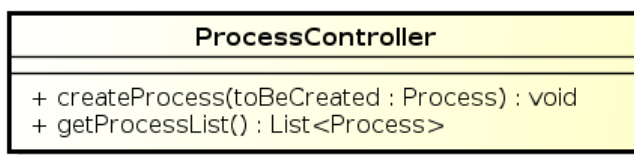


Figura 27: Diagramma classe - ProcessController

- **Descrizione:** Questa classe permetterà la creazione di un processo da parte del *process owner* e sarà adibita a fornire la lista di tutti i processi esistenti nel sistema;
- **Mappatura base:** `\process\processowner`
- **Relazioni con altri componenti:** La classe utilizzerà le seguenti classi:

- `com.sirius.sequenziatore.server.model.Process;`
- `com.sirius.sequenziatore.server.model.ProcessDao;`

tramite le interfacce:

- `com.sirius.sequenziatore.server.model.ITransferObject;`
- `com.sirius.sequenziatore.server.model.IDataAccessObject;`

- **Metodi:**

- `+void createProcess(Process toBeCreated):`  
questo metodo gestisce una richiesta di tipo **POST** e permette l' inserimento del processo fornito nel *database*;
- `+List<Process> getProcessList():`  
questo metodo gestisce una richiesta di tipo **GET** e restituisce al *process owner* una lista di processi che può visualizzare;

#### 4.2.2.3 ApproveStepController



Figura 28: Diagramma classe - ApproveStepController

- **Descrizione:** Questa classe serve per fornire al *process owner* i dati da approvare e per gestire quali passi siano stati approvati quali no, qualora un passo non venga approvato, verrà rimosso dal *database*;
- **Mappatura base:** `\approvedata`
- **Relazioni con altri componenti:** La classe utilizzerà le seguenti classi:

- `com.sirius.sequenziatore.server.model.DataSent;`
- `com.sirius.sequenziatore.server.model.StepDao;`

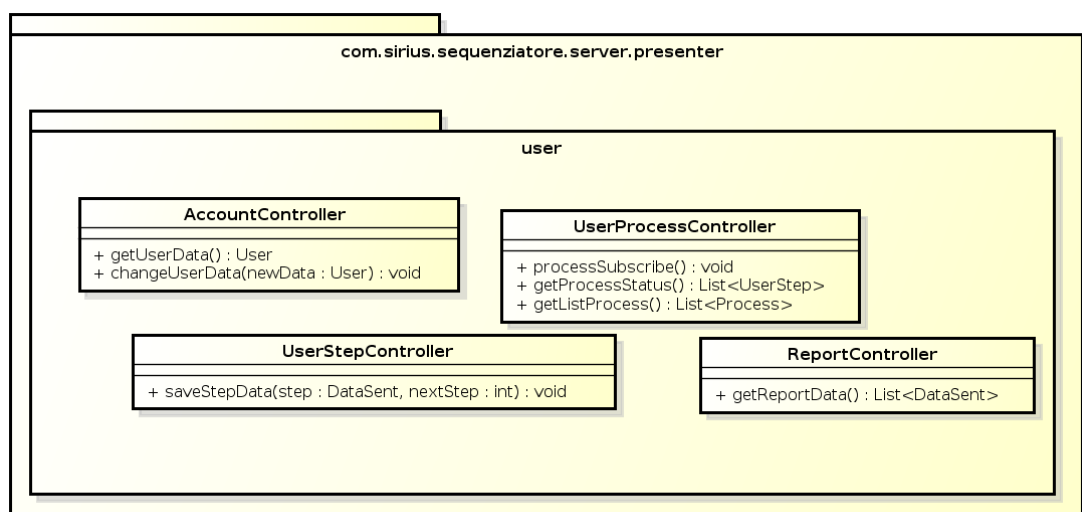
tramite le interfacce:

- `com.sirius.sequenziatore.server.model.ITransferObject;`
- `com.sirius.sequenziatore.server.model.IDataAccessObject;`

- **Metodi:**

- `+List<DataSent> getStepToApprove():`  
il metodo gestisce una richiesta di tipo *GET*, e restituirà un oggetto di tipo `List<DataSent>` contenente tutti i dati che richiedono approvazione;
- `+void approveResponse(int idStep,String usernameww):`  
il metodo gestisce una richiesta di tipo *POST*, riceve i dati di un passo che ha subito la moderazione del *process owner*, tale passo verrà eliminato dal database se il processowner lo ha rifiutato altrimenti verrà approvato definitivamente;

### 4.2.3 Package com.sirius.sequenziatore.server.presenter.user



powered by Astah

Figura 29: Diagramma package - com.sirius.sequenziatore.server.presenter.user

#### 4.2.3.1 UserStepController



Figura 30: Diagramma classe - UserStepController

- **Descrizione:** Questa classe gestisce la ricezione dei dati di un passo inviati da un utente tramite una richiesta di tipo *POST*, tale passo dovrà essere inserito nel database, ponendo attenzione se è un passo che richiede approvazione o meno;
- **Mappatura base:** `\stepdata\user`
- **Relazioni con altri componenti:** La classe utilizzerà le seguenti classi:

- com.sirius.sequenziatore.server.model.DataSent;
- com.sirius.sequenziatore.server.model.UserStep;
- com.sirius.sequenziatore.server.model.StepDao;

tramite le interfacce:

- com.sirius.sequenziatore.server.model.ITransferObject;
- com.sirius.sequenziatore.server.model.IDataAccessObject;

- **Metodi:**

- `+void saveStepData(DataSent step,int nextStep):`  
questo metodo gestisce una richiesta **POST** da un utente, riceve i dati inerenti a un passo e li inserisce nel database e se non necessita di approvazione lo segna come completato e modifica quale sarà il passo o i passi che si potranno eseguire;

#### 4.2.3.2 UserProcessController

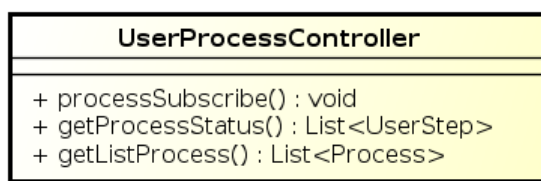


Figura 31: Diagramma classe - UserProcessController

- **Descrizione:** Questa classe permette all' utente varie operazioni, innanzitutto l' iscrizione ad un processo, poi restituisce il passo a cui è arrivato e il suo stato per tale processo e infine fornisce una lista di processi con tutti i processi a cui si può iscrivere e i processi per i quali può chiedere di fare il *report*;
- **Mappatura base:** `\user\{username}`
- **Relazioni con altri componenti:** La classe utilizzerà le seguenti classi:

- `com.sirius.sequenziatore.server.model.Process;`
- `com.sirius.sequenziatore.server.model.UserStep;`
- `com.sirius.sequenziatore.server.model.ProcessDao;`
- `com.sirius.sequenziatore.server.model.StepDao;`

tramite le interfacce:

