



**UNIVERSITÀ  
DEGLI STUDI  
DI UDINE**  
hic sunt futura

**Dipartimento di Scienze  
Matematiche, Informatiche e Fisiche**

TESI DI LAUREA IN  
INFORMATICA

# **Sito web statico per gruppo di ricerca accademico: analisi e implementazione**

CANDIDATO

Federico Dittaro

RELATORE

Prof. Marino Miculan

CORRELATORE

Dott. Matteo Paier

Anno accademico 2022-2023

CONTATTI DELL'ISTITUTO

Dipartimento di Scienze Matematiche, Informatiche e Fisiche

Università degli Studi di Udine

Via delle Scienze, 206

33100 Udine — Italia

+39 0432 558400

<https://www.dmif.uniud.it/>

Dedica da scrivere



# Indice

<b>1</b>	<b>Introduzione</b>	<b>1</b>
<b>2</b>	<b>Analisi del software</b>	<b>3</b>
2.1	SSG	3
2.2	principali SSG	3
2.2.1	Hugo	3
2.2.2	Jekyll	4
2.2.3	Gridsome	5
2.2.4	Eleventy	6
2.2.5	Pelican	7
2.2.6	Tabella riassuntiva	8
2.3	Differenze tra siti statici e dinamici	8
2.4	Hugo	9
2.4.1	Formati supportati	9
2.4.2	Template actions	10
2.4.3	Plugin	10
2.4.4	Creazione del sito	12
2.4.5	Costruzione del sito	13
2.4.6	Hosting e distribuzione	13
2.4.7	Web Analytics	14
<b>3</b>	<b>Realizzazione di un sito web</b>	<b>17</b>
3.1	Analisi dei requisiti	17
3.2	User Experience	17
3.3	Scelta del tema	18
3.3.1	Tipologie di temi	19
<b>4</b>	<b>Caso di studio</b>	<b>21</b>
4.1	Tema utilizzato	21
4.1.1	Struttura di Tella	21
4.1.2	Implementazione di Tella	21
4.2	Personalizzazione del tema	22
4.3	Implementazione di funzionalità aggiuntive	23
4.3.1	Barra di ricerca	23
4.3.2	Gestione delle pubblicazioni	28
4.3.3	Umami	32
4.4	Hosting del sito	35
4.4.1	Pubblicazione	35
4.4.2	Aggiornare il sito	35
4.4.3	Risultati ottenuti	35
4.5	Portabilità del sito	35
4.5.1	Adattamento della sezione People	35
4.6	Accessibilità del sito	35

<b>5</b>	<b>Conclusioni</b>	<b>37</b>
<b>A</b>	<b>Glossario</b>	<b>39</b>

# 1

## Introduzione

In hac habitasse platea dictumst. Vestibulum consectetur dictum pellentesque. Suspendisse nunc neque, commodo ac imperdiet nec, sollicitudin vitae libero. Donec bibendum vel nunc vitae pharetra. In vel volutpat odio, et interdum dui. Duis mauris ligula, congue eget molestie at, tincidunt nec diam. Nam vitae eros nec arcu suscipit vehicula. Aliquam consectetur imperdiet elit, eget pretium arcu fringilla at. Maecenas [1] sed libero pulvinar, mattis tortor vel, fermentum enim.





# 2

## Analisi del software

### 2.1 SSG

Gli SSG (in inglese, Static Site Generator) sono dei tool che permettono la creazione di tutti i contenuti presenti nei siti web a partire da file di configurazione e contenuti scritti in formati più generali (tipicamente markdown). La caratteristica principale di tali siti è che a fronte di una richiesta da parte dell'utente di visualizzare determinati contenuti del sito, il web server fornisce pagine statiche, delle quali l'utente non è in grado di modificare il contenuto né possiede alcun tipo di stato che ne permetta la personalizzazione. Non esiste quindi una elaborazione back-end sul lato server e non esistono database, qualsiasi funzionalità "dinamica" associata al sito statico viene eseguita sul lato client.

I principali vantaggi riguardanti la scelta di utilizzo di un SSG sono:

- Ottimizzazione delle prestazioni: avendo poche o nessuna parte dinamica sono più facili da ottimizzare ed il caricamento è molto rapido;
- Richiesta di meno risorse al server: dato che non è richiesta nessuna elaborazione lato server, quest'ultimo svolge meno lavoro migliorando prestazioni e scalabilità;
- Servizio di hosting molto economico: possono essere utilizzati per la pubblicazione servizi di host completamente gratuiti come GitHub Pages (esattamente come nel caso di studio);
- Maggiore sicurezza: non utilizzando server o database sono molto sicuri da eventuali attacchi esterni.

### 2.2 principali SSG

In questa sezione verranno menzionati i cinque principali SSG e discusse le loro differenze più significative.

#### 2.2.1 Hugo

Hugo verrà brevemente presentato in modo da poterlo confrontare con gli altri principali SSG, per poi essere ripreso più approfonditamente nella Section 2.4 dato che si tratta del framework scelto per sviluppare il progetto.



Figura 2.1: Hugo logo

Hugo è un generatore di siti web statici scritto in Go ideato inizialmente da Steve Francia nel 2013 e successivamente sviluppato da Bjørn Erik. L'ultima versione, la 0.119.0 è stata rilasciata a settembre 2023. Per utilizzare Hugo non è necessario conoscere Go in quanto il sito web viene creato attraverso file HTML e CSS, inoltre c'è una separazione tra il contenuto e la presentazione permettendo così di modificare l'aspetto senza modificare l'informazione presente. Oltre ai tipi di file citati precedentemente, Hugo supporta anche file di tipo javascript, Markdown, TOML, YAML e JSON.

Le informazioni necessarie per creare, modificare, stilizzare o eliminare pagine e/o contenuti sono racchiuse all'interno di specifiche cartelle che possono essere successivamente estese. La struttura generale è la seguente:

- la cartella *archetypes* contiene i file che vengono utilizzati come template per la creazione di nuovi contenuti del sito in modo da standardizzare la struttura ed il formato;
- la cartella *content* è forse la più importante in quanto racchiude tutto il contenuto del sito. Al suo interno tutti i file sono in formato Markdown;
- la cartella *data* contiene esclusivamente file di tipo JSON, TOML, YAML o XML utilizzati per aggiungere strutture specifiche al sito;
- la cartella *layouts* contiene file HTML usati per creare l'aspetto visivo del sito;
- la cartella *static* contiene file statici come ad esempio immagini, file CSS, file Javascript;
- la cartella *themes* contiene i file che definiscono il tema del sito ed il suo aspetto visivo;
- il file *config.toml* oppure *config.yaml* è fondamentale in quanto rappresenta il file di configurazione e contiene informazioni globali come il titolo del sito, la sua descrizione e molto altro.

Hugo è noto per la sua velocità ed inoltre supporta una grande varietà di temi scaricabili direttamente dal sito ufficiale. A differenza di altri SSG, Hugo non è indicato solamente per la creazione di blog ma anche per la creazione di siti generici come ad esempio siti aziendali o, come nel caso di studio, per siti accademici. Hugo mette inoltre a disposizione una grande varietà di Plugin molto utili come ad esempio il servizio per la rappresentazione delle icone social o il supporto multilingua.

### 2.2.2 Jekyll

Jekyll è un generatore di siti web statici, ideato da Tom Preston-Werner, la prima versione del software risale al 2008 mentre l'ultima, la 3.9.3, è uscita a gennaio 2023. Jekyll si basa sul linguaggio Ruby, perciò richiede un'installazione ed una configurazione corretta e funzionante di tale ambiente. Successivamente si scarica la versione desiderata di Jekyll e si segue la procedura di installazione, così come descritta sulla documentazione. In Jekyll tutti i contenuti e i layout del sito vengono salvati localmente e vengono



Figura 2.2: Jekyll logo

classificati in una struttura a cartelle, principalmente orientata alla costruzione di blog.

Una volta creato il sito, la struttura trovata sarà la seguente:

- la cartella `_posts` contiene gli articoli del sito (composti da file Markdown);
- i contenuti delle pagine, sempre composti da file Markdown, sono salvati nella cartella `root`, in alternativa si può decidere di creare una gerarchia di sottocartelle per una migliore organizzazione dei contenuti;
- la cartella `_layouts` contiene i vari template del sito che decidono la grafica delle singole pagine e dei singoli articoli (questi file sono sempre di tipo HTML);
- la cartella `_site` contiene tutte le informazioni necessarie per esportare il sito funzionante nel dominio o in sistemi cloud;
- la cartella `_data` può essere creata per contenere dei file JSON in cui saranno costruiti dei database per immagazzinare stringhe, numeri e altri dati simili;
- la cartella `assets` contiene immagini, pdf o altri file statici per il sito.

Come Hugo anche Jekyll mette a disposizione centinaia di temi prefabbricati per aiutare lo sviluppo del sito web, ed entrambi forniscono degli shortcode, ovvero funzioni che permettono la comunicazione tra i layout delle pagine con i loro contenuti (ad esempio le template actions per Hugo). Anche Jekyll presenta una moltitudine di Plugin che possono essere integrati attraverso Ruby, permettendo di aggiungere e semplificare la costruzione di determinati servizi per il sito web.

Una delle differenze principali di Hugo rispetto a Jekyll è che il primo non è legato ad ambienti esterni, infatti dopo aver scaricato la versione desiderata ed estratto il contenuto nella cartella prescelta, il software è pronto per essere usato, mentre Jekyll si deve appoggiare a Ruby.

In conclusione, Jekyll è un'ottima scelta se si ha familiarità con l'ambiente Ruby o se si vuole costruire un sito complesso usando gli innumerevoli Plugin e template messi già a disposizione.

