A Dissertation Report on

Website Builder with Recommendation System



Aadarsha Shahi

23834806

BSc Computing (Software Engineering)



Supervisor: Nischal Khandka:

NAMI College

University of Northampton

New Baneshwor, Kathmandu, Nepal

August 2025

Abstract

This dissertation presents a no-code website builder for non-technical users. It adopts MVC architecture, React for an interactive editor/preview, Express for secure APIs, and MongoDB/Mongoose for flexible persistence. Features: block drag-and-drop, inline editing, autosave, and publish/unpublish. A hybrid recommender combines weighted rules with TF-IDF/cosine similarity (optional learned slot). A privacy-preserving lexical RAG chatbot provides deterministic, source-linked help. Policy-driven text/image moderation (lists, PII, NSFW, CLIP) returns allow/review/block verdicts.

Implementation followed incremental engineering with layered validation. Unit, integration, API, and end-to-end tests covered components, contracts, and full journeys across major browsers; 120 automated tests passed, supplemented by manual checks. Results show functional completeness, cross-browser consistency, and reliable APIs. Contributions include a maintainable MVC architecture, an auditable hybrid recommender, a deterministic Chatbot, and multi-signal moderation tailored to no-code authoring. Future work: feedback-aware re-ranking, collaboration and governance features, automated WCAG checks, CI performance budgets, exportable templates, and optional self-hosted deployment.

Acknowledgement

I express my sincere gratitude to my supervisor, Mr. Nischal Khadka, for steady guidance, constructive critique, and patience throughout this dissertation. His clear expectations, timely feedback, and insistence on methodological rigor shaped the project's scope and quality from refining the problem and requirements to strengthening testing strategy and evaluation.

I am grateful to the teaching staff and technical teams, including module tutors for research-method and engineering guidance, the library staff for help with sources, and lab teams for a dependable environment. Thanks to interview and survey participants whose inputs grounded features in real needs, and to classmates and friends who reviewed wireframes, tried early builds, and reported issues that improved usability and documentation.

Finally, I thank my family for encouragement, patience, and quiet space, especially during testing and report revisions. I also acknowledge the open-source community behind React, Express, MongoDB, Vitest, Playwright, and related tools. Any errors are my own; this work reflects a collective effort.

Table of Contents

1. Introduction 1

1.1 Project Background 1

1.2 Project Rational 2

1.3 Aim & Objectives 2

1.4 Dissertation Methodology 3

1.4.1 Software Requirement Engineering and Solution Specification: 3

1.4.2 System Analysis and Design 4

1.4.3 System Implementation 4

1.4.4 Database Development 4

1.4.5 System Testing 4

2. Requirement Engineering 4

2.1 Elicitation Activity 5

2.1.1 Interview Plan 5

2.1.2 Interview Findings 6

2.1.3 Comparable System 8

2.1.4 Ethnography Research 13

2.1.5 Research Methodology 18

2.2 Requirement Specification 18

2.2.1 Problem Domain Description 18

2.2.2 Proposed Solution Strategy 18

2.2.3 Alternative Solution Strategy 19

2.2.4 Functional Requirements 19

2.2.5 Non-functional Requirements 20

2.2.6 Goals of Implementation 21

2.2.7 Performance Requirements 22

3 System Analysis and Design 23

3.1 Use Case Diagram 23
3.2 Class Diagram 24
3.3 Sequence Diagram 26
3.4 Activity Diagram 30
3.5 Entity Relationship Diagram 34
3.6 Database Schema 35
4 System Interface Design 36
4.1 Wireframe 36
4.2 Screen Mock-ups 43
5 System Built & Technical Notes 53
5.1 Technologies Used 53
5.2 Third Party Integration 60
5.3 Recommendation System 61
5.4 Chatbot 63
5.5 Content Moderation System 64
5 Testing 65
5.1 Overview of Test Strategy 65
5.2 Sample Test Result 65
5.3 Test Evaluation 92
6 Conclusion 93
Bibliography 94
Appendix 96
Appendix 1 96
Appendix 2 97
Appendix 3 101

List of Figures

Figure 2.1: Landing Page of Odoo Website (Odoo, 2025) 8
Figure 2.2: Landing Page of Wix Website (Wix, 2025) 9
Figure 2.3: Landing Page of Site123 Website (Site123, 2025) 10
Figure 2.4: Landing page of Design Website (Design, 2025) 11
Figure 3.1: Use Case Diagram 23
Figure 3.2: Class Diagram 25
Figure 3.3: Sequence Diagram for CreateWebsite 26
Figure 3.4: Sequence Diagram for Preview Website 27
Figure 3.5: Sequence Diagram for Publish Website 28
Figure 3.6: Sequence Diagram for Browse Templates 29
Figure 3.7: Activity Diagram Showing Create Website Flow 30
Figure 3.8: Activity Diagram Showing Drag-and-Drop 31
Figure 3.9: Activity Diagram Showing Preview Flow 32
Figure 3.10: Activity Diagram Showing Publish Website Flow 33
Figure 3.11: Entity Relationship Diagram 34
Figure 3.12: Database Schema 35
Figure 4.1: Wireframe of Website Layout 36
Figure 4.2 Wireframe of Pricing Page 37
Figure 4.3: Wireframe of Templates Page 38
Figure 4.4: Wireframe of Register Modal 39
Figure 4.5: Wireframe of Login Modal 40
Figure 4.6: Wireframe of User Dashboard 41
Figure 4.9: Wireframe of Website Builder 42
Figure 4.13: Screen Mockup of Landing Page 43
Figure 4.14: Screen Mockup of Pricing Page 44
Figure 4.15: Screen Mockup of Pricing Page 45
Figure 4.16: Screen Mockup of Service Page 46
Figure 4.17: Screen Mockup of Contact Page 47
Figure 4.18: Screen Mockup of Login Modal 48
Figure 4.19: Screen Mockup of Register Modal 48
Figure 4.20: Screen Mockup of User Dashboard 49

Figure 4.21: Screen Mockup of Create Website Modal 50
Figure 4.22: Screen Mockup of Website Builder 50
Figure 4.23: Screen Mockup of Page Panel 51
Figure 4.24: Screen Mockup of Theme Panel 51
Figure 4.25: Screen Mockup of Templates Selection 52
Figure 5.1: Model Class Implementation Example of Website Model 54
Figure 5.2:Model Class Implementation Example of Website Controller 55
Figure 5.3:Model Class Implementation Example of Website Route 56
Figure 5.4: Example of Implementation of View Layer – Builder (1) 57
Figure 5.5: Example of Implementation of View Layer - Builder(2) 57
Figure 5.6: Example of Implementation of View Layer - Builder(3) 58
Figure 5.7: Example Implementation of Templates File – IntroA 59
Figure 5.8: Third Party (PayPal) Integration From Backend 60
Figure 5.9: Third Party (PayPal) Integration from Frontend 60
Figure 5.10: Recommendation System Example of IDF, Vectorize & Cosine 61
Figure 5.11: Recommendation System Implementation Example of Score Rule 62
Figure 5.12: Example Implementation of Chatbot Implementation Using RAG 63
Figure 5.13: Example Implementation of Content Moderation Using Regex Engine 64
Figure 6.1: Test Evidence: Test ID : API-001 70
Figure 6.2: Test Evidence: Test ID : API-002 71
Figure 6.3: Test Evidence: Test ID : API-003 71
Figure 6.4: Test Evidence: Test ID : API-004 72
Figure 6.5: Test Evidence: Test ID : API-005 72
Figure 6.6: Test Evidence: Test ID: API-006 72
Figure 6.7: Test Evidence - Test ID - API-007 73
Figure 6.8: Test Evidence; Test ID - API-008 73
Figure 6.9: Test Evidence - Test ID - API-009 73
Figure 6.10: Test Evidence - Test Id: API-010 74
Figure 6.11: Test Evidence - TestID - API-011 74
Figure 6.12: Test Evidence - Test ID: API-012 74
Figure 6.13: Test Evidence - Test ID: API-013 75
Figure 6.14: Test Evidence - Test ID: API-014 75
Figure 6.15: Test Evidence - Test ID: API-015 75
Figure 6.16: Test Evidence - Test ID: API-016 76

Figure 6.17: Test Evidence - Test ID: API-017 76
Figure 6.18: Test Evidence - Test Id: API-018 76
Figure 6.19: Test Evidence - Test ID: API-020 77
Figure 6.20: Test Evidence - Test ID: API-021 77
Figure 6.21: Test Evidence - Test ID: API-023 77
Figure 6.22: Test Evidence - UT-001 78
Figure 6.23: Test Evidence - UT-002 78
Figure 6.24:Test Evidence: UT-003 79
Figure 6.25: Test Evidence: UT-004 79
Figure 6.26: Test Evidence - UT-005 79
Figure 6.27: Test Evidence: UT-006 80
Figure 6.28: Test Evidence: UT-007 80
Figure 6.29: Test Evidence: UT-008 80
Figure 6.30: Test Evidence: UT-009 81
Figure 6.31: Test Evidence: UT-010 81
Figure 6.32: Test Evidence: UT-011 82
Figure 6.33: Test Evidence: UT-012 82
Figure 6.34: Test Evidence: UT-013 82
Figure 6.35: Test Evidence: UT-014 83
Figure 6.36: Test Evidence - UT-015 83
Figure 6.37: Test Evidence: UT-016 83
Figure 6.38: Test Evidence: UT-017 84
Figure 6.39: Test Evidence: IT-001 84
Figure 6.40: Test Evidence: IT-002 85
Figure 6.41: Test Evidence: IT-003 85
Figure 6.42: Test Evidence - IT-004 85
Figure 6.43: Test Evidence: IT-005 86
Figure 6.44: Test Evidence: IT-006 86
Figure 6.45: Test Evidence: UT-007 86
Figure 6.46: Test Evidence: IT:008 87
Figure 6.47: Test Evidence: IT-009 87
Figure 6.48: Test Evidence: IT-010 88
Figure 6.49: Test Evidence: IT-011 88
Figure 6.50: Test Evidence: IT-012 89

Figure 6.51: Test Evidence: IT-013 89
Figure 6.52: Test Evidence: IT-014 90
Figure 6.53: Test Evidence: IT-015 90
Figure 6.54: Test Evidence: IT-016 91
Figure 6.55: Test Evidence: IT-017 91
Figure 6.56: Test Evidence: IT-018 92

List of Tables

Table 2-1: Interview Plan.....5
Table 2-2: Interview Findings.....6
Table 2-3: Table Specifying Non-Functional Requirements.....21
Table 4-1: Label & Description for Wireframe of Website Layout.....36
Table 4-2: Label & Description for Wireframe of Website - Pricing Page.....37
Table 4-3: Label & Description for Wireframe of Website - Templates Page.....38
Table 4-4: Label & Description for Wireframe of Register Modal.....39
Table 4-5: Label & Description for Wireframe of Login Modal.....40
Table 4-6: Label & Description for Wireframe of User Dashboard.....41
Table 4-9: Label & Description for Wireframe of Website Builder.....42
Table 5-1: Table Mentioning Technology Used.....53
Table 6-1: Test Table of API Testing.....66
Table 6-2: Test Table of Unit Testing.....67
Table 6-3: Test Table of Integration Testing.....68
Table 6-4: Test Table of End-to-End Testing.....69

Abbreviation

API - Application Programming Interface

BaaS - Backend-as-a-Service

CDN - Content Delivery Network

CSS - Cascading Style Sheets

ERD - Entity-Relationship Diagram

GDPR - General Data Protection Regulation

HTML - Hyper Text Markup Language

IT - Information Technology

JWT - JSON Web Token

LC/NC - Low-Code/No-Code

LCDP - Low-Code Development Platform

ML - Machine Learning

MVC - Model-View-Controller

REST - Representational State Transfer

SEO - Search Engine Optimization

SLR - Systematic Literature Review

SME - Small-to-Medium Enterprises

SSL - Secure Sockets Layer

UI/UX - User Interface/User Experience

UML -Unified Modeling Language

WCAG - Web Content Accessibility Guidelines

1. Introduction

A modern website is essential for schools, community groups, and enterprises, yet limited time, budget, and design skills make creation slow and inconsistent. Existing tools are too complex or too rigid, producing stalled projects and uneven results. This project proposes a no-code builder with clear guidance and standard design practices, offering a straightforward path from goals to publishable pages, reducing unnecessary choices, and letting authors focus on message and audience.

The work focuses on settings with limited resources and seeks to make professional website creation more attainable. By framing the problem, stating the purpose, and outlining a guided workflow approach, this introduction sets the foundation for the project's design, development, and evaluation in educational, community, and small-business contexts. It situates the project within real-world constraints and clarifies how the following chapters build on this direction.

1.1 Project Background

Websites have become essential for visibility and audience engagement for small enterprises, community organizations, and educational initiatives. However, limited time, budget, and design expertise often make web projects slow to start and difficult to maintain. These constraints commonly result in delays, reliance on external contractors, and inconsistent quality across pages and devices.

The current options present a familiar trade-off. Custom development offers flexibility but requires coding skills, specialist design input, and ongoing budgets for updates. Full content management systems provide breadth but introduce technical setup, configuration overhead, and maintenance tasks. Template-based builders are simpler to begin with, yet they can restrict branding and site structure, and they often lead to decision overload or gaps in accessibility and mobile responsiveness.

This project emerges from that context to reduce the barrier to establishing and sustaining a professional web presence in resource-constrained settings. The focus is on a no-code direction that emphasizes clear steps, industry standard design, and maintainable outcomes. By prioritizing predictable setup and consistent presentation, the project aims to lessen dependence

on technical specialists and enable entrepreneurs, educators, and community groups to publish and manage websites with greater confidence and lower ongoing effort.

1.2 Project Rational

The global demand for easy-to-use website builder continues to grow, yet many platforms do not balance ease-of-use with professional outcomes. The common issues include complex interfaces that require training, generic templates that weaken branding, and added fees for essential capabilities such as responsive design and accessibility checks. These barriers slow delivery, raise costs, and make ongoing maintenance difficult for entrepreneurs, educators, and creators.

To address these gaps, the project adopts three design pillars. A Recommendation System proposes layouts, font pairings, and colour schemes based on stated goals and draft content, reducing decision load without removing control. Adaptive Templates align with brand preferences and remain consistent across devices, improving readability and visual coherence. Scalable Tools allow projects to start with core features and add modules over time, avoiding upfront complexity and limiting long-term lock-in. Where feasible, processing is kept local to improve privacy and fit institutional constraints.

This rationale supports a no-code platform that emphasizes clarity, consistency, and maintainability. The approach is intended to shorten setup time, improve brand alignment, and make standards such as accessibility and responsiveness easier to achieve. By focusing on guidance and gradual growth rather than custom-built solution, the platform provides a practical path to professional websites in resource-constrained settings.

1.3 Aim & Objectives

The project aims to build a user-friendly website builder with template recommendations for easy design and customization.

The project's main aim will be achieved through the following objectives: -

- Perform requirements gathering and analysis for diverse client segments.
- Conduct a feature gap analysis to determine competitive advantages and function priorities.

- Perform technical feasibility studies to evaluate technology options against business goals.
- Designed a scalable backend architecture to support template customization, user data management, and real-time recommendation services.
- Designed user interface prototypes and wireframes prioritizing navigational clarity and accessibility for non-technical users.
- Developed a modular template system enabling user customization, structured layout adaptation, and inline content editing.
- Built a responsive drag-and-drop composition tool to streamline template editing and enhance user autonomy.
- Implementation of a recommendation system for personalized template and layout suggestions.
- Develop a moderation system for automated detection of guideline violations.
- Implementation of a Chatbot to handle common user inquiries and reduce manual support workload.
- Developed a secure database with optimized access patterns for user preferences and component metadata.
- Execute comprehensive testing across all system layers to validate functionality, reliability, and performance.

1.4 Dissertation Methodology

Project follows Agile Scrum: a lightweight, iterative framework emphasizing transparency, inspection, and adaptation. Two-week sprints pull backlog through planning, daily scrums, review, and retrospective, delivering a shippable increment per sprint.

1.4.1 Software Requirement Engineering and Solution Specification:

Requirements are gathered via interviews, surveys, and comparable-system review, then consolidated into SRS with user stories, acceptance criteria, and NFRs. MoSCoW prioritization and a traceability matrix guide scope and verification.

1.4.2 System Analysis and Design

Functional flows and structures are modeled using use-case, activity/sequence, and component diagrams, alongside information architecture and UI wireframes/prototypes. Design decisions address responsiveness, accessibility (WCAG 2.2 targets), performance, security, and maintainability, resulting in a baseline architecture and interface specifications.

1.4.3 System Implementation

Development proceeds incrementally in time-boxed sprints, translating approved designs into working software with coding standards, version control, and continuous integration. Each increment adheres to the Definition of Done, including code review, basic security checks, and updated technical documentation.

1.4.4 Database Development

A conceptual model is refined into logical and physical schemas, defining entities, relationships, validations, and indexes aligned to core queries. Migration plans, seed data, backup/restore procedures, and data protection measures are established to ensure integrity, performance, and reliability.

1.4.5 System Testing

A test plan defines entry/exit criteria and maps tests to requirements. Quality is verified via unit, integration, and end-to-end testing, complemented by performance and accessibility evaluations and cross-browser/device checks; defects are logged, triaged, and re-tested until acceptance criteria are satisfied.

2. Requirement Engineering

Requirements engineering is the systematic elicitation, analysis, specification, and validation of stakeholder needs to define project scope and constraints. In this project, requirements were derived from interviews, survey of target users, and comparable system analysis to identify capabilities and gaps. These inputs were consolidated into a clear requirements list and scoped features to guide design and implementation.

2.1 Elicitation Activity

Elicitation activity is conduct to captures stakeholder needs to define project requirements. This project conduct survey and interviews with target users, identified pain points, technical complexity, design limits, and resource constraints, which informed the requirements list and its priorities.

2.1.1 Interview Plan

Interviews were conducted with three stakeholder groups: - students, school administrators, and travel agency owners, to elicit domain-specific needs, workflows, and constraints. Targeted questions covered content types, publishing/approval flow, and maintenance responsibilities. Findings were consolidated into segment-specific requirements to guide design priorities and project scope.

Agenda:
9th April 2025
Duration: One Hour
Objective:
To understand user expectations for a website builder that supports creating personal sites easily and effectively.

Table 2-1: Interview Plan

Duration (in minutes)	Interview Session & Objective
10 minutes	Introduction
10 minutes	Nora Young (Student Freelancer) • Identify essential portfolio setup requirements. • Understand layout customization expectations. • Discover key visual preferences. • Assess user support requirements. • Know content update preferences.
10 minutes	Brian Carter (School Administrator) • Identify school specific builder needs.

	• Learn school content update methods. • Find team collaboration requirements. • Understanding content structuring needs. • Identify desired accessibility features.
10 minutes	Oliver Cruz (Travel Agency Owner) • Gather essential travel features. • Learn content presentation expectations. • Determine offer update preferences. • Understand navigation and discovery flow. • Discover expectation design assistance.

2.1.2 Interview Findings

Interview results highlight user needs, preferences, and challenges, informing the core capabilities required to address pain points and expectations.

Table 2-2: Interview Findings

Interviewee	Questions & Answers
Nora Young	**Which features would make setting up an online portfolio easier?** I would prefer a well-structured, easily editable template would best meet the requirement.
	Describe the expected workflow for choosing and adjusting a layout. I prefer a category-specific layout and customize it by adding or deleting blocks.
	Specify the key visual cues that best represent the brand. The most important visuals include a minimal design, customized color scheme, a professional photograph, and a work showcase area.
	Do you need guidance or support on building your site? I prefer step-by-step guidance on presenting work effectively and achieving a professional look, with a real-time preview feature.
	What is the preferred workflow for updating portfolio content? I prefer an easy-to-use content editor that allows quick updates to projects, text, and images.

Brian Carter	**Specify the features needed for administering school website.** The builder should enable easy creation and maintenance of pages for school news, events, staff profiles, and student resources.
	What is the preferred workflow for updating events and announcements? I prefer a simple editor that allows adding and editing of events and announcements, with scheduling for future publication.
	Do you need to work in a team? The team collaboration is useful especially for updating news, staff profiles, and events but not required if the workflow is simple.
	Which approach would make it easy to organize different kinds of content in the builder? The builder should provide pre-designed templates for content types to enable quick selection and customization to fit requirements.
	What changes would make the site fully accessible to all users? The builder should provide accessibility features such as adjustable text size, contrast controls, and image alt text in order to support universal access.
Oliver Cruz	**Which core features are essential for a travel agency website builder?** The builder should include a booking form, a destination gallery, and straightforward tools to showcase travel packages and offers.
	How can the builder make presenting travel packages and deals easier? The builder should provide configurable templates for travel packages with pricing, itinerary, and images, plus simple update workflows to keep offers current.
	What features would simplify updates to offers and images? Builder should include a simple image uploader and an editable offers section to change prices and package descriptions without coding.
	How should visitors explore services and destinations? Builder should ensure a clear site menu and filters to organize and sort destinations.

	What forms of support are expected when making design decisions? I prefer templates for travel agencies, with brand-aligned layout and color suggestions, plus real-time preview.

2.1.3 Comparable System

This section reviews four most prominent existing website-builders: - Odoo, Wix, Site123, and Design in order to identify core features, strengths, and limitations. The comparative analysis informs the design and development of the proposed website builder.

2.1.3.1 Odoo (https://www.odoo.com/)

Odoo provides an open-source ERP platform with modules for CRM, online sales, finance, and inventory; its user-friendly design and extensibility support efficient, cross-department workflows.

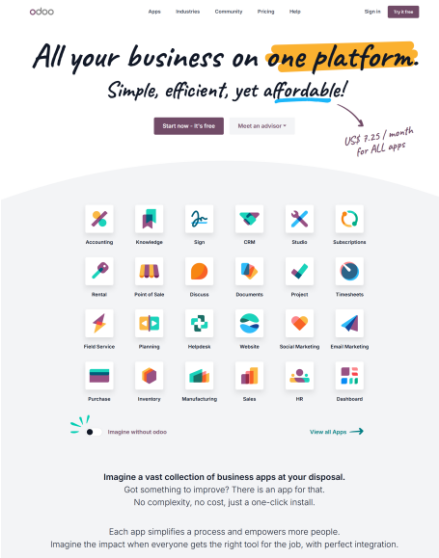


Figure 2.1: Landing Page of Odoo Website (Odoo, 2025)

Table 2.3: Pros Vs. Cons. for Odoo Website

Pros	Cons

Full source-code access (HTML/CSS/JS) enabling deep customization.	Visual drag-and-drop builder is dated and lacks modern UX patterns.
Built-in Git workflows for versioning and collaborative development.	Sparse template catalog, so more design work is required upfront.
Extensible REST/HTTP APIs for bespoke integrations.	Overkill for simple brochure sites; setup feels heavy.
Proven scalability for high traffic and ERP-class processes.	Performance needs tuning; defaults are not optimized.
Granular role-based access control (admin/editor/viewer).	No automated design suggestions or recommendation tooling.

2.1.3.2 Wix (https://www.wix.com/)

Wix provides a code-free platform with AI design help, drag-and-drop tools, and configurable templates, enabling professional website creation and management for users and organizations.

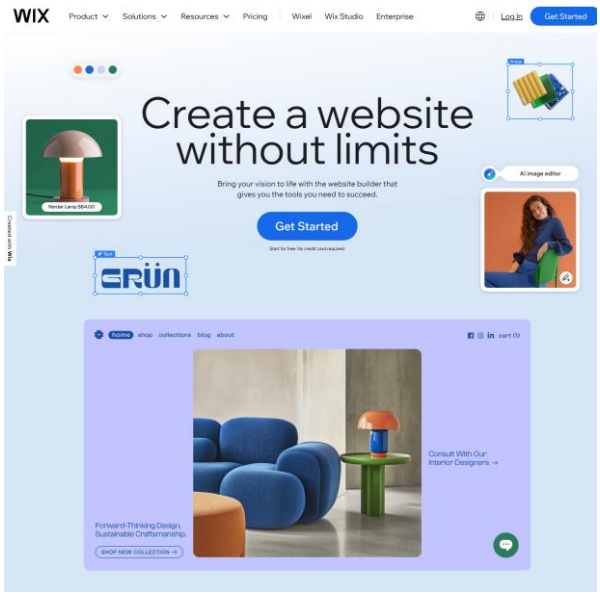


Figure 2.2: Landing Page of Wix Website (Wix, 2025)

Table 2.4: Pros Vs. Cons for Wix Website

Pros	Cons
Visual drag-and-drop editor with precise placement and alignment controls.	Template lock-in after publish; the base theme cannot be changed.

