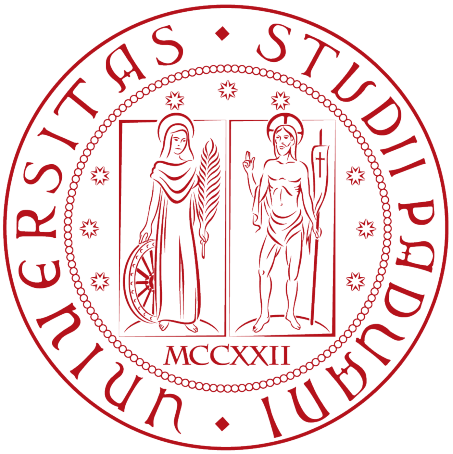
Università degli studi di Padova
Dipartimento di Matematica "Tullio Levi-Civita"
Corso di Laurea in Informatica



Storystorm

Gestore portabile di contenuti social interattivi

Tesi di Laurea

Relatore
Professor Navarin Nicolò

Laureando
Gallo Edoardo
Matricola 2042357

Anno Accademico 2023-2024

"Sì, ne è valsa la pena
Che pena, però
Io ti prometto che sarà bellissimo"

— Quei ricordi là - Olly, JVLI.

Ringraziamenti

Desidero innanzitutto esprimere la mia gratitudine al Professor Nicolò Navarin, relatore della mia tesi, per essere stato sempre disponibile a chiarire prontamente ogni mio dubbio durante il percorso di stage e durate la stesura della tesi.

Ringrazio infinitamente mia mamma e mio papà per aver sempre creduto in me e per non avermi mai fatto mancare nulla delle cose che contano nella vita.
Grazie a mio fratello, ai miei nonni, ai miei zii e ai miei cugini per esserci stati fin dall'inizio e per avermi sempre sostenuto.
Un grazie a chi è al mio fianco, perché sta rendendo questo momento, come tutti gli altri, sempre più speciale.

Desidero poi ringraziare tutti i miei amici per avermi accompagnato in questo percorso e che, tra esperienze di vita, scuola, università, parrocchia e lavoro, mi hanno reso la persona che sono oggi.

Padova, Dicembre 2024 *Gallo Edoardo*

Sommario

Il presente documento descrive il lavoro svolto durante il periodo di stage, dall'analisi delle tecnologie e dei requisiti per lo sviluppo del prodotto finale al completamento dello stesso.
Storystorm nasce come soluzione ad un problema: cercare di aumentare la fidelizzazione dei propri utenti in modo semplice ed economico. E quale migliore soluzione se non quella di integrare facilmente nei propri applicativi un componente che abbiamo tutti imparato ad usare in questi ultimi anni? Le storie social. Grazie al loro formato, e alla cancellazione dopo 24 ore, questi contenuti invogliano gli utenti a tornare sulla piattaforma per non perdersi i vari aggiornamenti.
Storystorm è un servizio che, previa registrazione, permette la creazione delle varie storie social e fornisce all'acquirente un modo facile e veloce per integrarle dove vuole, mantenendo il costo il più basso possibile.

Indice

Elenco delle figure

Elenco delle tabelle

Elenco dei codici sorgenti

Capitolo 1

Introduzione

1.1 L'azienda

HNRG è nata come start up nel 2009 e ad oggi è diventata l'anima più digitale di Altea Federation, che da oltre 30 anni collabora con i leader mondiali dell'innovazione tecnologica.
Con le sue 6 sedi in tutta Italia e oltre 120 dipendenti, HNRG oggi ha come obiettivo portare l'essere umano al centro di ogni interazione digitale, migliorando i processi e i risultati dei suoi clienti.
In poche parole, grazie anche alla collaborazione con aziende e servizi di alto livello (uno dei quali essenziale per il completamento di questo progetto), l'azienda cerca di offrire un'ampia rosa di soluzioni, personalizzate secondo le varie esigenze, ai propri clienti.

Figura 1.1: Logo dell'azienda HNRG

1.2 La scelta dell'azienda

L'individuazione dell'azienda è stata molto diretta e senza problemi e mi venne suggerita da un conoscente che già lavorava all'interno di una delle *company* di Altea Federation. Dopo un breve colloquio conoscitivo, è stato elaborato un piano di lavoro adeguato e in poco tempo tutto era già pronto per iniziare.

1.3 Organizzazione del testo

1.3.1 Struttura del documento

Il documento è diviso in 9 capitoli e illustra in maniera dettagliata l'esperienza di stage:

Il primo capitolo è una breve introduzione all'azienda in cui si è svolto lo stage.

Il secondo capitolo descrive il lavoro svolto durante lo stage, approfondendo gli obiettivi e l'idea del progetto.

Il terzo capitolo delinea le informazioni raccolte in fase di analisi del progetto tramite requisiti e casi d'uso.

Il quarto capitolo introduce le tecnologie e gli strumenti essenziali che sono stati usati durante tutto il corso dello stage, da quelli di sviluppo fino a quelli organizzativi.

Il quinto capitolo approfondisce la prima delle due tecnologie essenziali utilizzate nel progetto: *Directus*.

Il sesto capitolo affronta nel dettaglio *Stencil*, l'altra tecnologia fondamentale del progetto.

Il settimo capitolo descrive le problematiche sull'accessibilità e la fase di test svolta nell'ultima settimana di stage.

L'ottavo capitolo mostra un esempio concreto di come l'applicazione sviluppata andrebbe utilizzata, integrandola in un sito esterno (*Demeter*).

Il nono capitolo analizza gli obiettivi raggiunti e le prospettive future del progetto, oltre a fornire una valutazione personale dell'esperienza di stage.

1.3.2 Convenzioni tipografiche

Per quanto riguarda la stesura e lo stile del testo nel documento, sono state adottate le seguenti convenzioni tipografiche:

- i termini in lingua straniera o del gergo tecnico sono evidenziati dallo stile *corsivo*;
- le abbreviazioni e gli acronimi sono scritti per la prima volta come *parola (abbreviazione)* mentre per le successive compare solo dall'acronimo;
- le parole del glossario all'interno del documento sono evidenziate e seguite da una lettera G ad apice.

Capitolo 2

Descrizione dello stage

2.1 L'esigenza iniziale

Come ogni progetto, anche questo stage curricolare nasce da un'esigenza. L'azienda aveva bisogno di aumentare la fidelizzazione degli utenti delle piattaforme che venivano sviluppate per i loro clienti.
Per raggiungere tale obiettivo, era necessario una soluzione semplice, di uso quotidiano e soprattutto adattabile a qualsiasi applicazione.
La soluzione proposta per il progetto di stage era già stata analizzata dall'azienda ma poi messa da parte per mancanza di tempo e risorse per verificarne la fattibilità.

2.2 Il progetto

Il progetto consiste nello sviluppo di una componente applicativa (definita *Widget*), integrabile in qualsiasi applicativo web, che permetta l'esposizione di contenuti multimediali (definite *Stories*).
Storystorm è un gestore portabile di contenuti social interattivi. Permette infatti di essere integrato in un qualsiasi sito web, consentendo agli utenti che lo visitano di visualizzare ed interagire con le ormai rinomate storie social che tutti hanno imparato a conoscere ed apprezzare. Tramite una soluzione di gestione (definita *Backoffice*), il servizio permette la creazione e la modifica di questi contenuti multimediali, la visualizzazione delle loro bozze e risultati delle interazioni.
Implementando le storie social in un qualsiasi applicativo web, si risolve quella che era l'esigenza dell'azienda. Grazie alle loro funzioni (come i *mi piace* e i sondaggi) e soprattutto alla cancellazione dopo 24 ore, si aumenta la fidelizzazione degli utenti, invogliandoli a tornare sulla piattaforma per rimanere aggiornati su qualunque tema siano interessati.

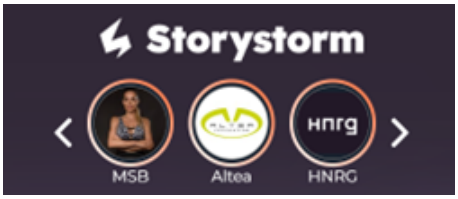


Figura 2.1: Esempio di storie social integrate tramite Storystorm

2.3 Analisi delle soluzioni già presenti

Data la rapida ed esponenziale espansione del web negli ultimi decenni, è logico pensare che l'idea di Storystorm sia già stata implementata. Ed effettivamente è così poiché esistono già diversi servizi che offrono la creazione e l'integrazione di storie social nelle proprie applicazioni.
Uno dei problemi principali di questi servizi è l'elevato costo. Altri invece sono gratuiti ma solo per un numero limitato di storie oppure non hanno tutte le funzionalità disponibili.

Utenti mensili	Costo al mese	Costo all'anno
fino a **10.000**	$500,00	**$6.000,00**
100.000	$1.200,00	**$14.400,00**
1 milione	$3.300,00	**$39.600,00**
10 milioni	$8.600,00	**$103.200,00**

Figura 2.2: Lista prezzi di uno dei *competitors*

L'obiettivo principale di Storystorm è offrire un prodotto che possa essere competitivo sul mercato a livello di funzionalità ma anche su quello economico.

2.4 Obiettivi

Si farà riferimento ai requisiti secondo le seguenti notazioni:

- **Ob** per i requisiti obbligatori, vincolanti in quanto obiettivo primario richiesto dal committente;
- **D** per i requisiti desiderabili, non vincolanti o strettamente necessari, ma dal riconoscibile valore aggiunto;

- **Op** per i requisiti opzionali, rappresentanti valore aggiunto non strettamente competitivo.

Le sigle precedentemente indicate saranno seguite da un numero identificativo del requisito.
Si prevede lo svolgimento dei seguenti obiettivi:

Obbligatori	
Ob1	Progettazione e realizzazione della struttura *database* per la gestione dei contenuti e relative informazioni
Ob2	Realizzazione/configurazione delle pagine di *backoffice* per permettere la creazione/gestione dei contenuti
Ob3	Realizzazione ed esposizione di *Application Programming Interface (API)*[G] *layer* per la fruizione dei contenuti
Ob4	Realizzazione della libreria[G] JavaScript (*Widget*)
Ob5	Scrittura dei casi di test
Desiderabili	
D1	Realizzazione dei test automatici
D2	Inserimento della libreria[G] in un applicativo di demo sviluppato con *framework*[G] *JS (Angular/React/Vue)*
Opzionali	
Op1	Integrazione dell'Intelligenza Artificiale (IA)[G] per la generazione di contenuti multimediali

Tabella 2.1: Tabella degli obiettivi dello stage

2.5 Pianificazione del lavoro

Fase 1 - Formazione (40 ore)
Scopo: formazione sulle problematiche, tool e tecnologie che si incontreranno durante lo stage. In dettaglio:

- linguaggi e tecnologie: *JavaScript, Sass, Directus, Stencil*;
- ambiente di sviluppo e tool di supporto: *Bitbucket, VsCode, Jira*;
- linee guida e metodologie: *Atomic Design, Design System.*

Fase 2 - Analisi e progettazione (40 ore)
Scopo: analisi delle specifiche funzionali e progettazione tecnica della soluzione da realizzare. In dettaglio:

- analisi delle specifiche funzionali;
- progettazione in dettaglio;
- documentazione.

Fase 3 - Implementazione (180 ore)
Scopo: preparazione dell'ambiente di sviluppo e implementazione della soluzione. In dettaglio:

- implementazione tecnologica dei requisiti definiti in fase di analisi;
- documentazione tecnica.

Fase 4 - Test e Verifica (60 ore)
Scopo: test funzionale, verifica di conformità dell'implementazione rispetto ai requisiti e stesura della documentazione. In dettaglio:

- verifica funzionale;
- verifica di conformità rispetto ai requisiti;
- revisione e correzione di eventuali bug;
- documentazione di validazione ed esito dei test;
- pubblicazione delle modifiche nel *repository*^{G} dei sorgenti aziendali.

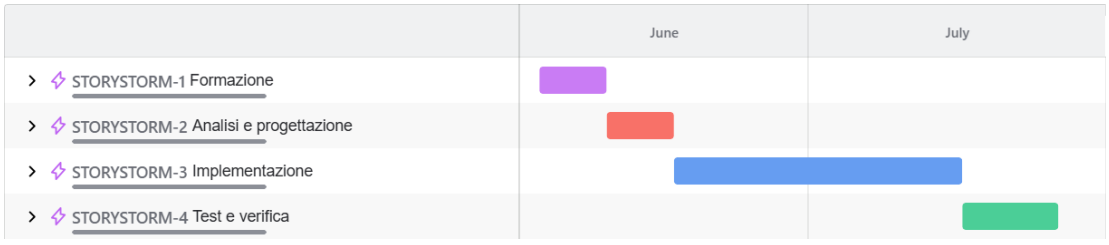


Figura 2.3: Diagramma di GANTT preventivo

Capitolo 3

Analisi dei requisiti

3.1 Casi d'uso

Prima di poter procedere con la codifica della soluzione, è essenziale fare un passo indietro. Analizzare per bene quali sono le tecnologie da utilizzare e soprattutto come impostare le fasi successive del progetto per evitare di dover tornare indietro per qualche errore di comunicazione o comprensione.
Un modo semplice e veloce per tracciare questi requisiti è rappresentare simbolicamente i dati e le informazioni raccolte sul progetto. I diagrammi dei casi d'uso ([4]) sono dei diagrammi di tipo *Unified Modeling Language (UML)*^{G} che danno una rappresentazione grafica alle varie interazione che un utente può avere con un sistema, facilitando la comprensione e l'analisi dei requisiti.
Ogni caso d'uso rappresenta uno scenario di utilizzo del programma da parte di un utente (che d'ora in poi verrà definito come *attore*) e, oltre alla rappresentazione grafica, sarà correlato dalle seguenti informazioni:

- Descrizione: breve definizione del caso d'uso;
- Attori principali: utenti che eseguono le azioni descritte;
- Precondizioni: serie di condizioni che devono avvenire prima che l'utente svolga l'azione;
- Postcondizioni: stato del programma dopo il caso d'uso;
- Scenario principale: serie di azioni svolte prima, durante e dopo il caso d'uso dall'utente;
- Generalizzazioni: se presenti, aggiungono o modificano le caratteristiche base di un caso d'uso;
- Estensioni: casi d'uso collegati a quello descritto (come ad esempio i messaggi d'errore).

3.1.1 Attori

Gli attori sono i vari tipi di utenti che interagisco con il sistema. Nel caso di Storystorm, gli utenti possono essere categorizzati in maniera più dettagliata, in base a se possiedono o meno le credenziali per accedere al *backoffice*:

- Utente non autenticato: utente che, se provvisto di credenziali può effettuare il *login*, altrimenti può solo visualizzare e interagire con le storie nelle applicazioni in cui il *widget* viene integrato;
- Utente autenticato: utente che ha le credenziali per accedere al *backoffice* e può creare le storie e visualizzarne i risultati;
- Cliente: utente che ha acquistato il servizio. Può svolgere tutte le funzioni di un *Utente autenticato* ma può inoltre creare nuove istanze, in modo tale da non dover gestire tutto da solo;
- Amministratore: è il fornitore del servizio. Possiede tutti i permessi e quando viene effettuato un acquisto, deve creare le istante degli utenti *Cliente* e fornire loro il *token*^{G} per inserire il *widget* nella loro applicazione.

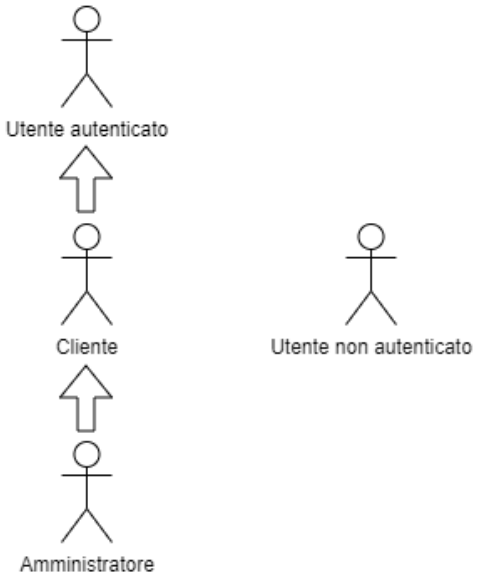


Figura 3.1: Attori in Storystorm

3.1.2 Elenco dei casi d'uso

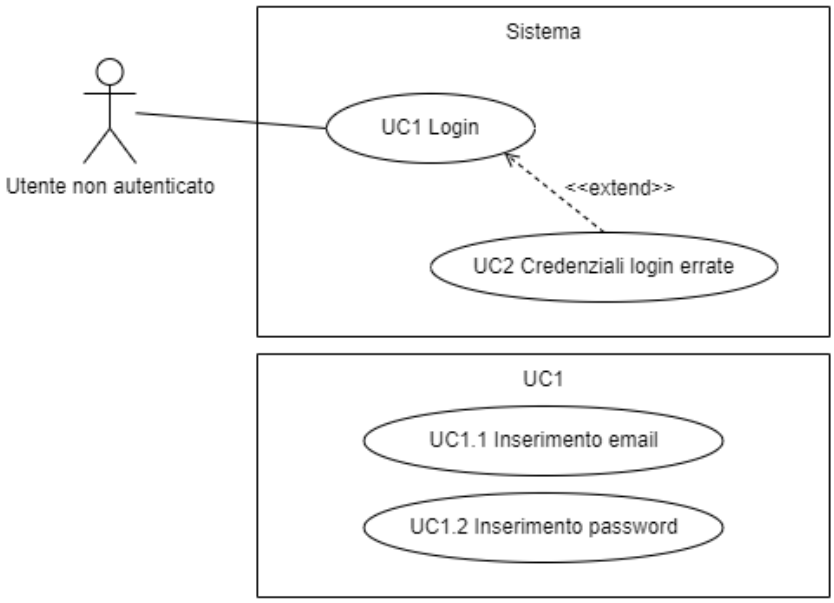


Figura 3.2: UC1-UC2: *Login* - Credenziali *login* errate

UC1 - *Login*

Attori Principali: Utente non autenticato.