### 2.2.3 Gridsome

Gridsome è un SSG molto recente, è stato infatti ideato da Johannes Schickling nel 2018 subendo poi miglioramenti negli anni successivi fino all'ultima versione disponibile, la 0.7.23, rilasciata a settembre 2021. Si tratta di un framework basato su Vue.js e GraphQL che permette di creare una configurazione "headless", consentendo così di sfruttare la separazione dei contenuti dalla loro presentazione. Le cartelle



Figura 2.3: Gridsome logo



Figura 2.4: Eleventy logo

di lavoro possono variare in base alla configurazione specifica di un progetto, ma in generale sono strutturate nel modo seguente:

- *src* è la cartella principale al cui interno si trova il contenuto sorgente. È suddivisa in sottocartelle come *assets* per file statici, *components* per componenti Vue.js, *layouts* per i layout del sito, *pages* per le pagine principali e *templates* per i template utilizzati per la generazione di pagine dinamiche;
- *static* è la cartella utilizzata per i file statici che verranno serviti direttamente, come immagini, file CSS o JavaScript;
- *.gridsome* è la cartella che contiene le configurazioni specifiche di Gridsome. Include i file di configurazione e i dati temporanei generati durante la compilazione del sito;
- *gridsome.config.js* è il file di configurazione principale, definisce le impostazioni globali e le opzioni del progetto.
- *package.json* è il file che definisce le dipendenze del progetto e gli script personalizzati.

Anche Gridsome offre una vasta raccolta di plugin che possono essere utilizzati per estendere le funzionalità del generatore ma a differenza degli altri SSG presentati non è così adatto ai principianti, necessita infatti di una certa esperienza nello sviluppo web per poter riuscire a trarre il massimo da questo software.

#### 2.2.4 Eleventy

Eleventy (chiamato anche 11ty), come Gridsome, è un SSG molto recente ideato da Zach Leatherman nel 2018 e migliorato fino alla versione 2.0.0, rilasciata a febbraio 2023. La sua crescita è dovuta sia alla sua flessibilità e potenza ma anche al sempre maggior utilizzo delle piattaforme di hosting come "Chrome Developers" e "Netlify".

Questo software si basa su Node.js e javascript, il che richiede una buona conoscenza di quest'ultimo linguaggio per poter sfruttare a pieno la sua potenzialità. Eleventy supporta molteplici linguaggi di modello ma fondamentalmente si basa su Liquid, il che lo rende simile a Jekyll. Come tutti gli altri SSG supporta diversi linguaggi come ad esempio Markdown, JSON, YAML.

La struttura di base contiene almeno le seguenti cartelle:



Figura 2.5: Pelican logo

- *\_includes*: in questa cartella possono essere messi frammenti di codice HTML o template come ad esempio intestazioni o piè di pagina;
- *\_layouts*: questa cartella contiene i layout dei template come ad esempio un layout principale che include un'intestazione, piè di pagina e altri layout specifici;
- *\_data*: questa cartella è utilizzata per i file di dati, come file JSON o YAML, che vengono utilizzati per fornire dati dinamici alle pagine del sito;
- *\_pages*: in questa cartella vengono collocate le pagine del sito, utilizzando file che verranno convertiti in pagine HTML;
- *\_posts*: questa cartella viene utilizzata per i post del blog o altri contenuti dinamici;
- *\_assets*: questa cartella può essere utilizzata per archiviare i file statici come fogli di stile CSS, immagini e JavaScript. Tali file verranno copiati direttamente nella cartella *\_site* durante la generazione del sito;
- *.eleventy.js*: questo è il file di configurazione principale di Eleventy, in cui è possibile personalizzare il comportamento del generatore, definire i percorsi delle cartelle e altro ancora.

Eleventy non è sicuramente il miglior SSG per iniziare a lavorare in questo campo, sia per la necessità di conoscere un linguaggio come javascript sia per la documentazione poco esaustiva e ancora in fase di sviluppo, tuttavia lo sforzo viene guadagnato in termini di velocità e prestazioni.

### 2.2.5 Pelican

Pelican è meno conosciuto e utilizzato rispetto agli altri presenti in questa sezione, ma si tratta comunque di una valida alternativa data la sua flessibilità e la possibilità di importare siti già esistenti creati con altre piattaforme come ad esempio Wordpress. Questo SSG è stato creato da Justin Mayer nel 2010, l'ultima versione disponibile è la 4.8.0 rilasciata a luglio 2023 ed è stato principalmente sviluppato per la creazione di blog. A differenza degli altri ruota completamente attorno a python, creando pagine statiche attraverso file markdown e reStructuredText.

la struttura generale delle cartelle è la seguente:

- *content*: è la cartella principale in cui verranno collocati articoli, pagine e contenuti del blog;
- *output*: questa cartella contiene l'output generato da Pelican, ovvero il sito web statico completo con tutti i file HTML, CSS, JavaScript e le altre risorse pronte per la pubblicazione;

- *settings*: in questa cartella possono essere collocati file di configurazione specifici per il progetto, utilizzati per personalizzare il comportamento di Pelican;
- *themes*: qui si possono collocare i temi o modelli per il sito web;
- *plugins*: utilizzata per estendere le funzionalità di Pelican. I plugin possono essere utilizzati per eseguire varie azioni, come la generazione automatica di mappe del sito o l'integrazione con servizi di terze parti;
- *media*: questa cartella può essere utilizzata per archiviare file multimediali, come immagini o video, che verranno inclusi nel sito web;
- *pages*: In questa cartella vengono collocate le pagine statiche che non sono articoli del blog;
- *archives*: questa cartella può essere utilizzata per archiviare articoli o pagine che non sono attualmente pubblici ma potrebbero essere utilizzati in futuro;
- *lib*: contiene script personalizzati o strumenti per il progetto Pelican.

Pelican offre inoltre una vasta serie di temi e plugin per estendere le funzionalità di base del generatore, come il supporto multilingua o la possibilità di importare dati di terze parti (es. Wordpress, feed RSS).

### 2.2.6 Tabella riassuntiva

La seguente tabella riassume le caratteristiche dei cinque principali SSG visti nelle sezioni precedenti.

	Anno creazione	Ultima versione	Linguaggio di base
Hugo	2013	0.119.0 settembre 2023	Go
Jekyll	2008	3.9.3 gennaio 2023	Ruby
Gridsome	2018	0.7.23 settembre 2021	Vue.js, GraphQL
Eleventy	2018	2.0.0 febbraio 2023	Node.js, javascript
Pelican	2010	4.8.0 luglio 2023	Python

Tabella 2.1: Tabella riassuntiva SSG.

## 2.3 Differenze tra siti statici e dinamici

Riprendendo la Section 2.1, un'ulteriore differenza che si può notare tra siti statici e dinamici riguarda l'architettura client/server utilizzata.

Per quanto riguarda i siti dinamici, l'architettura adottata prende il nome di WAMP/LAMP, l'acronimo è composto dalle seguenti parole: la W/L riguarda il sistema operativo utilizzato (rispettivamente Windows oppure Linux), la A sta per Apache ovvero il server http impiegato per eseguire il server web, la M per il DBMS MySQL e la P per PHP. Alcune architetture utilizzano al posto di MySQL il DBMS MariaDB e al posto del PHP i linguaggi Python oppure Pearl (l'acronimo rimane comunque invariato).

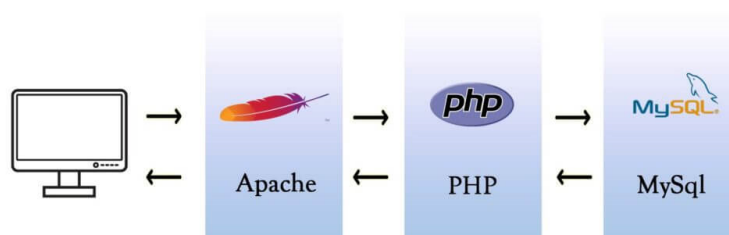


Figura 2.6: Architettura WAMP/LAMP

WAMP è quindi la piattaforma di sviluppo web grazie a cui è possibile realizzare dei siti web dinamici attraverso la programmazione con il linguaggio PHP, permettendo così la creazione dei siti direttamente sul computer. Utilizzare questa architettura attraverso il PHP offre diversi vantaggi in quanto si tratta di uno dei linguaggi più frequentemente utilizzati in rete ed inoltre è il linguaggio di riferimento per WordPress, la principale piattaforma software per lo sviluppo di siti dinamici.

Per quanto riguarda i siti statici invece, una volta generato il sito è sufficiente caricare i file su qualsiasi web server o servizio di hosting, non avendo infatti la necessità di eseguire codice dinamico (es. PHP) e di interfacciarsi con un database, la struttura risultante è molto più snella. Avere un'architettura di questo tipo permette di ottenere delle ottimizzazioni delle prestazioni e della velocità necessaria per servire la pagina richiesta.

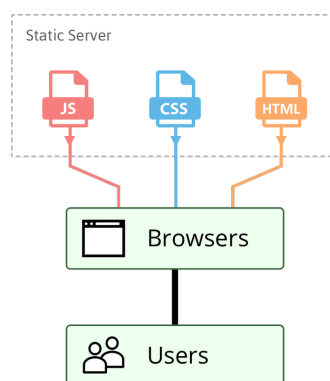


Figura 2.7: Architettura client/server SSG

## 2.4 Hugo

In questa sezione verrà ripreso ed ampliato il contenuto della Section 2.2.1 in quanto Hugo rappresenta l'SSG scelto per la realizzazione del caso di studio.

### 2.4.1 Formati supportati

Hugo supporta una grande varietà di formati, i principali sono i seguenti:

- **MARKDOWN**: formato di markup contenente esclusivamente il contenuto informativo, in questi file non viene definito nessun tipo di stile;

- **HTML:** formato utilizzato per definire l'organizzazione e la struttura delle pagine;
- **CSS:** formato utilizzato per definire lo stile delle pagine (es. colore, sfondo, dimensioni e tipo di carattere, ...);
- **JavaScript:** questi file vengono utilizzati principalmente per l'esecuzione di script, sono fondamentali per l'esecuzione di specifici task.
- **JSON:** vengono utilizzati per definire metadati e contenuti strutturati all'interno del sito web;
- **TOML, YAML:** questi file vengono utilizzati per definire o aggiungere dettagli strutturali o globali del sito e delle singole pagine.

Vengono inoltre supportati formati più specifici come `reStructuredText`, un formato markup utilizzato ad esempio per la documentazione tecnica, oppure `AsciiDoc`, anch'esso un formato markup spesso utilizzato in ambienti di sviluppo.

### 2.4.2 Template actions

Le Template actions, scritte in Go Template, si riferiscono a una serie di azioni o comandi disponibili nei modelli (templates) di Hugo, che vengono utilizzati per manipolare e visualizzare i dati all'interno dei contenuti. Le template actions consentono di eseguire operazioni come l'iterazione attraverso gli elenchi, il controllo di flusso condizionale e l'estrazione di valori dai dati dei contenuti. Vengono utilizzate direttamente all'interno dei file HTML all'interno delle doppie graffe.

