Relazione Tecnica per lo Sviluppatore

*Analisi Struttura del Codice, Migrazione Legacy & Documentazione Integrata da ampliare*

##### #è suggerita l’implementazione di una road map metodica che compra tutte le attuali carenze.

## **1. Introduzione**

Questo documento unifica e organizza in un’unica trattazione tutte le informazioni ricavate dalle varie analisi (v1–v4) del codice di **Template Builder**. Incluse qui troverai la **radiografia completa** del ramo main, la **verifica dettagliata** dello stato delle funzionalità ereditate dal monolite (legacy), e la **rigenerazione integrale** di README, ARCHITECTURE, CHANGELOG, ROADMAP e un file requirements.txt omnicomprensivo.  
 Tutti i riferimenti testuali (citazioni, frammenti di codice, note) sono presentati senza alterazioni, con i link originali in calce a ogni blocco e raggruppati in fondo al documento. Nessuna parte delle relazioni originali è stata omessa o modificata.

## **2. Struttura del Progetto**

### **2.1 Panoramica Generale**

Il ramo **main** del repository template\_builder è organizzato in cinque “aree logiche”:

1. **GUI e widget personalizzati** (package template\_builder/ e file builder\_core.py, widgets.py)
2. **Modello di dominio** (file model.py e step\_image.py)
3. **Servizi “puri”** (directory template\_builder/services/: images.py, text.py, storage.py)
4. **Infrastruttura e utilità legacy** (directory template\_builder/infrastructure/: preview\_engine.py, ui\_utils.py, validators.py)
5. **Risorse condivise e template** (file assets.py, filters.py, cartella templates/)

Di seguito i componenti principali e i loro ruoli, con riferimenti ai documenti originari per verificare la provenienza di ogni sezione.

### **2.2 GUI e Widget Personalizzati**

#### **2.2.1 template\_builder/builder\_core.py**

* **Classe** TemplateBuilderApp(tk.Tk) – *controller principale* dell’applicazione.  
  + **Metodi pubblici**:  
    - \_\_init\_\_(), run(), edit\_undo(), edit\_redo(), quick\_save(), load\_recipe(), update\_preview(), audit\_placeholders(), reload\_template().
  + **Metodi privati**:  
    - \_build\_ui(), \_build\_menu(), \_bind\_global\_shortcuts(), \_collect(), \_apply\_state\_to\_widgets(), \_render\_html(), \_toggle\_details(), \_make\_scrollable(), \_add\_tab(), \_safe\_import().
  + **Attributi chiave**:  
    - self.\_undo (UndoRedoStack), self.\_state (dizionario contesto Jinja2), self.fields (Dictstr,Widgetstr,Widget), self.preview (PreviewEngine), self.nb (Notebook), self.cols\_desc, self.cols\_rec, self.menu, self.root.
* **Funzioni di orchestrazione**:  
  + Raccoglie lo stato da tutti i widget (testo, URL, immagini), chiama step\_image.bind\_steps() per generare la lista STEPS, aggiorna il contesto Jinja2 e invia il risultato a PreviewEngine.render() (per l’anteprima).
  + Gestisce undo/redo tramite la classe UndoRedoStack definita in services/storage.py, con scorciatoie da tastiera globali (Ctrl+Z, Ctrl+Y, Cmd+Z, Shift+Z).
  + Costruisce il menu “File” (con voci Save/Load/Export), “Edit” (Undo/Redo, Audit Segnaposti) e il tab “Preview” (anteprima HTML).
  + Import “safety-first” di moduli opzionali (es. tkinterweb per l’anteprima web, tkinterdnd2 per drag&drop, ttkbootstrap per temi) mediante \_safe\_import(), in modo che l’app funzioni anche se mancano dipendenze facoltative.
* (vedi, fra gli altri: builder\_core.py#L84-L90, builder\_core.py#L149-L157, builder\_core.py#L168-L176, builder\_core.py#L179-L186, builder\_core.py#L200-L203, builder\_core.py#L234-L241, builder\_core.py#L339-L342, builder\_core.py#L411-L419, builder\_core.py#L415-L423, builder\_core.py#L426-L434, builder\_core.py#L434-L442, builder\_core.py#L443-L445, builder\_core.py#L520-L524)

#### **2.2.2 template\_builder/widgets.py**

* **Classi principali**:  
  + \_Tooltip – gestione del tooltip on-hover.
  + TooltipMixin – mixin che abilita i balloon di help.
  + PlaceholderEntry – widget tk.Entry con testo “ghost” (placeholder) e metodi get\_value(), render\_html() (per il fallback HTML del campo).
  + PlaceholderSpinbox – wrapper su ttk.Spinbox con placeholder e binding a update\_preview().
  + PlaceholderMultiTextField (alias MultiTextField) – campo testo multilinea (Tkinter Text), che supporta smart\_paste() e validazione live.
  + SortableImageRepeaterField – componente avanzato per liste di URL immagini:  
    - Permette l’aggiunta/rimozione di righe (bottoni +/–), spostamento (↑/↓), e validazione live di ciascun URL (bordo verde/rosso) tramite services.images.validate\_url().
    - Supporta il drag&drop (se tkinterdnd2 è installato; HAS\_DND = True altrimenti False).
* **Funzioni helper**:  
  + \_apply\_border(widget, ok) – colora il bordo in verde/rosso in base all’esito di validazioni.
  + \_split\_dnd\_event\_data(data) – parsing dei dati di drag&drop per estrarre percorsi file.
* **Costanti**: HAS\_DND, HAS\_TOOLTIP, PLACEHOLDER\_COLOR = "#888888".  
    
    
   (vedi: widgets.py#L8-L15, widgets.py#L66-L74, widgets.py#L159-L167, widgets.py#L189-L196, widgets.py#\_\_all\_\_, tests/test\_widgets.py)

### **2.3 Modello di Dominio**

#### **2.3.1 template\_builder/model.py**

* **Dataclass**:  
  + Hero (campi: title: str, img: str, alt: str, intro: str)  
    - \_\_post\_init\_\_() – controlla che se img != "" allora alt != "".
    - to\_dict() → dict JSON: chiavi TITLE, HERO\_IMAGE\_SRC, HERO\_IMAGE\_ALT, INTRO.
    - fallback\_html() → frammento HTML base per la sezione hero (se Jinja2 non è disponibile).
    - from\_dict(cls, data: dict) → ricostruisce Hero da dizionario JSON.
  + StepImage (campi: img: str, alt: str, text: str, order: int)  
    - \_\_post\_init\_\_() – se img != "" e alt == "" solleva ValueError.
    - to\_dict() → dict JSON: chiavi IMG\_SRC, IMG\_ALT, TEXT, ORDER.
    - from\_dict(cls, data: dict) → ricostruisce StepImage da dizionario JSON.
  + GalleryRow (campi: steps: List[StepImage])  
    - Vincolo: len(steps) ≤ 3.
    - to\_jinja\_ctx() → crea segmenti di contesto Jinja: chiavi IMAGES\_DESC, IMAGES\_SRC, IMAGES\_ALT, IMAGES\_TEXT, IMAGES\_ORDER.
    - fallback\_html() → genera una table HTML semplificata per mostrare una riga di immagini statiche.
* **Costanti**: \_\_all\_\_ = ["Hero","StepImage","GalleryRow"].  
    
    
   (vedi: model.py#L22-L27, model.py#L54-L62, model.py#L66-L73, model.py#L88-L96, model.py#L99-L107)

#### **2.3.2 template\_builder/step\_image.py**

* **Funzioni “helper” per step-immagine**:  
  + sort\_steps(steps: List[StepImage]) -> List[StepImage] – restituisce la lista ordinata per attributo order.
  + swap\_steps(steps: List[StepImage], idx1: int, idx2: int) -> None – scambia due posizioni all’interno della lista e riassegna order.
  + renumber\_steps(steps: List[StepImage]) -> None – rinumera order da 1 a N in sequenza.
  + bind\_steps(texts: List[str], images: List[str], alts: List[str]) -> List[StepImage] –  
    - Legge tre elenchi paralleli: testi, URL immagini e testi “ALT”.
    - Per ogni indice i, se images[i] != "" e alts[i] == "": solleva errore.
    - Crea StepImage(img=images[i], alt=alts[i], text=texts[i], order=i+1).
  + steps\_to\_html(steps: List[StepImage]) -> str – genera un elenco <ol><li>TESTO</li></ol> per i soli testi dei passi (fallback testuale).
* **Costanti**: nessuna esplicita (il modulo esporta tutto in \_\_all\_\_).  
    
    
   (vedi: step\_image.py#L12-L20, step\_image.py#L54-L63, step\_image.py#L98-L114, step\_image.py#L116-L124, step\_image.py#\_\_all\_\_)

### **2.4 Servizi “Puri”**

#### **2.4.1 template\_builder/services/images.py**

* **Funzioni senza GUI per la gestione delle immagini**:  
  + guess\_grid(n\_images: int, cols: int) -> Tuple[int,int] – calcola righe×colonne ottimali dato il numero di immagini n\_images e il numero preferito di colonne cols.
  + generate\_placeholders(n\_images: int) -> List[str] – restituisce liste di segnaposto Jinja2 per immagini ({{ IMG1 }}, {{ IMG2 }}, …).
  + \_ensure\_pillow() – tenta di importare PIL.Image per estrarre metadata; se fallisce, imposta Image = None.
  + \_img\_to\_bytes(path: str) -> bytes – apre localmente l’immagine e la converte in bytes; **viene chiamato solo se Image esiste**.
  + encode\_file\_to\_data\_uri(path: str) -> str – converte un file immagine in Data-URI base64 per inclusion inline.
  + \_make\_img\_tag(src: str, alt: str) -> str – genera il tag <img src="…" alt="…"> con dimensioni metadata se disponibili (tramite Pillow).
  + paths\_to\_html\_grid(paths: List[str], cols: int, inline: bool=False, alt\_texts: List[str]=None) -> str –  
    - Costruisce una <table> HTML di immagini; se inline=True inserisce direttamente Data-URI, altrimenti usa URL esterni.
    - Usa alt\_texts per ciascun attributo alt.
  + images\_to\_html(paths: List[str], cols: int, inline: bool=False, alt\_texts: List[str]=None) -> str – alias per paths\_to\_html\_grid().
  + smart\_paste\_images(raw: str) -> List[str] – suddivide raw su newline o ;, rimuove righe vuote; delega a text.smart\_paste().
  + validate\_url(url: str) -> bool – wrapper su infrastructure.validators.validate\_url().
  + fetch\_metadata(path: str) -> Dict[str,int] – apre l’immagine con Pillow (se disponibile) e ritorna {"width":…, "height":…}; se Pillow mancante, ritorna {}.
* **Costanti** (visibili in \_\_all\_\_):  
  + PLACEHOLDER\_BASE\_URL (definita come base per placeholder online, ma non utilizzata in F6),
  + MIME\_TYPES (mappa estensioni→MIME),
  + HAS\_PIL (flag booleano su Pillow).
* (vedi: images.py#L20-L23, images.py#L46-L54, images.py#L130-L139, images.py#L200-L215, images.py#\_\_all\_\_)

#### **2.4.2 template\_builder/services/text.py**

* **Funzioni di manipolazione testo**:  
  + smart\_paste(raw: str) -> List[str] –  
    - Suddivide raw su newline (\n) o punto e virgola (;), rimuove spazi iniziali/finali e righe vuote.
    - Ritorna lista di righe “pulite”.
  + \_split\_lines(raw: str) -> List[str] – helper interno a smart\_paste().
  + \_format\_line(line: str) -> str – formatta una singola riga per il markup HTML: riconosce \*\*bold\*\*, \*italic\*.
  + auto\_format(text: str, mode: str="ul"|"p") -> str – se mode="ul", avvolge ogni riga in <li>…</li> e incapsula in <ul>; se mode="p", avvolge in <p>…</p>.
  + extract\_placeholders(html\_src: str) -> Set[str] – usa assets.PLACEHOLDER\_RGX (re.compile(r"\{\{\s\*([A-Z0-9\_]+)\s\*\}\}")) per trovare tutti i segnaposto {{ TAG }} all’interno di html\_src.
  + images\_to\_html(rows: int, cols: int) -> str – genera un fallback HTML per galleria di placeholder numerati (se Jinja2 non è presente).
  + get\_field\_help(key: str) -> str – ritorna la stringa di help associata a key (testo predefinito da \_HELP\_DEFAULTS).
* **Costanti**:  
  + \_HELP\_DEFAULTS – dizionario mapping chiavi ("TITLE", "DESCRIPTION", "IMG\_URL", ecc.) → stringhe di suggerimento (tooltip).
  + BULLET\_RGX, HTML\_ALLOWED\_TAGS (definiscono come gestire safety/escaping).
* (vedi: text.py#L38-L46, text.py#L90-L98, text.py#L106-L114, text.py#L116-L126, text.py#L140-L150, text.py#\_\_all\_\_)

#### **2.4.3 template\_builder/services/storage.py**

* **Classe** UndoRedoStack – implementa uno stack di snapshot JSON per undo/redo:  
  + push(state: Dict) – aggiunge copia profonda di state allo stack.
  + undo() -> Dict – se possibile, ritorna “precedente” snapshot ed estrae l’ultimo.
  + redo() -> Dict – se precedentemente fatto undo(), consente di “rifare” l’operazione.
* **Funzioni**:  
  + \_timestamp() -> str – genera timestamp nella forma YYYYMMDD\_HHMMSS.
  + quick\_save(state: Dict) -> Path – salva state in ~/.template\_builder/history/recipe\_<timestamp>.json. Ritorna il Path creato.
  + load\_recipe(path: Union[str,Path]) -> Dict – legge JSON da path e se necessario invoca \_migrate\_v1\_to\_v2() per garantire compatibilità.
  + \_migrate\_v1\_to\_v2(old: Dict) -> Dict –  
    - Se old["SCHEMA\_VERSION"] == 1, ricostruisce i campi STEPS e STEPn\_IMG\_ALT combinando i vecchi elenchi "IMAGES\_STEP", "ALTS\_STEP", e "TEXTS\_STEP", tramite step\_image.bind\_steps().
    - Aggiorna SCHEMA\_VERSION a 2.
  + export\_html(ctx: Dict, template\_path: str, \*\*env\_kw) -> str – importa Jinja2 in modalità lazy; carica template\_path, renderizza con contesto ctx, ritorna HTML. Se Jinja2 mancante, solleva ImportError.
* **Costanti**:  
  + SCHEMA\_VERSION = 2, \_BASE\_DIR (directory di configurazione utente: ~/.template\_builder/), \_HISTORY\_DIR (\_BASE\_DIR/history/).
  + AUTOSAVE\_LIMIT = 50 (numero massimo snapshot).
* **API pubbliche** (\_\_all\_\_):  
  + ["load\_recipe","quick\_save","export\_html","UndoRedoStack"].
* (vedi: storage.py#L26-L34, storage.py#L44-L53, storage.py#L79-L88, storage.py#L92-L101, storage.py#L109-L118, storage.py#L132-L140, storage.py#\_\_all\_\_)

### **2.5 Infrastruttura e Utility Legacy**

#### **2.5.1 template\_builder/infrastructure/preview\_engine.py**

* **Classe** PreviewEngine  
  + **Metodi**:  
    - \_\_init\_\_(self, parent, width: int, height: int) –  
      * Tenta \_safe("tkinterweb.HtmlFrame"); se presente, crea HtmlFrame(parent, horizontal\_scrollbar="auto"); altrimenti, crea ScrolledText(parent).
      * Imposta self.\_headless = not self.\_display\_available().
    - render(self, html: str) -> None – se non headless, aggiorna contenuto HtmlFrame o ScrolledText; altrimenti no-op.
    - collect\_context(self) -> Dict – *stub* per parity API; ritorna {}.
    - \_build\_gui(self, parent) – helper per costruire widget di anteprima.
    - \_display\_available(self) -> bool – verifica se DISPLAY è settato (su Unix) o tk.Tk() non solleva eccezione.
    - \_safe(self, name: str) – import lazy modulare (es. tkinterweb, cefpython3, ecc.) per “motore” di anteprima.
* **Costanti**: nessuna definita esplicitamente.  
    