Precondizioni: L'utente non ha ancora effettuato l'accesso e possiede le credenziali per effettuarlo.

Descrizione: L'utente, se in possesso di credenziali, può effettuare il login all'interno del *backoffice*.

Postcondizioni: L'utente viene riconosciuto dal sistema.

Scenario Principale:

1. L'utente inserisce la propria *email* (UC1.1);
2. L'utente inserisce la propria password (UC1.2);
3. Il sistema verifica che le credenziali inserite siano corrette.

Estensioni:

- Viene mostrato un messaggio di errore se le credenziali non sono corrette (UC2).

UC1.1 - Inserimento *email*

Attori Principali: Utente non autenticato.

Precondizioni: L'utente sta effettuando il *login* (UC1).

Descrizione: L'utente inserisce l'*email* fornita in fase di registrazione.

Postcondizioni: L'utente inserisce correttamente la propria *email*.

Scenario Principale:

1. L'utente inserisce la propria *email*.

UC1.2 - Inserimento password

Attori Principali: Utente non autenticato.

Precondizioni: L'utente sta effettuando il *login* (UC1).

Descrizione: L'utente inserisce la password fornita in fase di registrazione.

Postcondizioni: L'utente inserisce correttamente la propria password.

Scenario Principale:

1. L'utente inserisce la propria password.

UC2 - Credenziali *login* errate

Attori Principali: Utente non autenticato.

Precondizioni: L'utente sta effettuando il *login* (UC1) e ha già inserito *email* (UC1.1) e password (UC1.2).

Descrizione: L'utente visualizza un messaggio di errore.

Postcondizioni: L'utente non viene riconosciuto correttamente dal sistema.

Scenario Principale:

1. L'utente non viene riconosciuto da sistema;
2. Viene visualizzato un messaggio di errore di autenticazione.

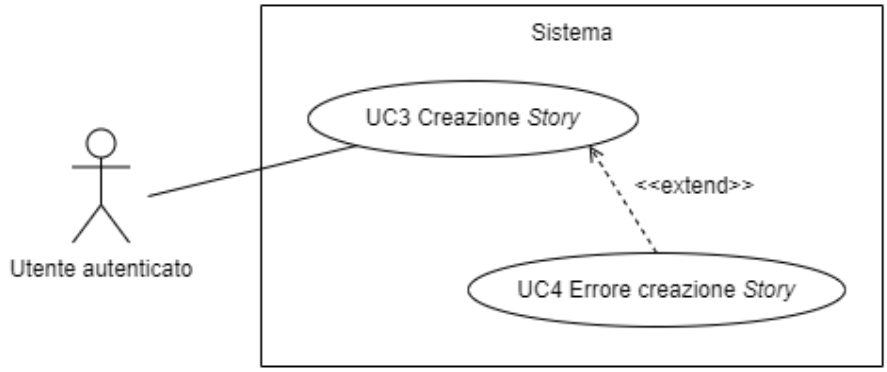


Figura 3.3: UC3-UC4: Creazione *Story* - Errore creazione *Story*

UC3 - Creazione *Story*

Attori Principali: Utente autenticato, Cliente, Amministratore.

Precondizioni: L'utente ha effettuato il *login* (UC1).

Descrizione: L'utente deve creare una *story* (assieme ai componenti aggiuntivi come testi, link, sondaggi...).

Postcondizioni: La *story* viene creata in stato di bozza.

Scenario Principale:

1. Tramite *backoffice*, l'utente compila il form[G] di creazione di una *story*.

Estensioni:

- Viene mostrato un messaggio di errore se la *story* non è stata creata correttamente (UC4).

UC4 - Errore creazione *Story*

Attori Principali: Utente autenticato, Cliente, Amministratore.

Precondizioni: L'utente ha effettuato l'accesso (UC1) e sta creando una *story* (UC3).

Descrizione: L'utente visualizza un messaggio di errore sulla creazione di una *story*.

Postcondizioni: La *story* non viene creata correttamente.

Scenario Principale:

1. Alla *story* creata mancano dei dettagli oppure esiste già una storia con lo stesso nome;

2. Viene visualizzato un messaggio di errore.

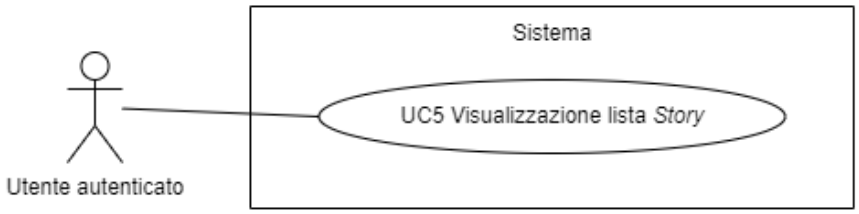


Figura 3.4: UC5: Visualizzazione lista *Story*

UC5 - Visualizzazione lista *Story*

Attori Principali: Utente autenticato, Cliente, Amministratore.

Precondizioni: L'utente ha effettuato l'accesso (UC1).

Descrizione: L'utente visualizza una lista delle *stories*.

Postcondizioni: L'utente visualizza una lista delle sue *stories* e quelle create dagli utenti che ha generato.

Scenario Principale:

1. Il sistema mostra una lista delle *stories* che l'utente può visualizzare;
2. L'utente può visualizzare la singola *story* nel dettaglio (UC6), vederne una bozza (UC9) oppure cambiarne lo stato (UC10).

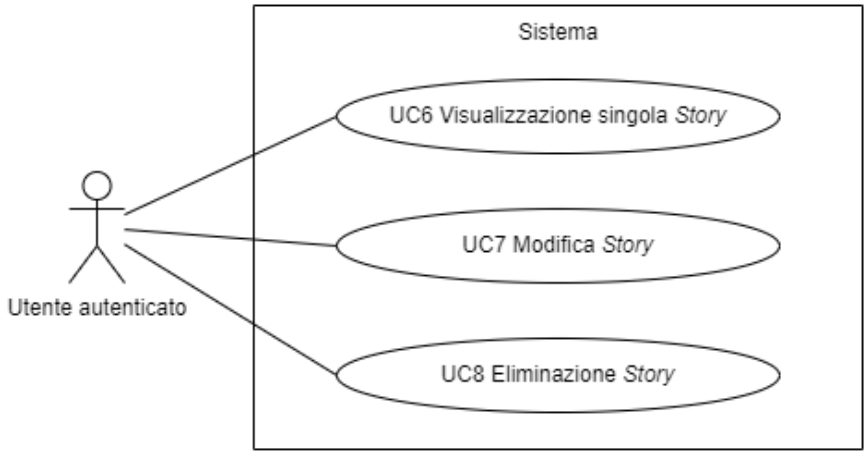


Figura 3.5: UC6-UC7-UC8: Visualizzazione, modifica ed eliminazione di una *Story*

UC6 - Visualizzazione singola *Story*

Attori Principali: Utente autenticato, Cliente, Amministratore.

Precondizioni: L'utente ha effettuato l'accesso (UC1) e sta visualizzando una lista delle *stories* permesse dal sistema (UC5).

Descrizione: L'utente visualizza nel dettaglio una singola *story*.

Postcondizioni: L'utente visualizza un form[G] già compilato con i dati della *story* selezionata.

Scenario Principale:

1. Il sistema mostra la *story* nel dettaglio, con possibilità di modificarla (UC7) o eliminarla (UC8).

UC7 - Modifica *Story*

Attori Principali: Utente autenticato, Cliente, Amministratore.

Precondizioni: L'utente ha effettuato l'accesso (UC1) e sta visualizzando una singola *story* (UC6).

Descrizione: L'utente modifica la *story* che sta visualizzando.

Postcondizioni: Il sistema aggiorna la *story* modificata.

Scenario Principale:

1. L'utente modifica la *story* selezionata;
2. Il sistema verifica che le modifiche effettuate siano applicabili.

Estensioni:

- Viene mostrato un messaggio di errore se la *story* non è stata modificata correttamente (UC4).

UC8 - Eliminazione *Story*

Attori Principali: Utente autenticato, Cliente, Amministratore.

Precondizioni: L'utente ha effettuato l'accesso (UC1) e sta visualizzando una singola *story* (UC6).

Descrizione: L'utente elimina la *story* che sta visualizzando.

Postcondizioni: La *story* viene eliminata dal sistema.

Scenario Principale:

1. L'utente preme il pulsante per eliminare la *story* selezionata.
2. Il sistema mostra un messaggio per chiedere conferma dell'eliminazione;
3. In caso di conferma, la *story* viene eliminata. Altrimenti l'utente ritorna alla visualizzazione della *story*.

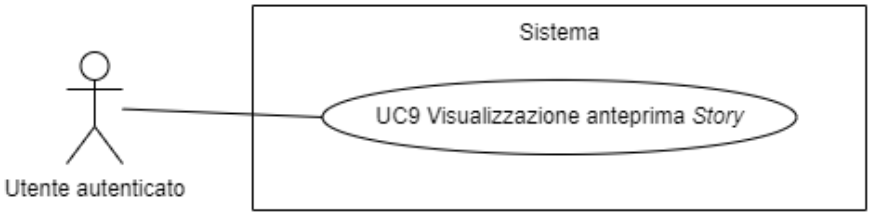


Figura 3.6: UC9: Visualizzazione anteprima *Story*

UC9 - Visualizzazione anteprima *Story*

Attori Principali: Utente autenticato, Cliente, Amministratore.

Precondizioni: L'utente ha effettuato l'accesso (UC1) e sta visualizzando una lista delle *stories* permesse dal sistema (UC5).

Descrizione: L'utente deve poter visualizzare le bozze delle *stories* all'interno della lista.

Postcondizioni: L'utente visualizza l'anteprima della *story* selezionata.

Scenario Principale:

1. Il sistema mostra un'anteprima della *story*;
2. L'utente, dopo aver valutato, può tornare indietro per modificare la *story* (UC7) oppure cambiare il suo stato (UC10).

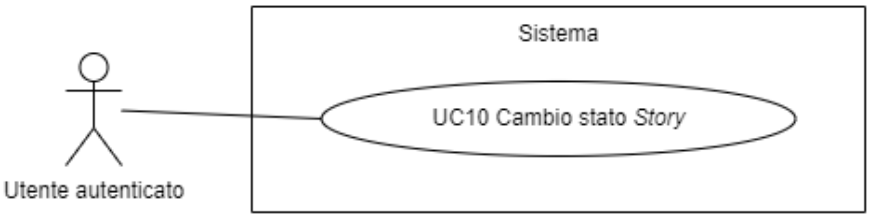


Figura 3.7: UC10: Cambio stato *Story*

UC10 - Cambio stato *Story*

Attori Principali: Utente autenticato, Cliente, Amministratore.

Precondizioni: L'utente ha effettuato l'accesso (UC1) e sta visualizzando una lista delle *stories* permesse dal sistema (UC5).

Descrizione: L'utente deve poter cambiare lo stato delle *stories* all'interno della lista. Una *story* può essere: una bozza, archiviata o pubblicata.

Postcondizioni: L'utente cambia lo stato della *story* in uno di quelli predefiniti.

Scenario Principale:

1. L'utente preme il pulsante per cambiare lo stato della *story* selezionata.
2. Il sistema mostra un messaggio per chiedere conferma del cambio;
3. In caso di conferma, lo stato della *story* viene modificato. Altrimenti l'utente ritorna alla visualizzazione della lista delle *stories*.

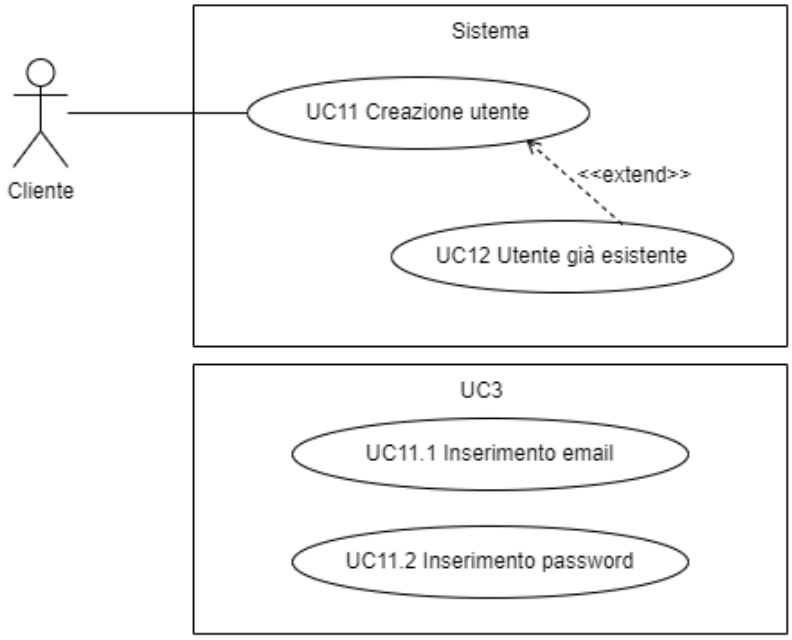


Figura 3.8: UC11-UC12: Creazione utente - Utente già esistente

UC11 - Creazione utente

Attori Principali: Cliente, Amministratore.

Precondizioni: L'utente ha effettuato il *login* (UC1).

Descrizione: L'utente deve poter creare un'istanza di un altro utente con un ruolo di livello inferiore al suo nella gerarchia.

Postcondizioni: La nuova istanza utente viene creata con un livello di permessi

inferiore a quello dell'utente creatore.

Scenario Principale:

1. L'utente accede al form[G] di creazione di un utente;
2. Inserisce l'*email* del nuovo utente (UC11.1);
3. Inserisce la password del nuovo utente (UC11.2);
4. Il sistema verifica che non esista già un altro utente con la stessa *email*.

Estensioni:

- Viene mostrato un messaggio di errore se l'*email* del nuovo utente è già presente nel sistema (UC12).

UC11.1 - Inserimento *email*

Attori Principali: Cliente, Amministratore.

Precondizioni: L'utente ha effettuato il *login* (UC1) e sta creando una nuova istanza di utente (UC11).

Descrizione: L'utente inserisce l'*email* del nuovo utente.

Postcondizioni: L'utente inserisce correttamente l'*email* del nuovo utente.

Scenario Principale:

1. L'utente inserisce l'*email* della nuova istanza di utente.

UC11.2 - Inserimento password

Attori Principali: Cliente, Amministratore.

Precondizioni: L'utente ha effettuato il *login* (UC1) e sta creando una nuova istanza di utente (UC11).

Descrizione: L'utente inserisce la password del nuovo utente.

Postcondizioni: L'utente inserisce correttamente la password del nuovo utente.

Scenario Principale:

1. L'utente inserisce la password della nuova istanza di utente.

UC12 - Utente già esistente

Attori Principali: Cliente, Amministratore.

Precondizioni: L'utente ha effettuato il *login* (UC1), sta creando una nuova istanza di utente (UC11) e ha già inserito *email* (UC11.1) e password (UC11.2).

Descrizione: L'utente visualizza un messaggio di errore sulla creazione del nuovo utente.

Postcondizioni: La nuova istanza di utente non viene creata correttamente.

Scenario Principale:

1. Al nuovo utente mancano dei dettagli oppure esiste già un utente con la stessa *email*;
2. Viene visualizzato un messaggio di errore.

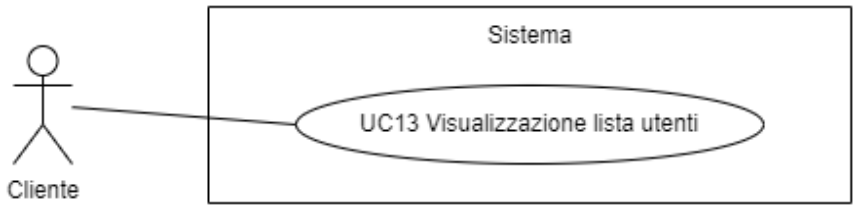


Figura 3.9: UC13: Visualizzazione lista utenti

UC13 - Visualizzazione lista utenti

Attori Principali: Cliente, Amministratore.

Precondizioni: L'utente ha effettuato l'accesso (UC1).

Descrizione: L'utente visualizza una lista degli utenti.

Postcondizioni: L'utente visualizza una lista degli utenti da lui creati e se stesso (l'utente amministratore visualizza una lista di tutti gli utenti della piattaforma).

Scenario Principale:

1. Il sistema mostra una lista degli utenti che l'utente può visualizzare;
2. L'utente può visualizzare un singolo utente nel dettaglio (UC14).

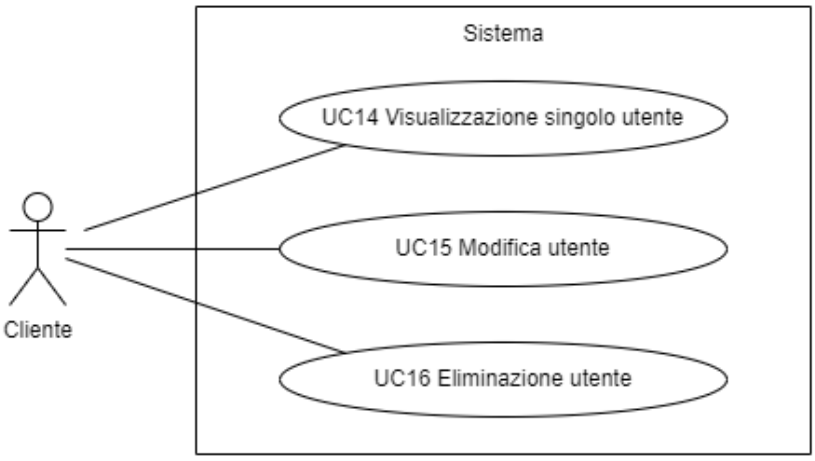


Figura 3.10: UC14-UC15-UC16: Visualizzazione, modifica ed eliminazione di un utente

UC14 - Visualizzazione singolo utente

Attori Principali: Cliente, Amministratore.

