



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 | <i>v1.0.0</i> |

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 fron- tend |
| 0.0.2 | 2023/05/3 | Nicola Lazzarin (Progettista) | Marco Brugin | Stesura § componente di bac- kend |
| 0.0.1 | 2023/03/11 | Nicola Lazzarin (Progettista) | Marco Brugin | Creazione struttura documento e stesura Introduzione |

Indice

| | | |
|----------|---|-----------|
| 1 | Introduzione | 1 |
| 1.1 | Scopo del documento | 1 |
| 1.2 | Scopo del prodotto | 1 |
| 1.3 | Glossario | 1 |
| 1.4 | Riferimenti | 1 |
| 1.4.1 | Normativi | 1 |
| 1.4.2 | Informativi | 1 |
| 2 | Architettura | 3 |
| 2.1 | Componente di back-end | 3 |
| 2.1.1 | TCoin | 3 |
| 2.1.1.1 | Metodi | 3 |
| 2.1.2 | <i>Trustify</i> | 3 |
| 2.1.2.1 | Enumerazione | 4 |
| 2.1.2.2 | Strutture dati | 4 |
| 2.1.2.3 | Mappe | 4 |
| 2.1.2.4 | Interfacce e librerie | 5 |
| 2.1.2.5 | Metodi | 5 |
| 2.2 | Componente di front-end | 8 |
| 2.2.1 | <i>Webapp</i> | 8 |
| 2.2.1.1 | <i>Angular Material</i> | 8 |
| 2.3 | Componente di di interazione tra front-end e back-end | 9 |
| 2.3.1 | Servizi <i>Angular</i> | 9 |
| 2.4 | Componente di <i>API</i> | 11 |
| 2.4.1 | Descrizione delle classi | 11 |
| 2.5 | Design pattern utilizzati | 11 |
| 3 | Requisiti soddisfatti | 13 |
| 4 | Grafici riassuntivi | 18 |

Elenco delle tabelle

2 Requisiti Funzionali 13

Elenco delle figure

| | | |
|---|---|----|
| 1 | Diagramma delle classi della componente di back-end | 8 |
| 2 | Diagramma delle classi delle componenti di front-end e di interazione tra front-end e back-end | 10 |
| 3 | Diagramma delle classi della componente di API | 12 |
| 4 | Requisiti funzionali soddisfatti | 18 |
| 5 | Requisiti obbligatori soddisfatti | 18 |

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 architetture adottate dal team di sviluppo durante le attività 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_G* che permetta di rilasciare e visualizzare recensioni certificate tramite uno *smart contract_G* risiedente in una *blockchain_G Ethereum_G* compatibile, al fine di minimizzare la compravendita di recensioni e il *review bombing_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 v2.0.0*

1.4 Riferimenti

1.4.1 Normativi

- *Norme di progetto v2.0.0*
- Regolamento del progetto didattico:

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

- Presentazione Capitolo 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

In modo da renderne più chiara la descrizione le scelte architetturelle attuate dal gruppo di progetto verranno analizzate con granularità sul singolo componente.

2.1 Componente di back-end

Gli (*smart contract*)_G che costituiscono il componente di *back-end* sono stati progettati per essere eseguibili sulla rete *Ethereum*_G e sulle reti basate su *Ethereum*, come imposto dal requisito di vincolo **RVO6**. I due contratti che costituiscono il componente sono:

- **TCoin**: rappresenta un *token*_G usato per il trasferimento di valuta tra gli utenti del sistema;
- **Trustify**: rappresenta la *business logic*_G del sistema, ovvero la gestione di pagamenti e recensioni.

Entrambi sono stati realizzati tramite il linguaggio di programmazione *Solidity*_G, come imposto dal requisito di vincolo **RVO7**.

2.1.1 TCoin

Il contratto *TCoin* estende il contratto *ERC20*_G fornito dalla libreria *OpenZeppelin*, uno standard per la rappresentazione e gestione di *token*_G supportati dalla rete *Ethereum*.

Il contratto *TCoin* rappresenta la valuta usata all'interno del sistema, usata per pagare le imposte collegate al rilascio delle recensioni e per i pagamenti che legittimano gli utenti a rilasciare recensioni. Le funzioni per il trasferimento di *token* tra *wallet*_G vengono fornite dallo contratto *ERC20*, pertanto non è stato necessario ridefinirle nel contratto *TCoin*.