- `com.sirius.sequenziatore.server.model.ITransferObject;`
- `com.sirius.sequenziatore.server.model.IDataAccessObject;`

- **Metodi:**

- `+void processSubscribe():`  
questo metodo mappa su `\subscribe\{processid}` e gestisce una richiesta di tipo **POST** che permette ad un utente di iscriversi al processo voluto;

- `+List<UserStep> getProcessStatus():`  
questo metodo mappa su `\subscribe\{processid}` e gestisce una richiesta **GET** che restituisce all'utente il proprio status per tale processo, restituendo il passo o i passi che può eseguire e quanti passi ha completato del processo;
- `+List<Process> getListProcess():`  
questo processo mappa su `\processlist` e gestisce una richiesta di tipo **GET** andando e restituire una lista di processi che contiene tutti i processi a cui è iscritto e quelli a cui si può iscrivere;

#### 4.2.3.3 AccountController

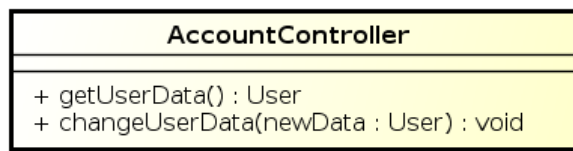


Figura 32: Diagramma classe - AccountController

- **Descrizione:** Classe che fornisce i dati di un utente e ne permette la modifica dei suddetti;
- **Mappatura base:** `\account\{username}`
- **Relazioni con altri componenti:** La classe utilizzerà le seguenti classi:

- `com.sirius.sequenziatore.server.model.User;`
- `com.sirius.sequenziatore.server.model.UserDao;`

tramite le interfacce:

- `com.sirius.sequenziatore.server.model.ITransferObject;`
- `com.sirius.sequenziatore.server.model.IDataAccessObject;`

- **Metodi:**

- `+User getUserData():`  
questo metodo gestisce una richiesta di tipo **GET** e restituisce un oggetto di tipo `User` contenente tutti i dati di un utente;
- `+void changeUserData(User newData):`  
questo metodo gestisce una chiamata di tipo **POST** e permette la modifica dei dati di un account di un utente;

#### 4.2.3.4 ReportController



Figura 33: Diagramma classe - ReportController

- **Descrizione:** Questa classe fornirà al client tutti i dati necessari per creare il report di un utente per un certo processo;
- **Mappatura base:** `\report\{username}\{processid}`
- **Relazioni con altri componenti:** La classe utilizzerà le seguenti classi:

- `com.sirius.sequenziatore.server.model.User;`
- `com.sirius.sequenziatore.server.model.Process;`
- `com.sirius.sequenziatore.server.model.Step;`
- `com.sirius.sequenziatore.server.model.UserDao;`
- `com.sirius.sequenziatore.server.model.ProcessDao;`
- `com.sirius.sequenziatore.server.model.StepDao;`

tramite le interfacce:

- `com.sirius.sequenziatore.server.model.ITransferObject;`
- `com.sirius.sequenziatore.server.model.IDataAccessObject;`

- **Metodi:**
  - `+List<DataSent> getReportData():`  
questo metodo gestisce una richiesta di tipo **GET** e fornirà tutti i dati inseriti da un utente per un certo processo;



## 5 Specifica della componente model

Questa componente consente di rappresentare i dati e gestire la loro persistenza, e viene suddivisa in due parti: *client* e *server*.

### 5.1 Client

Il *model* lato *client* consente di gestire i dati dell'applicazione e la comunicazione con il *server<sub>G</sub>*.

La componente è formata dalle seguenti *classi*:

- [com.sirius.sequenziatore.client.model.UserDataModel](#);
- [com.sirius.sequenziatore.client.model.ProcessModel](#);
- [com.sirius.sequenziatore.client.model.ProcessDataModel](#);
- [com.sirius.sequenziatore.client.model.StepModel](#);
- [com.sirius.sequenziatore.client.model.collection.ProcessCollection](#);
- [com.sirius.sequenziatore.client.model.ProcessDataCollection](#);
- [com.sirius.sequenziatore.client.model.collection.StepCollection](#).

#### 5.1.1 Package com.sirius.sequenziatore.client.model

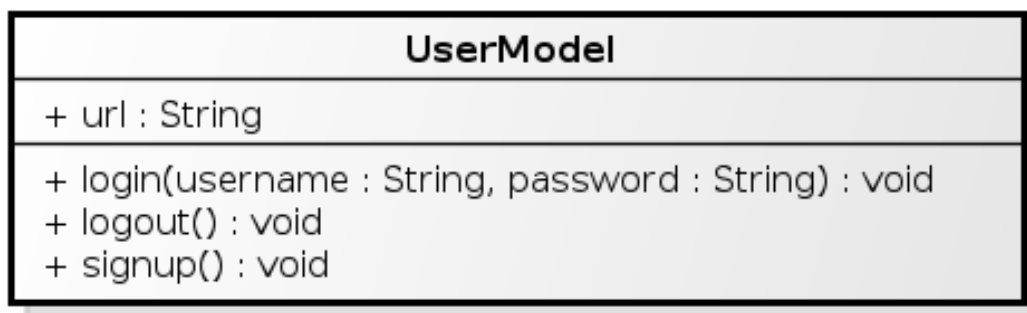


Figura 34: Diagramma classe *UserModel*

##### 5.1.1.1 UserModel

- **Descrizione:** Classe che permette di gestire i dati di una sessione di un utente autenticato o di un *Process Owner<sub>G</sub>*;
- **Attributi:**

- + `String url`:  
campo dati di ridefinito da `Backbone.Model` che contiene l'indirizzo `urlG` per comunicare con il `serverG`;

- **Metodi:**

- + `void login(String username, String password)`:  
delega al server il controllo delle credenziali e, al completamento della richiesta, salva i dati di sessione in caso di successo;
- + `void logout()`:  
cancella di dati di sessione dell'utente;
- + `void signup()`:  
effettua una richiesta di registrazione al `serverG` inviando i dati della classe.