Precondizioni: L'utente ha effettuato l'accesso (UC1) e sta visualizzando una lista degli utenti permessi dal sistema (UC13).

Descrizione: L'utente visualizza nel dettaglio un singolo utente da lui creato (oppure se stesso).

Postcondizioni: L'utente visualizza un form[G] già compilato con i dati dell'utente selezionato.

Scenario Principale:

1. Il sistema mostra l'utente nel dettaglio, con possibilità di modificarlo (UC15) o eliminarlo (UC16).

UC15 - Modifica utente

Attori Principali: Cliente, Amministratore.

Precondizioni: L'utente ha effettuato l'accesso (UC1) e sta visualizzando un singolo utente (UC14).

Descrizione: L'utente modifica l'utente che sta visualizzando.

Postcondizioni: Il sistema aggiorna l'utente modificato.

Scenario Principale:

1. L'utente modifica l'utente selezionato;
2. Il sistema verifica che le modifiche effettuate siano applicabili.

Estensioni:

- Viene mostrato un messaggio di errore se l'utente non è stato modificato correttamente (UC12).

UC16 - Eliminazione utente

Attori Principali: Cliente, Amministratore.

Precondizioni: L'utente ha effettuato l'accesso (UC1) e sta visualizzando un singolo utente (UC14).

Descrizione: L'utente elimina l'utente che sta visualizzando.

Postcondizioni: L'utente viene eliminato dal sistema.

Scenario Principale:

1. L'utente preme il pulsante per eliminare l'utente selezionato;
2. Il sistema mostra un messaggio per chiedere conferma dell'eliminazione;
3. In caso di conferma, l'utente viene eliminato. Altrimenti l'utente ritorna alla visualizzazione degli utenti.

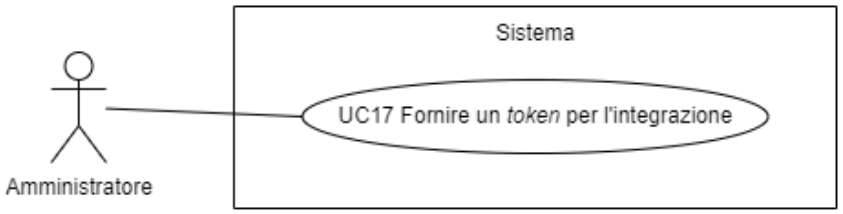


Figura 3.11: UC17: Fornire un *token* per l'integrazione

UC17 - Fornire un *token* per l'integrazione

Attori Principali: Amministratore.

Precondizioni: L'utente ha effettuato l'accesso (UC1) e si trova nella pagina di dettaglio di un utente (UC14).

Descrizione: L'utente deve poter fornire all'utente cliente che ha acquistato il servizio, un *token*^{G} univoco per l'integrazione del *widget* nella sua pagina web.

Postcondizioni: L'utente invia il codice univoco ed identificativo all'utente cliente.

Scenario Principale:

1. L'utente crea l'istanza del nuovo utente cliente (UC11);
2. L'utente si reca nella pagina di visualizzazione del nuovo utente (UC14);
3. Copia il codice univoco del nuovo utente che fungerà da *token*^{G} e lo invia al nuovo utente.

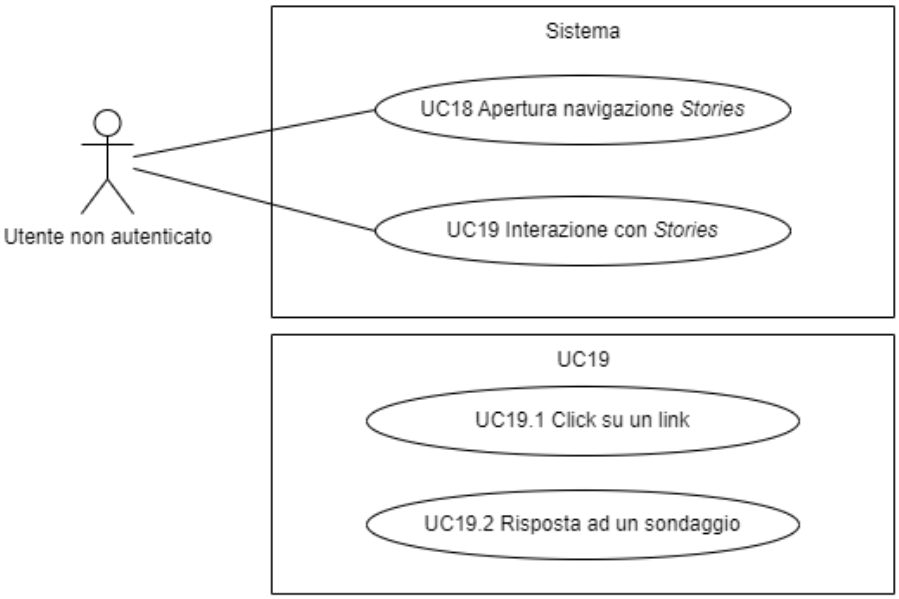


Figura 3.12: UC18-UC19: Apertura navigazione ed interazione con *Stories*

UC18 - Aperture navigazione *Stories*

Attori Principali: Utente non autenticato.

Precondizioni: L'utente deve trovarsi in una delle pagine in cui il *widget* è integrato.

Descrizione: L'utente deve poter aprire la schermata di navigazione delle *stories*.

Postcondizioni: L'utente naviga tra le storie presenti nel sito web che sta visitando.

Scenario Principale:

1. L'utente visita un sito in cui il *widget* è integrato;
2. L'utente preme su una delle storie che vuole visualizzare;
3. L'utente, navigando tra le *stories*, può: interagire con i suoi componenti (UC19), mettere *mi piace* (UC20), condividerle (UC21), fermare o riprendere la riproduzione (UC22), mutare o riattivare il volume (UC23) oppure chiudere la schermata di navigazione (UC24).

UC19 - Interazione con *Stories*

Attori Principali: Utente non autenticato.

Precondizioni: L'utente deve trovarsi in una delle pagine in cui il *widget* è integrato e aver aperto la schermata di navigazione (UC18).

Descrizione: L'utente deve poter interagire con i vari componenti delle *stories* (se presenti).

Postcondizioni: L'utente, navigando tra le storie presenti, può interagire con i componenti al loro interno.

Scenario Principale:

1. L'utente visita un sito in cui il *widget* è integrato;
2. L'utente preme su una delle storie che vuole visualizzare;
3. L'utente, navigando tra le *stories*, può interagire con i suoi componenti (se presenti): fare click su un link (UC19.1) e rispondere ad un sondaggio (UC19.2).

UC19.1 - Click su un link

Attori Principali: Utente non autenticato.

Precondizioni: L'utente deve trovarsi in una delle pagine in cui il *widget* è integrato e aver aperto la schermata di navigazione (UC18).

Descrizione: L'utente deve poter fare click sul link inserito nella storia (se presente).

Postcondizioni: L'utente viene reindirizzato al link della pagina.

Scenario Principale:

1. L'utente visita un sito in cui il *widget* è integrato;
2. L'utente preme su una delle storie che vuole visualizzare;
3. L'utente preme sul link integrato in una delle storie (se presente);
4. Appare un *popup* per chiedere conferma dell'azione all'utente e la visualizzazione della storia viene messa in pausa;
5. In caso affermativo, l'utente viene reindirizzato al link premuto. Altrimenti la visualizzazione della storia riprende.

UC19.2 - Risposta ad un sondaggio

Attori Principali: Utente non autenticato.

Precondizioni: L'utente deve trovarsi in una delle pagine in cui il *widget* è integrato e aver aperto la schermata di navigazione (UC18).

Descrizione: L'utente deve poter rispondere al sondaggio inserito nella storia (se presente).

Postcondizioni: Il sistema registra il voto dell'utente e mostra in percentuale i voti del sondaggio.

Scenario Principale:

1. L'utente visita un sito in cui il *widget* è integrato;
2. L'utente preme su una delle storie che vuole visualizzare;
3. L'utente risponde ad un sondaggio integrato in una delle storie (se presente);
4. Il sistema registra il voto dell'utente, aumentando di uno i voti totali e quelli della risposta votata;
5. Vicino alle risposte viene mostrata la percentuale dei voti fino ad allora registrati.

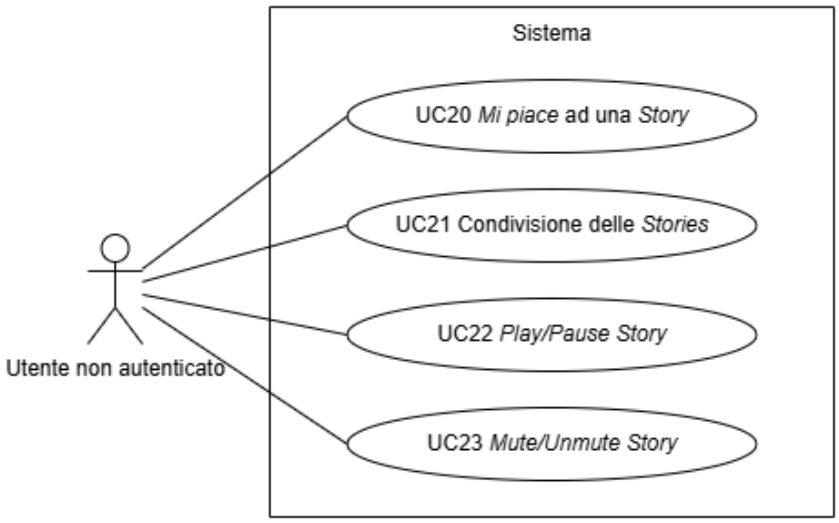


Figura 3.13: UC20-UC21-UC22-UC23: Interazioni primarie con *Stories*

UC20 - *Mi piace* ad una *Story*

Attori Principali: Utente non autenticato.

Precondizioni: L'utente deve trovarsi in una delle pagine in cui il *widget* è integrato e aver aperto la schermata di navigazione (UC18).

Descrizione: L'utente deve poter mettere o togliere *mi piace* alle *stories*.

Postcondizioni: Il sistema registra l'avvenuta azione e ne mostra il risultato a schermo.

Scenario Principale:

1. L'utente visita un sito in cui il *widget* è integrato;
2. L'utente preme su una delle storie che vuole visualizzare;
3. L'utente, navigando tra le *stories*, può mettere oppure togliere *mi piace*;
4. Il sistema aumenta o diminuisce di uno il totale dei *mi piace* della storia in questione e mostra il risultato dell'azione all'utente.

UC21 - Condivisione delle *Stories*

Attori Principali: Utente non autenticato.

Precondizioni: L'utente deve trovarsi in una delle pagine in cui il *widget* è integrato e aver aperto la schermata di navigazione (UC18).

Descrizione: L'utente deve poter condividere le *stories* ad altre persone.

Postcondizioni: Il sistema apre la schermata di condivisione del sistema operativo (se non presente verrà copiato il link della storia).

Scenario Principale:

1. L'utente visita un sito in cui il *widget* è integrato;
2. L'utente preme su una delle storie che vuole visualizzare;
3. L'utente preme il pulsante per condividere una storia;
4. Il sistema apre la schermata di condivisione del sistema operativo (oppure copia il link della storia se questa funzionalità non è presente).

UC22 - *Play/Pause Story*

Attori Principali: Utente non autenticato.

Precondizioni: L'utente deve trovarsi in una delle pagine in cui il *widget* è integrato e aver aperto la schermata di navigazione (UC18).

Descrizione: L'utente deve poter mettere in pausa o riprendere la visualizzazione delle *stories*.

Postcondizioni: Il sistema blocca o riprende la visualizzazione della storia (nel

caso il contenuto fosse un video, esso verrà messo in pausa o riprenderà in base all'azione eseguita).

Scenario Principale:

1. L'utente visita un sito in cui il *widget* è integrato;
2. L'utente preme su una delle storie che vuole visualizzare;
3. L'utente preme il pulsante per fermare o riprendere la riproduzione di una storia;
4. La storia viene messa in pausa o riprende in base all'azione richiesta dall'utente.

UC23 - *Mute/Unmute Story*

Attori Principali: Utente non autenticato.

Precondizioni: L'utente deve trovarsi in una delle pagine in cui il *widget* è integrato, aver aperto la schermata di navigazione (UC18) e il contenuto della storia deve essere un video.

Descrizione: L'utente deve poter mutare oppure riattivare il volume della storia visualizzata.

Postcondizioni: Il sistema muta o riattiva il volume della storia.

Scenario Principale:

1. L'utente visita un sito in cui il *widget* è integrato;
2. L'utente preme su una delle storie che vuole visualizzare;
3. L'utente preme il pulsante per mutare o riattivare il volume una storia;
4. Il volume del video della storia viene tolto o riattivato in base all'azione richiesta dall'utente.

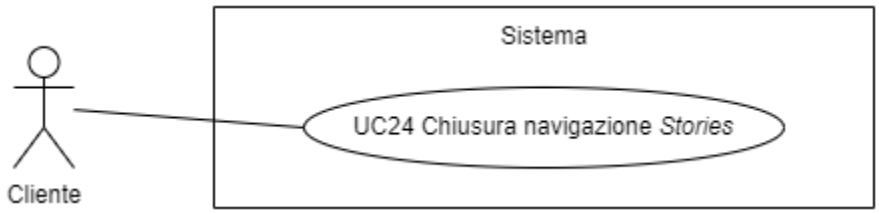


Figura 3.14: UC24: Chiusura navigazione *Stories*

UC24 - Chiusura navigazione *Stories*

Attori Principali: Utente non autenticato.

Precondizioni: L'utente deve trovarsi in una delle pagine in cui il *widget* è integrato e aver aperto la schermata di navigazione (UC18).

Descrizione: L'utente deve poter chiudere la schermata di navigazione delle *stories*.

Postcondizioni: La schermata viene chiusa e l'utente può continuare la navigazione sul sito che sta visitando.

Scenario Principale:

1. L'utente visita un sito in cui il *widget* è integrato;
2. L'utente preme su una delle storie che vuole visualizzare;
3. L'utente, dopo aver navigato tra le *stories*, chiude la schermata di navigazione.

3.2 Requisiti

Dopo aver raccolto e rappresentato graficamente le informazioni raccolte sul progetto tramite i diagrammi UML[G], si può procedere con il tracciamento dei requisiti veri e propri. Per una migliore suddivisione, i requisiti saranno descritti in tre tabelle, una per i requisiti funzionali, una per quelli di vincolo e una per quelli di qualità. Ogni requisito viene identificato dalla seguente struttura:

R[Tipo][Numero requisito]

Dove il tipo può essere:

- **F**, per i requisiti funzionali;
- **V**, per i requisiti di vincolo;
- **Q**, per i requisiti di qualità.

Oltre al codice, ogni requisito si compone di una breve descrizione, della classificazione (che può essere **Ob** se obbligatorio, **D** se desiderabile oppure **Op** nel caso fosse opzionale) e della fonte del requisito stesso (**UC** se la fonte è un caso d'uso, **PdL** se il requisito viene dal piano di lavoro oppure **Interno** se individuato assieme al tutor aziendale).

Cod.	**Descrizione**	**Class.**	**Fonti**
RF1	L'utente che possiede le credenziali deve poter accedere al *backoffice*	Ob	UC1
RF2	L'utente deve visualizzare un messaggio di errore nel caso in cui le credenziali fossero errate	Ob	UC2
RF3	L'utente autenticato deve poter creare una nuova storia direttamente dall'elenco	Ob	UC3
RF4	Ogni utente autenticato deve poter includere nelle proprie *stories* elementi aggiuntivi (come link, sondaggi...)	Ob	UC3
RF5	L'utente autenticato deve visualizzare un messaggio di errore se la creazione di una *story* non è andata a buon fine	Ob	UC4
RF6	L'utente autenticato deve poter visualizzare la lista delle *stories* da lui caricate (o caricate dagli utenti da lui creati)	Ob	UC5
RF7	L'utente autenticato deve poter visualizzare nel dettaglio ogni *story* da lui creata (o creata dagli utenti da lui creati)	Ob	UC6
RF8	Dalla pagina di dettaglio, ogni utente autenticato deve poter modificare le *stories* da lui create (o create dagli utenti da lui creati)	Ob	UC7
RF9	Dalla pagina di dettaglio, ogni utente autenticato deve poter eliminare le *stories* da lui create (o create dagli utenti da lui creati)	Ob	UC8
RF10	Prima della pubblicazione di una *story*, l'utente autenticato deve poterne visualizzare un'anteprima	Ob	UC9
RF11	Dalla lista delle *stories* che l'utente autenticato può visualizzare deve essere possibile cambiare il loro stato	Ob	UC10
RF12	Il cliente deve poter creare un'istanza di utente direttamente dal *backoffice*	Ob	UC11
RF13	Il cliente deve visualizzare un messaggio di errore se la creazione di un utente non è andata a buon fine	Ob	UC12
RF14	Il cliente deve poter visualizzare una lista degli utente da lui creati (e se stesso)	Ob	UC13
Continua nella prossima pagina...			