2.1.1.1 Metodi Il contratto *TCoin* espone i seguenti metodi:

- **constructor(string, string)**: costruttore del contratto;
 - **string**: Nome da assegnare al *token*, valorizzato a "TCoin";
 - **string**: Simbolo da assegnare al *token*, valorizzato a "TCoin".
- **drip()**: Genera 100.000 unità del *token* *TCoin* e le aggiunge al *wallet* del chiamante. Utilizza la funzione **mint(string, uint)** fornita dal contratto *ERC20*.

2.1.2 Trustify

Il contratto *Trustify* è stato scritto nel linguaggio di programmazione *Solidity*_G e rappresenta la *business logic*_G del prodotto. Fornisce un sistema di recensioni che consente agli utenti di leggere e scrivere recensioni relative ad aziende e/o servizi, identificati da un indirizzo *wallet*_G univoco. Il contratto è stato progettato per utilizzare apposite mappe per tenere traccia delle recensioni rilasciate degli utenti e delle aziende/negozi che hanno ricevuto recensioni. Il contratto inoltre permette di rilasciare una recensione verso una determinata azienda/servizio solamente agli utenti che abbiano precedentemente effettuato "un pagamento" verso tale azienda, ovvero il deposito di un *token* (nel nostro caso il *token* *TCoin*). Oltre alla funzione di scrittura delle recensioni il contratto include funzioni di lettura recensioni di una specifica azienda/negozio in pagine di dimensione variabile da 1 a 25.

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 realmente cancellata nel contratto ma gli verrà applicato lo stato **DELETED**. Ogni richiesta di lettura formulata al contratto ritornerà anche le recensioni marcate come **DELETED**, delegando al chiamante l'eventuale filtraggio.

Se una recensione è stata erroneamente cancellata e l'utente vuole ripristinarla basterà modificarne il contenuto per aggiornarne lo stato 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;
- **havePaid**: un booleano che indica se l'utente ha eseguito una transazione;
- **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_G* di indirizzi *Ethereum* che contiene gli indirizzi degli utenti che hanno rilasciato una recensione per l'azienda rappresentata dalla struttura dati;
- **reviewMap**: una mappa che associa ad ogni recensione relativa all'azienda rappresentata dalla struttura dati **Company** l'indirizzo dell'utente autore della 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 dall'utente rappresentato dalla struttura dati;
- **reviewMap**: una mappa che associa ad ogni indirizzo di azienda o esercizio commerciale la recensione rilasciata dall'utente rappresentato dalla struttura dati.

Queste strutture vengono usate per memorizzare dati nel contratto e poterli 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 propria struttura dati **Company** e ad ogni indirizzo di utente la propria struttura dati **Customer**.

- **companyMap**: una mappa che associa ad ogni indirizzo di azienda o esercizio commerciale la propria struttura dati **Company**;
- **customerMap**: una mappa che associa ad ogni indirizzo di utente la propria 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 havePaid(address myAddress, address companyAddress) public view returns (bool) :`

Funzione di sola lettura che controlla se uno specifico utente abbia effettuato un pagamento che lo autorizzi a rilasciare una recensione per una determinata azienda. Prende in input gli indirizzi dell'utente (`myAddress`) e 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 `texttt{havePaid}`: Se il valore è `true`, l'utente ha pagato per la recensione, altrimenti non ha pagato. La funzione non modifica lo stato delle mappe, pertanto 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 nella mappa `reviewMap` della `Company` relativa all'indirizzo dell'azienda e all'interno della mappa `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 del chiamante della funzione.
- `WriteAReview`: permette di scrivere una recensione sull'azienda rappresentata dall'indirizzo passato come parametro. L'indirizzo non può coincidere con l'indirizzo del 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 dal parametro `addressToDeposit`, mentre il numero di *token* da depositare viene specificato dal parametro `amount`. Prima di effettuare il deposito, viene verificato che l'utente abbia inserito un indirizzo diverso dal proprio tramite il modificatore `checkActionToYourself`. Solo in caso di esito positivo i *token* vengono trasferiti dal portafoglio dell'utente al portafoglio dell'azienda, specificato dal parametro `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 a `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 riceve un valore dell'enumerazione `ReviewState` e restituisce una stringa che rappresenta lo stato corrispondente.
 - Lo stato `ACTIVE` restituirà la stringa *"ACTIVE"*;
 - Lo stato `MODIFIED` restituirà la stringa *"MODIFIED"*;
 - Lo stato `DELETED` restituirà la stringa *"DELETED"*.

