



SIRIUS

SEQUENZIATORE

Definizione di prodotto

Versione 1.0.0

Ingegneria Del Software AA 2013-2014

Informazioni documento

Titolo documento:	Definizione Di Prodotto
Data creazione:	2014-01-29
Versione attuale:	1.0.0
Utilizzo:	Interno
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 *Specifica Tecnica_v1.0.0.pdf*

Diario delle modifiche

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

Indice

1	Introduzione	1
1.1	Scopo del documento	1
1.2	Glossario	1
2	Standard di progetto	1
3	Specifica della componente view	1
3.1	Package com.sirius.sequenziatore.client.view	2
3.2	Package com.sirius.sequenziatore.client.view.user	2
3.3	Package com.sirius.sequenziatore.client.view.processowner	3
4	Specifica della componente presenter	4
4.1	Client	4
4.1.1	Package com.sirius.sequenziatore.client.presenter	5
4.1.2	Package com.sirius.sequenziatore.client.presenter.processowner	7
4.2	Server	7
4.2.1	Package com.sirius.sequenziatore.server.presenter.common	8
4.2.2	Package com.sirius.sequenziatore.server.presenter.processowner	9
4.2.3	Package com.sirius.sequenziatore.server.presenter.user	11
5	Specifica della componente model	12

1 Introduzione

1.1 Scopo del documento

In questo documento si prefigge come obiettivo la definizione in modo approfondito della struttura, delle componenti e delle relazioni tra queste ultime del prodotto il *Sequenziatore* approfondendo quanto riportato nel documento di Specifica Tecnica

1.2 Glossario

Al fine di facilitare la comprensione del seguente documento, ed in generale di ogni documento che verrà fornito da parte del team *Sirius*, è stato creato appositamente un glossario (*Glossario_v2.0.0.pdf*) contenente la definizione dei termini più complessi o di quelli che necessitano un approfondimento. Questi vocaboli sono contrassegnati in ogni documento dal pedice G (_G).

2 Standard di progetto

3 Specifica della componente view

La componente *view* è formata da *template HTML_G* che possono contenere codice *javascript_G* 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_G* a seconda dei parametri del metodo. Per questo motivo, le interfacce presenti nel *package* *com.sirius.sequenziatore.client.view* definite nel documento *SpecificiTecnica_v1.0.0.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_G*.

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 *iwidget* 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_G* 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 alla classe principale **Router**, che estende la classe **Backbone.Router** fornita dal *framework_G 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 composta dalle seguenti *classi*:

- [com.sirius.sequenziatore.client.presenter.Router](#);
- [com.sirius.sequenziatore.client.presenter.Login](#);

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

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.AddStep`;
- `com.sirius.sequenziatore.client.presenter.processowner.CheckStep`;
- `com.sirius.sequenziatore.client.presenter.processowner.ManageProcess`;

- **Attributi:**

- `Session session`:
oggetto di tipo `com.sirius.sequenziatore.client.modelUserData`, 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_G*, un metodo della classe;

- **Metodi:**

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

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 tramite metodi della classe `com.sirius.sequenziatore.client.viewLogin`.

- **Attributi:**

- `UserDataModel model`:
campo dati 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:**

- `+ null initialize()`:
metodo ridefinito da `Backbone.View`, invocato alla costruzione di ciascun oggetto della classe, che consente di aggiungere la pagina *login* alla pagina `HTMLG`, se non è ancora presente;
- `+ null render()`:
metodo ridefinito da `Backbone.View`, che consente di aggiungere alla pagina `HTMLG` il *template* campo dati della classe;
- `+ null 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.processowner`

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`

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

4.2.1.1 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`
- **Attributi:**
- **Metodi:**
 - `+RegisterUser(Utente toBeRegistered)`, questo metodo gestirà un metodo **POST** e restituirà un errore qualora ci siano stati problemi nella registrazione;

4.2.1.2 LoginController

- **Descrizione:** Questa classe gestirà le richieste di *log in*, dovrà controllare se l'utente esiste nel sistema e se le credenziali d'accesso siano corrette;
- **Mappatura base:** `\login`
- **Attributi:**
- **Metodi:**
 - `+CheckLogin(Utente toBeLogged)` , questo metodo gestirà un metodo di tipo **POST** , controllerà le credenziali di accesso e dovrà restituire un errore qual'ora ci siano stati problemi nella login;

4.2.1.3 StepInfoController

- **Descrizione:** Questa classe restituirà lo scheletro, quindi la composizione del passo richiesto;
- **Mappatura base:** `\step\{id}`
- **Attributi:**
- **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*;

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}`
- **Attributi:**
- **Metodi:**

–

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

IMMAGINE PACKAGE

4.2.2.1 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`
- **Attributi:**
- **Metodi:**

—

4.2.2.2 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`
- **Attributi:**
- **Metodi:**
 - +void CreateProcess(Process) questo metodo gestisce una richiesta di tipo **POST** e permette l' inserimento del processo fornito nel *database*;
 - +ProcessList 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

- **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`
- **Attributi:**
- **Metodi:**
 - +StepList GetStepToApprove(), il metodo gestisce una richiesta di tipo *GET*, e restituirà un oggetto di tipo StepList contenente tutti i dati che richiedono approvazione;

- +void ApproveResponse(Step), il metodo gestisce una richiesta di tipo *POST*, riceve 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

IMMAGINE PACKAGE

4.2.3.1 UserStepController

- **Descrizione:** Questa classe gestisce la ricezione di un passo di 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`
- **Relazione con altri componenti:**
- **Attributi:**
- **Metodi:**
 -

4.2.3.2 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}`
- **Attributi:**
- **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;
 - +Status 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;

- +ProcessList 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 ProcessInfoController

- **Descrizione:**
- **Mappatura base:** $\backslash process \backslash \{id\}$
- **Attributi:**
- **Metodi:**

–

4.2.3.4 ProcessInfoController

- **Descrizione:**
- **Mappatura base:** $\backslash process \backslash \{id\}$
- **Attributi:**
- **Metodi:**

–

5 Specifica della componente model