Continuazione della tabella 3.1			
Cod.	**Descrizione**	**Class.**	**Fonti**
RF15	Il cliente deve poter visualizzare nel dettaglio gli utenti da lui creati e se stesso	Ob	UC14
RF16	Il cliente deve poter modificare un utente tra quelli visualizzabili	Ob	UC15
RF17	Il cliente deve poter eliminare l'account degli utenti da lui creati	Ob	UC16
RF18	L'amministratore deve poter fornire ad un cliente il *token*^{G} per l'integrazione del *widget*	Ob	UC17
RF19	L'utente non autenticato deve poter aprire la schermata di visualizzazione delle *stories* e navigare tra di esse	Ob	UC18
RF20	L'utente non autenticato deve poter interagire con i componenti aggiuntivi delle storie	Ob	UC19
RF21	L'utente non autenticato deve poter accedere ai link inseriti nelle *stories*	Ob	UC19.1
RF22	L'utente non autenticato deve poter rispondere ai sondaggi inseriti nelle *stories*	Ob	UC19.2
RF23	L'utente non autenticato deve poter mettere *mi piace* alle storie	Ob	UC20
RF24	L'utente non autenticato deve poter condividere le *stories*	Ob	UC21
RF25	L'utente non autenticato deve poter mettere in pausa o riprendere la visualizzazione delle storie	Ob	UC22
RF26	L'utente non autenticato deve poter mutare o riattivare il volume dei video delle storie	Ob	UC23
RF27	L'utente non autenticato deve poter chiudere la schermata di navigazione delle *stories*	Ob	UC24
RF28	L'utente autenticato deve poter inserire immagini generate dall'intelligenza artificiale direttamente dal *backoffice*	Op	PdL

Tabella 3.1: Tabella dei requisiti funzionali

Cod.	Descrizione	Class.	Fonti
RQ1	Il codice sorgente del *backend*^{G} deve essere presente su *Bitbucket* all'interno di un *repository*^{G}	Ob	Interno
RQ2	Il codice sorgente del *frontend*^{G} deve essere presente su *Bitbucket* all'interno di un *repository*^{G}	Ob	Interno
RQ3	Il codice *frontend*^{G} deve essere coperto da test di unità	Ob	PdL
RQ4	L'applicazione deve essere *responsive* ed accessibile da tutti i dispositivi mobili e dai principali *browser*	Ob	Interno

Tabella 3.2: Tabella dei requisiti di qualità

Cod.	Descrizione	Class.	Fonti
RV1	Il *backend*^{G} deve essere sviluppato utilizzando *Directus*	Ob	PdL
RV2	Il *frontend*^{G} deve essere sviluppato utilizzando *JavaScript* e *Stencil*	Ob	PdL

Tabella 3.3: Tabella dei requisiti di vincolo

Capitolo 4

Tecnologie e strumenti

Storystorm si compone di due parti essenziali per il funzionamento dell'intero applicativo. Il *backend*^{G} è gestito interamente da *Directus* in modo da facilitare lo sviluppo e concentrarsi maggiormente sul *frontend*^{G}, che viene sviluppato tramite *Stencil*, un compilatore che permette di creare componenti web personalizzati e versatili. Entrambe queste tecnologie vengono approfondite nei rispettivi capitoli.
Questo capitolo si limita ad introdurre tutti gli strumenti, le tecnologie e i linguaggi che sono stati utilizzati per lo sviluppo dell'applicazione.

4.1 Tecnologie e linguaggi principali

JavaScript

JavaScript è un linguaggio di programmazione orientato agli eventi. Sviluppato nel 1995, al giorno d'oggi questo linguaggio di *scripting* è supportato da tutti i *browser* moderni e viene utilizzato per creare potenti applicazioni dinamiche. *JavaScript* permette di interfacciarsi con il *Document Object Model (DOM)*^{G} dei *browser* e aggiornare dinamicamente la struttura, lo stile e il contenuto dei vari elementi che compongono una pagina web.
Oltre alla programmazione lato *client*, *JavaScript* può essere anche utilizzato per la programmazione lato server grazie a *Node.js*.

TypeScript

TypeScript ([23]) è un linguaggio di programmazione di alto livello. Al contrario di *JavaScript*, che ha una tipizzazione dinamica, *TypeScript* aggiunge la tipizzazione statica delle variabili e consente di controllare costantemente i tipi durante la fase di compilazione. Assegnando ad ogni variabile il proprio tipo, questo viene sempre controllato, permettendo di evitare errori ed evidenziare comportamenti inaspettati del codice, diminuendo quindi la possibilità di *bug*.

Come *JavaScript*, anche *TypeScript* può essere utilizzato sia per la programmazione lato *client* sia per quella lato server.

CSS - Sass

Il CSS (*Cascading Style Sheets*, in italiano Fogli di Stile a Cascata) è un potente linguaggio che permette di applicare diversi aspetti di visualizzazione agli elementi HTML. Un foglio di stile è un insieme di regole, con diverse priorità, da applicare agli elementi a cui si vuole cambiare qualche aspetto grafico.
Nonostante la sua potenza, questo linguaggio non è perfetto. Molte funzionalità che potrebbero migliorare l'usabilità, la leggibilità e la comprensione non sono ancora disponibili. È qui che interviene un preprocessore: Sass (*Syntactically Awesome Style Sheets*, [18]). Sass consente di utilizzare variabili, funzioni e molto altro, migliorando anche la leggibilità di un foglio di stile. Una volta scritte alcune regole in Sass, si può compilare il file e ricavarne uno analogo ma scritto usando il linguaggio CSS.

Node.js - npm - nvm

Node.js ([17]) è un ambiente di *runtime* che permette di eseguire codice *JavaScript* al di fuori del *browser* grazie al motore *JavaScript V8* di *Google Chrome*. Questo aiuta gli sviluppatori a costruire applicazioni web, col linguaggio *JavaScript*, non più solo lato *client* ma anche lato server, migliorando di molto anche le prestazioni.
Con *Node.js* viene installato anche *Node Package Manager (npm)*, il gestore di pacchetti *JavaScript* per eccellenza. Gli sviluppatori sono così in grado di installare e gestire facilmente questi pacchetti e utilizzarli nelle loro applicazioni.
All'interno del progetto, sia *Directus* sia *Stencil* utilizzano *Node.js*, ma hanno bisogno di versioni differenti (in particolare *Directus* ha come requisito l'uso di *Node.js* versione 18.17 o superiore). Proprio per questo motivo, anche il sito ufficiale di npm ([16]) consiglia l'uso di un *Node Version Manager* (come ad esempio nvm), in modo da poter cambiare velocemente la versione di *Node.js* senza dover reinstallarlo completamente.

Vue.js

Vue.js ([24]) è un *framework*^{G} *JavaScript* utilizzato per costruire interfacce utente. Nel progetto è stato necessario per implementare direttamente all'interno del *backoffice* di *Directus* una pagina personalizzata che aggiungesse alcune funzionalità utili e individuate in fase di analisi dei requisiti.

Vue.js è stato utilizzato anche per completare uno degli obiettivi desiderabili previsti (precisamente il D2 della tabella 2.1).

Jest

Jest ([14]) è un *framework*^{G} di test *JavaScript*. Può essere integrato in una vasta gamma di progetti che usano *TypeScript*, *Vue.js*, *Node.js* e molti altri. Anche *Stencil* utilizza *Jest* come *framework*^{G} di test ed è subito disponibile alla creazione di un progetto.

4.2 Strumenti di sviluppo e supporto

Git

Git ([13]) è un *software* di controllo versione. Rientra nella categoria dei sistemi di controllo versione distribuiti e ha quindi una migliore risoluzione dei conflitti e permette di impostare diversi tipi di flussi di lavoro.

Bitbucket

Bitbucket ([3]) è un servizio di *hosting* di *repository*^{G} basato su *Git*. È ottimizzato per i team che utilizzano *Jira*.

Visual Studio Code

Visual Studio Code (abbreviato *VsCode*) è un editor di codice sorgente sviluppato da Microsoft. È uno degli strumenti più versatili sul mercato per scrivere codice grazie soprattutto alle sue numerose estensioni. Proprio queste ultime permettono di trasformare quello che di base sarebbe un semplice editor di testo in un ambiente di sviluppo integrato (IDE) leggero e utilizzabile anche dagli sviluppatori meno esperti.

Draw.io

Draw.io è un *software* online per il disegno di grafici di diversi tipi, utilizzato anche in ambito di ingegneria del *software*. Per il progetto, *draw.io* è stato utile per tracciare graficamente i casi d'uso e gli schemi *Entity-Relationship (ER)* (usati per descrivere più facilmente le relazioni tra i modelli del database).

4.3 Strumenti organizzativi

Jira

Jira ([15]) è uno dei migliori strumenti Agile per la gestione dei progetti. Permette di pianificare, monitorare, rilasciare e supportare *software* di alta qualità e consente anche ai progetti più semplici di essere condotti in maniera professionale.

Microsoft Teams

Per quanto riguarda invece la comunicazione e l'organizzazione delle riunioni, soprattutto in caso di *smart working*, lo strumento utilizzato dall'azienda è *Microsoft Teams*. Tale strumento consente di mantenere tutti gli aspetti organizzativi e di comunicazione (come calendari, chiamate, chat e molti altri) in un unico luogo, rendendo così più facile e immediata l'interazione tra colleghi e con i clienti.

Capitolo 5

Backend: Directus

Figura 5.1: Logo *Directus*

La scelta dell'utilizzo di *Directus* ([7]) deriva puramente da una questione di tempistica. Il dover sviluppare entrambe le parti dell'applicazione da zero avrebbe richiesto molto più tempo di quello che era disponibile per lo stage. Proprio per questo motivo, uno dei requisiti di vincolo è quello di sviluppare il *backend*^{G} dell'applicazione intorno a questo potente strumento, riuscendo quindi a sviluppare più funzionalità per la parte *frontend*^{G}.

5.1 CMS Headless

Un *Content Management System (CMS) Headless* è una piattaforma per creare, modificare, eliminare e gestire delle risorse, tenendo però separata questa parte dalla presentazione dei contenuti. Tra le diverse funzionalità che *Directus* offre, una delle più importanti è proprio la creazione automatica di *endpoint* API^G in modo da accedere facilmente a tutti i contenuti da qualunque applicazione. L'obiettivo principale di Storystorm era creare e fornire un servizio che potesse essere integrato in modo facile e veloce all'interno di una qualsiasi applicazione web. Grazie a *Directus*, infatti, è stato possibile separare la creazione dei contenuti dalla loro presentazione e quindi permettere agli utenti di creare le loro storie direttamente da un unico *backoffice* e poi integrare separatamente il *widget* all'interno della loro applicazione.

5.2 Installazione

Come per ogni nuova tecnologia, prima di potersi cimentare nella sua implementazione è necessario studiarla e capire bene come poterla utilizzare al meglio. La documentazione di *Directus* ([9]) è ben fornita e permette a chiunque (sviluppatore o utente finale) di imparare a padroneggiare questo strumento nel minor tempo possibile.
Directus viene installato come *layer* aggiuntivo sopra ad un nuovo *database* oppure ad uno già esistente. L'applicazione e le API "riflettono" in tempo reale i contenuti della base di dati collegata, consentendo ai diversi utenti di interagire in modo semplice con essa, grazie anche ad un'interfaccia intuitiva ed immediata da utilizzare.
Per quanto riguarda l'installazione vera e propria, il servizio fornisce tre diversi metodi. Data la dimensione del progetto, al momento ridotta, *Directus* consente l'installazione direttamente da linea di comando, senza dover pagare per usufruire del servizio (in caso il fatturato annuo superi i 5 milioni di dollari, allora è necessario passare ad un piano a pagamento). Una volta soddisfatti il requisito, cioè aver installato nel proprio computer la versione 18 di *Node.js* (precisamente la versione 18.17 o superiore), è possibile procedere con l'installazione tramite npm e proseguire con la configurazione delle variabili d'ambiente.
Terminata la configurazione del progetto è possibile fin da subito iniziare a creare e gestire il *database* tramite *Directus*.
L'applicazione configurata e pronta all'uso è disponibile a:

`https://storystorm-bo-demo.hnrg.it/admin/login`

5.3 Data model

L'applicazione di *Directus* permette la creazione e la gestione di un *database* SQL senza dover effettivamente scrivere linee di codice o *query* SQL. Sono presenti tutte le funzionalità e i concetti di base di un *database* relazionale: tabelle e relazioni tra loro, oggetti, campi e molto altro. Il tutto contenuto in un unico posto e facile da utilizzare.
I *data model* si suddividono in due categorie: quelli personalizzati, creati dall'utente che utilizza il servizio, e quelli predefiniti. Questi ultimi sono delle tabelle che vengono create all'interno del *database* all'installazione di *Directus* e permettono al servizio di operare correttamente con tutte le sue funzionalità.
Durante la fase di analisi, oltre ai requisiti, è stato utile costruire uno schema ER dei vari *data model*, rappresentando graficamente tabelle e relazioni. La figura 5.2, per una questione di spazio e visibilità, raffigura solo le entità con le loro chiavi univoche e le relazioni tra di loro.

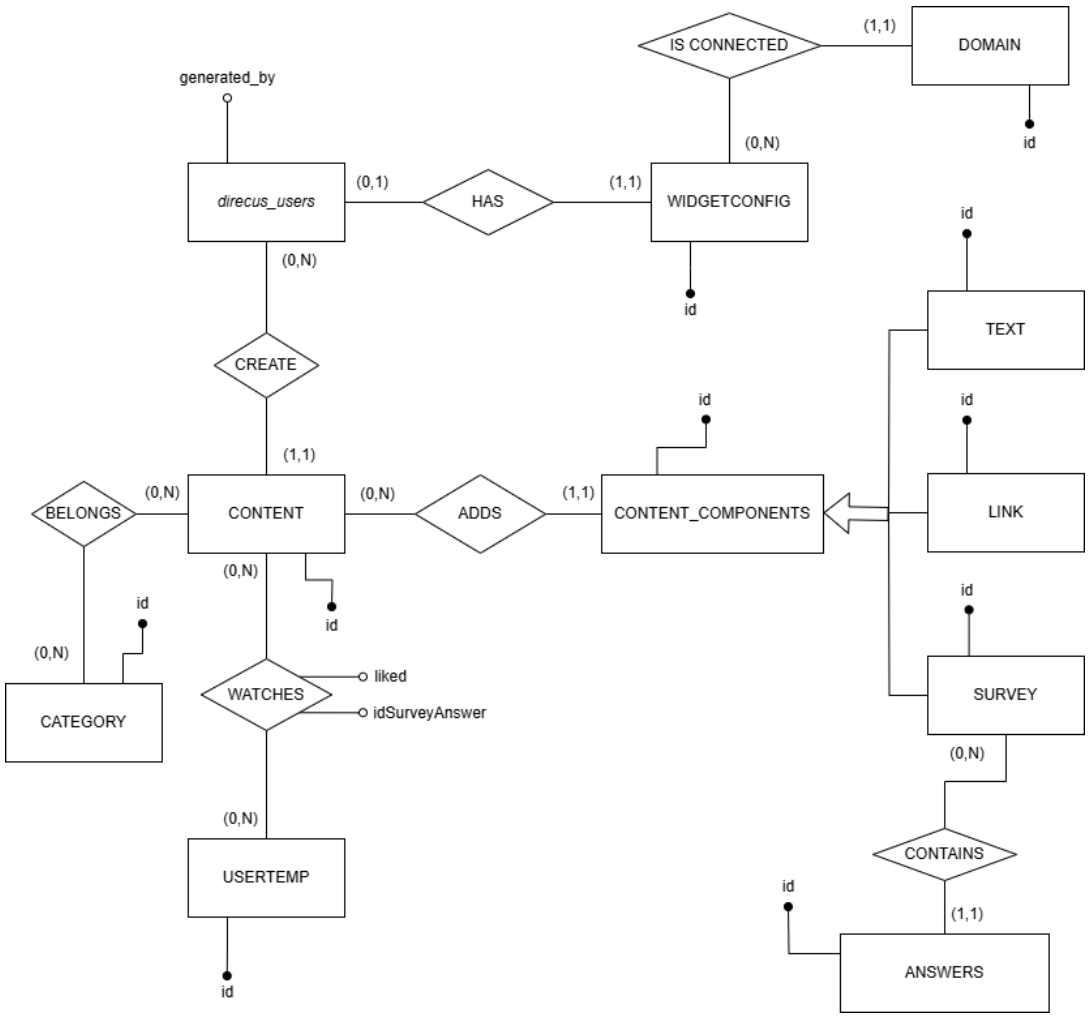


Figura 5.2: Schema ER del *database*

5.3.1 Data model personalizzati

Per Storystorm sono stati individuati i seguenti *data model* (ogni oggetto è identificato univocamente all'interno della tabella tramite un id):

- ***contents***, ossia il contenuto principale del *widget*: la storia social. Ogni oggetto è composto da un nome, dal contenuto multimediale, dal numero di *like* che ha ricevuto la storia, dallo stato della stessa e dalle informazione sulla creazione;

- ***categories***, l'insieme delle categorie dove verranno raggruppate le storie quando verranno pubblicate. Anche queste composte da un nome e da un'immagine di riferimento;

- ***widgetconfig***, si tratta dei vari aspetti di personalizzazione del *widget*. Quando un amministratore crea un'istanza di utente cliente deve inizializzare un'i-

stanza associata di *widgetconfig* in modo che poi possa essere modificata direttamente dal nuovo utente;

- ***domains***, ossia l'elenco dei domini consentiti. Questa funzionalità è stata aggiunta per evitare che qualcuno in possesso del *token*^{G} di un altro utente possa integrare il *widget* anche in un sito non autorizzato;

- ***usertemp***, cioè la lista degli utenti non autenticati che interagiscono con le storie nelle varie applicazioni. Questo è stato aggiunto per garantire l'affidabilità dei risultati (come visualizzazioni, *mi piace*, risposte ai sondaggi e altro).