Catalogue of 800+ responsive templates/themes for rapid launch.	No direct access to underlying HTML/CSS/JS.
Built-in CDN and SSL/TLS for faster delivery and secure transport.	Limited API surface for external integrations.
Automatic image compression with lazy-load behaviour.	No Git integration or code versioning workflow.
Reliable for small-to-mid traffic volumes with consistent uptime.	Basic roles/permissions; collaboration features are limited.

2.1.3.3 Site123 (https://www.site123.com/)

SITE123 is a no-code website platform that pairs structured templates with mobile-ready design, built-in SEO, e-commerce modules, and secure hosting to streamline website creation.

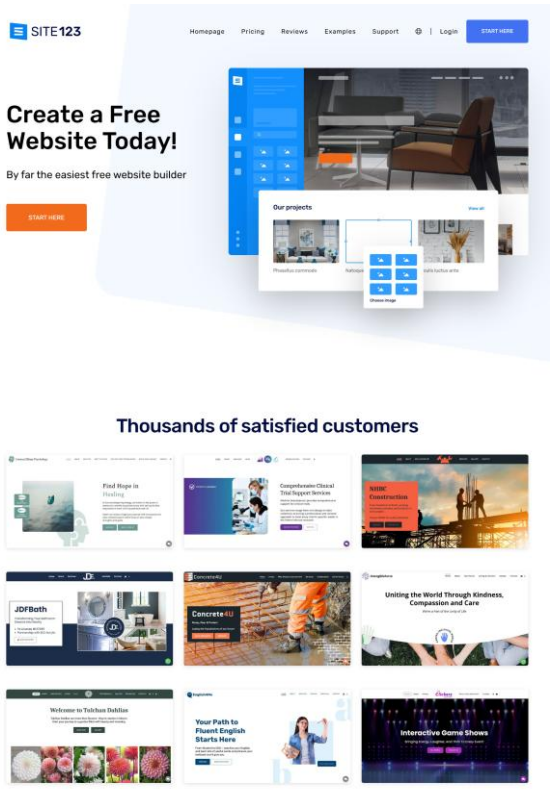


Figure 2.3: Landing Page of Site123 Website (Site123, 2025)

Table 2.5: Pros vs. Cons for Site123 Website

Pros	Cons
Single-click setup with guided onboarding for non-technical users.	No direct access to source code (HTML/CSS/JS) or APIs.

Automatically responsive layouts for mobile and desktop.	Fixed section designs with no pixel-level control.
Section-based drag-and-drop editor for quick content assembly.	No built-in CDN or advanced performance optimization.
Lightweight output that delivers fast page loads.	Lacks inventory-management capabilities.
Stable uptime appropriate for small-scale sites.	Limited scalability for high traffic or complex features.

2.1.3.4 Design (https://www.design.com/website-builder)

Design.com is a website builder that rapidly generates professional sites from a business name, offering industry-specific templates, straightforward layout/typography/color customization, and brand-asset integration (logos for web and print), making it well-suited to small businesses and beginners.

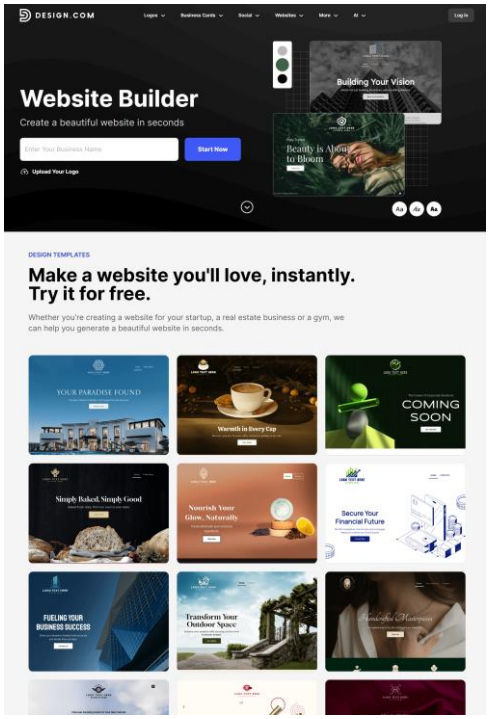


Figure 2.4: Landing page of Design Website (Design, 2025)

Table 2.6: Pros Vs Cons for Design Website

Pros	**Cons**

Modern, visually oriented templates suited to portfolio sites.	No e-commerce functionality.
AI branding tools for logo creation and color palette generation.	No access to source code (HTML/CSS/JS) or APIs.
Drag-and-drop editor with grid snapping/alignment.	Outputs static sites only; dynamic features aren't supported.
Automatic image compression (WebP).	No version control or team collaboration tools.
Mobile-optimized (responsive) layouts by default.	No staging environment for pre-publish testing.

2.1.3.5 Checklist of Functionalities

Table 2.7: Table showing Checklist of Functionalities

Feature	**Odoo**	**Wix**	**Site123**	**Design**
Drag and Drop	✓	✓	✓	✓
Design Components	✓	✓	✗	✗
Mobile Optimization	✓	✗	✓	✓
Custom Domain	✓	✓	✓	✓
Pre-Design Templates	✓	✓	✓	✓
Images & Gallery Sliders	✗	✓	✗	✗
Video Embeds	✓	✓	✗	✗
Custom Fonts & Colour	✓	✓	✓	✓
Animation Effects	✓	✓	✓	✓
Reusable Sections	✓	✗	✗	✗
AI Content Generator	✗	✓	✗	✗
Template Recommendation System	✗	✓	✗	✗
Code Access(HTML/CSS/JavaScript)	✓	✗	✗	✗
API Access	✓	✓	✗	✗
Analytics & Insights	✓	✓	✗	✗

Form Builder	✗	✗	✓	✗
User Access Management	✓	✓	✗	✗
Version Control	✓	✓	✓	✓

2.1.3.6 Performance Metric Table

The table summarizes how Design.com performs and how suitable it is compared with alternative platforms, based on key features and capabilities.

Table 2.8: Performance Metric Table

Feature	**Odoo**	**Wix**	**Site123**	**Design**
Drag and Drop	3	5	4	4
Pre-Designed Templates	3	5	4	4
Custom Fonts & Colors	5	5	4	5
Animation Effects	4	5	3	4
Reusable Sections	5	1	1	1
AI Content Generator	1	5	1	1
Template Recommendation System	1	5	1	1
Code Access (HTML/CSS/JS)	5	1	1	1
API Access	5	3	1	1
Version Control	5	1	1	1
Mobile Optimization	4	5	4	5
Images & Gallery Sliders	1	5	1	1
Video Embeds	5	5	1	1
User Access Management	5	3	1	1
Form Builder	1	1	5	1
Analytics & Insights	4	5	1	1

2.1.4 Ethnography Research

Ethnography research involves a systematic analysis of academic and industry sources to surface user behaviors, latent needs, and social dynamics. It identifies gaps between formal processes and actual practice and provides an evidence base for user-centered software design.

2.1.4.1 Requirements for the Development of a Website Builder with Adaptive Design, Gulmira Bekmanova, Banu Yergesh, Assel Omarbekova, Laura Orynbay, Aiganym Bessembayeva, Dinara Kabdylova, Altanbek Zulkhazhav, Beibarys Sultan, 2024

The paper "Requirements for the Development of a Website Builder with Adaptive Design" sets out requirements for a system that enables non-technical users to create responsive, accessible websites. It combines automatic site generation with adaptive design and proposes server-side rendering (SSR) using Nuxt.js, culminating in a working prototype that includes a user-friendly builder and automated content display. The work is positioned as an inclusive, potentially commercialisable approach, funded by the Ministry of Science and Higher Education of Kazakhstan.

The review categorizes automatic website generation into three families: mock-up (wireframes to GUI code), example-based (learned from existing sites), and AI-driven (infers user-preferred designs). It charts the shift from fixed layouts to responsive (RWD) and adaptive (AWD), then toward content- and profile-aware adaptation, such as Responsive Universal Design, supported by component frameworks enabling device-sensitive layouts. The analysis argues for server-side rendering (SSR) for better first-load speed, SEO, and reduced client work, and highlights Nuxt.js as a flexible choice offering hybrid SSG/SSR, dynamic imports, code splitting, and routing. It notes adaptive practices in Wix, Weebly, and Ecwid, highly relevant to education.

Gap Analysis

While the paper states system requirements and demonstrates a prototype, it provides limited empirical evaluation: there are no reported usability studies, comparative benchmarks against alternative SSR frameworks in the same implementation context, or quantified accessibility audits (e.g., WCAG conformance checks). Security and data-governance requirements are noted but not validated through testing, and the role model remains coarse (administrator/user) without team-workflow evidence. The prototype is described as basic and "requires further optimization," with development "ongoing," indicating open work on performance under load, extensibility with third-party systems, and rigorous assessment in real deployments. These gaps

suggest the need for controlled user testing, accessibility verification, performance/load testing, and comparative studies to confirm generalizability beyond the chosen Nuxt.js stack.

2.1.4.2 Systematic Literature Review of low-code and its future trends, Axel Panama Velasquez, Christian Perez-Salazar, Lizbeth A. Hernandez-Gonzalez, Angel J. Sanchez-Garcia, 2024

Systematic Literature Review of Low-Code and Its Future Trends surveys the state of low-code in software engineering, following Kitchenham's SLR method and narrative synthesis. The review targets characteristics, benefits, challenges, and future trends of low-code development platforms (LCDPs), drawing on major databases (IEEE Xplore, ACM DL, ScienceDirect, SpringerLink) and selecting 15 primary studies published between 2018 and 2023. The paper positions low-code as a response to demand for faster delivery without sacrificing quality and outlines its protocol, scope, and organisation.

Across the sampled studies, LCDPs share several traits: visual GUIs with drag-and-drop, connectivity to external services/data sources, collaboration support, reuse, and role specification. Reported benefits include faster development, reduced cost, ease of use, and strong integration options; challenges concentrate on limited customization and scalability. Future trends emphasise adoption by SMEs, linkage with DevOps practices, and broader digitisation.

Gap Analysis

The authors note a lack of prior reviews focused *exclusively* on low-code in software engineering, which motivates the study but also signals a developing evidence base. Methodologically, quality assessment of included studies was not performed (relying on venue peer review), and the time window (2018–2023) plus database scope may limit coverage. These factors point to opportunities for deeper validation e.g., comparative benchmarking, longitudinal adoption/effectiveness studies, and more rigorous appraisal of accessibility, security, and scalability claims.

2.1.4.3 Evaluating the Impact of No-Code/Low-Code Backend Services on API Development and Implementation: A Case Study Approach, Pratik Dhoke, Dr. Prasad Lokulwar, 2023

The paper Evaluating the Impact of No-Code/Low-Code Backend Services on API Development and Implementation: A Case Study Approach examines how visual, block-based tools and low/no-code backends can lower the barriers to building APIs. It frames the problem (boilerplate, complexity of business logic, external integrations, error handling, testing), surveys related paradigms (flow-based programming) and platforms (Firebase, AWS Amplify, NoFlo), and proposes a case-study prototype that lets non-experts assemble backend logic via drag-and-drop and generate runnable endpoints. Architectural discussion emphasizes SOA/microservices and common API practices (REST/GraphQL), with market context for low/no-code adoption.

The review and case study argue that visual composition reduces syntax burden, speeds prototyping, and broadens participation in backend/API work. Reported advantages include modularity and reuse via SOA, easier integration through standard protocols, and developer productivity from tools such as Swagger/OpenAPI and visual editors. The document also notes ecosystem momentum (growth forecasts for low/no-code and BaaS) and highlights mainstream building blocks (authentication, data sync, messaging) that can be composed rather than implemented from scratch.

Gap Analysis

Evidence is primarily descriptive and architectural; there is limited empirical evaluation (e.g., usability studies with target users, controlled comparisons against traditional stacks, or measurements of performance, scalability, and maintenance effort). Security, persistence, and operations are acknowledged as future work (e.g., hardening against XSS/CSRF/SQLi, moving from in-memory to durable storage, monitoring/logging/caching, autoscaling), and governance aspects (versioning, collaboration, auditability) are not validated in the prototype. These gaps suggest the need for task-based user testing, accessibility checks, load/stress testing, and longitudinal studies on cost, reliability, and extensibility in real deployments.

2.1.4.4 Low-code/No-code Development: A systematic literature review, Hind EL KAMOUCHI, Mohamed KISSI, Omar EL BEGGAR, 2023

Low-code/No-code Development: A Systematic Literature Review surveys LC/NC platforms using a hybrid SLR protocol that combines Kitchenham, PRISMA, and PICO. The review defines six research questions (application domains, enabling technologies, platform features, widely used tools, benefits, and limitations), searches major digital libraries, applies

inclusion/exclusion criteria (2014–July 2023, peer-reviewed, full text), and reports a PRISMA flow from 610 records to 29 primary studies. It also specifies a quality-assessment scheme (QA1–QA6) and a structured data-extraction form to synthesise results.

The SLR reports broad LC/NC adoption, with strongest concentration in web/mobile and healthcare, followed by manufacturing, finance, and IoT. Core support technologies include visual programming, model-driven engineering, and (to a lesser extent) AI/ML; platform features cluster around visual modelling, data integration, prebuilt components/templates, APIs, deployment/security, and collaboration. Frequently cited platforms include Mendix, Microsoft Power Platform, Salesforce, OutSystems, Appian, Google AppSheet, and Zoho Creator. Reported benefits emphasise reduced cost/time, "citizen developer" participation, reuse, and agility; recurrent limitations include vendor lock-in, performance concerns, security/scalability issues, steep learning curves, restricted customisation, interoperability/debugging challenges, and limited on-premises support.

Gap Analysis

Although the review is systematic, the evidence base it aggregates is heterogeneous and often descriptive; many included works do not provide rigorous empirical evaluations (e.g., controlled comparisons on performance, security, accessibility, or long-term maintainability). Coverage is bounded by the 2014–2023 window and selected libraries, which may omit fast-moving practitioner evidence and very recent advances. Feature availability is not standardised across platforms, complicating cross-study comparisons, and the prominence of vendor-lock-in/performance concerns indicates a need for deeper benchmarking and longitudinal studies on portability, cost of ownership, and governance in real deployments.

2.1.5 Research Methodology

An iterative Agile Scrum process delivered a low-code backend and no-code builder. Requirements lived in a prioritized backlog enabling rapid prototyping, stakeholder feedback, and incremental releases. Practices: 2–4-week sprints, daily scrums, weekly refinement, and reviews/retrospectives with merged code, passing tests, updated docs, and CI/CD artifacts. Mixed-methods such as surveys, interviews, and four-system comparisons, informed design; GDPR-compliant ethics with anonymization/consent. Analysis used Excel and thematic coding; limitations include sample size and response bias.

2.2 Requirement Specification

A requirements specification is a formal, structured document that defines the system's required functions, performance standards, and constraints to satisfy stakeholder needs and achieve project objectives.

2.2.1 Problem Domain Description

Key challenges for non-technical users in web development include:

- High costs for small changes when outsourcing to developers.
- Reliance on developer availability, creating delays and workflow bottlenecks.
- Traditional system demand coding expertise for both frontend and backend work.
- Limited self-service scaling and updates e.g., blogs, e-commerce, payments.
- Difficulty on adapting systems as requirements evolve.
- Many low-code platforms ignore backend logic, data models, and third-party integrations.

2.2.2 Proposed Solution Strategy

To mitigate challenges experienced by non-technical users, adopt the following approach:

- Unified interface enabling end-to-end website and backend creation without developer dependence.
- Pre-built templates and modular components to streamline layout, logic, and common workflows.
- Automatic generation of data models, databases, CRUD APIs, and third-party integrations, no coding required.

- Real-time updates with live preview, allowing content and workflow changes without redeployment.
- Step-by-step, in-product guidance to reduce the learning curve.
- Free core features, with low-cost paid tiers for advanced capabilities and higher limits.

2.2.3 Alternative Solution Strategy

The strategy specifies focused measures to address key challenges, with an emphasis on collaboration, flexibility, and scalability:

- Establish a marketplace for distributing pre-built website and backend templates, enabling fast deployment without coding.
- Use AI to translate natural-language prompts into executable code or visual workflows.
- Provide no-code builders for entry-level tasks, with exportable code for developer-driven extensions.
- Provide starter templates first, then enable advanced tooling as competency grows.
- Set up user forums for support, shared solutions, and collaborative fixes, lowering developer dependence.

2.2.4 Functional Requirements

Functional requirements specify the functions, features, and behaviors the system must provide to satisfy user needs. The system requires the following functions: -

Table 2.9: Functional Requirements

Requirement	**Description**
Template Library	Offers pre-designed templates for common use cases with preview and one-click apply.
Content Customization	Allows text, images, and settings to be edited via a visual UI without coding.
Drag-and-Drop Interface	Enables placement, movement, and resizing of elements with grid snapping and alignment guides.
Real time Preview	Instant preview across devices for frontend and backend updates.

Template Recommendation	Templates are suggested based on provided input requirements.
Secure Authentication	Provides account registration and login with hashed credentials, MFA, and session management.
Preview-mode	Loads the current website in a new tab for live, non-published preview.
Auto-save	Automatically saves edits in the background to prevent data loss; creates revision checkpoints for recovery.
Chatbot	AI assistant for visitors that handles FAQs, captures leads, and triggers workflows; connects to knowledge bases or live chat.
Content Moderation	Detects and blocks unethical or policy-violating content and issues clear warning message to user.
Payment Integration	Connects to payment providers for checkout and subscriptions.

2.2.5 Non-functional Requirements

Non-functional requirements specify the system's quality attributes, performance targets, and constraints rather than its features. Non-functional requirements include: -

Table 2-3: Table Specifying Non-Functional Requirements

Functions	**Description**
Security	Enforces strong authentication, authorization, and encryption across all services.
Reliability & Availability	Ensures consistent uptime with graceful degradation and automatic recovery.
Scalability	Supports horizontal growth in users, projects, and assets without performance decline.
Usability & Onboarding	Provides intuitive workflows, clear guidance, and minimal learning effort.
Accessibility	Conforms to WCAG guidelines with keyboard support and appropriate ARIA semantics.
Privacy & Data Protection	Limits data collection, applies consent controls, and enables data export/deletion.

2.2.6 Goals of Implementation

The implementation objectives define what the no-code platform should deliver. Key objectives are: -

- Enable non-technical users to create and run website without relying on developers.
- Provide live, drag-and-drop editing for UI design and backend workflows.
- Include templates and recommendations for branding, layouts, and service offerings.
- Allow instant deployment with one click to scalable clouds.
- Provide secure user authentication and role-based access control.
- Guarantee responsive behavior with consistent cross-browser performance.
- Automate compliance workflows with zero user setup.
- Smart help bot for support and onboarding: guides users, collects info, and launches workflows from chat.
- Automated content moderation that detects and blocks prohibited content, warns users, and logs incidents.
- Preview mode that instantly renders draft changes without publishing.

2.2.7 Performance Requirements

Performance requirements define quantitative targets for response time, throughput, concurrency, and resource utilization; the items below specify the Website Builder's performance requirements.

Table 2.10: Non-functional Requirements

Category	**Requirement**	**Description**
Performance	Response Time	The site should render in ≤500 milliseconds following user input.
Usability	Cross-Browser Compatibility	Ensure consistent performance across different browsers.
Scalability	Concurrent Users	The system must handle more than 1,000 concurrent users without latency experience.
Security	Data Encryption	Encrypt all user data and credentials to ensure security.
Reliability	Uptime	Maintain 99.9% uptime throughout designated peak hours.

3 System Analysis and Design

The system architecture integrates UML modeling use case, sequence, activity and class with data design ERDs and schemas. Interactions are defined via use cases; workflows via sequence/activity diagrams; data via ERDs; and storage via schemas, ensuring alignment of user needs with logical coherence and scalability.

3.1 Use Case Diagram

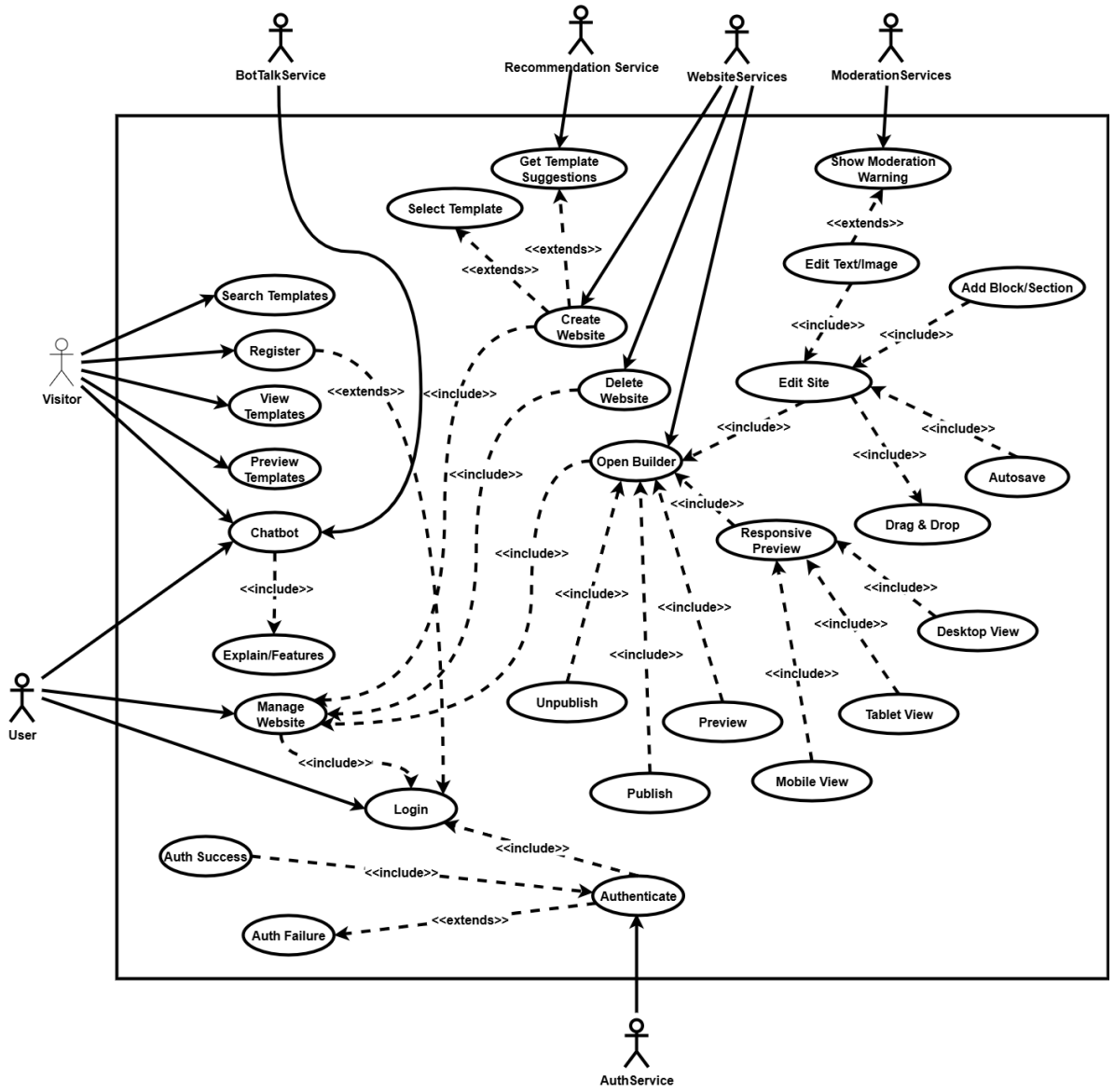


Figure 3.1: Use Case Diagram

The use case diagram models a website-builder system with two main actors, Visitor and User and external Auth, BotTalk, Recommendation, Website, and Moderation services. Visitors can search, view, and preview templates, chat with a bot to learn features, and register. Users authenticate via AuthService, then Manage Website: create a site (select template with optional recommendations), open the builder to drag-and-drop, add blocks, edit text/images with autosave, and use responsive preview (desktop/tablet/mobile), plus publish, unpublish, or delete. Moderation can conditionally show warnings during edits. «include» captures required sub-steps (e.g., device previews), while «extend» marks optional/conditional flows (e.g., auth failure, template suggestions, moderation warnings).

