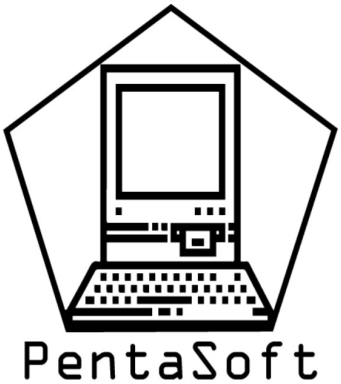

Specifica tecnica
Progetto Trustify

pentasoftswe@gmail.com

Informazioni sul documento

Responsabile	Luca Marcato
Redattori	Nicola Lazzarin
	Marco Brugin
	Luca Marcato
Verificatori	Marco Brugin
	Nicola Lazzarin
Uso	Esterno
Destinatari	Prof. Tullio Vardanega
	Prof. Riccardo Cardin
Versione	*v1.0.0*

Sommario

Il presente documento riporta tutte le scelte architetturali fatte dal gruppo *PentaSoft* durante lo sviluppo del prodotto *Trustify*. Per la descrizione del prodotto, vengono utilizzati i diagrammi delle classi.

Registro delle Modifiche

Versione	Data	Autore	Verificatore	Descrizione
1.0.0	2023/05/11	Luca Marcato (Responsabile)	Nicola Lazzarin	Approvazione per il rilascio
0.0.5	2023/05/9	Nicola Lazzarin (Progettista)	Marco Brugin	Stesura § requisiti soddisfatti
0.0.4	2023/05/9	Marco Brugin (Progettista)	Nicola Lazzarin	Stesura § componente di API-REST
0.0.3	2023/05/5	Luca Marcato (Progettista)	Marco Brugin	Stesura § componente di frontend
0.0.2	2023/05/3	Nicola Lazzarin (Progettista)	Marco Brugin	Stesura § componente di backend
0.0.1	2023/03/11	Nicola Lazzarin (Progettista)	Marco Brugin	Creazione struttura documento e stesura Introduzione

Indice

Elenco delle tabelle

Elenco delle figure

1 Introduzione

1.1 Scopo del documento

La Specifica tecnica ha lo scopo di fornire una descrizione completa dell'architettura del prodotto software, delle tecnologie utilizzate e delle scelte architetturali adottate dal team di sviluppo del gruppo PentaSoft durante la fase di progettazione e codifica del prodotto. In particolare, la Specifica tecnica include diagrammi delle classi per descrivere l'architettura e le funzionalità principali del prodotto, al fine di fornire una panoramica completa del sistema e delle sue interazioni. Inoltre, la Specifica tecnica include una sezione dedicata ai requisiti soddisfatti dal prodotto, per consentire al team di valutare lo stato di avanzamento del lavoro e garantire il rispetto degli obiettivi prefissati.

1.2 Scopo del prodotto

Scopo del progetto è la realizzazione di una *webapp*$_{\mathbf{G}}$ che permetta di rilasciare e visualizzare recensioni certificate tramite uno *smart contract*$_{\mathbf{G}}$ risiedente in una *blockchain*$_{\mathbf{G}}$ *Ethereum*$_{\mathbf{G}}$ compatibile, al fine di minimizzare la compravendita di recensioni e il *review bombing*$_{\mathbf{G}}$.

1.3 Glossario

Alcuni dei termini utilizzati in questo documento potrebbero generare dei dubbi riguardo al loro significato, al fine di evitare tali ambiguità è necessario dar loro una definizione. Tali termini vengono contassegnati da una G maiuscola finale,se questa non compare in un titolo di sezione, a pedice della parola ed essa non verrà ripetuta più di una volta per paragrafo/sottosezione/sezione onde evitare fastidiose ripetizioni. La loro spiegazione è riportata nel *Glossario v1.0.0*

1.4 Riferimenti

1.4.1 Normativi

- ***Norme di progetto v1.0.0***
- **Regolamento del progetto didattico**:

  https://www.math.unipd.it/ tullio/IS-1/2022/Dispense/PD02.pdf

- **Presentazione Capitolato C7 - Trustify**:

  https://www.math.unipd.it/ tullio/IS-1/2022/Progetto/C7.pdf

1.4.2 Informativi

- ***Analisi dei requisiti v3.0.0***
- **Qualità di prodotto - slide T12 del corso di Ingegneria del Software**:

  https://www.math.unipd.it/ tullio/IS-1/2022/Dispense/T12.pdf

- **Qualità di processo - slide T13 del corso di Ingegneria del Software**:

  https://www.math.unipd.it/ tullio/IS-1/2022/Dispense/T13.pdf

- **Verifica e validazione: introduzione - slide T14 del corso Ingegneria del Software**:

  https://www.math.unipd.it/ tullio/IS-1/2022/Dispense/T14.pdf

- **Verifica e validazione: introduzione - slide T15 del corso Ingegneria del Software**:

  https://www.math.unipd.it/ tullio/IS-1/2022/Dispense/T15.pdf

- **Verifica e validazione: introduzione - slide T16 del corso Ingegneria del Software**:

  https://www.math.unipd.it/ tullio/IS-1/2022/Dispense/T16.pdf

2 Architettura

Le scelte architetturali fatte dal gruppo di progetto durante la progettazione e sviluppo del prodotto verranno illustrate ed analizzate per componenete singolo in modo da rendere più chiara la descrizione.

2.1 Componente di back-end

I *contratti intelligenti*$_{\mathbf{G}}$ *(smart contract)*$_{\mathbf{G}}$ che costituiscono il sistema *Trustify* sono stati progettati per essere eseguiti sulla rete *Ethereum* o su qualsiasi altra rete compatibile con *Ethereum*. I due contratti che compongono il sistema sono:

- **TCoin**: rappresenta un token usato per il trasferimento di una valuta fra utenti;
- **Trustify**: rappresenta il contratto principale che gestisce e mantiene le recensioni e i pagamenti fra vari utenti.

2.1.1 TCoin

Il contratto *TCoin* è stato scritto nel linguaggio di programmazione *Solidity*$_{\mathbf{G}}$ ed estende il contratto *ERC20*$_{\mathbf{G}}$ della libreria *OpenZeppelin*, che implementa uno standard per i *token*$_{\mathbf{G}}$ Ethereum.