Alcune delle template actions più comuni in Hugo includono:

- **range:** utilizzata per iterare all'interno di un elenco, permette ad esempio di creare dei cicli.
- **if, with:** consentono di aggiungere logica condizionale nei modelli. Possono essere utilizzati per visualizzare o nascondere parti del contenuto in base a condizioni specifiche;
- **index:** utilizzata per accedere a elementi specifici all'interno di elenchi o dizionari;
- **partial:** per includere modelli o contenuti da altri file.

La Fig. 2.8 mostra un esempio di utilizzo del controllo di flusso attraverso il costrutto "if", nello specifico verifica se nei parametri che descrivono la persona è presente un'immagine: in caso positivo mostra l'immagine corrispondente altrimenti un'immagine di default. La Fig. 2.9 mostra invece l'utilizzo della template action "partial", permettendo così di includere il file *recent.html*.

Nell template actions è possibile anche selezionare specifici parametri come ad esempio `{{ .Title }}` che permette di inserire il titolo della pagina corrente. Allo stesso modo si può ottenere la descrizione (`{{ .Description }}`) o il sommario (`{{ .Summary }}`).

### 2.4.3 Plugin

Hugo supporta una grande varietà di Plugin che permettono di estendere le funzionalità di base, in questa sezione ne verranno presentati due tra i più utilizzati.



## Hugo-authors

Il plugin "hugo-authors" è progettato per semplificare la gestione dei dati degli autori. Consente di definire e visualizzare le informazioni sugli autori come nome, biografia, immagine del profilo e collegamenti ai social media, in modo più strutturato.

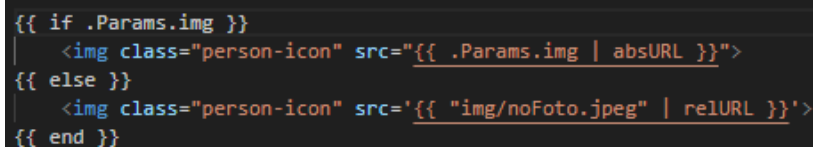
Per poter utilizzare questo plugin è necessario prima installarlo attraverso il modulo Hugo o includendolo nel file di configurazione (config.toml oppure config.yaml). Successivamente è possibile creare dei nuovi autori attraverso la seguente sintassi:

```
[author]
[[author.authors]]
name = "author name"
bio = "author bio"
image = "author.jpg"
social = [
{ name = "Twitter", link = "link" },
{ name = "LinkedIn", link = "link" }
]
```

A questo punto è possibile associare gli autori al rispettivo contenuto specificando nel *front-matter* del file markdown il nome corrispondente. Il nome svolge quindi la funzione di ID. Ad esempio:

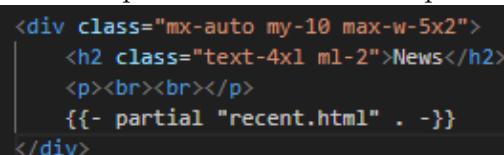
```
+++
title = "Il mio articolo"
author = "author name"
+++
```

Sfruttando infine le Template actions è possibile accedere alle caratteristiche dell'autore, ad esempio si può ricavare il nome scrivendo `{{ .Params.author }}` oppure applicando la stessa logica si possono



```
{{ if .Params.img }}
  
{{ else }}
  
{{ end }}
```

Figura 2.8: Esempio di utilizzo della template action "if"



```
<div class="mx-auto my-10 max-w-5x2">
  <h2 class="text-4xl ml-2">News</h2>
  <p><br><br></p>
  {{- partial "recent.html" . -}}
</div>
```

Figura 2.9: Esempio di utilizzo della template action "partial"

ricavare le varie informazioni inserite (ad esempio la biografia).

## Servizio delle icone social

Questo plug-in non necessita di installazione e permette di visualizzare facilmente icone e collegamenti ai profili social. Per poterlo utilizzare bisogna aggiungere le seguenti righe di codice al file di configurazione

```
[social]
platforms = [
name = "Twitter", icon = "twitter" ,
name = "LinkedIn", icon = "linkedin" ,
name = "GitHub", icon = "github" ,
# Altre piattaforme social
]

icons = [
name = "twitter", link = "https://example.com/twitter-icon.svg" ,
name = "linkedin", link = "https://example.com/linkedin-icon.svg" ,
name = "github", link = "https://example.com/github-icon.svg" ,
# Altre icone social
]
```

Solo successivamente possono essere specificate all'interno della definizione degli autori (sezione precedente) o nel front matter di un contenuto. Allo stesso modo del plugin precedente, le informazioni possono essere richiamate all'interno dei file HTML sfruttando le Template actions (in questo caso tramite `{{ .Site.Data.social.icons }}`).

### 2.4.4 Creazione del sito

Dopo l'installazione di Hugo, per creare un nuovo sito è sufficiente digitare da terminale il seguente comando **hugo new site NOME\_SITO**. In seguito a questo comando verrà costruita la struttura delle cartelle così come descritto nella Section 2.2.1 che può essere modificata secondo le esigenze.

Per poter visualizzare in locale il sito che si sta creando, è necessario lanciare da terminale il comando **hugo server** oppure **npm run start** (se è stato installato npm), in seguito il sito sarà visibile all'indirizzo `localhost:1313` (`127.0.0.1:1313`).

Altri comandi utili per la creazione del sito sono **hugo server -D** che, a differenza del comando precedente, permette di visualizzare anche i contenuti classificati come bozze ed il comando **hugo**, necessario per la costruzione del sito. Quest'ultimo comando verrà discusso nella sezione successiva.

### 2.4.5 Costruzione del sito

La costruzione del sito è divisa in due fasi principali: la fase di generazione e la fase di compilazione.

#### Fase di generazione

Il processo di generazione viene avviato attraverso il comando **hugo**, il quale genera un unico file CSS combinando i dati di file di template dei temi con eventuali fogli di stile personalizzati, unendo successivamente anche il contenuto dei file Markdown.

#### Fase di compilazione

Terminata la fase di generazione, Hugo applica i temi e genera il codice HTML e CSS per ogni pagina del sito. L'ultima fase prima di poter rendere le pagine distribuibili e visualizzabili consiste nel produrre un insieme di fogli di stile CSS, immagini e altri file statici, ciascuno rappresentante una pagina del sito. Una variante più efficiente è il comando **hugo - -minify** che ottimizza il caricamento delle pagine minimizzando la dimensione dei file attraverso l'eliminazione di spazi vuoti, commenti e caratteri non necessari.

Il risultato del processo di compilazione è la creazione di due cartelle:

- **public**: contiene il sito costruito ed è organizzata a sua volta in sottocartelle ognuna delle quali contiene una pagina singola oppure un gruppo di pagine correlate. Ogni sottocartella comprende un file HTML, i fogli di stile e i file JavaScript;
- **resources**: contiene le cache di output del processo di generazione; nello specifico, contiene risorse come immagini, file CSS, file JavaScript e altri assets statici utilizzati nel sito.

### 2.4.6 Hosting e distribuzione

In seguito alla generazione e compilazione, il sito può essere caricato su qualsiasi web server o servizio di hosting. Come detto precedentemente, non è necessario eseguire elaborazioni lato server oppure richieste ad un database, pertanto è possibile utilizzare servizi di hosting completamente gratuiti.

Un esempio di questi servizi è *Github Pages*, offerto da GitHub. Quest'ultimo è uno dei servizi di hosting di repository Git più popolari al mondo che offre servizi per il controllo di versione, la collaborazione e l'hosting di progetti software (attraverso appunto Github Pages). Alcune delle principali funzionalità di GitHub includono il controllo di versione distribuito, il tracciamento delle modifiche, la collaborazione tra sviluppatori e molto altro. GitHub Pages è uno dei servizi offerti da Github che consente di pubblicare siti web statici direttamente da una repository GitHub, in particolare utilizza uno specifico branch chiamato "gh-pages" dal quale preleva il contenuto che viene poi pubblicato. Di default, l'indirizzo che viene assegnato è "[https://NOME\\_UTENTE.github.io/NOME\\_REPOSITORY/](https://NOME_UTENTE.github.io/NOME_REPOSITORY/)" ma supporta anche domini personalizzati.

GitHub Pages tuttavia presenta dei limiti sulla dimensione dei siti che consente di hostare:

- Le dimensioni della repository utilizzata per creare il sito web non deve essere superiore ad un 1GB;
- Le risorse messe a disposizione sono limitate a 100GB di banda e 100.000 richieste al mese;
- Non si dovrebbero eseguire più di dieci "build" per ora, per evitare di sovraccaricare i server e mantenerli performanti per tutti gli utenti.

I limiti appena citati possono essere superati attraverso un piano di pagamento del servizio.

Altri limiti imposti e non superabili riguardano il contenuto del sito web, GitHub infatti non consente di utilizzare questo servizio per gestire attività online e servizi di software commerciale, inoltre l'utilizzo della pagine è soggetto ai Termini di servizio di Github, per cui sono vietati contenuti sessuali oppure violenti/minacciosi.

### 2.4.7 Web Analytics

Le web analytics sono il processo di raccolta, misurazione, analisi e reporting dei dati relativi all'uso di un sito web al fine di comprendere il comportamento degli utenti, valutare le prestazioni del sito e trarre informazioni utili per migliorare l'esperienza degli utenti. Le web analytics sono fondamentali per il miglioramento dell'usabilità, l'ottimizzazione dei motori di ricerca e l'analisi delle conversioni.

