Allegato tecnico

Sons Of SWE - Progetto Marvin

sonsofswe.swe@gmail.com

Informazioni sul documento

Uso	Esterno
Distribuzione	Vardanega Tullio Cardin Riccardo Gruppo Sons Of SWE

Descrizione

Documento che ha lo scopo di illustrare la Product Baseline, ponendo attenzione alle scelte architetturali e alla copertura di use case e requisiti funzionali.

Indice

1 Introduzione **4**
1.1 Scopo del documento 4
1.2 Scopo del prodotto 4
1.3 Glossario 4
1.4 Riferimenti 4
1.4.1 Normativi 4
1.4.2 Informativi 4

2 Requisiti di sistema **5**

3 Installazione ed esecuzione **6**

4 Confronto con la Poc **8**

5 Architettura del prodotto **9**
5.1 View 10
5.1.1 Design patterns 10
5.1.2 Diagramma delle classi 10
5.2 ViewModel 10
5.2.1 Design patterns 10
5.2.2 Diagramma delle classi 10
5.3 Model 13
5.3.1 Architettura 13
5.3.2 Diagramma delle classi 13
5.4 Diagrammi di sequenza 15
5.4.1 Login 15
5.4.2 Inserimento di un utente 15
5.4.3 Inserimento di un anno accademico 15

6 Requisiti soddisfatti **17**
6.1 Tabella del soddisfacimento dei requisiti 17
6.2 Grafici sui requisiti soddisfatti 18

7 Casi d'uso soddisfatti **19**
7.1 Tabella del soddisfacimento dei casi d'uso 19
7.2 Grafici sui casi d'uso soddisfatti 20

Elenco delle figure

1 Setta l'RPC inserendo **http://localhost:9545** 6
2 Clicca su **Import Existing DEN** 7
3 Inserisci la seed phrase e la password che vuoi usare 7
4 Diagramma generale dei package 9
5 Diagramma delle classi del componente View 11
6 Diagramma delle classi del componente ViewModel 12
7 Diagramma delle classi del componente Model 14
8 Diagramma di sequenza del processo di login 15
9 Diagramma di sequenza del processo di inserimento di un utente 16
10 Diagramma di sequenza del processo di inserimento di un anno accademico 16
11 Requisiti soddisfatti 18
12 Requisiti obbligatori soddisfatti 18
13 Casi d'uso soddisfatti 20

Elenco delle tabelle

1 Confronto tra Proof of Concept e Product Baseline 8
2 Requisiti soddisfatti 17
3 Casi d'uso soddisfatti 19

1 Introduzione

1.1 Scopo del documento

Questo documento ha lo scopo di descrivere gli obiettivi di qualità, di processo e di prodotto da raggiungere nella realizzazione del progetto e le strategie di verifica e validazione adottate per il raggiungimento di tali obiettivi.

1.2 Scopo del prodotto

Lo scopo del prodotto è quello di realizzare un *prototipo*$_{\text{G}}$ di Uniweb come una *ĐApp*$_{\text{G}}$ in esecuzione su *Ethereum*$_{\text{G}}$. I cinque attori principali che si rapportano con Marvin sono:

- Utente non autenticato;
- Università;
- Amministratore;
- Professore;
- Studente.

Il portale deve quindi permettere agli studenti di accedere alle informazioni riguardanti le loro carriere universitarie, di iscriversi agli esami, di accettare o rifiutare voti e di poter vedere il loro libretto universitario. Ai professori deve invece essere permesso di registrare i voti degli studenti. L'università ogni anno crea una serie di corsi di laurea rivolti a studenti, dove ognuno di essi comprende un elenco di esami disponibili per anno accademico. Ogni esame ha un argomento, un numero di crediti e un professore associato. Gli studenti si iscrivono ad un corso di laurea e tramite il libretto elettronico mantengono traccia ufficiale del progresso.

1.3 Glossario

Nel documento *Glossario_v1.0.0* i termini tecnici, gli acronimi e le abbreviazioni sono definiti in modo chiaro e conciso, in modo tale da evitare ambiguità e massimizzare la comprensione dei documenti.
I vocaboli presenti in esso saranno posti in corsivo e presenteranno una "G" maiuscola a pedice.

1.4 Riferimenti

1.4.1 Normativi

- **Capitolato d'appalto C6 - Marvin: dimostratore di Uniweb su Ethereum** http://www.math.unipd.it/~tullio/IS-1/2017/Progetto/C6.pdf;
- ***NormeDiProgetto_v4.0.0***;

1.4.2 Informativi

- ***AnalisiDeiRequisiti_v4.0.0***;

2 Requisiti di sistema

Per l'installazione e l'utilizzo di questo software sono richiesti alcuni prerequisiti:

- Browser web Google Chrome (aggiornato alla versione 60 o superiori) o Mozilla Firefox (aggiornato alla versione 50 o superiori);
- Plugin Metamask (aggiornato alla versione 4.7.1 o superiori) per i browser di cui sopra;
  https://metamask.io/
- Git
  https://git-scm.com/downloads
- Python aggiornato alla versione 2.7;
  https://www.python.org/downloads/
- Node package manager alla versione 6, e Node alla 8.11.2
  https://nodejs.org/it/
- Nel caso si utilizzi Windows sarà necessario installare *windows-build-tools* digitando nella powershell:

```
1     npm install --global --production windows-build-tools
```

3 Installazione ed esecuzione

Il codice relativo alla Product Baseline lo si può trovare al seguente link:

linkAllaRepo

Una volta fatto il clone della repository o dopo aver scaricato lo zip, sono necessari i seguenti passi per far partire l'applicazione:

1. Posizionarsi nella root della repo ed eseguire nella shell:

```
npm install -g ganache-cli
npm install -g truffle
npm i
```

2. In seguito sempre nella shell:

```
./startBlockchain.ps1
```

3. Infine è necessario eseguire:

```
./loadProject.ps1
```

A questo punto noterai che il tuo browser predefinito ha aperto automaticamente l'homepage. Ora dovrai connetteri a Metamask: nel tuo browser clicca sull'icona di Metamask e accetta l'informativa sulla privacy e le condizioni d'uso. Poi clicca su **Main Network** e scegli **Custom RPC**, inserisci nel primo form **http://localhost:9545** come in Figura 1 e clicca su **Save**.