Il contratto oltre ad avere un costruttore che imposta il nome del token e il simbolo con la stringa *TCoin* possiede un'unica funzione pubblica chiamata *drip()* che emette 100.000 unità del token *TCoin* all'indirizzo chiamante. La funzione *drip()* fa ciò utilizzando la funzione *mint()* ereditata dal contratto *ERC20*, che crea nuove unità del token e le assegna all'indirizzo specificato come primo parametro della funzione *mint()*. Questo contratto *ERC20* viene usato come valuta per il pagamento delle recensioni, in quanto è possibile trasferire i token da un indirizzo ad un altro utilizzando funzioni apposite di trasferimento token *ERC20*.

2.1.2 *Trustify*

Il contratto *Trustify* è stato scritto nel linguaggio di programmazione *Solidity*$_{\mathbf{G}}$ e prevede un sistema di recensioni che consente agli utenti di scrivere e leggere recensioni su aziende e/o negozi identificati da un indirizzo *wallet*$_{\mathbf{G}}$ univoco. Il contratto è progettato per utilizzare delle mappe per tenere traccia delle recensioni degli utenti e delle aziende/negozi che hanno ricevuto recensioni. Inoltre, il contratto richiede agli utenti, come pagamento, il deposito di un *token* (nel nostro caso il token *TCoin*) per poter poi rilasciare una recensione. Oltre alla funzione di scrittura delle recensioni il contratto include funzioni di lettura di un'azienda/negozio specifico utilizzando range che vanno dalla singola recensione fino a massimo 25 per volta.

2.1.2.1 Enumerazione Il contratto *Trustify* definisce un'enumerazione che rappresenta lo stato di una recensione.

- **ACTIVE**: indica che la recensione non è mai stata modificata o eliminata;
- **MODIFIED**: indica che la recensione è stata modificata almeno una volta;
- **DELETED**: indica che la recensione è stata cancellata.

Questi stati servono per far capire se le recensioni sono state modificate, eliminate oppure se sono state lasciate inalterate dall'utente che le ha scritte.
Quando un utente cancella una recensione essa non verrà cancellata realmente nel contratto ma gli verrà applicato uno stato. Colui che richiederà le recensioni riceverà tutte le recensioni con i realivi stati e starà a lui decidere come filtrarle.
Se una recensione è stata erroneamente cancellata e l'utente vuole rimetterla visibile basterà modificarla e lo stato cambiarà da **DELETED** a **MODIFIED**.

2.1.2.2 Strutture dati Il contratto usa tre strutture dati: **Review**, **Company** e **Customer**. La struttura **Review** rappresenta una singola recensione ed è composta da quattro campi:

- **review**: una stringa che contiene il testo della recensione;
- **stars**: un valore intero che rappresenta il numero di stelle assegnate alla recensione;
- **havePayed**: un booleano che indica se l'utente ha eseguito una translazione.
- **state**: rappresenta lo stato della recensione, può essere ACTIVE, MODIFIED o DELETED.

La struttura **Company** rappresenta una singola azienda o esercizio commerciale ed è composta da due campi:

- **allCustomerAddress**: un array di indirizzi Ethereum che contiene gli indirizzi degli utenti che hanno scritto una recensione per questa azienda;
- **reviewMap**: una mappa che associa ad ogni indirizzo di utente la sua recensione;

La struttura **Customer** rappresenta un singolo utente ed è composta da due campi:

- **allCompanyAddress**: un array di indirizzi Ethereum che contiene gli indirizzi delle aziende o esercizi commerciali che hanno ricevuto una recensione da questo utente;
- **reviewMap**: una mappa che associa ad ogni indirizzo di azienda o esercizio commerciale la recensione dell'utente.

Queste strutture vengono usate per imagazzinare dati nel contratto e per poterle poi richiamare in seguito.

2.1.2.3 Mappe Il contratto definisce due mappe private che permettono di associare ad ogni indirizzo di azienda o esercizio commerciale la sua struttura dati **Company** e ad ogni indirizzo di utente la sua struttura dati **Customer**.

- **companyMap**: una mappa che associa ad ogni indirizzo di azienda o esercizio commerciale la sua struttura dati **Company**;
- **customerMap**: una mappa che associa ad ogni indirizzo di utente la sua struttura dati **Customer**.

2.1.2.4 Interfacce e librerie Il contratto implementa la seguente interfaccia e libreria:

- **IERC20.sol**: l'interfaccia standard per i token *ERC20*;
- **SafeERC20.sol**: una libreria che definisce metodi sicuri per il trasferimento dei token *ERC20*;

2.1.2.5 Metodi Il contratto definisce i seguenti metodi:

- **function havePayed(address myAddress, address companyAddress) public view returns (bool)** :
  La funzione *havePayed* è una funzione di sola lettura che controlla se un utente specifico ha pagato per lasciare una recensione per una determinata azienda. Prende in input l'indirizzo dell'utente (myAddress) e l'indirizzo dell'azienda (companyAddress) e restituisce un valore booleano true se l'utente ha pagato per la recensione, false altrimenti. La funzione legge lo stato della recensione dell'utente all'interno della mappa delle recensioni dell'azienda e controlla il valore booleano havePayed. Se il valore è true, l'utente ha pagato per la recensione, altrimenti non ha pagato. La funzione non modifica lo stato della mappa delle recensioni dell'azienda o della mappa delle recensioni dell'utente quindi può essere dichiarata come view.

- **modifier checkTransaction(address companyWalletAddress)**:
  Il modificatore *CheckTransaction* viene usato all'interno delle funzioni *WriteAReview* e *DeleteReview* per verificare che l'utente abbia effettuato una transazione verso l'indirizzo dell'azienda a cui sta scrivendo o cancellando una recensione. In particolare, *CheckTransaction* controlla che l'utente abbia già effettuato un pagamento (trasferimento di token) all'azienda corrispondente, salvato nel mapping *reviewMap* della *Company* relativa all'indirizzo dell'azienda e all'interno del mapping *customerMap* della *Customer* relativa all'indirizzo dell'utente che ha effettuato la transazione. Se l'utente non ha ancora effettuato una transazione, il modificatore emette un messaggio di errore. Questo modificatore viene usato esclusivamente all'interno del contratto e non può essere chiamata da un utente esterno.

- **modifier checkActionToYourself(address yourAddress)**:
  Il modificatore *CheckActionToYourself* impone che l'azione richiesta non possa essere eseguita dall'indirizzo chiamante stesso. In particolare, viene effettuato un controllo sulla condizione che l'indirizzo passato come parametro *yourAddress* non coincida con l'indirizzo chiamante *msg.sender*. Nel caso dovesse coincidere restituisce un errore.

  Il modificatore viene utilizzato in due funzioni del contratto *Trustify*:

  - **DepositTokens**: permette di depositare token in un determinato indirizzo passato come parametro. L'indirizzo non può coincidere con l'indirizzo chiamante della funzione.
  - **WriteAReview**: permette di scrivere una recensione su un indirizzo aziendale passato come parametro. L'indirizzo non può coincidere con l'indirizzo chiamante della funzione.