- **Raccolta dei Dati:** la raccolta dei dati inizia con l'implementazione di strumenti di analisi web che tracciano il comportamento degli utenti sul sito. Questi strumenti registrano dati come il numero di visite, il tempo trascorso sul sito, le pagine visitate e molto altro;
- **Misurazione delle Metriche Chiave:** le web analytics misurano una vasta gamma di metriche, tra cui il traffico del sito, l'origine del traffico, il tasso di rimbalzo, il tempo medio sulla pagina, le conversioni e altro ancora. Queste metriche consentono di valutare le prestazioni del sito;
- **Analisi dei Dati:** l'analisi dei dati comporta l'interpretazione delle metriche raccolte per comprendere i modelli e le tendenze. Ad esempio, analizzare quale fonte di traffico genera più conversioni o quali pagine del sito hanno un alto tasso di rimbalzo;
- **Reporting:** la creazione di report è un passaggio importante nelle web analytics. I report consentono di comunicare i risultati dell'analisi in modo chiaro e comprensibile per poter essere condivisi con i membri del team, i responsabili decisionali o i clienti;
- **Ottimizzazione:** le web analytics sono utilizzate per prendere decisioni sull'ottimizzazione del sito. Ciò può includere l'ottimizzazione del contenuto, delle conversioni, del carico delle pagine e altro ancora al fine di migliorare l'esperienza dell'utente e raggiungere gli obiettivi stabiliti;
- **Segmentazione degli Utenti:** le web analytics consentono di suddividere gli utenti in segmenti in base a criteri come la provenienza geografica, il comportamento di navigazione, il dispositivo utilizzato e altro ancora. Questa segmentazione aiuta a comprendere meglio il pubblico e a personalizzare l'esperienza dell'utente.

Uno delle più famose piattaforme di analisi web è sicuramente *Google Analytics* che utilizza come strumento di analitica i cookies. Nel sito web che si è realizzato con questo progetto, viene utilizzata come piattaforma di analisi uno strumento meno conosciuto di Google Analytics: *umami*. Si tratta di un'applicazione open-source compatibile con il GDPR che a differenza di Google Analytics non utilizza cookie di tracciamento ma allo stesso tempo permette di ottenere informazioni come le pagine visitate, il browser, il sistema operativo ed il dispositivo utilizzati, nonché il paese di provenienza. Queste informazioni di base possono poi essere ampliate comprendendo ad esempio specifici eventi di interesse per l'analisi del sito web. Umami mette inoltre a disposizione una dashboard per la presentazione del numero di visite nel corso del tempo, permettendo di visualizzare gli accessi nella giornata, nell'ultima settimana, nell'ultimo mese oppure nell'anno corrente.

L'utilizzo e l'implementazione di umami verranno riprese nel Chapter 4.



# 3

## Realizzazione di un sito web

### 3.1 Analisi dei requisiti

Nella realizzazione di un sito web bisogna tenere in considerazione diversi aspetti che possono influenzare la buona riuscita del progetto, come ad esempio l'usabilità, l'accessibilità, l'organizzazione dei contenuti e molto altro. Per questo motivo esistono delle linee guida e degli approcci consolidati che permettono di creare siti web di buona qualità. Queste "best practice" si basano sull'esperienza e su ricerche condotte nel campo della progettazione web, le più comuni sono:

- **Usabilità:** il sito deve essere intuitivo da utilizzare, la navigazione deve essere chiara e gli utenti dovrebbero essere in grado di trovare facilmente ciò di cui hanno bisogno;
- **Responsive Design:** il sito deve essere ottimizzato per dispositivi desktop, tablet e smartphone. Deve quindi adattarsi a diverse dimensioni di schermo;
- **Velocità di caricamento:** I siti web devono caricarsi rapidamente, riducendo al minimo il tempo di caricamento delle pagine e delle immagini;
- **Struttura delle informazioni:** organizzare il contenuto in modo logico utilizzando una struttura di navigazione chiara con menu, sottosezioni e collegamenti interni;
- **Chiarezza e semplicità:** mantenere il design pulito e senza distrazioni, l'utente dovrebbe poter comprendere facilmente il messaggio e la struttura del sito.

La loro applicazione dipenderà ovviamente dalle esigenze specifiche del sito e dal pubblico di destinazione, devono perciò essere adattate in base alle esigenze.

Queste linee guida verranno poi riprese ed implementate nel Chapter 4.

### 3.2 User Experience

Il fattore forse più importante da tenere in considerazione nella realizzazione di un sito web è l'esperienza dell'utente (UX), questo termine si riferisce a quello che prova una persona quando interagisce o immagina di interagire con un prototipo, un prodotto finito, un sistema od un servizio. L'UX oltrepassa

i limiti di altri termini come "interfaccia utente" e "usabilità" in quanto comprende tutti gli aspetti dell'esperienza che le persone hanno con i sistemi interattivi.

Sono stati identificati tre livelli, basati sulle risorse fisiologiche e psicologiche mobilitate dall'utente durante l'interazione:

- *viscerale*: ricevere e interpretare gli stimoli che provengono dal mondo esterno in modo automatico ed immediato;
- *comportamentale*: viene analizzato l'uso del sistema, la prestazione;
- *riflessivo*: considera gli aspetti culturali e il significato dell'uso di un prodotto.

L'esperienza dell'utente ha inizio prima dell'effettivo utilizzo in quanto, una tecnologia o l'immagine che vediamo di essa, può incuriosire, emozionare ed indurre risposte "viscerali", lasciando immaginare quello che potremmo fare avendola a nostra disposizione. Durante l'interazione si possono identificare la soddisfazione dell'utente, ovvero il grado con cui gli utenti giudicano quanto le tecnologie incontrano le loro esigenze, e la piacevolezza, considerata una componente rilevante della soddisfazione e provata dalle persone durante l'uso di un prodotto. Quest'ultima non è legata a termini funzionali ma all'esperienza sensoriale e alle emozioni positive suscitate.

La user experience è ormai diventata una condizione necessaria per la sopravvivenza e la crescita di qualsiasi prodotto o servizio, una UX insoddisfacente può portare ad un calo dei guadagni e della visibilità generale.

### 3.3 Scelta del tema

La scelta di un tema è una decisione importante nel processo di sviluppo di un sito web, la cui decisione dipende ovviamente del tipo di progetto che si sta realizzando. La sua importanza non è dovuta solo all'aspetto visivo ma soprattutto all'organizzazione dei contenuti e alle funzionalità che offre.

Hugo mette a disposizione sul sito ufficiale una grande varietà di temi suddivisi in categorie (Blog, Company, Minimal, ...) che facilitano quindi la scelta in base a ciò che si vuole realizzare. Inoltre per la maggior parte dei temi è possibile provare una demo che permette la visualizzazione dell'aspetto generale e la navigazione nelle varie sezioni in modo da poterne analizzare la struttura. Sono poi generalmente presenti la documentazione del tema stesso, le specifiche per scaricarlo, il collegamento alla repository di GitHub ed informazioni più specifiche come la versione di Hugo necessaria per eseguire il tema e l'ultimo aggiornamneto disponibile.

In seguito alla procedura di installazione, avendo a disposizione i file sorgenti, è possibile modificare il comportamento o aggiungere funzionalità al tema scelto. Allo stesso modo, se nessun tema dovesse soddisfare le esigenze richieste, è possibile realizzare un tema personalizzato creando una sottocartella *layouts* all'interno della directory *themes* e caricare al suo interno i vari file HTML, CSS e JS.

Il progetto implementato in questa tesi prevede la realizzazione di un sito web accademico, pertanto la scelta del tema non è casuale. Osservando altri siti accademici di varie università italiane si può vedere una struttura comune. In linea genarale si può notare infatti un menù orizzontale posto nella parte superiore dello schermo che fornisce i link interni alle varie informazioni presenti. Un ulteriore problema che si pone è la scelta del numero di voci, questa problematica verrà ripresa nel Chapter 4





Figura 3.1: Menù laboratorio ARC Roma

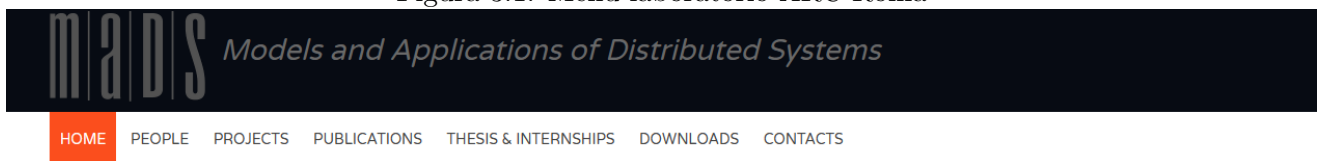


Figura 3.2: Menù MADS Uniud

### 3.3.1 Tipologie di temi

Un'altra distinzione degna di nota in Hugo riguarda la possibilità di installare "temi standard" e "temi non standard". I primi si riferiscono ai temi precedentemente descritti mentre i secondi sono più complessi in quanto offrono delle funzionalità e dell'estetica superiore, rendendo necessariamnete più complesso e costoso lo sviluppo ed il mantenimento.

#### Installazione temi standard

La configurazione di un tema standard è relativamente semplice, una volta inizializzato un nuovo sito (così come descritto nella Section 2.4.4), il nuovo tema va scaricato seguendo le istruzioni all'interno della cartella *themes*. A questo punto l'ultimo passaggio è collegare il sito creato con il tema appena scaricato, per farlo è necessario modificare il file di configurazione (`config.toml` oppure `config.yml`) aggiungendo la riga **theme="NOME\_TEMA"**.

Se successivamente si desidera cambiare il tema, è necessario scaricarlo all'interno della stessa cartella del precedente e modificare il nome nel file di configurazione (NB: i temi possono coesistere all'interno della cartella *themes*, non è necessario cancellare il precedente).

#### Installazione temi non standard

I temi non standard modificano significativamente la struttura dei siti in cui vengono utilizzati, generalmente hanno più file di configurazione e modificano anche altre cartelle (ad esempio la cartella *content*) e per questo necessitano la creazione di altre cartelle per il salvataggio di determinati contenuti.



# 4

## Caso di studio

### 4.1 Tema utilizzato

Nel caso di studio sono state prese in considerazione le assunzioni e le regole generali della Section 3.3, per questo motivo il tema che è stato scelto è **Tella**. La pagina di presentazione presente sul sito ufficiale di Hugo mostra alcune informazioni generali come l'autore, l'ultimo update rilasciato ed il collegamento alla repository di GitHub.

Nella stessa pagina è poi disponibile la possibilità di provare la demo e sono inoltre presenti le varie modalità di installazione ed inizializzazione. Questi passaggi verranno ripresi ed utilizzati nelle sezioni successive.