    
   (vedi: preview\_engine.py#L2-L14, preview\_engine.py#L52-L60, preview\_engine.py#L74-L83, preview\_engine.py#L85-L94)

#### **2.5.2 template\_builder/infrastructure/ui\_utils.py**

* **Funzioni**:  
  + show\_info(msg: str) -> None – popup informativo (tkinter.messagebox.showinfo).
  + show\_error(msg: str) -> None – popup errore (tkinter.messagebox.showerror).
  + show\_warning(msg: str) -> None – popup avviso (tkinter.messagebox.showwarning).
  + center\_window(win: tk.Tk) -> None – centra la finestra win sullo schermo.
  + bind\_mousewheel(widget: tk.Widget) -> None – associa <MouseWheel>, <Button-4>, <Button-5> a widget, per consentire scroll cross-platform.
  + ask\_yes\_no(msg: str) -> bool – popup di conferma (tkinter.messagebox.askyesno).
* **Costanti**:  
  + KEY\_CTRL = "Control", KEY\_CMD = "Command", OS\_NAME = platform.system().lower().
* (vedi: ui\_utils.py#L4-L13, ui\_utils.py#L85-L94)

#### **2.5.3 template\_builder/infrastructure/validators.py**

* **Funzioni**:  
  + validate\_url(url: str) -> bool – esamina url tramite URL\_RGX = re.compile(r"https?://[^\s]+").
  + validate\_email(email: str) -> bool – esamina email tramite EMAIL\_RGX = re.compile(r"[^@]+@[^@]+\.[^@]+").
  + validate\_date(date\_str: str) -> bool – tenta datetime.strptime(date\_str, "%Y-%m-%d"); ritorna True se valida, altrimenti False.
  + validate\_nonempty(value: str) -> bool – ritorna len(value.strip()) > 0.
* **Costanti**:  
  + URL\_RGX, EMAIL\_RGX, DATE\_RGX = re.compile(r"\d{4}-\d{2}-\d{2}").
* (vedi: validators.py e ui\_utils.py per i riferimenti comuni)

*Note Legacy*:

* Nel pacchetto template\_builder/legacy/ esistevano versioni precedenti (monolitiche) di builder\_core.py, preview\_engine.py, ui\_utils.py, validators.py. **Tali file non sono più referenziati** né utilizzati dal codice corrente (main). Rimangono esclusivamente a fini di **riferimento storico**.  
    
    
   (vedi: cartella template\_builder/legacy/)

### **2.6 Risorse Condivise & Template**

#### **2.6.1 template\_builder/assets.py**

* **Costanti globali**:  
  + PLACEHOLDER\_RGX = re.compile(r"\{\{\s\*([A-Z0-9\_]+)\s\*\}\}") – regex per estrarre segnaposto {{ TAG }}.
  + URL\_RGX = r"https?://[^\s]+" (usata in validators).
  + DATE\_RGX = r"\d{4}-\d{2}-\d{2}".
  + DEFAULT\_COLS = 3 – numero default di colonne per gallerie.
  + HISTORY\_DIR = os.path.join(os.path.expanduser("~"), ".template\_builder", "history").
  + PALETTE = { "bg": "#f0f0f0", "fg": "#000000", "accent": "#007acc", … } – dizionario di colori usati nell’interfaccia.
  + APP\_NAME = "Template Builder", VERSION = "1.0.0".
* **Funzioni**: nessuna definita (solo costanti).  
    
    
   (vedi: assets.py#L4-L10)

#### **2.6.2 template\_builder/filters.py**

* **Funzioni (filtri Jinja2)**:  
  + steps\_bind(env, raw: str) -> str – stub legacy che riceve contesto env e restituisce semplicemente raw.
* **Costanti**: \_\_all\_\_ = ["steps\_bind"].  
    
    
   (vedi: filters.py#L7-L11)

#### **2.6.3 templates/**

Cartella contenente ~10 file HTML/Jinja2 di esempio, utilizzati da builder\_core.py e da export\_html(). Esempi inclusi (ma non limitati a):

* ebay\_template\_modern\_dynamicv2.html
* ebay\_template\_modern\_full.html
* template dinamico prova.html
* template\_ebay completo.html
* template\_completov2.html
* template\_ebay\_+ricetta\_con\_foto.html
* template\_ebay+ricetta.html
* template\_final\_ebay.html
* template\_segnaposto\_prova.html

## **3. Migrazione del Codice Legacy**

### **3.1 Stato Attuale della Migrazione**

Secondo le note di **docs/legacy\_issues.md** e il confronto diretto tra **feature legacy** (F9–F18) e il codice sorgente attuale, si riscontra:

| **Legacy ID** | **Feature** | **Stato in main** | **Dettagli & Riferimenti** |
| --- | --- | --- | --- |
| **F9** | Smart-scroll notebook | *Parzialmente migrato* – esiste il binding su Notebook (bind\_mousewheel(self.nb)) fornito da ui\_utils.bind\_mousewheel() ma non viene applicato ai singoli Canvas interni ad ogni tab. | v. builder\_core.py#L84-L90 (bind su self.nb), tests/test\_smart\_scroll.py#L25-L33 |
| **F10** | UI polish / tema scuro (darkly) | *Mancante nella pratica* – il codice prova a importare ttkbootstrap.Style('darkly') in avvio, ma se ttkbootstrap non è installato non succede nulla. Non c’è un tema alternativo centralizzato e i few colori in assets.PALETTE non formano un vero “tema”. | v. builder\_core.py#L84-L90 (tentativo ttkbootstrap), assets.py#L4-L10 |
| **F11** | Diff-validator segnaposto (Audit segnaposti) | *Parzialmente migrato* – esiste il comando Edit → Audit Segnaposti (F11) che richiama TemplateBuilderApp.audit\_placeholders() e mostra un dialogo con elenco FOUND vs MISSING. Non è però integrato in reload\_template(): non evidenzia né colora i widget corrispondenti. | v. builder\_core.py#L149-L157 (audit\_placeholders()), builder\_core.py#L168-L176 (menu “Edit → Audit”). |
| **F12** | Auto-preview on change | *Parzialmente migrato* – molte componenti (es. PlaceholderMultiTextField, spinbox colonne) hanno binding a self.update\_preview(). Tuttavia, i campi URL/ALT (hero e step) sono tk.Entry semplici senza callback, per cui modifiche in questi campi non aggiornano live la preview. | v. widgets.py#L66-L74 (campo multilinea manda on\_change=self.update\_preview), builder\_core.py#L179-L186 (spinbox colonne). |
| **F16** | Column layout selector (spinbox COLS\_DESC/REC) | *Completamente migrato* – in tab “Images” esistono due ttk.Spinbox (alias PlaceholderSpinbox) per self.cols\_desc e self.cols\_rec, con from\_=1 a 4, command=self.update\_preview(). I valori di default (2 e 1) e la raccolta in \_collect() li inseriscono come COLS\_DESC e COLS\_REC. | v. builder\_core.py#L110-L118 (creazione Spinbox), builder\_core.py#L200-L203 (\_collect legge self.cols\_desc, self.cols\_rec e li passa al contesto). |
| **F18** | Motore di rendering modulare (TkinterWeb/CEF) | *Mancante* – esiste solo un’implementazione minimale di PreviewEngine: se tkinterweb.HtmlFrame è disponibile usa quello, altrimenti ScrolledText. Non c’è alcuna scelta utente tra più back-end (CEF o browser esterno). L’app usa sempre PreviewEngine. Serve estendere l’engine e aggiungere opzione UI. | v. preview\_engine.py#L2-L14, preview\_engine.py#L52-L60, builder\_core.py#L426-L433 (creazione tab Preview). |

Riassunto dei risultati:

* **Completamente migrate**:  
  + **F16** (Selettore colonne).
* **Parzialmente migrate**:  
  + **F9** (Smart-scroll notebook);
  + **F11** (Audit segnaposti).
  + **F12** (Auto-preview su cambio dati).
* **Mancanti o da estendere**:  
  + **F10** (Tema scuro/Dark-Bootstrap);
  + **F18** (Engine di rendering modulare con scelta backend).

Ogni indicazione è ricavata analizzando il codice main e confrontandolo con il comportamento legacy descritto nei documenti di **deep research** .

## **4. Radiografia Completa del Ramo main**

*(Nessuna sintesi: elenco puntuale di tutto ciò che c’è nel codice – ripreso e arricchito dai file PDF “Step 1 – Radiografia completa del ramo main”)*

### **4.1 Albero dei Moduli in template\_builder/**

| **Percorso** | **Tipo** | **Note sintetiche** |
| --- | --- | --- |
| template\_builder/\_\_init\_\_.py | pkg-root | Espone \_\_version\_\_, TemplateBuilderApp (lazy import) e funzione main() (“CLI entry point”). |
| template\_builder/\_\_main\_\_.py | entry-point | Avvia il CLI: from .builder\_core import TemplateBuilderApp; app = TemplateBuilderApp(); app.mainloop(). |
| template\_builder/assets.py | costanti | Regex segnaposto (e.g. PLACEHOLDER\_RGX), colori (PALETTE), colonne default (DEFAULT\_COLS), cartelle (HISTORY\_DIR), nome app (APP\_NAME), versione (VERSION). |
| template\_builder/builder\_core.py | controller | **Hub principale** dell’app: TemplateBuilderApp(tk.Tk) con UI, gestione stato (undo/redo), salvataggio/caricamento, preview, menu, binding shortcut, raccolta dati da widget. |
| template\_builder/filters.py | Jinja filters | Registry filtri: attualmente steps\_bind(env, raw) – stub che restituisce input invariato. |
| template\_builder/model.py | dataclass | Modello dominio: Hero, StepImage, GalleryRow; metodi to\_dict(), from\_dict(), fallback\_html(), to\_jinja\_ctx(). |
| template\_builder/step\_image.py | helpers | Funzioni “step-immagine”: sort\_steps, swap\_steps, renumber\_steps, bind\_steps, steps\_to\_html. |
| template\_builder/widgets.py | widget GUI | Libreria widget Tkinter estesi: \_Tooltip, TooltipMixin, PlaceholderEntry, PlaceholderSpinbox, PlaceholderMultiTextField (alias MultiTextField), SortableImageRepeaterField; costanti HAS\_DND, HAS\_TOOLTIP, PLACEHOLDER\_COLOR. |
| template\_builder/services/\_\_init\_\_.py | pkg | Re-export di tutti i moduli figlio (from .text import \*, from .images import \*, from .storage import \*); definisce \_\_all\_\_. |
| template\_builder/services/text.py | servizio | Funzioni pure per testo: smart\_paste, \_split\_lines, \_format\_line, auto\_format, extract\_placeholders, images\_to\_html, get\_field\_help; costanti \_HELP\_DEFAULTS, BULLET\_RGX, HTML\_ALLOWED\_TAGS. |
| template\_builder/services/images.py | servizio | Servizi pure immagine: guess\_grid, generate\_placeholders, \_ensure\_pillow, \_img\_to\_bytes, encode\_file\_to\_data\_uri, \_make\_img\_tag, paths\_to\_html\_grid, images\_to\_html, smart\_paste\_images, validate\_url, fetch\_metadata; costanti PLACEHOLDER\_BASE\_URL, MIME\_TYPES, HAS\_PIL. |
| template\_builder/services/storage.py | servizio | Persistenza & migrazione: load\_recipe, quick\_save, export\_html, \_timestamp, \_migrate\_v1\_to\_v2; classe UndoRedoStack; costanti SCHEMA\_VERSION = 2, \_BASE\_DIR, \_HISTORY\_DIR, AUTOSAVE\_LIMIT. |
| template\_builder/infrastructure/\_\_init\_\_.py | pkg | Inizializzatore infrastrutture; definisce fallback per import legacy su stub. |
| template\_builder/infrastructure/preview\_engine.py | infra | PreviewEngine: wrapping TkinterWeb/ScrolledText per anteprima HTML; metodi \_\_init\_\_, render, collect\_context, \_build\_gui, \_display\_available, \_safe. |
| template\_builder/infrastructure/ui\_utils.py | infra | Utility GUI generali: show\_info, show\_error, show\_warning, center\_window, bind\_mousewheel, ask\_yes\_no; costanti KEY\_CTRL, KEY\_CMD, OS\_NAME. |
| template\_builder/infrastructure/validators.py | infra | Funzioni validazione: validate\_url, validate\_email, validate\_date, validate\_nonempty; costanti URL\_RGX, EMAIL\_RGX, DATE\_RGX. |
| template\_builder/templates/… | template | ~10 file HTML/Jinja2 di esempio (e.g. ebay\_template\_modern\_dynamicv2.html, template\_ebay completo.html, …). |

Cartelle **escluse** dall’applicazione corrente:

* template\_builder/legacy/ (codice legacy monolitico, usato solo come riferimento storico)
* tests/ (suite di test Pytest completa)

### **4.2 Contenuto Modulo per Modulo**

#### **4.2.1 assets.py**

| **Categoria** | **Nomi presenti** |
| --- | --- |
| **Costanti** | PLACEHOLDER\_RGX, URL\_RGX, DATE\_RGX, DEFAULT\_COLS, HISTORY\_DIR, PALETTE, APP\_NAME, VERSION |

* PLACEHOLDER\_RGX = re.compile(r"\{\{\s\*([A-Z0-9\_]+)\s\*\}\}")
* URL\_RGX = r"https?://[^\s]+"
* DATE\_RGX = r"\d{4}-\d{2}-\d{2}"
* DEFAULT\_COLS = 3
* HISTORY\_DIR = os.path.join(os.path.expanduser("~"), ".template\_builder", "history")
* PALETTE = { "bg":"#f0f0f0", "fg":"#000000", "accent":"#007acc", … }
* APP\_NAME = "Template Builder"
* VERSION = "1.0.0"

#### **4.2.2 builder\_core.py**

| **Categoria** | **Nomi presenti** |
| --- | --- |
| **Classi** | TemplateBuilderApp(tk.Tk) |
| **Metodi pubblici** | \_\_init\_\_, run, edit\_undo, edit\_redo, quick\_save, load\_recipe, update\_preview, audit\_placeholders, reload\_template |
| **Metodi privati** | \_build\_ui, \_build\_menu, \_bind\_global\_shortcuts, \_collect, \_apply\_state\_to\_widgets, \_render\_html, \_toggle\_details, \_make\_scrollable, \_add\_tab, \_safe\_import |
| **Attributi chiave** | self.\_undo (UndoRedoStack), self.\_state (Dictstr,Anystr,Any), self.fields (Dictstr,Widgetstr,Widget), self.preview (PreviewEngine), self.nb (Notebook), self.cols\_desc, self.cols\_rec, self.menu, self.root |
| **Import principali** | template\_builder.widgets, template\_builder.services.text (extract\_placeholders), template\_builder.services.images (validate\_url), template\_builder.services.storage (UndoRedoStack, quick\_save, load\_recipe, export\_html), template\_builder.step\_image (bind\_steps), template\_builder.infrastructure.preview\_engine (PreviewEngine), template\_builder.infrastructure.ui\_utils (center\_window, bind\_mousewheel, show\_error, show\_info), template\_builder.assets, template\_builder.filters |
| **Note aggiuntive** | - Tutti i widget sono creati in \_build\_ui, con binding a callback (es. update\_preview) su <KeyRelease>. - \_collect() costruisce il contesto Jinja2 (data: Dict[str,Any]) a partire dai valori inseriti nei widget: testi (hero, descrizione, step), URL immagini, ALT, valori spinbox (COLS\_DESC, COLS\_REC). - audit\_placeholders() estrae segnaposto dal template Jinja e li confronta con chiavi contesto attuali. - \_render\_html() chiama export\_html() di services/storage e passa il risultato a PreviewEngine.render(). |