- **function depositTokens(address addressToDeposit, uint amount) public checkActionToYourself (addressToDeposit)**:
  Questa funzione consente agli utenti di depositare token *ERC20* dall'indirizzo del chiamante ad un altro passato come parametro. L'indirizzo del destinatario del deposito viene specificato come parametro *addressToDeposit*, mentre il numero di token da depositare viene specificato come parametro *amount*. Prima di effettuare il deposito, viene verificato che l'utente abbia inserito un indirizzo diverso dal proprio chiamando il modificatore *checkActionToYourself*. Dopo di che i token vengono trasferiti dal portafoglio dell'utente al portafoglio dell'azienda specificata tramite l'indirizzo *addressToDeposit*.

- **function deleteReview(address reviewAddress) public checkActionToYourself(reviewAddress)**:
  La funzione *DeleteReview* viene utilizzata per eliminare una recensione. Questo viene fatto impostando lo stato della recensione come *ReviewState.DELETED*. In particolare, la funzione aggiorna lo stato della recensione dell'utente chiamante e lo stato della recensione dell'azienda a cui è stata lasciata la recensione. La funzione non elimina effettivamente la recensione dal contratto, ma imposta solo il suo stato come *"eliminato"*.

- **function stateToString(ReviewState state) private pure returns (string memory)** :
  La funzione *StateToString* è una funzione privata (che può essere chiamata solo all'interno del contratto) che prende in input un valore dell'enum *ReviewState* e restituisce una stringa che rappresenta lo stato corrispondente. Se il valore dell'enum è:

  - ACTIVE, restituisce la stringa "ACTIVE".
  - MODIFIED, restituisce la stringa "MODIFIED".
  - DELETED, restituisce la stringa "DELETED".

  Questa funzione viene utilizzata all'interno del contratto per convertire i valori dell'enum ReviewState in stringhe interpretabili da vari linguaggi di programmazione.

- **function writeReview(address companyWalletAddress, string memory review, uint stars, uint amount) public**
  checkActionToYourself(addressToReview),
  checkTransaction(addressToReview):
  Questa funzione consente agli utenti di scrivere una recensione per un'azienda specificata tramite l'indirizzo *addressToReview*. La recensione e il numero di stelle sono specificati come parametri review e stars. Prima di scrivere la recensione, viene verificato che l'utente abbia effettuato una transazione con l'azienda tramite la funzione *CheckTransaction* e viene verificato che non stia facendo un'azione su se stesso chiamando la funzione *checkActionToYourself*. Se l'utente ha effettuato una transazione, la recensione viene scritta. Se invece non ha eseguito una transazione il contratto restituirà un errore. Nel caso in cui l'utente abbia già scritto una recensione per l'azienda, la vecchia recensione viene sovrascritta con la nuova. Viene anche aggiornato il mapping della mappa *customerMap* con l'indirizzo dell'azienda e la nuova recensione.

- **function getCompanyReview(uint start, uint end, address companyAddress) public view returns (string[] memory, uint8[] memory)**:
  La funzione *GetCompanyReview* restituisce un array di stringhe contenente le review scritte da un numero specificato di clienti per una determinata azienda, insieme ad un array di numeri interi che rappresentano il numero di stelle assegnate ad ogni recensione e allo stato delle recensioni. La funzione prende tre parametri in input:

  - **start** è l'indice del primo elemento dell'array di recensioni da restituire.
  - **end** è l'indice dell'ultimo elemento dell'array di recensioni da restituire.
  - **companyAddress** è l'indirizzo dell'azienda di cui si vogliono visualizzare le recensioni.

  La funzione effettua alcuni controlli sulla validità degli indici start e end rispetto alla lunghezza dell'array delle recensioni dell'azienda specificata, e in caso di superamento delle 25 recensioni per chiamata restituisce un errore. In caso contrario, la funzione restituisce le recensioni per quel range di indici.

- **function getSpecificReview(address addressReviewed) public view returns (string memory, uint8)**:
  Questa funzione, chiamata *getSpecificReview*, è una funzione di tipo *view* che restituisce una tupla contenente una stringa e un numero intero. Questi valori rappresentano rispettivamente una recensione specifica e il numero di stelle assegnato a tale recensione. La funzione accetta un parametro *addressReviewed* che indica l'indirizzo dell'azienda per la quale si desidera ottenere una recensione specifica. La funzione utilizza poi l'indirizzo del chiamante della funzione (msg.sender) per accedere alla recensione specifica dell'azienda. La recensione e il numero di stelle assegnate sono quindi restituiti nella tupla di output. Inoltre, la funzione utilizza anche un'istruzione require per verificare che il chiamante abbia effettivamente rilasciato una recensione per l'azienda specificata. Se il numero di stelle assegnate è zero, viene restituito un messaggio di errore che indica che il chiamante non ha rilasciato alcuna recensione per l'azienda specificata.

- **function getMyReview(uint start,uint end) public view returns (string[] memory, uint8[] memory, address[] memory)**:
  La funzione *getMyReview* restituisce un insieme di recensioni fatte da un utente specifico. Prende in input due parametri start e end che rappresentano rispettivamente l'indice iniziale e finale delle recensioni che si vogliono ottenere. La funzione verifica che l'utente abbia effettuato almeno una recensione, che l'indice di partenza sia minore o uguale alla lunghezza totale delle recensioni e che la differenza tra l'indice finale e quello di partenza non sia superiore a 25. In caso contrario, viene sollevata una eccezione. La funzione restituisce quindi quattro array contenenti le stringhe

delle recensioni, le stelle assegnate, lo stato delle recensioni e gli indirizzi delle aziende recensite nell'intervallo specificato.

- **function getAverageStars(address addressReviewed) public view returns (uint[] memory)**:
  Questa funzione restituisce un array di valori interi contenente tutte le stelle assegnate alle recensioni ricevute da una specifica azienda, identificata dall'indirizzo *addressReviewed*. La funzione inizia controllando la lunghezza dell'array di indirizzi degli utenti che hanno lasciato recensioni per l'azienda. Se la lunghezza è zero, significa che l'azienda non ha ricevuto alcuna recensione e la funzione genera un errore. Altrimenti, la funzione inizializza un array di interi con la stessa lunghezza dell'array di indirizzi degli utenti, quindi, mediante un ciclo for, riempie l'array di stelle assegnate a ogni recensione. Infine, restituisce l'array di stelle.