#### 4.1.1 Struttura di Tella

Sebbene Tella sia stato utilizzato per la realizzazione di un sito accademico, nel sito di Hugo viene classificato nel gruppo "Company". Tuttavia possiede alcune caratteristiche riconducibili ad un sito adatto per un'università, soprattutto se vengo effettuati alcuni accorgimenti e rimosse alcune sezioni. In particolare si può sfruttare la presenza del menù orizzontale, in quanto rappresenta proprio una delle caratteristiche cercate.

Il file di configurazione generale *config.toml* è presente all'interno della cartella *themes/tella/exampleSite* e permette di modificare le configurazioni globali del sito come il logo, la struttura della pagina di index e del menù orizzontale.

#### 4.1.2 Implementazione di Tella

Prima di poter installare il tema scelto è necessario creare un nuovo sito da terminale attraverso il comando **hugo new site NOME\_SITO**, a questo punto si seguono le istruzioni riportate sulla pagina di presentazione di Tella (è richiesta l'installazione di Git):

1. Dalla cartella creata attraverso il comando precedente, digitare  
**git clone https://github.com/opera7133/tella themes/tella**, il risultato sarà la creazione di una nuova cartella "Tella" all'interno di "Themes" dove sarà presente tutto il contenuto della repository clonata;

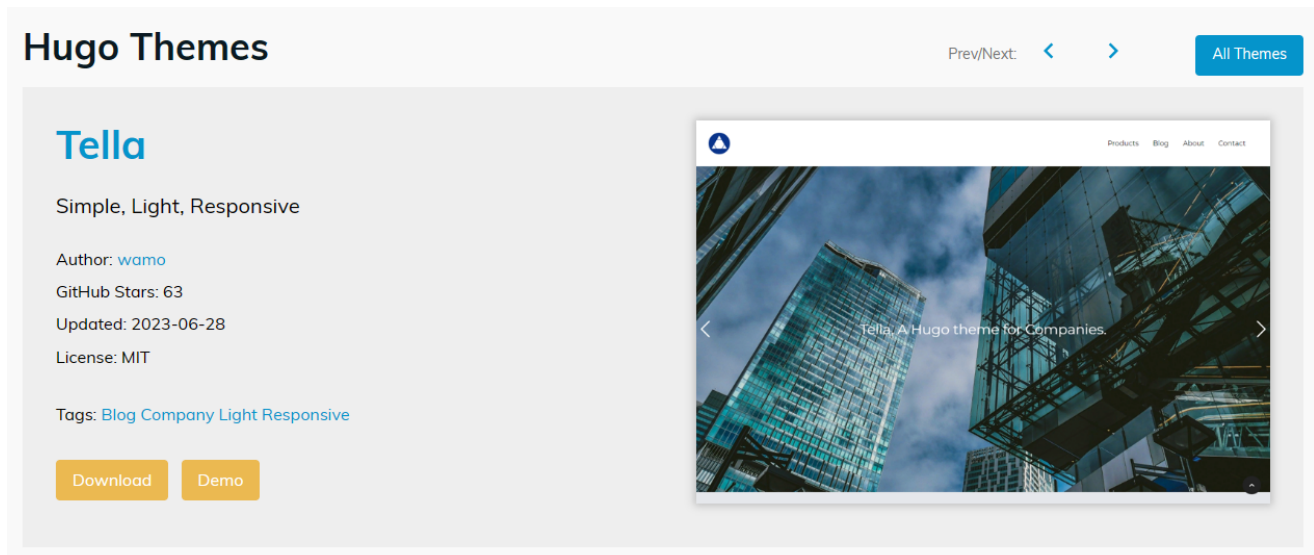


Figura 4.1: Pagina di presentazione Tella

2. Il seguente comando è necessario per aggiornare il tema in caso di modifiche alla repository di Github, alla prima creazione non è pertanto necessario. Per farlo bisogna prima spostarsi all'interno della cartella "Tella" attraverso il comando **cd tella**, a questo punto si può eseguire **git pull**;
3. Installazione delle dipendenze: seguendo i passi elencati nel sito ufficiale è necessario copiare i file "package.json", "tailwind.config.js" e "postcss.config.js" dalla cartella *exampleSite* alla cartella radice del progetto;
4. L'ultimo passo prima di poter visualizzare il sito è quello di installare npm attraverso il comando **npm install**;
5. Una volta installato, npm permette di visualizzare il sito digitando **npm run start** (equivalente a **hugo server**) all'indirizzo 127.0.0.1:1313.

## 4.2 Personalizzazione del tema

In questa sezione verranno descritte le modifiche iniziali apportate al tema per renderlo compatibile con il risultato cercato. Saranno illustrate le soluzioni adottate e le scelte effettuate.

### Personalizzazione del menù orizzontale



Figura 4.2: Voci presenti nel menù

Come detto precedentemente, il menù orizzontale già presente può essere modificato e adattato in modo da essere funzionale alle specifiche. Una considerazione importante riguarda la scelta del numero di voci, troppe possono infatti rendere la visualizzazione confusa ma la ricerca di informazioni più efficiente, mentre troppo poche rendono la ricerca di contenuti più complicata ma permettono una visualizzazione più compatta. Analizzando siti accademici già esistenti si è scelto di utilizzare sei voci, organizzate come illustrato in Fig. 4.2.

## Modifica della pagina iniziale

La pagina iniziale, come si può vedere nelle Fig. 4.3 e Fig. 4.4, è stata modificata rispetto all'originale, sono stati infatti rimossi e modificati diversi elementi:

- Le scritte presenti nelle immagini sono state cancellate attraverso l'eliminazione di parte del contenuto presente nel file *slide.json* all'interno della cartella *data*;
- Il blocco intermedio posto sotto le foto scorrevoli è stato rimosso eliminando il file *features.json* nella stessa cartella del punto precedente;
- Parte del menù posto in fondo alla pagina è stato cancellato con l'eliminazione del contenuto del file *footer.html* all'interno della cartella *themes/tella/layouts/partials*

È stata poi creata una sezione di "About us" immediatamente sotto le immagini, la quale conterrà una descrizione del lavoro svolto dal specifico gruppo di ricerca. Si è scelto di assegnare un colore di sfondo diverso dal resto della pagina per risaltarne la posizione ed il contenuto.

L'ultima modifica effettuata riguarda l'informazione ed il collegamento interno presente nella pagina di index. Si è scelto di riservare questo spazio per l'elencazione delle news in modo tale che vengano presentate le tre più recenti mentre l'elenco completo è reso disponibile sia attraverso la voce nel menù orizzontale sia attraverso il tasto "**All news**" in fondo alla pagina. Per realizzare questa idea sono state sfruttate le Template actions descritte nella Section 2.4.2, la pagina di index include infatti il file *recent.html* posto all'interno della cartella *partials* che attraverso la funzione di range (in questo caso impostata al valore 3) permette di realizzare l'obiettivo. Questo file richiama a sua volta *summary.html*, utilizzato per rappresentare il titolo, la breve descrizione e la data presenti in ogni news.

## 4.3 Implementazione di funzionalità aggiuntive

Nella sezione precedente sono state elencate le modifiche e le cancellazioni rispetto all'originale. Verranno ora trattate le aggiunte più significative non presenti nella versione iniziale ed adottate in quanto necessarie ad un generale sito web accademico.

### 4.3.1 Barra di ricerca

L'implementazione della barra di ricerca è stata svolta utilizzando la libreria *Lunr.js* ed utilizza una serie di file organizzati in una struttura non banale, la quale verrà descritta nelle pagine successive. La soluzione è ispirata da **link**.

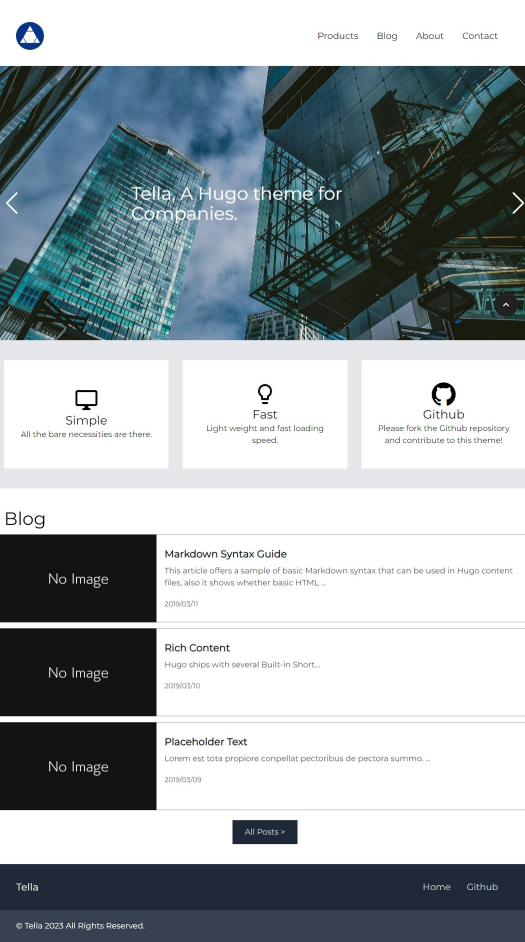


Figura 4.3: Pagina di index originale

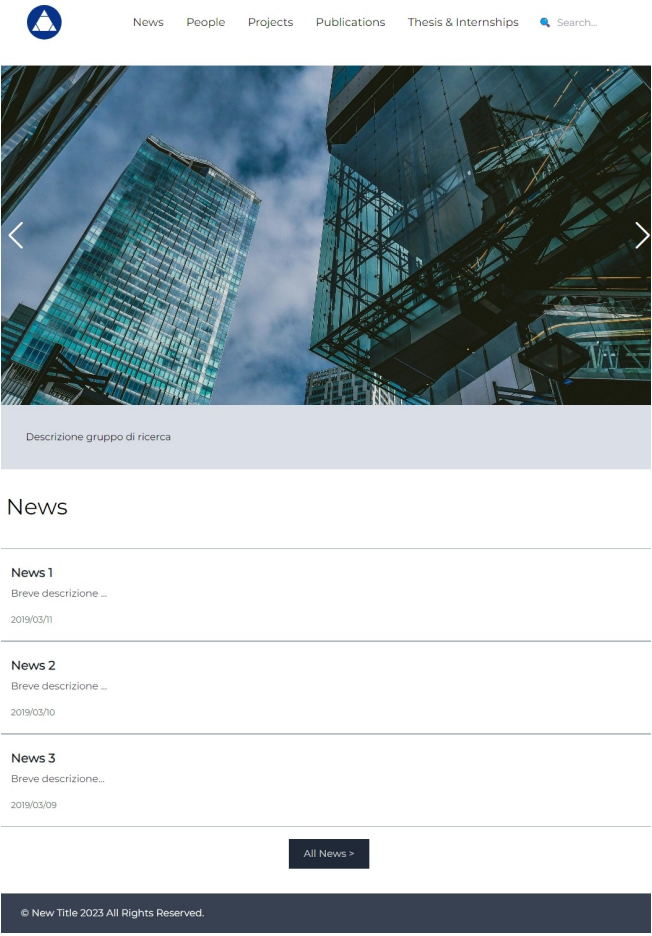


Figura 4.4: Pagina di index modificata

Dato che nell'architettura di un sito web statico (descritta in Fig. 2.7) non è presente un'elaborazione lato server, in quanto non ci sono database per eseguire le computazioni, l'operazione di ricerca verrà gestita completamente nel lato client. In particolare la libreria permette di effettuare staticamente delle ricerche e di indicizzare i contenuti. I passi seguiti per la realizzazione della barra di ricerca sono i seguenti:

1. Creazione della pagina Markdown per la visualizzazione dei risultati: è necessario creare una cartella "search" all'interno della cartella *content*, questa cartella conterrà il file "\_index.md" il quale racchiude solamente il front-matter con la riga **title: Search**;
2. Ogni file markdown necessita la creazione di un file HTML di template per la presentazione dei contenuti. Tale file, chiamato *list.html* è contenuto all'interno della cartella *layouts/search*.

La Fig. 4.5 mostra l'organizzazione della pagina di visualizzazione dei risultati. Il file è organizzato in due parti importanti, la prima riguarda la sezione **h2** dove l'id "search-title" verrà utilizzato come titolo della pagina, in quanto viene salvato e richiamato dalla template action {{ .Name }}. La seconda parte riguarda il tag **ul** che verrà utilizzato per elencare i risultati di ricerca.

3. Costruzione della struttura dati: questo avviene attraverso il file "search-index.html" posto all'interno della cartella *themes/tella/layouts/partials*. Il file contiene il seguente codice:

```

<script>
window.store = {
  {{ range .Site.RegularPages }}
  "{{ .Permalink }}" : {
    "title": "{{ .Title }}",
    "content": {{ .Content | plainify }}, // Strip out HTML tags
    "url": "{{ .Permalink }}"
  },
  {{ end }}
}

```

```

{{ define "main" }}

<h2 id="search-title", class="title-research">
  {{ .Name }}
</h2>

<!-- risultati di ricerca -->
<ul id="results">
  <li>
    Search results would be shown here.
  </li>
</ul>

{{ partial "search-lunr.html" . }}

{{ end }}

```

Figura 4.5: Contenuto del file *list.html*

```

{{ partial "search-index.html" . }}

<form id="search" action='{{ with .GetPage "/search" }}{{.Permalink}}{{end}}' method="get">
  <input type="text" id="search-input" name="query" placeholder="🔍 Search..." style="width: 150px; height:40px;">
</form>

```

Figura 4.6: Contenuto del file *search-form.html*

```

};
i/script;

```

*window.store* è una variabile JavaScript definita nell’ambiente del browser web, l’oggetto *window* rappresenta appunto la finestra del browser mentre *.store* è un meccanismo di archiviazione dei dati in locale che risulta fondamentale nella realizzazione della barra di ricerca in quanto questo archivio verrà successivamente utilizzato da Lunr.js.

Lo script processa ciclicamente tutte le RegularPages, ovvero quelle pagine che nel front matter hanno la voce **draft: false** (se la voce non è presente viene considerata come se fosse false) ed il Permalink viene utilizzato come identificatore dell’oggetto creato da *window.store*.

4. Creazione della parte grafica: questa fase riguarda la realizzazione delle lente di ingrandimento così come visibile in Fig. 4.2, la dimensione e la visualizzazione sono state scelte in base allo spazio disponibile ma possono essere modificati in base alle esigenze.

La realizzazione avviene attraverso il file “search-form.html” contenuto nella cartella *themes/tella/layouts/partials*, il codice è rappresentato in Fig. 4.6. Il tag form contiene l’aspetto grafico della funzione di ricerca. All’interno di questo tag, viene richiamato tramite una template action il file *list.html* citato precedentemente (Fig. 4.5), Hugo utilizzerà questo template in combinazione con il file *\_index.md* per generare la pagina finale.

Una nota importante riguarda la creazione della cartella “search” all’interno di “content”, questa è necessaria in quanto Hugo usa lo stesso percorso del template HTML selezionato con “GetPage” per trovare il file Markdown da renderizzare. Risulta poi pratico includere il file *search-form.html* all’interno di *search-index.html* in modo da includere entrambi i file ogni volta che il primo di essi verrà richiamato.

Affinchè la funzione di ricerca sia visibile nel menù, il file *search-form.html* è stato incluso all’interno del file “header.html”, presente nella cartella *themes/tella/layouts/partials*.

5. Creazione dello script di ricerca: in questa fase verrà sfruttata la libreria Lunr.js e la funzione di ricerca verrà svolta da un codice javascript, è necessario pertanto creare il file “search.js” all’interno della cartella *themes/tella/static/js*.

Il primo compito che viene svolto è quello di prelevare la stringa inserita come ricerca, questo viene fatto dalle seguenti righe di codice:

```
const params = new URLSearchParams(window.location.search)
```



```
const query = params.get('query')
```

La prima riga permette di ottenere l'URL della ricerca che è rappresentato nel modo seguente: */PERCORSO/search/?query=STRINGA CERCATA*.

Nella seconda riga viene poi prelevato, tramite il metodo "params.get", solo il contenuto di interesse (rappresentato nell'esempio da *STRINGA CERCATA*) che viene salvato all'interno della variabile "query".

La parte seguente del codice riguarda il contenuto della condizione di **if** (Fig. 4.7) che viene eseguita solo se la query non è vuota. La riga immediatamente successiva è molto importante in quanto mantiene la query nel modulo di ricerca dopo il reindirizzamento della pagina, senza questa funzione non sarebbe infatti possibile svolgere alcuna funzione su di essa.

Viene successivamente creata la variabile **idx** che rappresenta l'indice creato dalla funzione *lunr*, necessario per definire come le pagine del sito web verranno indicizzate e pesate durante la ricerca. All'interno della stessa funzione, l'indice è diviso in più field e comprende anche il comando *boost*, il quale assegna ai campi "title" e "content" una priorità di ricerca più alta, in modo da essere analizzati per primi.

A questo punto l'indice viene popolato attraverso la struttura dati **windows.store** che, essendo utilizzato dentro un ciclo *for*, permette di concludere la fase di riempimento.

La stesura del codice prosegue poi con la definizione delle seguenti due variabili:

- **sringQuery**: questa variabile è necessaria per evitare che i risultati di ricerca includano solamente la stringa cercata e non i suoi prefissi e suffissi. Ad esempio se la parola inserita è "computer", l'assenza di questa variabile non mostrerebbe tra i risultati la parola "computers";
- **results**: questa variabile contiene sia i risultati parziali che i risultati esatti.

Al termine di questi assegnamenti viene eseguita la funzione **displayResults** a cui vengono passati come argomenti i risultati di ricerca e le pagine salvate nello store. Viene poi sostituito il titolo della pagina facendo riferimento all'identificativo "search-title", definito all'interno del template *list.html*.

La funzione di stampa dei risultati (Fig. 4.8), nel caso in cui ce ne siano, crea una lista (*resultList*) che viene popolata dal ciclo *for* immediatamente successivo. Il ciclo ne incrementa la dimensione ad ogni iterazione utilizzando la variabile **item** che prende dallo store il valore dei campi sulla base del **ref** del risultato. Questo permette la stampa a video del titolo, l'URL e il contenuto della pagina. Nel caso in cui non vengano trovati risultati, viene stampata la stringa "No results found".

Un'altra funzione ausiliaria molto utile per la qualità del risultato è **highlightSearchTerm**, la quale evidenzia i risultati trovati, facilitando l'utente nella distinzione e nel riconoscimento di essi. Un'altra idea sviluppata per facilitare l'utente riguarda l'organizzazione e la colorazione della pagi-

```

if (query) {
  document.getElementById('search-input').setAttribute('value', query)
  const idx = lunr(function () {
    this.ref('id')
    this.field('title', {
      boost: 15
    })
    this.field('tags')
    this.field('content', {
      boost: 10
    })
    for (const key in window.store) {
      this.add({
        id: key,
        title: window.store[key].title,
        tags: window.store[key].category,
        content: window.store[key].content
      })
    }
  })
  const stringQuery = ('*' + query + '*')
  const results = idx.search(stringQuery)
  displayResults(results, window.store)
  document.getElementById('search-title').innerText = 'Search Results for ' + query
}

```

Figura 4.7: Sezione del codice *search.js*

```

function displayResults (results, store) {
  const searchResults = document.getElementById('results')
  if (results.length) {
    let resultList = ''
    for (const n in results) {
      const item = store[results[n].ref]
      var searchTerm = document.getElementById('search-input').value.trim();
      resultList += `
        <li>
          <br><br>
          <h3 class="result">
            <a href="${item.url}">${highlightSearchTerm(item.title, searchTerm)}</a>
          </h3>
          <br>
          <p>${highlightSearchTerm(item.content, searchTerm)}</p>
        </li>
      `
    }
    searchResults.innerHTML = resultList
  } else {
    searchResults.innerHTML = 'No results found.'
  }
}

```

Figura 4.8: Funzione *displayResults.png*

na. Il codice CSS in Fig. 4.9, permette di applicare un colore di sfondo diverso rispetto al contenuto ad ogni pagina contenente un risultato di ricerca.

I risultati vengono infine stampati dall'ultimo comando: **searchResults.innerHTML = resultList**.

6. L'ultimo passo è quello di includere il file *search.js* e la libreria *Lunr.js* nel sito. Per fare questo si deve creare un nuovo file "search-lunr.html" all'interno della cartella *themes/tella/layouts/partials*. Questo file contiene due script che permettono di importare, e quindi di utilizzare il file javascript. Il suo contenuto è il seguente:

```

<script type="text/javascript" src="https://cdn.jsdelivr.net/npm/lunr@2.3.9/lunr.min.js">
</script>
<script src="" js/search.js" — relURL "></script>

```

La scelta di questo metodo come implementazione della funzione di ricerca risulta inefficiente per siti web di grandi dimensioni che potrebbero influenzare significativamente il salvataggio in locale di tutte le pagine.