Oltre a questi modelli, essenziali per il funzionamento dell'applicazione, sono stati aggiunti, come da requisito, tre *data model* per poter inserire i rispettivi componenti all'interno di una storia:

- ***text***, per i componenti di tipo testo;

- ***link***, per inserire dei link su cui l'utente può cliccare;

- ***survey***, per inserire dei sondaggi a cui gli utenti che li visualizzano possono rispondere. Per facilitare il conteggio e la tenuta delle risposte, ad ogni sondaggio si possono aggiungere fino a 4 risposte, che verranno inserite nella tabella ***answers***.

La scelta della struttura di queste tabelle è stata fatta consapevolmente e in modo che in futuro risulti facile e immediata l'aggiunta di nuovi componenti.
Grazie a *Directus* non è necessario configurare le relazioni tra le tabelle. Per quanto riguarda le relazioni 1:1 (uno a uno) e 1:N (uno a molti), *Directus* crea automaticamente nella tabella corretta un campo dedicato alla chiave esterna. Per le relazioni N:N (molti a molti) e le gerarchie, viene creata, sempre in modo automatico, una tabella aggiuntiva per permettere la corretta configurazione del *database* relazionale. In questo caso, i modelli creati da *Directus* sono: ***contents_components***, ***categories_contents*** e ***usertemp_contents***.

5.3.2 Data model predefiniti

I *data model* predefiniti sono quelli che permettono alle funzionalità di *Directus* (come l'autenticazione e l'accesso alla piattaforma) di operare correttamente. Quelli che vengono utilizzati principalmente da Storystorm per aggiungere funzionalità personalizzate all'applicazione sono:

- ***directus_collections***, cioè la lista di tutti i *data model* (personalizzati e predefiniti) con le loro configurazioni;

- ***directus_extensions***, che contiene dati e riferimenti alle estensioni aggiunte al progetto;
- ***directus_fields***, ossia i campi delle tabelle;
- ***directus_files***, dove vengono immagazzinati i dati e i permessi dei *files* (come immagini e video) aggiunti direttamente tramite *Directus*;
- ***directus_flows***, che consentono l'elaborazione personalizzata dei dati in base a degli eventi e l'automatizzazione delle attività all'interno del *backoffice*;
- ***directus_relations***, per tenere traccia delle relazioni tra le tabelle;
- ***directus_roles***, ossia i ruoli dei vari utenti della piattaforma;
- ***directus_users***, cioè tutte le informazioni su amministratori, clienti e utenti autenticati (i tipi di utente individuati in fase di analisi dei requisiti). Oltre ai campi di *default*, è stato aggiunto un campo personalizzato (*generated_by*) in modo da tener traccia di chi ha creato un determinato utente.

5.4 API

Directus offre sia API[G] di tipo *REST* sia di tipo *GraphQL*. La documentazione riporta che non c'è alcuna differenza tra le funzionalità delle due e che la decisione finale può basarsi anche solo sulle preferenze dello sviluppatore. Per Storystorm è stata scelta un'API[G] di tipo *REST*, principalmente per l'immediato uso che ha permesso.
Il servizio mette a disposizione diversi *endpoint* per interagire con il *database* e tutti i *data model* creati. Per accedere a questi *endpoint* in Storystorm è sufficiente recarsi ai seguenti URL:

- https://storystorm-bo-demo.hnrg.it/:*directus_collection*, per accedere alle informazioni dei *data model* predefiniti di *Directus*;
- https://storystorm-bo-demo.hnrg.it/items/:*collection*, per accedere ai modelli personalizzati.

Aggiungendo a questi URL "/:*id*" è possibile accedere all'unico oggetto all'interno del modello selezionato che è identificato tramite l'id.
Per eseguire operazioni di tipo CRUD (*Create, Read, Update, Delete*) nel *database*, bisogna inviare una richiesta, ad uno dei vari *endpoint* che la piattaforma mette a disposizione, utilizzando uno dei seguenti metodi:

- ***GET***, per accedere alle informazioni e ottenere una lista degli oggetti richiesti;

- ***POST***, per creare l'oggetto passato come corpo della richiesta;
- ***PATCH***, per modificare un oggetto. Per farlo è necessario passare come corpo della richiesta un oggetto parziale contenente le informazioni che si vogliono modificare;
- ***DELETE***, per eliminare un oggetto.

In *JavaScript* si può utilizzare l'API[G] *fetch* per inviare queste richieste.

5.4.1 Global Parameters e Filter Rules

La maggior parte degli *endpoint* possono essere manipolati aggiungendo alla *query string* dell'URL diversi parametri. La documentazione elenca e descrive molto bene tutti i parametri globali ([11]) e tutti i filtri ([10]) che possono essere aggiunti per rendere la richiesta sempre più specifica.
Non tutti questi parametri e filtri vengono utilizzati dalla parte *frontend*[G] di Storystorm. Analizziamo quindi la chiamata API[G] più importante di tutta l'applicazione (in una forma ridotta e più semplice).

```
const response = await fetch(
    process.env.API_HOST +
    '/items/contents' +
    '?fields=*.*.*' +
    '&filter[_or][1][user_created][_eq]=' +
    this.token +
    '&filter[_or][2][user_created][generated_by][id][_eq]=' +
    this.token +
    '&filter[user_created][status]=active'+
    '&filter[user_created][generated_by][status]=active'+
    '&filter[status][_eq]=published'+
    '&filter[publish_date][_gt]=' +
    this.getYesterday() +
    '&sort[]=publish_date',
);
```

Codice 5.1: Esempio di chiamata API all'interno di Storystorm

1. Viene chiamata l'API[G] *fetch* tramite l'omonimo metodo (in questo caso non viene passato un metodo specifico con cui eseguire la richiesta quindi viene usato quello di *default*, il metodo GET);

2. ***process.env.API_HOST*** va a richiamare la variabile d'ambiente definita come *API_HOST* nel file *.env*. In questo modo se il dominio del *backoffice* dell'applicazione dovesse cambiare, basterà sostituire una sola volta il valore di questa variabile;

3. Tramite ***/items/contents*** accediamo all'*endpoint* creato da *Directus* che ci permette di visualizzare la lista di tutti gli oggetti presenti all'interno del *data model* personalizzato *contents*;

4. Grazie al parametro ***fields*** possiamo scegliere quali campi effettivamente vengono restituiti dalla *query* (ha la stessa funzionalità che ha il *SELECT* in una *query* SQL). Nonostante nell'esempio venga utilizzato ***.*.***, che permette di ottenere tutti i campi della tabella e tutti quelli dei due livelli relazionali sottostanti, nell'applicazione vengono richiesti solo i campi necessari. Questo velocizza la richiesta e diminuisce la dimensione dell'*output*;

5. Con ***filter*** possiamo aggiungere delle condizioni alla *query* (come con il *WHERE* nelle *query* SQL). In questo caso le condizioni da rispettare sono le seguenti:

   - l'id dell'utente che ha creato il contenuto oppure l'id del suo creatore deve essere uguale al *token*[G] inserito nel *widget*. Questo in modo da garantire che vengano visualizzate le storie corrette;
   - lo stato dell'utente che ha creato la storia e quello del suo creatore deve essere uguale ad *active*. Questo permette agli utenti amministratori o agli utenti clienti di bloccare alcune utenze modificando il loro stato;
   - lo stato della storia deve essere uguale a *published*;
   - la data di pubblicazione della storia deve essere maggiore di quella restituita dalla funzione *getYesterday()* (che restituisce la data di oggi meno 24 ore).

6. Il parametro ***sort*** permette di ordinare in modo crescente gli oggetti restituiti secondo la loro data di pubblicazione;

7. Il risultato di questa chiamata API[G] viene assegnato alla costante *response* e potrà essere utilizzato all'interno dell'applicazione.

5.5 Ruoli e permessi

Per rispecchiare ciò che è emerso dall'analisi dei requisiti, possono essere creati all'interno del *backoffice* diversi ruoli da assegnare ai vari utenti. I ruoli in questione sono: Amministratore, Cliente, Utente autenticato e *Public* (che è già presente di

default in *Directus* e rappresenta gli Utenti non autenticati).
Oltre alle divisioni in ruoli, *Directus* permette di assegnare a ciascuno di questi un livello diverso di permessi. Per ogni *collection*, che sia predefinita o personalizzata, è possibile modificare i permessi sulla creazione, visualizzazione, modifica, eliminazione e condivisione degli oggetti garantendo un accesso completo, personalizzato oppure nessun accesso.

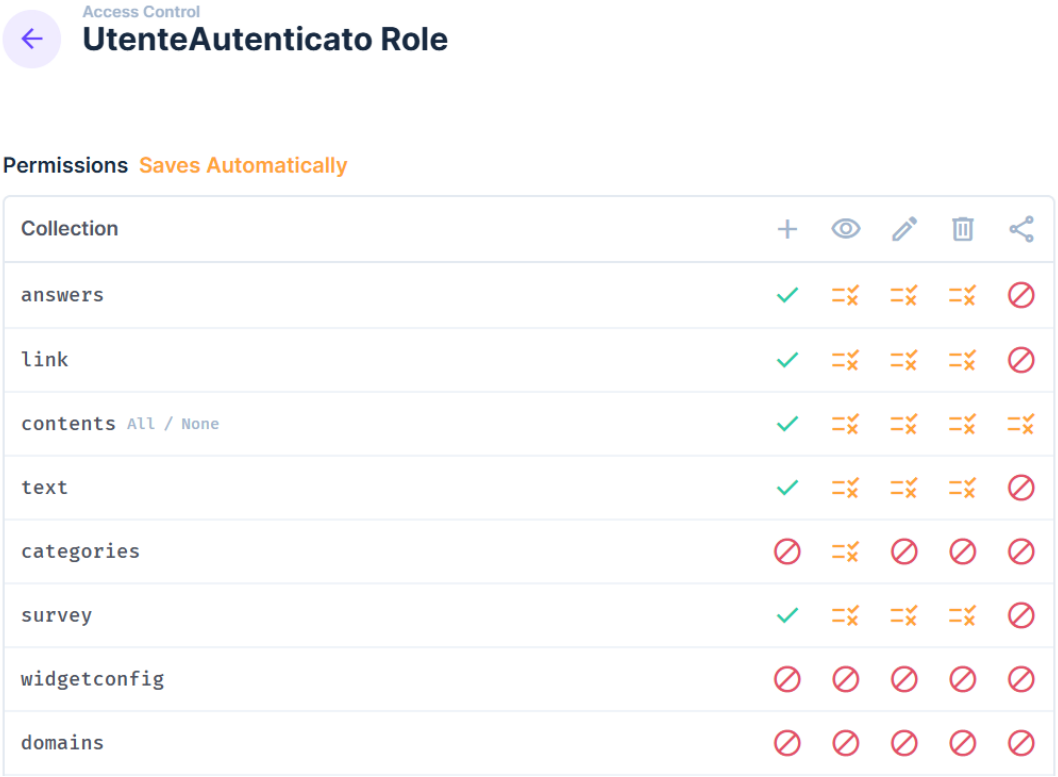


Figura 5.3: Permessi del ruolo di Utente autenticato in Storystorm

Come si evince dalla figura 5.3, gli utenti autenticati possono creare le storie ma hanno ad esempio alcune limitazioni sulla loro visualizzazione e possibilità di modifica ed eliminazione. Quello che invece non possono fare è cambiare la configurazione del *widget* e creare nuove categorie (compiti che spettano esclusivamente all'utente cliente).

5.6 Flows

I *flows* consentono l'elaborazione personalizzata dei dati in base a degli eventi e l'automatizzazione delle attività all'interno di *Directus*. Ognuno di questi "flussi" è composto da un *trigger* e una serie di operazioni scatenate da quest'ultimo.
In Storystorm sono stati implementati due *flows*:

- ***Archive Stories***, che ogni ora modifica le storie che sono state pubblicate più di 24 ore prima, aggiornando il loro stato in *archived*. In questo modo si ottiene una lista delle storie più corretta e modificando automaticamente lo stato di una storia una volta che non verrà più visualizzata nel *widget*;
- ***Generate Image***, che permette di implementare la generazione di contenuti multimediali direttamente all'interno del *backoffice*. Quando si visualizza un contenuto creato in precedenza, basterà aprire il menù sulla destra e cliccare su *"Generate Image"*. Inserendo la propria chiave API[G] di OpenAI e un *prompt*[G], il *flow* passa queste informazioni a DALL-E (algoritmo di IA[G] in grado di generare immagini a partire da un testo) e inserisce l'immagine come contenuto multimediale dell'oggetto che si sta visualizzando (il *setup* di questa funzionalità è ben descritto da uno dei tanti tutorial che *Directus* mette a disposizione ([8])).

5.7 Estensioni

Oltre a fornire già di default una vasta gamma di funzionalità, *Directus* offre la possibilità di creare ed installare delle estensioni migliorando e personalizzando l'esperienza d'uso dell'applicazione. Esistono 3 tipi di estensioni:

- quelle per l'applicazione, che vanno ad aggiungere o migliorare gli aspetti visivi (come interfacce, temi, moduli e molti altri);
- quelle per le API[G], utili a modificare o creare degli *endpoint* personalizzati;
- quelle ibride, che modificano sia le funzionalità *frontend*[G] sia quelle *backend*[G] di *Directus*.

In Storystorm sono state installate due estensioni. La prima viene utilizzata dal *flow Generate Image* ed è stata scaricata dal *marketplace*. La seconda invece è stata sviluppata da zero ed ha permesso di soddisfare alcuni requisiti.

5.7.1 Moduli personalizzati

I moduli personalizzati permettono di aggiungere un livello ulteriore di personalizzazione all'applicazione. Consentono infatti di creare nuove pagine da zero che verranno integrate direttamente nel menù di *Directus*.
Per soddisfare alcuni requisiti (in particolare RF6, RF10 e RF11 della tabella 3.1) e per fare in modo che le loro funzionalità venissero integrate direttamente all'interno del *backoffice*, è stato necessario creare ***Manage Stories***: un modulo personalizzato che mostra una lista delle storie visibili all'utente che ha effettuato l'acceso. Questa estensione, sviluppata con *Vue.js*, permette all'utente di:

- visualizzare una lista delle storie a cui ha accesso. Per ogni storia è specificato il suo stato, il nome, l'utente che l'ha creata, la data di creazione e quella di pubblicazione e il numero di *like*;
- ritornare alla pagina di modifica delle storie presenti nella lista;
- visualizzare una bozza per ogni storia presente nella lista;
- cambiare lo stato di ogni storia da bozza a pubblicata, da pubblicata ad archiviata e viceversa.

Il modulo si compone di due pagine: una per visualizzare la lista vera e propria delle storie e una che cambia dinamicamente in base a quale bozza si vuole visualizzare. La bozza di una storia non ha alcuna funzionalità e non si può interagire in alcun modo con essa.

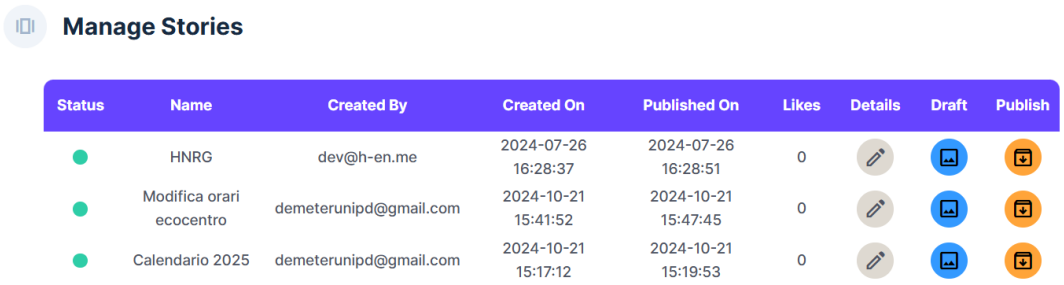


Figura 5.4: Lista delle storie nel modulo personalizzato *Manage Stories*

Capitolo 6

Frontend: Stencil

Figura 6.1: Logo *Stencil*

Stencil ([20]) è un compilatore che permette di creare componenti web personalizzati e versatili. Nasce proprio per fornire agli sviluppatori un modo per costruire dei componenti web riutilizzabili tramite TypeScript, JSX e CSS e convertirli nei *framework*^{G} più popolari (come *Angular*, *React* e *Vue.js*) senza dover conoscere in dettaglio questi ultimi.
In Storystorm viene utilizzato per sviluppare il *frontend*^{G} dell'applicazione in tutta la sua interezza.

6.1 Atomic Design

L'idea dell'*atomic design* ([1]) prende spunto della chimica. In questo campo infatti, gli elementi atomici si uniscono per formare le molecole, che a loro volta si combinano per formare organismi complessi.
Proprio da questo processo si sviluppa la metodologia dell'*atomic design* che si compone di 5 parti: tre che fanno riferimento al campo della chimica e due aggiuntive. Insieme lavorano per creare un sistema di progettazione di interfacce. Queste parti sono:

- **atomi**, ossia la base e la più piccola parte della materia che non può essere suddivisa in ulteriori parti funzionali. Nell'interfaccia gli atomi corrispondono a tutti quei componenti essenziali che costituiscono una pagina web. Possono essere bottoni, input, etichette dei form^{G} e molti altri che non sono suddivisibili ulteriormente senza perdere funzionalità. Come ogni atomo in natura ha le

sue proprietà uniche, anche questi componenti hanno le loro proprietà, come ad esempio la dimensione del carattere di un'intestazione;

- **molecole**, insiemi di atomi legati tra di loro. Nelle interfacce sono gruppi semplici di elementi che funzionano assieme (come un input, un'etichetta e un bottone possono formare un form[G] di ricerca). Quando uniti, gli atomi acquistano uno scopo (l'etichetta identifica il campo di input e il bottone invia il form[G]);
- **organismi**, componenti abbastanza complessi formati da insiemi di molecole, atomi o altri organismi;
- ***template***, cioè dei modelli privi di dati e composti da diversi organismi che definiscono la struttura di una pagina web;
- **pagine**, ossia istanze specifiche dei *template* a cui vengono applicati dei contenuti reali e adeguati a formare l'interfaccia finale della pagina web.