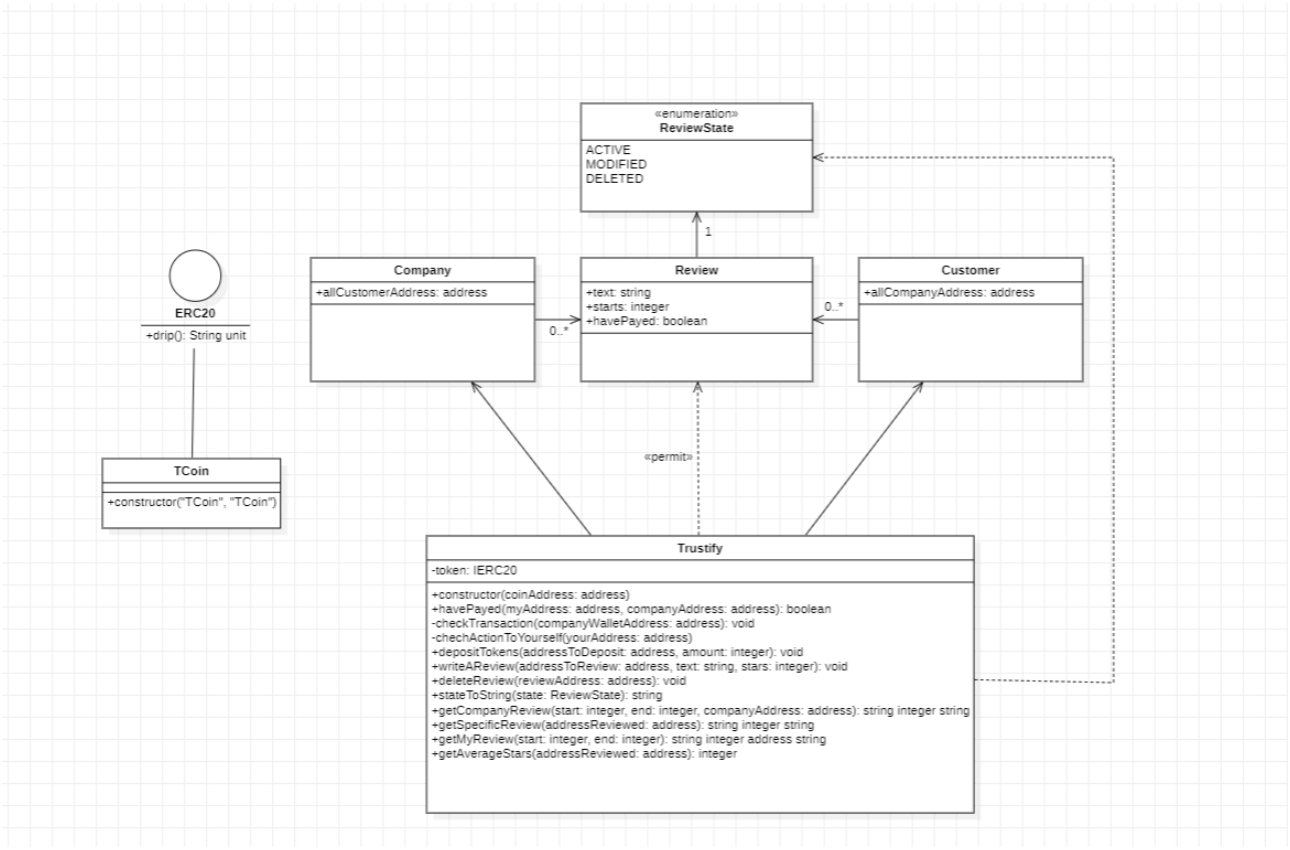

Figura 1: Diagramma delle classi della componente di back-end

2.2 Componente di front-end

L'obiettivo principale della *webapp* è quello di fornire agli utenti un'esperienza d' interazione intuitiva e facile con il sistema decentralizzato *Trustify*. In particolare, la piattaforma consente di visualizzare le recensioni di un'azienda, fornendo un quadro completo e trasparente delle esperienze degli utenti con quella specifica attività commerciale. Gli utenti possono anche scrivere e pubblicare recensioni per un'azienda, condividendo le loro opinioni e raccomandazioni.

2.2.1 *Webapp*

La *webapp*$_{\mathbf{G}}$ per il sistema *Trustify* è stata creata utilizzando *Angular*$_{\mathbf{G}}$, uno dei framework open source più utilizzati per lo sviluppo di applicazioni web moderne. Inoltre, il team di sviluppo ha scelto di utilizzare *TypeScript* come linguaggio principale per la scrittura del codice, poiché questo linguaggio fornisce una tipizzazione statica e altre caratteristiche avanzate che aiutano a rendere il codice più facile da mantenere e da scalare.

2.2.1.1 *Angular Material* Angular Material è un'implementazione di Google Material Design in Angular. Si tratta quindi di una libreria di componenti dell'interfaccia utente (UI) per gli sviluppatori che usano Angular. Nel nostro progetto abbiamo fatto ampio uso di questa libreria per la creazione di componenti grafici quali bottoni, form, stelle per le recensioni, ecc.

2.3 Componente di di interazione tra front-end e back-end

L'obiettivo di questa componente è quello di fornire un'interfaccia per la comunicazione tra il front-end e il back-end. In particolare, il front-end comunica con il back-end attraverso delle chiamate *REST* che vengono gestite da questa componente.

2.3.1 *Angular* service

In Angular, un service è semplicemente qualsiasi set di funzionalità che vogliamo rendere disponibili a componenti multipli. Nel nostro progetto abbiamo creato diversi service per la gestione della webapp ma i 2 principali sono:

- **WalletService** La classe *WalletService* definisce un servizio per la gestione della connessione con il wallet *MetaMask*$_{\mathbf{G}}$ e la comunicazione con la *blockchain*. Il servizio espone metodi per verificare se *MetaMask* è installato e se l'utente è connesso ad esso. Inoltre, il servizio permette di recuperare l'indirizzo dell'account corrente e l'URL del provider HTTP di *Infura*$_{\mathbf{G}}$ per la rete di test *Sepolia*.

  Metodi principali:

  - **getAccount()** è un metodo asincrono che restituisce l'indirizzo dell'account corrente. L'indirizzo dell'account viene utilizzato per inviare e ricevere transazioni sulla blockchain.
  - **isInstalled()** è un metodo asincrono che controlla se il plugin *MetaMask* è installato nel browser dell'utente poiché l'interazione con *MetaMask* è necessaria per gestire la connessione con la blockchain.
  - **isWalletConnected()** è un metodo asincrono che restituisce true se l'utente è attualmente connesso a *MetaMask*.