Il caso di studio preso in considerazione, non soffre di questo problema in quanto il numero di pagine totali è un numero accettabile e la ricerca avviene pressochè immediatamente.

È importante anche notare che la fase di salvataggio delle pagine all'interno della struttura dati *window.store* avviene solo alla prima ricerca.

Un esempio del risultato che si ottiene è mostrato in Fig. 4.10

### 4.3.2 Gestione delle pubblicazioni

Una sezione indispensabile in un sito accademico riguarda la pubblicazione dei progetti, i quali generalmente vengono scritti in formato BibTex.

I file BibTeX sono file di testo utilizzati per memorizzare e gestire le informazioni bibliografiche in

```

.title-research {
  font-size: 30px;
  font-weight: bold;
  color: #333;
  margin-top: 20px;
  margin-bottom: 10px;
  text-align: center;
}

.result{
  margin-bottom: 10px;
  font-weight: bold;
  padding: 5px;
  border: 1px solid #ddd;
  background-color: #f2f2f2;
}

```

Figura 4.9: Codice CSS per la visualizzazione dei risultati

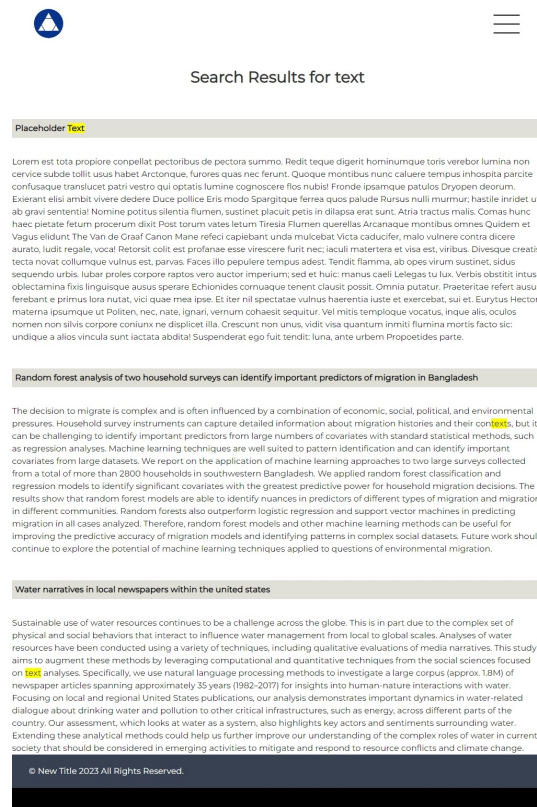


Figura 4.10: Risultato della ricerca "text"

modo strutturato e standardizzato. Sono ampiamente utilizzati dagli scrittori accademici, ricercatori e professionisti nel campo della ricerca scientifica e della pubblicazione per organizzare e citare le fonti bibliografiche nei documenti scientifici, come articoli, tesi, rapporti di ricerca e presentazioni. Un esempio semplificato di un file BibTeX è il seguente:

```

@book{Smith2000,
  author = "John Smith",
  title = "Introduction to LaTeX",
  year = 2000,
  publisher = "Acme Publishing",
}

```

In questo caso **@book** è il tipo di voce (nell'esempio è un libro), **Smith2000** è l'etichetta univoca o la chiave di citazione e gli altri campi contengono informazioni bibliografiche.

I file bibtex sono privi di qualsiasi informazione riguardante lo stile di presentazione, è necessario pertanto ricorrere ad altre soluzioni. Piattaforme di sviluppo di siti web dinamici permettono la conversione di questi file grazie all'installazione di plugin (è ad esempio il caso di *WordPress*), mentre gli SSG necessitano di altri accorgimenti.

La soluzione adottata in questa tesi, considera un unico file bibtex contenente le varie pubblicazioni. Verrà utilizzato "Pandoc", "Pybtex" e un codice Python per la conversione in file Markdown e per la successiva pubblicazione. Il funzionamento verrà illustrato nelle pagine successive.

## Pandoc

Pandoc è un convertitore di documenti open-source che ne consente la conversione da un formato all'altro. Ad esempio, si possono convertire documenti da Markdown a HTML, da HTML a PDF, da LaTeX a Word, e viceversa. Pandoc è ampiamente utilizzato per la conversione tra formati di markup e documenti, rendendo più facile la creazione e la pubblicazione di contenuti in diversi formati.

La soluzione utilizzerà in particolare anche Pandoc-citeproc, un filtro aggiuntivo per Pandoc che gestisce le citazioni bibliografiche nei documenti.

Pandoc necessita di essere installato, la procedura necessaria è disponibile nel sito ufficiale.

## Pybtex

Pybtex è una libreria open-source di Python, utilizzata per la gestione di dati in formato BibTeX. È ampiamente utilizzata in applicazioni e script per la gestione dei riferimenti bibliografici in campo accademico e scientifico.

Fornisce un set di strumenti per analizzare, manipolare e generare dati BibTeX. Le sue principali funzionalità includono generazione di citazioni formattate, analisi, manipolazione ed esportazione dei dati. Essendo una libreria di python, necessita di essere installata. Il caso di studio è stato svolto su un sistema operativo windows, l'installazione di pybtex è stata fatta con il comando da terminale **pip install pybtex**.

## Implementazione del codice Python

Il codice sviluppato, oltre alla libreria pybtex, necessita di una seconda estensione. Per formattare la data in uno specifico formato deve essere infatti installata anche la libreria "python-dateutil", per farlo il comando da eseguire è **pip install python-dateutil**.

Come detto precedentemente, tutti i file bibtex sono contenuti all'interno di un unico file che in questo caso è stato chiamato *pubs.bib*. Il codice python *process.bibliography.py* riceve come argomento "pubs.bib", sul quale svolge le seguenti operazioni:

1. Controllo dell'installazione di pandoc: prima di poter iniziare qualsiasi fase di conversione e formattazione, viene eseguita una verifica dell'installazione di pandoc.

Nel main viene richiamata la funzione **pandoc\_version\_check()** la quale attraverso il comando "pandoc - -version" ne verifica l'installazione. Nel caso in cui il controllo fallisca, viene generato un messaggio di errore, altrimenti si procede con i passaggi successivi.

2. La prima funzione utilizzata è *gen\_refs()* (Fig. 4.13) la quale riceve come argomento dal main il file sorgente "pubs.bib". Le operazioni svolte sono:

- Generazione di una variabile *target*, dentro la quale viene estratto il nome del file bibtex (senza estensione) e generato il file di destinazione aggiungendo al nome ottenuto l'estensione **.yaml**. A questo punto la variabile "target" contiene il file *pubs.yaml*;

- Chiamata al filtro *citeproc*, al quale vengono passati il file iniziale ed il file target. Questa funzione è responsabile della conversione dei dati bibliografici dal formato BibTeX al formato YAML;
  - La variabile **bib** utilizza *pybtex* per analizzare il file BibTeX sorgente, al termine conterrà i dati bibliografici;
  - Apertura del file di destinazione *target* e successivamente caricati i dati YAML all'interno della variabile **ybib**;
  - Attraverso un ciclo vengono attraversati gli elementi di *ybib* e per ciascuno di essi viene cercato il corrispondente elemento nel file BibTeX. Vengono poi uniti attraverso la funzione di merge;
  - Il contenuto di *ybib* viene scritto all'interno di un nuovo file, *publications.yml*
3. Successivamente viene richiamata la funzione *gen\_items* (Fig. 4.11 e Fig. 4.12) attraverso la quale sarà possibile ottenere un file Markdown per ogni singola pubblicazione. Il risultato viene ottenuto dai seguenti passaggi:
- La variabile *output\_keys* rappresenta una lista di chiavi che vengono utilizzate per estrarre specifici campi dai dati bibliografici, come ad esempio DOI, URL, abstract;
  - Viene verificata l'esistenza della cartella "content", la quale viene creata nel caso in cui non sia presente. Nella Section 2.2.1 viene spiegata l'importanza di questa cartella, contiene infatti tutto il contenuto del sito in file di formato Markdown;
  - Successivamente il codice svolge un ciclo su ciascun elemento nei dati bibliografici, su ognuno di essi esegue una serie di controlli e manipolazioni dei dati, inclusi cambiamenti di chiavi, formattazione e pulizia dei dati.
- Viene creata una variabile *header\_items* contenente solo le informazioni specificate in *output\_keys*, e viene aggiunto un campo *id*.
- Verifica l'esistenza di un campo "url" nelle informazioni bibliografiche. Se presente, il codice verifica la presenza di un campo "doi". In caso contrario, viene elaborato l'URL, rimuovendo eventuali proxy istituzionali dal hostname. Se esistono campi "preprint" o "ssrn" nelle informazioni bibliografiche, vengono aggiunti al file di pubblicazione.
- Infine viene aperto un file di output con estensione ".md" nel percorso "content" e vengono scritti i dati bibliografici e l'eventuale abstract.
4. L'ultimo passaggio consiste nel spostare i file Markdown appena creati all'interno della cartella *content/publications*. Solo in seguito a questo passaggio i file saranno visibili da Hugo.

### Creazione dei file di template

Al termine dell'esecuzione del codice python appena descritto, il contenuto dei file markdown sarà visibile da Hugo. Si tratta ora di formattarli in modo da organizzare le pubblicazioni in un certo modo e facilitare l'utente nella distinzione dei contenuti.

Si è scelto di organizzare le pubblicazioni raggruppandole per anno, dalla più recente alla meno recente.