3.2 Class Diagram

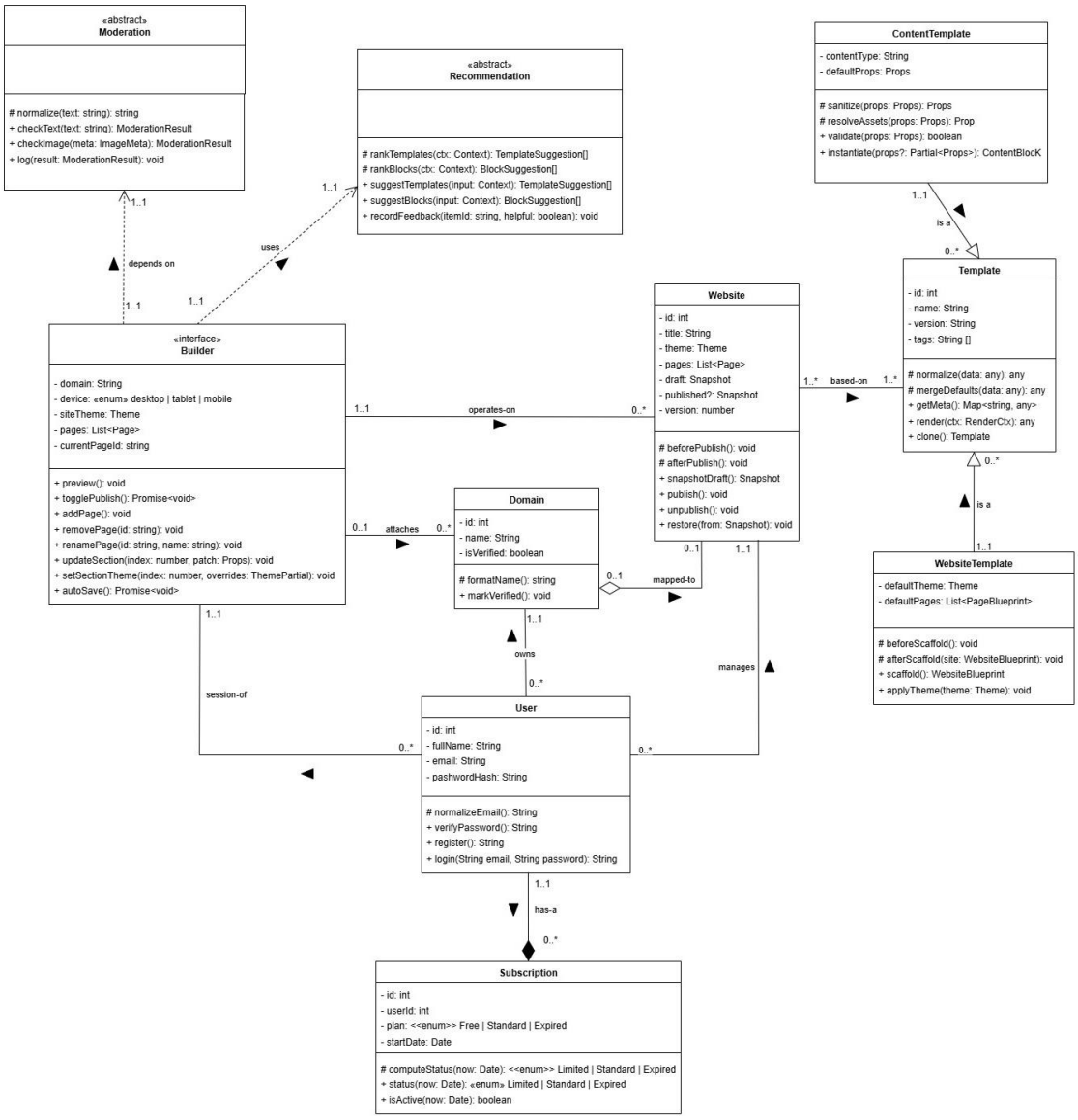


Figure 3.2: Class Diagram

This class diagram models website builder system. User manages many Websites and may have a single Subscription (Limited/Standard/Expired). A Domain belongs to a user and maps to at most one website, tracking name and verification. The Website holds title, theme, pages, snapshots/versions, and lifecycle hooks (before/after publish, rollback, snapshot). A Template (and concrete WebsiteTemplate) provides the base layout/content via ContentTemplate/ ContentBlock types that websites are based on. A session-scoped Builder interface operates on one website, switches device views (desktop/tablet/mobile), edits pages/blocks/themes, and handles preview/publish actions. The Builder uses a Recommendation service (rank/suggest templates/blocks, record feedback) and depends on Moderation (normalize/check text/images, log results). Cardinalities show User-Websites (1..*), Website-Domain (0..1-1), and a 1:1 Builder-Website during an editing session.

3.3 Sequence Diagram

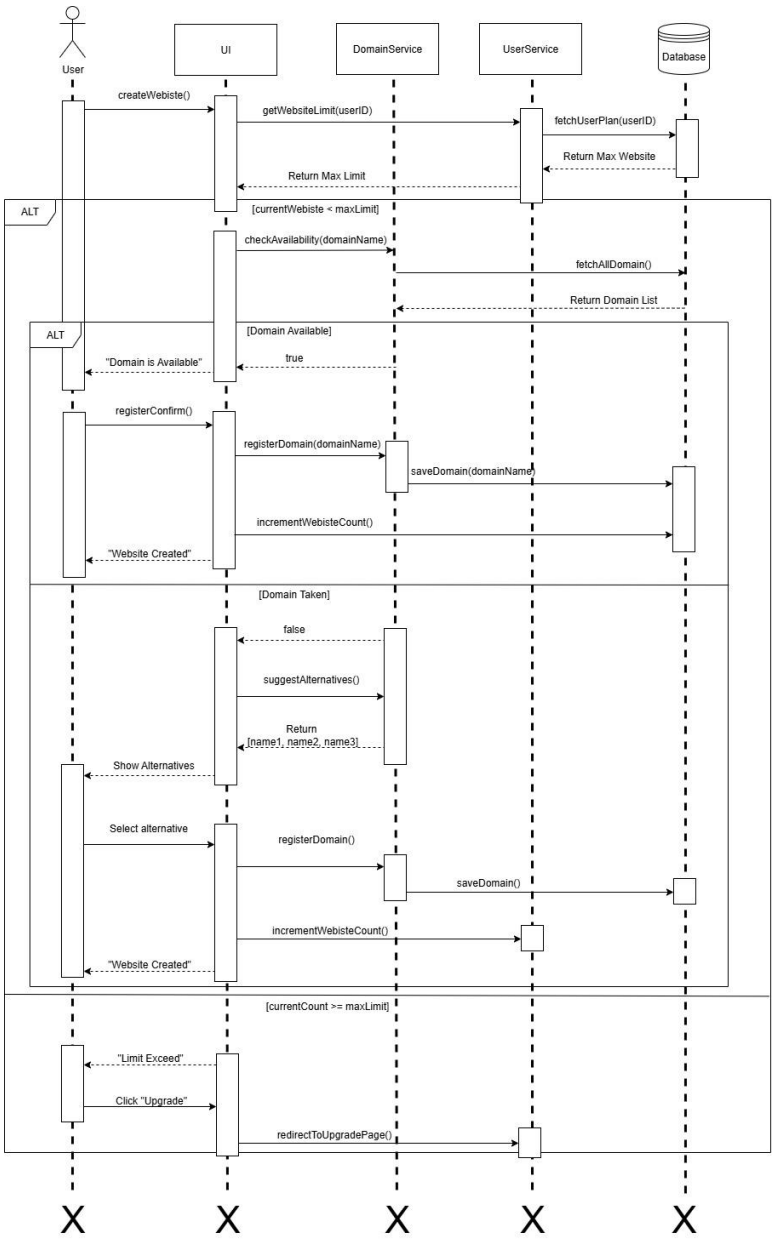


Figure 3.3: Sequence Diagram for CreateWebsite

This sequence diagram shows the Create Website flow across User-UI-DomainService-UserService-Database. The UI first asks UserService for the plan's max website limit; if the user is under the limit, it checks domain availability via DomainService (which queries the DB). If available, the system confirms, registers the domain, saves it, increments the website count, and reports "Website Created." If taken, DomainService returns alternative names; the user selects one, then the system registers/saves it and completes creation. If the limit is reached, the UI shows "Limit Exceed" and redirects the user to the Upgrade page.

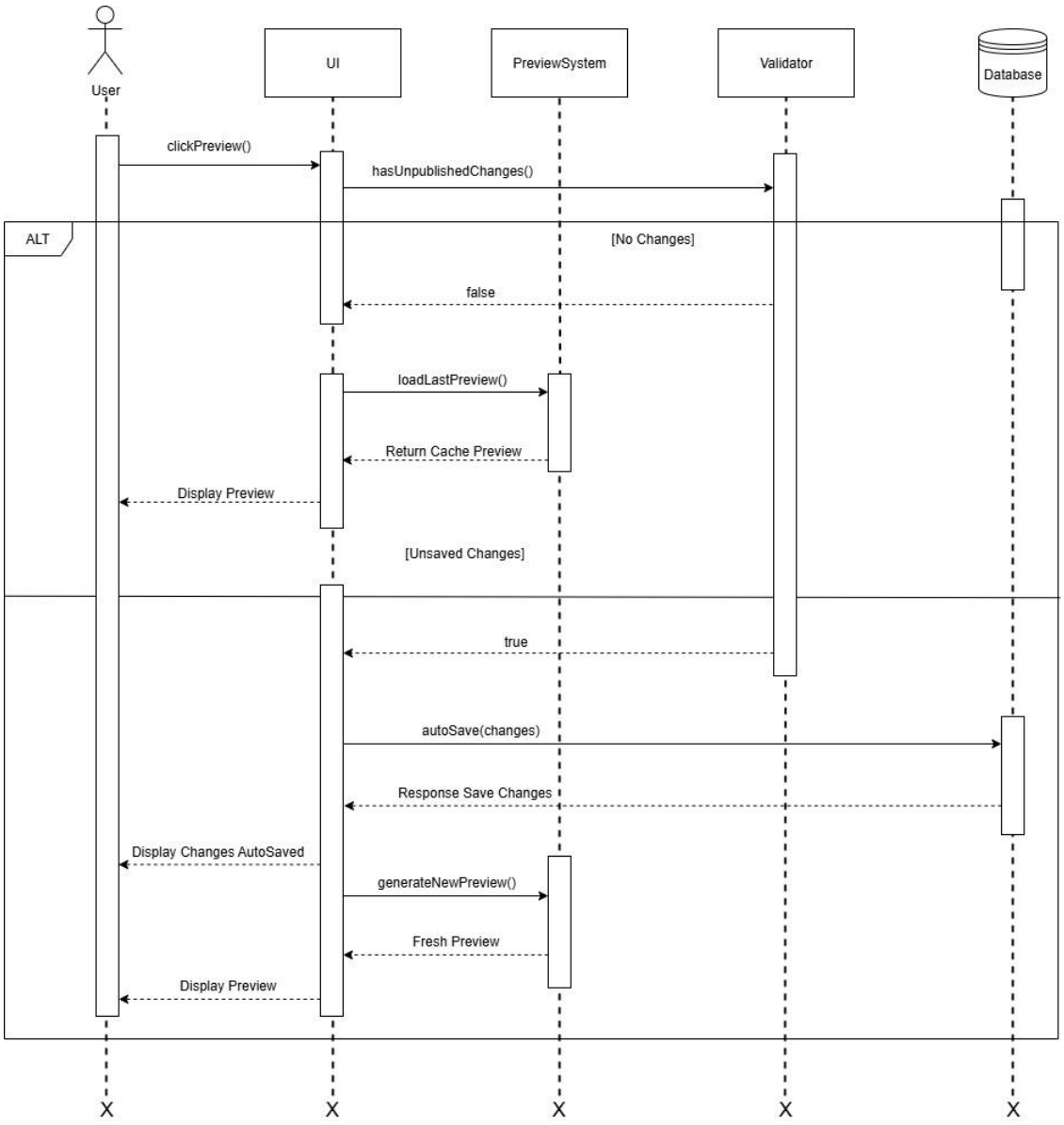


Figure 3.4: Sequence Diagram for Preview Website

This sequence shows the Preview button flow. The UI asks the PreviewSystem whether there are unpublished changes. If none, it loads the cached preview and displays it. If there are changes, the system auto-saves them to the database (via the preview layer/validator), then generates a fresh preview and returns it to the UI for display the ALT frame captures these two branches.

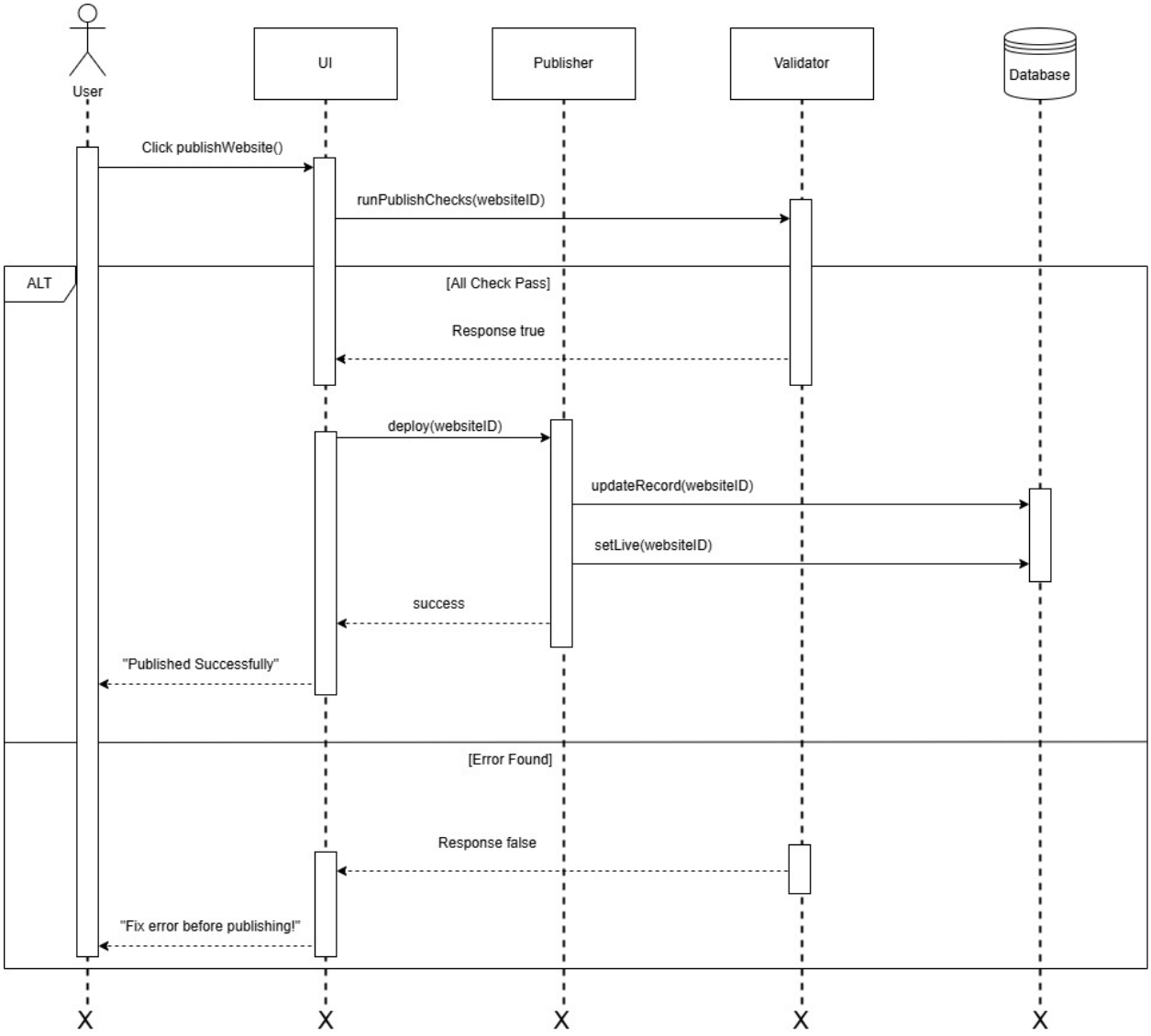


Figure 3.5: Sequence Diagram for Publish Website

This sequence models Publish Website. The user clicks Publish; the UI asks the Publisher to run pre-publish checks (via the Validator). If all checks pass, the UI tells the Publisher to deploy; the Validator updates the Database (update record, set site live), the Publisher returns success, and the UI shows “Published Successfully.” If an error is found, the Publisher returns false and the UI prompts the user to fix errors before publishing.

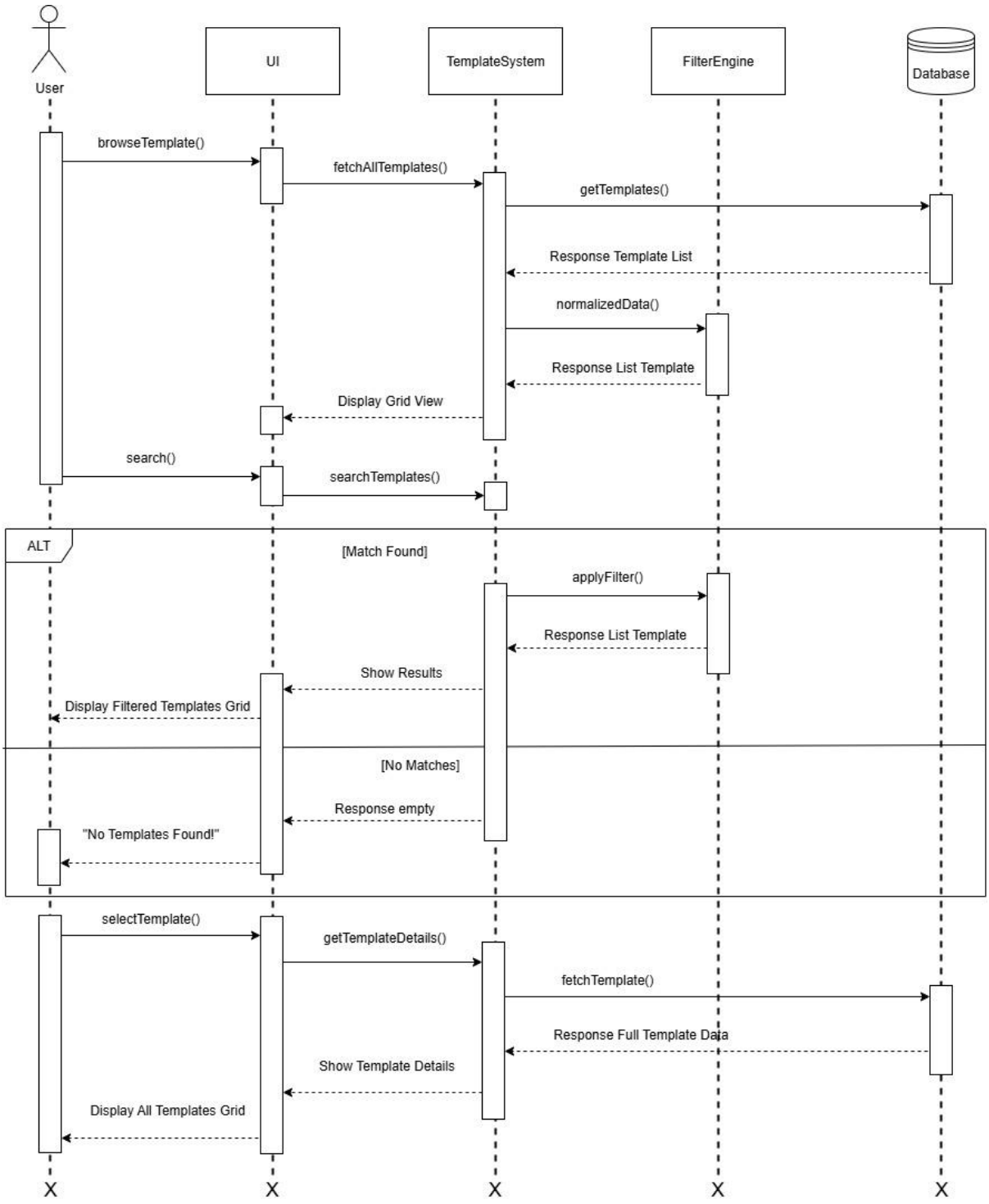


Figure 3.6: Sequence Diagram for Browse Templates

This sequence covers browsing and selecting templates. The user triggers browseTemplate(), the UI asks TemplateSystem to getTemplates(), which pulls from the DB, normalizes data, and returns a list for the UI's grid view. On search(), the UI calls searchTemplates(), which invokes

FilterEngine.applyFilter(): if matches exist, a filtered grid is shown; if none, the UI displays "No Templates Found." When the user selectTemplate(), the UI requests getTemplateDetails(), fetchTemplate() returns full data from the DB, and the UI shows the template details.

3.4 Activity Diagram

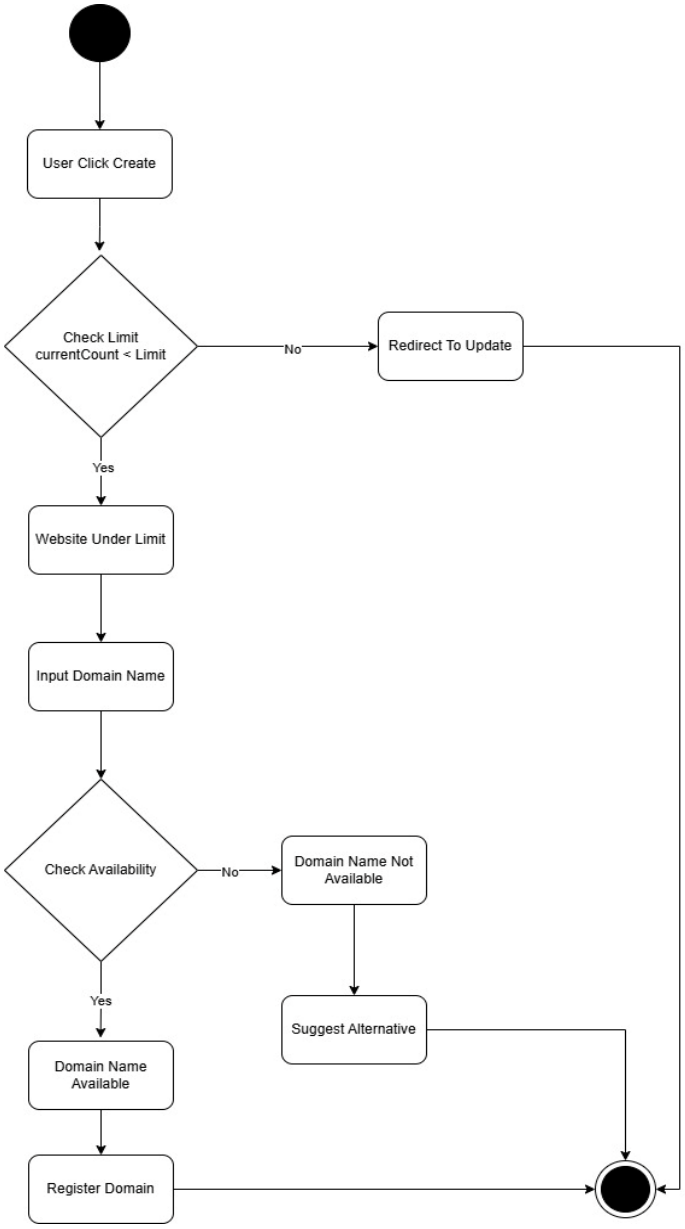


Figure 3.7: Activity Diagram Showing Create Website Flow

This activity diagram shows the Create Website flow with domain registration. After the user clicks Create, the system checks the plan limit (currentCount < limit). If the limit is exceeded, the user is redirected to upgrade. If under the limit, the user enters a domain name, the system

checks availability, and if available, registers the domain and ends. If the name is taken, the system suggests alternatives and loops until an available name is chosen, then registers and finishes.