- **connectToMetamask()** è un metodo asincrono che apre la finestra di dialogo di connessione di *MetaMask* e, se l'utente accetta la connessione, imposta l'indirizzo dell'account corrente e il provider $Web3_{G}$. Questo metodo viene utilizzato per connettersi a MetaMask e impostare le variabili di stato del servizio.
- **switchNetwork()** è un metodo asincrono che permette di cambiare rete in cui si trova il wallet *MetaMask*, in questo caso viene utilizzato per passare alla rete *Sepolia*. Il metodo controlla prima se la rete è già presente in *MetaMask*, altrimenti la aggiunge.
- **connect()** è un metodo asincrono che chiama il metodo *connectToMetamask()* e il metodo *switchNetwork()* per connettersi a *MetaMask* e passare alla rete *Sepolia*.

- **Web3Service** Questa classe, chiamata *Web3Service*, rappresenta un servizio *Angular* che fornisce una serie di funzionalità per interagire con la *blockchain Ethereum* attraverso la libreria *Web3.js*. In particolare, la classe fornisce metodi per effettuare operazioni con i contratti (*TCoin e Trustify*) presenti in *blockchain*.

  Metodi principali:

  - **pullTCoin** è un metodo asincrono che invia una transazione allo smart contract *TCoin* per richiedere l'invio di token.
  - **getTokenBalance** è un metodo asincrono che restituisce il saldo di token *TCoin* dell'account con cui il servizio comunica.
  - **approveTokens** è un metodo asincrono che richiede l'autorizzazione per la spesa di un determinato numero di token *TCoin* allo smart contract *Trustify*.
  - **depositTokens** è un metodo asincrono che invia una transazione alla smart contract *Trustify* per depositare un determinato numero di token *TCoin*.
  - **writeAReview** è un metodo asincrono che invia una transazione alla smart contract *Trustify* per scrivere una recensione per un'azienda.
  - **getCompanyReview** è un metodo asincrono che restituisce un array contenente le recensioni delle aziende tra un intervallo di indici.
  - **getSpecificReview** è un metodo asincrono che restituisce la recensione e il punteggio in stelle associati a un'azienda specifica.
  - **getMyReview** è un metodo asincrono che restituisce la recensione e il punteggio in stelle dell'utente corrente per una determinata azienda.
  - **deleteReview** è un metodo asincrono che invia una transazione all contratto *Trustify* per cancellare la recensione dell'utente corrente per una determinata azienda.

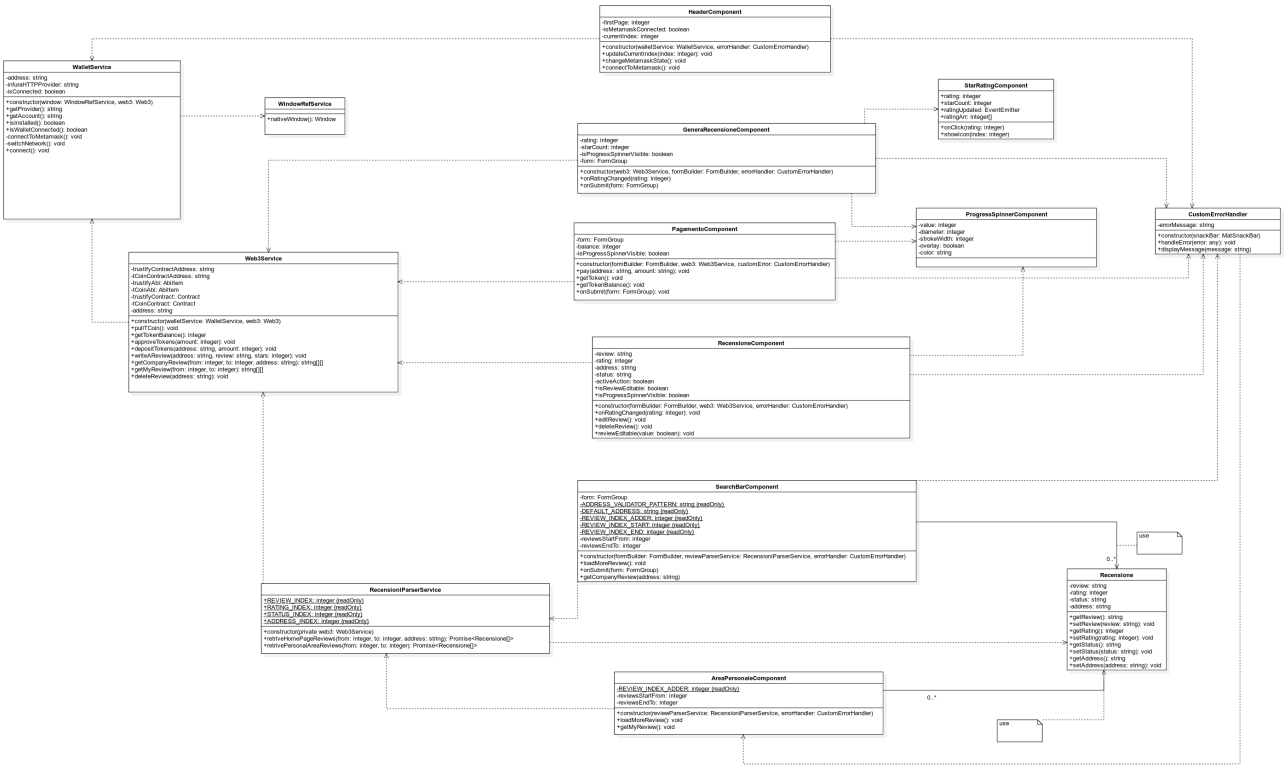

Figura 2: Diagramma delle classi delle componenti di front-end e di interazione tra front-end e back-end

2.4 Componente di *API*

L'obiettivo di questa componente è quella di fornire per gli *E-commerce*$_{\mathbf{G}}$ un'interfaccia per poter dialogare con la *blockchain Ethereum*$_{\mathbf{G}}$ e poter usufruire dei servizi offerti da *Trustify*$_{\mathbf{G}}$. Per far ciò è stata sviluppata una *API*$_{\mathbf{G}}$ conforme allo stile architetturale *REST-API*$_{\mathbf{G}}$. La componente di *API* è stata sviluppata utilizzando il framework *Spring Boot* e il linguaggio di programmazione *Java*.