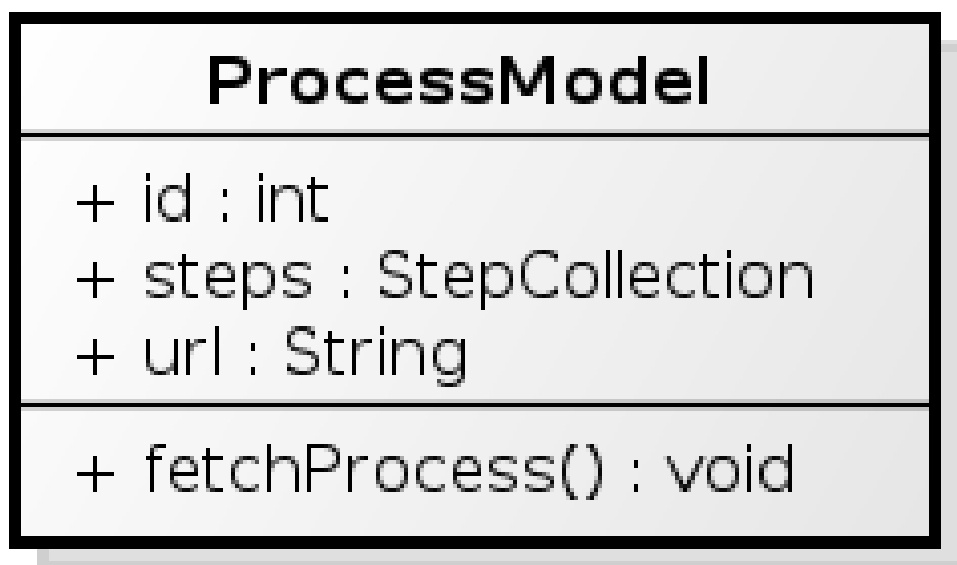


Figura 35: Diagramma classe *ProcessModel*

#### 5.1.1.2 ProcessModel

- **Descrizione:** Classe che permette di gestire i dati di un processo, e di salvarli o recuperarli dal `serverG`;
- **Relazioni con altri componenti:**  
La classe contiene un oggetto di tipo `com.sirius.sequenziatore.client.model.collection.StepCollection`.

- **Attributi:**

- + `int id`:  
campo dati ridefinito da `Backbone.model` che rappresenta l'identificatore del processo;
- + `StepCollection steps`:  
campo dati di tipo `com.sirius.sequenziatore.client.model.collection.StepCollection` che contiene la collezione dei passi del processo;
- + `String url`:  
campo dati di ridefinito da `Backbone.Model` che contiene l'indirizzo `urlG` per comunicare con il `serverG`;

- **Metodi:**

- + `void fetchProcess()`:  
recupera dal `serverG` i dati del processo, e i dati dei passi che assegna alla collezione `steps`, sincronizzando le operazioni.

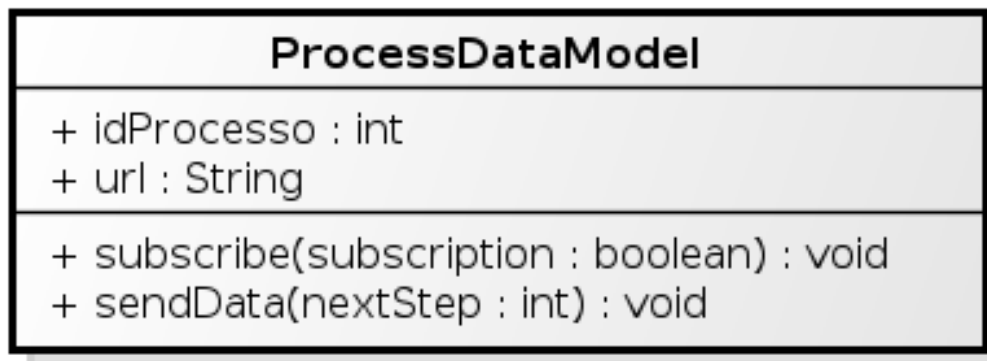


Figura 36: Diagramma classe *ProcessDataModel*

#### 5.1.1.3 ProcessDataModel

- **Descrizione:** Classe che permette di gestire i dati inviati da un utente relativi ad un processo, e di salvarli o recuperarli dal `serverG`;
- **Attributi:**
  - + `int idProcesso`:  
rappresenta l'identificatore del processo a cui i dati si riferiscono;

- + `String url`:  
campo dati di ridefinito da `Backbone.Model` che contiene l'indirizzo `urlG` per comunicare con il `serverG`;

- **Metodi:**

- + `void subscribe(bool subscription)`:  
effettua una richiesta di iscrizione o disiscrizione al `serverG` a seconda del valore del parametro `subscription`, riguardante il processo con id `idProcesso`;
- + `void sendData(int nextStep)`:  
invia al `serverG` i dati della classe e l'id del prossimo passo da eseguire, che identifica una condizione del processo con id `idProcesso`.

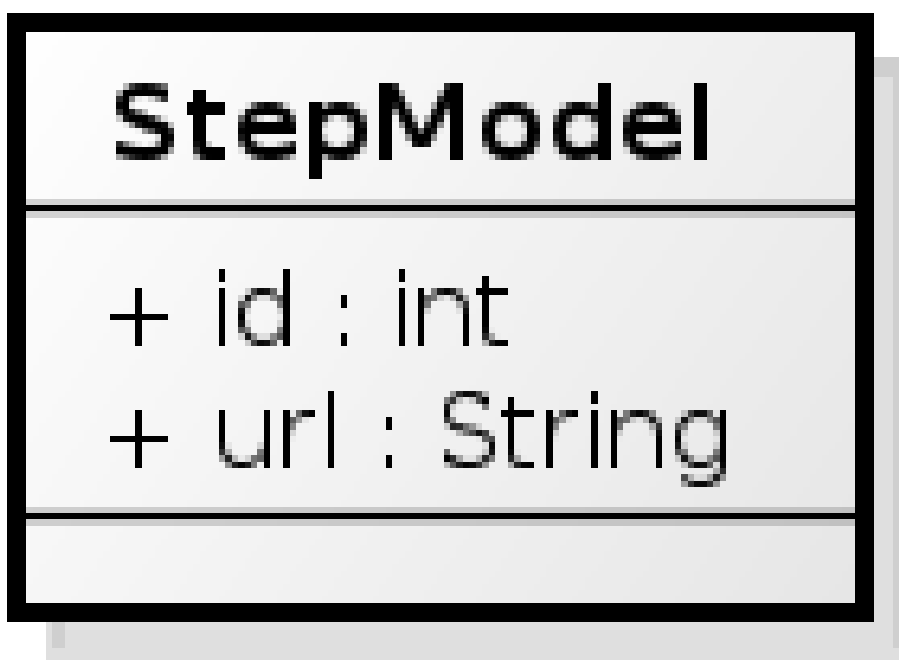


Figura 37: Diagramma classe *StepModel*

#### 5.1.1.4 StepModel

- **Descrizione:** Classe che permette di gestire i dati di un passo di un processo, e di salvarli o recuperarli dal `serverG`;

- **Attributi:**

- + `int id`:  
campo dati ridefinito da `Backbone.model` che rappresenta l'identificatore del passo;
- + `String url`:  
campo dati di ridefinito da `Backbone.Model` che contiene l'indirizzo `urlG` per comunicare con il `serverG`;