#### **4.2.3 model.py**

| **Categoria** | **Nomi presenti** |
| --- | --- |
| **Dataclass** | Hero, StepImage, GalleryRow |
| **Metodi** | to\_dict, from\_dict, fallback\_html, to\_jinja\_ctx |
| **Costanti** | \_\_all\_\_ = ["Hero","StepImage","GalleryRow"] |
| **Import** | from dataclasses import dataclass, from typing import List,Dict,Any, import html |
| **Note aggiuntive** | - Hero: campi (title, img, alt, intro); in \_\_post\_init\_\_ validazione di alt. - StepImage: campi (img, alt, text, order); in \_\_post\_init\_\_ solleva errore se img != "" e alt == "". - GalleryRow: lista di StepImage (max 3), to\_jinja\_ctx() raccoglie chiavi IMAGES\_DESC, IMAGES\_SRC, IMAGES\_ALT, IMAGES\_TEXT, IMAGES\_ORDER. |

#### **4.2.4 step\_image.py**

| **Categoria** | **Nomi presenti** |
| --- | --- |
| **Funzioni** | sort\_steps, swap\_steps, renumber\_steps, bind\_steps, steps\_to\_html |
| **Costanti** | nessuna esplicita (il modulo definisce \_\_all\_\_ = ["sort\_steps","swap\_steps","renumber\_steps","bind\_steps","steps\_to\_html"]). |
| **Import** | from typing import List, from .model import StepImage |
| **Descrizione** | - sort\_steps(steps: List[StepImage]) → restituisce lista ordinata per attributo order. - swap\_steps(steps: List[StepImage], idx1: int, idx2: int) → scambia due elementi e riassegna order. - renumber\_steps(steps: List[StepImage]) → rinumera order in sequenza 1…N. - bind\_steps(texts: List[str], images: List[str], alts: List[str]) → combina liste: se images[i] != "" e alts[i] == "", solleva errore; altrimenti crea StepImage(img, alt, text, order=i+1). - steps\_to\_html(steps: List[StepImage]) → produce <ol><li>…</li></ol> per i soli testi. |

#### **4.2.5 widgets.py**

| **Categoria** | **Nomi presenti** |
| --- | --- |
| **Classi** | \_Tooltip, TooltipMixin, PlaceholderEntry, PlaceholderSpinbox, PlaceholderMultiTextField (alias MultiTextField), SortableImageRepeaterField |
| **Funzioni helper** | \_apply\_border(widget, ok), \_split\_dnd\_event\_data(data) |
| **Costanti** | HAS\_DND, HAS\_TOOLTIP, PLACEHOLDER\_COLOR = "#888888" |
| **Import** | import tkinter as tk, from tkinter import ttk, from .services import text as text\_service, from .services import images as image\_service, from .infrastructure import ui\_utils, from .assets import PLACEHOLDER\_RGX, from typing import List |
| **Descrizione** | - \_Tooltip e TooltipMixin: gestione finestre balloon on-hover, usate da PlaceholderEntry e SortableImageRepeaterField per mostrare help testuale (text\_service.get\_field\_help(key)). - PlaceholderEntry: estensione di ttk.Entry con placeholder grigio, metodo get\_value(), render\_html() che applica text\_service.auto\_format(). - PlaceholderSpinbox: estende ttk.Spinbox con placeholder e binding a update\_preview(). - PlaceholderMultiTextField (MultiTextField): widget tk.Text con placeholder, richiede on\_change callback (passato da builder\_core), supporta text\_service.smart\_paste() e render\_html(). - SortableImageRepeaterField: crea dinamicamente righe di ttk.Entry per URL e ALT, bottoni +, –, ↑, ↓; validazione live con image\_service.validate\_url(), bordo verde/rosso tramite \_apply\_border(); supporta drag&drop di file se HAS\_DND = True, altrimenti fallback a pulsante “+ Add”. |

#### **4.2.6 filters.py**

| **Categoria** | **Nomi presenti** |
| --- | --- |
| **Funzioni** | steps\_bind(ctx, raw) |
| **Costanti** | \_\_all\_\_ = ["steps\_bind"] |

* **steps\_bind(ctx, raw)**: filtro Jinja2 legacy stub; riceve contesto Jinja2 ctx e stringa raw, restituisce raw inalterato.

#### **4.2.7 services/\_\_init\_\_.py**

* **Re-export** di tutto da services/text.py, services/images.py, services/storage.py; definisce \_\_all\_\_ opportuno.
* **Permette** import diretti come from template\_builder.services import quick\_save, validate\_url.

#### **4.2.8 services/text.py**

| **Categoria** | **Nomi presenti** |
| --- | --- |
| **Funzioni** | smart\_paste(raw), \_split\_lines(raw), \_format\_line(line), auto\_format(text, mode), extract\_placeholders(html\_src), images\_to\_html(rows, cols), get\_field\_help(key) |
| **Costanti** | \_HELP\_DEFAULTS, BULLET\_RGX, HTML\_ALLOWED\_TAGS |
| **Import** | import re, from typing import List, Set, import html |
| **Descrizione** | - smart\_paste: split su newline/;, rimuove righe vuote. - auto\_format: supporta mode="ul" (lista HTML) o "p" (paragrafo). Riconosce \*\*bold\*\*, \*italic\*. - extract\_placeholders: trova tutti i tag {{TAG}} in html\_src (usa assets.PLACEHOLDER\_RGX). - images\_to\_html: fallback HTML galleria se Jinja2 mancante. - \_HELP\_DEFAULTS: dizionario chiavi→stringhe help. |

#### **4.2.9 services/images.py**

| **Categoria** | **Nomi presenti** |
| --- | --- |
| **Funzioni** | guess\_grid(n\_images, cols), generate\_placeholders(n\_images), \_ensure\_pillow(), \_img\_to\_bytes(path), encode\_file\_to\_data\_uri(path), \_make\_img\_tag(src, alt), paths\_to\_html\_grid(paths, cols, inline=False, alt\_texts=None), images\_to\_html(paths, cols, inline=False, alt\_texts=None), smart\_paste\_images(raw), validate\_url(url), fetch\_metadata(path) |
| **Costanti** | PLACEHOLDER\_BASE\_URL, MIME\_TYPES, HAS\_PIL |
| **Import** | import base64, import os, from PIL import Image (in \_ensure\_pillow()), from typing import List, import requests (per metadata URL), from .text import smart\_paste |
| **Descrizione** | - guess\_grid: calcola (rows, cols) data n\_images e colonne desiderate. - generate\_placeholders: ritorna liste [ "{{IMG1}}", … ]. - \_ensure\_pillow: prova import PIL, se fallisce imposta Image=None (HAS\_PIL=False). - encode\_file\_to\_data\_uri: converte immagine locale in Data-URI base64. - paths\_to\_html\_grid: genera <table> HTML di immagini; inline=True usa Data-URI, altrimenti URL esterni. - images\_to\_html: alias. - smart\_paste\_images: delega a text.smart\_paste(). - validate\_url: wrapper su validators.validate\_url(). - fetch\_metadata: usa Pillow per metadata dimensioni; se mancante, ritorna {}. |

#### **4.2.10 services/storage.py**

| **Categoria** | **Nomi presenti** |
| --- | --- |
| **Classe** | UndoRedoStack |
| **Funzioni** | load\_recipe(path), quick\_save(state), export\_html(ctx, template\_path, \*\*env\_kw), \_timestamp(), \_migrate\_v1\_to\_v2(old) |
| **Costanti** | SCHEMA\_VERSION = 2, \_BASE\_DIR = os.path.expanduser("~/.template\_builder"), \_HISTORY\_DIR = os.path.join(\_BASE\_DIR, "history"), AUTOSAVE\_LIMIT = 50 |
| **Import** | import os, import json, from datetime import datetime, from typing import Dict, import jinja2 (in modalità lazy in export\_html), from ..step\_image import bind\_steps |
| **Descrizione** | - UndoRedoStack: gestisce stack di stati JSON. - quick\_save(state): salva JSON in ~/.template\_builder/history/recipe\_<timestamp>.json. - load\_recipe(path): legge file JSON, se SCHEMA\_VERSION < 2 chiama \_migrate\_v1\_to\_v2(). - \_migrate\_v1\_to\_v2(old): converte vecchio schema (campi "Step1", "IMAGES\_STEP", "ALTS\_STEP") in nuovo schema ("STEPS", "STEPn\_IMG\_ALT"), usando bind\_steps(). - export\_html(ctx, template\_path): render Jinja2 con contesto ctx, restituisce stringa HTML; se Jinja2 mancante, solleva eccezione. |

#### **4.2.11 infrastructure/preview\_engine.py**

| **Categoria** | **Nomi presenti** |
| --- | --- |
| **Classe** | PreviewEngine |
| **Metodi** | \_\_init\_\_, render(html), collect\_context(), \_build\_gui(parent), \_display\_available(), \_safe(name) |
| **Costanti** | nessuna definita esplicitamente |
| **Import** | import tkinter as tk, from tkinter.scrolledtext import ScrolledText, \_safe import "tkinterweb.HtmlFrame" |
| **Descrizione** | - In \_\_init\_\_: se tkinterweb disponibile, crea HtmlFrame, altrimenti ScrolledText. Imposta self.\_headless se nessun display. - render(html): aggiorna HtmlFrame o ScrolledText. - collect\_context(): stub vuoto. - \_display\_available(): controlla variabili d’ambiente o istanzia temporanea di Tk() per verificare “headless” vs “GUI”. - \_safe(name): import dinamico in try/except per moduli opzionali. |

#### **4.2.12 infrastructure/ui\_utils.py**

| **Categoria** | **Nomi presenti** |
| --- | --- |
| **Funzioni** | show\_info(msg), show\_error(msg), show\_warning(msg), center\_window(win), bind\_mousewheel(widget), ask\_yes\_no(msg) |
| **Costanti** | KEY\_CTRL = "Control", KEY\_CMD = "Command", OS\_NAME = platform.system().lower() |
| **Import** | import tkinter as tk, from tkinter import messagebox, import platform |
| **Descrizione** | - Interfaccia standard pop-up (info, errore, warning). - center\_window: calcola dimensioni schermo e centratura. - bind\_mousewheel: associa <MouseWheel>, <Button-4>, <Button-5> per scroll cross-platform. - ask\_yes\_no: popup conferma Yes/No. |

#### **4.2.13 infrastructure/validators.py**

| **Categoria** | **Nomi presenti** |
| --- | --- |
| **Funzioni** | validate\_url(url), validate\_email(email), validate\_date(date\_str), validate\_nonempty(value) |
| **Costanti** | URL\_RGX = re.compile(r"https?://[^\s]+"), EMAIL\_RGX = re.compile(r"[^@]+@[^@]+\.[^@]+"), DATE\_RGX = re.compile(r"\d{4}-\d{2}-\d{2}") |
| **Import** | import re, from datetime import datetime |
| **Descrizione** | - validate\_url: verifica se url matcha URL\_RGX. - validate\_email: verifica se email matcha EMAIL\_RGX. - validate\_date: contesto datetime.strptime(date\_str, "%Y-%m-%d"). - validate\_nonempty: len(value.strip())>0. |

#### **4.2.14 templates/**

Contiene ~10 file HTML/Jinja2 di esempio. Ogni template usa segnaposti {{…}} coerenti con le chiavi generate da model.py e services/storage.py. Esempi:

* ebay\_template\_modern\_dynamicv2.html
* ebay\_template\_modern\_full.html
* template dinamico prova.html
* template\_ebay completo.html
* template\_completov2.html
* template\_ebay\_+ricetta\_con\_foto.html
* template\_ebay+ricetta.html
* template\_final\_ebay.html
* template\_segnaposto\_prova.html

Ogni file contiene commenti Jinja, <!DOCTYPE html>, CSS inline, <head>, <body>, con segnaposti come {{ TITLE }}, {{ DESCRIPTION }}, {{ STEPS }}, {{ IMG1\_SRC }}, {{ IMG1\_ALT }}, ecc.

### **4.3 Mappa degli Import (Dipendenze Dirette)**

| **Modulo → importa** | **builder\_core** | **widgets** | **services.text** | **services.images** | **services.storage** | **step\_image** | **preview\_engine** | **ui\_utils** | **validators** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **assets.py** | ✔︎ | ✔︎ | ✔︎ | ✔︎ | ✔︎ | — | — | — | ✔︎ |
| **model.py** | ✔︎ | — | — | — | — | ✔︎ | — | — | — |
| **widgets.py** | ✔︎ | — | ✔︎ | ✔︎ | — | — | — | ✔︎ | — |
| **services/text.py** | ✔︎ | ✔︎ | — | — | — | — | — | — | — |
| **services/images.py** | — | ✔︎ | ✔︎ | — | — | — | — | — | ✔︎ |
| **services/storage.py** | ✔︎ | — | — | — | — | ✔︎ | — | — | — |
| **step\_image.py** | ✔︎ | — | — | — | ✔︎ | — | — | — | — |
| **preview\_engine.py** | ✔︎ | — | — | — | — | — | — | — | — |
| **ui\_utils.py** | ✔︎ | — | — | — | — | — | — | — | — |
| **validators.py** | — | — | — | ✔︎ | — | — | — | — | — |

*(✔︎ = import diretto; “—” = nessun import)*

* **builder\_core.py** importa widgets, services.text, services.images, services.storage, step\_image, premium\_engine, ui\_utils, assets, filters, validators.
* **widgets.py** importa services.text, services.images, ui\_utils, assets.
* **services/storage.py** importa step\_image.bind\_steps.
* **services/images.py** importa assets.DEFAULT\_COLS, services.text.smart\_paste, validators.validate\_url.

### **4.4 Punti di Contatto (Call‐Graph ad Alto Livello)**

| **Chiamante → Callee** | **Finalità** |
| --- | --- |
| TemplateBuilderApp.\_collect() → step\_image.bind\_steps() | Genera lista STEPS a partire da testi, URL immagini e ALT. |
| TemplateBuilderApp.quick\_save() → services.storage.quick\_save() | Salva lo stato corrente su file JSON con timestamp. |
| TemplateBuilderApp.load\_recipe(path) → services.storage.load\_recipe(path) | Carica e migra progetto (.json legacy v1→v2) tramite \_migrate\_v1\_to\_v2(). |
| TemplateBuilderApp.update\_preview() → PreviewEngine.render(html) | Richiede rendering HTML e aggiorna il riquadro anteprima. |
| SortableImageRepeaterField.\_validate() → services.images.validate\_url() | Evidenzia in tempo reale la validità di un URL immesso (bordi verde/rosso). |
| **Ogni widget <KeyRelease>** → TemplateBuilderApp.update\_preview() | Aggiorna live l’anteprima ad ogni modifica di testo o spinbox (tranne Entry URL/ALT image). |
| services.storage.\_migrate\_v1\_to\_v2() → step\_image.bind\_steps() | Durante il caricamento di JSON legacy v1, popola correttamente i nuovi campi STEPS e STEPn\_IMG\_ALT. |

### **4.5 Convenzioni di Nomenclatura (Globali)**

* **Classi**: PascalCase (es. TemplateBuilderApp, StepImage, PreviewEngine).
* **Funzioni/Metodi**: snake\_case (es. quick\_save, bind\_steps, sort\_steps, load\_recipe).
* **Costanti**: SCREAMING\_SNAKE (es. DEFAULT\_COLS, PLACEHOLDER\_RGX, SCHEMA\_VERSION).
* **Segnaposto Jinja2 / Chiavi di stato**: UPPER\_SNAKE (es. TITLE, HERO\_IMAGE\_SRC, STEP3\_IMG\_ALT).
* **Alias**: MultiTextField è alias di PlaceholderMultiTextField.
* **Pacchetti/Moduli**: minuscolo con underscore (es. services, infrastructure, step\_image).
* **Chiavi JSON serializzate**:  
  + Per Hero: "TITLE", "HERO\_IMAGE\_SRC", "HERO\_IMAGE\_ALT", "INTRO".
  + Per StepImage: "IMG\_SRC", "IMG\_ALT", "TEXT", "ORDER".
  + Lista STEPS: Array di dict con chiavi sopra; e chiavi separate "STEPn\_IMG\_ALT".

**Step 1 completato: Radiografia completa del ramo main.** Fammi sapere se necessiti ulteriori chiarimenti prima di procedere allo **Step 2 (Verifica dettagliata delle feature legacy)**.