Tra i diversi vantaggi che offre questo approccio ci sono sicuramente la facilità di passare da un concetto astratto di pagina web ad uno più concreto e la chiara separazione tra struttura e contenuto.
Per costruire tutti i componenti che formano in Storystorm è stato usato proprio questo approccio, sviluppando i componenti uno ad uno e poi assemblandoli per ottenere il *widget* completo.

6.2 Installazione

Come per quella di *Directus*, anche la documentazione di *Stencil* ([21]) è completa e copre la maggior parte degli aspetti chiave dello sviluppo di componenti web.
Una volta soddisfatti i requisiti, ossia aver installato nel proprio computer una versione LTS recente di *Node.js*, si può procedere con l'installazione tramite npm. Completata la fase di configurazione, *Stencil* è pronto all'uso e si può immediatamente iniziare a sviluppare dei componenti web personalizzati.
I tre comandi essenziali per compilare i componenti e sfruttare al meglio *Stencil* sono:

- ***npm start***, che avvia un server di sviluppo locale che ci permette di osservare le modifiche dei componenti ad ogni salvataggio senza dover compilare ogni volta il codice;
- ***npm run build***, che crea una versione dei componenti pronta alla distribuzione e all'uso. I file generati da questo comando, se inseriti in un altro progetto, permettono di integrare facilmente il *widget* (soddisfacendo uno dei principali requisiti del progetto);

- ***npm test***, che esegue i test del progetto.

6.3 Decorators e struttura di un componente

Uno dei concetti e parte fondamentale di ogni componente sono i *decorator*. Un *decorator* è un'annotazione speciale in *TypeScript* utilizzata da *Stencil* per raccogliere tutti i metadati di un componente, i suoi attributi, i metodi che utilizza e molto altro. I *decorator* per sviluppare l'applicazione sono:

- ***@Component***, che dichiara un nuovo componente web;
- ***@Prop***, che dichiara un attributo che il componente può utilizzare;
- ***@State***, che dichiara uno stato interno del componente. Ogni qual volta che il valore di questa variabile cambia, il componente verrà aggiornato;
- ***@Watch***, che dichiara una serie di istruzioni che vengono eseguite quando una proprietà o uno stato cambia;
- ***@Element***, che dichiara un riferimento all'elemento *host* di un componente;
- ***@Method***, che dichiara un metodo pubblico che può essere richiamato anche da altri componenti.

Per comprendere al meglio come funziona e come si costruisce un componente web usando *Stencil*, prendiamo come esempio una versione ridotta del componente che viene creato all'installazione.

```
import { Component, Prop, h } from '@stencil/core';

@Component({
    tag: 'my-component',
    styleUrl: 'my-component.css',
})
export class MyComponent {
    @Prop() content: string;

    render() {
        return this.content;
    }
}
```

Codice 6.1: Componente web creato da *Stencil* all'installazione

1. Dopo tutti gli *import*, che permettono di usare le funzionalità di *Stencil*, la prima cosa da inserire è il *@Component decorator*. Questa parte del componente consente di configurarlo, scegliendo il nome ed indicando l'url del foglio di stile da applicare;

2. Successivamente c'è una dichiarazione standard di una classe in *JavaScript*. All'interno di questa classe verrà inserito tutto il codice al fine di far funzionare correttamente il componente;

3. Il componente è composto concretamente da un *@Prop decorator* che permette di dichiarare l'attributo *content* e dalla funzione *render()* che restituisce il codice JSX dando la possibilità di visualizzare a schermo il componente stesso.

L'utilizzo di questo componente risulterà nella creazione di un elemento *div* con all'interno ciò che andremo a specificare nell'attributo *content*.

6.4 Metodi del ciclo di vita dei componenti

Ogni componente ha diversi metodi del ciclo di vita che possono essere utilizzati per sapere ad esempio quando il componente viene caricato, oppure quando viene renderizzato. Questi metodi possono essere implementati direttamente all'interno della classe del componente, *Stencil* poi richiamerà automaticamente queste funzioni nell'ordine corretto.
Molti dei seguenti metodi vengono usati anche in Storystorm per eseguire alcune istruzioni in determinati punti del ciclo di vita di un componente:

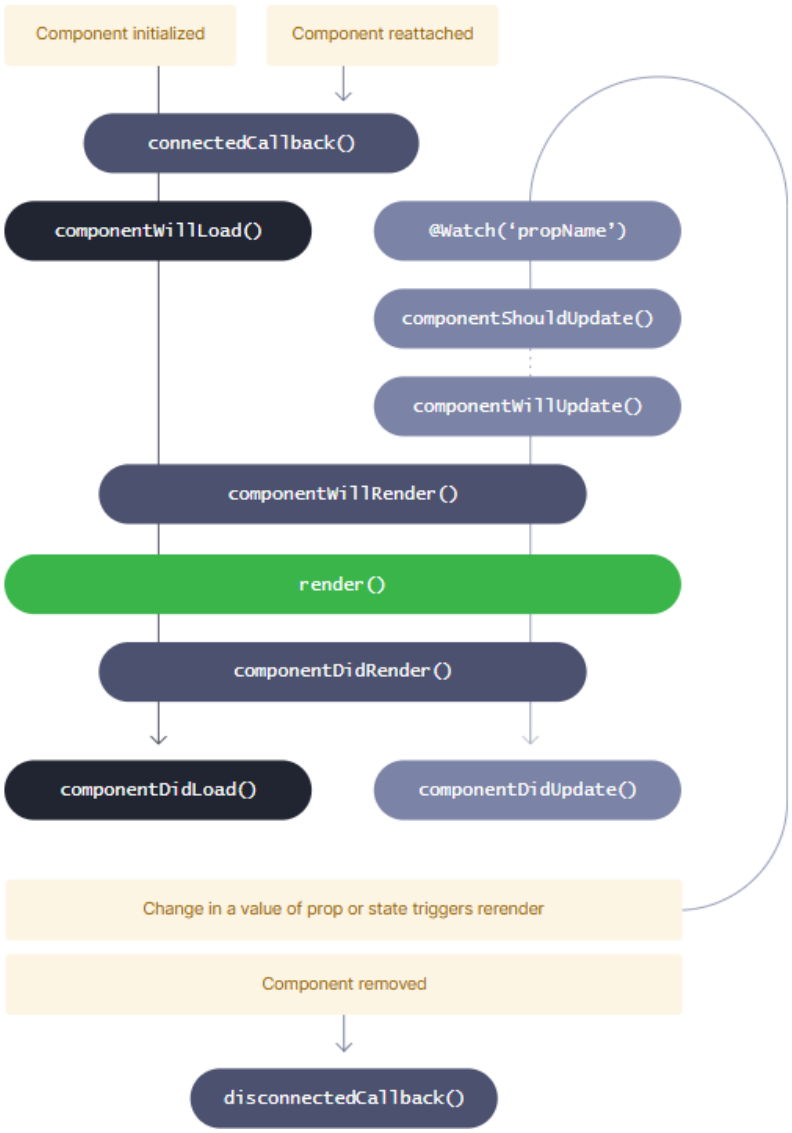


Figura 6.2: Metodi del ciclo di vita dei componenti

- ***connectedCallback()*** e ***disconnectedCallback()***, che permettono rispettivamente di eseguire istruzioni quando il componente viene connesso per la prima volta e quando viene disconnesso dal DOM[G]. Entrambi questi metodi possono essere richiamati più di una volta;
- ***componentWillLoad()*** e ***componentDidLoad()***, richiamati una sola volta, rispettivamente quando il componente viene connesso per la prima volta al DOM[G] e dopo che lo stesso è stato caricato nella sua interezza. Come suggerisce la documentazione, *componentWillLoad()* è un buon metodo dove inserire le chiamate API[G];
- ***componentShouldUpdate()***, ***componentWillUpdate()*** e ***componentDidUpdate()***, richiamati solo quando una proprietà *Prop* o *State*

viene modificata;

- ***componentWillRender()*** e ***componentDidRender()***, richiamati rispettivamente prima e dopo ogni *render* del componente. In Storystorm questi metodi vengono utilizzati per eseguire alcune istruzioni sia quando il componente viene creato sia ogni volta che lo stesso viene modificato;
- ***render()***, funzione indispensabile di ogni componente che permette di restituire ciò che dovrà essere mostrato a schermo.

6.5 Componenti web personalizzati

Per creare il *widget* di Storystorm, sono stati sviluppati 13 componenti web personalizzati. Per una migliore comprensione, sono stati raggruppati in 3 gruppi significativi:

- i componenti *Box*, ossia quelli che consentono di visualizzare il *widget* da chiuso;
- i componenti *View*, che permettono la visualizzazione delle singole storie;
- i componenti *Components*, ossia i componenti essenziali che ogni storia deve avere e quelli aggiuntivi individuati in fase di analisi dei requisiti.

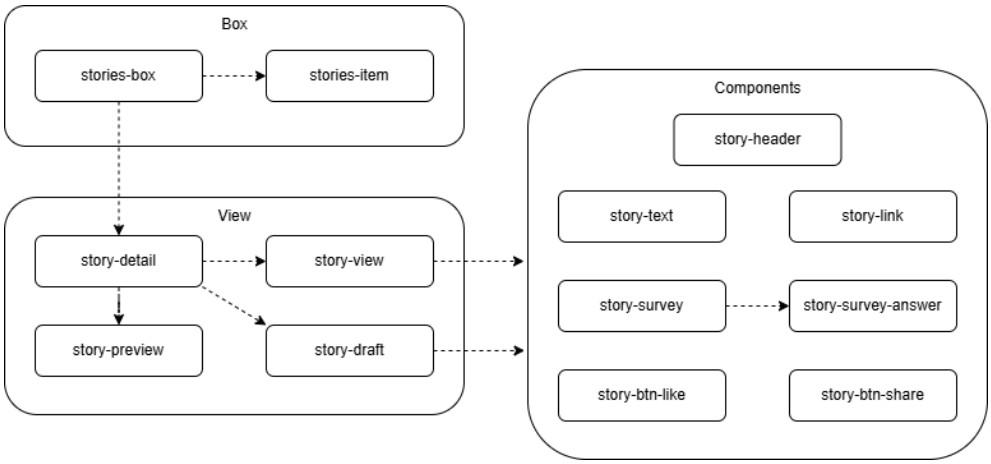


Figura 6.3: Schema dei componenti web personalizzati

6.5.1 Box

Figura 6.4: Insieme dei componenti della categoria *Box*

stories-box è il componente principale di tutta l'applicazione. Ha il compito di costruire il *widget* sia da chiuso sia quando si naviga attraverso le storie. Tramite un *token*[G] (passato come attributo al componente), esegue le chiamate API[G] necessarie per recuperare la lista delle storie da visualizzare dal *database*. Passa poi tutte le informazioni recuperate ai componenti sottostanti in modo da visualizzare correttamente tutte le storie recuperate.

stories-item ha il compito di costruire un pulsante con il quale accedere alla visualizzazione di una specifica categoria di storie. Viene creato tante volte quante sono le categorie con delle storie all'interno e mostra un cerchio (o un rettangolo, dipende dalla configurazione del *widget*) con l'immagine della categoria e il nome della stessa. Nel dettaglio, quando l'utente preme sulla categoria che vuole visualizzare, il componente cambia l'url in quello che corrisponde alla storia e richiama la funzione in *stories-box* che permette di renderizzare nuovamente il componente.

6.5.2 View

Figura 6.5: Insieme dei componenti della categoria *View*

story-detail è l'altro componente creato direttamente da *stories-box*. Se l'url della pagina in cui è integrato il *widget* contiene i parametri *storycategory* e *storyid*, allora verrà visualizzato questo componente. Si tratta semplicemente di una schermata di navigazione, composta da un logo, un pulsante per chiuderla e poi la vera e propria navigazione tra le storie (composta da *story-preview* per la categoria precedente e quella successiva a quella visualizzata e *story-view* per la visualizzazione delle storie nella categoria corrente).

story-view permette la visualizzazione di una storia nella propria categoria. È il componente che mette insieme effettivamente tutti gli elementi di una storia social (*header*, pulsanti *mi piace* e condividi, componenti aggiuntivi...). Ogni volta che viene creato o aggiornato, esegue due chiamate API[G]: una per recuperare tutte le informazioni riguardanti la storia che si vuole visualizzare e una per recuperare le informazioni sulle eventuali interazioni precedenti dell'utente con la stessa. Mette insieme queste informazioni e crea la vera e propria schermata di visualizzazione della storia (passando ai vari componenti sottostanti i dati necessari alla loro corretta inizializzazione).

story-preview aggiunge ai lati di *story-view* due piccole anteprime della categoria precedente e di quella successiva (se presenti). Tramite un semplice click è possibile passare direttamente alla categoria selezionata (senza dover navigare attraverso tutte le storie della categoria che si sta visualizzando). Questo componente è visibile solamente da *desktop* dato che da *mobile* non ci sarebbe spazio sufficiente per garantire una visualizzazione corretta.

story-draft è un componente che viene inizializzato solo se nell'url è presente esclusivamente il parametro *storyfullid*. Ha il compito di creare un'anteprima della storia identificata dall'id nella *query string*, togliendo però le funzionalità come il *mi piace* o la risposta ad un sondaggio. Questo componente viene utilizzato dal modulo personalizzato *Manage Stories* per permette la visualizzazione di un'anteprima delle storie direttamente dal *backoffice*.

6.5.3 Components

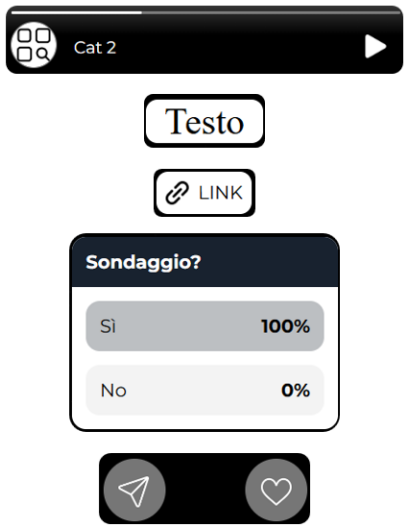


Figura 6.6: Insieme dei componenti della categoria *Components*

story-header ha il compito di creare l'intestazione della storia visualizzata. Comprende una barra di progressione della storia, l'immagine e il nome della categoria, i pulsanti per mettere in pausa e riprodurre la storia. In caso il contenuto sia un video, aggiunge anche un pulsante per mutare e riattivare il volume del video stesso. Se il *widget* viene visualizzato da un dispositivo mobile, *story-header* comprende anche un pulsante per chiudere la schermata di visualizzazione.

story-text utilizza le informazioni ricavate da *story-view* e crea un componente aggiuntivo di tipo testo (qualora sia stato creato dal *backoffice*). Questo testo verrà visualizzato nella storia secondo le configurazioni scelte.

story-link è molto simile a *story-text*. Date le informazioni ottenute dalle chiamate API[G], esso crea un pulsante personalizzato che funge da link. Premuto il pulsante, la riproduzione della storia viene messa in pausa e si aprirà un piccolo *popup* per confermare la scelta.

story-survey permette l'aggiunta di un sondaggio all'interno delle storie. Ogni sondaggio può avere al massimo 4 risposte. Una volta che l'utente avrà votato, il componente si aggiorna mostrando i risultati del sondaggio.

story-survey-answer restituisce semplicemente un bottone cliccabile contenente una delle risposte di un sondaggio. Al voto dell'utente, il componente ha il compito di registrare il voto all'interno del *database*.

story-btn-like aggiunge un bottone con il quale è possibile mettere o togliere il *mi*

piace alla storia visualizzata. Il risultato dell'operazione viene registrato nel *database*.

story-btn-share permette di condividere la storia. Se il metodo *navigator.share()* è supportato allora il pulsante apre la schermata di condivisione del sistema operativo, altrimenti copia il link della storia che si sta visualizzando.

6.6 Funzionamento, problemi e soluzioni

Come per molti progetti è impossibile prevedere tutto quello che bisognerà fare per ottenere un prodotto utilizzabile e funzionante. Anche per Storystorm non è stato tutto pianificato fin dall'inizio.

6.6.1 Navigazione tra le storie

La prima cosa da affrontare è stata l'implementazione della navigazione tra le storie. Dopo molti tentativi con diversi approcci, si è arrivati ad una buona soluzione. Dato che il *widget* deve funzionare in qualsiasi sito web e poter essere integrato in modo facile e veloce, è stato impossibile non implementare la navigazione tra le storie utilizzando i parametri nella *query string*. I tre parametri che vengono utilizzati dal *widget*, e che quindi non possono essere usati in alcun modo da un sito che vuole integrarlo, sono:

- *storycategory* che tiene traccia della categoria della storia che si sta visualizzando;
- *storyid* che tiene traccia di quale storia all'interno della categoria si sta visualizzando;
- *storyfullid* che, se presente, permette al *widget* di creare un'anteprima della storia.

Una volta entrati in un sito con il *widget* integrato, esso verrà visualizzato da chiuso (come in figura 6.4), recuperando dal *database* solo le informazioni necessarie. Per accedere ad una storia basterà cliccare nella categoria che si vuole visualizzare. Il *widget* aggiorna l'url della pagina, aggiungendo i parametri necessari alla corretta visualizzazione della storia selezionata e viene renderizzato nuovamente. Data la presenza dei parametri, il componente *stories-box* sa che deve visualizzare la storia e quindi crea il componente *story-detail* (e a cascata tutti gli altri). Per recuperare le informazioni necessarie alla visualizzazione della storia viene effettuata una chiamata API[G] prima che avvenga il *render* del componente *story-view*.
Chiudendo la schermata di visualizzazione delle storie, l'url viene ripristinato a quello

originale e il *widget* viene nuovamente aggiornato, ritornando allo stato iniziale.
Questo approccio ha permesso inoltre di implementare senza problemi la funzionalità di condivisione di una storia, visto che per accedere nuovamente ad essa basta recarsi al link con i parametri corretti.

