

**SUPSI**

# Gestionale per la formazione basata su blockchain

---

Studente/i

**Bonardi Gionas**  
**Pulfer Brian**

Relatore

**Brocco Amos**

---

Correlatore

-

---

Committente

**Brocco Amos**

---

Corso di laurea

**Ingegneria Informatica**

Modulo

**Progetto di semestre**

---

Anno

**2019**

---

Data

-

STUDENTSUPSI

# Indice

<b>1. ABSTRACT.....</b>	<b>4</b>
<b>2. TIFORMA .....</b>	<b>5</b>
2.1. ELEMENTI PRINCIPALI.....	5
2.1.1. <i>Contatto</i> .....	6
2.1.2. <i>Formazione</i> .....	6
<b>3. PROTOTIPO .....</b>	<b>6</b>
3.1. FUNZIONALITÀ.....	6
<b>4. TECNOLOGIE UTILIZZATE .....</b>	<b>8</b>
4.1. BLOCKCHAIN .....	8
4.2. HYPERLEDGER .....	9
4.2.1. <i>Hyperledger Fabric</i> .....	9
4.2.2. <i>Hyperledger Composer</i> .....	9
<b>5. BACK-END.....</b>	<b>11</b>
5.1. PARTICIPANTS .....	11
5.2. ASSETS .....	12
5.3. TRANSACTIONS.....	14
5.4. QUERIES .....	15
5.5. ACCESS CONTROL .....	17
5.6. REST-SERVER .....	17
5.7. TEST .....	18
<b>6. FRONT-END .....</b>	<b>19</b>
6.1. ARCHITETTURA PROGETTO ANGULAR.....	20
6.2. DASHBOARD .....	21
6.3. NAVBAR.....	22
6.3.1. <i>Routing</i> .....	22
6.4. CONTAINER .....	23
6.5. MODULI.....	23
6.5.1. <i>Routing</i> .....	25
6.5.2. <i>Service</i> .....	26
6.5.3. <i>Lista</i> .....	28
6.5.4. <i>Dettaglio</i> .....	30
6.5.5. <i>Nuovo</i> .....	32
6.5.6. <i>Loading</i> .....	33
<b>7. SVILUPPI FUTURI .....</b>	<b>34</b>
7.1. GESTIONE DELLE QUERIES .....	34
7.2. PANNELLO DEI MODULI .....	35
7.3. CONTROLLO SUI CAMPI .....	35
<b>8. CONCLUSIONE .....</b>	<b>36</b>
<b>9. BIBLIOGRAFIA.....</b>	<b>37</b>

<b>10.</b>	<b>AMBIENTE DI SVILUPPO .....</b>	<b>38</b>
10.1.	INSTALLAZIONE DI HYPERLEDGER COMPOSER.....	38
10.2.	SVILUPPO DELL'APPLICAZIONE .....	40
10.3.	ANGULAR E DIPENDENZE .....	41
<b>11.</b>	<b>GUIDA ALL'UTILIZZO.....</b>	<b>42</b>
11.1.	INTRODUZIONE.....	42
11.2.	MODO D'USO .....	43
11.2.1.	<i>Menu</i> .....	43
11.3.	SEZIONI.....	44
11.3.1.	<i>Studenti</i> .....	46
11.3.2.	<i>Dipartimenti</i> .....	46
11.3.3.	<i>Corsi</i> .....	46
11.3.4.	<i>Moduli</i> .....	46
11.3.5.	<i>Formazione</i> .....	46
11.3.6.	<i>Semestre</i> .....	46
11.3.7.	<i>Certificazione</i> .....	46
11.3.8.	<i>Ricerca</i> .....	47

## 1. Abstract

---

I percorsi formativi degli studenti sono attualmente gestiti tramite un applicativo software chiamato TIFORMA, sviluppato, allo stato attuale, per soddisfare i requisiti della SUPSI (Scuola Universitaria Professionale della Svizzera Italiana). Il software in questione è programma installato su sistema operativo di Windows e permette di gestire la fase di preiscrizione e iscrizione degli studenti a dei piani di studio contenenti moduli singoli o modelli, l'iscrizione effettiva degli studenti ai moduli e la creazione di sessioni di certificazione nella quale gli utenti che effettueranno gli esami, si vedranno assegnare una nota.

La soluzione, qui sopra descritta, si basa su un database centralizzato. Il problema principale è che è difficile verificare chi ha fatto cosa non avendo la possibilità di consultare uno storico delle aggiunte/modifiche/eliminazioni effettuate dagli utenti autorizzati ad accedere alla piattaforma. Tali utenti si devono, quindi, fidare dello stato del database centralizzato senza la possibilità di verificarne la consistenza dei dati.

L'obiettivo di questo progetto di semestre è quello di implementare una soluzione basata su blockchain al problema della gestione dei percorsi formativi. Quindi creare sia una struttura back-end (modellizzazione dei dati, generazione di un web server con metodi CRUD) e una front-end, offrendo un prototipo con delle funzionalità di base basate sulla versione attualmente utilizzata.

Attraverso la tecnologia blockchain si vuole creare una soluzione in cui i dati siano distribuiti/replicati e verificabili da chiunque disponga di una copia della blockchain stessa.

## 2. TIFORMA

L'applicazione attualmente utilizzata nella sede SUPSI per gestire il percorso formativo degli studenti è sviluppata come applicazione Windows e permette una moltitudine di operazioni. Come mostrato in *Figura 1*, tutte le funzionalità sono gestite tramite un menù laterale (rosso) che permette di accedere alle varie pagine dell'applicazione. Una volta effettuato l'accesso al menu, la barra in alto si modifica in base alla voce scelta (contestuale).



**Figura 1 - TIFORMA Windows Version**

### 2.1. Elementi principali

Gli elementi principali che compongono questa applicazione sono diversi e ben distinti nel menu verticale sulla sinistra.

### 2.1.1. Contatto

È la gestione di tutte le informazioni dei contatti (studenti, collaboratori). In queste pagine possiamo inserire tutte quelle informazioni per registrare una persona al sistema.

Per quanto riguarda il prototipo sviluppato, ci si è focalizzati sullo studente e sulle informazioni di base per inserirlo a sistema.

### 2.1.2. Formazione

In questa area si organizzano tutte le informazioni riguardanti il percorso formativo vero e proprio. Infatti, si trovano le pagine:

- **Gestioni moduli:** nella quale si possono creare i moduli.
- **Modelli di piano di studio (formazione):** sono i bachelor/master offerti come ad esempio Bachelor Informatica. In questi modelli vengono inseriti i moduli.
- **Offerta formativa:** sono i semestri che contengono i moduli e gli studenti iscritti.
- **Sessioni di certificazione:** dove lo studente riceve una nota una volta fatto un esame.

## 3. Prototipo

---

Il prototipo sviluppato permette di creare dei Corsi che possono essere aggiunti a dei Moduli, questi Moduli possono essere inseriti in una Formazione. La formazione serve per filtrare la ricerca dei moduli e degli studenti nel Semestre. Infine la creazione di una certificazione per gli studenti che hanno eseguito un esame.

### 3.1. Funzionalità

L'obiettivo di questo progetto era quello di sviluppare un primo prototipo con funzionalità di base. La prima fase che ha contraddistinto il progetto è stato quello di decidere, leggendo il manuale utente di TIFORMA, quali funzionalità implementare in questo primo prototipo.

Le operazioni di base scelte sono state:

- Creazione, Modifica e Eliminazione degli **Studenti**.
- Creazione, Modifica e Eliminazione dei **Dipartimenti**.
- Creazione, Modifica e Eliminazione dei **Corsi**.

- Creazione, Modifica e Eliminazione dei **Moduli**.
  - Inserire o eliminare **Corsi** nei **Moduli**.
- Creazione, Modifica e Eliminazione della **Formazione**.
  - Inserire o eliminare Moduli nella **Formazione**.
- Creazione, Modifica e Eliminazione dei **Semestri**.
  - Inserire o eliminare un **Modulo** nel **Semestre**.
  - Iscrizione di uno **Studente** in un **Modulo**.
- Creazione, Modifica e Eliminazione di una **Certificazione**.
- Funzionalità di ricerca/filtraggio.
- Funzionalità di stampa di report.

## 4. Tecnologie utilizzate

Per questo progetto si sono utilizzate diverse tecnologie, tra cui la blockchain e Hyperledger per la parte back-end e Angular 7 per quella front-end.

### 4.1. Blockchain

Questa tecnologia è una catena in continua crescita di blocchi che sono legati tra di loro e resi sicuri grazie all'uso della crittografia. Ogni blocco contiene un puntatore (hash code) che si collega al blocco precedente, contiene inoltre un timestamp e i dati della transazione.

Una volta che i dati in un blocco vengono registrati non possono essere alterati senza che vengano poi modificati tutti i blocchi successivi ad esso, ciò sarebbe possibile solo avendo il consenso della maggioranza della rete.