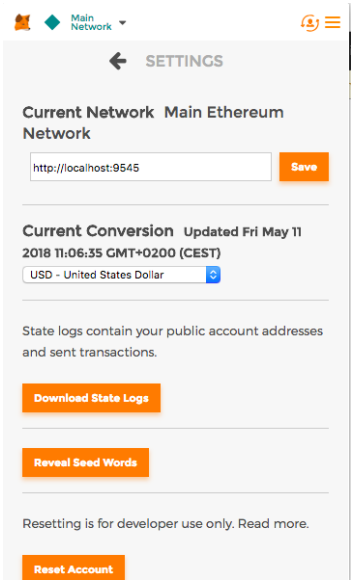

Figura 1: Setta l'RPC inserendo **http://localhost:9545**

Ora, come in Figura 2, clicca su **Import Existing DEN** e (vedi Figura 3) inserisci la frase **candy maple cake sugar pudding cream honey rich smooth crumble sweet treat** e la password che vuoi usare per il tuo account.

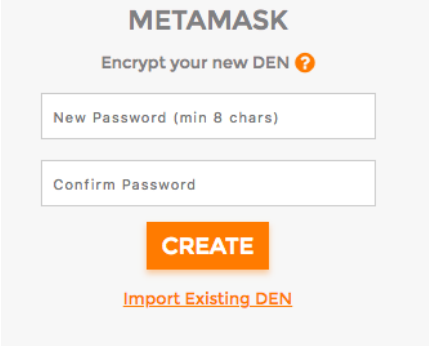

Figura 2: Clicca su **Import Existing DEN**

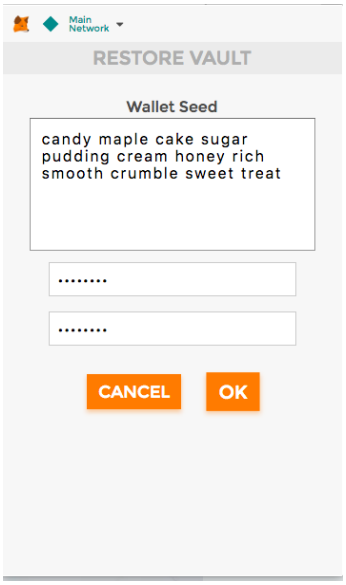

Figura 3: Inserisci la seed phrase e la password che vuoi usare

4 Confronto con la Poc

In precedenza è stato realizzata una *Proof of Concept* in cui si è data dimostrazione delle tecnologie che si sarebbero dovute utilizzare per la realizzazione del progetto Marvin. Tale produzione è disponibile all'indirizzo: https://github.com/SOS-SonsOfSwe/Marvin-PoC. Durante la ricerca di strumenti adatti allo scopo il team ha deciso inizialmente di basare la prima produzione del prodotto sul framework *Truffle*, dal momento che esso presentava molte delle tecnologie di interesse. Con uno studio più approfondito del pacchetto il gruppo si è reso conto che esso era una buona base di partenza per la realizzazione di un prodotto più complesso, sebbene necessitasse di alcune correzioni strutturali. Qui di seguito si elencano i limiti riscontrati e le soluzioni adottate:

Proof of Concept	**Product Baseline**
Architettura ben strutturata ma male incapsulata	Organizzazione migliore della struttura del pacchetto a fronte di una conoscenza più approfondita del design pattern MVVM e di altri su cui il framework si basa
Interfaccia povera e strutturata in maniera confusionaria, non chiara e difficilmente intuibile	Ristrutturazione completa e incremento dell'interfaccia utente, ora provvista di tutte le parti atte alla soddisfazione della maggior parte dei requisiti opzionali e obbligatori
Database solidity estremamente scarno e con poche funzionalità di sicurezza e di ottimizzazione	Strutturazione avanzata del backend di solidity, desisamente più ottimizzato e performante; implementazione già in fase di completamento

Tabella 1: Confronto tra Proof of Concept e Product Baseline

5 Architettura del prodotto

Dopo un approfondito studio il team ha optato per l'utilizzo del design pattern **Model View View-Model**, che prevede tre macrosezioni:

- **Model**: rappresenta i dati contenuti nel sito, ma non i comportamenti o i servizi che manipolano l'informazione. Non è responsabile della renderizzazione;
- **View**: si occupa di rappresentare le informazioni contenute nel sito ed è quello con cui l'utente interagisce; la view in questo design pattern è attiva, al contrario di quello che succede nell'MVC, questo perché contiene comportamenti, eventi e riferimenti a stati del sito, che quindi presuppongo una conoscenza della logica che sta dietro ai dati;
- **Viewmodel**: fornisce i dati dal Model in una forma in cui la View può usufruirne. Si occupa inoltre della logica della vista e di mantenersi costantemente sincronizzato con la View.

Per poter apprezzare questa suddivisione nel pacchetto del team viene riportato qui di seguito il diagramma generale dei package. Si proseguirà successivamente alla descrizione delle implementazioni di ogni sezione in relazione all'architettura della Product Baseline, illustrandone i rispettivi design pattern utilizzati.
Per facilitare la comprensione della struttura dei diagrammi delle classi, il team ha deciso di raggruppare tutti i *Containers*, i *Components* e le *actions* nei loro rispettivi packages. Infatti il comportamento di ogni classe di ogni package è modulare:

- Tutti i *Components* importano la classe *React.Component*
- Tutti i *Containers* importano il rispettivo *Component*, la rispettiva *Action* e il pacchetto *react-redux*.
- Tutte le *actions* importano le *costants* che azionano il *reducer*, il pacchetto *react-router*, lo *store*, il pacchetto *truffle-contract* e la classe *IpfsUtils*

I diagrammi presentati in questo documento illustrano la struttura sinteticamente; la loro versione integrale si può trovare nella cartella Diagrammi.