Questa funzione viene utilizzata all'interno del contratto per convertire i valori dell'enumerazione `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 ad un utente di scrivere una recensione per l'azienda specificata dal parametro `addressToReview`. Il corpo della recensione e il numero di stelle sono specificati dai 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 recensioni scritte da un numero specificato di clienti per una determinata azienda, insieme ad un *array* di numeri interi rappresentanti 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` ed `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 comprese nel range di indici indicato.

- `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 dal parametro `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.

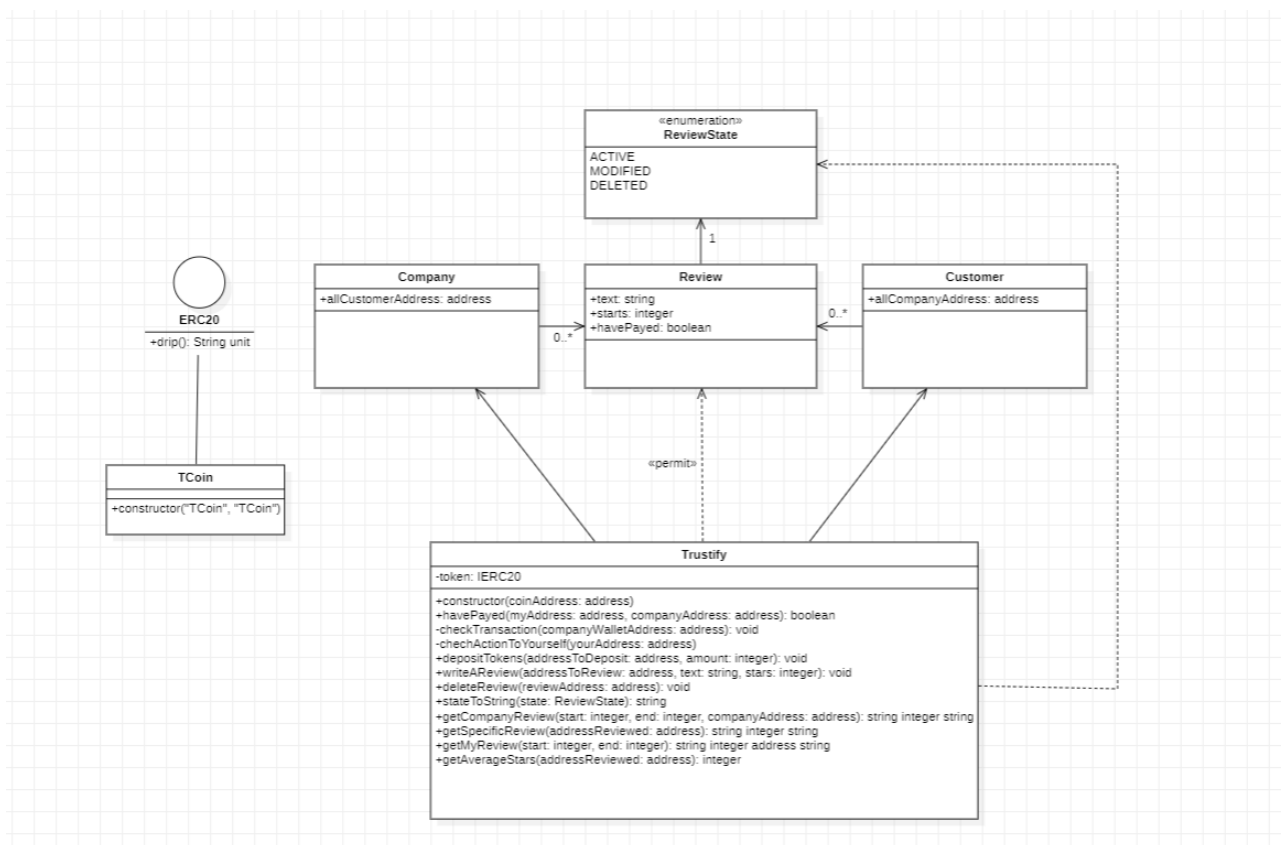


Figura 1: Diagramma delle classi della componente di back-end

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_G* per il sistema *Trustify* è stata creata utilizzando *Angular_G*, uno dei *framework_G open source_G* più utilizzati per lo sviluppo di applicazioni web moderne. Inoltre, il team di sviluppo ha scelto di utilizzare *TypeScript_G* 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 manutenibile, testabile e scalabile.