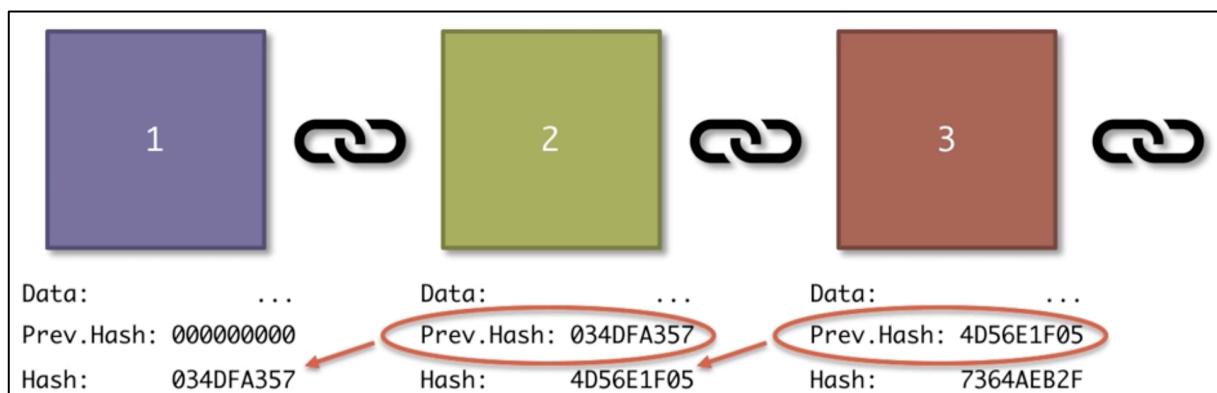


Figura 2 - Esempio Blockchain

In definitiva è un registro aperto, una sorta di libro mastro distribuito, che registra le transazioni tra più parti in modo verificabile, permanente e tramite un meccanismo di consenso.

I dati non vengono memorizzati su un solo computer ma su sistemi collegati tramite una rete peer-to-peer (P2P). Ogni nodo può aggiornare e gestire la catena in modo indipendente ma sotto il controllo consensuale di tutti i partecipanti della rete stessa.

## 4.2. Hyperledger

Hyperledger è un progetto open source fondato dalla Linux Foundation nel dicembre 2015 con l'obiettivo di portare la tecnologia Blockchain nel settore industriale. In questo progetto è stato utilizzato lo strumento Hyperledger Composer che, sfruttando il framework Hyperledger Fabric, permette di costruire un'applicazione sfruttando questa nuova tecnologia.

### 4.2.1. Hyperledger Fabric

È un framework originariamente sviluppato da IBM e inglobato poi all'interno del progetto Hyperledger. L'interessante funzionalità offerta da Fabric è la possibilità di creare una rete private e permissioned, che permette di gestire le autorizzazioni.

### 4.2.2. Hyperledger Composer

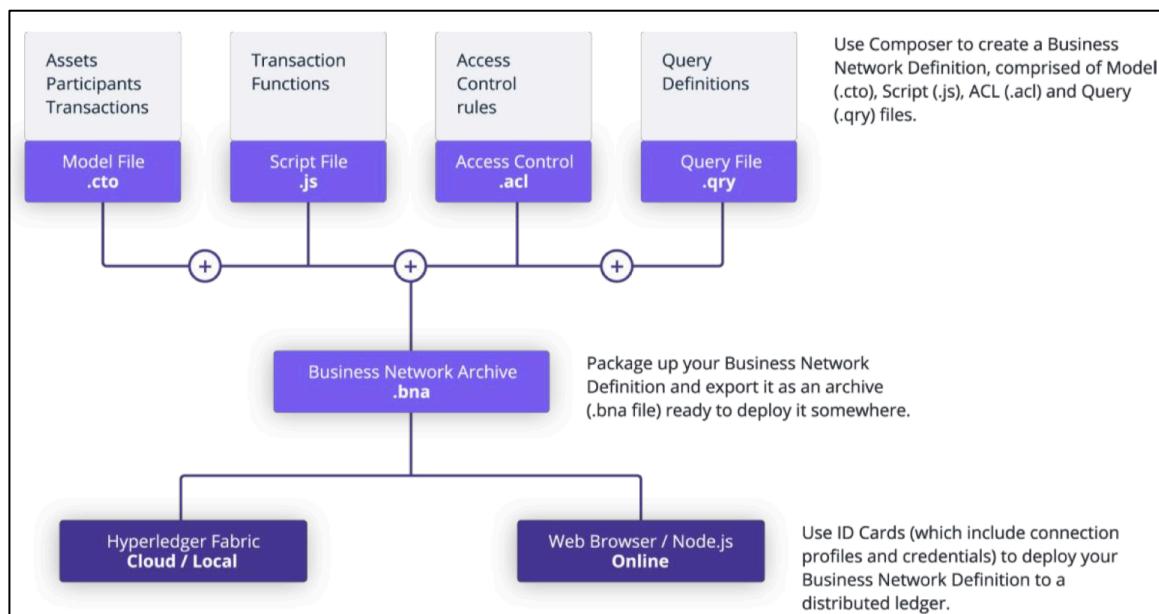
Strumento sviluppato sempre da IBM che permette di semplificare lo sviluppo di applicazioni blockchain, quindi facilitare la creazione e l'integrazione di queste applicazioni nei contesti aziendali già esistenti.

Questo strumento utilizza le API offerte da Fabric, una sorta di livello più alto, e permette di modellare una business network contenente risorse e transazioni. Per modellare tale rete, bisogna definire delle transazioni che successivamente interagiscano con le risorse. Queste transazioni sono delle funzioni scritte in Javascript.



Figura 3 - Soluzione Hyperledger

Modellata la rete, la si carica su Fabric generando un pacchetto chiamato Business Network Archive (.BNA). Si può successivamente generare un web-server permettendo così di eseguire operazioni CRUD facendo delle chiamate http.



**Figura 4 - Struttura Angular**

I principali elementi di una network sono:

- **Assets:** sono le risorse che vengono archiviati nei registri. Possono rappresentare qualsiasi cosa.
- **Partecipanti:** sono i membri di una business network. Posseggono le risorse e possono eseguire transazioni.
- **Transazioni:** sono il meccanismo con cui i membri interagiscono con le risorse.
- **Eventi:** scatenati con l'esecuzione delle transazioni.

## 5. Back-end

---

Tutta la struttura back-end è stata costruita con il framework Hyperledger Composer andando solo a definire i participant, gli assets e le transactions possibili all'interno della network.

Una volta definiti questi elementi è possibile generare il file .bna che consente ai peer della rete l'installazione e l'esecuzione della stessa, nonché l'avvio di un rest-server per poter operare.

### 5.1. Participants

Come già anticipato, i partecipanti di una rete sono i possessori delle risorse che possono effettuare delle transazioni per scambiarne la proprietà. Per questo progetto sono stati definiti i seguenti partecipanti.

**Contact:** È un'entità astratta del linguaggio di modelling. È possibile creare delle entità concrete a partire da questa. Questo partecipante è stato creato con la prospettiva che tutti i partecipanti della network dovrebbero avere delle informazioni di base come nome, cognome, e-mail ed altri.

Gli attributi di questo partecipante sono: un ID, nome, cognome, informazioni di contatto (nr. Telefono, e-mail, domicilio...), data di nascita, nazionalità.

**Student:** È un partecipante che estende contact. È la rappresentazione di uno studente. Oltre le informazioni di base, sono presenti pure le informazioni relative a: statuto, numero di matricola, un commento e piano di studio associato.

**Collaborator:** Rappresenta invece un collaboratore SUPSI. Anche questo estende il participant contact senza però nessun altro campo aggiuntivo.

## 5.2. Assets

Gli asset definiti per questo progetto sono invece i seguenti.

**Address:** È un campo opzionale di un qualsiasi contact. Un contact può avere 0 o più addresses. All'interno di un address sono presenti: un ID, via, numero civico, CAP, Città, paese, tipo di indirizzo (privato, lavoro...), e-mail, numero telefonico.

**Department:** Rappresenta un dipartimento della SUPSI. È composto da un ID e dal suo nome.

**Course:** È l'asset che identifica un corso. È composto da un ID ed il nome del corso.

**Module:** È l'asset che identifica un modulo. È composto da un ID, il nome del modulo, una durata in semestri, numero di ECTS, il dipartimento di riferimento, lo stato, il responsabile, il nome in lingua inglese, un commento e un insieme di corsi del quale il modulo è composto.

**StudyPlan:** Rappresenta un piano di studio. Ogni studente deve avere un piano di studio. Uno studyplan è formato da: nome, dipartimento, stato, commento e un insieme di moduli coi quali il piano di studio è composto. I moduli in questione sono tutti quelli che lo studente dovrà certificare durante la sua formazione per l'ottenimento del titolo di studio.

Uno studyplan ha i seguenti attributi: nome, dipartimento, stato, commento e insieme di moduli.

**Semester:** Rappresenta un semestre. Uno studente che è iscritto ad un semestre è automaticamente iscritto ai relativi moduli che sono di tipo 'StudentModule'. Un semester è composto da nome, descrizione e insieme di moduli.

**StudentModule:** Questo asset rappresenta un collegamento fra un modulo e gli studenti che vi sono iscritti. È l'equivalente nella realtà di una classe che segue un modulo durante un semestre. Uno StudentModule è composto da un ID, un modulo ed una lista di studenti iscritti.