### 5.1.2 Package `com.sirius.sequenziatore.client.model.collection`

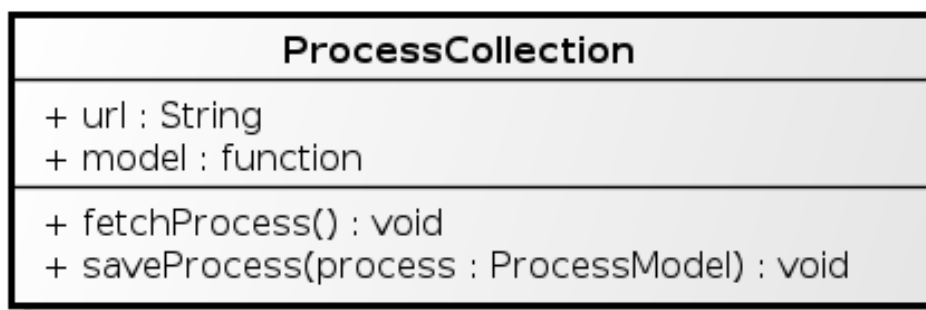


Figura 38: Diagramma classe *ProcessCollection*

#### 5.1.2.1 ProcessCollection

- **Descrizione:** Classe che permette di gestire un insieme di dati inviati da un utente relativi ad un processo;

- **Relazioni con altri componenti:**

La classe definisce una collezione di  
`com.sirius.sequenziatore.client.model.ProcessDataModel`.

- **Attributi:**

- + `String url`:  
campo dati di ridefinito da `Backbone.Collection` che contiene l'indirizzo `urlG` per comunicare con il `serverG`;
- + `function model`:  
campo dati di ridefinito da `Backbone.Collection` che contiene la definizione della classe  
`com.sirius.sequenziatore.client.model.ProcessDataModel`;

- **Metodi:**

- + void `fetchProcesses()`:  
richiede al server la lista dei processi a cui l'utente identificato dai dati di sessione può accedere;
- + void `saveProcess(ProcessModel process)`:  
aggiunge il processo `process` alla collezione dei processi nel `serveG`.

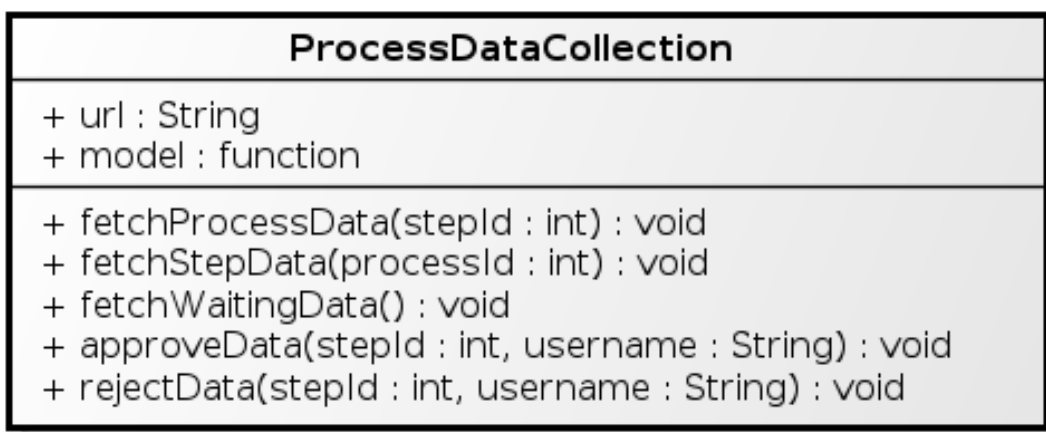


Figura 39: Diagramma classe *ProcessDataCollection*

#### 5.1.2.2 ProcessDataCollection

- **Descrizione:** Classe che permette di gestire un insieme di dati inviati dagli utenti;

- **Relazioni con altri componenti:**

La classe definisce una collezione di  
`com.sirius.sequenziatore.client.model.ProcessDataModel`.

- **Attributi:**

- + String `url`:  
campo dati di ridefinito da `Backbone.Collection` che contiene l'indirizzo `urlG` per comunicare con il `serverG`;
- + function `model`:  
campo dati di ridefinito da `Backbone.Collection` che contiene la definizione della classe  
`com.sirius.sequenziatore.client.model.ProcessDataModel`;

- **Metodi:**

- + void fetchProcessData(int stepId):  
richiede al *server<sub>G</sub>* la lista dei dati inviati riguardanti il passo con id *stepId*, ai quali l'utente identificato dai dati di sessione può accedere;
- + void fetchStepData(int processId):  
richiede al *server<sub>G</sub>* la lista dei dati inviati riguardanti il processo con id *processId*, ai quali l'utente identificato dai dati di sessione può accedere;
- + void fetchWaitingData():  
richiede al *server<sub>G</sub>* la lista dei dati inviati che richiedono controllo umano;
- + void approveData(int stepId, String username):  
invia al *server<sub>G</sub>* la richiesta di approvazione dei dati riguardanti il passo con id *stepId* e l'utente con username *username*.
- + void rejectData(int stepId, String username):  
invia al *server<sub>G</sub>* l'esito negativo del controllo dei dati riguardanti il passo con id *stepId* e l'utente con username *username*.

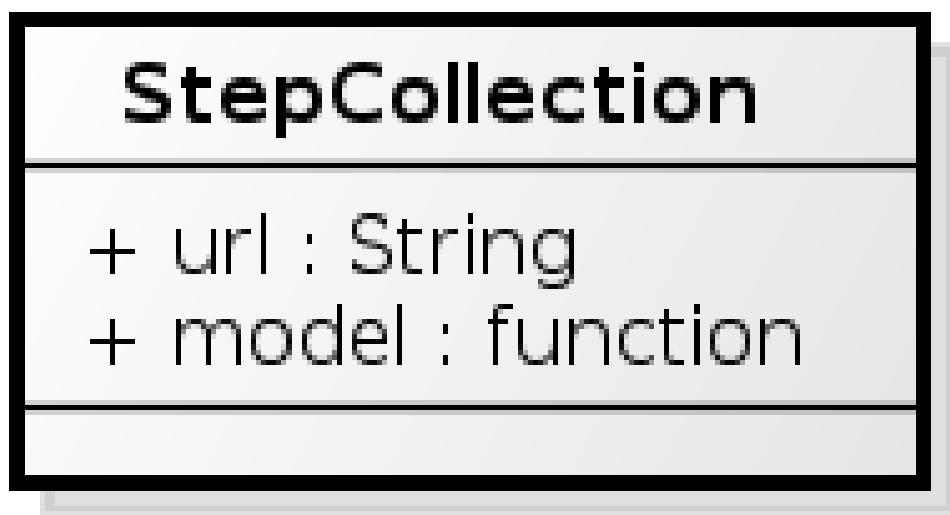


Figura 40: Diagramma classe *StepCollection*

### 5.1.2.3 StepCollection

- **Descrizione:** Classe che permette di gestire un insieme di passi di un processo;

- **Relazioni con altri componenti:**

La classe definisce una collezione di  
`com.sirius.sequenziatore.client.model.StepModel`.