2.2.1.1 Angular Material *Angular Material* è una libreria che aggiunge ad *Angular* lo stile grafico *Material* sviluppato da Google. Fornisce componenti prefabbricati per l'interfaccia utente (UI) agli sviluppatori che usano *Angular*. Nel progetto è stato fatto ampio uso di questa libreria al fine di ottenere un aspetto grafico uniforme spendendo il minor numero possibile di ore da *Programmatore*, in modo da poter dedicare più tempo alla codifica della *business logic*.

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

L'obiettivo di questa componente è di fornire un'interfaccia per la comunicazione tra *front-end* e *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 Servizi *Angular*

In *Angular*, un "servizio" identifica qualsiasi insieme di funzionalità che vogliamo rendere disponibili a componenti multipli. La *web app* fa uso di svariati servizi, di cui i due principali sono:

- **WalletService** La classe **WalletService** definisce un servizio per la gestione della connessione con il *wallet MetaMask_G* e la comunicazione con la *blockchain_G*. Il servizio espone metodi per verificare la presenza dell'estensione *MetaMask* e se l'utente sia connesso ad esso. Inoltre, il servizio permette di ottenere l'indirizzo dell'account con cui l'utente si sia connesso a *MetaMask* e l'URL del provider HTTP di *Infura_G* per la rete di test *Sepolia*.

Metodi principali:

- **getAccount()**: metodo asincrono che restituisce l'indirizzo dell'account con cui l'utente si sia connesso a *MetaMask*. L'indirizzo dell'account viene utilizzato per inviare e ricevere transazioni sulla *blockchain*;
 - **isInstalled()**: metodo asincrono che verifica la presenza del *plugin MetaMask* nel browser dell'utente. Necessario poiché l'interazione con *MetaMask* è obbligatoria per gestire la connessione con la *blockchain*;
 - **isWalletConnected()**: metodo asincrono che restituisce **true** se l'utente è attualmente connesso a *MetaMask*;
 - **connectToMetamask()**: metodo asincrono che apre la finestra di dialogo di connessione di *MetaMask* e, nel caso in cui l'utente accetti 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()**: 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 sia già presente in *MetaMask*, altrimenti la aggiunge.
 - **connect()**: metodo asincrono usato come *wrapper* per i metodi **connectToMetamask()** e **switchNetwork()**. Usato per connettersi a *MetaMask* e passare alla rete *Sepolia*.
- **Web3Service** La classe **Web3Service** rappresenta un servizio *Angular* che fornisce una serie di funzionalità per interagire con la *blockchain Ethereum* tramite la libreria *Web3.js*. In particolare, la classe fornisce metodi per effettuare operazioni con i contratti (*TCoin* e *Trustify*) presenti sulla *blockchain*.

Metodi principali:

- **pullTCoin**: metodo asincrono che invia una transazione allo *smart contract TCOin* per richiedere l'invio di *token*;
- **getTokenBalance**: metodo asincrono che restituisce il saldo di *token TCOin* dell'account con cui il servizio comunica;
- **approveTokens**: metodo asincrono che richiede l'autorizzazione per la spesa di un determinato numero di *token TCOin* allo *smart contract Trustify*;

- **depositTokens**: metodo asincrono che invia una transazione allo *smart contract Trustify* per depositare un determinato numero di *token TCoin*;
- **writeAReview**: metodo asincrono che invia una transazione allo *smart contract Trustify* per scrivere una recensione per un’azienda;
- **getCompanyReview**: metodo asincrono che restituisce un *array* contenente le recensioni delle aziende tra un intervallo di indici;
- **getSpecificReview**: metodo asincrono che restituisce la recensione e il punteggio in stelle associati a un’azienda specifica;
- **getMyReview**: metodo asincrono che restituisce la recensione e il punteggio in stelle dell’utente corrente per una determinata azienda;
- **deleteReview**: metodo asincrono che invia una transazione al contratto *Trustify* per cancellare la recensione dell’utente corrente per una determinata azienda.