**Certification:** È l'asset che certifica l'ottenimento di una nota da parte di uno studente in un modulo. Una certificazione è possibile solo per uno studente che è iscritto ad uno StudentModule che comprenda il modulo che si vuole certificare.

La Certification è composta da un ID, uno studente, un modulo ed un voto.

**CertificationSession:** Rappresenta l'insieme di certificazioni emanate in un semestre. Ha i seguenti attributi: nome, dipartimento, semestre, titolo, data ed un insieme di certification.

### 5.3. Transactions

Hyperledger Composer mette da subito a disposizione un set di transactions per ciascun asset o participant, come operazioni di get, create, update e delete. Ciò nonostante è stato optato per creare delle transazioni personalizzate di create, update e delete per tutti gli assets e tutti i participants in modo tale da aver più controllo sulle operazioni effettuabili nella rete.

Oltre le 3 operazioni di base per ciascun asset e participant, sono state sviluppate altre transazioni:

- AddCourseToModule
- RemoveCourseFromModule
- AddModuleToStudyPlan
- RemoveModuleFromStudyPlan
- AddStudentModuleToSemester
- RemoveStudentModuleFromSemester
- AddStudentToStudentModule
- RemoveStudentFromStudentModule
- AddCertificationToCertificationSession
- RemoveCertificationFromCertificationSession
- SubscriptionToSemester
- CreateCertification

Queste transazioni permettono l'aggiunta/rimozione di: un corso ad un modulo; di un modulo ad un piano di studio; di un modulo ad un semestre; di uno studente ad una 'classe'; di una certificazione ad una sessione di certificazione.

Le ultime due transazioni elencate permettono di passare alla seconda fase (iscrizione ad un semestre di uno studente) ed alla terza fase (certificazione di uno studente ad un modulo).

## 5.4. Queries

Hyperledger Composer prevede inoltre un proprio query language. È possibile definire un file (con estensione qry) che definisce un insieme di query attuabili sul database distribuito.

Le queries implementate sono le seguenti:

- selectStudentByID
- selectStudentByName
- selectStudentsBySurname
- selectStudentBySerialNumber
- selectCoursesByName
- selectCoursesByCourseCode
- selectModulesByName
- selectModulesByModuleCode
- selectStudentModuleByModule
- selectStudentModuleByModuleName
- selectSemestersByName
- selectSemestersByModule
- selectStudyPlanByName
- selectStudyPlanByModule
- selectCertificationsByStudent
- selectCertificationsByStudentName
- selectCertificationsByStudentSurname
- selectCertificationsByModuleName
- selectCertificationSessionsByName
- selectDepartmentByName

Queste queries permettono di trovare: degli studenti dato l'ID, il nome, il cognome o il numero di matricola; dei corsi in base al nome o al codice; dei moduli dato il nome o il codice; degli studentmodule dato il modulo o il nome del modulo; dei semestri a partire dal un nome o da un modulo contenuto; dei piani di studio con un certo nome o che contengono un certo modulo; delle certificazioni in base allo studente, il suo nome o il suo cognome o in base al

nome del modulo; delle sessioni di certificazione in base al nome; dei dipartimenti in base al loro nome.

## 5.5. Access Control

Composer prevede che le regole della network, come quali partecipanti possono effettuare quali transazioni o accedere a quali dati, siano definite in un file apposito (con estensione acl). Il file in questione non è stato modificato rispetto a quello fornito con lo scaffolding di Yeoman, pertanto tutti i partecipanti hanno tutti i privilegi.

## 5.6. REST-Server

A partire dalla network così definita (file con estensione bna), è possibile generare un REST-Server.

Il REST-Server espone diverse funzionalità, con le quali è possibile mandare richieste:

- GET che specificato il nome di un Asset o Participant, ritorna una lista di tutte le istanze di quel tipo presenti;
- GET che specificato il nome di una transazione, ritornano una lista di tutte le transazioni di quel tipo che sono avvenute nella network;
- GET che specificato un'id della transazione, ritornano il contenuto della transazione stessa;
- GET delle query specificate nella network: prendono il parametro specificato nel query file e ritornano il risultato della query;
- POST che permettono di effettuare tutte le transactions specificate della network;
- Varie operazioni di sistema (ottenimenti di informazioni circa lo storico della network, le identità, ping, ...).

L'applicazione sviluppata per il front-end si basa sul REST-Server e sfrutta esclusivamente le richieste di tipo GET per gli asset ed i participants e le richieste di tipo POST per effettuare le transazioni.

## 5.7. Test

Per testare il corretto funzionamento della rete, sono stati creati degli script bash automatici. Gli script effettuano delle richieste al REST-Server tramite il comando **curl**, ed in base al comportamento della rete stampano a console gli eventuali errori.

Per ogni partecipante e per ogni asset è stato creato un apposito script che testi le funzionalità di base (lettura, creazione, modifica, rimozione). Inoltre, è stato creato uno script che testi il funzionamento della rete per intero, creando degli asset ed effettuando transazioni a volte possibili e a volte no, come per esempio la certificazione ad uno studente di un modulo che non è presente nel suo piano di studio o al quale non è iscritto.

## 6. Front-end

---

Per la creazione del front-end abbiamo optato per l'utilizzo del framework Angular 7, consigliato dagli sviluppatori di Hyperledger, per la creazione delle pagine web.

Le applicazioni sviluppate in Angular vengono eseguite dal web browser dopo essere state scaricate dal web server. Il linguaggio utilizzato per lo sviluppo è TypeScript con l'aggiunta dell'HTML e il CSS. Il vantaggio di usare questo framework è la possibilità di suddividere la pagina in componenti e lavorare su ognuno di esso in modo indipendente dagli altri.

## 6.1. Architettura Progetto Angular

Come mostrato nella *Figura 5*, la cartella app del progetto contiene una serie di moduli e file che compongono l'applicazione vera e propria. L'applicazione parte dal file *app.module.ts*, al suo interno, sotto il valore *bootstrap*, è presente il componente iniziale che viene caricato quando si fa partire l'app, cioè *app.component.ts*.

Il componente principale richiama due altri componenti: la **navbar** e il **container**.

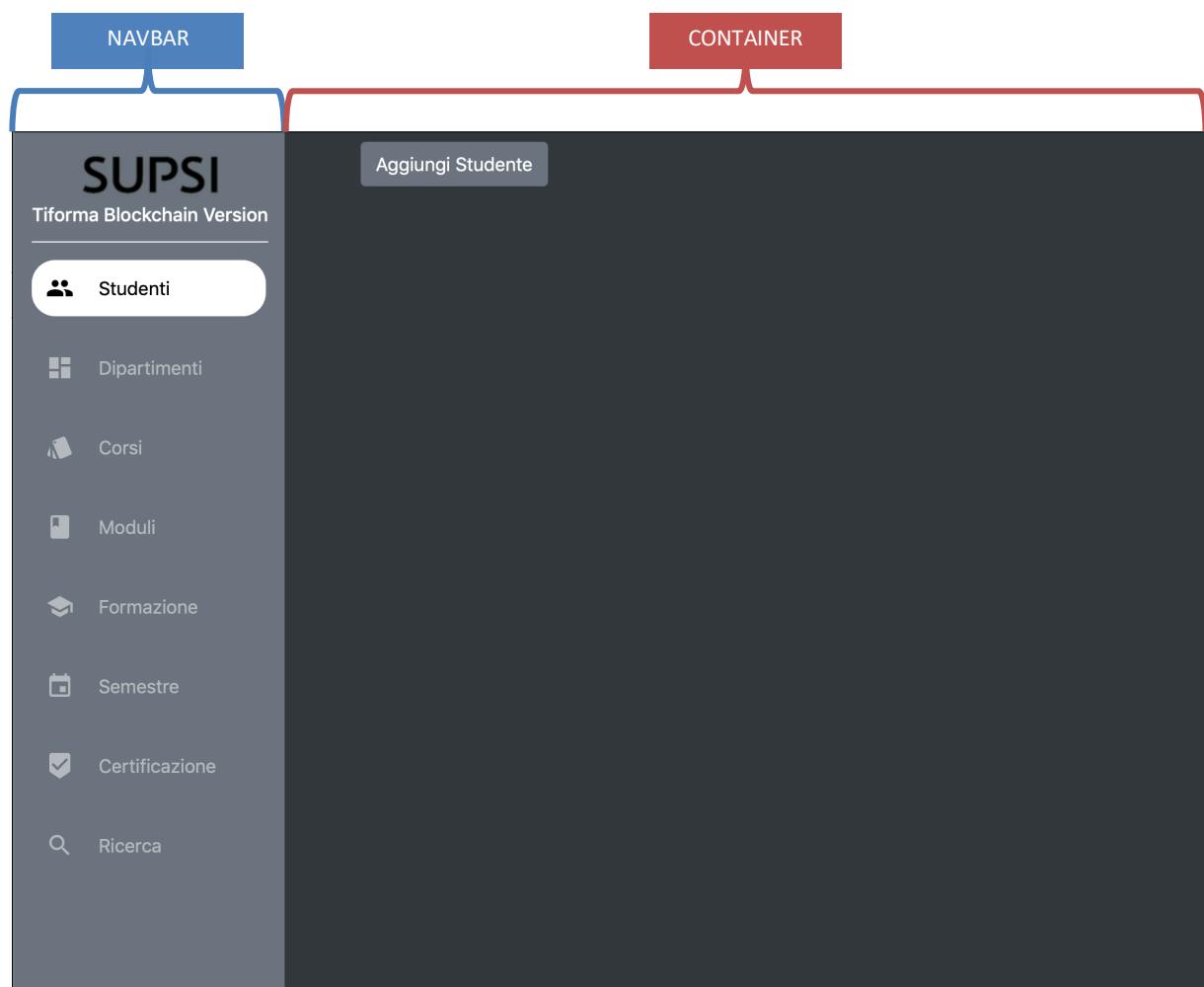
◀ app	
▶ certifications	Modulo per la gestione delle certificazioni.
▶ container	Componente che gestisce gli altri componenti.
▶ courses	Modulo per la gestione dei corsi.
▶ departments	Modulo per la gestione dei dipartimenti.
▶ formation	Modulo per la gestione delle formazioni.
▶ loading	Modulo contenente l'elemento di caricamento.
▶ modules	Modulo per la gestione dei moduli.
▶ navbar	Componente per la gestione della navbar.
▶ queries	Modulo per la gestione della pagina dei filtri.
▶ semesters	Modulo per la gestione del semestre.
▶ students	Moduli per la gestione degli studenti.
TS app-routing.module.ts	Contiene tutti i percorsi dell'applicazione.
# app.component.css	CSS associato al componente principale.
< app.component.html	HTML associato al componente principale.
TS app.component.spec.ts	File di test autogenerato.
TS app.component.ts	Componente principale.
TS app.module.ts	Modulo principale.
TS ch.supsi.ts	Contiene la struttura degli oggetti.
TS rest.service.ts	Contiene le chiamate REST per i filtri.
TS supsi.service.ts	Contiene le chiamate CRUD fatte dai moduli.