- **Attributi:**

- + `String url`:  
campo dati di ridefinito da `Backbone.Collection` che contiene l'indirizzo *url<sub>G</sub>* per comunicare con il *server<sub>G</sub>*;
- + `function model`:  
campo dati di ridefinito da `Backbone.Collection` che contiene la definizione della classe  
`com.sirius.sequenziatore.client.model.StepModel`;



## 5.2 Server

Il *model* lato *server* gestisce la persistenza dei dati all'interno del *database* consentendo interrogazione, inserimento, cancellazione e aggiornamento.

La componente è formata dalle seguenti *classi*:

- `com.sirius.sequenziatore.server.model.IDataAccessObject;`
- `com.sirius.sequenziatore.server.model.ITransferObject;`
- `com.sirius.sequenziatore.server.model.UserDao;`
- `com.sirius.sequenziatore.server.model.ProcessDao;`
- `com.sirius.sequenziatore.server.model.ProcessOwnerDao;`
- `com.sirius.sequenziatore.server.model.StepDao;`
- `com.sirius.sequenziatore.server.model.User;`
- `com.sirius.sequenziatore.server.model.Process;`
- `com.sirius.sequenziatore.server.model.Step;`
- `com.sirius.sequenziatore.server.model.Data;`
- `com.sirius.sequenziatore.server.model.DataType;`
- `com.sirius.sequenziatore.server.model.Condition;`
- `com.sirius.sequenziatore.server.model.Constraint;`
- `com.sirius.sequenziatore.server.model.NumericConstraint;`
- `com.sirius.sequenziatore.server.model.TemporalConstraint;`
- `com.sirius.sequenziatore.server.model.GeographicConstraint;`
- `com.sirius.sequenziatore.server.model.DataSent;`
- `com.sirius.sequenziatore.server.model.IDataValue;`
- `com.sirius.sequenziatore.server.model.TextualValue;`
- `com.sirius.sequenziatore.server.model.NumericValue;`
- `com.sirius.sequenziatore.server.model.ImageValue;`
- `com.sirius.sequenziatore.server.model.GeographicValue;`
- `com.sirius.sequenziatore.server.model.UserStep;`
- `com.sirius.sequenziatore.server.model.ProcessOwner;`

### 5.2.1 Package `com.sirius.sequenziatore.client.model`

#### 5.2.1.1 `IDataAccessObject`

- **Descrizione:** Interfaccia che permette di gestire la comunicazione e l'interrogazione con il *database*.
- **Metodi:**
  - `+ void setJdbcTemplate(JdbcTemplate jdbcTemplate):`  
Imposta i parametri per l'accesso alla sorgente dei dati;
  - `+ ITransferObject getAll():`  
Ritorna tutti i dati di competenza della classe che estende questa interfaccia.

#### 5.2.1.2 `ITransferObject`

- **Descrizione:** Interfaccia realizzata dai tipi che modellano i dati del *database*.

#### 5.2.1.3 `UserDao`

- **Descrizione:** Classe che si occupa delle interrogazioni del *database* relative agli utenti del sistema.
- **Relazione con altre componenti:** la classe implementa la seguente interfaccia:

- `com.sirius.sequenziatore.server.model.IDataAccessObject`.

La classe invoca i metodi della classe:

- `com.sirius.sequenziatore.server.model.User`.

- **Attributi:**

- `JdbcTemplate jdbcTemplate:`  
Oggetto che fornisce l'accesso alla sorgente dei dati;

- **Metodi:**

- `+ User getUser(String userName):`  
Ritorna l'utente con il nome utente specificato;
- `+ List<User> getAllUser():`  
Ritorna tutti gli utenti;

- + boolean insertUser(User user) :  
Aggiunge l'utente passato come parametro;
- + public boolean updateUser(User user) :  
Aggiorna i dati dell'utente con il nome utente corrispondente a quello dell'utente passato, con i dati dell'utente passato.

#### 5.2.1.4 ProcessDao

- **Descrizione:** Classe che si occupa delle interrogazioni del *database* relative ai processi.
- **Relazione con altre componenti:** la classe implementa la seguente interfaccia:

- com.sirius.sequenziatore.server.model.IDataAccessObject.

La classe invoca i metodi della classe:

- com.sirius.sequenziatore.server.model.Process.

- **Attributi:**

- - JdbcTemplate jdbcTemplate:  
Oggetto che fornisce l'accesso alla sorgente dei dati;

- **Metodi:**

- + Process getProcess(int id):  
Ritorna il processo con l'id specificato;
- + List<Process> getAllProcess():  
Ritorna tutti i processi;
- + boolean insertProcess(Process process) :  
Aggiunge il processo passato come parametro;
- + public boolean updateProcess(Process process) :  
Aggiorna i dati del processo con lo stesso id di quello del processo passato, con i dati del processo passato.

#### 5.2.1.5 ProcessOwnerDao

- **Descrizione:** Classe che si occupa delle interrogazioni del *database* relative all'autenticazione del *ProcessOwner*.
- **Relazione con altre componenti:** la classe implementa la seguente interfaccia:

- `com.sirius.sequenziatore.server.model.IDataAccessObject`.

La classe invoca i metodi della classe:

- `com.sirius.sequenziatore.server.model.ProcessOwner`.

- **Attributi:**

- `- JdbcTemplate jdbcTemplate:`  
Oggetto che fornisce l'accesso alla sorgente dei dati;

- **Metodi:**

- `+ Process getProcessOwner():`  
Ritorna l'oggetto rappresentante il *ProcessOwner*.

#### 5.2.1.6 StepDao

- **Descrizione:** Classe che si occupa delle interrogazioni del *database* relative a tutte le operazioni sui passi dei processi.

- **Relazione con altre componenti:** la classe implementa la seguente interfaccia:

- `com.sirius.sequenziatore.server.model.IDataAccessObject`.