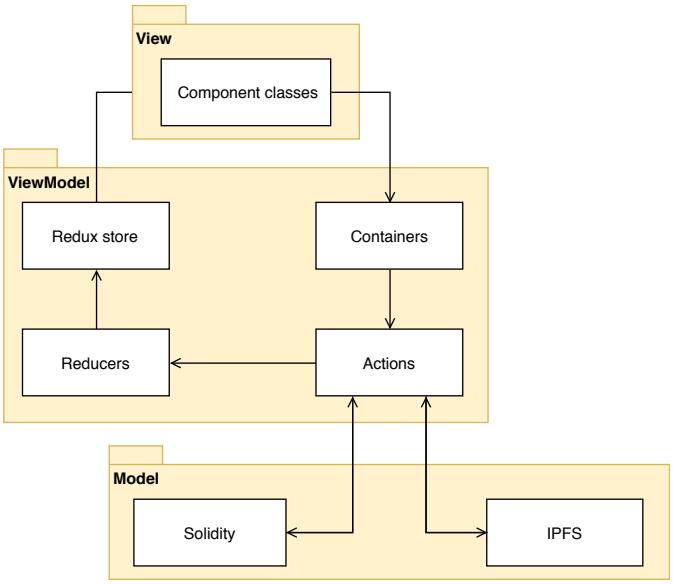

Figura 4: Diagramma generale dei package

5.1 View

5.1.1 Design patterns

Progettando questa sezione ci si è resi conto che si sarebbero potute generare delle classi *dumb*, ovvero slegate dalla logica del sistema e che si sarebbero dovute occupare solamente della rappresentazione delle informazioni. Per ottenere questo si è ricorso all'utilizzo del seguente design pattern:

- **Decorator**: sfruttando il macro-package *Container*, che poi si occupa di collegare il componente allo stato di *redux*, si possono decorare tutte le componenti ivi presenti con dei componenti react puri, ai quali vengono passate eventualmente le informazioni e le funzioni a disposizione attraverso un accesso al *this.props*. In questa maniera si ha la completa separazione tra rappresentazione e dati - cosa auspicata dal framework MVVM - e si facilitano modifiche future.

5.1.2 Diagramma delle classi

In Figura 5 è riportato il diagramma delle classi relativo a questa sezione con evidenziato la posizione del design pattern utilizzato.

5.2 ViewModel

5.2.1 Design patterns

Essendo una tra le sezioni più complesse da gestire e quella che poi si sarebbe dovuta occupare della gestione degli stati dell'applicazione, si è deciso di seguire in parte i design pattern offerti dal framework sfruttato e di aggiungerne altri in seguito ad un'analisi più profonda del problema. In Figura 6 si è ricorso all'utilizzo dei seguenti design pattern:

- **Observer**: implementato già nel framework offerto da Truffle, si occupa di gestire gli stati dell'applicazione sfruttando il framework *react-redux*. Il team lo ha declinato sfruttando la classe *index* come *observer* e la classe *store* come subject. Lo *store* viene aggiornato ogni qual volta si registra un cambiamento nello *state* dell'applicazione, grazie agli strumenti offerti da *redux*; questo aziona l'update dell'*index*, che quindi si occupa di eseguire un render a schermo aggiornando le informazioni cambiate;
- **Façade**: sfruttato in maniera estensiva, questo pattern ha permesso al team di generare interfacce per semplificare gli import all'interno di classi generali (vedasi, ad esempio, l'import di *index* che deve importare tutte le componenti);
- **Adapter**:
- **Adapter**: per interagire con i contratti *solidity truffle* ha sfruttato alcune componenti come *adapter*: la classe *migrate*, in particolare, si occupa di importare i contratti *solidity*, di farne il *deploy* all'interno della blockchain istanziata e poterli sfruttare come dei più leggibili *json*. Dopodiché la classe *truffle-contract* offre dei metodi per arricchire le istanze dei contratti ottenuti dal deploy con informazioni utili al loro sfruttamento nell'applicazione.

5.2.2 Diagramma delle classi

In seguito è riportato il diagramma delle classi relativo a questa sezione con evidenziato la posizione del design pattern utilizzato.