Figura 5 - Cartella App

## 6.2. Dashboard

L'applicazione, come mostra la *Figura 6*, è suddivisa in due componenti principali: la **navbar** e il **container**.

La **navbar**, sempre visibile sulla sinistra, si occupa di gestire la navigazione tra le pagine (**routing**). Il **container** è contestuale alla pagina scelta nella barra di navigazione.



**Figura 6 - Dashboard**

## 6.3. Navbar

All'interno del file che definisce l'HTML del componente navbar, *navbar.component.html*, oltre a tutti i tag HTML e alle classi CSS, sono presenti tutti i link per poter navigare tra le varie pagine dell'applicazione, come mostrato in *Figura 7*. Il tag **routerLinkActive** ci permette di sapere quale percorso è attivo e mostrarlo con uno stile CSS diverso.

```
<li class="nav-item"><a class="nav-link" [routerLinkActive]="['active']" routerLink="/students">
  <i class="material-icons">group</i><span>Studenti</span></a></li>
```

**Figura 7 - Link per pagina studenti**

### 6.3.1. Routing

Per la gestione della navigazione, si utilizza un modulo particolare di tipo routing. Il file in questione è l'*app-routing.module.ts*, che contiene tutti i percorsi per accedere ai vari componenti. Come visto precedentemente in *Figura 7*, tramite il tag **routerLink** definiamo quale percorso del routing andiamo a richiamare, infatti per lo **studente**, che è la pagina principale, va a caricare nel componente **Container** lo **StudentsModule**, come mostra la *Figura 8*.

```
const routes: Routes = [
  { path: '', redirectTo: '/students', pathMatch: 'full' },
  { path: 'students', loadChildren: './students/students.module#StudentsModule' },
  { path: 'departments', loadChildren: './departments/departments.module#DepartmentsModule' },
  { path: 'courses', loadChildren: './courses/courses.module#CoursesModule' },
  { path: 'modules', loadChildren: './modules/modules.module#ModulesModule' },
  { path: 'formations', loadChildren: './formation/formation.module#FormationModule' },
  { path: 'semesters', loadChildren: './semesters/semesters.module#SemestersModule' },
  { path: 'search', loadChildren: './queries/queries.module#QueriesModule' },
  { path: 'certifications', loadChildren: './certifications/certifications.module#CertificationsModule' }
];
```

**Figura 8 - Routes**

## 6.4. Container

Questo componente, come mostra la *Figura 9*, include nel suo HTML un tag chiamato **router-outlet**, che permette al **Routing** di caricare il componente selezionato dalla **Navbar**.

```
<router-outlet #routerOutlet="outlet"></router-outlet>
```

**Figura 9 - Tag nel Container**

Questo componente inoltre ha una particolarità, infatti nel CSS è presente un **padding** di **240px a sinistra**, cioè lo spazio che necessita la **Navbar** per poter essere visibile.

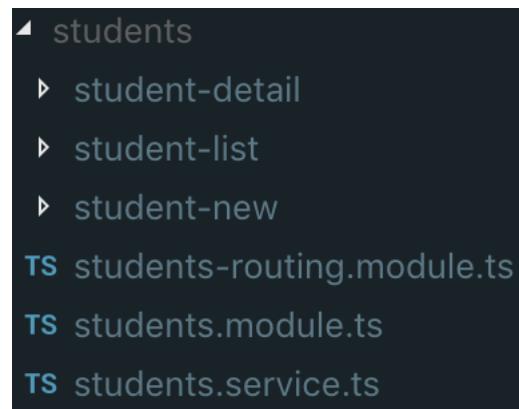
## 6.5. Moduli

La struttura delle pagine dei vari elementi del sistema sono tutte molto simili tra di loro. Infatti, possiamo trovare un bottone posizionato in alto per l'aggiunta dell'elemento, associato alla pagina come per esempio Studente o Dipartimento, e una lista per selezionare e modificare un elemento situato sulla parte destra:

- Se si è premuto il bottone di aggiunta mostrerà un form di inserimento.
- Se si è selezionato un elemento nella lista mostrerà un form di modifica.

**Figura 10 - Pagina Studenti**

Per la spiegazione dei moduli, visto la struttura similare che li accomuna, prendiamo come esempio il modulo Studente, *Figura 11*.



**Figura 11 - Modulo studente**

Ogni modulo è provvisto:

- Un modulo di **Routing**, che differisce da quello principale visto che permette la navigazione nel modulo in questione.
- Un service che permette di fare chiamate http richiamando il service principale.
- Tre componenti di base:
  - Dettaglio
  - Lista
  - Nuovo

### 6.5.1. Routing

Come la Navbar, anche i moduli che rappresentano le pagine contengono un modulo di **Routing** per la navigazione interna. Infatti, permette di accedere alla pagina di dettaglio o alla pagina per l'aggiunta di un nuovo elemento. Come mostra la *Figura 12*, abbiamo un percorso vuoto che sarebbe il componente della lista, mentre come figli di questo componente:

- **new** che permette di accedere al componente **StudentNewComponent**, che verrà caricato sulla sinistra e conterrà un form di input.
- **:id** che mostrerà un form di modifica e di visualizzazione.

```
const routes: Routes = [
  {
    path: '',
    component: StudentListComponent,
    children: [
      {
        path: 'new',
        component: StudentNewComponent
      },
      {
        path: ':id',
        component: StudentDetailComponent,
      }
    ]
};
```

Figura 12 - Routing dello Studente

### 6.5.2. Service

Ogni **Service** dei vari moduli si collegano direttamente con il **Service** principale denominato **supsi-service.ts**. Questo servizio permette di eseguire delle chiamate al web server generato lato back-end da Hyperledger.

Come prima variabile viene definito un endpoint, che è il link di base generato da Hyperledger per richiamare le varie API, poi nelle varie funzioni viene modificato a dipendenza dell'elemento che si vuole andare a modificare.

```
export class SUPSIService {
    private endpoint = 'http://localhost:3000/api/';

    constructor(private http: HttpClient) { }

    getElements(type: String) {
        return this.http.get(this.endpoint + type);
    }

    getElement(type: String, id) {
        return this.http.get(this.endpoint + type + '/' + id);
    }

    updateElement(type: String, id, elementData) {
        return this.http.put(this.endpoint + type + '/' + id, elementData);
    }

    deleteElement(type: String, id) {
        return this.http.delete(this.endpoint + type + '/' + id);
    }

    operationToElement(type: String, elementData) {
        return this.http.post(this.endpoint + type, elementData);
    }
}
```

**Figura 13 - SUPSI Service**

Mentre per quanto riguarda il servizio presente nei vari moduli, viene iniettato il **SUPSIService** e tramite altri metodi vengono chiamati i metodi di base. Come esempio nella *Figura 14* viene mostrato il servizio presente nello studente.