La classe invoca i metodi della classe:

- `com.sirius.sequenziatore.server.model.Step;`
- `com.sirius.sequenziatore.server.model.UserStep;`
- `com.sirius.sequenziatore.server.model.DataSent.`

- **Attributi:**

- `- JdbcTemplate jdbcTemplate:`  
Oggetto che fornisce l'accesso alla sorgente dei dati;

- **Metodi:**

- `+ Step getStep(int id):`  
Ritorna il passo con l'id specificato;
- `+ List<Step> getAllStep():`  
Ritorna tutti i passi;
- `+ List<Step> getStepOf(int ProcessId):`  
Ritorna tutti i passi appartenenti al processo di cui si è passato l'id;

- + boolean insertStep(Step step) :  
Aggiunge il passo passato come parametro;
- + public boolean updateStep(Step step) :  
Aggiorna i dati del passo con l'id corrispondente a quello del passo passato, con i dati del passo passato;
- + List<UserStep> userStep(String userName)  
Ritorna una lista di oggetti informativi sullo stato dei passi in corso da parte dell'utente di cui si è passato il nome utente;
- + List<UserStep> userProcessStep(String userName, processId)  
Ritorna una lista di oggetti informativi sullo stato dei passi in corso appartenenti al processo di cui si è passato l'id da parte dell'utente di cui si è passato il nome utente;
- + List<DataSent> getData(Step step)  
Ritorna tutti i dati da tutti gli utenti relativi al passo passato;
- + List<DataSent> getData(String userName, Step step)  
Ritorna tutti i dati inviati dall'utente di cui si è passato il nome utente relativi al passo passato;
- + List<DataSent> getWaitingData()  
Ritorna tutti i dati di tutti i passi in attesa di approvazione;
- + boolean completeStep(String userName, Step step, List<DataSent> data, Step next)  
Notifica e aggiorna nel *database* lo stato dell'utente quando completa o tenta di completare un passo.

#### 5.2.1.7 User

- **Descrizione:** Classe che modella gli utenti del sistema e che funge da interscambio dei dati di quest'ultimi con il *database*.
- **Relazione con altre componenti:** la classe implementa la seguente interfaccia:
  - com.sirius.sequenziatore.server.model.ITransferObject.
- **Attributi:**
  - - String userName:  
Nome utente;
  - - String password:  
Password dell'utente;

- - **String name:**  
Nome anagrafico dell'utente;
- - **String surName:**  
Cognome dell'utente;
- - **Date dateOfBirth:**  
Data di nascita dell'utente;
- - **String email:**  
Indirizzo di posta elettronica dell'utente;
- - **int id:**  
Codice identificativo id associato all'utente.

- **Metodi:**

- + **String getUsername():**  
Ritorna il nome utente;
- + **void setUsername(String userName):**  
Imposta il nome utente;
- + **String getPassword():**  
Ritorna la password dell'utente;
- + **void setPassword(String password):**  
Imposta la password dell'utente;
- + **String getName():**  
Ritorna il nome anagrafico dell'utente;
- + **void setName(String name):**  
Imposta il nome anagrafico dell'utente;
- + **String getSurName():**  
Ritorna il cognome dell'utente;
- + **void setSurName(String surName):**  
Imposta il cognome dell'utente;
- + **Date getDateOfBirth():**  
Ritorna la data di nascita dell'utente;
- + **void setDateOfBirth(Date dateOfBirth):**  
Imposta la data di nascita dell'utente;
- + **String getEmail():**  
Ritorna l'indirizzo di posta elettronica dell'utente;
- + **void setEmail(String email):**  
Imposta l'indirizzo di posta elettronica dell'utente;

- + `int getId()`:  
Ritorna il codice `id` associato all'utente;
- + `void setId(int id)`:  
Imposta il codice `id` associato all'utente.

#### 5.2.1.8 Process

- **Descrizione:** Classe che modella i processi del sistema e che funge da interscambio dei dati di quest'ultimi con il *database*.
- **Relazione con altre componenti:** la classe implementa la seguente interfaccia:

- `com.sirius.sequenziatore.server.model.ITransferObject`.

- **Attributi:**

- - `String name`:  
Nome del processo;
- - `String description`:  
Descrizione del processo;
- - `int completionsMax`:  
Numero massimo di completamenti del processo;
- - `Date dateOfTermination`:  
Data di terminazione del processo;
- - `boolean terminated`:  
Boolano vero quando il processo è terminato;
- - `int maxTree`:  
Massimo alberi del processo;
- - `List<Integer> stepsId`:  
Lista di codici `id` relativi ai passi del processo;
- - `int id`:  
Codice identificativo `id` associato al processo.

- **Metodi:**

- + `String getName()`:  
Ritorna il nome del processo;
- + `void setName(String name)`:  
Imposta il nome del processo;

- + `String getDescription():`  
Ritorna la descrizione del processo;
- + `void setDescription(String description):`  
Imposta la descrizione del processo;
- + `int getCompletionsMax():`  
Restituisce il numero massimo di completamenti del processo;
- + `void setCompletionsMax(int completionsMax):`  
Imposta il numero massimo di completamenti del processo;
- + `Date getDateOfTermination():`  
Ritorna data di terminazione del processo;
- + `void setDateOfTermination(Date dateOfTermination):`  
Imposta la data di terminazione del processo;
- + `boolean isTerminated():`  
Ritorna vero se il processo è terminato;
- + `void setTerminated(boolean terminated):`  
Imposta vero se il processo è terminato;
- + `int getMaxTree():`  
Ritorna il massimo alberi del processo;
- + `void setMaxtree(int maxTree):`  
Imposta il massimo alberi del processo;
- + `List<Integer> getStepsId():`  
Ritorna lista di codici id relativi ai passi del processo;
- + `void setStepsId(List<Integer> stepsId):`  
Imposta lista di codi id relativi ai passi del processo;
- + `int getId():`  
Ritorna codice identificativo id associato al processo;
- + `void setId(int id):`  
Imposta codice identificativo id associato al processo.

#### 5.2.1.9 Step

- **Descrizione:** Classe che modella i passi del sistema e che funge da interscambio dei dati di quest'ultimi con il *database*.
- **Relazione con altre componenti:** la classe implementa la seguente interfaccia:
  - `com.sirius.sequenziatore.server.model.ITransferObject`.



La classe contiene istanze di:

- `com.sirius.sequenziatore.server.model.Condition;`
- `com.sirius.sequenziatore.server.model.Data.`

• **Attributi:**

- `int id:`  
Codice identificativo `id` associato al passo;
- `String description:`  
Descrizione del passo;
- `List<Data> data:`  
Lista con i campi dato del passo;
- `List<Condition> conditions:`  
Lista delle condizioni di avanzamento del passo;
- `int processId:`  
Codice identificativo `id` associato al processo padre;
- `boolean first:`  
Booleano vero se il passo è primo per il processo padre.

• **Metodi:**

- `+ int getId():`  
Ritorna codice identificativo `id` associato al passo;
- `+ void setId(int id):`  
Imposta codice identificativo `id` associato al passo;
- `+ String getDescription():`  
Ritorna descrizione del passo;
- `+ void setDescription(String description):`  
Imposta descrizione del passo;
- `+ List<Data> getData():`  
Ritorna lista con i campi dato del passo;
- `+ void setData(List<Data> data):`  
Imposta lista con i campi dato del passo;
- `+ List<Condition> getConditions():`  
Ritorna lista delle condizioni di avanzamento del passo;
- `+ void setConditions(List<Condition> conditions):`  
Imposta lista delle condizioni di avanzamento del passo;

- + `int` `getProcessId()`:  
Ritorna codice `id` associato al processo padre;
- + `void` `setProcessId(int processId)`:  
Imposta codice `id` associato al processo padre;
- + `boolean` `isFirst()`:  
Ritorna vero se il passo è primo per il processo padre;
- + `void` `setFirst()`:  
Imposta vero se il passo è primo per il processo padre.