2.4.1 Descrizione delle classi

- La classe *TrustifyContractReader* è una classe che permette di ottenere le recensioni di una determinata azienda attraverso l'uso di una smart contract Ethereum. In particolare, è possibile creare un'istanza della classe *TrustifyContractReader* passando come argomenti l'indirizzo dell'azienda, la posizione iniziale e finale dell'intervallo di recensioni da recuperare. La classe utilizza la libreria *Web3j* per interagire con la *blockchain Ethereum* e la libreria *Tuples*$_{\mathbf{G}}$ per gestire i risultati delle transazioni. Il metodo *getReviews()* è quello che effettivamente esegue la chiamata allo *Smart contract* e restituisce una lista di oggetti *Review*.

- La classe *Review* rappresenta una recensione con le seguenti proprietà: una stringa *text* che rappresenta il testo della recensione e un intero *stars* che rappresenta il numero di stelle attribuite alla recensione. Il costruttore della classe accetta due parametri, una stringa *text* e un intero *stars*, e inizializza le proprietà della classe con questi valori.

- La classe *ErrorMessage* rappresenta un messaggio di errore con la proprietà *message* che contiene il messaggio di errore. Viene usata per far sapere che errore è avvenuto durante l'esecuzione di una richiesta HTTP ritorando l'errore stesso al posto del JSON.

- La classe *AppExceptionHandler* è una classe di gestione delle eccezioni, estende *ResponseEntityExceptionHandler* e definisce un metodo che gestisce tutte le eccezioni lanciate dall'applicazione. In particolare, il metodo restituisce un oggetto *ErrorMessage* che contiene il messaggio di errore.
- La classe *SmartContractReaderController* è una classe di controller REST che definisce un'API per accedere alle recensioni tramite un'interfaccia HTTP. Il metodo *SmartContractReader* della classe si occupa di chiamare il metodo *getReviews()* della classe *TrustifyContractReader* per ottenere le recensioni.
- La classe *TrustifyRestApplication* è la classe di avvio dell'applicazione Spring Boot e contiene il metodo *main* che avvia l'applicazione.

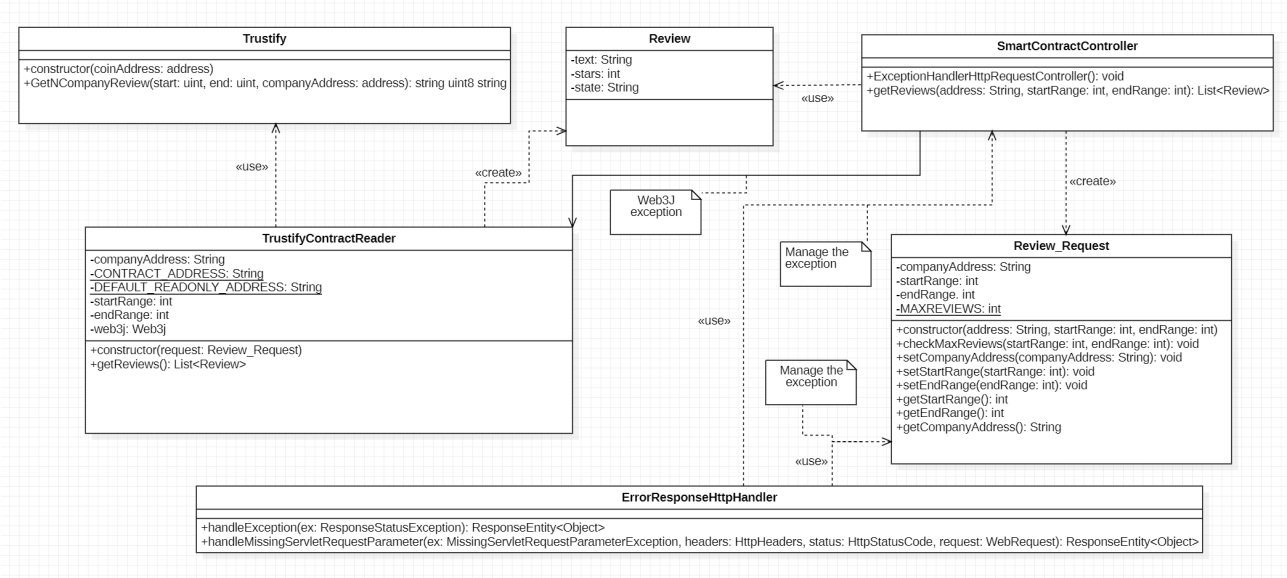

Figura 3: Diagramma delle classi della componente di API

2.5 Design pattern utilizzati

Per il progetto *Trustify* sono stati utilizzati i seguenti design pattern:

- **Dependency Injection**: è stato utilizzato nella webapp per iniettare i servizi tra i vari componenti dell'applicazione. Questo pattern è usato di default nelle applicazioni Angular.

3 Requisiti soddisfatti

Tabella 2: Requisiti Funzionali

Codice	Descrizione	Classificazione	Fonti
RFO1	L'utente deve poter collegare il proprio *wallet*$_{\mathbf{G}}$ alla *webapp*$_{\mathbf{G}}$ tramite *Metamask*$_{\mathbf{G}}$.	Obbligatorio	Soddisfatto
RFO1.1	L'utente deve poter visualizzare un messaggio di errore nel caso *Metamask* non sia installato.	Obbligatorio	Soddisfatto
RFO1.2	L'utente deve poter visualizzare un messaggio informativo quando il proprio *wallet* venga connesso correttamente.	Obbligatorio	Soddisfatto
RFO2	L'utente deve poter rilasciare una recensione relativa a un'attività per cui abbia precedentemente effettuato un pagamento.	Obbligatorio	Soddisfatto
RFO2.1	L'utente deve poter visualizzare un messaggio di errore connessione nel caso in cui per assenza di connessione non sia possibile effettuare una nuova recensione.	Obbligatorio	Soddisfatto
RFO2.2	L'utente deve poter inserire l'indirizzo dell'attività da recensire.	Obbligatorio	Soddisfatto
RFO2.2.1	L'utente deve poter visualizzare un messaggio di errore nel caso non sia stato inserito alcun indirizzo *wallet*.	Obbligatorio	Soddisfatto
RFO2.2.2	L'utente deve poter visualizzare un messaggio di errore nel caso l'indirizzo *wallet* dell'attività inserito non sia corretto.	Obbligatorio	Soddisfatto
RFO2.3	L'utente deve poter inserire la descrizione della recensione.	Obbligatorio	Soddisfatto
RFO2.4	L'utente deve poter inserire il parametro di valutazione della recensione.	Obbligatorio	Soddisfatto
RFO2.4.1	L'utente deve poter visualizzare un messaggio di errore nel caso il parametro di valutazione della recensione non sia stato inserito.	Obbligatorio	Soddisfatto
RFO2.4.2	L'utente deve poter visualizzare un messaggio di errore nel caso il valore del parametro di valutazione della recensione non sia valido.	Obbligatorio	Soddisfatto