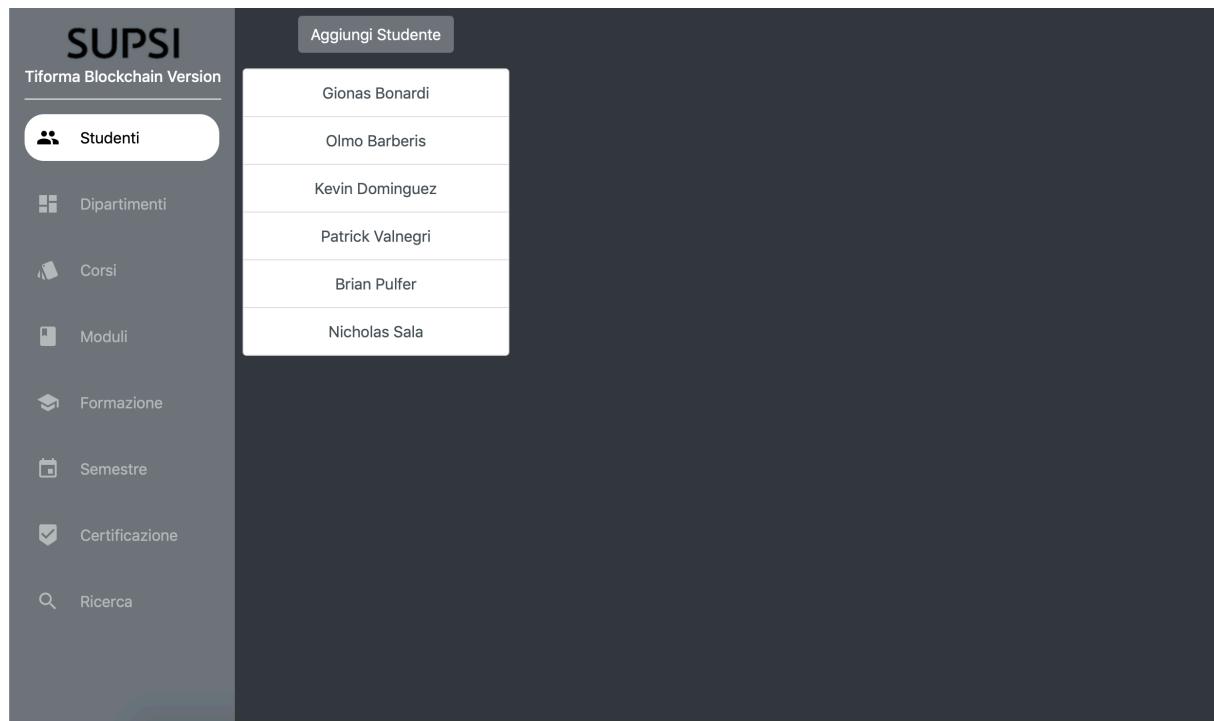
Un esempio di operazioni è quando viene caricato il componente lista dello studente: la lista viene riempita chiamando il metodo **getStudents()** dello **StudentsService** che tramite il **SUPSIService** crea il link <http://localhost:3000/api/Student>. Questa chiamata ritorna una lista di studenti sotto forma di file JSON che poi viene usata per creare la lista in HTML. I metodi che usano **operationToElement**, vanno a utilizzare le transazioni custom create nel back-end per operazioni CRUD.

```
export class StudentsService {  
  
    private base = "Student";  
  
    constructor(private supsiService: SUPSIService) { }  
  
    getStudents() {  
        return this.supsiService.getElements(this.base);  
    }  
  
    getStudent(id) {  
        return this.supsiService.getElement(this.base, id);  
    }  
  
    createStudent(studentData) {  
        return this.supsiService.operationToElement("Create" + this.base, studentData);  
    }  
  
    deleteStudent(studentData) {  
        return this.supsiService.operationToElement("Delete" + this.base, studentData);  
    }  
  
    updateStudent(studentData) {  
        return this.supsiService.operationToElement("Update" + this.base, studentData);  
    }  
}
```

**Figura 14 - StudentsService**

### 6.5.3. Lista

Ogni Modulo, come visto precedentemente contiene un componente che rappresenta una lista. Questo componente è quello principale infatti divide la pagina ancora in due. Come è possibile vedere dalla *Figura 15*, abbiamo una lista sulla sinistra e uno spazio lasciato libero per il form di creazione o di modifica.



**Figura 15 - Pagina Studente**

Quando premiamo su un elemento sulla lista viene caricata la pagina di modifica dell'elemento in questione, infatti tramite il **Routing** viene passato l'id dell'elemento, vengono recuperato le informazioni tramite chiamate al **Service** e mostrate a schermo per la modifica o la rimozione.

Come mostra la *Figura 16*, la struttura HTML è così composta:

- Un div contenente il bottone per creare un nuovo studente e la lista effettiva recuperata tramite il **Service**.
- Un div contenente il router-outlet che permette di caricare i componenti **student-new** o **student-detail**.

```
<div class="col-md-3 mt-2 text-center">
  <button routerLink="new" type="button" class="btn btn-secondary">
    Aggiungi Studente
  </button>
  <ul class="list-group mt-3" *ngIf="students">
    <li *ngFor="let p of students" class="list-group-item list-group-item-action" [routerLinkActive]=["active"]"
      routerLink="{{p.contactID}}">{{p.name + " " + p.surname}}</li>
  </ul>
</div>
<div class="col-md-9">
  <router-outlet></router-outlet>
</div>
```

Figura 16 - HTML del componente student-ist

Come visto nel capitolo 6.5.2 *Service*, quando viene caricato questo componente viene fatta una chiamata http la quale viene ritornata una lista sotto forma di JSON. Il componente si occupa di inserirla in un array di studenti, il quale viene ciclato nell'HTML e generata la lista con nome e cognome e nel link l'id.

Il bottone in cima, carica il componente **student-detail-new** che viene caricato nella colonna di destra. Mentre premendo su uno studente viene caricato il componente **student-detail-component**.

#### 6.5.4. Dettaglio

Il componente dettaglio, presente in ogni modulo ma qui viene mostrato solo quello dello studente, mostra le varie informazioni e permette la modifica o l'eliminazione. Come viene mostrato in *Figura 17*, viene generato un form di input già riempito con lo studente selezionato dalla lista. Queste informazioni vengono aggiunte tramite il service dello studente che attraverso una chiamata alle API di Hyperledger richiede le informazioni in formato JSON dello studente stesso.

The screenshot shows the SUPSI Blockchain Platform interface. On the left is a sidebar with navigation links: Studenti (highlighted), Dipartimenti, Corsi, Moduli, Formazione, Semestre, Certificazione (with a checked checkbox), and Ricerca. The main area has a title 'Aggiungi Studente' and a list of student names: Ivan Pavic, Gionas Bonardi (selected and highlighted in blue), Olmo Barberis, Kevin Dominguez, Elia Perrone, Tommaso Agnola, Patrick Valnegri, Brian Pulfer, and Nicholas Sala. To the right, a detailed view for 'Gionas Bonardi' is shown. The form fields include: Numero Matricola (16-682-081), Statuto (Immatricolato), Name (Gionas), Surname (Bonardi), Anno di Nascita (2019-05-16), Nazionalità (Swiss), Piano di studi (Ingegneria informatica TP), and Commento (empty). At the bottom are 'Salva' (Save) and 'Elimina' (Delete) buttons, and a small camera icon.

**Figura 17 - Pagina di dettaglio**

Tramite i due bottoni Salva e Elimina vengono richiamate due funzioni, rispettivamente updateStudent e deleteStudent, mostrati in *Figura 18*.

```
updateStudent() {
  this.loading = true;
  this.studentsService.updateStudent(this.studentData).subscribe((result) => {

    window.location.reload();
  }, (err) => {
    console.log(err);
  });
}

deleteStudent() {
  this.loading = true;
  this.studentsService.deleteStudent(this.studentDataToDelete).subscribe((result) => {

    window.location.reload();
  }, (err) => {
    console.log(err);
  });
}
```

**Figura 18 - Update e Delete Studente**

Per la spiegazione della variabile di loading, si legga il capitolo *6.5.6 Loading*.

Quando si aggiorna uno studente si richiama il Service che, tramite una chiamata asincrona, aggiornerà via transazione le informazioni dello studente. Una volta finito viene ricaricata la pagina. Stessa cosa co il metodo di eliminazione, una volta che viene eliminato uno studente la pagina viene ricaricata.

### 6.5.5. Nuovo

Il componente che permette l'inserimento ha gli stessi campi della modifica, ma ciò che lo differenzia sono il bottone di creazione e la struttura dietro il componente. Infatti se prima facevamo una richiesta di modifica al service dello studente, li passavamo un elemento JSON del tipo **ch.supsi.UpdateStudent**, mentre in questo caso viene passato un **ch.supsi.CreateStudent**.

Figura 19 - Nuovo Studente

Il metodo **createStudent** genera un **contactID** casuale e va a richiamare tramite il service, le API di Hyperledger passandoli l'oggetto della transazione Studente in formato JSON. Una volta creato viene ricaricata la pagina.

```
createStudent(){
  this.studentData.contactID = Math.random().toString(36).substring(2, 15) + Math.random().toString(36).substring(2, 15);
  this.loading = true;
  this.studentsService.createStudent(this.studentData).subscribe((result) => {
    window.location.reload();
  }, (err) => {
    console.log(err);
  });
}
```

Figura 20 - Create Studente

### 6.5.6. Loading

Ogni componente che richiama i metodi asincroni dei service presenti nei vari moduli, deve aspettare la risposta, in questo lasso di tempo l'utente non può eseguire nessuna operazione. Questa funzione viene gestita tramite una variabile loading booleana presente in ogni componente, che se è impostata vera mostra la rotellina di caricamento mentre se falsa è nascosta.