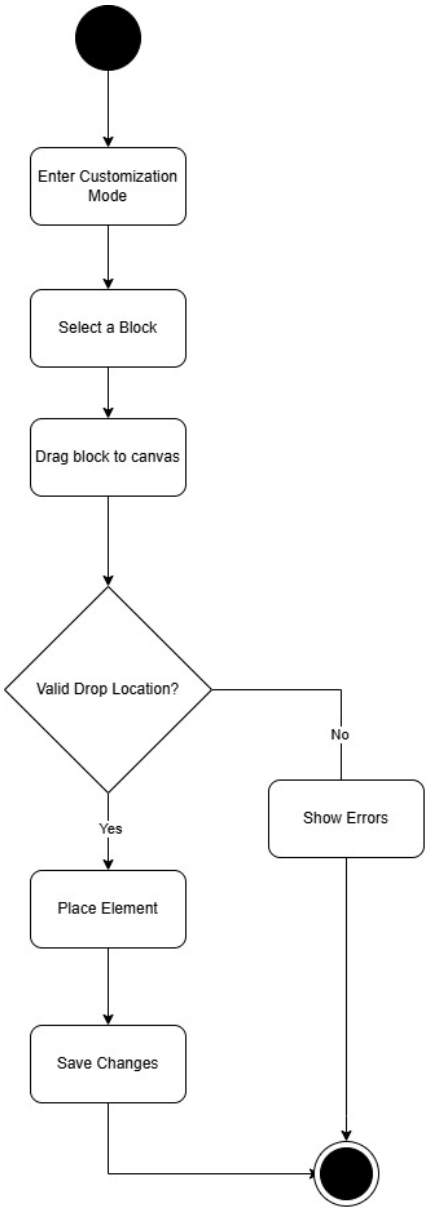


Figure 3.8: Activity Diagram Showing Drag-and-Drop

This activity diagram captures the drag-and-drop block placement flow: the user enters Customization Mode, selects a block, and drags it onto the canvas. The system checks whether the target is a valid drop location (respecting layout zones/constraints). If valid, it places the element and saves changes; if not, it shows errors and terminates the action.

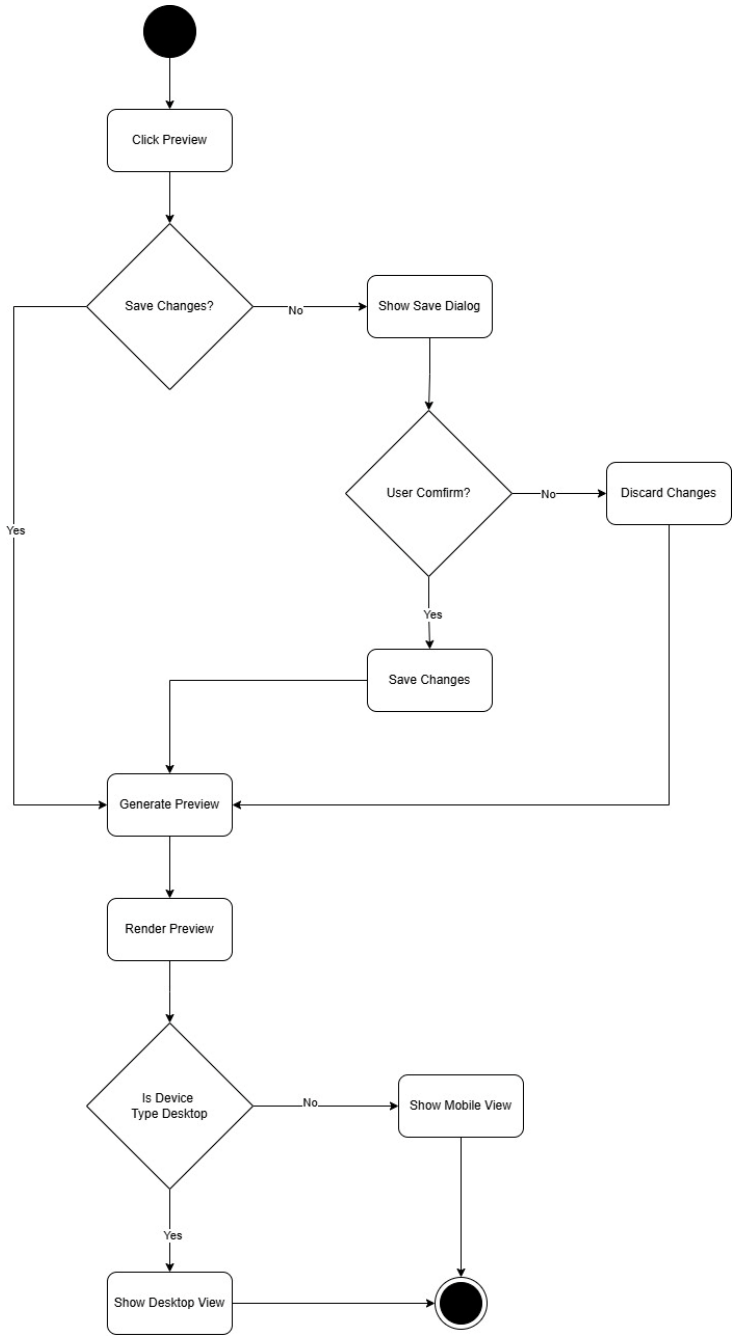


Figure 3.9: Activity Diagram Showing Preview Flow

This activity diagram outlines the Preview flow: the user clicks Preview; if there are unsaved changes, a save dialog appears on confirmation the system saves (or discards if declined). Then the system generates and renders the preview. Finally, it selects the device view Desktop if desktop, otherwise Mobile and displays the result.

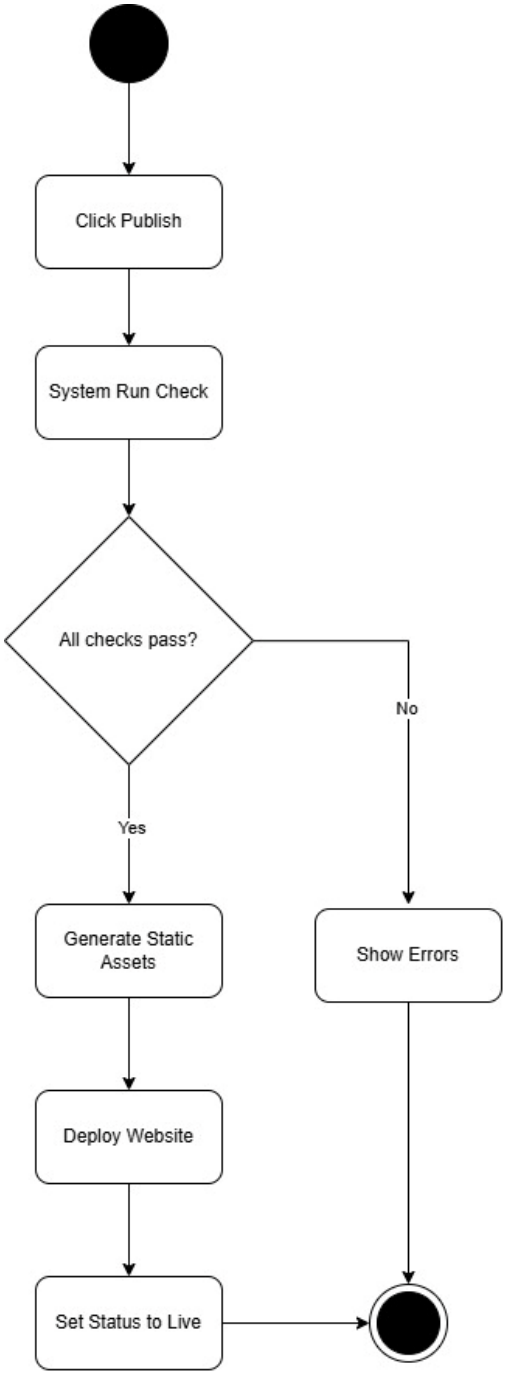


Figure 3.10: Activity Diagram Showing Publish Website Flow

This activity shows the Publish flow: the user clicks Publish, the system runs pre-publish checks, and a decision is made. If all checks pass, the system generates static assets, deploys the website, and sets status to Live. If any check fails, it shows errors and stops without deploying.

3.5 Entity Relationship Diagram

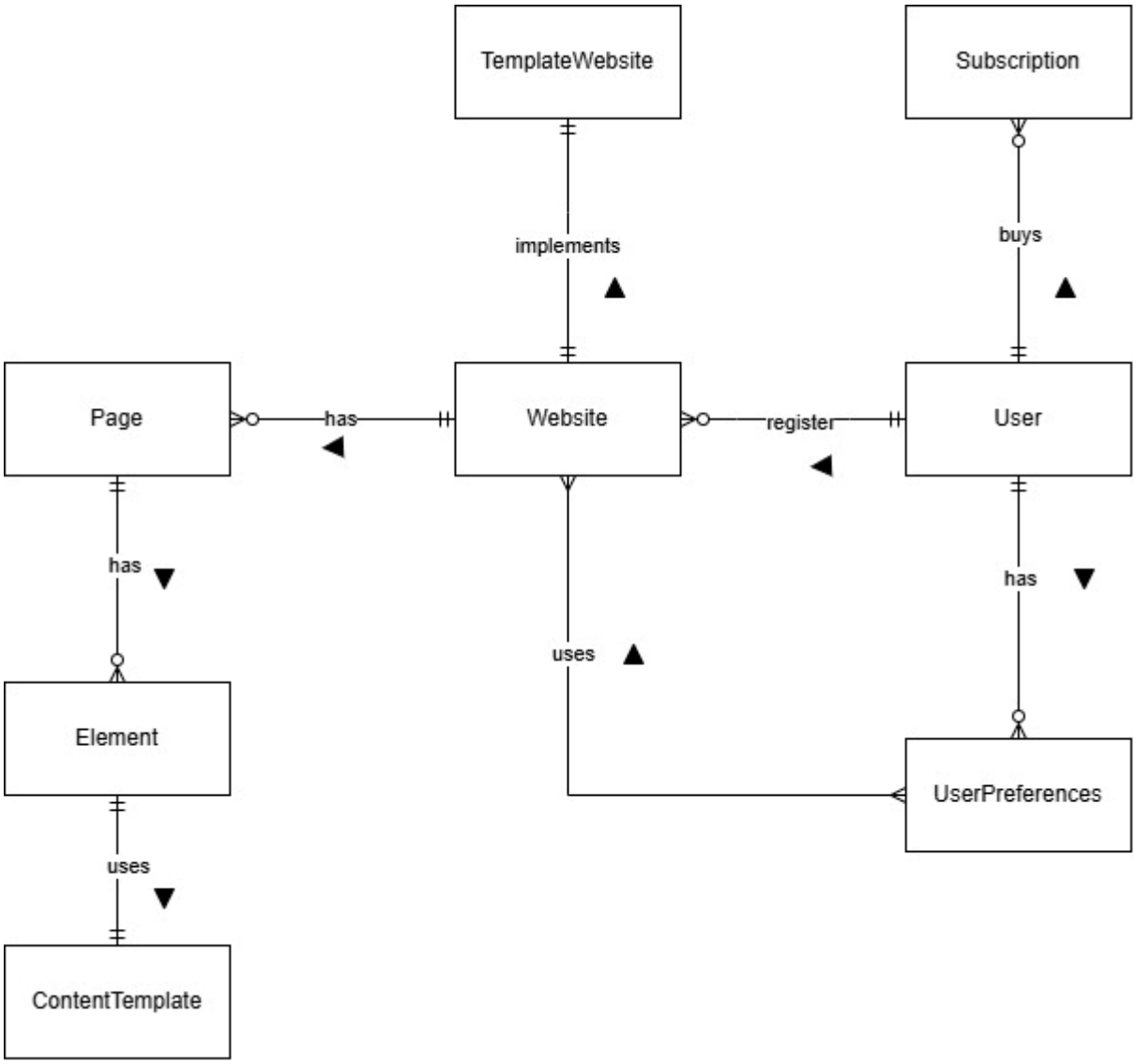


Figure 3.11: Entity Relationship Diagram

Domain model at a glance: User owns multiple Websites and can buy a Subscription. Each Website is based on/implements a TemplateWebsite, has many Pages, and uses UserPreferences for personalization. A Page contains many Elements, and each Element uses a ContentTemplate for its content. User also has one UserPreferences record that the Website reads to tailor content. Overall: User 1..* Websites-Pages-Elements-ContentTemplate; User-Subscription, User-UserPreferences.

3.6 Database Schema

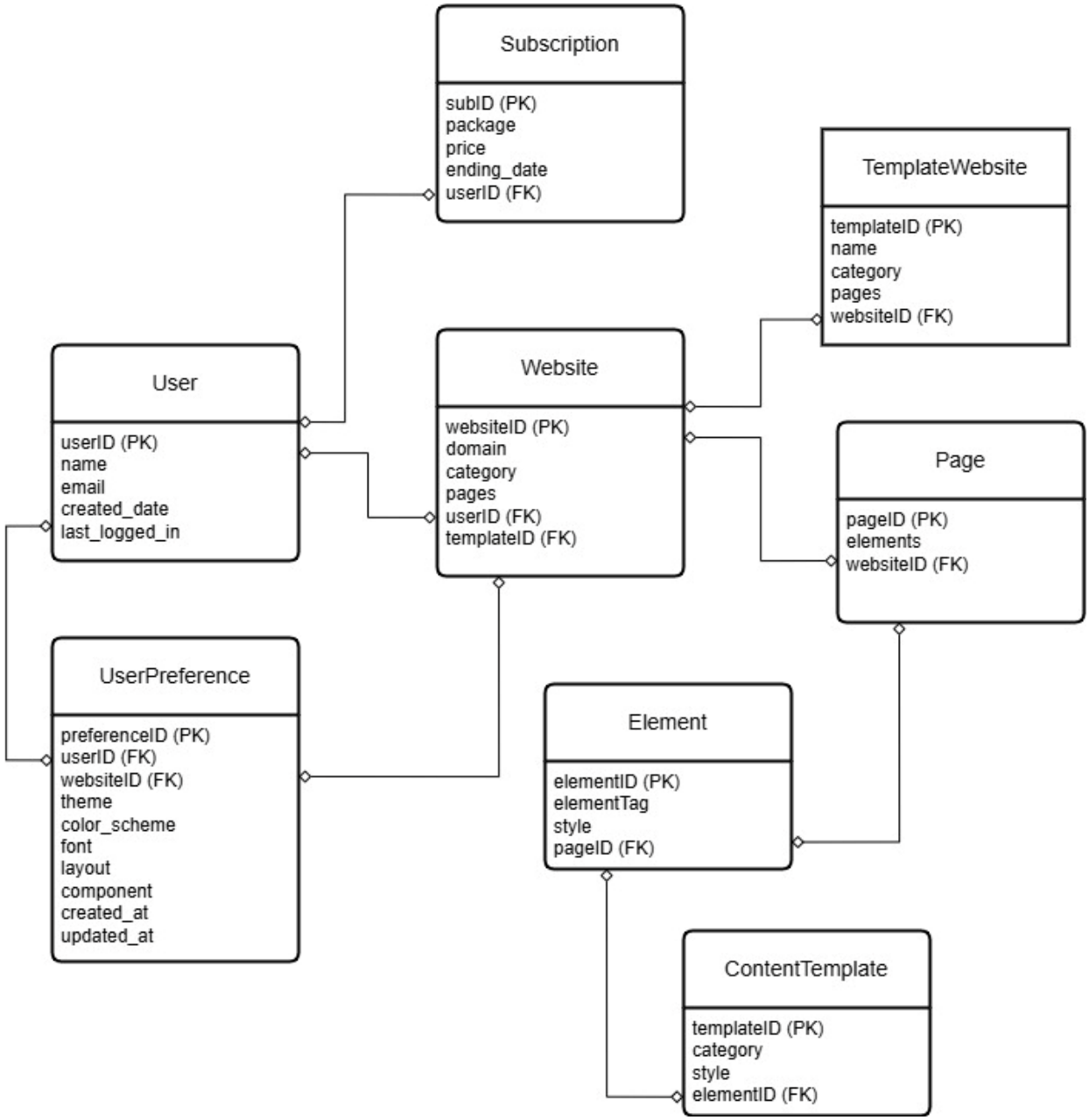


Figure 3.12: Database Schema

This ERD shows a User who can own multiple Websites (Website.userID-User.userID) and can have Subscriptions. Each website is based on a TemplateWebsite and is personalized by UserPreference. A website contains many Pages (Page.websiteID-Website.websiteID), each page contains many Elements (Element.pageID-Page.pageID), and elements can reference a ContentTemplate for reusable content/layout. Together it's: User-Websites-Pages-Elements with Subscription and UserPreference attached for plan and styling.

4 System Interface Design

System interface design defines interactions, content, and feedback, aligning goals with domain via IA, consistent patterns, controls. It reveals state, handles errors gracefully, reduces latency, and supports progress, validation, undo.

4.1 Wireframe

Wireframes are low-fidelity blueprints mapping layout, hierarchy, and navigation without styling. Using simple boxes and labels, they clarify IA, align scope, validate with users, and enable rapid iteration before design.

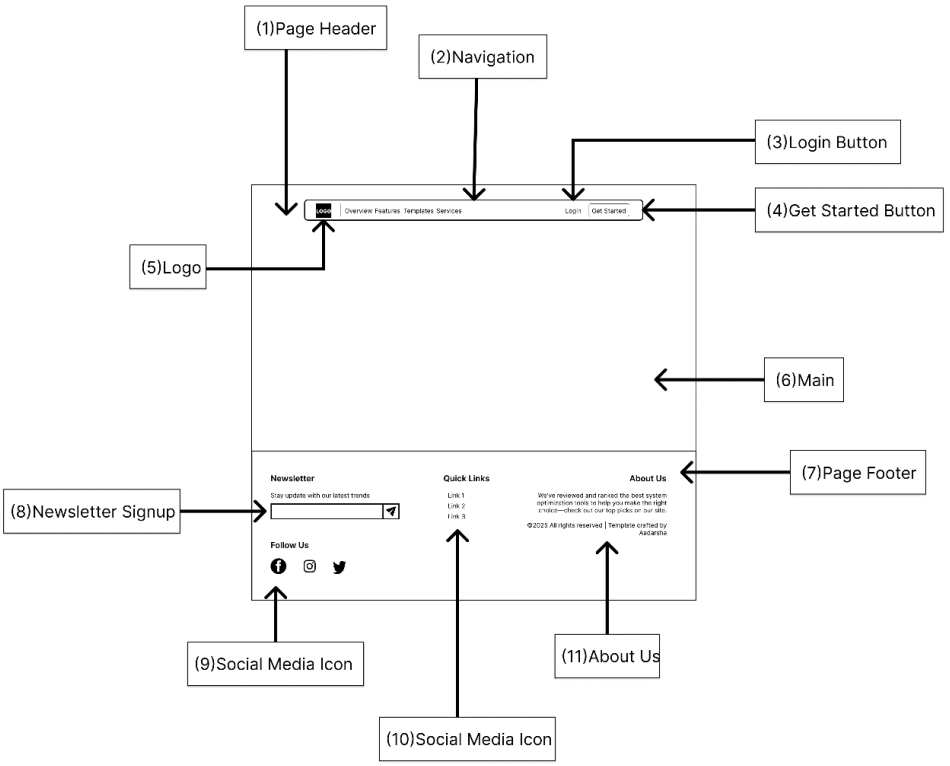


Figure 4.1: Wireframe of Website Layout

Table 4-1: Label & Description for Wireframe of Website Layout

S.N.	Label	Description
1	Page Header	Fixed top bar with logo, navigation, and quick actions.
2	Navigation	Primary menu linking to key product and info pages.

3	Login Button	Opens sign-in page or modal for returning users.
4	Get Started Button	Prominent CTA to create an account or start building.
5	Logo	Brand mark; clickable to return to the homepage.
6	Main	Central content area showing page-specific sections.
7	Page Footer	Bottom area with links, legal text, and contact info.
8	Newsletter Signup	Email field + submit to subscribe for updates.
9	Social Media Icon	Link to official social profile (e.g., X/Twitter).
10	Quick Link	Links for the most accessed pages
11	About Us	Short summary of company mission, background, and purpose.

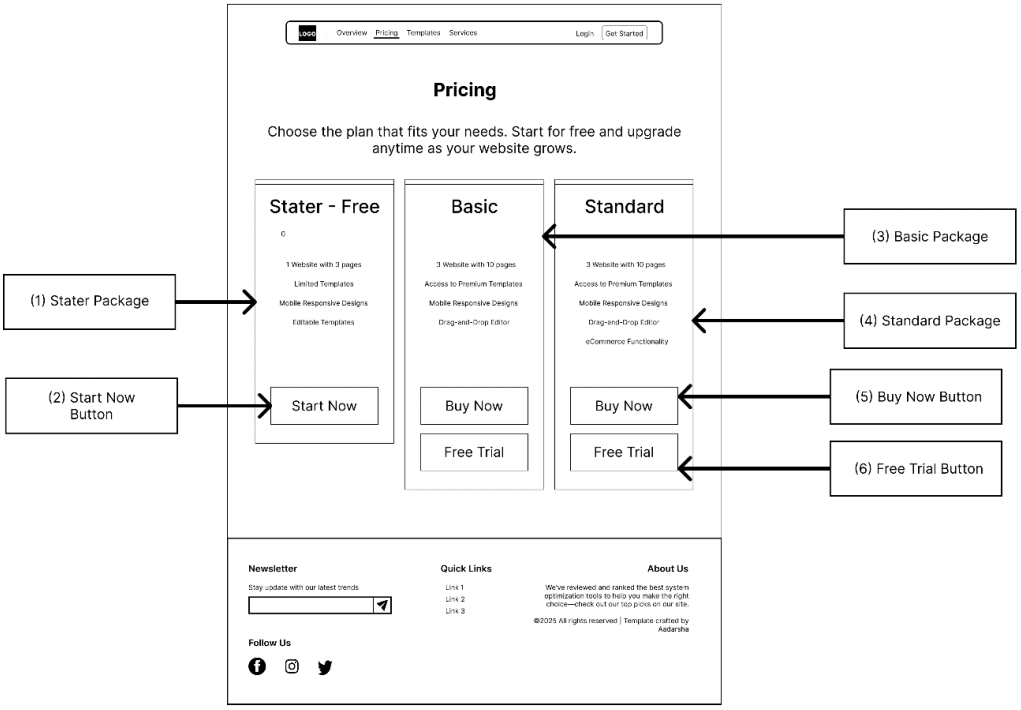


Figure 4.2 Wireframe of Pricing Page

Table 4-2: Label & Description for Wireframe of Website - Pricing Page

S.N.	**Label**	**Description**
1	Starter Package	Free entry plan to try the builder.
2	Start Now Button	Open sign-up for the free plan.