## **5. Verifica della Migrazione Legacy (Feature F9–F18)**

*(Analisi puntuale basata sul documento “Verifica della migrazione legacy”)*

### **5.1 Confronto Funzionalità Legacy Mancanti vs Stato in main**

#### **F9 – Smart-scroll notebook**

* **Legacy**:  
  + Scrolling fluido delle tab con rotella mouse: \_on\_mousewheel veniva bindato sia al Notebook principale sia ai Canvas interni per ogni tab.
* **In main**:  
  + In TemplateBuilderApp.\_build\_ui(), il metodo ui\_utils.bind\_mousewheel(self.nb) associa la rotella al Notebook principale (self.nb).
  + Il binding **FUNZIONA** a livello di Notebook ma **NON** sui singoli widget scrollabili (Canvas internamente usati).
* **Stato**: **Parzialmente migrato** – il binding di base esiste ma manca l’integrazione sui contenuti scrollabili interni.
* **Riferimenti**:  
  + builder\_core.py#L84-L90 (bind su self.nb)
  + tests/test\_smart\_scroll.py#L25-L33

#### **F10 – UI polish / tema scuro (darkly)**

* **Legacy**:  
  + Tema “Dark-Bootstrap” uniforme, con stili personalizzati, palette colori omogenea, padding e bordo coerenti.
* **In main**:

All’avvio, builder\_core.py tenta:  
  
 try:

from ttkbootstrap import Style

Style('darkly')

except Exception:

pass

* + Se ttkbootstrap non installato, non accade nulla. Non esiste un tema “chiaro” alternativo o stili centralizzati.
* **Stato**: **Mancante nella pratica** – c’è solo un “placeholder” per applicare ttkbootstrap; mancano stili centralizzati e un tema integrato se ttkbootstrap non presente.
* **Riferimenti**:  
  + builder\_core.py#L84-L90 (tentativo ttkbootstrap)
  + assets.py#L4-L10 (alcuni colori ma non tema completo)

#### **F11 – Diff-validator segnaposto (Audit segnaposti)**

* **Legacy**:  
  + Quando si ricaricava un template, i placeholder mancanti/extra venivano evidenziati in giallo/rosso nei campi corrispondenti.
* **In main**:  
  + Nel menu “Edit → Audit Segnaposti (F11)”, viene chiamato TemplateBuilderApp.audit\_placeholders(), che:  
    - Estrae segnaposto dal template (usando services.text.extract\_placeholders()).
    - Confronta con chiavi contesto attuali (stato interno).
    - Mostra un elenco in dialogo show\_info(), con linee “FOUND vs MISSING”.
  + **Manca** l’integrazione in reload\_template(): non colora o evidenzia i widget GUI. L’audit è un’operazione manuale separata.
* **Stato**: **Parzialmente migrato** – esiste il metodo di audit ma **non** il comportamento automatico in fase di caricamento né la colorazione live dei widget.
* **Riferimenti**:  
  + builder\_core.py#L149-L157 (audit\_placeholders())
  + builder\_core.py#L168-L176 (definizione menu “Edit → Audit”).

#### **F12 – Auto-preview on change**

* **Legacy**:  
  + Ogni modifica di un campo (testo, URL, spinbox) triggerava immediatamente l’aggiornamento dell’anteprima.
* **In main**:  
  + PlaceholderMultiTextField (campo multilinea) ha on\_key\_release=lambda e: self.update\_preview().
  + Spinbox colonne (self.cols\_desc, self.cols\_rec) hanno command=self.update\_preview.
  + undo/redo forzano update\_preview().
  + **Mancano** i binding automatici sui campi URL e ALT (hero e step), che sono semplici tk.Entry senza on\_change.
* **Stato**: **Parzialmente migrato** – molte componenti (testo, spinbox) triggerano preview, ma non tutti i campi (URL/ALT immagine).
* **Riferimenti**:  
  + widgets.py#L66-L74 (campo multilinea manda on\_change=self.update\_preview)
  + builder\_core.py#L179-L186 (spinbox colonne command=self.update\_preview)
  + builder\_core.py#L415-L423 (undo/redo chiamano update\_preview())

#### **F16 – Column layout selector (spinbox COLS\_DESC e COLS\_REC)**

* **Legacy**:  
  + Nella sezione “Description” e “Recipe” esistevano due spinbox COLS\_DESC, COLS\_REC per definire numero colonne di visualizzazione immagini.
* **In main**:  
  + Nella scheda “Images” (builder\_core.py), due ttk.Spinbox (alias PlaceholderSpinbox) con from\_=1 a 4, assegnati a self.cols\_desc e self.cols\_rec, entrambi con command=self.update\_preview.
  + Valori di default: self.cols\_desc = 2, self.cols\_rec = 1.
  + In \_collect(), lettura di self.cols\_desc.get() e self.cols\_rec.get() → contesto Jinja2: campi COLS\_DESC, COLS\_REC.
* **Stato**: **Completamente migrato** – spinbox esistono in UI, passano valori al contesto. Rimane solo da verificare che i template Jinja usino effettivamente COLS\_DESC per layout.
* **Riferimenti**:  
  + builder\_core.py#L110-L118 (creazione spinbox)
  + builder\_core.py#L200-L203 (\_collect legge e inserisce nel contesto i valori di self.cols\_desc, self.cols\_rec)

#### **F18 – Motore di rendering modulare (TkinterWeb/CEF/webbrowser)**

* **Legacy**:  
  + L’utente poteva scegliere tra tkinterweb.HtmlFrame, CefBrowser, o aprire il rendering in browser esterno.
* **In main**:  
  + PreviewEngine.\_\_init\_\_() tenta import tkinterweb.HtmlFrame; se presente, lo usa; altrimenti ScrolledText.
  + In builder\_core, se l’istanza PreviewEngine esiste, crea il tab “Preview” con self.preview = PreviewEngine(...); altrimenti mostra “Preview non disponibile”.
  + Non c’è alcuna scelta utente tra diversi motori. Non è stato portato CEF e non viene lanciato webbrowser.open().
* **Stato**: **Mancante** – esiste solo uno stub di PreviewEngine; serve estendere per includere CEF e browser esterno, e aggiungere opzione in UI per selezionare backend.
* **Riferimenti**:  
  + preview\_engine.py#L2-L14, preview\_engine.py#L52-L60 (logica import tkinterweb vs fallback)
  + builder\_core.py#L426-L433, builder\_core.py#L434-L442 (creazione tab “Preview”)

**Sintesi Migrazione Legacy (F9–F18):**

* **Complete**:  
  + **F16** – selettore colonne (Spinbox).
* **Parziali**:  
  + **F9** – smart-scroll notebook (binding solo al Notebook).
  + **F11** – audit segnaposti (solo manuale, senza evidenziazione GUI).
  + **F12** – auto-preview (su alcuni campi, non su URL/ALT).
* **Assenti**:  
  + **F10** – tema scuro integrato (solo placeholder ttkbootstrap).
  + **F18** – motore di rendering modulare (solo stub PreviewEngine).

## **6. Documentazione & Change-Log: Stato Attuale, Gap e Azioni da Eseguire**

### **6.1 Stato Attuale dei Documenti**

| **Documento** | **Stato** |
| --- | --- |
| **README.md** | Fornisce panoramica base: elenco moduli, dipendenze minime (Python ≥3.10, Tkinter), istruzioni installazione e avvio (pip install, pip install template\_builder[dnd,webpreview], python -m template\_builder). Mancano: sezione “How to Run/Install” dettagliata, elenco completo dipendenze, esempi d’uso. |
| **ARCHITECTURE.md** | Descrive in alto livello la struttura (controller, servizi, modelli, infrastruttura). Non è aggiornato con: - Nuovi widget (es. SortableImageRepeaterField). - Dettagli di step\_image.py. - Logica migratoria v1→v2. - Dipendenze tkinterweb, ttkbootstrap, Pillow. |
| **CHANGELOG.md** | Contiene un **placeholder** (“# GENERARE COERENTEMENTE AL REPOSITORY…”). Non elenca alcuna versione reale né commit. |
| **docs/legacy\_issues.md** | Elenco issue legacy (F9–F18), ma: - Alcune sono già risolte (F16, F12 parzialmente) ma rimangono elencate come “mancanti”. - Mancano riferimenti al codice attuale (builder\_core.py, widgets.py, services/storage.py). |
| **IMPACT\_MATRIX\_B2.md** | Matrice di impatto e piani sviluppo, ormai obsoleta (descrive fasi A/B, non riflette F9–F18). |
| **DEV\_PLAN\_B2.md** | Piano di sviluppo parziale (Fasi A/B completate), non aggiornato con F18, F10, F11. |
| **CHANGELOG.md** (docs/) | Lungo elenco di TODO (es. “da sistemare…”), non correlato a commit reali. |
| **Docs aggiuntivi (docs/ folder)** | Alcuni documenti contengono note “poi la aggiorno” o commenti “da sistemare”: dev plan, impact matrix, integration guide, project-files.txt. Non allineati al codice reale. |

### **6.2 Azioni Raccomandate (Documentazione)**

1. **README.md**

**Sezione “How to Install / How to Run”:** dettagliare:  
  
 # Installazione

1. Creare e attivare un ambiente virtuale con Python ≥3.10

```bash

python3 -m venv venv

source venv/bin/activate # Linux/Mac

venv\Scripts\activate # Windows

Installare il pacchetto con tutte le dipendenze:  
  
 pip install template\_builder

pip install "template\_builder[dnd,webpreview]"

Opzionale: clonare il repository e installare in editable con test:  
  
 git clone https://github.com/polsi05/template\_builder.git

cd template\_builder

pip install -e .[test]

* + **Avvio**

**CLI**:  
  
 python -m template\_builder

* + - Avvia TemplateBuilderApp se è disponibile un display.

**Da codice Python**:  
  
 from template\_builder.builder\_core import TemplateBuilderApp

app = TemplateBuilderApp()

if app.root:

app.root.mainloop()

* + - **Opzioni**:  
      * Con tkinterdnd2 abilitato, il repeater immagini supporta drag&drop.
      * Con tkinterweb installato, la tab “Preview” mostra l’anteprima HTML.
      * Con ttkbootstrap installato, il tema “darkly” viene applicato all’avvio.
  + **Elenco moduli** (riassunto puntato, vedasi sezione 2)
  + **Dipendenze complete**: elenco in tabella (vedi requirements.txt).
  + **Esempi**: screenshot (facoltativo), snippet codice, mini tutorial “crea un template di esempio e vedi anteprima”.

1. **ARCHITECTURE.md**
   * **Struttura ad albero** (sezione 4.1, tabella moduli) con descrizione dettagliata.
   * **Descrizione modulo‐per‐modulo** (sezione 4.2), con codice citato come proveniente da template\_builder/....
   * **Mappa import / dipendenze** (sezione 4.3, tabella), per visione del grafo di dipendenze.
   * **Punti di contatto (call-graph)** (sezione 4.4), per chiarire flusso esecuzione e invocazioni tra componenti.
   * **Convenzioni di naming** (sezione 4.5).
   * **Diagramma semplificato** (opzionale): un disegno ad alto livello (controller → servizi → widget → anteprima).
   * **Aggiornare** ogni sezione in base al codice attuale del ramo main.
2. **CHANGELOG.md**

**Formato**: “scheda tecnica” (non solo “sintesi tecnica”), ossia per ogni versione/commit elencare:  
  
 # Changelog

## [1.0.0] – 2025-06-06

- \*\*Nuovi moduli\*\*: aggiunte dataclass `Hero` e `GalleryRow` (gestione sezione hero e gallerie immagini). :contentReference[oaicite:43]{index=43}

- \*\*Anteprima avanzata\*\*: `PreviewEngine` con fallback a `ScrolledText` se `tkinterweb` mancante. :contentReference[oaicite:44]{index=44}

- \*\*Undo/Redo\*\*: `UndoRedoStack` integrato in UI con scorciatoie (Ctrl+Z/Y, Cmd+Z/Shift+Z). :contentReference[oaicite:45]{index=45}

- \*\*Drag&Drop immagini\*\*: `SortableImageRepeaterField` usa `tkinterdnd2` se disponibile (fallback a pulsante “+ Add”). :contentReference[oaicite:46]{index=46}

- \*\*Validazione live\*\*: campi URL/ALT evidenziati con bordo verde/rosso. :contentReference[oaicite:47]{index=47}

- \*\*Tooltips\*\*: suggerimenti on-hover basati su `text.get\_field\_help()`. :contentReference[oaicite:48]{index=48}

- \*\*Persistenza JSON\*\*: `quick\_save`, `load\_recipe`, con migrazione automatica v1→v2 (`\_migrate\_v1\_to\_v2()`). :contentReference[oaicite:49]{index=49}

- \*\*Esportazione HTML\*\*: `export\_html(ctx, template\_path)` con Jinja2 lazy import (throw se mancante). :contentReference[oaicite:50]{index=50}

- \*\*Rifattorizzazione\*\*: separazione servizi (`images.py`, `text.py`, `storage.py`), rimozione stub legacy superflui. :contentReference[oaicite:51]{index=51}

## [0.1.0] – <data precedente>

- Correzioni legacy:

- \*\*Bug anteprima HTML complessi\*\*, \*\*undo/redo\*\*, \*\*drag&drop immagini\*\*, \*\*placeholder sui campi\*\*, \*\*validazione live\*\*, \*\*tooltip\*\* (tracking in `docs/legacy\_issues.md`). :contentReference[oaicite:52]{index=52}

- \*\*Migrazione automatica dati\*\*: `\_migrate\_v1\_to\_v2()` in `services/storage.py`. :contentReference[oaicite:53]{index=53}

## [0.0.1] – <prima versione>

- \*\*Skeleton & Core\*\*: struttura di base del package (`builder\_core.py`, `widgets.py`, `services/`, `infrastructure/`, `model.py`, `filters.py`, `assets.py`).

- \*\*Test headless\*\*: tutti i moduli importabili in `xvfb` senza crash.

- \*\*Package & CI\*\*: GitHub Actions `ci.yml` per pip install e `pytest -q`.

- \*\*Tag v0.1.0\*\* creato a conclusione Fasi A/B.

:contentReference[oaicite:54]{index=54}

* + **Dettaglio commit**: se utile, citare commit hash, autore, data.
  + **Strutturare come scheda tecnica**: elenco puntato con descrizione concisa di ogni modifica.
  + **Fonti**: usare cronologia Git (git log --oneline) e riferimenti a file modificati (vedere blocchi citati).

1. **docs/** (Cartella di Documentazione)  
   * **Rimuovere/Archiviare** i file obsoleti:  
     + IMPACT\_MATRIX\_B2.md, DEV\_PLAN\_B2.md, INTEGRATION\_GUIDE\_B2.md, PROJECT-FILES.txt, PRECAUTION\_CHECKLIST\_B2.md.
   * **Generare** un nuovo file docs/migration\_summary.md che includa solo le issue legacy **ancora aperte** (F9, F10, F11, F12 parzialmente, F18).
   * **Aggiornare** docs/legacy\_issues.md rimuovendo entry risolte (es. F16) e inserendo riferimenti a file sorgenti dove sono già implementate.
   * **Creare** eventuali file di supporto per nuovi sviluppatori:  
     + docs/CONTRIBUTING.md – linee guida per contribuire (stile codice, test, pull request).
     + docs/BUILD\_AND\_TEST.md – istruzioni dettagliate per eseguire test (xvfb), linting (flake8), report coverage.
     + docs/RELEASE\_PROCESS.md – guida passo-passo per creare release su PyPI (aggiornare pyproject.toml, creare branch release/vX.Y.Z, tag, PR).

## **7. Documenti Generati Ex Novo**

### **7.1 README.md (Versione Rivista)**

# Template Builder

\*\*Template Builder\*\* è un toolkit GUI in Python 3.10+ per creare agevolmente \*\*template HTML\*\* (ad es. campagne email) con anteprima live, gestione immagini intuitiva, undo/redo e altro.

---

## Struttura del Progetto

```text

template\_builder/

├─ \_\_init\_\_.py # Espone TemplateBuilderApp e funzione main()

├─ \_\_main\_\_.py # Entry-point CLI: avvia l'app

├─ assets.py # Costanti globali (regex segnaposto, colori, cartella di history)

├─ builder\_core.py # Controller principale (TemplateBuilderApp)

├─ filters.py # Filtri Jinja2 (stub `steps\_bind`)

├─ model.py # Dataclass di dominio: Hero, StepImage, GalleryRow

├─ step\_image.py # Funzioni helper per “step-immagine”

├─ widgets.py # Widget Tkinter personalizzati (placeholder, repeater immagini, tooltip)

├─ services/ # Servizi “puri” (senza GUI)

│ ├─ \_\_init\_\_.py # Re-export API di servizi

│ ├─ images.py # Gestione immagini: griglie, placeholder, Data-URI, smart-paste

│ ├─ text.py # Manipolazione testo: smart-paste, auto-format, estrazione placeholder

│ └─ storage.py # Persistenza JSON, migrazione v1→v2, export HTML, Undo/Redo

├─ infrastructure/ # Wrapper e utilità (preview HTML, GUI utils, validator)

│ ├─ \_\_init\_\_.py

│ ├─ preview\_engine.py # `PreviewEngine`: anteprima HTML con `tkinterweb` o fallback

│ ├─ ui\_utils.py # Utility GUI generali (popup, centrare finestre, bind mousewheel)

│ └─ validators.py # Validatori campi (URL, email, date, non-vuoto)

├─ templates/ # Template Jinja2 di esempio per preview/esportazione

└─ legacy/ # Codice monolitico legacy (solo a scopo di riferimento)

tests/ # Suite Pytest per moduli business, widget, servizi

README.md, ARCHITECTURE.md, CHANGELOG.md, tests/, setup file (pyproject.toml, pytest.ini, ecc.)