L'HTML e il CSS di questa rotellina di caricamento è situata in un modulo chiamato **loading**. Questo modulo viene poi importato in ogni singolo modulo presente nella cartella **app**. Per esempio, nei componenti `student-new-component` e `student-detail-component`, in fondo all'HTML è presente un tag nel quale viene caricato l'HTML del componente **loading**, *Figura 21*.

```
<app-loading *ngIf="loading"></app-loading>
```

Figura 21 - Tag del modulo Loading

## 7. Sviluppi futuri

---

Vi sono degli aspetti del progetto che si sarebbero voluti sviluppare durante l'arco del semestre ma che non sono stati implementati. In particolare, una migliore gestione delle queries, un pannello che dal dettaglio di uno studente mostra i moduli ai quali è iscritto e dei controlli sui valori che l'utente può inserire per ogni campo di ogni sezione.

### 7.1. Gestione delle queries

Al momento, il query language di Composer permette di trovare dei partecipanti o degli asset utilizzando gli operatori SELECT (unico operatore obbligatorio), FROM, WHERE, AND, OR, CONTAINS, ORDER BY.

Il linguaggio non consente di trovare degli elementi che contengono un'attributo di tipo stringa il quale è simile ad un parametro passato (operatore LIKE). Per questo motivo non è possibile sfruttando il query language, ad esempio, trovare tutti gli studenti il cui nome inizia per 'Ale', o tutti i moduli il cui codice termina per '002'.

Inoltre non è stato possibile utilizzare l'operatore FROM per selezionare un partecipante o asset a partire da un'altro partecipante o asset. Non è dunque possibile trovare un corso che sia contenuto in un modulo, o un modulo contenuto in un semestre. L'operatore CONTAINS permette invece di trovare il modulo che contiene un dato corso, o il semestre che contiene il dato modulo.

Per questi motivi, si è preferito gestire la questione delle queries lato front-end. L'applicazione lancia delle richieste al REST-Server per l'ottenimento di tutte le risorse del tipo cercato, e successivamente applica i filtri per finalmente mostrare a schermo solo i risultati coerenti.

Il problema di questo tipo di approccio è che al crescere dei dati, l'applicazione ne soffre in prestazioni specialmente su queries più complesse.

```
findStudentsByName(){
    let allStudents = [];
    this.restService.getAll('Student').subscribe(students => {
        allStudents = students;
        this.found = [];

        for(let i = 0; i<allStudents.length; i++){
            if(allStudents[i].name.includes(this.searchValue)){
                this.found.push(allStudents[i]);
            }
        }
    });
}
```

Figura 22 - Esempio di filtraggio dei dati

La soluzione a questo problema prevede sicuramente di utilizzare le queries offerte dal database che sono ottimizzate per avere performances migliori.

## 7.2. Pannello dei moduli

Nella schermata del dettaglio di uno studente, sarebbe interessante inserire un pannello che indichi tutti i moduli al quale lo studente è iscritto o delle ‘classi’ al quale lo studente partecipa.

## 7.3. Controllo sui campi

Al momento, l’applicazione non prevede un controllo sui dati inseriti dall’utente. È quindi possibile inserire dei dati non idonei ed incorretti in diversi campi.

Questo rappresenta sicuramente un aspetto da considerare per il futuro prossimo dello sviluppo del progetto. In particolare, bisognerebbe implementare:

- Studenti: controlli sul nome, cognome, data di nascita, numero di matricola;
- Dipartimenti: controlli sul nome;
- Corsi: controlli sul nome;
- Moduli: controlli sul nome, durata, ECTS e stato;
- Piani di studi: controlli sullo stato;
- Semestri: controlli sul nome;
- Certificazioni: controlli sul voto;

## 8. Conclusione

---

L'applicazione attuale è completa delle funzioni definite nell'obiettivo iniziale. È possibile gestire le informazioni sugli studenti con la sicurezza offerta da un sistema di persistenza dati basato su blockchain, il che rappresenta un miglioramento dell'attuale versione che utilizza un database centralizzato. Le funzionalità di base implementate in questo primo prototipo permettono un utilizzo basilare in confronto alla piattaforma Tiforma. In futuro bisognerà aggiungere altre funzionalità più avanzate e introdurre gli accessi al sistema.

Il progetto ci ha permesso di sperimentare con la tecnologia blockchain ed imparare ad utilizzare il framework Angular per la creazione dell'interfaccia grafica. Abbiamo inoltre approfondito le conoscenze in ambito dello sviluppo web, nello specifico con il linguaggio Javascript e TypeScript. Inoltre, è stato possibile approfondire la gestione delle dipendenze nei progetti web con il package manager npm e l'utilizzo di librerie esterne come Bootstrap.

## 9. Bibliografia

---

- **Hyperledger**
  - <https://www.hyperledger.org>
- **Hyperledger Composer**
  - <https://hyperledger.github.io/composer/latest/>
- **Angular 7**
  - <https://angular.io>
- **Bootstrap**
  - <https://getbootstrap.com>
- **Material Icons Guide**
  - <https://google.github.io/material-design-icons/>
- **NodeJS**
  - <https://nodejs.org/en/>
- **Manuale Tiforma**
  - **Tiforma - Funzionalità e utilizzo, Ester Tami e Sascha Matasci**

## 10. Ambiente di sviluppo

### 10.1. Installazione di Hyperledger Composer

Hyperledger composer è attualmente (aprile 2019) un framework disponibile unicamente per le piattaforme con sistema operativo MacOS ed Ubuntu. Presso il sito ufficiale del framework sono disponibili le guide all'installazione per entrambi i sistemi operativi. In questo documento viene trattata solo l'installazione per MacOS siccome è l'unico sistema sul quale Composer sia stato installato durante lo sviluppo.

Il primo passo, prevede l'installazione di **nvm** (Node Version Manager) tramite il seguente comando:

```
curl -o- https://raw.githubusercontent.com/creationix/nvm/v0.33.0/install.sh | bash
```

Lanciando questo comando, una finestra di pop-up compare a schermo richiedendo l'installazione di Xcode. Dopodiché è necessario aprire Xcode per l'installazione di pacchetti aggiuntivi.

Tornando al terminale, ora è possibile creare il proprio profilo bash con li comando

```
touch .bash_profile
```

Infine, eseguendo nuovamente il comando originale

```
curl -o- https://raw.githubusercontent.com/creationix/nvm/v0.33.0/install.sh | bash
```

Nvm dovrebbe essere correttamente installato sulla macchina. La versione di nvm è verificabile con il comando:

```
nvm --version
```

Il prossimo passo è l'installazione di Node versione 8.

L'installazione e l'utilizzo di questa versione di node sono possibili grazie ai comandi

```
nvm install v8  
nvm use 8
```

La versione di node è verificabile con il comando

```
node --version
```

Nel terzo step, si esegue l'installazione di Docker dal sito ufficiale. Per scaricare il file di installazione è necessario utilizzare un account o registrarne uno nuovo.

L'IDE suggerito dal sito ufficiale di Composer è **Visual Studio Code** siccome vi è una estensione apposita per il framework, anch'essa suggerita per l'installazione. L'estensione si trova facilmente digitando “*Hyperledger Composer*” nella barra di ricerca delle estensioni di VSCode.

Ora che nella macchina sono presenti node, Docker (in esecuzione avviando l'applicazione) e un IDE, possiamo installare Composer, i tool per il REST-Server, un tool per creare degli scaffoldings da combinare a Yeoman con i seguenti comandi:

```
npm install -g composer-cli@0.20
```

```
npm install -g composer-rest-server@0.20
```

```
npm install -g generator-hyperledger-composer@0.20
```

```
npm install -g yo
```

Il playground di Composer è un servizio eseguibile sulla propria macchina e navigabile su un browser. Permette di esercitare le proprie conoscenze sul framework verificando graficamente il funzionamento. Il playground è installabile con il comando

```
npm install -g composer-playground@0.20
```

L'installazione di Hyperledger Fabric avviene con il seguente comando:

```
mkdir ~/fabric-dev-servers && cd ~/fabric-dev-servers  
  
curl -O https://raw.githubusercontent.com/hyperledger/composer-tools/master/packages/fabric-dev-servers/fabric-dev-servers.tar.gz  
tar -xvf fabric-dev-servers.tar.gz
```

```
cd ~/fabric-dev-servers  
export FABRIC_VERSION=hlfv12  
. ./downloadFabric.sh
```

Il seguente comando esegue lo script di avvio che genera una PeerAdmin card. Questo script è eseguito solo una volta, dopodiché sarà possibile utilizzare la carta *PeerAdmin@hlfv1*.

```
cd ~/fabric-dev-servers
export FABRIC_VERSION=hlfv12
./startFabric.sh
./createPeerAdminCard.sh
```

Notare che l'esecuzione di fabric avviene con il comando

```
~/fabric-dev-servers/startFabric.sh
```

E può essere fermata con il comando

```
~/fabric-dev-servers/stopFabric.sh
```

A patto che Docker sia in esecuzione.