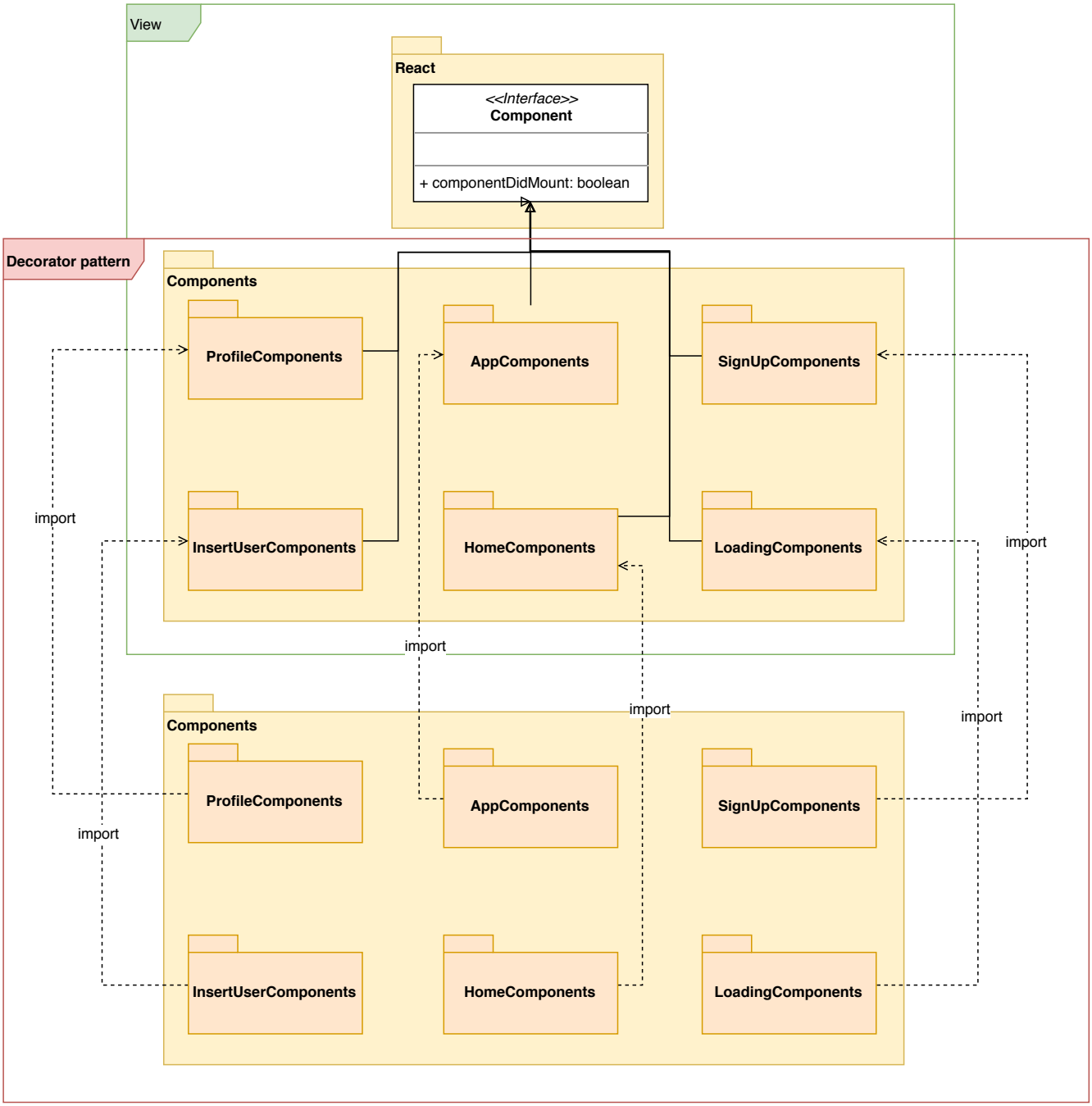

Figura 5: Diagramma delle classi del componente View

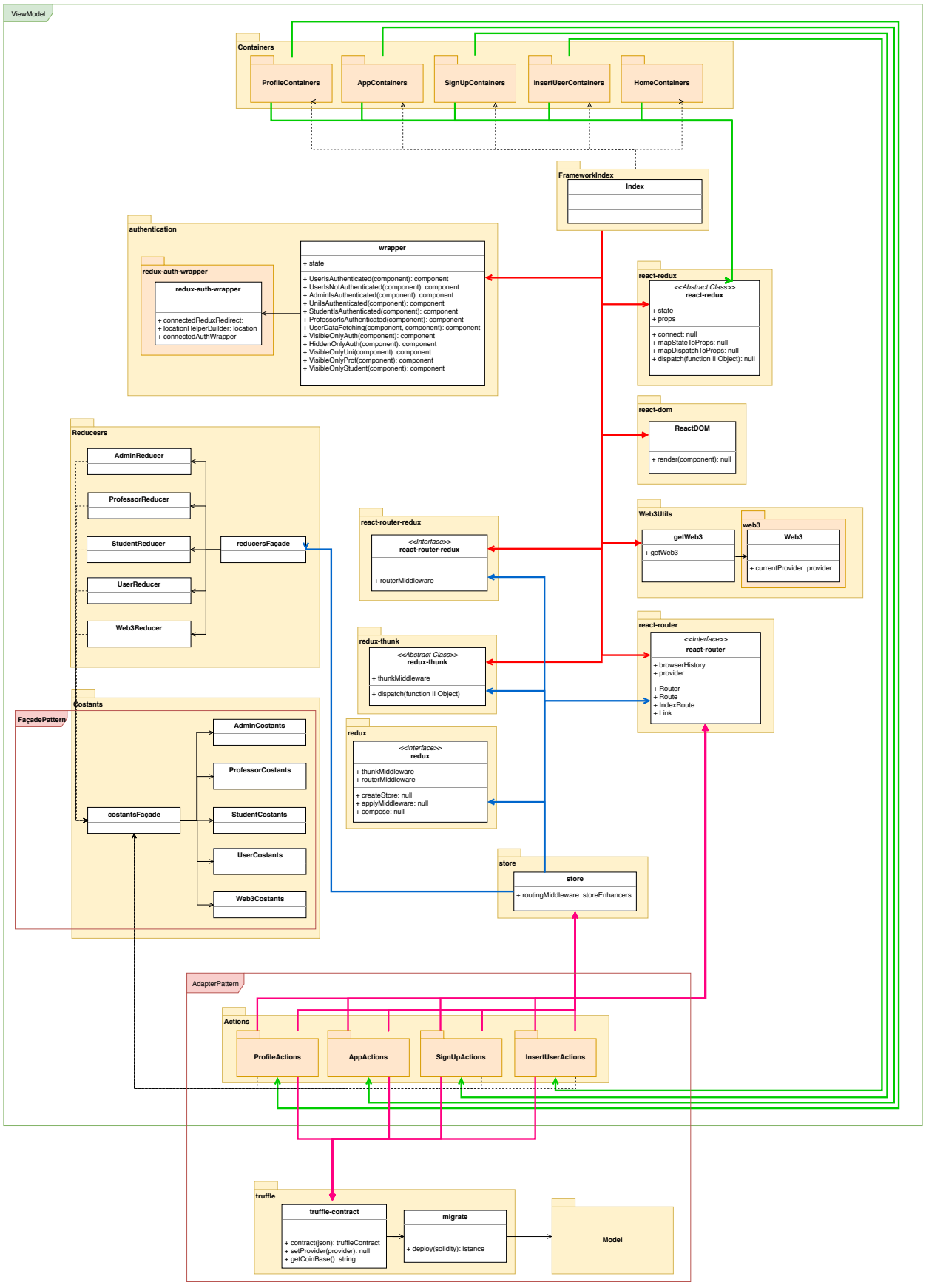

Figura 6: Diagramma delle classi del componente ViewModel

5.3 Model

5.3.1 Architettura

Per quanto riguarda la macrosezione relativa alla componente Model, composta dai contracts salvati su blockchain ed IPFS, si sono effettuate le seguenti scelte progettuali:

- **Modularità**: dividendo la parte logica e la struttura dei dati, in caso di aggiornamento della parte logica e quindi di un nuovo deploy dei contracts interessati, si ottiene la persistenza dei dati e un minor costo per il nuovo deploy. Infatti, se i dati e la logica fossero salvati nello stesso contracts, ad un nuovo deploy, i dati non sarebbero più consistenti, essendo contenuti in un altro contratto;
- **Istanze di contracts**: ogni contracts ha un'unica istanza: se i contracts contengono oggetti con molteplicità, essi saranno strutturati in una struct e si salveranno in un mapping con una key che li identifica ed una value che punta all'istanza della struct interessata. Si salvano, inoltre, per ogni mapping, un array contenente le key dei mapping. Questo perchè in Solidity (il linguaggio utilizzato per programmare i contracts), non è possibile in alcun modo recuperare quali key sono già salvate nel mapping e quindi si utilizzano gli array collegati per iterare sui dati;
- **Denormalizzazione**: come per altri database di tipo NoSQL e contrariamente ai database relazionali, si deve mantenere un compromesso fra ridondanza dei dati e costo del salvataggio maggiore. Infatti, per ottenere delle query più semplici e veloci, è necessario mantenere la ridondanza dei campi dati utilizzati più spesso. Questo si traduce in una maggior complessità per la modifica e l'eliminazione dei dati, ma si ottengono vantaggi significativi in caso di iterazioni sui dati non dovendo effettuare chiamate a contracts quando una transazione lo richiede e quindi richiedendo un costo minore;
- **IPFS**: il costo del salvataggio dei dati su blockchain Ethereum è molto elevato. Per questo motivo, alcuni dati sono salvati off-chain su un altro sistema distribuito, ovvero IPFS. La scelta di salvare un dato su blockchain Ethereum o su IPFS segue quindi queste regole:
  1. I dati utilizzati come identificativo per le query sono salvati su blockchain per evitare la latenza dovuta alla comunicazione fra backend-frontend-IPFS;
  2. I dati soggetti a problemi di concorrenza sono salvati su blockchain: in questo modo non serve porsi il problema dell'accesso concorrenziale per la modifica del dato da parte di molteplici utenti in contemporanea;
  3. Tutti i dati considerati "accessori" possono essere salvati su IPFS.