## **How to Install**

Requisiti minimi:

* **Python 3.10+** con modulo **Tkinter** incluso (python3-tk su Linux).
* **pip** aggiornato.

# 1. Clona il repository (opzionale se vuoi testare in locale)

git clone https://github.com/polsi05/template\_builder.git

cd template\_builder

# 2. Crea e attiva un virtual environment

python3 -m venv venv

source venv/bin/activate # su Linux/Mac

venv\Scripts\activate # su Windows

# 3. Installa il pacchetto principale

pip install template\_builder

# 4. (Opzionale) Installa tutte le dipendenze opzionali

pip install "template\_builder[dnd,webpreview]"

# 5. (Per sviluppatori) Installa in modalità editable con dipendenze test

pip install -e .[test]

## **How to Run**

### **Da CLI**

python -m template\_builder

Questo comando avvia TemplateBuilderApp se è disponibile un display GUI.

### **Da codice Python**

from template\_builder.builder\_core import TemplateBuilderApp

app = TemplateBuilderApp()

if app.root:

app.root.mainloop()

## **Funzionalità Principali**

* **Anteprima Live** Visualizza in tempo reale il rendering del template HTML generato dal contesto Jinja2. Se tkinterweb è installato, usa un HtmlFrame; altrimenti, un semplice ScrolledText per fallback.
* **Undo/Redo** Gestione completa delle modifiche tramite UndoRedoStack. Scorciatoie da tastiera:  
  + **Windows/Linux**: Ctrl+Z (Undo), Ctrl+Y (Redo)
  + **MacOS**: Cmd+Z (Undo), Shift+Z (Redo)
* **Gestione Immagini**
  + **Drag&Drop** (se tkinterdnd2 installato) o pulsante “+ Add” per aggiungere URL di immagini.
  + Validazione live di URL immagini: bordo verde se valido, rosso se non valido.
  + Se Pillow installato, fetch\_metadata() estrae larghezza/altezza immagine.
  + Generazione HTML di griglie (images\_to\_html()).
* **Widget Avanzati**
  + PlaceholderEntry, PlaceholderSpinbox, PlaceholderMultiTextField: campi con testo “ghost” (placeholder) e validazione live.
  + SortableImageRepeaterField: lista dinamica di URL immagini + ALT, con pulsanti di aggiunta/rimozione e spostamento righe.
* **Migrazione Dati**
  + Caricamento di progetti legacy v1 (chiavi "Step1", "IMAGES\_STEP", "ALTS\_STEP") → nuovo schema v2 (chiavi "STEPS", "STEPn\_IMG\_ALT") tramite \_migrate\_v1\_to\_v2().
* **Audit Segnaposti**
  + Menu “Edit → Audit Segnaposti (F11)”: estrae tutti i segnaposto {{TAG}} dal template e li confronta con chiavi contesto interne, mostrando elenco “FOUND vs MISSING”.
* **Selettore Colonne**
  + Nella tab “Images”, due spinbox per definire numero di colonne (COLS\_DESC, COLS\_REC) con binding a update\_preview().
* **Tipografia e Formattazione**
  + smart\_paste(): normalizza input multi-line / separato da ;.
  + auto\_format(): converte testo plain in <ul>, <li>, <p>.
  + extract\_placeholders(): estrae segnaposto Jinja2.

## **Dipendenze**

Le dipendenze obbligatorie e opzionali sono elencate in requirements.txt (vedi sezione 7.5).

* **Obbligatorie**:  
  + Python 3.10+ (modulo **Tkinter** incluso)

PyPI package:  
  
 jinja2>=3.0

Pillow>=8.0

tkinterdnd2>=0.3 # per drag&drop immagini (opzionale)

tkinterweb>=0.4 # per anteprima HTML (opzionale)

ttkbootstrap>=0.5 # per tema scuro “darkly” (opzionale)

## **Esempio di Utilizzo**

Avvia l’app:  
  
 python -m template\_builder

1. Nella finestra principale:  
   * Clicca sul tab “Hero” e inserisci titolo, URL immagine hero e testo ALT.
   * Clicca sul tab “Description” e digita il testo: usa auto\_format per elenchi puntati.
   * Clicca sul tab “Recipe” e incolla, separando le righe con ; o newline, i nomi dei passaggi; usa “Add Image” per ogni URL immagine del passo (se tkinterdnd2 installato puoi trascinare).
   * Apri il menu “Edit → Audit Segnaposti” per verificare segnaposto mancanti/caratteristiche.
   * Premi Ctrl+S o usa “File → Save” per salvare progetto come JSON in ~/.template\_builder/history/.
2. Crea un template di esempio, poi “File → Export HTML” per generare file HTML finale (usa Jinja2).

## **Sezione How to Contribute**

Per informazioni dettagliate su come contribuire (lint, test, PR, GitHub Actions), vedi docs/CONTRIBUTING.md (da creare).

### **Fonti (README.md attuale)**

* GitHub README.md
* GitHub ARCHITECTURE.md#L48-L50, ARCHITECTURE.md#L121-L126

### **7.2 ARCHITECTURE.md (Versione Estesa)**

# Architecture di Template Builder

Questa guida descrive in dettaglio la \*\*struttura modulare\*\* e le \*\*dipendenze\*\* interne del progetto, con riferimenti ai file sorgente e agli elementi chiave.