3	Basic Package	Mid-tier plan unlocking premium templates and more capacity; suited for individuals.
4	Standard Package	Highest plan with advanced features for teams.
5	Buy Now Button	Paid-plan checkout action.
6	Free Trial Button	Starts time-limited trial of a paid plan without immediate payment.

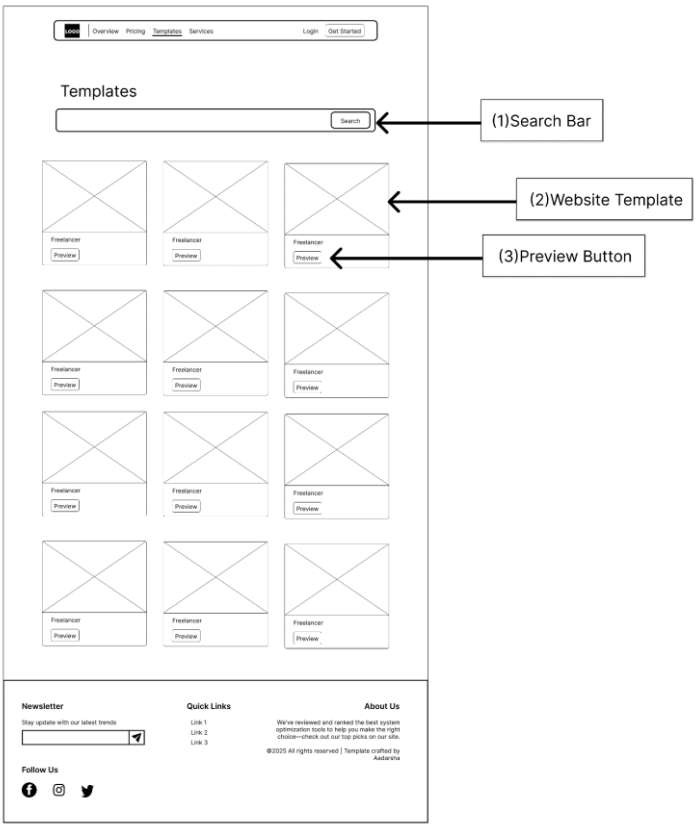


Figure 4.3: Wireframe of Templates Page

Table 4-3: Label & Description for Wireframe of Website - Templates Page

S.N.	Label	Description (short but clear)
1	Search Bar	Keyword field + Search action to filter the template grid by name, category, or tags.

2	Website Template	Clickable card in the grid showing a template thumbnail and name; opens details on selection.
3	Preview Button	Quick view of the template in a live/responsive preview (modal or new tab) without installing.

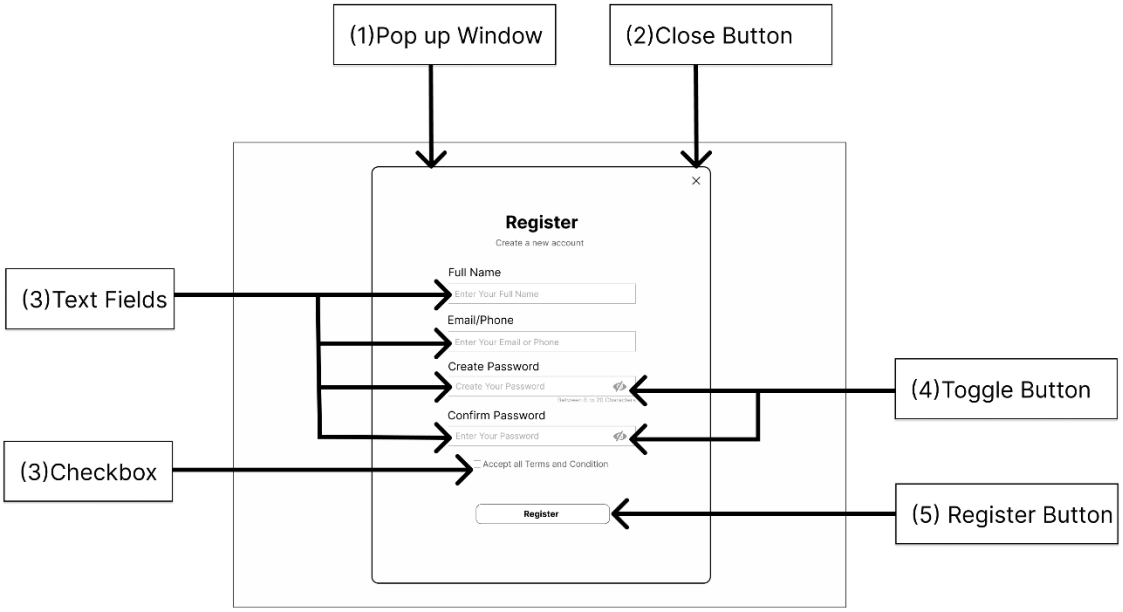


Figure 4.4: Wireframe of Register Modal

Table 4-4: Label & Description for Wireframe of Register Modal

S.N.	Label	Description
1	Pop up Window	Centered modal for account sign-up.
2	Close Button	Dismisses the modal; preserves entered data.

3	Text Fields	Full Name, Email/Phone, Password, Confirm Password with inline validation and placeholders.
4	Checkbox	“Accept Terms & Conditions” gate; must be checked to enable registration.
5	Toggle Button	Eye icon to show/hide password text for both password fields; keyboard accessible.
6	Register Button	Submits the form; disabled until fields are valid and terms accepted; shows loading on submit.

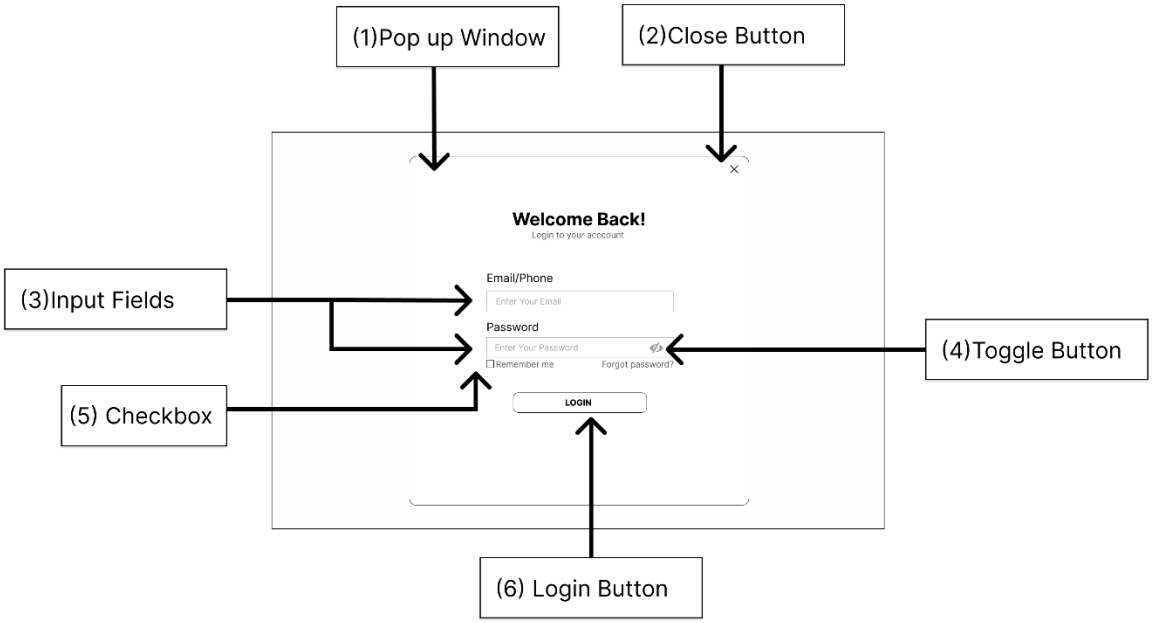


Figure 4.5: Wireframe of Login Modal

Table 4-5: Label & Description for Wireframe of Login Modal

S.N.	**Label**	**Description (short but clear)**
1	Pop up Window	Centered sign-in modal; blocks background and traps focus until closed.
2	Close Button	Dismisses the modal (click or Esc); keeps any typed values intact.
3	Input Fields	Email/Phone and Password with inline validation; supports browser autofill.

4	Toggle Button	Eye icon to show/hide password; keyboard-accessible and ARIA-labeled.
5	Checkbox	“Remember me” option to keep the user signed in on this device (optional).
6	Login Button	Submits credentials; disabled until valid; shows loading and surfaces error messages if any.

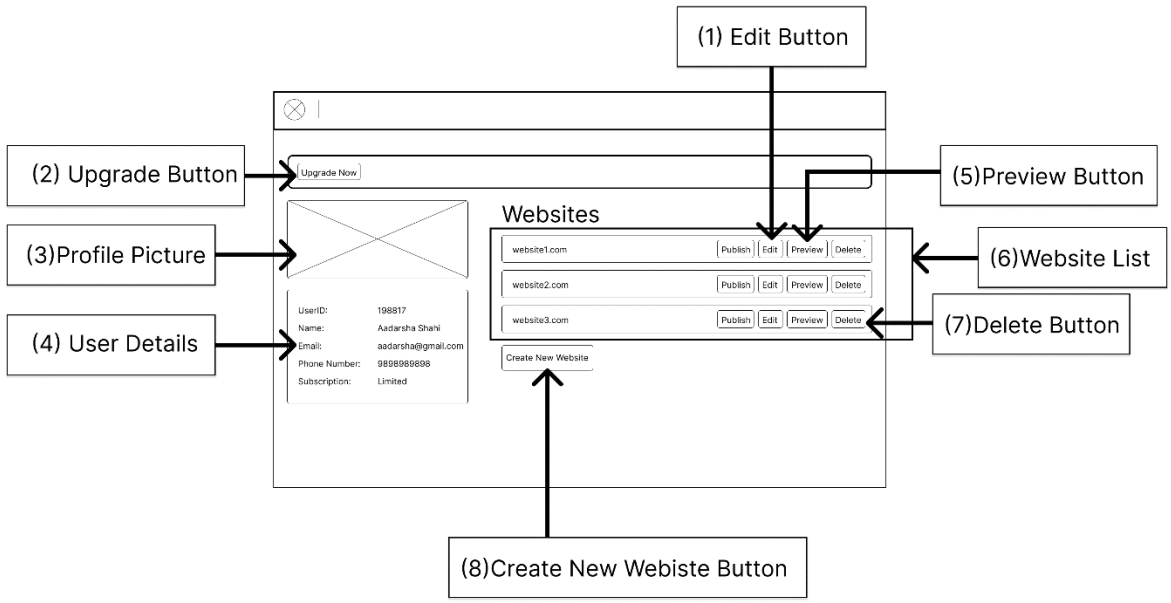


Figure 4.6: Wireframe of User Dashboard

Table 4-6: Label & Description for Wireframe of User Dashboard

S.N.	Label	Description
1	Edit Button	Opens the site in the builder to change pages, blocks, and settings.
2	Upgrade Button	Takes user to pricing/upgrade flow to raise plan limits and unlock features.
3	Profile Picture	User avatar; click to view/edit account profile and preferences.
4	User Details	Quick panel showing ID, name, email/phone, and current subscription status.
5	Preview Button	Launches responsive, non-published preview for the selected website.
6	Website List	Table of owned sites with domain/status and row actions.
7	Delete Button	Removes the selected website after confirmation; irreversible operation.

8	Create New Website	Starts a new site: choose template, set domain; counts against plan limits.

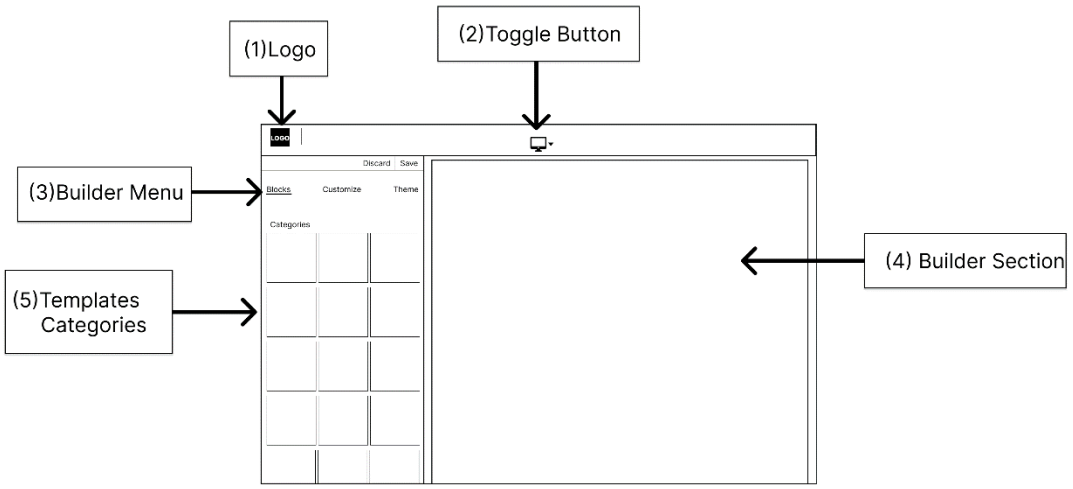


Figure 4.7: Wireframe of Website Builder

Table 4-7: Label & Description for Wireframe of Website Builder

S.N.	**Label**	**Description (short but clear)**
1	Logo	App brand mark; clickable shortcut back to dashboard/home.
2	Toggle Button	Switches responsive preview mode (desktop / tablet / mobile); can toggle preview frame.
3	Builder Menu	Top controls: Discard/Save and tabs (Blocks, Customize, Theme) for editing options.
4	Builder Section	Live canvas showing the page; drag-and-drop blocks and edit content in real time.
5	Template Categories	Left library of block categories; filter and drag items from here into the canvas.

4.2 Screen Mock-ups

Screen mock-ups communicate layout, hierarchy, and style before development, from low to high fidelity. They align stakeholders, accelerate feedback, guide engineering handoff, and are often static images or clickable prototypes.