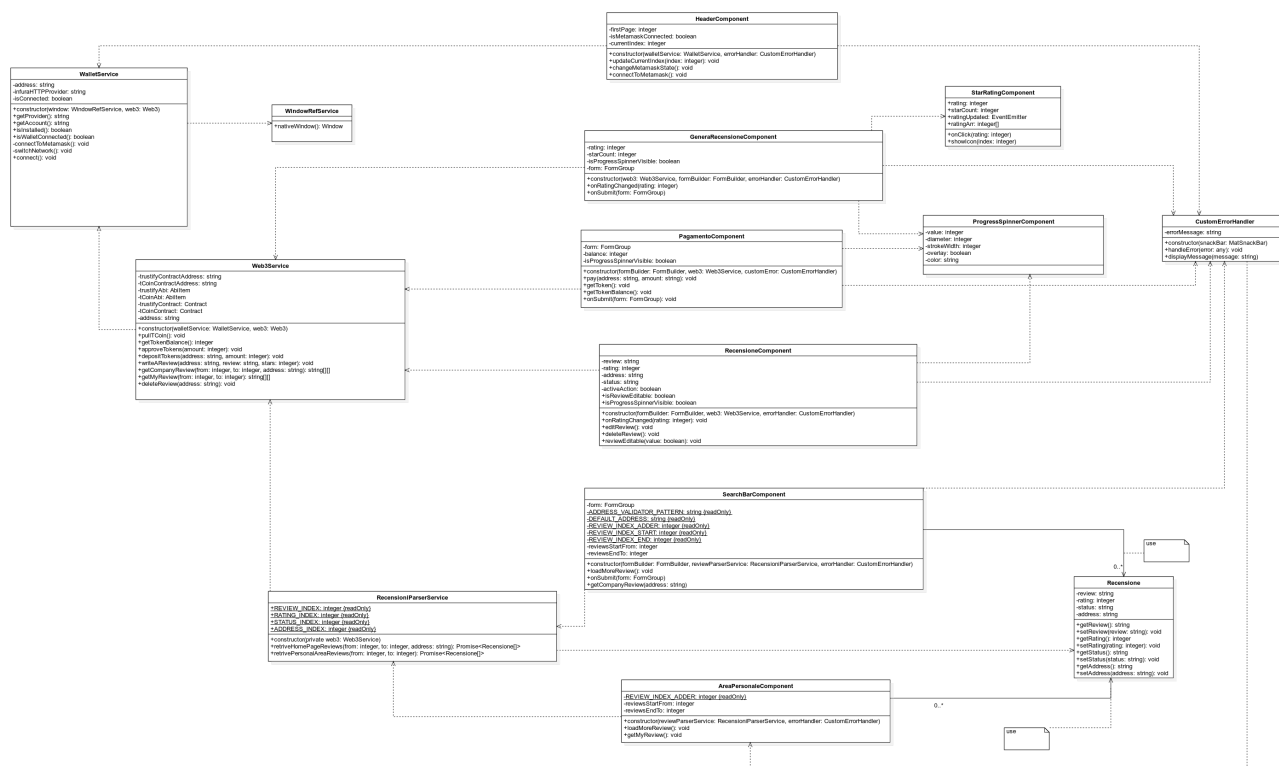


Figura 2: Diagramma delle classi delle componenti di front-end e di interazione tra front-end e back-end

2.4 Componente di *API*

L'obiettivo di questa componente è di fornire agli *e-commerce_G* un'interfaccia per poter dialogare con la *blockchain_G Ethereum_G* e poter usufruire dei servizi offerti da *Trustify_G*. Per far ciò è stata sviluppata una *API_G* conforme allo stile architetturale *REST_G*. La componente di *API* è stata sviluppata utilizzando il *framework_G Spring Boot* e il linguaggio di programmazione *Java_G*.

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_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.