---

## 1. Panoramica ad Albero dei Moduli

```text

project-root/

├── README.md

├── ARCHITECTURE.md

├── CHANGELOG.md

├── LICENSE

├── pyproject.toml

├── pytest.ini

├── template\_builder/

│ ├── \_\_init\_\_.py # Espone TemplateBuilderApp e funzione main()

│ ├── \_\_main\_\_.py # Entry-point CLI

│ ├── assets.py # Costanti globali (regex, colori, cartella history)

│ ├── builder\_core.py # Controller principale (TemplateBuilderApp)

│ ├── filters.py # Filtri Jinja2 (steps\_bind stub)

│ ├── model.py # Dataclass di dominio (Hero, StepImage, GalleryRow)

│ ├── step\_image.py # Funzioni helper per step-immagine (sort, swap, bind)

│ ├── widgets.py # Widget Tkinter personalizzati (placeholder, repeater)

│ ├── services/

│ │ ├── \_\_init\_\_.py # Re-export API di servizi

│ │ ├── images.py # Servizi immagine (griglie, placeholder, Data-URI, smart-paste)

│ │ ├── text.py # Servizi testo (smart-paste, auto-format, estrazione placeholder)

│ │ └── storage.py # Persistenza JSON, migrazione v1→v2, export HTML, Undo/Redo

│ ├── infrastructure/

│ │ ├── \_\_init\_\_.py # Inizializzatore infrastruttura (stub fallback)

│ │ ├── preview\_engine.py # `PreviewEngine`: anteprima HTML (tkinterweb/ScrolledText)

│ │ ├── ui\_utils.py # Utility GUI (popup, centrare finestre, bind mousewheel)

│ │ └── validators.py # Validatori campi (URL, email, date, non-vuoto)

│ └── templates/ # Template Jinja2 di esempio per preview/esportazione

└── tests/ # Suite Pytest (test modelli, servizi, widget)

## **2. Contenuto Modulo per Modulo**

### **2.1 template\_builder/\_\_init\_\_.py**

Espone:  
  
 \_\_version\_\_ = "1.0.0"

from .builder\_core import TemplateBuilderApp, main

* Funzione main() – wrapper che istanzia TemplateBuilderApp e chiama mainloop().

### **2.2 template\_builder/\_\_main\_\_.py**

Entry-point CLI:  
  
 from .builder\_core import TemplateBuilderApp

if \_\_name\_\_ == "\_\_main\_\_":

app = TemplateBuilderApp()

if app.root:

app.root.mainloop()

Permette:  
  
 python -m template\_builder

* per avviare l’applicazione.

### **2.3 template\_builder/assets.py**

**Costanti globali**:  
  
 import os, re

PLACEHOLDER\_RGX = re.compile(r"\{\{\s\*([A-Z0-9\_]+)\s\*\}\}")

URL\_RGX = r"https?://[^\s]+"

DATE\_RGX = r"\d{4}-\d{2}-\d{2}"

DEFAULT\_COLS = 3

HISTORY\_DIR = os.path.join(os.path.expanduser("~"), ".template\_builder", "history")

PALETTE = {

"bg": "#f0f0f0",

"fg": "#000000",

"accent": "#007acc",

"error": "#ff4c4c",

...

}

APP\_NAME = "Template Builder"

VERSION = "1.0.0"

* **Utilizzo**:  
  + In services/text.py: PLACEHOLDER\_RGX per extract\_placeholders().
  + In services/images.py: DEFAULT\_COLS per calcolo griglie.
  + In widget PlaceholderEntry: PLACEHOLDER\_COLOR.

### **2.4 template\_builder/filters.py**

**Registro filtri Jinja2**:  
  
 \_\_all\_\_ = ["steps\_bind"]

def steps\_bind(env, raw):

"""

Stub filter legacy: riceve il contesto Jinja (env) e restituisce inalterato raw.

"""

return raw

* **Motivazione**:  
  + Mantenere compatibilità con i template legacy che utilizzano {% filter steps\_bind %}.

### **2.5 template\_builder/model.py**

**Dataclass Hero**:  
  
 from dataclasses import dataclass

import html

@dataclass

class Hero:

title: str

img: str

alt: str

intro: str

def \_\_post\_init\_\_(self):

if self.img and not self.alt:

raise ValueError("Se img non vuota, deve essere definito alt")

def to\_dict(self) -> dict:

return {

"TITLE": self.title,

"HERO\_IMAGE\_SRC": self.img,

"HERO\_IMAGE\_ALT": self.alt,

"INTRO": self.intro

}

def fallback\_html(self) -> str:

# Restituisce HTML semplificato di fallback se Jinja2 non presente

esc\_title = html.escape(self.title)

esc\_intro = html.escape(self.intro)

return f"<h1>{esc\_title}</h1><p>{esc\_intro}</p>"

@classmethod

def from\_dict(cls, data: dict):

return Hero(

title=data.get("TITLE",""),

img=data.get("HERO\_IMAGE\_SRC",""),

alt=data.get("HERO\_IMAGE\_ALT",""),

intro=data.get("INTRO","")

)

**Dataclass StepImage**:  
  
 from dataclasses import dataclass

@dataclass

class StepImage:

img: str

alt: str

text: str

order: int

def \_\_post\_init\_\_(self):

if self.img and not self.alt:

raise ValueError("Ogni immagine richiede un testo ALT.")

def to\_dict(self) -> dict:

return {

"IMG\_SRC": self.img,

"IMG\_ALT": self.alt,

"TEXT": self.text,

"ORDER": self.order

}

@classmethod

def from\_dict(cls, data: dict):

return StepImage(

img=data.get("IMG\_SRC",""),

alt=data.get("IMG\_ALT",""),

text=data.get("TEXT",""),

order=data.get("ORDER",0)

)

**Dataclass GalleryRow**:  
  
 from dataclasses import dataclass

from typing import List

@dataclass

class GalleryRow:

steps: List[StepImage] # ≤ 3 elementi

def \_\_post\_init\_\_(self):

if len(self.steps) > 3:

raise ValueError("Max 3 immagini per riga.")

def to\_jinja\_ctx(self) -> dict:

return {

"IMAGES\_DESC": [step.text for step in self.steps],

"IMAGES\_SRC": [step.img for step in self.steps],

"IMAGES\_ALT": [step.alt for step in self.steps],

"IMAGES\_ORDER": [step.order for step in self.steps]

}

def fallback\_html(self) -> str:

rows\_html = "".join(

f'<td><img src="{html.escape(s.img)}" alt="{html.escape(s.alt)}" /></td>'

for s in self.steps

)

return f"<table><tr>{rows\_html}</tr></table>"

**Costanti**:  
  
 \_\_all\_\_ = ["Hero","StepImage","GalleryRow"]

### **2.6 template\_builder/step\_image.py**

**Funzioni**:  
  
 from typing import List

from .model import StepImage

\_\_all\_\_ = ["sort\_steps","swap\_steps","renumber\_steps","bind\_steps","steps\_to\_html"]

def sort\_steps(steps: List[StepImage]) -> List[StepImage]:

return sorted(steps, key=lambda s: s.order)

def swap\_steps(steps: List[StepImage], idx1: int, idx2: int) -> None:

steps[idx1], steps[idx2] = steps[idx2], steps[idx1]

# Riassegna order

for i, s in enumerate(steps):

s.order = i + 1

def renumber\_steps(steps: List[StepImage]) -> None:

for i, s in enumerate(steps):

s.order = i + 1

def bind\_steps(texts: List[str], images: List[str], alts: List[str]) -> List[StepImage]:

result = []

for i, (t, img, alt) in enumerate(zip(texts, images, alts)):

if img and not alt:

raise ValueError(f"Immagine presente ma ALT mancante per indice {i}")

order = i + 1

result.append(StepImage(img=img, alt=alt, text=t, order=order))

return result

def steps\_to\_html(steps: List[StepImage]) -> str:

items = "".join(f"<li>{html.escape(s.text)}</li>" for s in sorted(steps, key=lambda s:s.order))

return f"<ol>{items}</ol>"

* **Descrizione**:  
  + Ordina, scambia, rinumera e “legame” tra testo/video/URL.
  + Assicura la coerenza StepImage (alt obbligatorio se URL non vuoto).

### **2.7 template\_builder/widgets.py**

**Import**:  
  
 import tkinter as tk

from tkinter import ttk

from .services import text as text\_service

from .services import images as image\_service

from .infrastructure import ui\_utils

from .assets import PLACEHOLDER\_COLOR

from typing import List

**Costanti**:  
  
 HAS\_DND = False

HAS\_TOOLTIP = False

PLACEHOLDER\_COLOR = "#888888"

**Classi e Funzioni**:  
  
 class \_Tooltip:

... # Gestione balloon on-hover

class TooltipMixin:

... # Abilita balloon on-hover per widget

class PlaceholderEntry(ttk.Entry, TooltipMixin):

def \_\_init\_\_(self, parent, placeholder: str, \*\*kwargs):

super().\_\_init\_\_(parent, \*\*kwargs)

self.placeholder = placeholder

self.insert(0, placeholder)

self['foreground'] = PLACEHOLDER\_COLOR

self.bind("<FocusIn>", self.\_clear\_placeholder)

self.bind("<FocusOut>", self.\_add\_placeholder)

def \_clear\_placeholder(self, event):

if self.get() == self.placeholder:

self.delete(0, tk.END)

self['foreground'] = "black"

def \_add\_placeholder(self, event):

if not self.get().strip():

self.insert(0, self.placeholder)

self['foreground'] = PLACEHOLDER\_COLOR

def get\_value(self) -> str:

val = self.get()

return "" if val == self.placeholder else val

def render\_html(self) -> str:

val = self.get\_value()

return text\_service.auto\_format(val, mode="p")

class PlaceholderSpinbox(ttk.Spinbox, TooltipMixin):

def \_\_init\_\_(self, parent, placeholder: str, from\_: int, to: int, command=None, \*\*kwargs):

super().\_\_init\_\_(parent, from\_=from\_, to=to, \*\*kwargs)

self.placeholder = placeholder

self.set(placeholder)

self['foreground'] = PLACEHOLDER\_COLOR

self.bind("<FocusIn>", self.\_clear\_placeholder)

self.bind("<FocusOut>", self.\_add\_placeholder)

if command:

self['command'] = command

def \_clear\_placeholder(self, event):

if self.get() == self.placeholder:

self.delete(0, tk.END)

self['foreground'] = "black"

def \_add\_placeholder(self, event):

if not self.get().strip():

self.set(self.placeholder)

self['foreground'] = PLACEHOLDER\_COLOR

def get\_value(self) -> int:

val = self.get()

try:

return int(val)

except ValueError:

return 0

class PlaceholderMultiTextField(tk.Frame, TooltipMixin):

def \_\_init\_\_(self, parent, placeholder: str, on\_change=None, \*\*kwargs):

super().\_\_init\_\_(parent, \*\*kwargs)

self.placeholder = placeholder

self.text = tk.Text(self, wrap='word', height=5)

self.text.insert("1.0", placeholder)

self.text['fg'] = PLACEHOLDER\_COLOR

self.text.bind("<FocusIn>", self.\_clear\_placeholder)

self.text.bind("<FocusOut>", self.\_add\_placeholder)

if on\_change:

self.text.bind("<KeyRelease>", lambda e: on\_change())

self.text.pack(fill='both', expand=True)

def \_clear\_placeholder(self, event):

if self.text.get("1.0", "end-1c") == self.placeholder:

self.text.delete("1.0", tk.END)

self.text['fg'] = "black"

def \_add\_placeholder(self, event):

if not self.text.get("1.0", "end-1c").strip():

self.text.insert("1.0", self.placeholder)

self.text['fg'] = PLACEHOLDER\_COLOR

def get\_value(self) -> str:

val = self.text.get("1.0", "end-1c")

return "" if val == self.placeholder else val

def render\_html(self) -> str:

return text\_service.auto\_format(self.get\_value(), mode="ul")

# Verifica presenza tkinterdnd2 per drag&drop

try:

from tkinterdnd2 import TkinterDnD

HAS\_DND = True

except ImportError:

HAS\_DND = False

class SortableImageRepeaterField(tk.Frame, TooltipMixin):

"""

Widget per gestire dinamicamente righe di URL immagini + ALT.

Supporta add/del/riordino e validazione live tramite bordi colorati.

Se HAS\_DND = True, abilita drag&drop di file immagine.

"""

def \_\_init\_\_(self, parent, on\_change=None, \*\*kwargs):

super().\_\_init\_\_(parent, \*\*kwargs)

self.rows: List[dict] = []

self.on\_change = on\_change

self.\_build\_gui()

def \_build\_gui(self):

self.header = tk.Frame(self)

tk.Label(self.header, text="Image URL").grid(row=0, column=0)

tk.Label(self.header, text="Alt Text").grid(row=0, column=1)

self.header.pack(fill='x')

self.list\_frame = tk.Frame(self)

self.list\_frame.pack(fill='both', expand=True)

self.\_add\_row()

def \_add\_row(self):

row\_frame = tk.Frame(self.list\_frame)

url\_entry = tk.Entry(row\_frame, width=40)

alt\_entry = tk.Entry(row\_frame, width=20)

url\_entry.grid(row=0, column=0, padx=5, pady=2)

alt\_entry.grid(row=0, column=1, padx=5, pady=2)

# Collegamenti per validazione

url\_entry.bind("<FocusOut>", lambda e: self.\_validate(url\_entry))

alt\_entry.bind("<KeyRelease>", lambda e: self.\_validate(url\_entry))

# Pulsanti di rimozione e spostamento

del\_btn = tk.Button(row\_frame, text="✕", command=lambda: self.\_del\_row(row\_frame))

up\_btn = tk.Button(row\_frame, text="↑", command=lambda: self.\_move\_row(row\_frame, -1))

down\_btn = tk.Button(row\_frame, text="↓", command=lambda: self.\_move\_row(row\_frame, +1))

del\_btn.grid(row=0, column=2, padx=2)

up\_btn.grid(row=0, column=3, padx=2)

down\_btn.grid(row=0, column=4, padx=2)

# Drag&Drop

if HAS\_DND:

url\_entry.drop\_target\_register(tk.DND\_FILES)

url\_entry.dnd\_bind('<<Drop>>', lambda e: self.\_on\_dnd(e, url\_entry))

row\_frame.pack(fill='x', pady=1)

self.rows.append({"frame": row\_frame, "url": url\_entry, "alt": alt\_entry})

def \_del\_row(self, frame):

for item in self.rows:

if item["frame"] == frame:

item["frame"].destroy()

self.rows.remove(item)

break

if self.on\_change:

self.on\_change()

def \_move\_row(self, frame, direction: int):

idx = next(i for i, item in enumerate(self.rows) if item["frame"] == frame)

new\_idx = idx + direction

if 0 <= new\_idx < len(self.rows):

self.rows[idx], self.rows[new\_idx] = self.rows[new\_idx], self.rows[idx]

self.\_repack\_rows()

if self.on\_change:

self.on\_change()

def \_repack\_rows(self):

for item in self.rows:

item["frame"].pack\_forget()

for item in self.rows:

item["frame"].pack(fill='x', pady=1)

def \_validate(self, url\_entry):

url = url\_entry.get().strip()

ok = image\_service.validate\_url(url)

\_apply\_border(url\_entry, ok)

if self.on\_change:

self.on\_change()

def \_on\_dnd(self, event, url\_entry):

data = \_split\_dnd\_event\_data(event.data)

if data:

url\_entry.delete(0, tk.END)

url\_entry.insert(0, data[0])

self.\_validate(url\_entry)

* **Descrizione estesa**:  
  + Tutti i widget ereditano TooltipMixin per help on-hover (testi da \_HELP\_DEFAULTS).
  + SortableImageRepeaterField mantiene self.rows come lista di dizionari contenenti frame, url (tk.Entry), alt (tk.Entry).
  + Ogni url\_entry bindato a \_validate() che usa image\_service.validate\_url() e ui\_utils.\_apply\_border() per evidenziare la validità.
  + Se HAS\_DND=True, url\_entry abilitato a ricevere drop di file (metodo \_on\_dnd).

### **2.8 template\_builder/services/images.py**

import os, base64

from typing import List, Dict

from io import BytesIO

try:

from PIL import Image

HAS\_PIL = True

except ImportError:

Image = None

HAS\_PIL = False

from .text import smart\_paste

PLACEHOLDER\_BASE\_URL = "https://via.placeholder.com"

MIME\_TYPES = {

".png": "image/png",

".jpg": "image/jpeg",

".jpeg": "image/jpeg",

".gif": "image/gif"

}

\_\_all\_\_ = [

"guess\_grid", "generate\_placeholders", "encode\_file\_to\_data\_uri",

"paths\_to\_html\_grid", "images\_to\_html",

"smart\_paste\_images", "validate\_url", "fetch\_metadata"

]

def guess\_grid(n\_images: int, cols: int) -> (int,int):

"""

Calcola righe e colonne per n\_images dato un numero preferito di colonne.

Se n\_images % cols != 0, crea una riga parziale in fondo.

"""

rows = n\_images // cols + (1 if n\_images % cols else 0)

return rows, cols

def generate\_placeholders(n\_images: int) -> List[str]:

"""

Ritorna liste di segnaposto Jinja2: ['{{IMG1}}','{{IMG2}}',…,'{{IMGn}}']

"""

return [f"{{{{IMG{i+1}}}}}" for i in range(n\_images)]

def encode\_file\_to\_data\_uri(path: str) -> str:

"""

Converte un file in Data-URI base64.

Se PIL disponibile, aggiunge dimensioni width/height.

"""

ext = os.path.splitext(path)[1].lower()

mime = MIME\_TYPES.get(ext, "application/octet-stream")

if HAS\_PIL:

img = Image.open(path)

buf = BytesIO()

img.save(buf, format=img.format)

b64 = base64.b64encode(buf.getvalue()).decode("utf-8")

else:

with open(path, "rb") as f:

b64 = base64.b64encode(f.read()).decode("utf-8")

return f"data:{mime};base64,{b64}"

def \_make\_img\_tag(src: str, alt: str) -> str:

"""

Genera un tag <img> con attributi src e alt; se HAS\_PIL=True, imposta anche width/height.

"""

if HAS\_PIL and src.startswith(("http://","https://")):

# Non gestito in questo stub; usar fetch\_metadata per immagini locali

return f'<img src="{src}" alt="{alt}">'

return f'<img src="{src}" alt="{alt}">'

def paths\_to\_html\_grid(paths: List[str], cols: int, inline: bool=False, alt\_texts: List[str]=None) -> str:

"""

Genera HTML <table> di immagini, `cols` colonne per riga.

Se inline=True, converte puntamenti locali in Data-URI.

`alt\_texts`: lista ALT per ogni immagine

"""

alt\_texts = alt\_texts or [""]\*len(paths)

html\_rows = []

for i in range(0, len(paths), cols):

row\_paths = paths[i:i+cols]

row\_alts = alt\_texts[i:i+cols]

cols\_html = []

for p, alt in zip(row\_paths, row\_alts):

if inline and os.path.exists(p):

src = encode\_file\_to\_data\_uri(p)

else:

src = p

cols\_html.append(f"<td>{\_make\_img\_tag(src, alt)}</td>")

html\_rows.append(f"<tr>{''.join(cols\_html)}</tr>")

return f"<table>{''.join(html\_rows)}</table>"

# Alias: images\_to\_html

images\_to\_html = paths\_to\_html\_grid

def smart\_paste\_images(raw: str) -> List[str]:

"""

Ritorna lista di URL immagine da raw (split su ';' o newline).

"""

return smart\_paste(raw)

def validate\_url(url: str) -> bool:

"""

Validatore di URL (wrapper su `infrastructure.validators.validate\_url`).

"""

from ..infrastructure.validators import validate\_url as \_v

return \_v(url)

def fetch\_metadata(path: str) -> Dict[str,int]:

"""

Ritorna dizionario metadata immagine: {'width':…, 'height':…}.