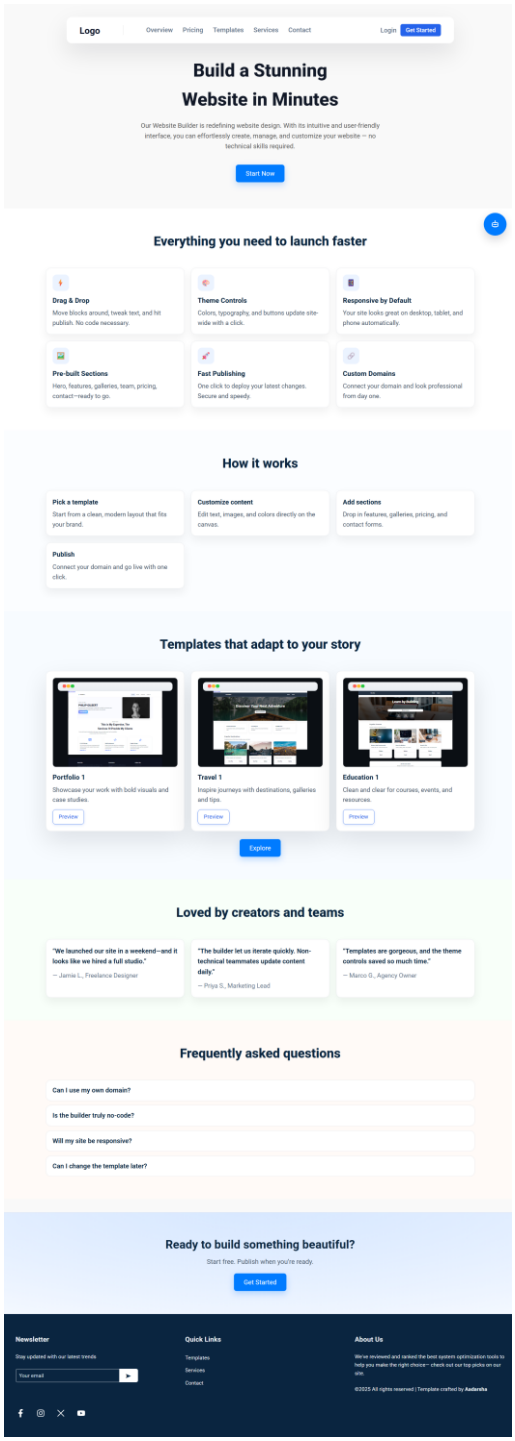


Figure 4.8: Screen Mockup of Landing Page

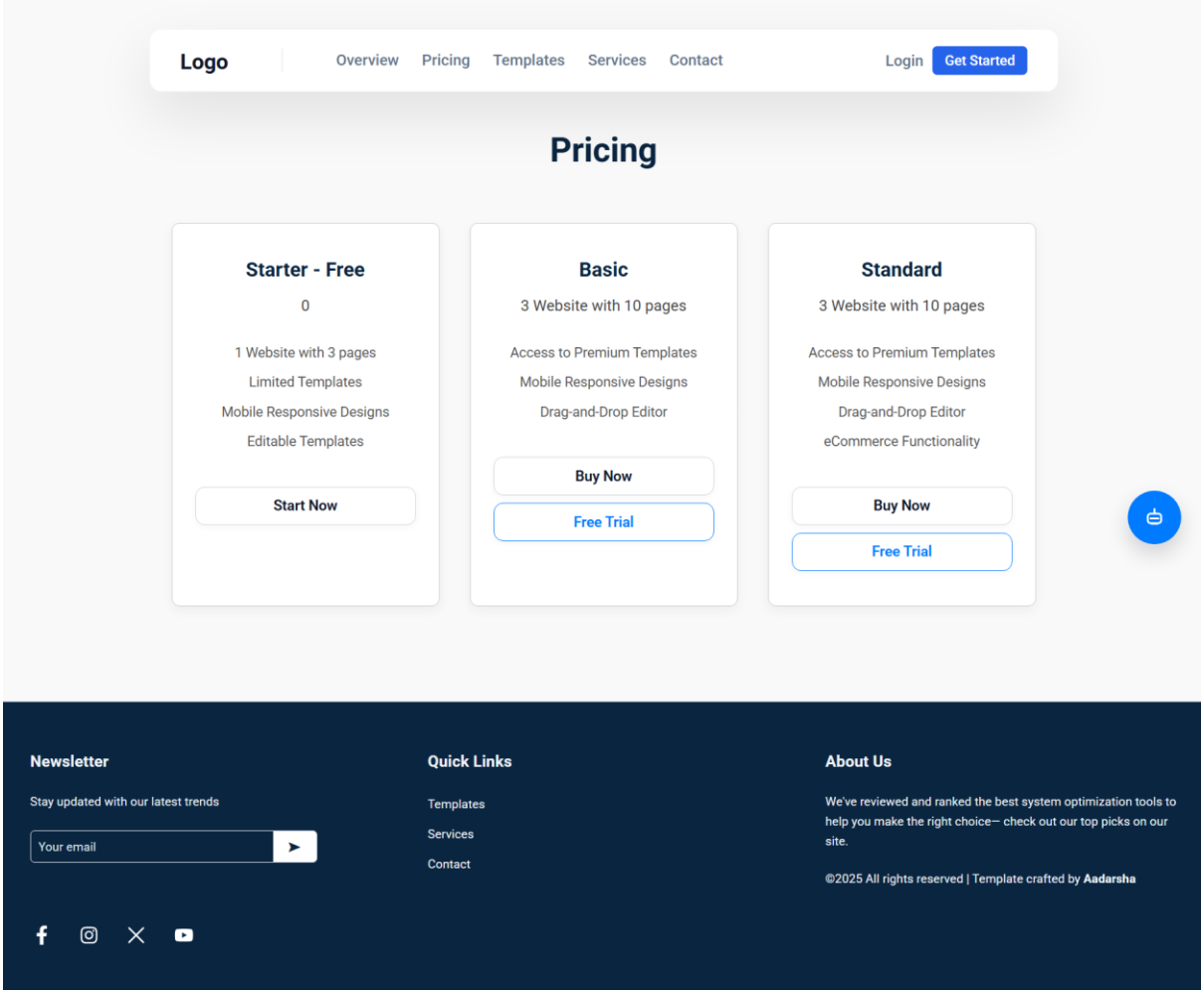


Figure 4.9: Screen Mockup of Pricing Page

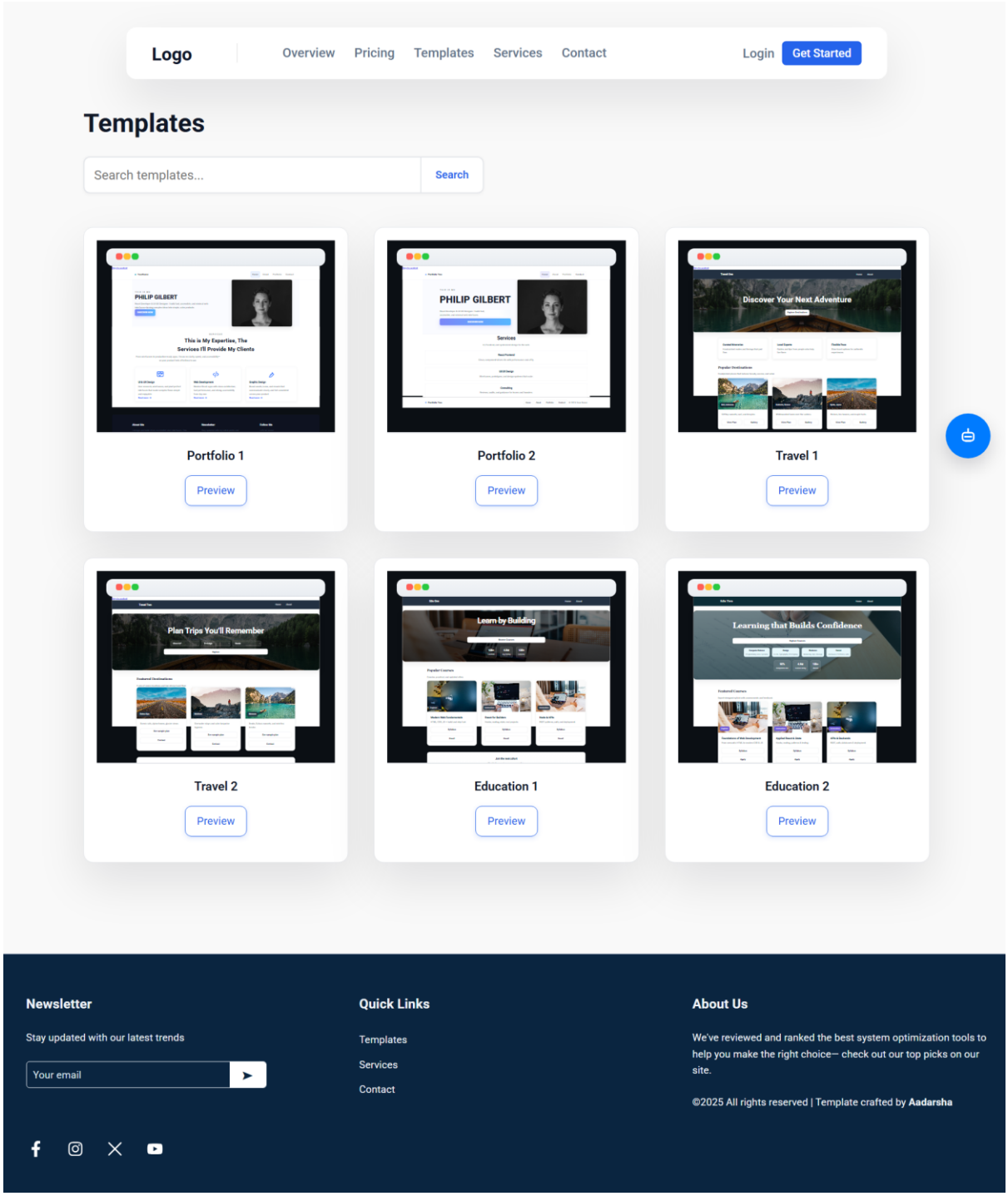


Figure 4.10: Screen Mockup of Pricing Page

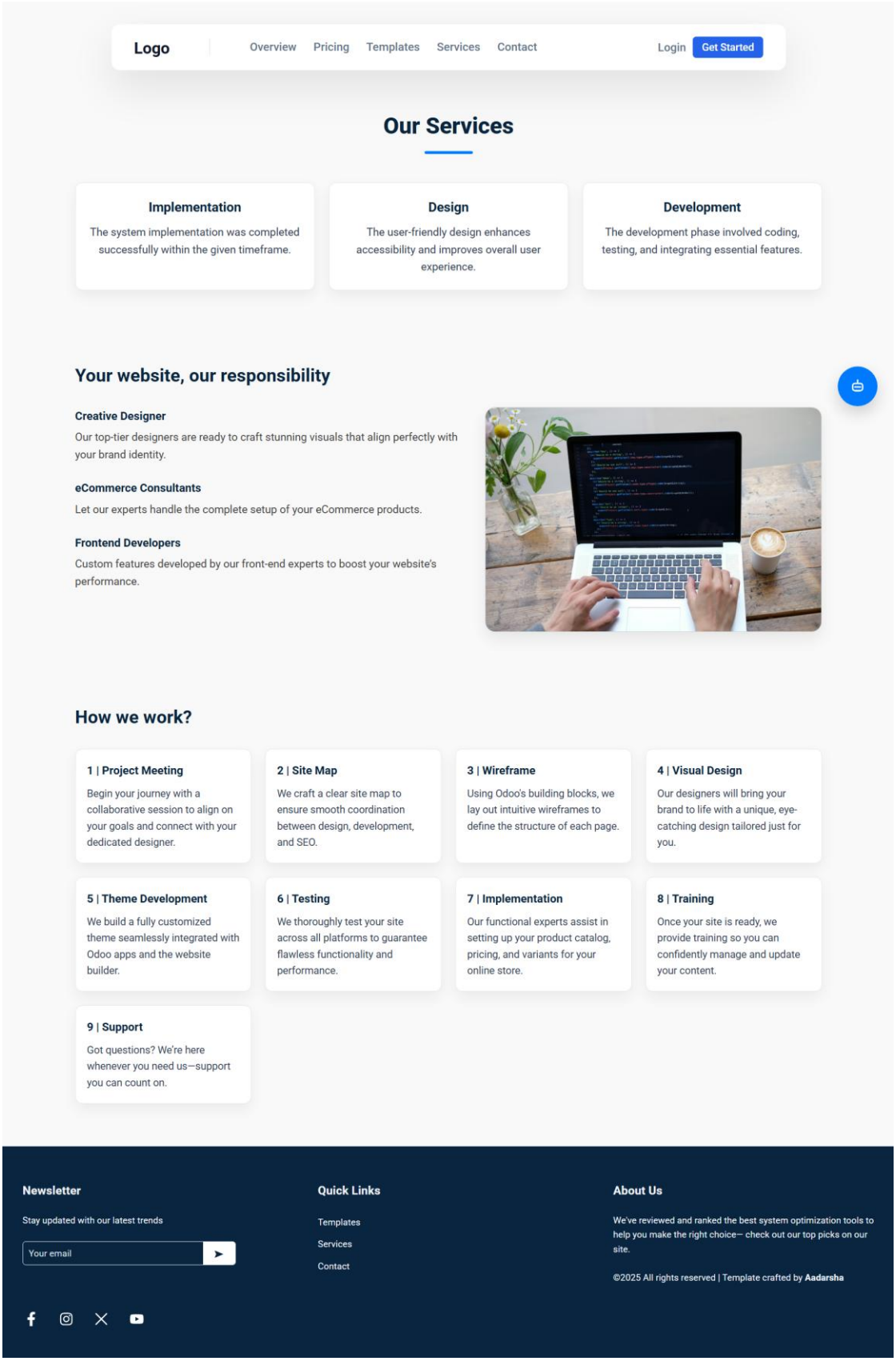


Figure 4.11: Screen Mockup of Service Page

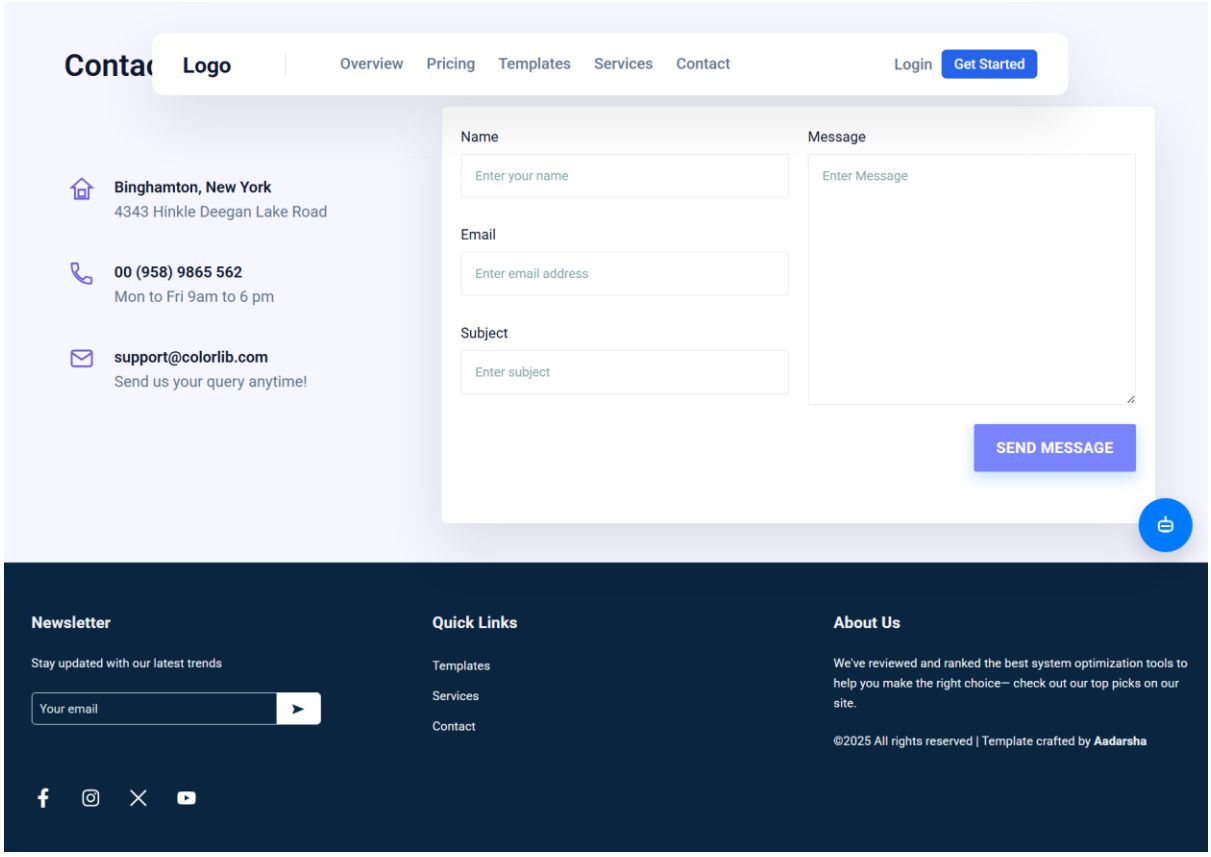


Figure 4.12: Screen Mockup of Contact Page

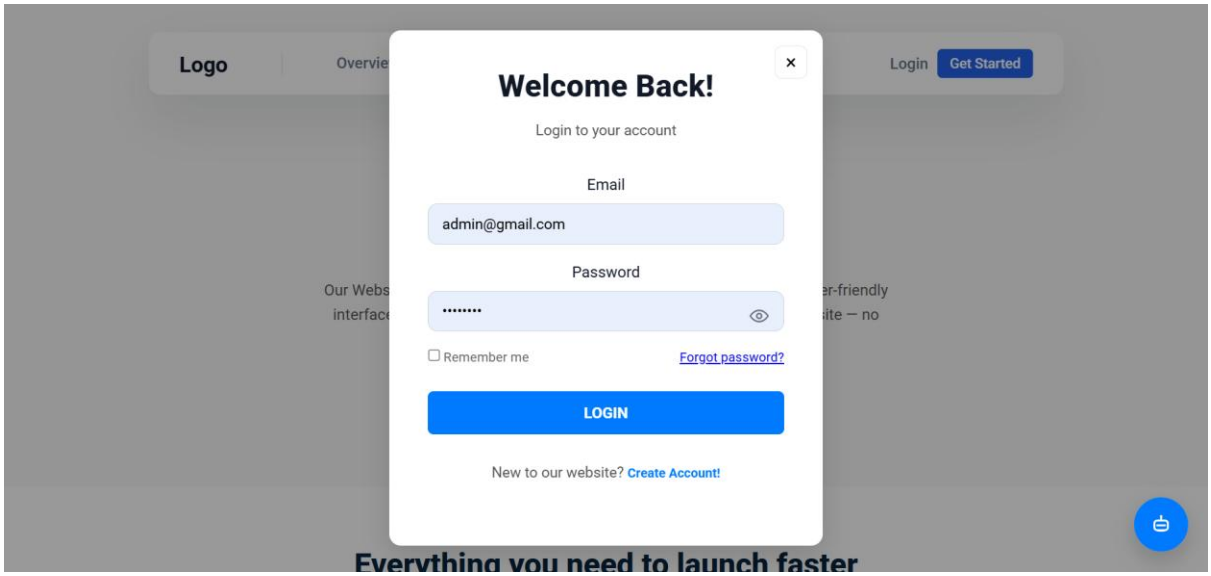


Figure 4.13: Screen Mockup of Login Modal

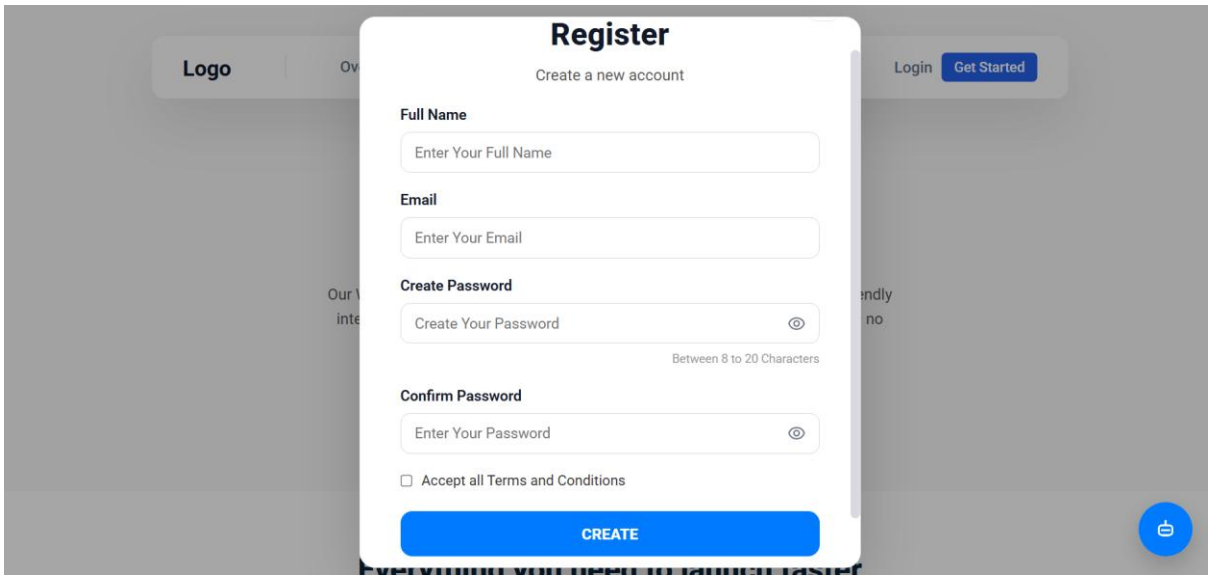


Figure 4.14: Screen Mockup of Register Modal

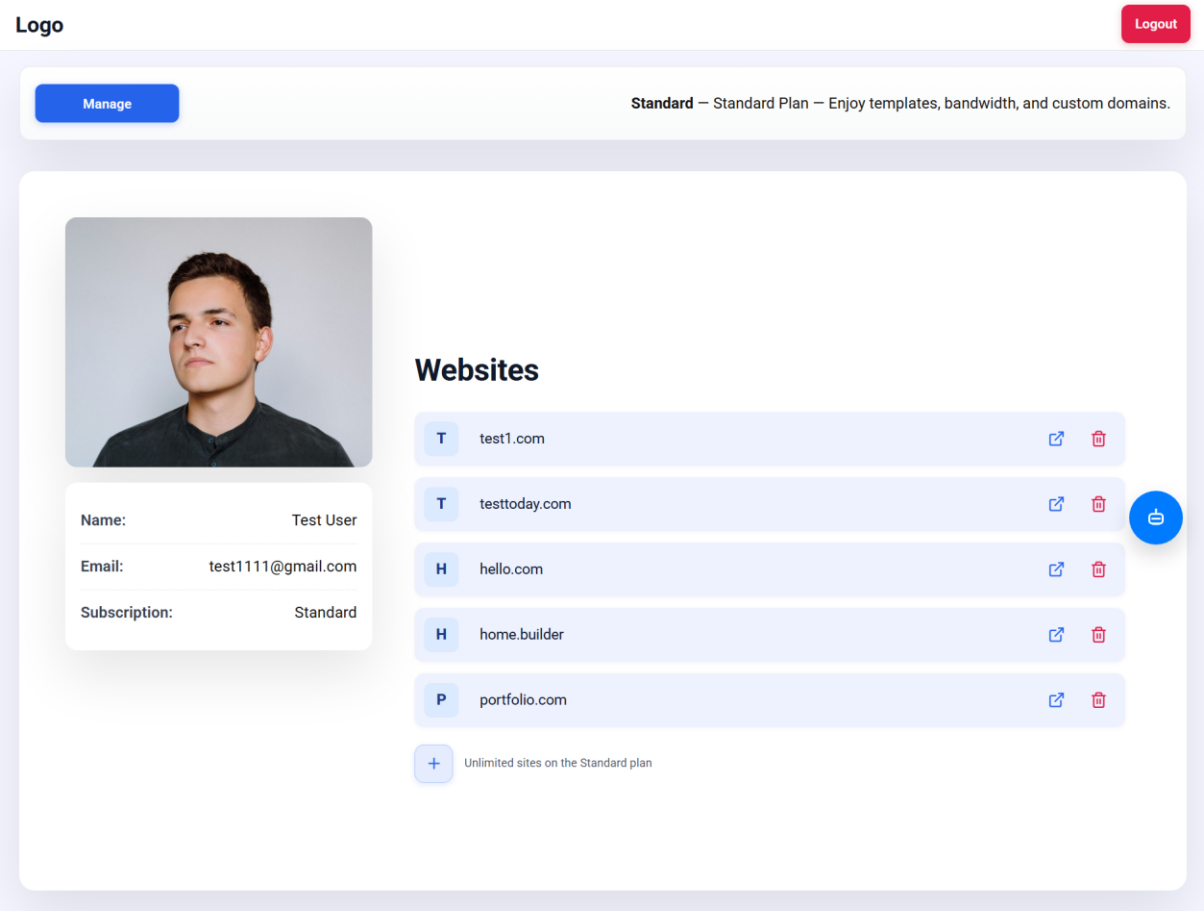


Figure 4.15: Screen Mockup of User Dashboard

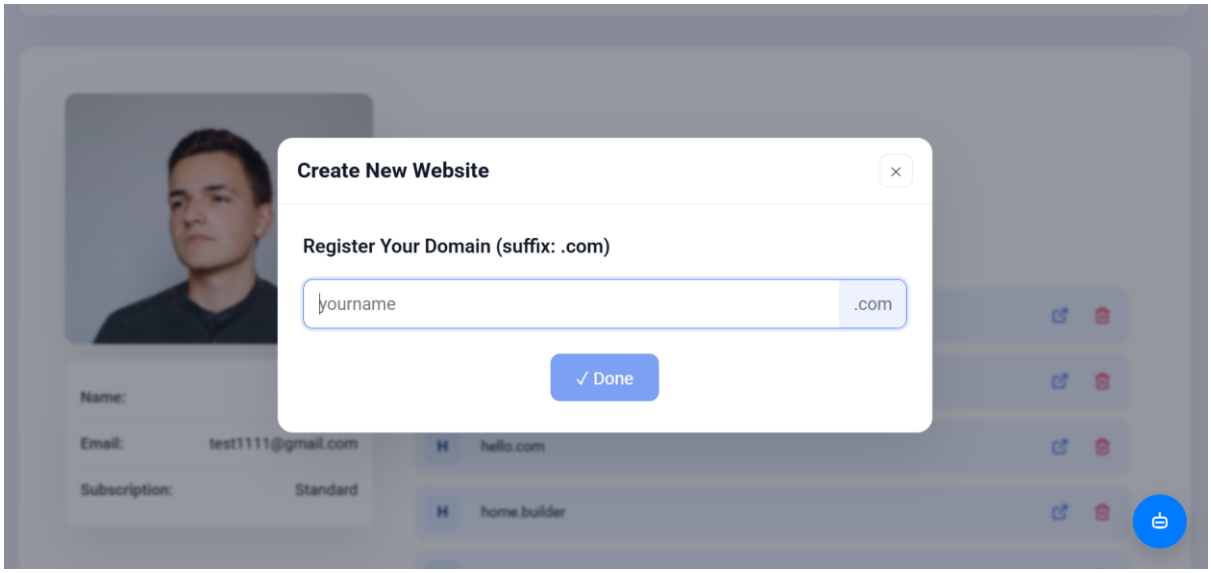


Figure 4.16: Screen Mockup of Create Website Modal

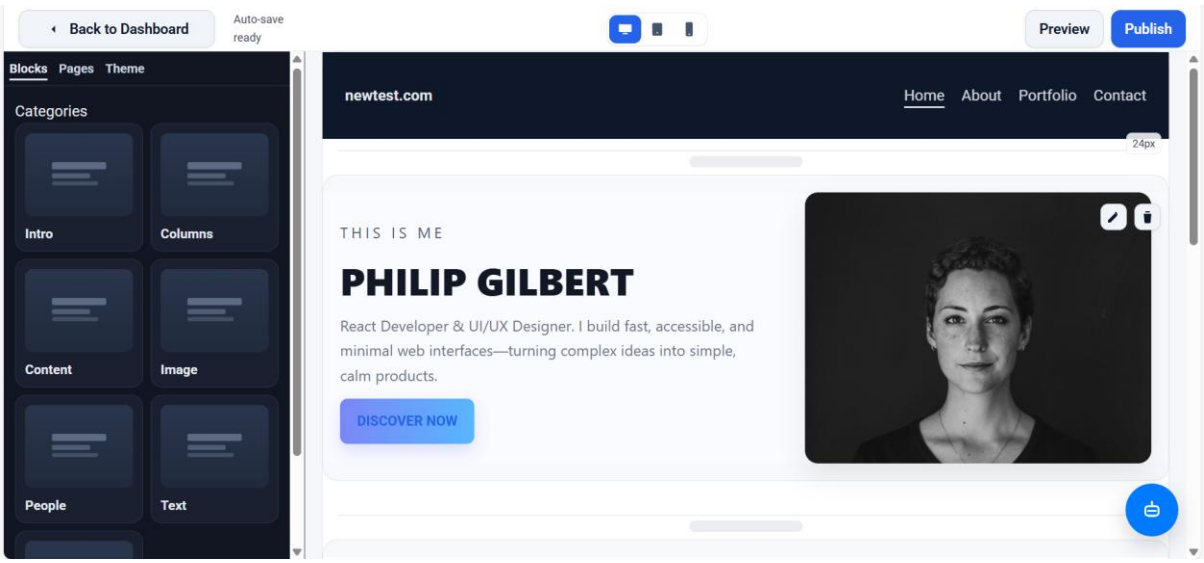


Figure 4.17: Screen Mockup of Website Builder

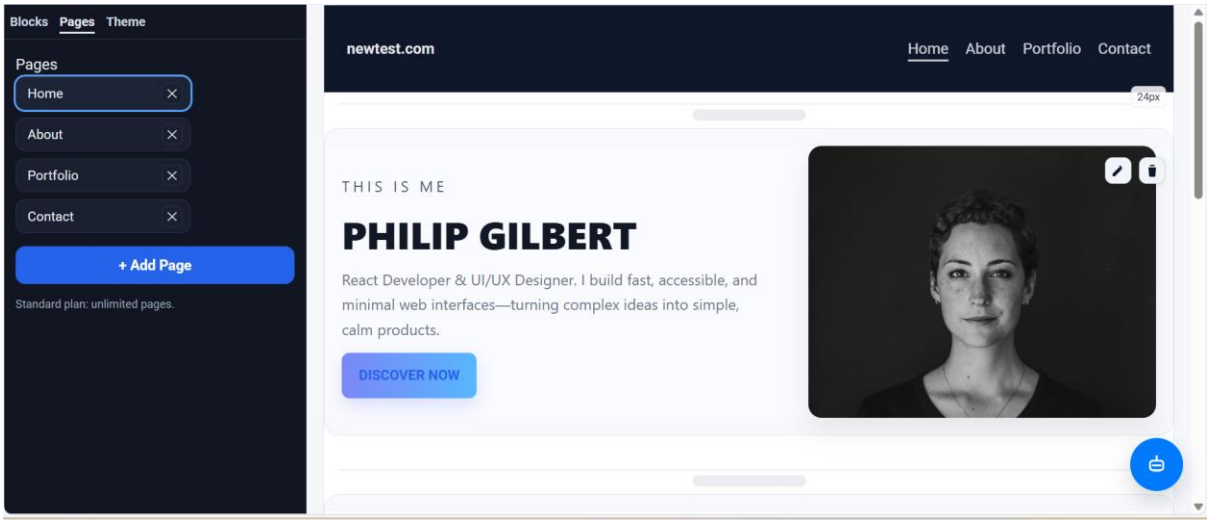


Figure 4.18: Screen Mockup of Page Panel

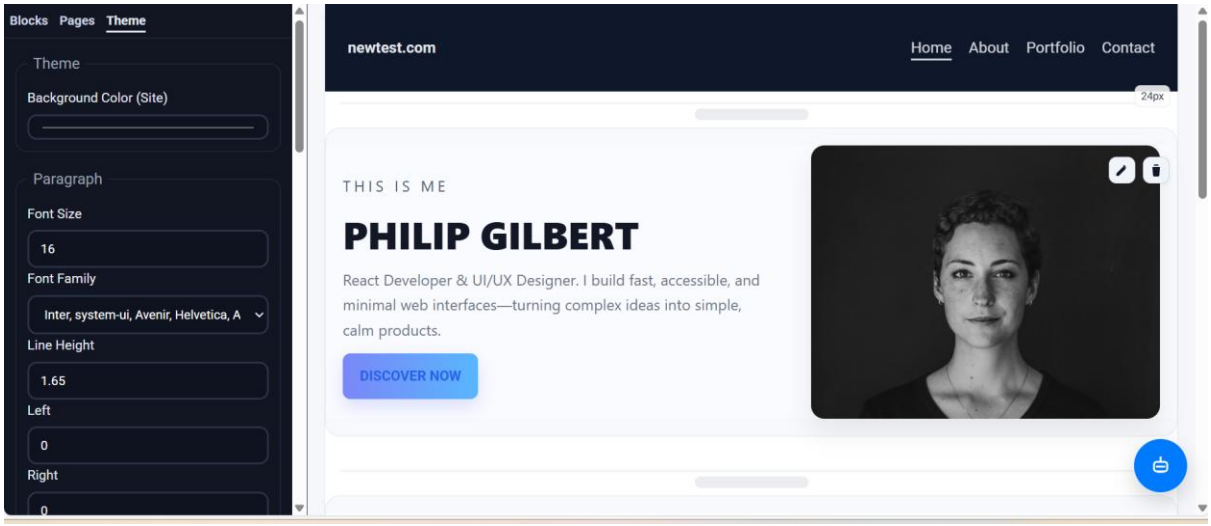


Figure 4.19: Screen Mockup of Theme Panel

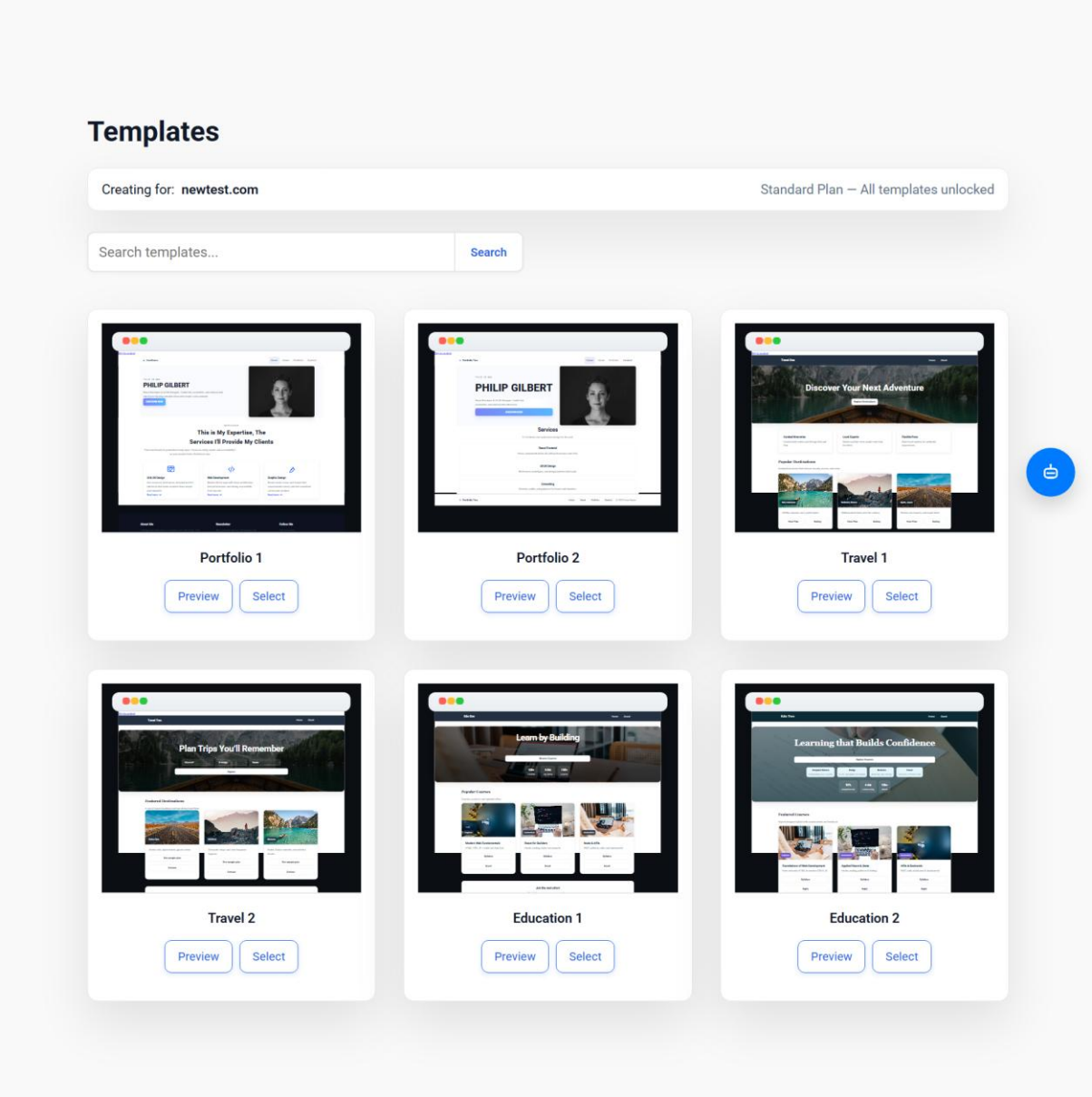


Figure 4.20: Screen Mockup of Templates Selection

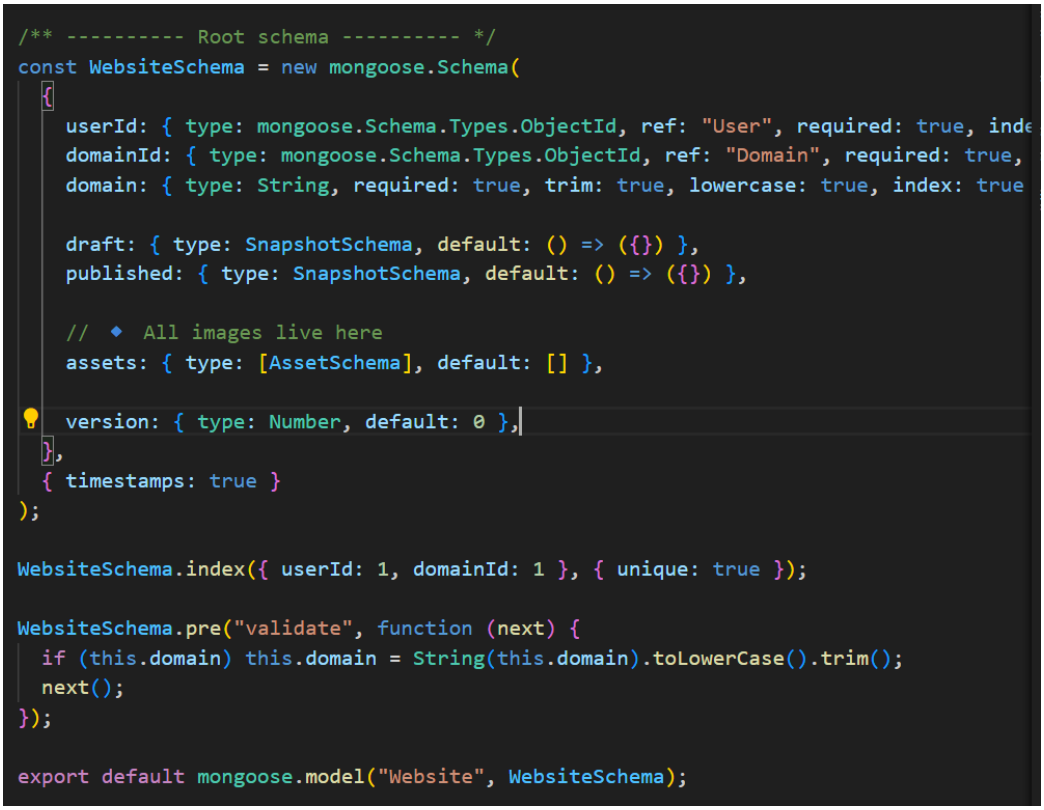
5 System Built & Technical Notes

System follows MVC on a web stack. Models use MongoDB/Mongoose (User, Website, Snapshot, Asset) for validation and persistence. Controllers are Express handlers enforcing auth, orchestrating logic, and exposing JSON APIs. Views are a React SPA consuming those APIs, managing editor state (editing, theming, autosave), and rendering a responsive UI. This separation yields maintainability, flexible documents, lightweight scalable endpoints, and a fast front end, enabling independent scaling and rapid iteration.

5.1 Technologies Used

Table 5-1: Table Mentioning Technology Used

Technology	Primary Role	Typical Use in This Project	Key Benefits
React (View)	Client-side UI and state	Builder interface, drag-and-drop canvas, theme controls, preview	Fast SPA, component reusability, hooks, virtual DOM
Express (Controller)	HTTP API & routing	Auth routes, website CRUD, publish/unpublish, image upload endpoints	Minimal, flexible middleware, easy REST design
MongoDB (Model)	Data storage (documents)	Users, Websites, Snapshots, Assets, Subscriptions	Schema-flexible, JSON-like docs, good for iterative content structures



```
/** ---------- Root schema ---------- */
const WebsiteSchema = new mongoose.Schema(
  {
    userId: { type: mongoose.Schema.Types.ObjectId, ref: "User", required: true, inde
    domainId: { type: mongoose.Schema.Types.ObjectId, ref: "Domain", required: true,
    domain: { type: String, required: true, trim: true, lowercase: true, index: true

    draft: { type: SnapshotSchema, default: () => ({}) },
    published: { type: SnapshotSchema, default: () => ({}) },

    //  ◆  All images live here
    assets: { type: [AssetSchema], default: [] },

    version: { type: Number, default: 0 },
  },
  { timestamps: true }
);

WebsiteSchema.index({ userId: 1, domainId: 1 }, { unique: true });

WebsiteSchema.pre("validate", function (next) {
  if (this.domain) this.domain = String(this.domain).toLowerCase().trim();
  next();
});

export default mongoose.model("Website", WebsiteSchema);
```

Figure 5.1: Model Class Implementation Example of Website Model

Example Mongoose model for the website builder: a root Website document that embeds everything needed to edit/publish. It composes SiteMeta, Block, Page, Snapshot and Asset. Constraints: slug/enums; pre-validate lowercases domain; indexes ensure domain and user+domain uniqueness. Autosave writes draft.builderState and metadata; publish promotes a consistent Snapshot with version/timestamp; live site renders from published snapshot; assets stream from embedded store.

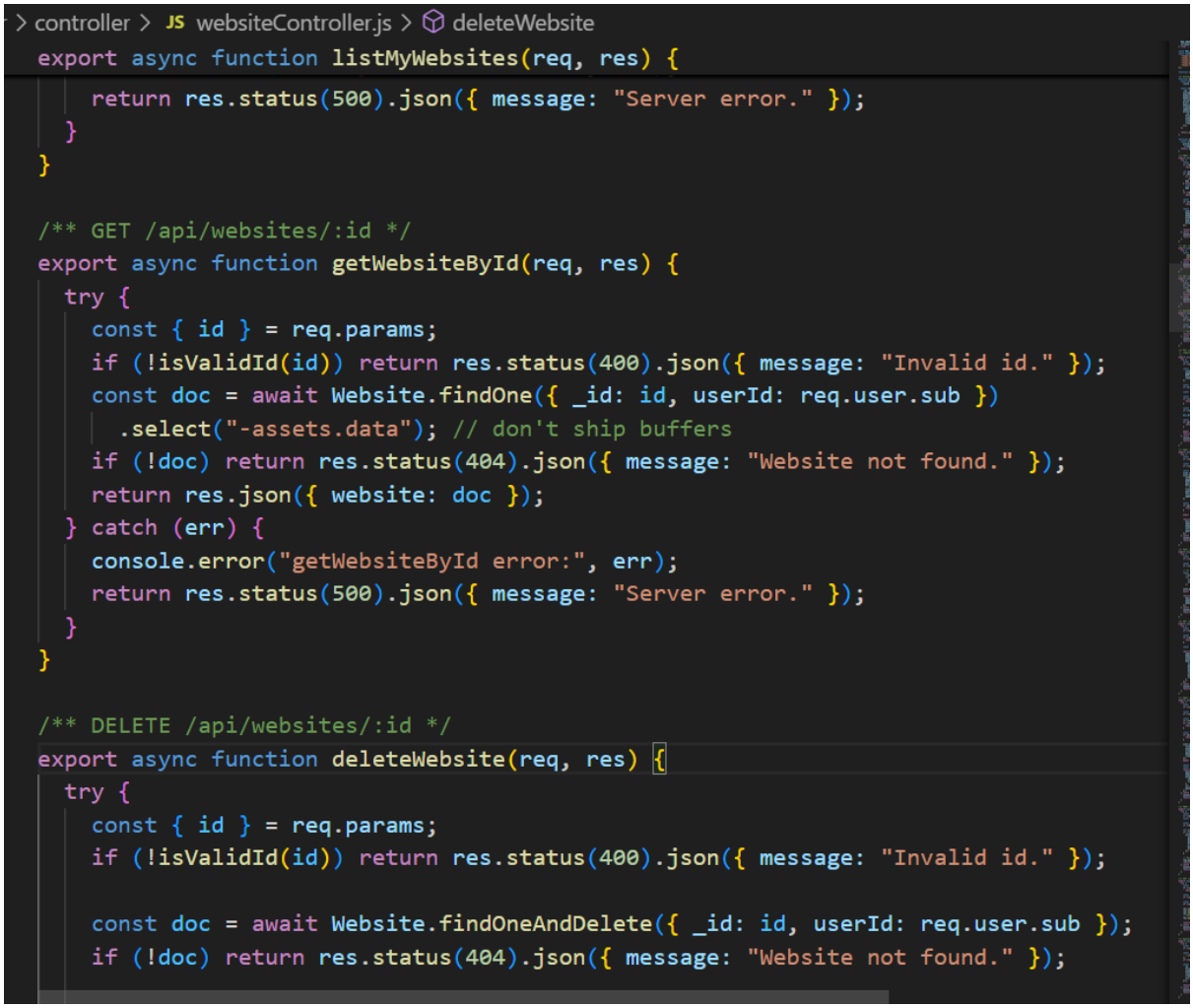


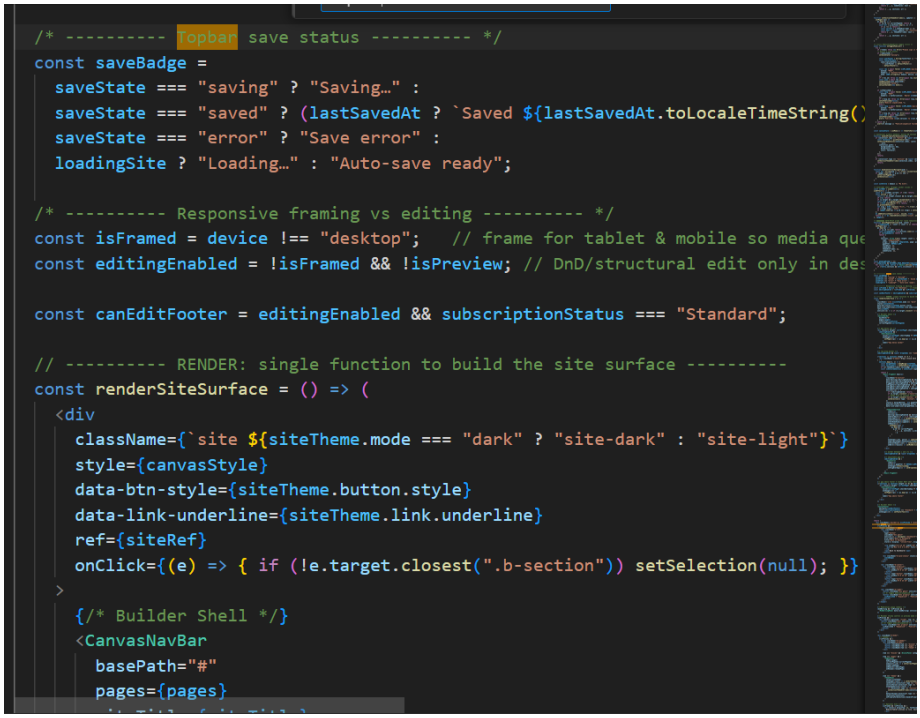
Figure 5.2:Model Class Implementation Example of Website Controller

Controller manages authenticated website workflow: create/list/get/delete sites; save validated draft snapshots (theme, pages, device, footer) with stripped data-URLs and strict size limits; publish/unpublish along with versioning and timestamps. A public endpoint serves published snapshots by domain. Image uploads are MIME- and size-checked; assets listed or streamed with proper headers and CORS. All queries validate ObjectIds and enforce ownership (req.user.sub).



Figure 5.3:Model Class Implementation Example of Website Route