2.5 Design pattern utilizzati

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

- **Dependency Injection**: è stato utilizzato nella *web app* per iniettare i servizi nei componenti dell'applicazione. L'implementazione di questo pattern viene fornita dal *framework_G Angular*.

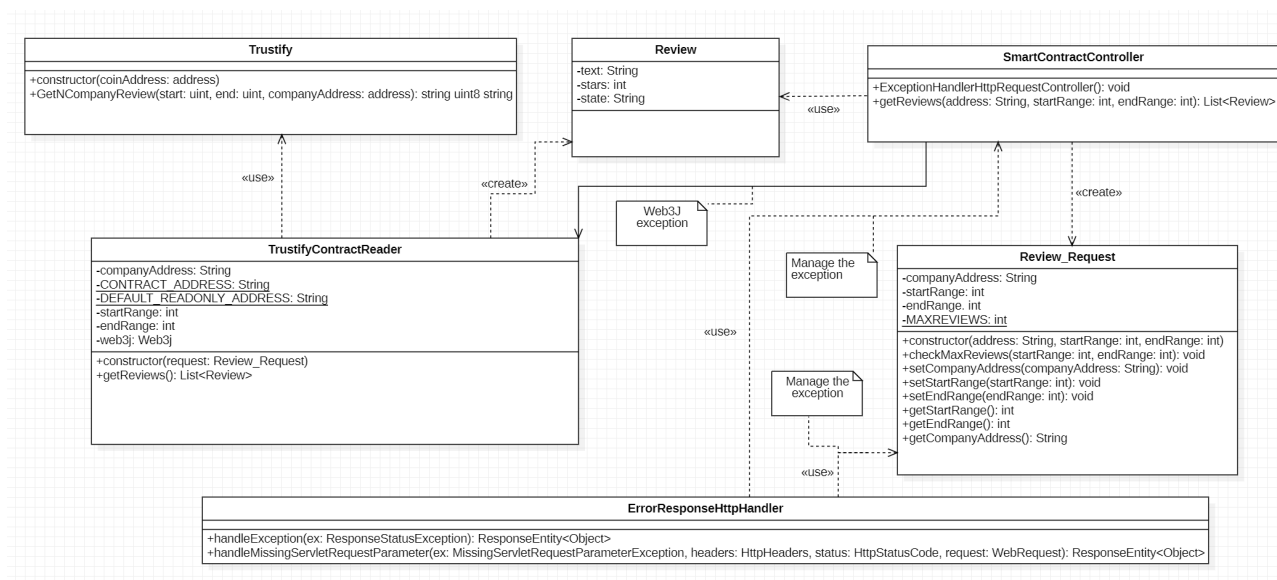


Figura 3: Diagramma delle classi della componente di API

3 Requisiti soddisfatti

Tabella 2: Requisiti Funzionali

| Codice | Descrizione | Classificazione | Fonti |
|----------|---|-----------------|-------------|
| RFO1 | L'utente deve poter collegare il proprio <i>wallet_G</i> alla <i>webapp_G</i> tramite <i>Metamask_G</i> . | Obbligatorio | Soddisfatto |
| RFO1.1 | L'utente deve poter visualizzare un messaggio di errore nel caso <i>Metamask</i> non sia installato. | Obbligatorio | Soddisfatto |
| RFO1.2 | L'utente deve poter visualizzare un messaggio informativo quando il proprio <i>wallet</i> 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 <i>wallet</i> . | Obbligatorio | Soddisfatto |
| RFO2.2.2 | L'utente deve poter visualizzare un messaggio di errore nel caso l'indirizzo <i>wallet</i> 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 <i>wallet</i> 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 <i>wallet</i> 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 <i>wallet</i> gestito da <i>Metamask</i> . | 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 <i>wallet</i> 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 <i>wallet</i> . | 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 <i>wallet</i> in un formato non valido. | Obbligatorio | Soddisfatto |
| RFO7.3 | L'utente deve poter inserire la quantità di <i>token ERC20_G</i> 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 <i>token ERC20</i> 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 <i>token ERC20</i> . | 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 <i>wallet</i> 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 <i>wallet</i> . | Obbligatorio | Soddisfatto |
| RFO8.2.2 | L'utente deve poter visualizzare un messaggio di errore nel caso in cui abbia inserito l'indirizzo <i>wallet</i> 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.