6.6.2 Percorso degli asset

Gli *asset* hanno un determinato percorso se si utilizza il server di sviluppo di *Stencil* mentre ne hanno uno differente se si integra il *widget* in un altro sito. Per fare in modo che i componenti ottenessero il corretto percorso delle immagini e dei video, è stato sufficiente utilizzare la funzione *getAssetPath()*. Seguendo poi la documentazione è stato possibile copiare tutti i file nella cartella di destinazione.

6.6.3 Affidabilità dei dati delle interazioni

Il problema più grande che si è dovuto affrontare è stato quello di garantire che le interazioni degli utenti venissero registrate dal sistema in modo corretto e affidabile. Siccome il *widget* può essere integrato ovunque, chi accede a questi siti non è sempre autenticato (e in ogni caso è impossibile accedere alle utenze dei siti altrui). Quindi all'inizio bastava ricaricare la pagina per poter mettere nuovamente *mi piace* ad una storia oppure rispondere di nuovo ad un sondaggio. Questo non è un problema da poco. Basti pensare ai social e a quanto le aziende si affidano a questi numeri per capire su chi spendere le loro risorse in pubblicità e sponsorizzazioni.
La prima soluzione provata è stata conservare le informazioni sulle interazioni di un utente utilizzando l'oggetto *localStorage*, che permette di salvare coppie di chiave/valore all'interno del *browser*. Seppur questa soluzione era apparentemente valida, un utente con un minimo di conoscenza in materia sa perfettamente che è possibile cancellare le informazioni contenute in *localStorage* con pochi semplici click, ritornando così al problema di partenza.
Come soluzione finale si è deciso di implementare all'interno di Storystorm una libreria che consente, attraverso diversi parametri, di identificare un utente in base al *browser* dal quale sta visitando il sito. La libreria in questione è *FingerprintJS* ([12]) e ha una precisione tra il 40% e il 60%. Dopo alcuni test si è concluso che è più che sufficiente per il progetto.
All'inizializzazione del *widget*, il componente *stories-box* richiama la funzione per generare questa impronta digitale. Se non è già presente nella tabella *usertemp* del *database* allora viene aggiunta, altrimenti prosegue con la creazione. In seguito, sempre *stories-box* esegue due chiamate API[G]: una per recuperare tutti i dati delle storie disponibili in quel momento e una per recuperare le informazioni sulle storie con cui l'utente identificato da *FingerprintJS* ha già interagito. Dopo un controllo incrociato, modificando i dati delle storie se sono presenti passate interazioni, vie-

ne creata una lista delle storie divise per categorie e viene visualizzata tramite il componente *stories-item*.

6.7 Integrazione dei framework

L'obiettivo principale di *Stencil* è rimuovere il bisogno di scrivere componenti usando le API[G] di un *framework*[G] specifico. Usando *Stencil* è possibile costruire i componenti una sola volta e lasciare che sia proprio questo strumento a generare le librerie da utilizzare in progetti in *React, Angular* oppure *Vue.js*.
Per raggiungere uno degli obiettivi desiderabili (precisamente il D2 della tabella 2.1) è stato sufficiente seguire i passaggi descritti dalla documentazione per integrare un componente *Stencil* in *Vue.js* ([22]), ottenendo quindi un'applicazione demo contenente il *widget* di Storystorm.

Capitolo 7

Verifica e validazione

Una parte essenziale della vita di un *software* è la fase di test. Essa, infatti, permette di individuare errori e altri difetti e verificare se il prodotto è pronto all'uso.

7.1 Accessibilità

Nonostante non fosse un requisito del progetto, anche l'accessibilità del componente ha un ruolo importante sulla sua usabilità.
Come si può immaginare, è molto complicato rendere accessibile a tutte le categorie di utenti un qualcosa che si basa esclusivamente su contenuti multimediali. Perciò è stato ritenuto opportuno che le immagini che venivano utilizzate all'interno del *widget* avessero almeno un'alternativa testuale che i lettori di schermo potessero interpretare. Tutte le immagini e i video caricati avranno quindi come alternativa testuale il nome che è stato assegnato alla storia in fase di creazione.
Il creatore della storia ha a disposizione tutti gli strumenti per rendere questi contenuti accessibili, ma la scelta finale spetta a lui. Se ad esempio, viene inserito un nome non significativo, per alcune categorie di utenti non sarà possibile usufruire a pieno del servizio. Anche per questo motivo l'accessibilità non era presente tra i requisiti del progetto.
Sebbene tutto quello che poteva essere adottato per rendere il *widget* più accessibile sia stato fatto (come, ad esempio, l'utilizzo dei tag corretti), l'effettiva usabilità da parte di tutti gli utenti dipende dall'ambiente in cui viene integrato. Storystorm in sé per sé è accessibile, ma se viene integrato in un sito che non lo è, tutte le soluzioni adottate per renderlo tale vengono vanificate. Come afferma infatti *AccessiWay*, il 98% dei siti web non è accessibile ([19]).

7.2 Test

I test di unità sono stati scritti utilizzando il *framework*[G] *Jest*. Tale strumento viene installato direttamente alla creazione di un progetto in *Stencil* e può essere usato fin da subito. Questo tipo di test è stato implementato per la maggior parte dei componenti personalizzati controllando che, data una serie di informazioni, venisse restituito il componente configurato in modo corretto.
Secondo le *best practice* per testare i componenti web ([2]), gli aspetti principali da controllare quando si scrivono i casi di test sono:

- la logica interna dei componenti. Si tratta di verificare che i metodi implementati all'interno del componente funzionino in maniera corretta e restituiscano il risultato atteso. Questo tipo di controllo è stato effettuato sulla funzione *getYesterday()* del componente *stories-box*. Il metodo consente al componente di recuperare, nel formato corretto, la data corrente meno 24 ore. Per non dover ripubblicare le storie ogni giorno però, questa funzione è stata modificata nella fase di sviluppo, restituendo la data corrente meno 30 giorni. Con tale modifica non è necessario pubblicare sempre nuove storie, non motivando così l'utente finale a visitare la piattaforma ogni giorno per visualizzare gli aggiornamenti. Il test in questo caso consente di verificare che la funzione venga ripristinata con i valori corretti prima che l'applicazione sia rilasciata al pubblico;

- gli attributi, per i quali bisogna assicurarsi che interagiscano con il componente come previsto. In Storystorm, tutti i componenti, tranne *stories-box*, hanno uno stile che viene passato come attributo allo stesso. Soprattutto per i componenti aggiuntivi delle storie, come testi, link o sondaggi, è fondamentale controllare che lo stile venga applicato in modo corretto, così da rispettare il risultato finale che l'utente creatore della storia voleva ottenere. Adottando la strategia di *Test-Driven Development (TDD)*[G], questi test di unità consentono agli sviluppatori di evitare errori nella modifica o nella creazione di nuovi componenti aggiuntivi;

- il *rendering* del componente. Grazie a questi test viene verificato che i componenti restituiscano gli elementi corretti in base ai loro attributi e al loro stato in modo da visualizzare a schermo le giuste informazioni. Vengono implementati nella maggior parte dei componenti in Storystorm, soprattutto per verificare che gli stili e lo stato dei componenti vengano inizializzati e modificati correttamente;

- la gestione degli eventi, che permette di simulare le diverse interazioni che l'utente può effettuare con il sistema, come l'inserimento di dati nei campi di

un form[G]. Al momento Storystorm consente di interagire con le storie solo tramite la premuta di vari pulsanti. I test implementati controllano che al loro click venga richiamata la funzione corretta;

- il comportamento delle chiamate API[G]. Storystorm fa molto affidamento su queste chiamate. Il non funzionamento anche di una solo, andrebbe a compromettere gravemente l'esperienza d'uso dell'utente. Per evitare che ciò accada, sono stati implementati diversi test di unità che controllano che il comportamento delle chiamate API[G] sia sempre quello previsto. Siccome andare ad interagire concretamente con il *database* ridurrebbe di molto la velocità e l'affidabilità, viene utilizzata la tecnica di *mocking*. Questa consiste nel creare dei dati che non esistono, simulando quello che restituirebbe effettivamente la chiamata API[G]. Ad esempio, nel componente *story-survey-answer* vengono testate:
  - le chiamate di tipo *GET*, per accedere alle informazioni relative alle passate interazioni dell'utente, al numero delle risposte totali del sondaggio e al numero delle risposte che ha ricevuto fino a quello momento la risposta del componente;
  - le chiamate di tipo *PATCH*, per modificare i dati del sondaggio dopo che l'utente ha interagito con esso. Quando l'utente vota, i test verificano che i voti totali e quelli della risposta vengano incrementati di uno e che venga anche aggiornata correttamente nel *database* l'avvenuta interazione da parte dell'utente stesso.

Per eseguire in maniera automatica questi test è stato sufficiente creare, all'interno di ogni componente, un file con estensione *.spec.ts* (oppure *.spec.tsx*) contenenti i casi di test e poi eseguire il seguente comando:

stencil test --spec

7.3 Accettazione e collaudo

Il tutor aziendale (e in generale tutti i colleghi) sono sempre stati molto disponibili. Ogni volta che c'era l'opportunità, ma in generale almeno una volta a settimana, ci si fermava il tempo necessario per revisionare l'avanzamento del progetto e capire cosa si potesse migliorare.
Durante l'ultima settimana di lavoro mi è stato richiesto di effettuare una presentazione del prodotto sviluppato, dimostrando il raggiungimento degli obiettivi e lo sviluppo delle funzionalità richieste dall'azienda.

Capitolo 8

Storystorm e Demeter

Per dimostrare il raggiungimento dell'obiettivo principale del progetto, Storystorm è stato integrato all'interno di un sito web (*Demeter*) e il *widget* è stato personalizzato e utilizzato da un utente esterno.

8.1 Cos'è Demeter

Figura 8.1: Logo *Demeter*

Demeter ([6]) è un progetto, sviluppato dal sottoscritto e altri tre compagni di università, oggetto di valutazione nel corso di Tecnologie Web. Dato che uno degli obiettivi del corso era quello di saper sviluppare un sito web accessibile, *Demeter* punta a rendere utilizzabile a tutte le categorie di utenti un sito per la raccolta differenziata.
Vincitore del Concorso Accattivante e Accessibile 2024 ([5]), *Demeter* è il perfetto tipo di sito web dove integrare un *widget* come quello di Storystorm. Il fatto di dover accedere al sito più volte per vedere gli aggiornamenti delle storie aiuta gli utenti a rimanere informati su tutte le notizie che riguardano il mondo della raccolta e del riciclaggio dei rifiuti, tema che al giorno d'oggi diventa sempre più importante.

8.2 Integrazione di Storystorm

8.2.1 Installazione

Per integrare il *widget* all'interno del sito web ci sono voluti pochi e semplici passaggi.

1. L'utente amministratore (in questo caso il sottoscritto) crea l'utenza richiesta dal cliente (*Demeter*) con le credenziali da lui fornite. Crea, inoltre, un'istanza di *widgetconfig* collegata al nuovo utente che contiene le configurazioni del *widget*;

2. L'amministratore conferma al cliente che la registrazione è avvenuta con successo, gli fornisce il *token*^{G} di integrazione e da questo momento in poi il cliente può utilizzare liberamente il *widget*;

3. Per poter poi integrare effettivamente il componente all'interno del sito web è bastato aggiungere ai *file* del sito la cartella *dist* generata dal comando *npm run build* e inserire le seguenti linee di codice all'interno della *homepage*:

```
<script type="module" src="dist/esm/stories-box.js"></script>
<script nomodule src="dist/stories-box/stories-box.esm.js">
```

Codice 8.1: Linee di *script* da aggiungere alla pagina del sito

```
<stories-box token=""></stories-box>
```

Codice 8.2: Tag HTML da aggiungere alla pagina del sito

Le due linee di *script* vanno a richiamare i *file* necessari all'integrazione dalla cartella *dist*. Il tag HTML invece, passando come attributo il *token*^{G} fornito dall'amministratore, va a creare effettivamente il componente *stories-box* all'interno della pagina.

In una decina di minuti questi tre passaggi era stati completati e il *widget* era pronto all'uso.

8.2.2 Personalizzazione del widget

Una volta effettuato l'accesso al *backoffice* tramite le credenziali fornite dall'utente amministratore, la prima cosa da fare è personalizzare il *widget* a proprio piacimento. Per una migliore esperienza e ottenere il massimo da Storystorm conviene impostare la maggior parte dei seguenti parametri:

- ***logo***, ossia il logo del sito dove verrà integrato il *widget*. Nella visualizzazione *desktop* il logo apparirà in alto a sinistra della schermata di visualizzazione delle storie;
- ***storiesStyle***, che permette di modificare lo stile del componente quando la schermata di visualizzazione è chiusa;
- ***theme***, che consente di attribuire al *widget* un tema chiaro oppure scuro a seconda del tema dell'applicazione nel quale viene inserito;
- ***circleColor*** permette di personalizzare il colore del cerchio esterno delle storie (quando non sono visualizzate). Si può aggiungere ***circleColorGradient2*** e ***circleColorGradient3*** per impostare un gradiente a questo elemento;
- ***storiesNoScroll***, che permette di impostare il numero massimo di *stories-item* visualizzabili senza eseguire uno scorrimento orizzontale;
- ***backgroundColor*** consente di aggiungere il colore di sfondo della porzione di sito dove il *widget* verrà inserito in modo da ottenere un effetto migliore;
- ***secondsPerStory*** per impostare la durata di una storia (a meno che non sia un video);
- ***domains***, ossia i domini in cui il *token*^{G} fornito in fase di registrazione, permette al *widget* di funzionare correttamente.

Per evitare conflitti, la modifica delle configurazioni del *widget* è consentita solo all'utente cliente.

8.2.3 Creazione e visualizzazione delle storie

La creazione delle storie è semplice e veloce. È sufficiente accedere alla sezione dedicata ai *data model* e creare un'istanza di *content*. Dall'interfaccia proposta da *Directus* sarà poi possibile creare anche le categorie e i componenti delle storie.
Per creare una storia è necessario inserire un nome (che servirà da alternativa testuale al contenuto), il contenuto multimediale (immagine o video) e la categoria in cui inserire la storia. È possibile poi inserire dei componenti aggiuntivi come testo, link o sondaggi.

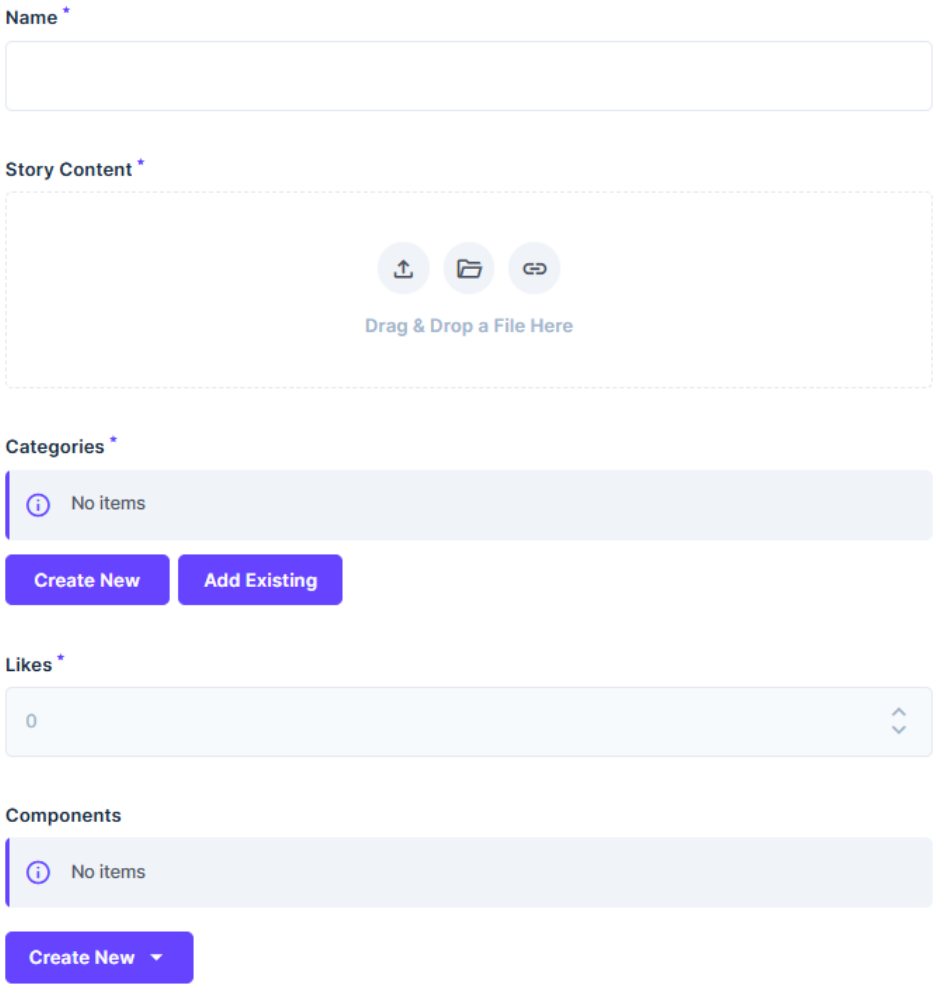


Figura 8.2: Interfaccia di creazione di una storia

Una volta creata una storia e pubblicata tramite il modulo personalizzato *Manage Stories*, essa sarà visibile all'interno del sito e ogni utente potrà visualizzarla e interagire con i suoi componenti.
Si viene a creare così un'istanza specifica del *template* del *widget* (che possiamo osservare in figura 6.4 e 6.5), andando a creare quella che nell'*Atomic Design* viene definita come *pagina*.