#### 5.2.1.10 Data

- **Descrizione:** Classe che modella i campi dato richiesti.

- **Attributi:**

- - `String` `name`:  
Nome del campo dati;
- - `DataType` `type`:  
Tipo di dato richiesto dal campo;
- - `int` `id`;  
Codice identificativo `id` associato al campo dati.

- **Metodi:**

- + `String` `getName()`:  
Ritorna il nome del campo dati;
- + `void` `setName(String name)`:  
Imposta il nome del campo dati;
- + `DataType` `getType()`:  
Ritorna il tipo di dato richiesto dal campo;
- + `void` `setType(DataType type)`:  
Imposta il tipo di dato richiesto dal campo;
- + `int` `getId()`:  
Ritorna il codice `id` associato al campo dati;
- + `void` `setId(int id)`:  
Imposta il codice `id` associato al campo dati.

#### 5.2.1.11 DataType

- **Descrizione:** Enumerazione tipo di dato.
- **Attributi:**
  - + enum DataType{TEXTUAL, NUMERIC, IMAGE, GEOGRAPHIC:}  
Enumerazione tipo di dato.

#### 5.2.1.12 Condition

- **Descrizione:** Classe che modella le condizioni di avanzamento di un passo.
- **Relazione con altre componenti:** La classe contiene istanze di:
  - com.sirius.sequenziatore.server.model.Constraint;
- **Attributi:**
  - - int id:  
Codice identificativo id associato alla condizione di avanzamento;
  - - boolean requiresApproval:  
Booleano vero se è richiesta l'approvazione del *Process Owner*;
  - - List<Constraint> constraints:  
Lista di vincoli che soddisfano la condizione di avanzamento;
  - - boolean optional:  
Booleano vero se la condizione è opzionale per l'avanzamento;
  - - int nextStepId:  
Codice identificativo id associato al passo successivo.
- **Metodi:**
  - + int getId():  
Ritorna il codice id associato alla condizione di avanzamento;
  - + void setId(int id):  
Imposta il codice id associato alla condizione di avanzamento;
  - + boolean isRequiresApproval():  
Ritorna vero se è richiesta l'approvazione del *Process Owner*;
  - + void setRequiresApproval(boolean requiresApproval):  
Imposta vero se è richiesta l'approvazione del *Process Owner*;
  - + List<Constraint> getConstraints():  
Ritorna lista di vincoli che soddisfano la condizione di avanzamento;

- + void setConstraints(List<Constraint> constraints):  
Imposta lista di vincoli che soddisfano la condizione di avanzamento;
- + boolean isOptional():  
Ritorna vero se la condizione è opzionale per l'avanzamento;
- + void setOptional(boolean optional):  
Imposta vero se la condizione è opzionale per l'avanzamento;
- + int getNextStepId():  
Ritorna codice id del passo successivo;
- + void setNextStepId(int nextStepId):  
Imposta codice id del passo successivo.

#### 5.2.1.13 Constraint

- **Descrizione:** Classe astratta che modella i vincoli.
- **Attributi:**
  - - Data associatedData:  
Campo dati su cui è posto il vincolo.
- **Metodi:**
  - + Data getAssociatedData():  
Ritorna il campo dati su cui è posto il vincolo;
  - + void setAssociatedData(Data associatedData):  
Imposta il campo dati sui cui è posto il vincolo.

#### 5.2.1.14 NumericConstraint

- **Descrizione:** Classe che modella i vincoli numerici.
- **Relazione con altre componenti:** la classe estende la seguente classe:
  - com.sirius.sequenziatore.server.model.Constraint.
- **Attributi:**
  - - int id:  
Codice identificativo id associato al vincolo;
  - - int minDigits:  
Minimo numero di cifre;

- - `int maxDigits:`  
Massimo numero di cifre;
- - `boolean decimal:`  
Booleano vero se atteso un decimale;
- - `double minValue:`  
Valore minimo;
- - `double maxValue:`  
Valore massimo;

• **Metodi:**

- + `int getId():`  
Ritorna codice id del vincolo;
- + `void setId(int id):`  
Imposta codice id del vincolo;
- + `int getMinDigits():`  
Ritorna minimo numero di cifre;
- + `void setMinDigits(int minDigits):`  
Imposta minimo numero di cifre;
- + `int getMaxDigits():`  
Ritorna massimo numero di cifre;
- + `void setMaxDigits(int maxDigits):`  
Imposta massimo numero di cifre;
- + `boolean isDecimal():`  
Ritorna vero se atteso un decimale;
- + `void setDecimal(boolean decimal):`  
Imposta vero se atteso un decimale;
- + `double getMinValue():`  
Ritorna valore minimo;
- + `void setMinValue(double minValue):`  
Imposta valore minimo;
- + `double getMaxValue():`  
Ritorna valore massimo;
- + `void setMaxValue(double maxValue):`  
Imposta valore massimo;

#### 5.2.1.15 TemporalConstraint

- **Descrizione:** Classe che modella i vincoli temporali.
- **Relazione con altre componenti:** la classe estende la seguente classe:
  - `com.sirius.sequenziatore.server.model.Constraint`.
- **Attributi:**
  - `int id`:  
Codice identificativo `id` associato al vincolo;
  - `Date begin`:  
Inizio arco temporale valido;
  - `Date end`:  
Fine arco temporale valido.
- **Metodi:**
  - `+ int getId()`:  
Ritorna codice `id` del vincolo;
  - `+ void setId(int id)`:  
Imposta codice `id` del vincolo;
  - `+ Date getBegin()`:  
Ritorna inizio arco temporale valido;
  - `+ void setBegin(Date begin)`:  
Imposta inizio arco temporale valido;
  - `+ Date getEnd()`:  
Ritorna fine arco temporale valido;
  - `+ void setEnd(Date end)`:  
Imposta fine arco temporale valido.

#### 5.2.1.16 GeographicConstraint

- **Descrizione:** Classe che modella i vincoli geografici.
- **Relazione con altre componenti:** la classe estende la seguente classe:
  - `com.sirius.sequenziatore.server.model.Constraint`.
- **Attributi:**
  - `int id`:  
Codice identificativo `id` associato al vincolo;

- - `double latitude:`  
    Latitudine richiesta;
- - `double longitude:`  
    Longitudine richiesta;
- - `double altitude:`  
    Altitudine richiesta;
- - `double radius:`  
    Raggio di tolleranza.