## 10.2. Sviluppo dell'applicazione

Lo sviluppo dell'intero applicativo comincia con il back-end e con la definizione della business network (BND). Possiamo utilizzare Yeoman per generare lo scaffolding della Business network.

```
yo hyperledger-composer:businessnetwork
```

Con l'esecuzione del comando sopra menzionato, vengono richiesti i seguenti parametri:

Nome della network (*supsi-tiforma*), descrizione, nome dell'autore, e-mail dell'autore, licenza (selezionare *Apache-2.0*) ed un namespace.

Dallo scaffolding, è poi finalmente possibile definire la business network andando a modificare i file .cto, .js, .acl, .qry.

Una volta finito con lo sviluppo dell'applicativo back-end, con Fabric in esecuzione, dalla directory del progetto è possibile generare il file .bna

```
composer archive create -t dir -n .
```

È poi possibile installare la stessa

```
composer network install --card PeerAdmin@hlfv1 --archiveFile supsi-tiforma@0.0.1.bna
```

Dove 0.0.1 è la versione del file bna. La versione del file è modificabile nel file *package.json* sotto il campo “**version**”. Ogni modifica alla Business Network comporta la modifica di versione e la generazione di un nuovo file .bna.

A questo punto, l'avvio della network avviene per mezzo del seguente comando:

```
composer network start --networkName supsi-tiforma --networkVersion 0.0.1 --  
networkAdmin admin --networkAdminEnrollSecret adminpw --card PeerAdmin@hlfv1 --file  
networkadmin.card
```

Ed è possibile importare la carta

```
composer card import --file networkadmin.card
```

Per testare il funzionamento della network, è possibile eseguire un ping

```
composer network ping --card admin@supsi-tiforma
```

Ed infine, il comando per generare il rest server

```
composer-rest-server
```

Con l'esecuzione di quest'ultimo comando, verranno chiesti i parametri del REST-Server. La carta utilizzata è **admin@supsi-edu**. Durante lo sviluppo dell'applicazione si sono accettate le impostazioni di default. Navigando con un browser web verso *localhost:3000* è possibile avere una visione generale del funzionamento del server ed effettuare delle prove.

### 10.3. Angular e Dipendenze

Per utilizzare Angular bisogna installare NodeJS successivamente installare tramite npm la CLI.

```
npm install -g @angular/cli
```

Le icone utilizzate nel progetto sono quelle offerte da Google Material, anch'esse installate tramite npm e incluse nel progetto.

```
npm install material-design-icons
```

Per la gestione dell'interfaccia grafica abbiamo optato per la libreria di Twitter: Bootstrap. Anch'essa installata tramite npm.

```
npm install bootstrap
```

## 11. Guida all'utilizzo

---

### 11.1. Introduzione

Questo documento ha lo scopo di illustrare e guidare all'utilizzo delle funzionalità offerte dall'applicativo *SUPSI Tiforma Blockchain Version*.

Il programma ha lo scopo di soddisfare principalmente il processo di formazione degli studenti della Scuola Universitaria Professionale della Svizzera Italiana basandosi su di una base di dati distribuita con il meccanismo blockchain, rimpiazzando il vecchio sistema (Tiforma V2).

Il processo di formazione è riassumibile in 3 fasi: iscrizione alla formazione, iscrizione ai semestri e quindi ai relativi moduli, iscrizione alle sessioni di certificazione.

## 11.2. Modo d'uso

### 11.2.1. Menu

Il menù principale è situato a sinistra ed è sempre visibile. Da questo menù è possibile, cliccando, navigare verso 8 diverse sezioni: Studenti, Dipartimenti, Corsi, Moduli, Formazione, Semestre, Certificazione, Ricerca.

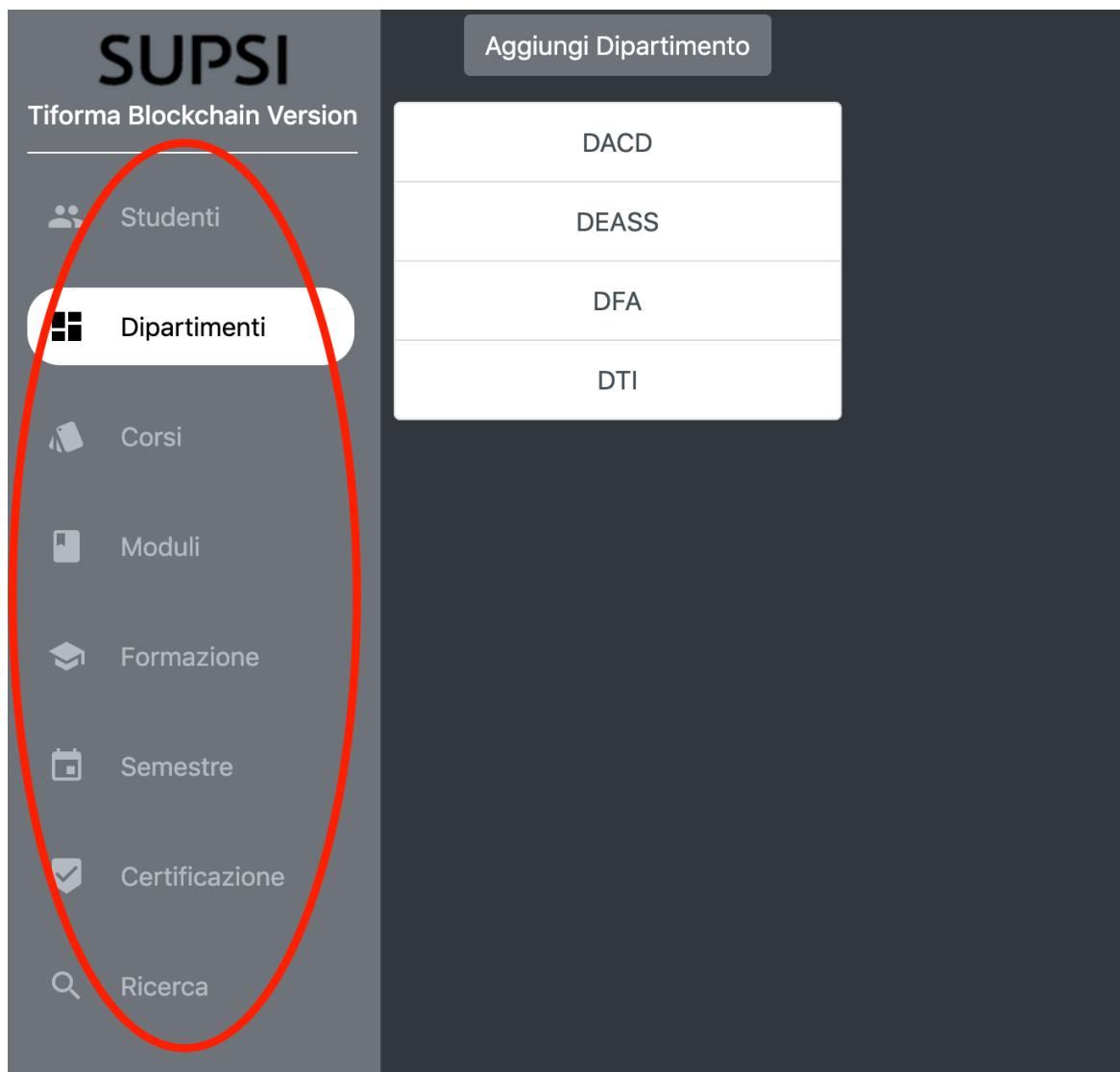


Figura 23 - Menu e pagina Dipartimenti

## 11.3. Sezioni

In ciascuna sezione, ad eccezione per quella di ricerca, è possibile creare, modificare, eliminare ed ottenere informazioni sugli elementi che la sezione tratta.

Le informazioni sull'elemento sono direttamente reperibili alla selezione dello stesso dalla lista. Queste stesse possono essere modificate direttamente ed essere salvate con il click sul pulsante verde "salva".

Le informazioni sull'elemento possono essere stampate cliccando sull'icona della stampante.