Continua nella prossima pagina

Tabella 2 – *Continua dalla pagina precedente*

Codice	Descrizione	Classificazione	Fonti
RFO2.4.3	L'utente deve poter inserire il valore 1 per il parametro di valutazione.	Obbligatorio	Soddisfatto
RFO2.4.4	L'utente deve poter inserire il valore 2 per il parametro di valutazione.	Obbligatorio	Soddisfatto
RFO2.4.5	L'utente deve poter inserire il valore 3 per il parametro di valutazione.	Obbligatorio	Soddisfatto
RFO2.4.6	L'utente deve poter inserire il valore 4 per il parametro di valutazione.	Obbligatorio	Soddisfatto
RFO2.4.7	L'utente deve poter inserire il valore 5 per il parametro di valutazione.	Obbligatorio	Soddisfatto
RFO3	L'utente deve poter visualizzare una lista di recensioni.	Obbligatorio	Soddisfatto
RFO3.1	L'utente deve poter visualizzare un messaggio di errore connessione nel caso in cui per assenza di connessione non sia possibile visualizzare la lista di recensioni.	Obbligatorio	Soddisfatto
RFO3.2	L'utente deve poter visualizzare un messaggio informativo nel caso la lista di recensioni sia vuota.	Obbligatorio	Soddisfatto
RFO3.3	L'utente deve poter visualizzare una singola recensione presente nella lista.	Obbligatorio	Soddisfatto
RFO3.3.1	L'utente deve poter visualizzare il parametro di valutazione di una singola recensione presente nella lista.	Obbligatorio	Soddisfatto
RFO3.3.2	L'utente deve poter visualizzare la descrizione di una singola recensione presente nella lista.	Obbligatorio	Soddisfatto
RFF3.3.2.1	L'utente deve poter visualizzare un messaggio informativo nel caso la descrizione della recensione sia vuota.	Facoltativo	Non Soddisfatto
RFO3.3.3	L'utente deve poter visualizzare l'indirizzo *wallet* dell'attività recensita dalla singola recensione presente nella lista.	Obbligatorio	Soddisfatto
RFO4	L'utente deve poter visualizzare la lista di recensioni che ha rilasciato.	Obbligatorio	Soddisfatto

Continua nella prossima pagina

Tabella 2 – *Continua dalla pagina precedente*

Codice	Descrizione	Classificazione	Fonti
RFO4.1	L'utente deve poter visualizzare un messaggio informativo nel caso non abbia rilasciato alcuna recensione.	Obbligatorio	Soddisfatto
RFO4.2	L'utente deve poter visualizzare un messaggio di errore nel caso un errore di connessione non permetta la visualizzazione delle sue recensioni.	Obbligatorio	Soddisfatto
RFO4.3	L'utente deve poter visualizzare una singola recensione presente nella lista delle proprie recensioni.	Obbligatorio	Soddisfatto
RFO4.3.1	L'utente deve poter visualizzare il parametro di valutazione di una singola recensione presente nella lista delle proprie recensioni.	Obbligatorio	Soddisfatto
RFO4.3.2	L'utente deve poter visualizzare la descrizione di una singola recensione presente nella lista delle proprie recensioni.	Obbligatorio	Soddisfatto
RFF4.3.2.1	L'utente deve poter visualizzare un messaggio informativo nel caso in cui la descrizione della propria recensione sia vuota.	Facoltativo	Non Soddisfatto
RFO4.3.3	L'utente deve poter visualizzare l'indirizzo *wallet* dell'attività recensita dalla singola recensione presente nella lista delle proprie recensioni.	Obbligatorio	Soddisfatto
RFO5	L'utente deve poter effettuare la modifica di una propria recensione rilasciata in precedenza.	Obbligatorio	Soddisfatto
RFO5.1	L'utente deve poter visualizzare un messaggio di errore nel caso un errore di connessione impedisca il completamento della modifica.	Obbligatorio	Soddisfatto
RFO5.2	L'utente deve poter effettuare la modificare del parametro di valutazione di una propria recensione rilasciata in precedenza.	Obbligatorio	Soddisfatto
RFO5.2.1	L'utente deve poter visualizzare un messaggio di errore nel caso in cui non abbia inserito nessun valore del parametro di valutazione.	Obbligatorio	Soddisfatto

Continua nella prossima pagina

Tabella 2 – *Continua dalla pagina precedente*

Codice	Descrizione	Classificazione	Fonti
RFO5.2.2	L'utente deve poter visualizzare un messaggio di errore nel caso in cui il valore modificato del parametro di valutazione non sia valido.	Obbligatorio	Soddisfatto
RFO5.2.3	L'utente deve poter modificare il valore del parametro di valutazione al valore 1	Obbligatorio	Soddisfatto
RFO5.2.4	L'utente deve poter modificare il valore del parametro di valutazione al valore 2	Obbligatorio	Soddisfatto
RFO5.2.5	L'utente deve poter modificare il valore del parametro di valutazione al valore 3	Obbligatorio	Soddisfatto
RFO5.2.6	L'utente deve poter modificare il valore del parametro di valutazione al valore 4	Obbligatorio	Soddisfatto
RFO5.2.7	L'utente deve poter modificare il valore del parametro di valutazione al valore 5	Obbligatorio	Soddisfatto
RFO5.3	L'utente deve poter modificare la descrizione di una propria recensione rilasciata in precedenza.	Obbligatorio	Soddisfatto
RFF5.3.1	L'utente deve poter visualizzare una richiesta di conferma di modifica di una recensione la cui descrizione sia vuota.	Facoltativo	Non Soddisfatto
RFO6	L'utente deve poter eliminare una propria recensione precedentemente rilasciata.	Obbligatorio	Soddisfatto
RFO6.1	L'utente deve poter visualizzare un messaggio di errore nel caso un errore di connessione impedisca il completamento dell'eliminazione.	Obbligatorio	Soddisfatto
RFO7	L'utente deve poter effettuare un pagamento dal proprio *wallet* gestito da *Metamask*.	Obbligatorio	Soddisfatto
RFO7.1	L'utente deve poter visualizzare un messaggio di errore nel caso un errore di connessione impedisca il completamento della transazione.	Obbligatorio	Soddisfatto
RFO7.2	L'utente deve poter inserire l'indirizzo del *wallet* destinatario del pagamento.	Obbligatorio	Soddisfatto
RFO7.2.1	L'utente deve poter visualizzare un messaggio di errore nel caso in cui non abbia inserito alcun indirizzo *wallet*.	Obbligatorio	Soddisfatto