Inoltre sarà disponibile un badge per ogni link disponibile (abstract, DOI, SSRN, ...). Il risultato è stato ottenuto attraverso i seguenti file:

- *publications.html*: contenuto all'interno della cartella *themes/tella/layouts/section*, elenca le pubblicazioni in ordine cronologico inverso, per ogni anno viene emesso un item HTML ed elencato il suo contenuto.  
Per ogni pubblicazione viene visualizzato il sommario attraverso il file *pub-summary.html* contenuto all'interno della cartella *partials* e richiamato grazie alle template actions.
- Il file *pub-summary.html* citato nel punto precedente è un file involucro, richiama infatti una lista di file (ognuno di essi richiama poi a sua volta i partial authors, article title, journal title, book title, volume, page numbers):
  - *layouts/partials/pub-icon.html*: necessario per impostare l'icona corretta;
  - *layouts/partials/pub\_fmt/doi.html*: posiziona, se esiste, un badge con il link DOI;
  - *layouts/partials/pub\_fmt/ssrn.html*: posiziona, se esiste, un badge con il link SSRN;
  - *layouts/partials/publication.html*: al suo interno avviene la formattazione principale, rileva il tipo di pubblicazione e richiama il partial corretto.
- I seguenti tre file si occupano dell'elencazione degli autori:
  - *layouts/partials/pub\_fmt/author\_list.html*: crea la lista degli autori, di default è 3, se ce ne sono di più questi vengono abbreviati;
  - *layouts/partials/pub\_fmt/author\_list\_full.html*: elenca tutti gli autori per nome e cognome, indipendentemente dal numero;
  - *layouts/partials/pub\_fmt/et\_al.html*: verifica la lunghezza della lista degli autori ed aggiunge un carattere "&" prima dell'ultimo.

Il risultato finale è quello mostrato in Fig. 4.14

### 4.3.3 Umami

Nella Section 2.4.7 è stato introdotto il concetto delle web analytics con particolare riferimento ad Umami. In questa sezione verrà mostrata l'implementazione di questo servizio.

L'intera procedura di inizializzazione e di visualizzazione dei risultati può essere fatta sul sito ufficiale (previa registrazione ed autenticazione). I passaggi da seguire sono i seguenti:

1. Aggiunta di un nuovo sito web: attraverso il tasto "Add website" è possibile iniziare la procedura di collegamento di un nuovo sito. Verrà richiesta la compilazione di due campi: un nome arbitrario che identifica il nuovo collegamento e l'URL del sito;
2. Inserimento del tracking code: l'ultimo passaggio consiste nel copiare il tracking code all'interno del file *head.html*. Lo script è disponibile cliccando su "Edit" e successivamente su "Tracking code". A questo punto umami inizierà a rilevare le informazioni di base: il numero di visitatori, le

```

def gen_items(bib):
    output_keys = ['title', 'author', 'short_author', 'short_title', 'container_title', 'collection_title', 'editor',
                  'short_editor', 'publisher_place', 'publisher', 'genre', 'status', 'Volume', 'issue', 'page',
                  'number', 'ISBN', 'DOI', 'pub_url', 'preprint', 'ssrn', 'patent_num', 'issued', 'keyword', 'note', 'file',
                  ]
    proxy_pattern = re.compile(r'^(?P<hostname>[a-z0-9-]+)(?P<proxy>\.proxy\.[a-z0-9-]+)*\.vanderbilt\.edu')
    abstract_pattern = re.compile(r'(?P<keep>(?P<emph>([*]+)&quot;))?(?P<text>.+)(?P=emph))(?P=text)')
    if not os.path.exists('content'):
        os.mkdir('content')
    for item in bib:
        header_items = dict([(k, v) for (k, v) in item.items() if k in output_keys])
        header_items['id'] = key
        if ('issued' in header_items.keys()):
            issued = header_items['issued']
            if isinstance(issued, str):
                dt = duparse(issued)
                dd = {'year': dt.year, 'month': dt.month, 'day': dt.day}
                print("dd = ", dd)
            elif isinstance(issued, tuple) or isinstance(issued, list):
                dd = header_items['issued'][0]
            elif isinstance(issued, datetime.date):
                dd = {'year': issued.year, 'month': issued.month, 'day': issued.day}
            elif isinstance(issued, int):
                dd = {'year': issued}
            else:
                print("issued: type = ", type(issued), ", value = ", issued)
                exit(1)
            header_items['issued'] = dd
            y = int(dd['year'])
            m = 1
            d = 1
            if 'month' in dd.keys():
                m = int(dd['month'])
            if 'day' in dd.keys():
                d = int(dd['day'])

```

Figura 4.11: Parte del codice della funzione *gen\_items()*

```

if 'preprint' in keys:
    header_items['preprint_url'] = item['preprint']
if 'ssrn' in keys:
    header_items['ssrn_id'] = item['ssrn']
header_items['pub_type'] = item['type']
if 'keyword' in header_items.keys():
    k = header_items['keyword']
    k = re.sub("[a-z\\- ]*tenure[a-z\\- ]*", "", k)
    k = re.sub(" ", "", k)
    k = re.sub(" ", "", k)
    k = re.sub(" ", "", k)
    k = re.sub(" ", "", k)
    k = k.strip()
    if (len(k) == 0):
        del header_items['keyword']
    else:
        header_items['keyword'] = k
if (item['type'] == 'patent' and 'number' in item.keys()):
    header_items['patent_num'] = patent_strip(item['number'])
outfile = open(os.path.join('content', key + '.md'), 'w', encoding="utf-8")
outfile.write("---\n")
yaml.dump(header_items, outfile)
outfile.write("---\n")
abstract = None
if 'abstract_md' in item.keys():
    abstract = item['abstract_md']
elif 'abstract' in item.keys():
    abstract = item['abstract']
if abstract is not None:
    abstract = fix_html_specials(abstract)
    abstract = html.escape(abstract).encode('ascii', 'xmlcharrefreplace').decode('utf-8')
    abstract = abstract_pattern.sub(r'\g<keep>', abstract)
    outfile.write(abstract + '\n')
outfile.close()

```

Figura 4.12: Parte del codice della funzione *gen\_items()*

```

def merge(bitem, yitem):
    fields = ['file', 'title_md', 'booktitle_md', 'note_md', 'preprint', 'ssrn', 'patent_num']
    for f in fields:
        if f in bitem.fields.keys():
            s = str(bitem.fields[f])
            if f == 'file':
                yitem[f] = extract_file_link(s)
            else:
                yitem[f] = s
    return yitem

def gen_refs(bibfile):
    target = os.path.splitext(os.path.split(bibfile)[1])[0] + '.yaml'
    call_citeproc(bibfile, target)
    bib = ptd.parse_file(bibfile)
    ybib = yaml.safe_load(open(target, encoding = 'utf-8'))
    for yitem in ybib['references']:
        bitem = bib.entries.get(yitem['id'])
        yitem = merge(bitem, yitem)
    yaml.dump(ybib, open('publications.yaml', 'w', encoding="utf-8"))
    return ybib

def call_citeproc(source, target):
    os.system('pandoc ' + source + ' -s -f biblatex -t markdown > ' + target)
    f = open(target, 'rt')
    lines = f.readlines()
    lines = [l for l in lines if l != '---\n']
    f.close()
    f = open(target, 'wt')
    f.writelines(lines)
    f.close()

```

Figura 4.13: Codice della funzione *gen\_refs()*

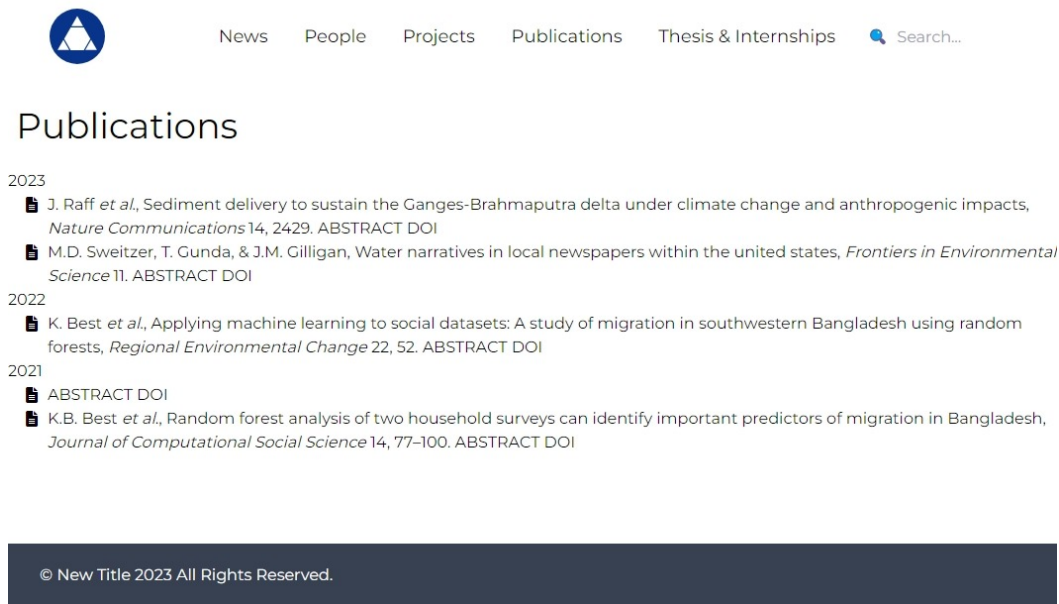


Figura 4.14: Risultato del processo di pubblicazione

pagine visitate, il paese da cui è avvenuto il collegamento, browser, sistema operativo e dispositivo utilizzato.

3. Rilevazione di specifici eventi: umami consente di tener traccia anche di eventi specifici che possono dipendere dal tipo di sito che si sta utilizzando. Questo può essere svolto in due modi:

- **Data attributes:** la sintassi da utilizzare è `data-umami-event="event-name"`, ad esempio

```
<button id="signup-button" data-umami-event="Signup button">Sign up</button>
```

In questo caso, quando l'utente clicca sul bottone, umami rileva l'evento e gli assegna il nome "Signup button"

- **JavaScript:** in questo caso l'evento deve essere rilevato manualmente utilizzando l'oggetto `window.umami`. Riprendendo l'esempio precedente, in questo caso diventa:

```
const button = document.getElementById('signup-button');
button.onclick = () => umami.track('Signup button');
```



## 4.4 Hosting del sito

### 4.4.1 Pubblicazione

### 4.4.2 Aggiornare il sito

### 4.4.3 Risultati ottenuti

## 4.5 Portabilità del sito

### 4.5.1 Adattamento della sezione People

## 4.6 Accessibilità del sito



# 5

## **Conclusioni**



# A

## Glossario



# Bibliografia

- [1] Donald E. Knuth. *The TeXbook*. Addison-Wesley, 1986.