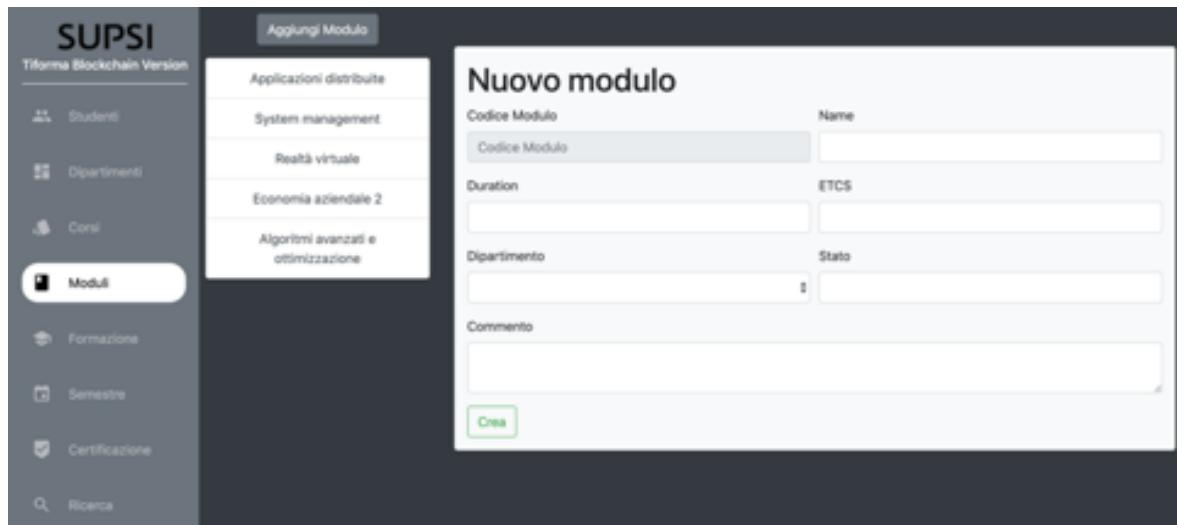
L'elemento può essere rimosso completamente dal database effettuando un click sul pulsante rosso elimina.

Un nuovo elemento può esser creato cliccando sul tasto di aggiunta presente sopra ogni lista di elementi.

**Figura 24 - Dettaglio in uno studente**

Nella Figura 24, il bottone segnalato dal cerchio 1 permette l'aggiunta di uno studente, il bottone segnalato con il cerchio numero 2 permette il salvataggio delle modifiche. Il bottone numero 3 permette invece l'eliminazione dei dati dello studente mentre il bottone numero 4 la stampa formattata di ciò che è mostrato a schermo.

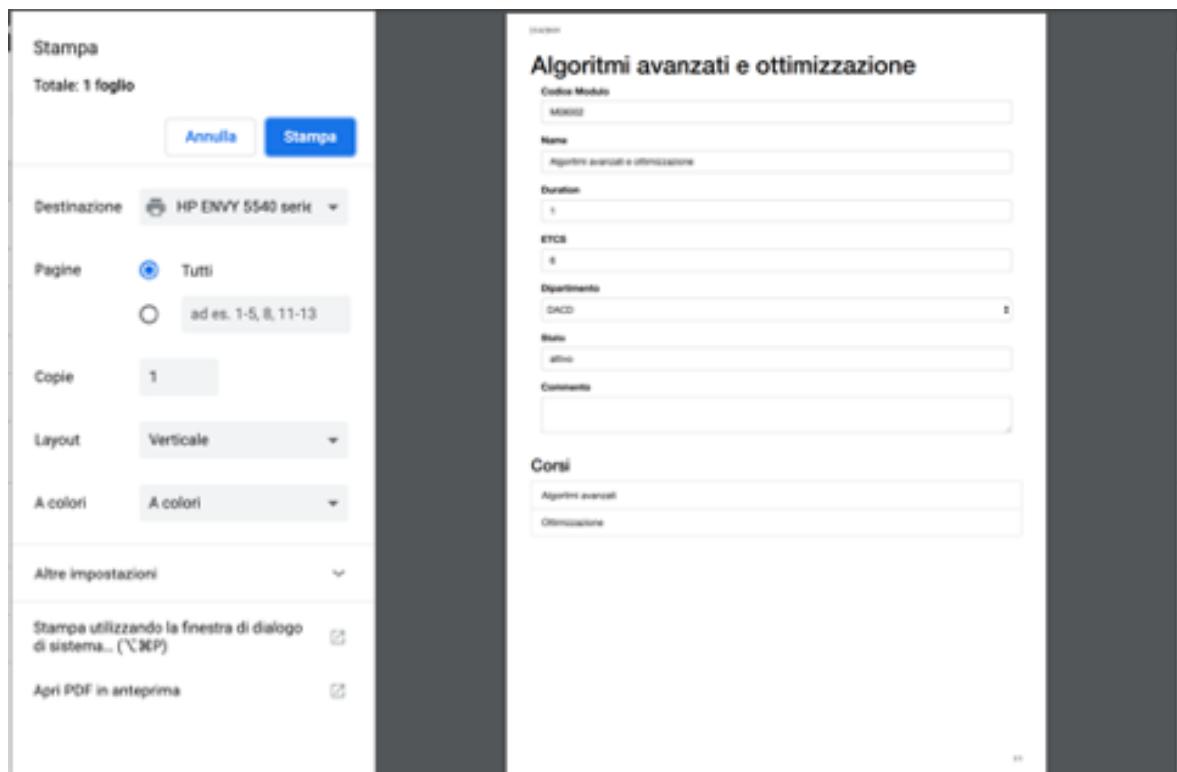
Cliccando sul pulsante di aggiunta (1), compare a schermo uno scheletro dati da riempire con i dati dell'elemento da aggiungere.



**Figura 25 - Scheda per creazione di un modulo**

Una volta riempiti tutti i campi necessari, premendo sul tasto verde “crea” l’istanza è generata e salvata sul database.

Cliccando invece sul pulsante di stampa (4), l’utente è rimandato alla schermata di stampa formattata dove può aggiustare le preferenze di stampa a piacere.



**Figura 26 - Anteprima di stampa di un modulo**

### **11.3.1. Studenti**

Per gli studenti è possibile operare sui seguenti dati: nome, cognome, numero di matricola, statuto (Immatricolato, Mai immatricolato, Exmatricolato, ospite), la data di nascita, nazionalità, piano di studi e commento.

### **11.3.2. Dipartimenti**

I dipartimenti sono formati esclusivamente dal loro nome.

### **11.3.3. Corsi**

Un corso è formato da un codice di corso ed un nome.

### **11.3.4. Moduli**

Un modulo è formato da un codice di modulo, un nome, una durata (in semestri), una quantità di ECTS assegnati agli studenti che certificano il modulo, un dipartimento di appartenenza, uno stato, un commento ed un insieme di corsi con i quali il modulo si compone.

### **11.3.5. Formazione**

In questa sezione vengono trattati i piani di studio. Un piano di studio è composto da un dipartimento di appartenenza, uno stato, un commento ed un insieme di moduli dei quali il piano di studio si compone.

### **11.3.6. Semestre**

Un semestre contiene informazioni relative al nome, una descrizione ed un insieme di moduli ai quali è correlata pure l'informazione relativa agli studenti iscritti a tali moduli.

### **11.3.7. Certificazione**

Una certificazione è un dato che associa uno studente ad un voto in un modulo. La certificazione è composta da uno studente, un modulo ed un voto decimale che va' da un minimo di 1 ad un massimo di 6. **Nota:** È possibile creare delle certificazioni per uno studente

solo se lo studente contiene il modulo nel suo piano di studio e solo se lo studente è iscritto a tale modulo.

### 11.3.8. Ricerca

Questa particolare sezione si distingue dalle altre per il fatto che il suo scopo sia quello di trovare degli elementi in base a criteri ben specifici.

Di seguito, una tabella che riassume quali elementi sono individuabili (prima colonna) in base a quali parametri (prima riga).

	Nome	Cognome	Nr. Matricola	Codice	Codice modulo presente	Nome studente	Cognome studente
<b>Studenti</b>	X	X	X				
<b>Dipartimenti</b>	X						
<b>Corsi</b>	X			X			
<b>Moduli</b>	X			X			
<b>Formazioni</b>	X				X		
<b>Semestri</b>	X				X		
<b>Certificazioni</b>					X	X	X

Tabella 1 - Possibili ricerche

Sotto la voce *tipo* è possibile selezionare il tipo dell'elemento ricercato, mentre sotto la voce *filtro* è invece possibile scegliere il criterio di ricerca. Inserendo poi il valore del criterio sotto la voce *valore* e cliccando sul pulsante blu *Cerca*, si ottiene una lista di elementi che rispettano i parametri di ricerca.

Cliccando su un elemento della lista, si viene re-indirizzati al dettagli dell'elemento.

The screenshot displays the SUPSI Blockchain Platform's search interface and its results. On the left, a sidebar lists navigation options: Studenti, Dipartimenti, Corsi, Moduli, Formazione, Semestre, Certificazione, and Ricerca. The main search area has dropdowns for 'Tipo' (set to 'Semestri') and 'Filtro' (set to 'Codice di un Modulo presente'), and a text input 'Valore' containing 'MD8002'. A blue 'Cerca' button is to the right. Below this, a search result for 'Semestre Primaverile 2019 - Informatica TP' is shown. A red arrow points from the search bar to this result. The result page includes tabs for 'SUPSI' (selected) and 'Aggiungi semestre'. It shows the semester details: Name 'Semestre Primaverile 2019 - Informatica TP', Description 'Semestre primaverile 2019 per gli studenti di Ingegneria Informatica in tempo pieno del terzo anno.', and two buttons: 'Salva' (green) and 'Elimina' (red). Below, a section titled 'Aggiungi moduli' lists five modules with red status indicators: 'Applicazioni distribuite', 'Sistemi Management', 'Realtà virtuale', 'Economia aziendale', and 'Algoritmi avanzati e ottimizzazione'.

Figura 27 - Sezione ricerca e risultato