Express router centralizes website APIs. Public routes: GET /public/:domain (published snapshot) and GET /:id/assets/:assetId/raw (serve file); all others require auth. Auth users can create, list, read, delete, draft-save, publish, and unpublish sites. Multer memory, 6 MB, handles image uploads via POST /:id/assets or /upload and supports asset metadata listing. Controllers implement logic.



```
/* ---------- Topbar save status ---------- */
const saveBadge =
  saveState === "saving" ? "Saving…" :
  saveState === "saved" ? (lastSavedAt ? `Saved ${lastSavedAt.toLocaleTimeString()
  saveState === "error" ? "Save error" :
  loadingSite ? "Loading…" : "Auto-save ready";

/* ---------- Responsive framing vs editing ---------- */
const isFramed = device !== "desktop";   // frame for tablet & mobile so media que
const editingEnabled = !isFramed && !isPreview; // DnD/structural edit only in des

const canEditFooter = editingEnabled && subscriptionStatus === "Standard";

// ---------- RENDER: single function to build the site surface ----------
const renderSiteSurface = () => (
  <div
    className={`site ${siteTheme.mode === "dark" ? "site-dark" : "site-light"}`}
    style={canvasStyle}
    data-btn-style={siteTheme.button.style}
    data-link-underline={siteTheme.link.underline}
    ref={siteRef}
    onClick={(e) => { if (!e.target.closest(".b-section")) setSelection(null); }}
  >
    {/* Builder Shell */}
    <CanvasNavBar
      basePath="#"
      pages={pages}
```

Figure 5.4: Example of Implementation of View Layer – Builder (1)

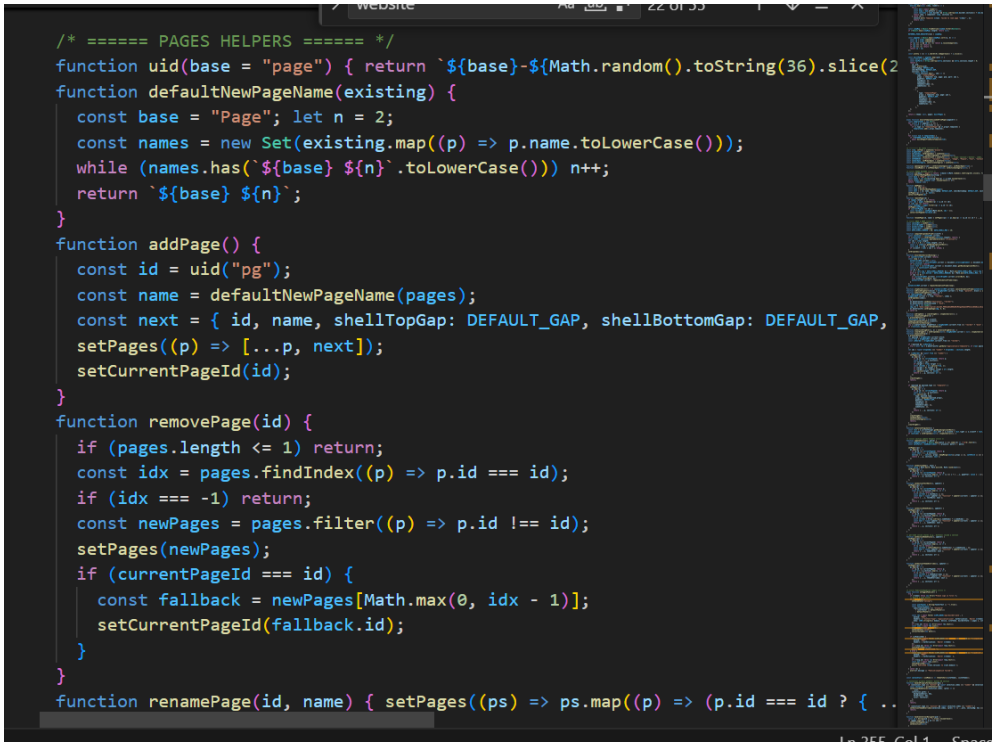


```
{tab === "blocks" && <BlocksPanel categories={CATEGORIES} onOpenCategory

{tab === "pages" && (
  <PagesPanel
    pages={pages}
    currentPageId={currentPageId}
    onSelect={(id) => { setCurrentPageId(id); setSelection(null); }}
    onAdd={addPage}
    onRemove={removePage}
    onRename={renamePage}
  />
)}

{tab === "theme" && (
  <ThemePanel
    value={siteTheme}
    onChange={(next) => { setSiteTheme(next); }}
    canSection={selection?.type === "section"}
    sectionValue={selection?.type === "section" ? (sections[selection.in
    onChangeSection={(next) => {
      if (selection?.type !== "section") return;
      setSectionThemeOverrides(selection.index, next);
    }}
    defaultScope={selection?.type === "section" ? "section" : "site"}
    selection={selection}
    onPrimaryForSelection={handlePrimaryForSelection}
  />
```

Figure 5.5: Example of Implementation of View Layer - Builder(2)



```
/* ====== PAGES HELPERS ====== */
function uid(base = "page") { return `${base}-${Math.random().toString(36).slice(2
function defaultNewPageName(existing) {
  const base = "Page"; let n = 2;
  const names = new Set(existing.map((p) => p.name.toLowerCase()));
  while (names.has(`${base} ${n}`.toLowerCase())) n++;
  return `${base} ${n}`;
}
function addPage() {
  const id = uid("pg");
  const name = defaultNewPageName(pages);
  const next = { id, name, shellTopGap: DEFAULT_GAP, shellBottomGap: DEFAULT_GAP,
  setPages((p) => [...p, next]);
  setCurrentPageId(id);
}
function removePage(id) {
  if (pages.length <= 1) return;
  const idx = pages.findIndex((p) => p.id === id);
  if (idx === -1) return;
  const newPages = pages.filter((p) => p.id !== id);
  setPages(newPages);
  if (currentPageId === id) {
    const fallback = newPages[Math.max(0, idx - 1)];
    setCurrentPageId(fallback.id);
  }
}
function renamePage(id, name) { setPages((ps) => ps.map((p) => (p.id === id ? { ..
```

Figure 5.6: Example of Implementation of View Layer - Builder(3)

View layer uses reusable React components: BlocksPanel and CategoryModal (plan-filtered templates in sandboxed PreviewFrame); PagesPanel (page CRUD); ThemePanel (CSS-variable theming); TemplateSection (inline edits, style tweaks, image replace/hide, moderation); CanvasNavBar/Footer; DeviceFrame; and Builder orchestrating state, drag-drop, autosave, plan checks, uploads, publishing.

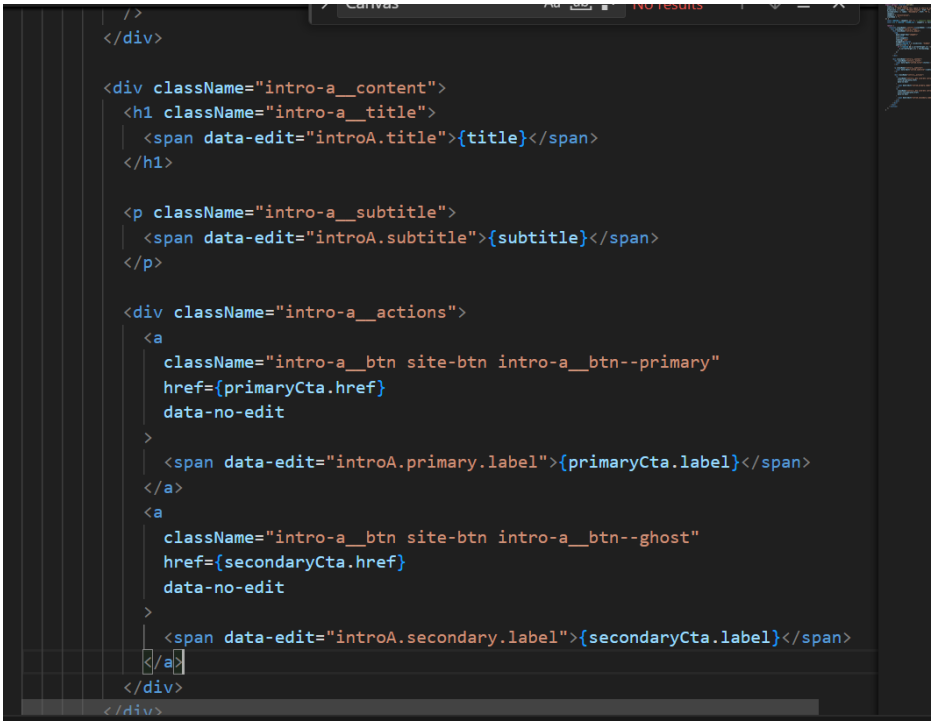


Figure 5.7: Example Implementation of Templates File – IntroA

IntroA is a reusable hero block: image, title, subtitle, and two CTAs, all prop-driven for reuse. Text spans expose data-edit keys for inline editing; the image uses data-image-key='imageSrc' and swaps to a 1×1 GIF when removed to preserve layout. UX guards include lazy loading and nondraggable media. Buttons are data-no-edit. Styles live in introA.css; accessibility uses semantic tags and aria-labels.