5.3.2 Diagramma delle classi

In seguito è riportato il diagramma delle classi relativo a questa sezione con evidenziato la posizione del design pattern utilizzato.

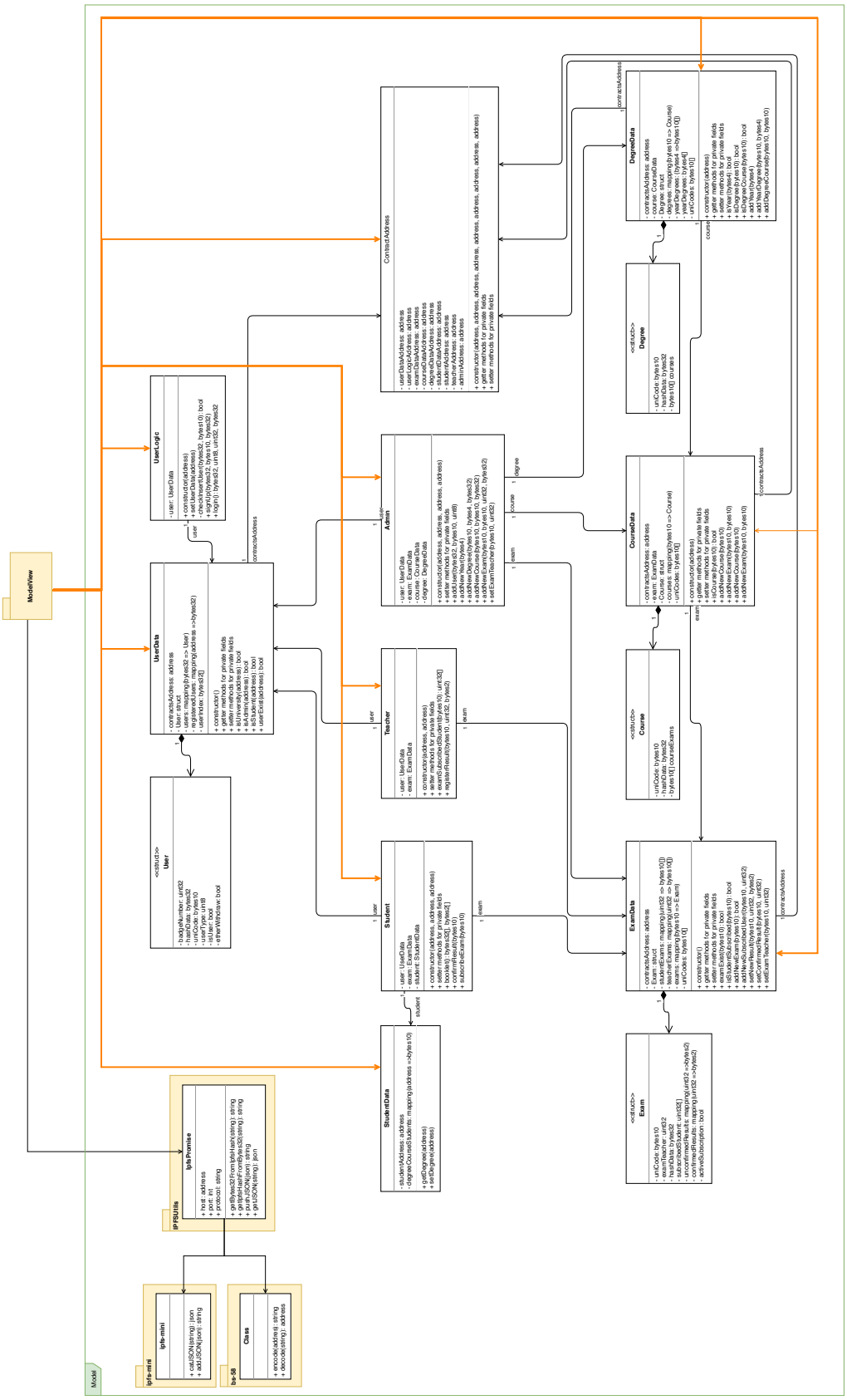

Figura 7: Diagramma delle classi del componente Model

5.4 Diagrammi di sequenza

Come per i diagrammi delle classi, sono stati redatti i seguenti diagrammi di sequenza, disponibili e meglio visualizzabili nella cartella Diagrammi;

5.4.1 Login

Il diagramma di sequenza riportato qui di seguito raffigura il processo di login, durante il quale l'utente che vuole accedere può essere autenticato dal sistema.