Figura 8.3: Visualizzazione del *widget* chiuso in *Demeter*

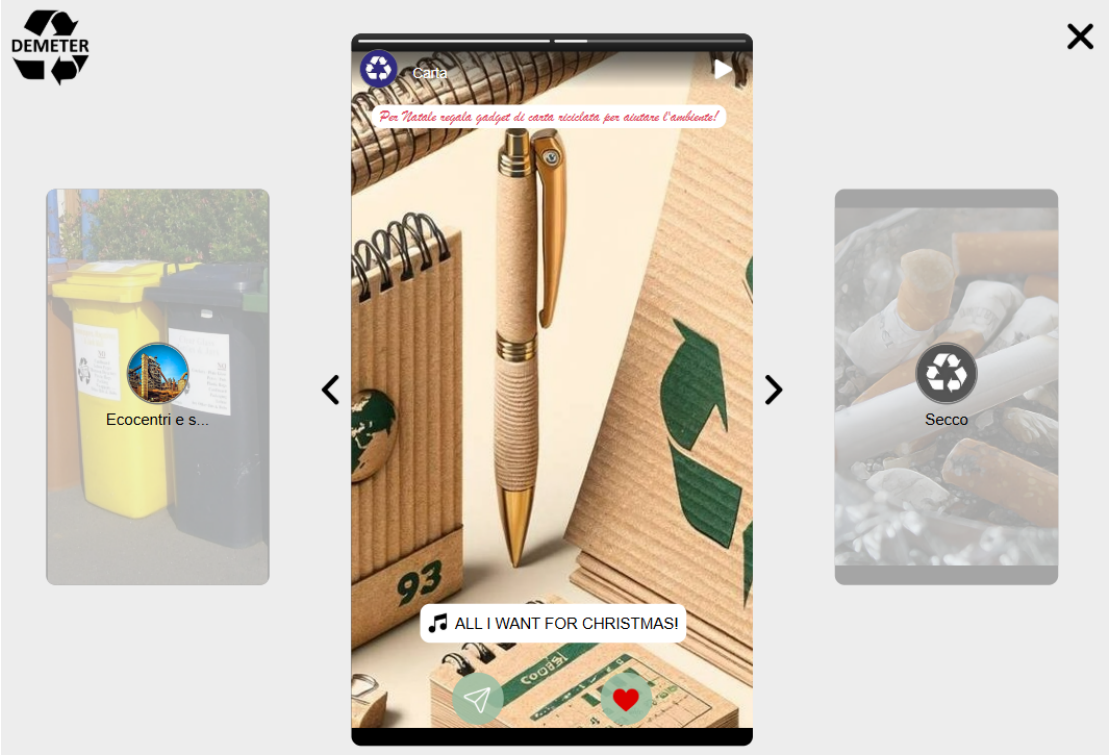


Figura 8.4: Visualizzazione delle storie in modalità *desktop* in *Demeter*

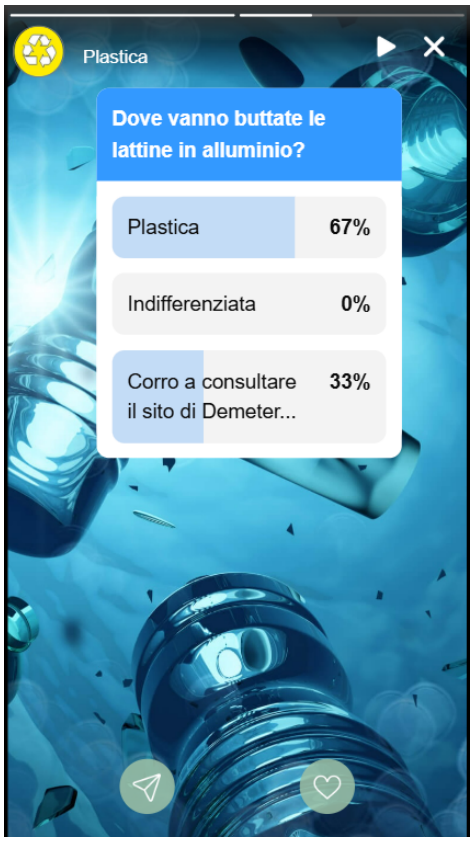


Figura 8.5: Visualizzazione delle storie in modalità *mobile* in *Demeter*

8.3 Esperienza d'uso

Per avere una valutazione obiettiva sull'esperienza d'uso di Storystorm, l'integrazione del *widget* in *Demeter* è stata effettuata da uno dei compagni di università che ha collaborato al progetto di Tecnologie Web (Gusatto Derek), dando poi un *feedback* su 4 aspetti essenziali dell'applicazione.

Integrazione nel sito web

- L'integrazione è stata fluida: in pochi passi si può aggiungere Storystorm al proprio sito senza alcuna difficoltà;
- Ottima l'integrazione visiva, per esempio la possibilità di avere il colore di sfondo (*padding* tra le categorie) uguale a quello della pagina, che aiuta a creare un'esperienza visiva coerente per gli utenti;
- Bene anche che la visualizzazione sia *responsive* e sensata sia da *mobile* sia da *desktop*;

Interfaccia di Storystorm

- L'interfaccia è semplice da utilizzare, anche per chi non ha competenze tecniche. In pochi minuti si riesce a creare e pubblicare la prima storia;
- Bene la differenziazione (di colore e icona) delle storie pubblicate, archiviate e da pubblicare. Sarebbe bello poter ordinare le storie in base a diversi criteri come la data di creazione o quella di pubblicazione;
- Difficoltoso vedere il numero di risposte ai sondaggi.

Funzionalità

- Ottima l'organizzazione delle storie in categorie personalizzabili;
- Bella la possibilità di inserire link differenziando il loro tipo (link semplice, musica o luogo). Limitata, ma comunque sufficiente e sensata, la scelta di colori per gli elementi;
- Il problema più grande è la complicata visualizzazione delle anteprime delle storie, è troppo macchinosa. L'ideale sarebbe averla in *real-time*.

Prestazioni

- Il caricamento del *tool* è rapido e non ha rallentato il sito, anche con molte immagini caricate;
- Alcune volte il caricamento della singola storia richiede qualche secondo in più, ma non al punto di rendere il sito poco *user-friendly*.

Capitolo 9

Conclusioni

9.1 Analisi del lavoro svolto

9.1.1 Consuntivo delle attività

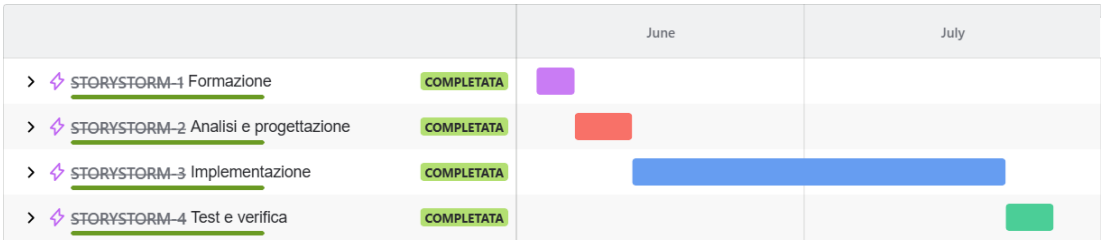


Figura 9.1: Diagramma di GANTT consuntivo

Il diagramma in figura 9.1 rappresenta graficamente la durata e le date di inizio e di fine delle fasi del progetto. Confrontandolo con il diagramma del preventivo (figura 2.3) sorgono alcune differenze rispetto al periodo delle varie fasi:

- la prima fase, quella di formazione, è durata meno del previsto. Conoscevo già molte delle tecnologie e degli strumenti necessari allo sviluppo del progetto mentre quelle più complesse avevano comunque alla base dei linguaggi in cui avevo già una certa dimestichezza;
- di conseguenza possiamo vedere come la fase di analisi e progettazione è potuta iniziare prima. Il tempo per tracciare i requisiti e progettare lo schema ER dell'applicazione è stato però in linea con quello del preventivo;
- la fase di implementazione è potuta iniziare in anticipo. Come si evince dal consuntivo però, questa ha richiesto circa una decina di giorni in più di quelli che erano stati preventivati. Le funzionalità da sviluppare erano davvero tante e per ottenere una versione base ed utilizzabile del prodotto è stato necessario allungare questa fase;

- l'ultima fase, quella di test e verifica, è durata qualche giorno in meno del previsto. Lo stage doveva necessariamente terminare venerdì 26 luglio. Nonostante il minor tempo a disposizione, anche questa fase è stata portata a termine senza difficoltà significative.

9.1.2 Raggiungimento degli obiettivi

Tutti gli obiettivi prefissati (elencati nella tabella 2.1) sono stati soddisfatti. Quelli obbligatori sono stati raggiunti in modo semplice. Nonostante alcune difficoltà con gli obiettivi desiderabili e quello opzionale, anche questi ultimi sono stati considerati soddisfatti nel resoconto finale con l'azienda.

9.2 Conoscenze acquisite

Lo sviluppo di Storystorm non ha richiesto numerose conoscenze pregresse (una solida base di *JavaScript* è stata sufficiente). Avendo sempre realizzato progetti con HTML, CSS, *JavaScript* e *php*, questo tirocinio mi ha dato l'opportunità di approfondire alcuni argomenti che non sono stati trattati nel mio percorso di studi. Durante lo stage ho scoperto i CMS *Headless* come *Directus*, che sono dei potenti strumenti per gestire i *database* anche da chi non ha alcuna conoscenza in materia. Questa esperienza mi ha permesso di entrare in confidenza con diversi *framework*^{G} di *JavaScript*, come *Vue.js*, e imparare a sviluppare componenti web personalizzati (abilità molto richiesta nel mondo del lavoro).

9.3 Autovalutazione sul prodotto

Sono consapevole che l'applicazione sviluppata è migliorabile. Il *feedback* ricevuto dal mio collega universitario non ha fatto altro che confermare alcune conclusioni a cui ero già arrivato.
Storystorm è integrabile ed utilizzabile facilmente anche da un utente meno esperto. Lo strumento permette un'ottima personalizzazione del *widget* e offre tutte le funzionalità base delle storie social. Il processo di creazione di una storia è però macchinoso e il tempo di caricamento delle storie da visualizzare è ancora troppo lento, nonostante le diverse migliorie effettuate.
In generale sono molto soddisfatto del prodotto che sono riuscito a sviluppare. Se ai vari *social network* ci sono voluti anni per perfezionare tutte le funzionalità, essere riuscito a sviluppare un'applicazione che funziona e ha raggiunto gli obiettivi richiesti mi rende molto orgoglioso.

9.4 Prospettive e sviluppi futuri dell'applicazione

Dal resoconto finale è emerso che l'azienda ha l'intenzione di proporre Storystorm ad alcuni dei loro clienti e che sta valutando come proseguire lo sviluppo. Che abbia o meno la possibilità di continuare a lavorare sull'applicazione assieme ad HNRG, vorrei migliorare l'esperienza d'uso di Storystorm attraverso:

- il precaricamento delle storie successive e precedenti a quella che si sta visualizzando, in modo da velocizzare la navigazione tra di esse;
- la visualizzazione della bozza di una storia in tempo reale. L'utente così è in grado di vedere immediatamente i componenti che aggiunge e le modifiche che apporta, permettendo un utilizzo più semplice e intuitivo;
- l'aggiunta di altri componenti (conti alla rovescia, risposte alle domande...) e il miglioramento di quelli già esistenti, consentendo all'utente finale di esprimere al meglio la sua creatività nella creazione delle storie.

9.5 Valutazione personale

In generale, l'esperienza di stage mi ha permesso di entrare concretamente nel mondo del lavoro. È da poco più di un anno, infatti, che sento di aver bisogno di cambiare *routine* e fare nuove esperienze che non riguardino esclusivamente lo studio.
All'interno dell'azienda ho trovato persone molto qualificate e sempre disponibili, con le quali sono tutt'ora in contatto nell'ottica di avviare collaborazione una volta terminato il mio percorso di studi.
L'opportunità di lavorare all'interno di HNRG mi ha fatto crescere sia personalmente sia professionalmente, rendendomi conto sempre di più di aver scelto il percorso adatto a me.

Acronimi e abbreviazioni

API Application Programming Interface. 5, 33, 34, 37–39, 41, 47, 49–54, 57, 69

CMS Content Management System. 33, 66

DOM Document Object Model. 29, 47

ER Entity-Relationship. 31, 34, 65

IA Intelligenza Artificiale. 5, 41

npm Node Package Manager. 30, 34, 44

TDD Test-Driven Development. 56

UML Unified Modeling Language. 7, 25

Glossario

API In informatica, con il termine API (*Application Programming Interface*, in italiano Interfaccia di Programmazione di un'Applicazione) si indica un insieme di procedure che consentono la comunicazione tra diversi componenti. 68

backend In informatica per *backend* si intende tutto ciò che è necessario ad un'applicazione web per funzionare, ma che gli utenti non vedono e con cui non interagiscono. Ad esempio, la gestione del *database* oppure l'interazione con le API. 28, 29, 33, 41

DOM In informatica, è lo standard del W3C per la rappresentazione di documenti strutturati (come a esempio l'HTML). Ha un supporto ormai completo e permette di accedere ai vari elementi di una pagina web. 68

form Indica un'interfaccia che permette all'utente di inserire dati (tramite input di testo, scelta multipla...) ed inviarli ad un sistema. 11, 13, 16, 18, 43, 44, 57

framework Architettura logica di supporto che, imponendo una determinata struttura e implementando un particolare *design pattern*, facilita lo sviluppo di un *software*. 5, 30, 31, 43, 54, 56, 66

frontend In informatica per *frontend* si intende la parte dell'applicazione con cui gli utenti possono interagire. Esso si occupa principalmente di gestire i vari input dell'utente. 28, 29, 33, 38, 41, 43

IA Per Intelligenza Artificiale si intende la capacità di un sistema artificiale, allenato su un certo insieme di dati, di simulare l'intelligenza umana. 68

libreria In informatica, un insieme di funzioni o strutture dati facilmente integrabili ed utilizzabili in altri programmi *software*. 5

prompt Un testo scritto in linguaggio naturale e interpretabile da un modello di intelligenza artificiale. 41

repository In informatica, viene considerato un ambiente dove "depositare" in modo digitale dati ed informazioni riguardanti un progetto. Una specie di archivio digitale per codici sorgente, documenti e molto altro. 6, 28, 31

TDD In informatica, è un modello di sviluppo *software* che prevede che la stesura dei casi di test avvenga prima dello sviluppo del *software* stesso. 68

token In informatica per *token* si intende un codice univoco (composto solitamente da lettere e numeri) che permette di identificare un oggetto (molto spesso un utente) all'interno del sistema. 8, 19, 20, 27, 36, 39, 49, 59, 60

UML In ingegneria del *software*, è un linguaggio di modellazione unificato, che permette di rappresentare graficamente i vari aspetti della vita di un *software* (come l'analisi dei requisiti, la progettazione e l'implementazione). 68

Bibliografia

Siti web consultati

[1] *Atomic Design*. URL: https://atomicdesign.bradfrost.com/chapter-2/ (cit. a p. 43).

[2] *Best Practices for Testing Web Components*. URL: https://blog.pixelfreestudio.com/best-practices-for-testing-web-components/ (cit. a p. 56).

[3] *Bitbucket*. URL: https://bitbucket.org/product/ (cit. a p. 31).

[4] *Casi d'uso*. URL: https://www.math.unipd.it/~rcardin/swea/2022/Diagrammi%20Use%20Case.pdf (cit. a p. 7).

[5] *Concorso Accattivante e Accessibile*. URL: https://web.math.unipd.it/CAA/ (cit. a p. 58).

[6] *Demeter*. URL: https://demetertecweb.altervista.org/ (cit. a p. 58).

[7] *Directus*. URL: https://directus.io/ (cit. a p. 33).

[8] *Directus - AI Image Generation*. URL: https://directus.io/tv/ai/ai-image-generation (cit. a p. 41).

[9] *Directus Docs*. URL: https://docs.directus.io/ (cit. a p. 34).

[10] *Directus Docs - Filter Rules*. URL: https://docs.directus.io/reference/filter-rules.html (cit. a p. 38).

[11] *Directus Docs - Global Parameters*. URL: https://docs.directus.io/reference/query.html (cit. a p. 38).

[12] *FingerprintJS*. URL: https://github.com/fingerprintjs/fingerprintjs (cit. a p. 53).

[13] *Git*. URL: https://git-scm.com/ (cit. a p. 31).

[14] *Jest*. URL: https://jestjs.io/ (cit. a p. 31).

[15] *Jira*. URL: https://www.atlassian.com/it/software/jira (cit. a p. 32).

[16] *Node Package Manager*. URL: https://www.npmjs.com/ (cit. a p. 30).

[17] *Node.js*. URL: https://nodejs.org/en (cit. a p. 30).

[18] *Sass*. URL: https://sass-lang.com/ (cit. a p. 30).

[19] *Statistiche sull'accessibilità*. URL: https://www.accessiway.com/blog/10-statistiche-fondamentali-su-disabilita-e-accessibilita (cit. a p. 55).

[20] *Stencil*. URL: https://stenciljs.com/ (cit. a p. 43).

[21] *Stencil Docs*. URL: https://stenciljs.com/docs/introduction (cit. a p. 44).

[22] *Stencil Docs - VueJS Integration*. URL: https://stenciljs.com/docs/vue (cit. a p. 54).

[23] *TypeScript*. URL: https://www.typescriptlang.org/ (cit. a p. 29).

[24] *Vue.js*. URL: https://vuejs.org/ (cit. a p. 30).