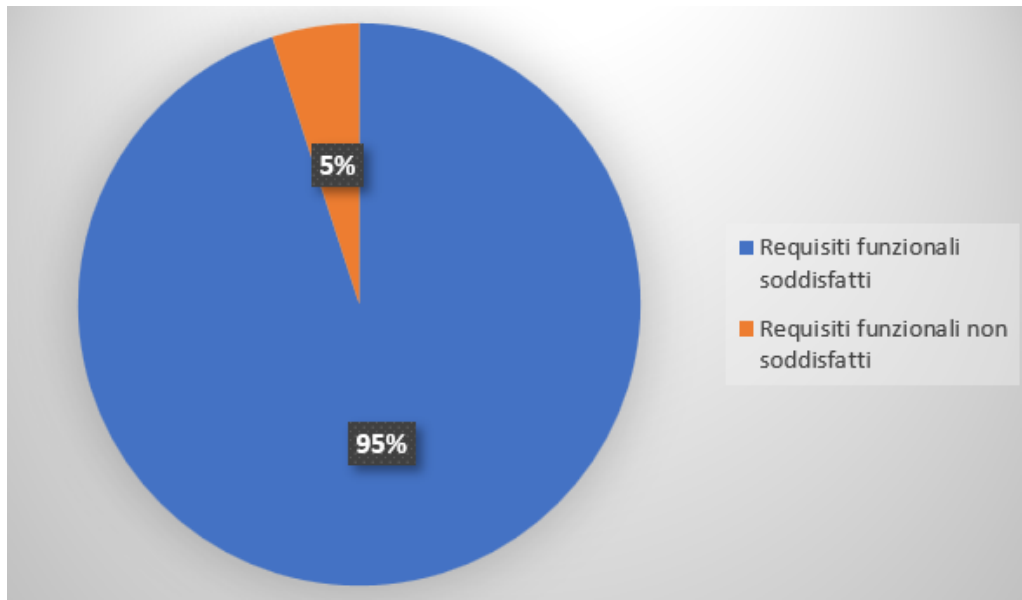


Figura 4: Requisiti funzionali soddisfatti

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

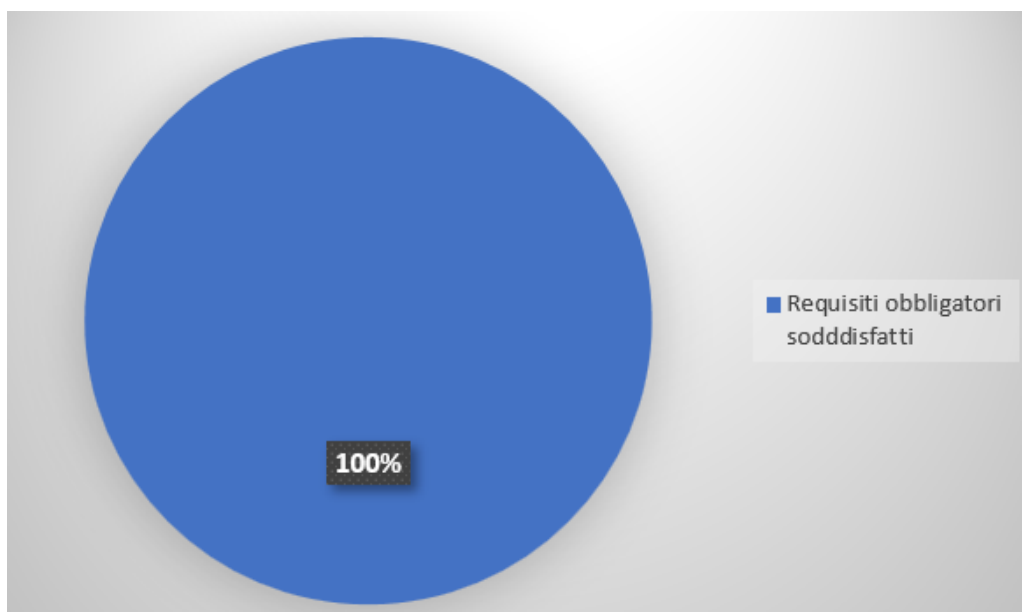


Figura 5: Requisiti obbligatori soddisfatti