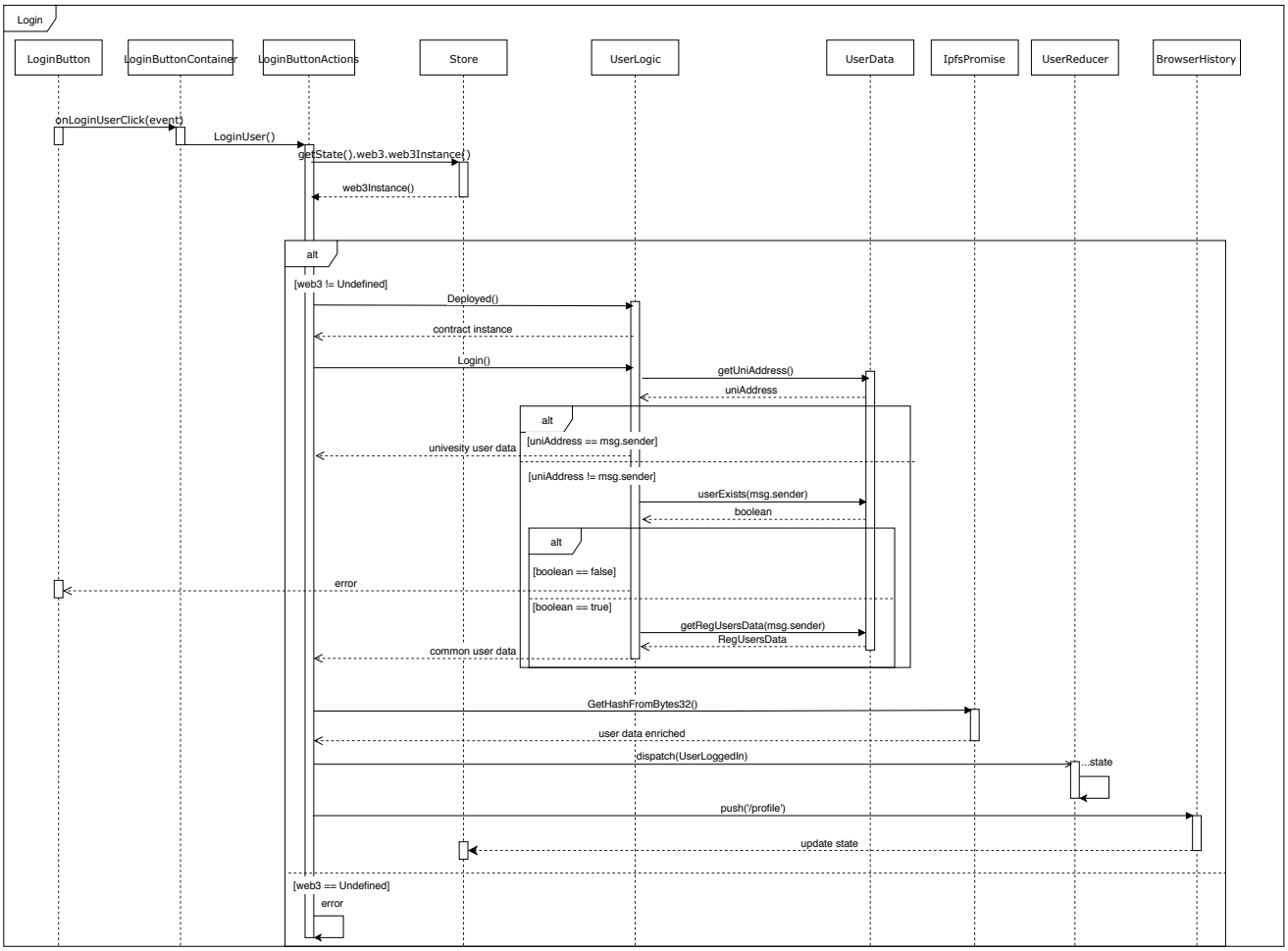

Figura 8: Diagramma di sequenza del processo di login

5.4.2 Inserimento di un utente

Il diagramma di sequenza in Figura 9 rappresenta l'azione di inserimento di un utente nel sistema.

5.4.3 Inserimento di un anno accademico

Il diagramma di sequenza riportato in Figura 10 raffigura il processo di inserimento di un nuovo anno accademico nel sistema.