- **Metodi:**

- + `int getId():`  
    Ritorna codice id del vincolo;
- + `void setId(int id):`  
    Imposta codice id del vincolo;
- + `double getLatitude():`  
    Ritorna latitudine richiesta;
- + `void setLatitude(double latitude):`  
    Imposta latitudine richiesta;
- + `double getLongitude():`  
    Ritorna longitudine richiesta;
- + `void setLongitude(double longitude):`  
    Imposta longitudine richiesta;
- + `double getAltitude():`  
    Ritorna altitudine richiesta;
- + `void setAltitude(double altitude):`  
    Imposta altitudine richiesta;
- + `double getRadius():`  
    Ritorna raggio di tolleranza;
- + `void setRadius(double radius):`  
    Imposta raggio di tolleranza.

#### 5.2.1.17 DataSent

- **Descrizione:** Classe che modella i dati ricevuti dagli utenti che funge da interscambio con il *database*.
- **Relazione con altre componenti:** la classe implementa la seguente interfaccia:

- `com.sirius.sequenziatore.server.model.ITransferObject`.

La classe invoca i metodi della classe:

- `com.sirius.sequenziatore.server.model.IDataValue`.

- **Attributi:**

- `String user`:  
Nome utente dell'utente che ha inviato il dato;
- `DataType type`:  
Tipo del dato inviato;
- `IDataValue value`:  
Oggetto con il valore del dato;
- `int stepId`:  
Codice id del passo richiedente il dato.

- **Metodi:**

- `+ String getUser()`:  
Ritorna nome utente dell'utente che ha inviato il dato;
- `+ void setUser(String user)`:  
Imposta nome utente dell'utente che ha inviato il dato;
- `+ DataType getType()`:  
Ritorna il tipo del dato inviato;
- `+ void setType(DataType type)`:  
Imposta il tipo del dato inviato;
- `+ IDataValue getValue()`:  
Ritorna oggetto con il valore del dato;
- `+ void setValue(IDataValue value)`:  
Imposta oggetto con il valore del dato;
- `+ int getStepId()`:  
Ritorna codice id del passo richiedente il dato;
- `+ void setStepId(int stepId)`:  
Imposta codice id del passo richiedente il dato.

#### 5.2.1.18 IDataValue

- **Descrizione:** Interfaccia che modella i valori dei dati ricevuti.
- **Metodi:**



- + int getId():  
Ritorna codice id associato al valore;
- + void setId(int id):  
Imposta codice id associato al valore.

#### 5.2.1.19 TextualValue

- **Descrizione:** Classe che modella i valori dei dati testuali.
- **Relazione con altre componenti:** la classe implementa la seguente interfaccia:
  - com.sirius.sequenziatore.server.model.IDataValue.
- **Attributi:**
  - - int id:  
Codice id associato al valore;
  - - String value:  
Valore testuale.
- **Metodi:**
  - + String getValue():  
Ritorna valore testuale;
  - + void setValue(String value):  
Imposta valore testuale.

#### 5.2.1.20 NumericValue

- **Descrizione:** Classe che modella i valori dei dati numerici.
- **Relazione con altre componenti:** la classe implementa la seguente interfaccia:
  - com.sirius.sequenziatore.server.model.IDataValue.
- **Attributi:**
  - - int id:  
Codice id associato al valore;
  - - double value:  
Valore numerico.

- **Metodi:**

- + double `getValue()`:  
Ritorna valore numerico;
- + void `setValue(double value)`:  
Imposta valore numerico.

#### 5.2.1.21 ImageValue

- **Descrizione:** Classe che modella i valori dei dati immagine.
- **Relazione con altre componenti:** la classe implementa la seguente interfaccia:

- com.sirius.sequenziatore.server.model.IDataValue.

- **Attributi:**

- - int `id`:  
Codice `id` associato al valore;
- - String `imageUrl`:  
Percorso *URL* dell'immagine.

- **Metodi:**

- + String `getImageUrl()`:  
Ritorna percorso *URL* dell'immagine;
- + void `setImageUrl(String imageUrl)`:  
Imposta percorso *URL* dell'immagine.

#### 5.2.1.22 GeographicValue

- **Descrizione:** Classe che modella i valori dei dati geografici.
- **Relazione con altre componenti:** la classe implementa la seguente interfaccia:

- com.sirius.sequenziatore.server.model.IDataValue.

- **Attributi:**

- - int `id`:  
Codice `id` associato al valore;

- - double latitude:  
    Latitudine;
- - double longitude:  
    Longitudine;
- - double altitude:  
    Altitudine.

- **Metodi:**

- + double getLatitude():  
    Ritorna latitudine;
- + void setLatitude(double latitude):  
    Imposta latitudine;
- + double getLongitude():  
    Ritorna longitudine;
- + void setLongitude(double longitude):  
    Imposta longitudine;
- + double getAltitude():  
    Ritorna altitudine;
- + void setAltitude(double altitude):  
    Imposta altitudine.

#### 5.2.1.23 UserStep

- **Descrizione:** Classe che modella i passi in corso e che funge da interscambio dei dati di quest'ultimi con il *database*.
- **Relazione con altre componenti:** la classe implementa la seguente interfaccia:

- com.sirius.sequenziatore.server.model.ITransferObject.

- **Attributi:**

- + enum stepStates{ONGOING, EXPECTANT, REJECTED, APPROVED}:  
    Enumerazione stato avanzamento;
- - int currentStepId:  
    Codice id del passo attuale;
- - stepStates state:  
    Stato avanzamento;

- - `String user`:  
Nome utente dell'utente del caso.

- **Metodi:**

- + `int getCurrentStepId()`:  
Ritorna il codice id del passo attuale;
- + `void setCurrentStepId(int currentStepId)`:  
Imposta il codice id del passo attuale;
- + `stepStates getState()`:  
Ritorna stato avanzamento;
- + `void setStates(stepStates state)`:  
Imposta stato avanzamento;
- + `String getUser()`:  
Restituisce nome utente dell'utente in caso;
- + `void setUser(String user)`:  
Imposta nome utente dell'utente in caso.

#### 5.2.1.24 ProcessOwner

- **Descrizione:** Classe che modella il ProcessOwner e che funge da interscambio dei dati di quest'ultimo con il *database*.

- **Relazione con altre componenti:** la classe implementa la seguente interfaccia:

- `com.sirius.sequenziatore.server.model.ITransferObject`.

- **Attributi:**

- - `String userName`:  
Nome utente *Process Owner*;
- - `String password`:  
Password *Process Owner*.

- **Metodi:**

- + `String getUserName()`:  
Ritorna il nome utente del *Process Owner*;
- + `void setUserName(String userName)`:  
Imposta il nome utente del *Process Owner*;

- + `String getPassword()`:  
Ritorna la password del *Process Owner*;
- + `void setPassword(String password)`:  
Imposta la password del *Process Owner*.