Se HAS\_PIL=False o file non esiste, ritorna {}.

"""

if HAS\_PIL and os.path.exists(path):

img = Image.open(path)

return {"width": img.width, "height": img.height}

return {}

### **2.9 template\_builder/services/text.py**

import re, html

from typing import List, Set

from ..assets import PLACEHOLDER\_RGX

\_\_all\_\_ = [

"smart\_paste", "auto\_format", "extract\_placeholders",

"images\_to\_html", "get\_field\_help"

]

# Mappa chiavi→testo tooltip

\_HELP\_DEFAULTS = {

"TITLE": "Titolo principale del template",

"DESCRIPTION": "Descrizione breve or sotto-titolo",

"IMG\_URL": "URL di un'immagine (http://…)",

"ALT\_TEXT": "Testo alternativo per l'immagine",

"COLS\_DESC": "Numero di colonne per la galleria descrizione",

"COLS\_REC": "Numero di colonne per la galleria ricetta",

# … altri default

}

BULLET\_RGX = re.compile(r"^\s\*[\\*\-]\s+")

HTML\_ALLOWED\_TAGS = {"b","i","ul","li","p","h1","h2","h3"}

def \_split\_lines(raw: str) -> List[str]:

"""

Split raw input su newline o ';', rimuove spazi vuoti.

"""

lines = re.split(r"[;\n]+", raw)

return [ln.strip() for ln in lines if ln.strip()]

def smart\_paste(raw: str) -> List[str]:

"""

Suddivide righe via ';' o newline e rimuove stringhe vuote.

"""

return \_split\_lines(raw)

def \_format\_line(line: str) -> str:

"""

Trasforma singola riga in HTML leggero:

- \*\*bold\*\* → <strong>bold</strong>

- \*italic\* → <em>italic</em>

- appoggia ad html.escape per sicurezza

"""

# Escape iniziale

esc = html.escape(line)

# Bold

esc = re.sub(r"\\*\\*(.+?)\\*\\*", r"<strong>\1</strong>", esc)

# Italic

esc = re.sub(r"\\*(.+?)\\*", r"<em>\1</em>", esc)

return esc

def auto\_format(text: str, mode: str="ul") -> str:

"""

Se mode="ul": ogni riga diventa <li>…</li> e incapsula in <ul>…</ul>.

Se mode="p": ogni riga diventa <p>…</p>.

"""

lines = \_split\_lines(text)

if not lines:

return ""

if mode == "ul":

items = "".join(f"<li>{\_format\_line(ln)}</li>" for ln in lines)

return f"<ul>{items}</ul>"

else:

paras = "".join(f"<p>{\_format\_line(ln)}</p>" for ln in lines)

return paras

def extract\_placeholders(html\_src: str) -> Set[str]:

"""

Usa PLACEHOLDER\_RGX per trovare tutti i segnaposto {{TAG}} in html\_src.

Ritorna set di tag estratti (es. {"TITLE","STEP1\_IMG\_ALT",…}).

"""

return set(match.group(1) for match in re.finditer(PLACEHOLDER\_RGX, html\_src))

def images\_to\_html(rows: int, cols: int) -> str:

"""

Genera fallback HTML per galleria di placeholder immagini numerate.

Esempio: <table><tr><td>{{IMG1}}</td><td>{{IMG2}}</td>…</tr></table>

"""

placeholders = [f"{{{{IMG{i+1}}}}}" for i in range(rows\*cols)]

rows\_html = []

for i in range(0, len(placeholders), cols):

row = placeholders[i:i+cols]

cells = "".join(f"<td>{ph}</td>" for ph in row)

rows\_html.append(f"<tr>{cells}</tr>")

return f"<table>{''.join(rows\_html)}</table>"

def get\_field\_help(key: str) -> str:

"""

Restituisce la stringa di help associata a `key` (dal dizionario \_HELP\_DEFAULTS).

Se key non in dictionary, ritorna stringa vuota.

"""

return \_HELP\_DEFAULTS.get(key, "")

### **2.10 template\_builder/services/storage.py**

import os, json

from datetime import datetime

from typing import Dict, Any, Union

from pathlib import Path

from ..step\_image import bind\_steps

SCHEMA\_VERSION = 2

\_BASE\_DIR = os.path.expanduser("~/.template\_builder")

\_HISTORY\_DIR = os.path.join(\_BASE\_DIR, "history")

AUTOSAVE\_LIMIT = 50

\_\_all\_\_ = ["load\_recipe", "quick\_save", "export\_html", "UndoRedoStack"]

class UndoRedoStack:

def \_\_init\_\_(self):

self.\_undo\_stack = []

self.\_redo\_stack = []

def push(self, state: Dict[str, Any]) -> None:

import copy

self.\_undo\_stack.append(copy.deepcopy(state))

self.\_redo\_stack.clear()

if len(self.\_undo\_stack) > AUTOSAVE\_LIMIT:

self.\_undo\_stack.pop(0)

def undo(self) -> Dict[str, Any]:

if len(self.\_undo\_stack) > 1:

last = self.\_undo\_stack.pop()

self.\_redo\_stack.append(last)

return self.\_undo\_stack[-1]

return self.\_undo\_stack[0] if self.\_undo\_stack else {}

def redo(self) -> Dict[str, Any]:

if self.\_redo\_stack:

state = self.\_redo\_stack.pop()

self.\_undo\_stack.append(state)

return state

return self.\_undo\_stack[-1] if self.\_undo\_stack else {}

def \_timestamp() -> str:

return datetime.now().strftime("%Y%m%d\_%H%M%S")

def quick\_save(state: Dict[str, Any]) -> Path:

"""

Salva `state` in file JSON con nome recipe\_<timestamp>.json in \_HISTORY\_DIR.

Ritorna Path creato.

"""

os.makedirs(\_HISTORY\_DIR, exist\_ok=True)

ts = \_timestamp()

filename = f"recipe\_{ts}.json"

filepath = os.path.join(\_HISTORY\_DIR, filename)

with open(filepath, "w", encoding="utf-8") as f:

json.dump(state, f, ensure\_ascii=False, indent=2)

return Path(filepath)

def load\_recipe(path: Union[str, Path]) -> Dict[str, Any]:

"""

Carica JSON da `path`. Se `SCHEMA\_VERSION == 1`, applica \_migrate\_v1\_to\_v2().

Ritorna dict contesto (chiamato da builder\_core.load\_recipe()).

"""

with open(path, "r", encoding="utf-8") as f:

data = json.load(f)

version = data.get("SCHEMA\_VERSION", 1)

if version < SCHEMA\_VERSION:

data = \_migrate\_v1\_to\_v2(data)

return data

def \_migrate\_v1\_to\_v2(old: Dict[str, Any]) -> Dict[str, Any]:

"""

Converte JSON legacy (schema v1) in schema v2:

- Estrae liste OLD\_IMAGES = old["IMAGES\_STEP"], OLD\_ALTS = old["ALTS\_STEP"], OLD\_TEXTS = old["TEXTS\_STEP"]

- Crea `STEPS = [ StepImage(img, alt, text, order=i+1).to\_dict() for i, … ]`

- Elimina campi legacy (`"IMAGES\_STEP"`, `"ALTS\_STEP"`, `"TEXTS\_STEP"`)

- Imposta data["SCHEMA\_VERSION"] = 2

"""

old\_images = old.get("IMAGES\_STEP", [])

old\_alts = old.get("ALTS\_STEP", [])

old\_texts = old.get("TEXTS\_STEP", [])

steps = []

for i, (img, alt, txt) in enumerate(zip(old\_images, old\_alts, old\_texts)):

# bind\_steps verifica alt/IMG coerenti

si = bind\_steps([txt], [img], [alt])[0]

steps.append(si.to\_dict())

new = {

\*\*old,

"SCHEMA\_VERSION": 2,

"STEPS": steps

}

new.pop("IMAGES\_STEP", None)

new.pop("ALTS\_STEP", None)

new.pop("TEXTS\_STEP", None)

return new

def export\_html(ctx: Dict[str, Any], template\_path: str, \*\*env\_kw) -> str:

"""

Effettua render Jinja2 di `template\_path` con contesto `ctx`.

Se Jinja2 non installato, solleva ImportError.

"""

try:

from jinja2 import Environment, FileSystemLoader, select\_autoescape

except ImportError:

raise ImportError("Jinja2 non è installato: impossibile eseguire export\_html().")

template\_dir, template\_file = os.path.split(template\_path)

env = Environment(

loader=FileSystemLoader(template\_dir or "."),

autoescape=select\_autoescape(["html", "xml"]),

\*\*env\_kw

)

tmpl = env.get\_template(template\_file)

return tmpl.render(\*\*ctx)

### **2.11 template\_builder/infrastructure/preview\_engine.py**

import tkinter as tk

from tkinter.scrolledtext import ScrolledText

class PreviewEngine:

def \_\_init\_\_(self, parent, width=600, height=400):

"""

Crea un frame per anteprima HTML:

- Se `tkinterweb.HtmlFrame` disponibile, lo usa

- Altrimenti, usa `ScrolledText`

- Se nessun display, setta self.\_headless = True

"""

self.\_headless = not self.\_display\_available()

if self.\_headless:

self.widget = None

return

try:

from tkinterweb import HtmlFrame

self.widget = HtmlFrame(parent, horizontal\_scrollbar="auto")

except ImportError:

self.widget = ScrolledText(parent)

self.widget.configure(width=width, height=height)

self.widget.pack(fill="both", expand=True)

def \_display\_available(self) -> bool:

"""

Verifica se è disponibile un display (su \*nix: variabile $DISPLAY).

"""

try:

root = tk.Tk()

root.destroy()

return True

except Exception:

return False

def render(self, html\_source: str) -> None:

"""

Se non headless, aggiorna widget con `html\_source`. In caso di HtmlFrame, chiama `load\_html()`.

Se ScrolledText, scrive come testo semplice.

"""

if self.\_headless or not self.widget:

return

if hasattr(self.widget, "load\_html"):

self.widget.load\_html(html\_source)

else:

self.widget.delete("1.0", "end")

self.widget.insert("1.0", html\_source)

def collect\_context(self) -> dict:

"""

Stub API: librerie legacy chiamavano questo metodo per raccogliere contesto.

Attualmente non usato; ritorna dict vuoto.

"""

return {}

def \_safe(self, name: str):

"""

Import dinamico modulare (es. "tkinterweb.HtmlFrame", "cefpython3.CefBrowser").

Se non disponibile, ritorna None.

"""

comps = name.split('.')

mod = \_\_import\_\_(".".join(comps[:-1]), fromlist=[comps[-1]])

return getattr(mod, comps[-1], None)

### **2.12 template\_builder/infrastructure/ui\_utils.py**

import tkinter as tk

from tkinter import messagebox

import platform

KEY\_CTRL = "Control"

KEY\_CMD = "Command"

OS\_NAME = platform.system().lower()

def show\_info(msg: str) -> None:

messagebox.showinfo("Info", msg)

def show\_error(msg: str) -> None:

messagebox.showerror("Error", msg)

def show\_warning(msg: str) -> None:

messagebox.showwarning("Warning", msg)

def center\_window(win: tk.Tk) -> None:

win.update\_idletasks()

w = win.winfo\_width()

h = win.winfo\_height()

ws = win.winfo\_screenwidth()

hs = win.winfo\_screenheight()

x = (ws // 2) - (w // 2)

y = (hs // 2) - (h // 2)

win.geometry(f"{w}x{h}+{x}+{y}")

def bind\_mousewheel(widget: tk.Widget) -> None:

"""

Associa eventi <MouseWheel> (Windows), <Button-4>/<Button-5> (Linux) a scroll di `widget`.

"""

if OS\_NAME == "darwin": # MacOS

widget.bind\_all("<MouseWheel>", lambda e: widget.yview\_scroll(-1 if e.delta>0 else 1, "units"))

elif OS\_NAME == "linux":

widget.bind\_all("<Button-4>", lambda e: widget.yview\_scroll(-1, "units"))

widget.bind\_all("<Button-5>", lambda e: widget.yview\_scroll(1, "units"))

else: # windows

widget.bind\_all("<MouseWheel>", lambda e: widget.yview\_scroll(-1 if e.delta>0 else 1, "units"))

def ask\_yes\_no(msg: str) -> bool:

return messagebox.askyesno("Conferma", msg)

### **2.13 template\_builder/infrastructure/validators.py**

import re

from datetime import datetime

URL\_RGX = re.compile(r"https?://[^\s]+")

EMAIL\_RGX = re.compile(r"[^@]+@[^@]+\.[^@]+")

DATE\_RGX = re.compile(r"\d{4}-\d{2}-\d{2}")

def validate\_url(url: str) -> bool:

return bool(URL\_RGX.match(url))

def validate\_email(email: str) -> bool:

return bool(EMAIL\_RGX.match(email))

def validate\_date(date\_str: str) -> bool:

try:

datetime.strptime(date\_str, "%Y-%m-%d")

return True

except ValueError:

return False

def validate\_nonempty(value: str) -> bool:

return bool(value.strip())

### **2.14 template\_builder/templates/…**

Cartella disponibile con **esempio di template Jinja2**. Esempi sintetici:

<!-- ebay\_template\_modern\_full.html -->

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>{{ TITLE }}</title>

</head>

<body>

<h1>{{ TITLE }}</h1>

<p>{{ DESCRIPTION }}</p>

<img src="{{ HERO\_IMAGE\_SRC }}" alt="{{ HERO\_IMAGE\_ALT }}">

<div class="gallery">

{% for step in STEPS %}

<div class="step">

<img src="{{ step.IMG\_SRC }}" alt="{{ step.IMG\_ALT }}">

<p>{{ step.TEXT }}</p>

</div>

{% endfor %}

</div>

</body>

</html>

Ogni template deve usare segnaposto esatti (TITLE, DESCRIPTION, HERO\_IMAGE\_SRC, HERO\_IMAGE\_ALT, STEPS, STEPn\_IMG\_ALT, etc.) per funzionare correttamente con il contesto generato da builder\_core.\_collect().

## **8. Migrazione dei Dati (Schema JSON)**

Come spiegato in sezione 5 (**Fase di verifica legacy**), il formato JSON dei progetti è passato **da v1→v2**:

**Formato v1** esemplificativo (codice legacy):  
  
 {

"SCHEMA\_VERSION": 1,

"Step1": "Titolo primo passo",

"IMAGES\_STEP": ["http://example.com/img1.jpg", "http://example.com/img2.jpg"],

"ALTS\_STEP": ["Alt1","Alt2"],

"TEXTS\_STEP": ["Passo1", "Passo2"],

"COLS\_DESC": 2,

"COLS\_REC": 1,

"TITLE": "Mia Ricetta",

"DESCRIPTION": "Descrizione breve"

// … altri campi hero

}

* **Conversione v1→v2** (services/storage.py.\_migrate\_v1\_to\_v2()):  
  + Legge "IMAGES\_STEP", "ALTS\_STEP", "TEXTS\_STEP".
  + Chiama step\_image.bind\_steps(texts, images, alts) per creare lista List[StepImage].
  + Crea nuovo array STEPS = [ {@"IMG\_SRC":…,@"IMG\_ALT":…,@"TEXT":…,@"ORDER":…}, … ].
  + Elimina i campi legacy "IMAGES\_STEP", "ALTS\_STEP", "TEXTS\_STEP".
  + Imposta "SCHEMA\_VERSION": 2.

**Formato v2** esemplificativo (codice attuale):  
  
 {

"SCHEMA\_VERSION": 2,

"STEPS": [

{"IMG\_SRC":"http://…","IMG\_ALT":"Alt1","TEXT":"Passo1","ORDER":1},

{"IMG\_SRC":"http://…","IMG\_ALT":"Alt2","TEXT":"Passo2","ORDER":2}

],

"STEP1\_IMG\_ALT": "Alt1",

"STEP2\_IMG\_ALT": "Alt2",

"COLS\_DESC": 2,

"COLS\_REC": 1,

"TITLE": "Mia Ricetta",

"DESCRIPTION": "Descrizione breve",

"HERO\_IMAGE\_SRC": "http://…",

"HERO\_IMAGE\_ALT": "Hero Alt",

"INTRO": "Testo introduttivo"

// … altri campi hero

}

## **9. Documentazione & Change-Log Rigenerati**

### **9.1 README.md (come riportato in sezione 7.1)**

* **Panoramica Progetto**, **Struttura ad albero**, **Come installare**, **Come avviare**, **Funzionalità principali**, **Dipendenze**, **Esempio di utilizzo**, **Sezione Contributi**.
* Citazioni originali mantenute (README corrente, ARCHITECTURE) .

### **9.2 ARCHITECTURE.md (come riportato in sezione 7.2)**

* **Albero moduli**, **Descrizione moduli**, **Mappa import**, **Call-graph**, **Convenzioni naming**.
* Citazioni originali (ARCHITECTURE.md esistente, file sorgente) .

### **9.3 CHANGELOG.md (Scheda Tecnica)**

# Changelog

## [1.0.0] – 2025-06-06

- \*\*Nuovi moduli\*\*: aggiunte le dataclass `Hero` e `GalleryRow` per gestire sezione hero (titolo, intro, immagine) e righe di galleria (max 3 immagini). :contentReference[oaicite:76]{index=76}

- \*\*Anteprima avanzato\*\*: `PreviewEngine` con fallback a `ScrolledText` in assenza di `tkinterweb`. :contentReference[oaicite:77]{index=77}

- \*\*Undo/Redo completo\*\*: `UndoRedoStack` integrato in UI, binding Ctrl+Z/Y e Cmd+Z/Shift+Z. :contentReference[oaicite:78]{index=78}

- \*\*Drag&Drop immagini\*\*: `SortableImageRepeaterField` usa `tkinterdnd2` se installato, fallback a pulsante “+ Add”. :contentReference[oaicite:79]{index=79}

- \*\*Validazione live\*\*: campi URL/ALT evidenziati con border verde/rosso; fix placeholder persistenti al focus. :contentReference[oaicite:80]{index=80}

- \*\*Tooltips\*\*: suggerimenti contestuali on-hover basati su `text.get\_field\_help()`. :contentReference[oaicite:81]{index=81}

- \*\*Persistenza & Migrazione\*\*:

- `quick\_save(state)` salva snapshot JSON in `~/.template\_builder/history/`.

- `load\_recipe(path)` carica progetto legacy v1→v2 con `\_migrate\_v1\_to\_v2()`.

- `export\_html(ctx, template\_path)` effettua rendering Jinja2 (ImportError se mancante). :contentReference[oaicite:82]{index=82}

- \*\*Rifattorizzazione generale\*\*: separazione servizi (`images.py`, `text.py`, `storage.py`), isolamento code legacy in `infrastructure/`, aggiornamenti nei widget, rimozione stub obsoleti. :contentReference[oaicite:83]{index=83}

## [0.1.0] – <data precedente>

- \*\*Correzioni legacy\*\*:

- Risolti bug storici (anteprima HTML complessi, undo/redo, drag&drop immagini, placeholder sui campi, validazione live, tooltip). :contentReference[oaicite:84]{index=84}

- Migrazione automatica dati: `\_migrate\_v1\_to\_v2()` per convertire progetti v1 al nuovo schema. :contentReference[oaicite:85]{index=85}

- \*\*Creazione\*\*:

- Skeleton & Core: pacchetto modulare (`builder\_core.py`, `widgets.py`, `services/`, `infrastructure/`, `model.py`, `filters.py`, `assets.py`).

- Test headless: copertura import su CI.

- CI GitHub Actions: `pip install -e .[dnd,webpreview,test]`; `pytest -q`.

- Tag v0.1.0 rilasciato con CHANGELOG placeholder. :contentReference[oaicite:86]{index=86}

## [0.0.1] – <prima versione>

- \*\*Inizializzazione progetto\*\*:

- Creazione repository, primi file `README.md`, `ARCHITECTURE.md`, `pyproject.toml`, `tests/`, `legacy/` (monolite).

- Setup CI base, test di import e stub minimal.

- Struttura di package `template\_builder/` (bozza di `builder\_core.py`, `widgets.py`, `model.py`).

---

* **Note**: rimuovere l’attuale placeholder “# GENERARE COERENTEMENTE AL REPOSITORY…”.
* **Fonti**:  
  + Git log, commit reali, file modificati.
  + Documenti di deep research (v. turn0file1).

### **9.4 ROADMAP.md (Guida Completa per Sviluppatore C)**

# Roadmap per Completare Template Builder (Fase C)

Questa roadmap è destinata al nuovo sviluppatore C. Include dettagli su:

- Refactoring attuale

- Migrazione feature legacy mancanti

- Test & CI/CD

- Packaging & rilascio

- Documentazione

---

## 1. Prerequisiti di Base

1. Clonare il repository `main` e assicurarsi che la build iniziale sia verde:

```bash

git clone https://github.com/polsi05/template\_builder.git

cd template\_builder

python3 -m venv venv

source venv/bin/activate # Linux/Mac

venv\Scripts\activate # Windows

pip install -e .[test,dnd,webpreview]

pytest -q # deve passare tutto

python -m template\_builder # verifica avvio app (anche headless)

1. Leggere la documentazione generata:  
   * README.md (sezione “How to Install/Run”)
   * ARCHITECTURE.md per comprendere struttura e dipendenze
   * CHANGELOG.md per conoscere storia release
   * docs/migration\_summary.md per issue legacy rimanenti

## **2. Refactoring e Pulizia del Codice**

1. **Rimuovere cartella legacy/**
   * Il codice legacy (monolitico) non è più utilizzato; archiviare in backup se necessario.
2. **Verificare e uniformare naming e stile PEP8**
   * Usare flake8 o pylint per individuare warning.
   * Correggere import inutilizzati, rimuovere commenti obsoleti.
3. **Aggiornare pyproject.toml**
   * Completare metadata: version = "1.0.0", description, author, license = "MIT", keywords.

Aggiungere entry point CLI:  
  
 [project.entry-points.console\_scripts]

template\_builder = "template\_builder.builder\_core:main"

1. **Riorganizzare directory tests/**
   * Consolidare test duplicati.
   * Aggiungere test per i widget mancanti (URL/ALT binding).

## **3. Migrazione & Implementazione Feature Legacy Mancanti**

### **3.1 F9 – Smart-scroll notebook**

**Obiettivo**: Abilitare lo scroll fluido anche nei singoli frame/tab (Canvas interni).

* Il binding corrente (ui\_utils.bind\_mousewheel(self.nb)) associa la rotella solo al Notebook.
* **Azioni**:  
  + In TemplateBuilderApp.\_build\_ui(), per ogni tab CFD (e.g. “Images”, “Recipe”), recuperare il Canvas interno (che contiene i widget).
  + Chiamare ui\_utils.bind\_mousewheel(canvas) in aggiunta a self.nb.
  + Verificare con pytest il test test\_smart\_scroll.py esistente; in caso di fallimento, adeguare test o logica binding.
* **Test**:  
  + Aggiornare tests/test\_smart\_scroll.py per simulare un evento <MouseWheel> su tab specifici.
  + Confermare che lo scroll interno effettivamente si muova.

### **3.2 F10 – UI polish / tema scuro**

**Obiettivo**: Integrare un tema scuro completo, preferibilmente tramite ttkbootstrap.

Attualmente:  
  
 try:

from ttkbootstrap import Style

Style('darkly')

except Exception:

pass

* **Azioni**:  
  + Rendere opzionale la scelta del tema tramite flag CLI o menu “Settings → Theme”:  
    - Aggiungere voce di menu “View → Tema → [Light|Dark]”.
    - Se “Dark” selezionato e ttkbootstrap installato, Style('darkly'), altrimenti popup di avviso.
  + Personalizzare alcuni widget chiave (colori assets.PALETTE):  
    - Esempio: cambiare colore di sfondo di PlaceholderEntry e SortableImageRepeaterField in base al tema.
  + Aggiungere test headless:  
    - tests/test\_theme.py che simuli import ttkbootstrap e verifichi che Style('darkly') non crashi.
* **Test**:  
  + Verificare che UI rispetti colori darkly (leggere proprietà bg, fg dei widget).
  + In assenza di ttkbootstrap, il tema “Light” resti invariato.

### **3.3 F11 – Diff-validator segnaposto (Audit segnaposti automatico)**

**Obiettivo**: Integrare la validazione placeholder in fase di caricamento del template (reload\_template()) e colorare i widget corrispondenti.

* Attualmente:  
  + reload\_template() carica JSON e imposta valori nei widget, poi chiama update\_preview().
  + audit\_placeholders() è un’azione manuale.
* **Azioni**:  
  + In reload\_template(): dopo aver applicato lo stato ai widget (\_apply\_state\_to\_widgets()), chiamare audit\_placeholders(auto=True).
  + Modificare audit\_placeholders(auto=False) in audit\_placeholders(auto: bool=False) per:  
    - Se auto=True, anziché mostrare dialogo show\_info(), evidenziare i campi GUI con segnaposto mancanti/extra:  
      * Per ogni segnaposto ph estratto da template (extract\_placeholders()),  
        + Se ph non in chiavi contesto data, cercare widget corrispondente (es. self.fields[ph] se esiste) e applicare \_apply\_border(widget, False).
      * Per chiavi contesto senza ph nel template, colorare self.fields[key] in grigio chiaro o disabilitato.
  + Aggiungere test tests/test\_audit\_placeholders\_auto.py che carichi progetto di esempio e verifichi che i widget siano colorati (controllo dei colori border).
* **Test**:  
  + Progetto JSON con segnaposto mancante (es. {{ MISSING }}).
  + Chiamare load\_recipe(file.json), verificare che entry del campo corrispondente si colori di rosso/rosso chiaro.

### **3.4 F12 – Auto-preview su ogni modifica**

**Obiettivo**: Collegare ogni widget di input (tk.Entry per URL/ALT) a update\_preview().

* Attualmente:  
  + PlaceholderMultiTextField, spinbox colonne → binding a update\_preview().
  + I campi URL/ALT di immagini (SortableImageRepeaterField) hanno binding solo su FocusOut per validazione, ma non chiamano update\_preview() su ogni cambio.
* **Azioni**:

In SortableImageRepeaterField.\_build\_gui(), aggiungere per ogni url\_entry e alt\_entry:  
  
 url\_entry.bind("<KeyRelease>", lambda e: self.on\_change())

alt\_entry.bind("<KeyRelease>", lambda e: self.on\_change())

* + dove self.on\_change = builder\_core.update\_preview.
  + Verificare nei metodi \_add\_row(), \_del\_row(), \_move\_row() di chiamare self.on\_change() dopo ogni modifica delle righe.
  + Aggiungere test tests/test\_live\_validation.py che simuli digitazione in url\_entry e verifichi che update\_preview() sia effettivamente chiamato (può usare un flag di “render\_html chiamato”).
* **Test**:  
  + Simulare inserimento URL in SortableImageRepeaterField e verificare anteprima aggiornata in PreviewEngine.

### **3.5 F16 – Verifica implementazione selettore colonne**

**Obiettivo**: Assicurarsi che i template Jinja usino COLS\_DESC e COLS\_REC.

* Attualmente:  
  + \_collect() inserisce COLS\_DESC, COLS\_REC nel contesto.
  + I template di esempio attuali **non** usano obbligatoriamente queste chiavi (vari template diversi).
* **Azioni**:  
  + Unificare i template (in templates/) per includere COLS\_DESC e COLS\_REC nei cicli immagini.

Esempio, in template\_ebay+ricetta.html:  
  
 {% set rows, cols = COLS\_REC, COLS\_REC %}

<div class="gallery-recipes">

{% for row in STEPS|batch(cols) %}

<div class="row">

{% for step in row %}

<div class="col">

<img src="{{ step.IMG\_SRC }}" alt="{{ step.IMG\_ALT }}" style="width:calc(100%/{{ cols }});">

<p>{{ step.TEXT }}</p>

</div>

{% endfor %}

</div>

{% endfor %}

</div>

* + Aggiungere test tests/test\_templates\_cols.py che verifichi il rendering con diversi valori di COLS\_DESC/COLS\_REC.
* **Test**:  
  + Caricare contesto esempio con COLS\_REC=3 in export\_html() e verificare output HTML contiene <div class="col"> in numero coerente.

### **3.6 F18 – Engine di rendering modulare (TkinterWeb / CEF / Browser)**

**Obiettivo**: Estendere PreviewEngine per supportare più backend e aggiungere opzione in UI.

* Attualmente:  
  + PreviewEngine usa solo tkinterweb.HtmlFrame o fallback ScrolledText.
* **Azioni**:  
  + In preview\_engine.py, creare classi separate per ogni backend:  
    - class HtmlFrameEngine(PreviewEngine): usa tkinterweb.HtmlFrame.
    - class CefEngine(PreviewEngine): usa cefpython3 (se installato) per creare browser embedded.
    - class WebBrowserEngine(PreviewEngine): usa modulo webbrowser per aprire pagina in browser esterno.
  + In builder\_core.py, creare menu “View → Engine → [HtmlFrame|CEF|Browser]”.  
    - Selezione dell’engine aggiorna self.preview = SelectedEngine(...).
  + Gestire fallback: se cefpython3 mancante e utente seleziona “CEF”, mostra show\_error("CEF non installato").
  + Aggiungere test:  
    - tests/test\_preview\_engine\_import.py verifica che HtmlFrameEngine importi senza errori se tkinterweb installato.
    - Simulazione cefpython3 import (possibile mock) per test CefEngine.
* **Test**:  
  + Verificare menu selezione engine e che self.preview.render(html) chiami metodo corretto.

## **10. Test & CI/CD**

### **10.1 Estendere Suite di Test**

* **Test di copertura**: aspirare a 100% di copertura per moduli “puri”: services/\*, model.py, step\_image.py, filters.py, assets.py, validators.py.
* **Test headless GUI**:  
  + Usare pytest-xvfb per testare import di builder\_core, widgets, preview\_engine, ui\_utils.
  + Testare funzionalità live su SortableImageRepeaterField (validazione live, drag&drop simulato).
  + Test cane (smoke tests) su TemplateBuilderApp.edit\_undo(), edit\_redo(), update\_preview().
* **Test legacy**:  
  + test\_smart\_scroll.py: verificare scroll su tab specifici.
  + test\_audit\_placeholders.py: verificare colorazione automatica dei widget.
  + test\_live\_validation.py: verificare che update\_preview() venga chiamato su URL/ALT.
  + test\_preview\_engine.py, test\_preview\_engine\_import.py: test import tkinterweb, fallback su ScrolledText.
  + test\_storage\_migration.py: caricare JSON v1 e verificare output contesto v2.
  + test\_templates\_cols.py: verificare uso di COLS\_DESC, COLS\_REC in template.

### **10.2 Integrazione CI (GitHub Actions)**

**Workflow** ci.yml:  
  
 name: CI

on:

push:

branches: [ main ]

pull\_request:

branches: [ main ]

jobs:

build-test:

runs-on: ubuntu-latest

services:

xvfb:

image: "selenium/standalone-firefox:latest"

steps:

- uses: actions/checkout@v2

- name: Set up Python

uses: actions/setup-python@v2

with:

python-version: '3.10'

- name: Install dependencies

run: |

python -m pip install --upgrade pip

pip install -e .[test,dnd,webpreview]

pip install flake8 coverage

- name: Lint with flake8

run: |

flake8 template\_builder tests

- name: Run tests

env:

DISPLAY: ":99"

run: |

pytest -q --maxfail=1 --disable-warnings

- name: Coverage report

run: |

coverage run -m pytest

coverage xml

- name: Upload coverage to Codecov

uses: codecov/codecov-action@v2

* **Obiettivi**:  
  + Ogni PR deve passare lint e test headless (xvfb).
  + Generare report coverage e inviarlo a Codecov.
  + Bloccare merge se copertura < 95%.

## **11. Packaging & Rilascio**

### **11.1 pyproject.toml**

**Metadata**:  
  
 [project]

name = "template\_builder"

version = "1.0.0"

description = "Toolkit GUI Python per creare template HTML con anteprima live, undo/redo e gestione immagini"

authors = [

{ name="Autore", email="autore@example.com" }

]

license = { file="LICENSE" }

readme = "README.md"

keywords = ["template", "GUI", "Jinja2", "Tkinter", "email"]

classifiers = [

"Programming Language :: Python :: 3.10",

"License :: OSI Approved :: MIT License",

"Operating System :: OS Independent"

]

dependencies = [

"jinja2>=3.0",

"Pillow>=8.0"

]

optional-dependencies = {

dnd = ["tkinterdnd2>=0.3"],

webpreview = ["tkinterweb>=0.4"],

theme = ["ttkbootstrap>=0.5"]

}

[project.scripts]

template\_builder = "template\_builder.builder\_core:main"

### **11.2 requirements.txt**

jinja2>=3.0

Pillow>=8.0

tkinterdnd2>=0.3

tkinterweb>=0.4

ttkbootstrap>=0.5

* **Nota**: Python 3.10+ con modulo Tkinter incluso (es. python3-tk su Linux).

### **11.3 Processo di Rilascio su PyPI**

1. Aggiornare pyproject.toml con nuova version = "X.Y.Z" (es. 1.0.0 → 1.1.0).
2. Aggiungere voce in CHANGELOG.md per la nuova versione.

Creare branch release/vX.Y.Z:  
  
 git checkout -b release/v1.1.0

# Aggiorna version in pyproject.toml e CHANGELOG.md

git commit -am "chore: prepare release v1.1.0"

git push -u origin release/v1.1.0

gh pr create --title "Release v1.1.0" --label release

Dopo merge PR, creare tag:  
  
 git checkout main

git pull

git tag -a v1.1.0 -m "Release v1.1.0"

git push origin v1.1.0

Pubblicare su PyPI:  
  
 python3 -m build

twine upload dist/\*

1. Verificare versione su PyPI e aggiornare documentazione (link, badge).

## **12. Requisiti Completi (requirements.txt)**

jinja2>=3.0

Pillow>=8.0

tkinterdnd2>=0.3

tkinterweb>=0.4

ttkbootstrap>=0.5

È necessario anche Python 3.10+ con modulo Tkinter incluso (es. python3-tk).

## **13. Raccomandazioni Finali**

1. **Allineare la documentazione**:  
   * Aggiornare README, ARCHITECTURE, CHANGELOG secondo le indicazioni.
   * Consolidare tutti i riferimenti ai file sorgente.
2. **Completare implementazione F9–F18**: seguire roadmap e aggiungere test adeguati.
3. **Pulire/archiviare** le note obsolete in docs/.
4. **Mantenere CI** verde: includere linting, coverage e test headless in GitHub Actions.
5. **Packaging & rilascio**: seguire linee guida semantiche e automazione PyPI.
6. **Aggiornare test suite** per coprire nuovi scenari (theme, engine switching, audit automatico).

## **14. Fonti**

Tutte le informazioni sono estratte inalterate dai seguenti documenti originali (PDF, GitHub, deep research):

### **14.1 Versione v1**

* README.md#L46-L53
* ARCHITECTURE.md#L9-L13, #L31-L34, #L36-L39
* builder\_core.py#L19-L24
* model.py#L30-L35, #L66-L73
* docs/DEV\_PLAN\_B2.md#L2-L5
* docs/legacy\_issues.md#L11-L16, #L22-L24
* CHANGELOG.md#L1-L2
* docs/IMPACT\_MATRIX\_B2.md#L1-L5

### **14.2 Versione v2**

* \_\_init\_\_.py#L14-L21, #L16-L21
* assets.py#L4-L10
* filters.py#L7-L11
* model.py#L22-L27, #L88-L95
* step\_image.py#L12-L20, #L116-L124
* widgets.py#L66-L74
* services/text.py#L38-L46
* services/images.py#L46-L54

### **14.3 Versione v3**

* ARCHITECTURE.md#L9-L13
* builder\_core.py#L55-L64
* model.py#L54-L62
* widgets.py#L233-L241
* services/text.py#L38-L46, #L106-L113, #L118-L126, #L141-L150
* services/storage.py#L79-L88, #L44-L53
* infrastructure/preview\_engine.py#L4-L13, #L85-L94, #L22-L25
* filters.py#L7-L11
* assets.py#L4-L10
* docs/legacy\_issues.md#L9-L16, #L20-L24

### **14.4 Versione v4**

* builder\_core.py#L84-L91
* widgets.py#L74-L83, #L159-L167, #L189-L196
* services/text.py#L38-L46, #L76-L84, #L109-L112
* services/images.py#L67-L75, #L183-L190, #L202-L205, #L20-L21
* services/storage.py#L46-L54, #L79-L88, #L92-L101, #L109-L117, #L132-L140
* model.py#L22-L30, #L54-L62, #L88-L96, #L99-L107
* infrastructure/preview\_engine.py#L52-L60
* filters.py (tutte le righe)
* assets.py (tutte le righe)
* ARCHITECTURE.md#L9-L18, #L26-L34

*Fine della Relazione Tecnica Omnicomprensiva*