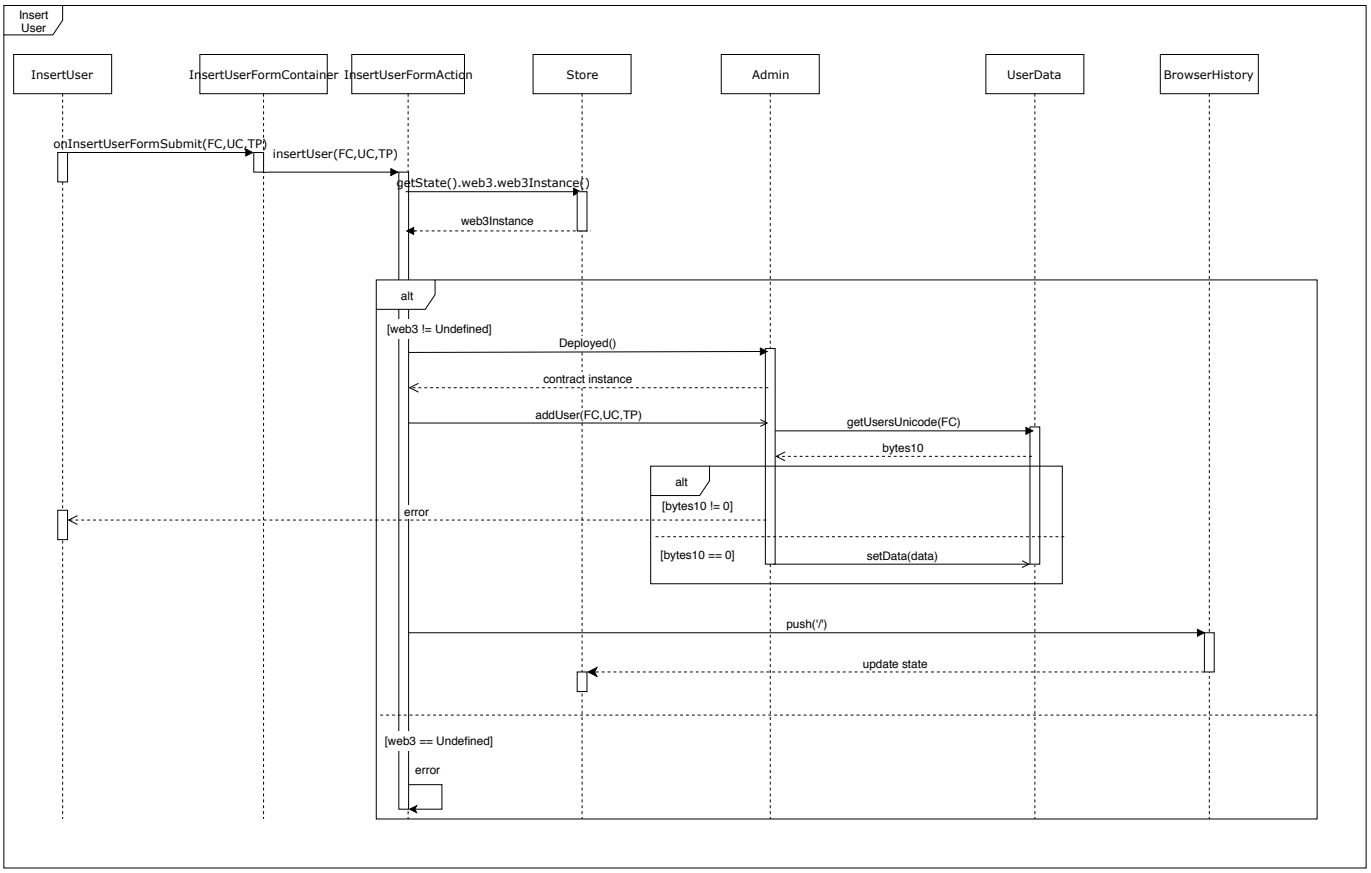

Figura 9: Diagramma di sequenza del processo di inserimento di un utente

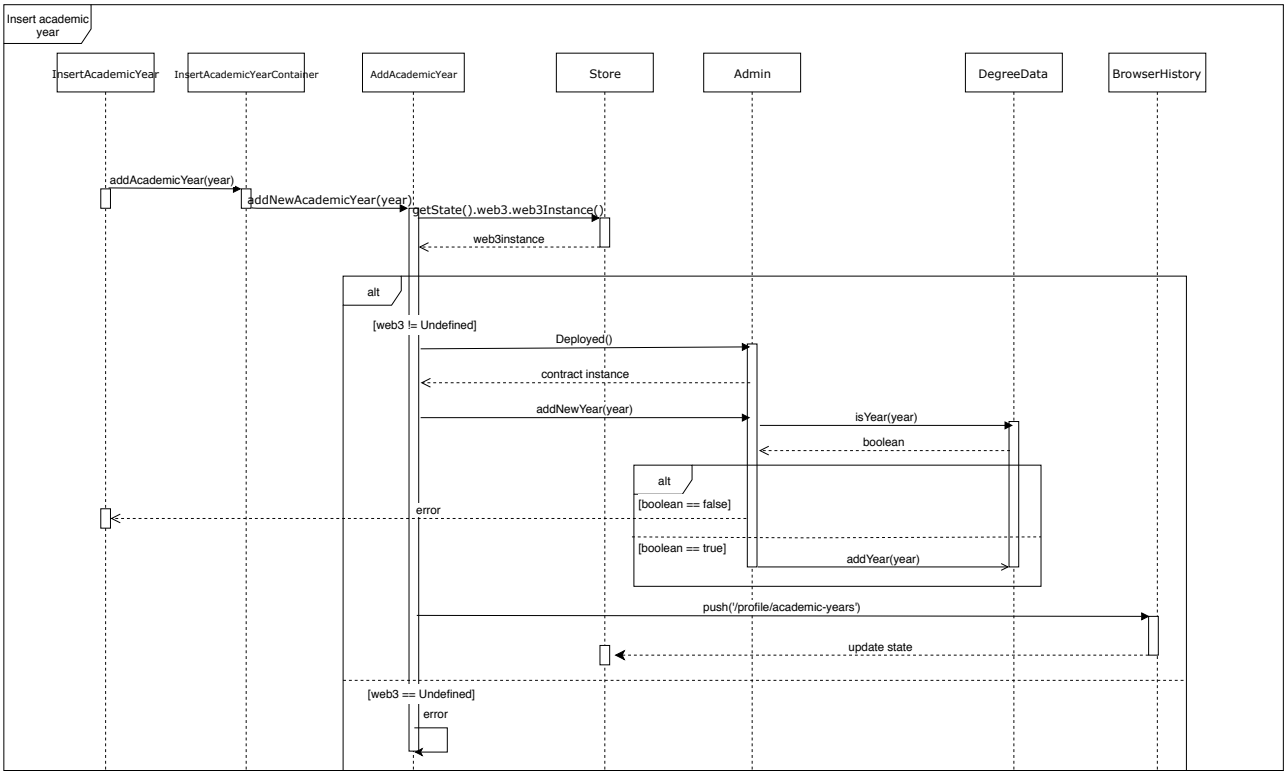

Figura 10: Diagramma di sequenza del processo di inserimento di un anno accademico

6 Requisiti soddisfatti

6.1 Tabella del soddisfacimento dei requisiti

ID requisito	Soddisfacimento nell'architettura	Soddisfacimento nel codice
R0F1	SODDISFATTO	SODDISFATTO
R0F2	SODDISFATTO	SODDISFATTO
R0F3	SODDISFATTO	SODDISFATTO
R0F4	SODDISFATTO	SODDISFATTO
R0F5	SODDISFATTO	SODDISFATTO
R0F6	SODDISFATTO	SODDISFATTO
R2F7	SODDISFATTO	NON SODDISFATTO
R2F8	SODDISFATTO	NON SODDISFATTO
R2F9	SODDISFATTO	NON SODDISFATTO
R2F10	SODDISFATTO	NON SODDISFATTO
R2F11	SODDISFATTO	NON SODDISFATTO
R2F13	SODDISFATTO	NON SODDISFATTO
R2F14	SODDISFATTO	NON SODDISFATTO
R0F15	SODDISFATTO	SODDISFATTO
R0F16	SODDISFATTO	SODDISFATTO
R0F17	SODDISFATTO	SODDISFATTO
R0F18	SODDISFATTO	SODDISFATTO
R0F19	SODDISFATTO	NON SODDISFATTO
R2F20	SODDISFATTO	NON SODDISFATTO
R2F21	SODDISFATTO	NON SODDISFATTO
R2F22	SODDISFATTO	NON SODDISFATTO
R0F23	SODDISFATTO	SODDISFATTO
R0F24	SODDISFATTO	SODDISFATTO
R2F25	SODDISFATTO	NON SODDISFATTO
R0F26	SODDISFATTO	SODDISFATTO