Continua nella prossima pagina

Tabella 2 – *Continua dalla pagina precedente*

Codice	Descrizione	Classificazione	Fonti
RFO7.2.2	L'utente deve poter visualizzare un messaggio di errore nel caso in cui abbia inserito l'indirizzo *wallet* in un formato non valido.	Obbligatorio	Soddisfatto
RFO7.3	L'utente deve poter inserire la quantità di *token ERC20*$_{\mathbf{G}}$ da inviare.	Obbligatorio	Soddisfatto
RFO7.3.1	L'utente deve poter visualizzare un messaggio di errore nel caso in cui non abbia inserito alcuna quantità di *token ERC20* da inviare.	Obbligatorio	Soddisfatto
RFO7.3.2	L'utente deve poter visualizzare un messaggio di errore nel caso in cui abbia inserito una quantità non valida di *token ERC20*.	Obbligatorio	Soddisfatto
RFO8	L'utente deve poter effettuare una ricerca sulle recensioni presenti nel sistema.	Obbligatorio	Soddisfatto
RFO8.1	L'utente deve poter visualizzare un messaggio di errore nel caso un errore di connessione impedisca di effettuare la ricerca.	Obbligatorio	Soddisfatto
RFO8.2	L'utente deve poter inserire l'indirizzo *wallet* di una attività di cui desidera cercare le recensioni.	Obbligatorio	Soddisfatto
RFO8.2.1	L'utente deve poter visualizzare un messaggio di errore nel caso in cui non abbia inserito alcun indirizzo *wallet*.	Obbligatorio	Soddisfatto
RFO8.2.2	L'utente deve poter visualizzare un messaggio di errore nel caso in cui abbia inserito l'indirizzo *wallet* in un formato non valido.	Obbligatorio	Soddisfatto

4 Grafici riassuntivi

Per quanto riguarda il soddisfacimento dei requisiti funzionali, il gruppo è riuscito a soddisfare 57 su 60.

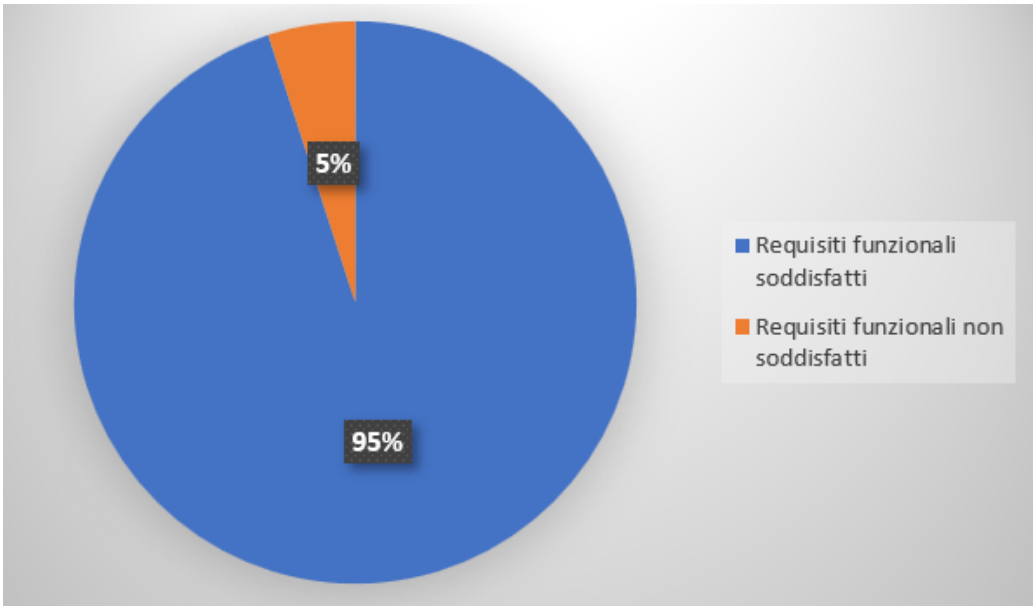

Figura 4: Requisiti funzionali soddisfatti

Mentre per quanto riguarda i requisiti obbligatori il gruppo ne ha soddisfatto 78 su 78.

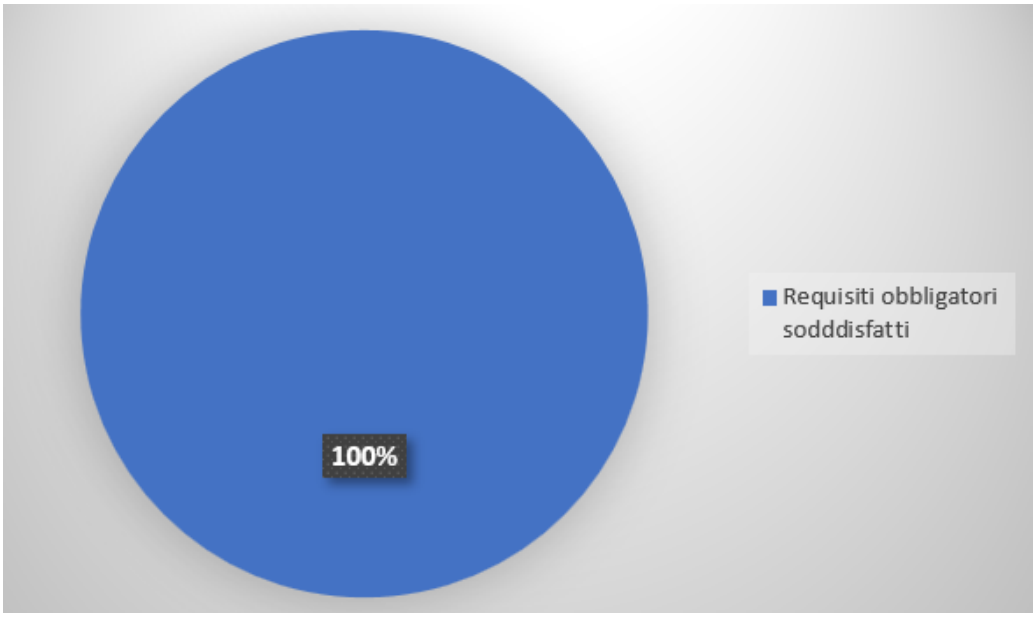

Figura 5: Requisiti obbligatori soddisfatti