Allegato tecnico

Sons Of SWE - Progetto Marvin

sonsofswe.swe@gmail.com

Informazioni sul documento

Versione	3.0.0
Redazione	Dalla Riva Giovanni Thiella Eleonora Cavallin Giovanni Panozzo Stefano Caldart Federico
Verifica	Lorenzo Menegon
Approvazione	Favero Andrea
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.

Registro delle modifiche

Versione	Data	Descrizione	Autore	Ruolo
1.0.0	2018-06-03	Approvazione	Andrea Favero	*Responsabile*
0.1.0	2018-06-01	Effettuata verifica	Lorenzo Menegon	*Verificatore*
0.0.7	2018-05-27	Stesura della sezione *Requisiti soddisfatti*	Giovanni Dalla Riva	*Progettista*
0.0.6	2018-05-26	Stesura della sezione riguardante *Diagrammi di sequenza* in *Architettura del prodotto*	Federico Caldart	*Progettista*
0.0.5	2018-05-23	Stesura della sezione riguardante *Model* in *Architettura del prodotto*	Stefano Panozzo	*Progettista*
0.0.4	2018-05-20	Stesura della sezione riguardante *View* e *ViewModel* in *Architettura del prodotto*	Giovanni Cavallin	*Progettista*
0.0.3	2018-05-19	Stesura della sezione *Confronto con il PoC*	Eleonora Thiella	*Progettista*
0.0.2	2018-05-19	Stesura della sezione *Requisiti di sistema* e *Installazione ed esecuzione*	Giovanni Dalla Riva	*Progettista*
0.0.1	2018-05-18	Creato lo scheletro del documento e stesura della sezione *Introduzione*	Giovanni Dalla Riva	*Progettista*

Indice

Elenco delle figure

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**.

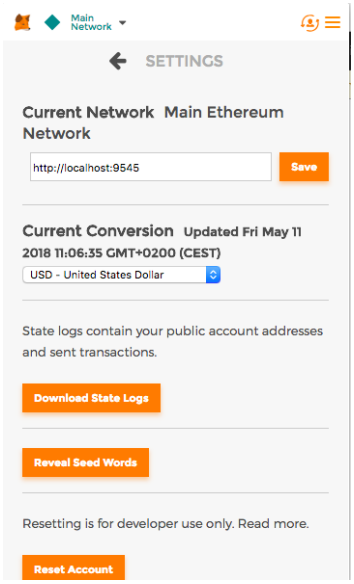

Figura 1: Setta l'RPC inserendo **http://localhost:9545**

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.

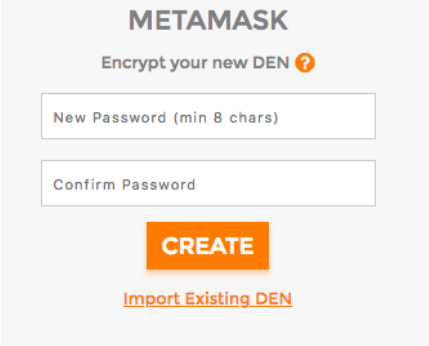

Figura 2: Clicca su **Import Existing DEN**

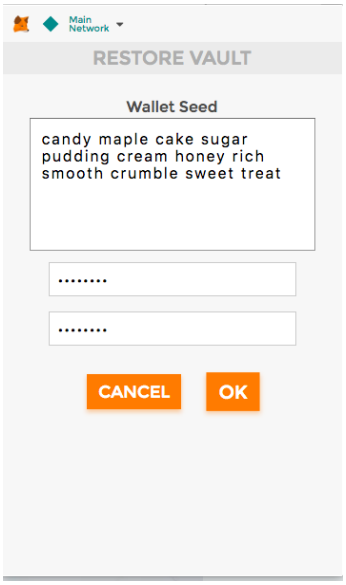

Figura 3: Inserisci la seed phrase e la password che vuoi usare

4 Confronto con il 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 2: 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 INSERIRE LA CARTELLA.

INSERIRE DIAGRAMMA DEI PACKAGE IN GENERALE

Figura 4: Diagramma generale dei package

5.1 Model

5.1.1 Design patterns

Per quanto riguarda la macrosezione relativa alla componente Model, si sono utilizzati i seguenti design pattern: INSERIRE I PATTERN

- : ;
- : .
- : .

5.1.2 Diagramma delle classi

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

INSERIRE DIAGRAMMA CON VISTA SU MODEL

Figura 5: Diagramma delle classi del componente Model

5.2 View

5.2.1 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.