Tabella 2: Requisiti soddisfatti

6.2 Grafici sui requisiti soddisfatti

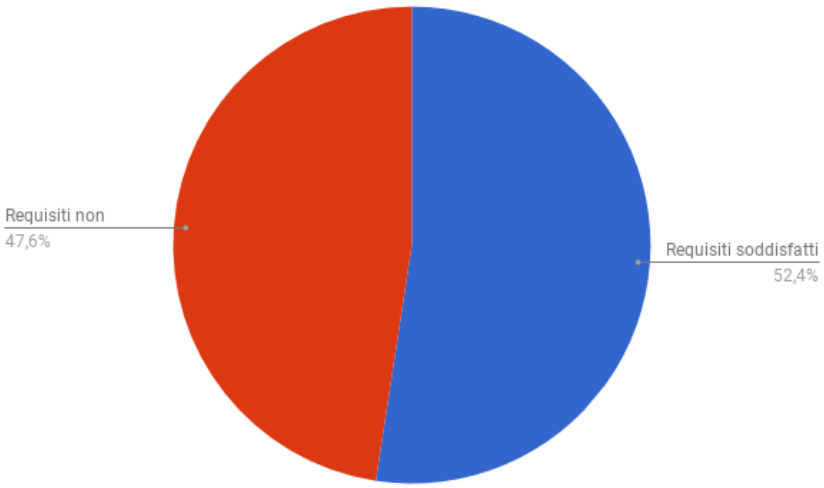

Figura 11: Requisiti soddisfatti

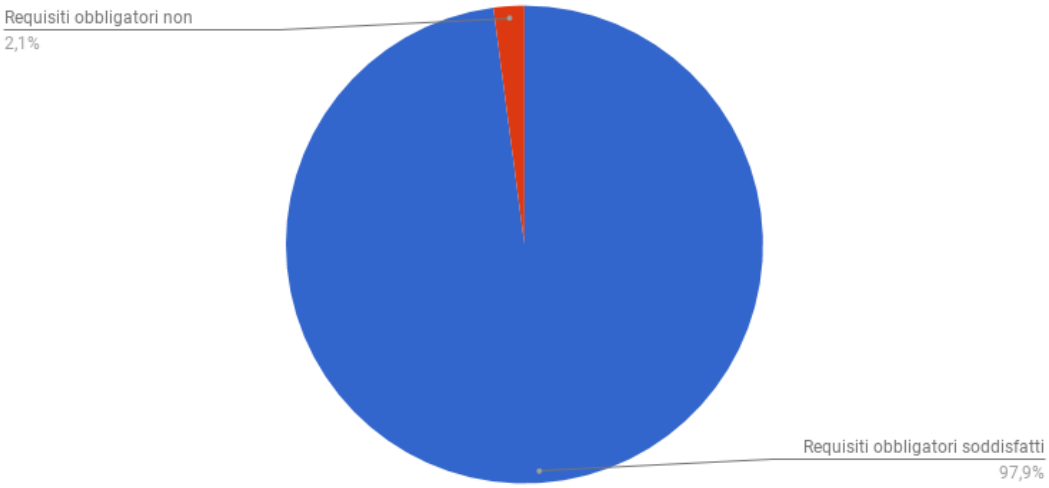

Figura 12: Requisiti obbligatori soddisfatti

7 Casi d'uso soddisfatti

7.1 Tabella del soddisfacimento dei casi d'uso

ID caso d'uso	Soddisfacimento nell'architettura	Soddisfacimento nel codice
UC1	SODDISFATTO	NON SODDISFATTO
UC2	SODDISFATTO	NON SODDISFATTO
UC3	SODDISFATTO	NON SODDISFATTO
UC4	SODDISFATTO	NON SODDISFATTO
UC6	SODDISFATTO	SODDISFATTO
UC7	SODDISFATTO	SODDISFATTO
UC8	SODDISFATTO	SODDISFATTO
UC9	SODDISFATTO	SODDISFATTO
UC10	SODDISFATTO	SODDISFATTO
UC11	SODDISFATTO	NON SODDISFATTO
UC12	SODDISFATTO	NON SODDISFATTO
UC13	SODDISFATTO	NON SODDISFATTO
UC14	SODDISFATTO	NON SODDISFATTO
UC15	SODDISFATTO	NON SODDISFATTO
UC16	SODDISFATTO	NON SODDISFATTO
UC17	SODDISFATTO	NON SODDISFATTO
UC18	SODDISFATTO	NON SODDISFATTO
UC19	SODDISFATTO	NON SODDISFATTO
UC20	SODDISFATTO	NON SODDISFATTO
UC21	SODDISFATTO	NON SODDISFATTO
UC22	SODDISFATTO	NON SODDISFATTO
UC23	SODDISFATTO	SODDISFATTO
UC24	SODDISFATTO	NON SODDISFATTO
UC25	SODDISFATTO	NON SODDISFATTO
UC26	SODDISFATTO	NON SODDISFATTO
UC27	SODDISFATTO	SODDISFATTO
UC28	SODDISFATTO	SODDISFATTO
UC29	SODDISFATTO	NON SODDISFATTO
UC30	SODDISFATTO	SODDISFATTO
UC31	SODDISFATTO	SODDISFATTO
UC32	SODDISFATTO	SODDISFATTO
UC33	SODDISFATTO	SODDISFATTO

Tabella 3: Casi d'uso soddisfatti

7.2 Grafici sui casi d'uso soddisfatti

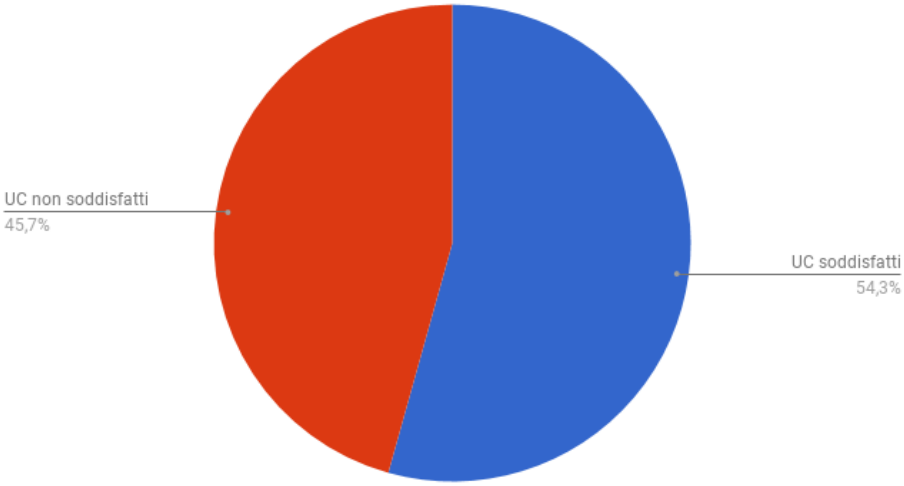

Figura 13: Casi d'uso soddisfatti