5.2.2 Diagramma delle classi

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

INSERIRE DIAGRAMMA CON VISTA SU VIEW

Figura 6: Diagramma delle classi del componente View

5.3 ViewModel

5.3.1 Design patterns

Per la sezione riguardante il ViewModel, si è ricorso all'utilizzo dei seguenti design pattern:

- **Observer**: descrizione per quanto riguarda il comportamento di trigger-client che opera Redux sullo stato di React;
- **Façade**: per generare un'interfaccia astratta dalle implementazioni delle classi che poi si vanno ad utilizzare;
- **Adapter**: per interagire col model si usa un adaptator offerto dal framework truffle che si occupa di fare-un-sacco-di-cose-belle-da-chiedere-a-Stefano;
- **Command**: come redux ordina a react di re-renderizzare le pagine al cambiamento dello state dovuto ad un dispatch di un'azione.

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.

INSERIRE DIAGRAMMA CON VISTA SU VIEWMODEL

Figura 7: Diagramma delle classi del componente ViewModel

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 INSERIRE LA CARTELLA;

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. INSERIRE DIAGRAMMA DI LOGIN

Figura 8: Diagramma di sequenza del processo di login

5.4.2 Inserimento di un utente

Il seguente diagramma di sequenza rappresenta l'azione di inserimento di un utente nel sistema. INSERIRE DIAGRAMMA DI INSERIMENTO UTENTE

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

5.4.3 Inserimento di un anno accademico

Il diagramma di sequenza riportato qui di seguito raffigura il processo di inserimento di un nuovo anno accademico nel sistema. INSERIRE DIAGRAMMA DI INSERIMENTO ANNO ACCADEMICO

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

5.4.4 Visualizzazione di un anno accademico

Il seguente diagramma di sequenza rappresenta l'azione di visualizzazione di un anno accademico. INSERIRE DIAGRAMMA DI VISUALIZZAZIONE ANNO ACCADEMICO

Figura 11: Diagramma di sequenza del processo di visualizzazione 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 3: Requisiti soddisfatti

6.2 Grafici sui requisiti soddisfatti

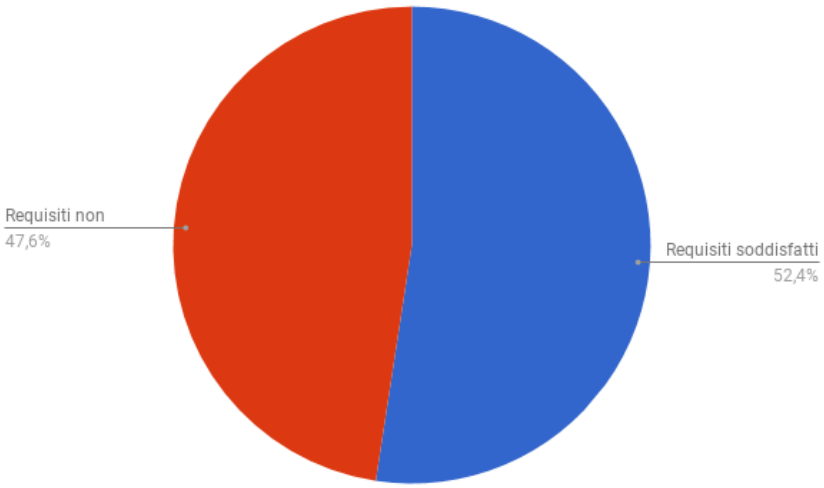

Figura 12: Requisiti soddisfatti

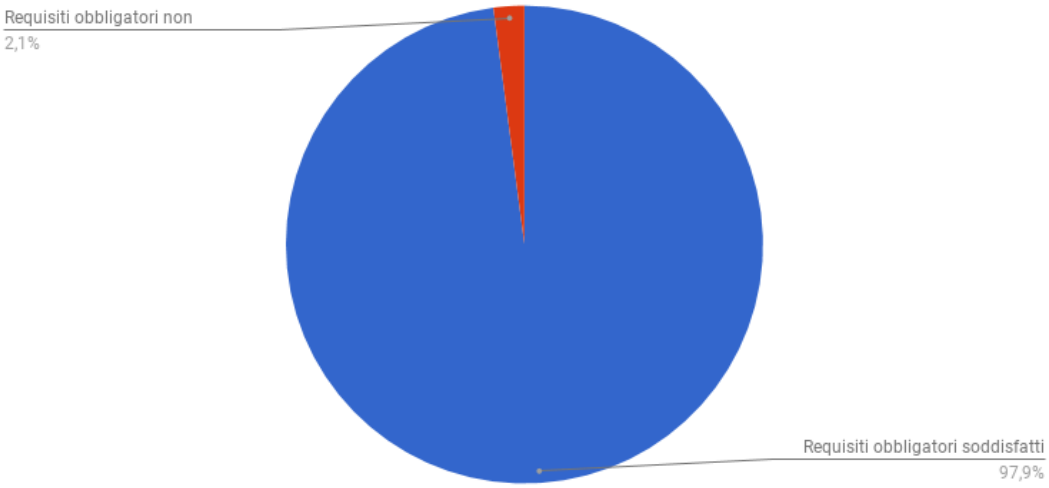

Figura 13: Requisiti obbligatori soddisfatti