5.2 Third Party Integration

```
export async function getAccessToken() {
  const base = API_BASES[PAYPAL_ENV] || API_BASES.sandbox;
  const auth = Buffer.from(`${PAYPAL_CLIENT_ID}:${PAYPAL_CLIENT_SECRET}`).toString("b

  const res = await fetch(`${base}/v1/oauth2/token`, {
    method: "POST",
    headers: {
      Authorization: `Basic ${auth}`,
      "Content-Type": "application/x-www-form-urlencoded",
    },
    body: "grant_type=client_credentials",
  });

  const data = await res.json();
  if (!res.ok) {
    throw new Error(data?.error_description || "Failed to get PayPal access token");
  }
  return { accessToken: data.access_token, base };
}

export async function createOrder({ amount = PAYMENT_PRICE, currency = PAYMENT_CURREN
  const { accessToken, base } = await getAccessToken();

  const res = await fetch(`${base}/v2/checkout/orders`, {
    method: "POST",
    headers: {
      Authorization: `Bearer ${accessToken}`,
      "Content-Type": "application/json",
```

Figure 5.8: Third Party (PayPal) Integration From Backend

```
// src/components/PayPalModal.js
import React, { useEffect, useRef, useState } from "react";
import { API_BASE } from "../config";
import { toast } from "react-toastify";

// Load SDK once (shared promise)
let paypalSDKPromise;
function ensurePayPalSDK(clientId) {
  if (window.paypal) return Promise.resolve(window.paypal);
  if (!paypalSDKPromise) {
    paypalSDKPromise = new Promise((resolve, reject) => {
      const s = document.createElement("script");
      s.src = `https://www.paypal.com/sdk/js?client-id=${encodeURIComponent(clientId)}&currency=USD`;
      s.onload = () => resolve(window.paypal);
      s.onerror = () => reject(new Error("Failed to load PayPal SDK"));
      document.body.appendChild(s);
    });
  }
  return paypalSDKPromise;
}

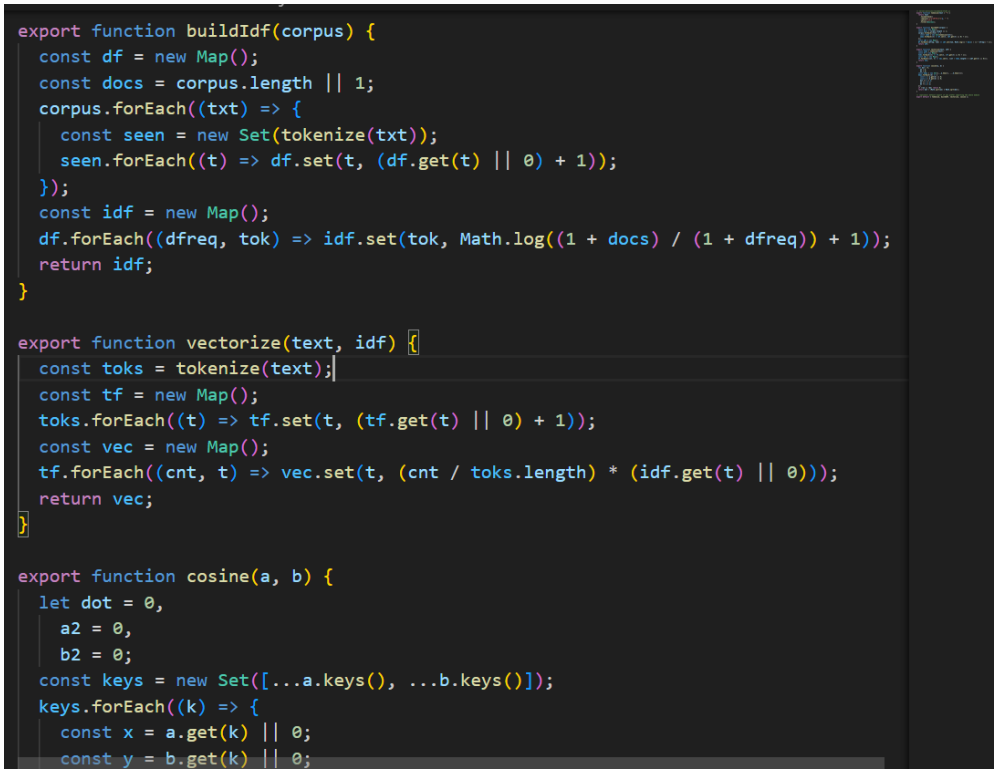
export default function PayPalModal({ isOpen, onClose, token, onUpgraded, clientId: clientIdProp }) {
  const [clientId, setClientId] = useState(clientIdProp || "");
  const [error, setError] = useState("");
  const [loading, setLoading] = useState(true);

  const containerRef = useRef(null);
  const renderedRef = useRef(false);
  const closingRef = useRef(false);
```

Figure 5.9: Third Party (PayPal) Integration from Frontend

The system integrates PayPal via a client–server flow. The frontend loads the PayPal SDK, requests the client ID from the API, creates the order, and captures it after approval. The backend uses env keys to obtain a token and call Orders API. On success, it grants a 30-day Standard subscription. Routes are JWT-protected; .env sets mode, price, currency; errors are user-friendly.

5.3 Recommendation System



```
export function buildIdf(corpus) {
  const df = new Map();
  const docs = corpus.length || 1;
  corpus.forEach((txt) => {
    const seen = new Set(tokenize(txt));
    seen.forEach((t) => df.set(t, (df.get(t) || 0) + 1));
  });
  const idf = new Map();
  df.forEach((dfreq, tok) => idf.set(tok, Math.log((1 + docs) / (1 + dfreq)) + 1));
  return idf;
}

export function vectorize(text, idf) {
  const toks = tokenize(text);
  const tf = new Map();
  toks.forEach((t) => tf.set(t, (tf.get(t) || 0) + 1));
  const vec = new Map();
  tf.forEach((cnt, t) => vec.set(t, (cnt / toks.length) * (idf.get(t) || 0)));
  return vec;
}

export function cosine(a, b) {
  let dot = 0,
    a2 = 0,
    b2 = 0;
  const keys = new Set([...a.keys(), ...b.keys()]);
  keys.forEach((k) => {
    const x = a.get(k) || 0;
    const y = b.get(k) || 0;
```

Figure 5.10: Recommendation System Example of IDF, Vectorize & Cosine

```
export function scoreRules(tpl, a) {
  let s = 0;
  const low = v => String(v || "").toLowerCase();
  const has = (xs, v) => Array.isArray(xs) && xs.map(low).includes(low(v));

  if (a.purpose && has(tpl.fits?.purpose, a.purpose)) s += W.purpose;
  if (a.industry && has(tpl.fits?.industries, a.industry)) s += W.industry;
  if (a.tone && has(tpl.fits?.tone, a.tone)) s += W.tone;

  if (a.purpose && a.industry) {
    const m = PURPOSE_INDUSTRY[low(a.purpose)]?.[low(a.industry)];
    if (m) s += m;
  }

  if (a.colors === "dark" && tpl.palette?.mode === "dark") s += W.colorMatch;
  if (a.colors === "light" && tpl.palette?.mode === "light") s += W.colorMatch;
  if (a.colors === "dark" && tpl.palette?.mode === "light") s += W.hardPenaltyDarkMis
  if (a.colors === "light" && tpl.palette?.mode === "dark") s += W.hardPenaltyDarkMis

  const featuresReq = (a.features || []).map(low);
  const pagesReq = (a.pages || []);
  const featOverlap = featuresReq.filter(f => (tpl.features || []).map(low).includes(
  s += featOverlap * W.featureMatch;

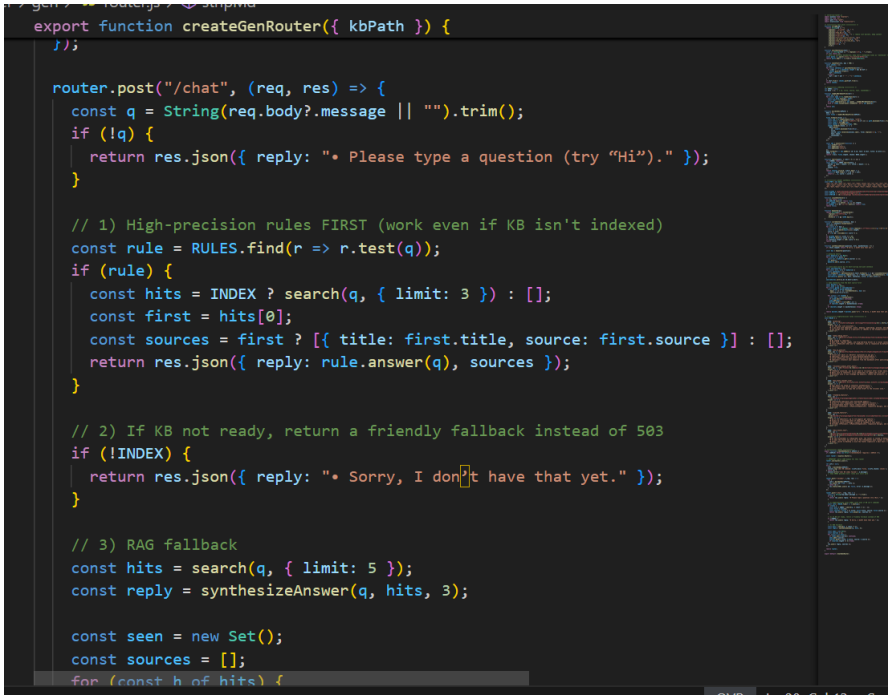
  const pageOverlap = pagesReq.filter(p => (tpl.pages || []).includes(p)).length;
  s += Math.min(pageOverlap, W.pageMatchCap) * W.pageMatch;

  if (pctOverlap(featuresReq, (tpl.features || []).map(low)) >= 0.7) s += W.coverageE
  if (pctOverlap(pagesReq, tpl.pages || []) >= 0.7) s += W.coveragePagesBonus;
```

Figure 5.11: Recommendation System Implementation Example of Score Rule

The hybrid recommender combines rules, semantics, and optional learning. From user answers it derives purpose, industry, vibe, emphasis, color mode, pages/features, booking/teaching flags. It computes rule scores for weights, bonuses, penalties and semantic scores via TF-IDF + cosine; a learned slot is reserved. Final rank: 0.55×rule + 0.30×semantic + 0.15×learned, with jitter tie-breaks, “why” reasons, diversify(), and composeFromComponents() applying colors/features and layout.

5.4 Chatbot



```
export function createGenRouter({ kbPath }) {
});

router.post("/chat", (req, res) => {
  const q = String(req.body?.message || "").trim();
  if (!q) {
    return res.json({ reply: "• Please type a question (try “Hi”)." });
  }

  // 1) High-precision rules FIRST (work even if KB isn't indexed)
  const rule = RULES.find(r => r.test(q));
  if (rule) {
    const hits = INDEX ? search(q, { limit: 3 }) : [];
    const first = hits[0];
    const sources = first ? [{ title: first.title, source: first.source }] : [];
    return res.json({ reply: rule.answer(q), sources });
  }

  // 2) If KB not ready, return a friendly fallback instead of 503
  if (!INDEX) {
    return res.json({ reply: "• Sorry, I don't have that yet." });
  }

  // 3) RAG fallback
  const hits = search(q, { limit: 5 });
  const reply = synthesizeAnswer(q, hits, 3);

  const seen = new Set();
  const sources = [];
  for (const h of hits) {
```

Figure 5.12: Example Implementation of Chatbot Implementation Using RAG

Chatbot uses a rules-first, lexical RAG pipeline. High-precision regex intents (greetings, publishing, plans) yield concise canned replies. Otherwise, ElasticLunr retrieves top Markdown chunks. A lightweight extractive summarizer scores sentences by keyword overlap, action verbs, preferred product terms, readability, and noise filtering. The bot returns the best 2–3 bullet sentences with source titles. The process is deterministic, fast, private, fully local, and updates through simple re-indexing.

5.5 Content Moderation System

```
function loadList(name) {
  return raw.split("\n").map(s => s.trim()).filter(Boolean);
}

/** Compile strings → RegExp once (case-insensitive unless provided). */
function compileRegex(list) {
  return (list || []).map((s) => {
    if (s instanceof RegExp) return s;
    const m = String(s).match(/^\/(.+)\/([a-z]*)$/i); // allow /.../gi notation
    return m ? new RegExp(m[1], m[2]) : new RegExp(String(s), "i");
  });
}

/** Build the in-memory policy by merging disk files with defaults. */
export function buildPolicy(defaults = {}) {
  // Top-level YAML (small stuff + pointers/overrides)
  const policyYml = loadYAML("policy.yaml");

  // Optional external files that can override/extend YAML
  // Prefer slurs.txt, but fall back to slur.txt for backward compatibility.
  let slurs = loadList("slur.txt");
  if (!slurs.length) slurs = loadList("slur.txt");

  const allowlist = loadList("allowlist.txt");
  const blocklist = loadList("blocklist.txt");
  const clipYml = loadYAML("clip.yaml");
  const brands = loadJSON("phishing_brands.json");

  // Merge (disk > YAML > defaults)
```

Figure 5.13: Example Implementation of Content Moderation Using Regex Engine

The moderation uses a hybrid, policy-control pipeline producing allow/review/block verdicts for text and images. For text, we normalize and normalize input; apply allow/block lists and review regexes; detect PII and off-platform handles; gate risky topics by inferred intent: - sale/facilitation/instruction; and run hate-speech: - heuristics violence verbs, dehumanization, slurs toward protected classes. Page-level cues e.g., password forms with brand mentions, magnet links, flag phishing/illicit risk. It aggregates up to 20k characters and escalate with explicit reasons. For images, NSFWJS thresholds and zero-shot CLIP scores drive the strictest outcome. All decisions are logged to MongoDB; APIs expose warning/stop headers; middleware enforces; policies hot-reload.

5 Testing

The system verifies real behavior through automated and hands-on checks. In total, 120 automated tests ran with 0 failures, and end-to-end runs plus manual passes confirmed that sign-up/login, template search & selection, drag-and-drop editing, preview, and publish/unpublished work as expected across Chrome/Firefox/Edge and desktop/tablet/mobile views. Issues found during early runs were fixed and re-verified, and the current build is stable and ready for submission.

5.1 Overview of Test Strategy

In this project, a layered test strategy is used for fast unit checks, targeted integration, strict API contract tests, and realistic end-to-end journeys, so that defects are caught early and real user workflows (build, preview, publish) are verified before release.

- **API:** Validate endpoint contracts, status codes, and payload schemas; cover auth/permissions, validation, and error paths using Express test harness with a test DB.
- **Unit:** Isolated checks of React components, hooks, utilities, and server modules using Vitest and Testing Library; dependencies mocked to hit edge cases quickly.
- **Integration:** Verify module interactions and client-server flows with real data wiring works.
- **End-to-End:** Playwright drives key user journeys, asserting visible outcomes and saving screenshots.

5.2 Sample Test Result

Base: http://localhost:8000/api

Key:

- M = Method
- H = Headers: A = Authorization: Bearer <token>, J = Content-Type: application/json, F = multipart/form-data (Postman handles boundary; don't set manually)
- Body keys: fn fullName, em email, pw password, sd startDate, ed endDate, snapshot minimal draft JSON
- Exp (expected): token(JWT), dup(duplicate), auth(unauthorized), invalid(validation), items/domain/website/meta/asset(presence), tooLarge, type(415), image(binary), published/unpub, deleted

Table 5-1: Test Table of API Testing

ID	Endpoint	M	H	Body	Exp
API-001	/register	POST	J	{fn,em,pw}	201 token
API-002	/register	POST	J	(same as 001)	409 dup
API-003	/login	POST	J	{em,pw}	200 token
API-004	/login	POST	J	{em,wrongPw}	401 auth
API-005	/domains	POST	A+J	{domain}	201 domain
API-006	/domains	POST	A+J	{domain}	409 dup
API-007	/profile-image/me	GET	A	-	404 none
API-008	/profile-image	POST	A+F	file	201 meta
API-009	/profile-image	PUT	A+F	file	200 meta
API-010	/subscription/me	GET	A	-	200 Limited
API-011	/subscription	POST	A+J	{sub:true,sd,ed(future)}	200 Standard
API-012	/subscription	POST	A+J	{sub:true,sd,ed(past)}	200 Expired
API-013	/websites	POST	A+J	{domain}	201 website
API-014	/websites/:id	GET	A	-	200 website
API-015	/websites/:id/draft	PUT	A+J	snapshot	200 saved
API-016	/websites/public/ :domain	GET	-	-	200 published
API-017	/websites/:id/publish	POST	A	-	200 published
API-018	/websites/:id/assets	POST	A+F	file	201 asset+url
API-019	/websites/:id/assets	GET	A	-	200 assets
API-021	/websites/:id/assets/ :assetId/raw	GET	-	-	200 bytes
API-022	/websites/upload	POST	A+F	domain,file	201 asset
API-023	/websites/:id/ unpublish	POST	A	-	200 unpublish

Table 5-2: Test Table of Unit Testing

Test ID	Function	Input/Action	Expected	Actual	Status
UT-001	RegisterModal submit success	Fill valid form submit	Close; onClose+onLogin	As expected	Pass
UT-002	AIRecommendTemplate flow	Q&A call recommender	Card + preview shown	As expected	Pass
UT-003	RegisterModal pw mismatch	Enter mismatched pw	Toast error	As expected	Pass
UT-004	RegisterModal loading guard	Submit while loading	Button disabled	As expected	Pass
UT-005	RegisterModal submit fail	Mock API error	Toast error; no close	As expected	Pass
UT-006	CategoryModal search	Type search term	Filtered list	As expected	Pass
UT-007	Website draft nav	Click navbar hashes	Page changes	As expected	Pass
UT-008	Preview draft nav	Click navbar hashes	Page changes	As expected	Pass
UT-009	LoginModal submit (session)	remember=false submit	Save to session; toast	As expected	Pass
UT-010	RegisterModal toggle pw	Toggle both fields	Visibility toggles	As expected	Pass
UT-011	Home CTA opens Register	Click Start Now	Register modal opens	As expected	Pass
UT-012	Dashboard render + row	Mount dashboard	Header/card/row visible	As expected	Pass
UT-013	CreateDomain invalid chars	Enter bad domain	Toast error	As expected	Pass
UT-014	Builder footer moderation	Save blocked footer	Warning modal	As expected	Pass
UT-015	Templates.plan baseline	Render page	6 previews + heading	As expected	Pass
UT-016	Services page content	Render page	Title + 3 highlights	As expected	Pass
UT-017	NavBar desktop actions	Click nav/buttons	Links + modals open	As expected	Pass

Table 5-3: Test Table of Integration Testing

Test ID	Function	Input/Action	Expected	Actual	Status
IT-001	App router: Home	Navigate	Home visible; BotTalk+toast shown	As expected	Pass
IT-002	App router: Pricing	Navigate/pricing	Pricing page renders	As expected	Pass
IT-003	App router: Template	Navigate /template/portfolio1	Template page renders	As expected	Pass
IT-004	App router: Public viewer	Navigate /Website/portfolio.com	Public viewer renders	As expected	Pass
IT-005	App router: Fallback	Navigate unknown/*	Falls back to Home	As expected	Pass
IT-006	Builder: autosave on theme	Change theme; wait	Draft autosaved	As expected	Pass
IT-007	Builder: footer (Standard)	Edit footer (Standard)	Edit allowed; saved	As expected	Pass
IT-008	Builder: footer (Limited)	Limited plan; attempt edit	Edit hidden; default saved	As expected	Pass
IT-009	Builder: preview share	Open Preview; broadcast	Token posted to preview	As expected	Pass
IT-010	NavBar-Register-Login	Open Register; switch	Login modal opens	As expected	Pass
IT-011	Register success flow	Submit valid form	API ok; Login opens after delay	As expected	Pass
IT-012	Register validations	Mismatch pw; Terms off	Toast; submit blocked	As expected	Pass
IT-013	Login success flow	Submit (remember on/off)	Token stored; go /dashboard	As expected	Pass
IT-014	Recommender flow	Q&A; call recommender	Recommended card shown	As expected	Pass

IT-015	Templates: smoke	Load default	All templates render	As expected	Pass
IT-016	Templates: search	Type query	List filtered	As expected	Pass
IT-017	Templates: links	Inspect card links	Correct preview routes	As expected	Pass
IT-018	Templates: with/without domain	Toggle ?domain	Renders in both cases	As expected	Pass

Table 5-4: Test Table of End-to-End Testing

ID	**Suite**	**Function**	**Time (s)**	**Status**
E2E-001	bottalk.spec.js:5	BotTalk: open-sendbullets-persist/min-restore	8.2	Pass
E2E-002	bottalk.spec.js:62	BotTalk: friendly error on server fail	4.5	Pass
E2E-003	builder.spec.js:42	Builder: template-device switch-edit-Preview	6.5	Pass
E2E-004	builder.spec.js:95	Builder: Publish-Unpublish toggles state	6.9	Pass
E2E-005	dashboard.spec.js:48	Dashboard: auth guard-create/preview/delete-logout	10.8	Pass
E2E-006	preview.spec.js:8	Preview: DRAFT only + draft footer	3.4	Pass
E2E-007	preview.spec.js:98	Preview: missing ?domain message	2.8	Pass
E2E-008	public-pages.spec.js:17	Public: Home nav + CTA opens Register	4.2	Pass
E2E-009	public-pages.spec.js:41	Public: Pricing CTAs open Register	3.4	Pass
E2E-010	public-pages.spec.js:55	Public: Templates search + preview new tab	11.4	Pass
E2E-011	public-pages.spec.js:78	Public: Services headline + device image	3.2	Pass
E2E-012	public-pages.spec.js:84	Public: Contact submit success	3.0	Pass
E2E-013	public-pages.spec.js:100	Public: Footer newsletter email validation	5.0	Pass
E2E-014	recommend.spec.js:55	Recommend: wizard-recommend-Preview&Apply	6.0	Pass
E2E-015	smoke.spec.js:3	Smoke: Home renders	3.8	Pass

E2E-016	templates.spec.js:27	Templates: title/plan/search/cards	11.8	Pass
E2E-017	templates.spec.js:48	Templates: Preview opens new tab	12.5	Pass
E2E-018	templates.spec.js:62	Templates: Select-builder (no domain)	10.9	Pass
E2E-019	templates.spec.js:79	Templates: Select-builder (with domain)	10.7	Pass
E2E-020	website-public.spec.js:12	PublicSite: renders, nav, slugs, footer meta	3.2	Pass
E2E-021	website-public.spec.js:93	PublicSite: unknown slug-base	2.6	Pass
E2E-022	website-public.spec.js:126	PublicSite: “Under Maintenance” on 404	2.6	Pass

Testing Evidences

API endpoints were validated in Postman server and screenshots are included. Unit, Integration, and E2E tests ran from dedicated files using npm run test:unit, npm run test:int, and npm run test:e2e:headed; screenshots are attached for Unit and Integration. E2E results were reviewed via Playwright’s HTML report (npx playwright show-report), which doesn’t produce static screenshots, so pass/fail summaries and logs serve as evidence.

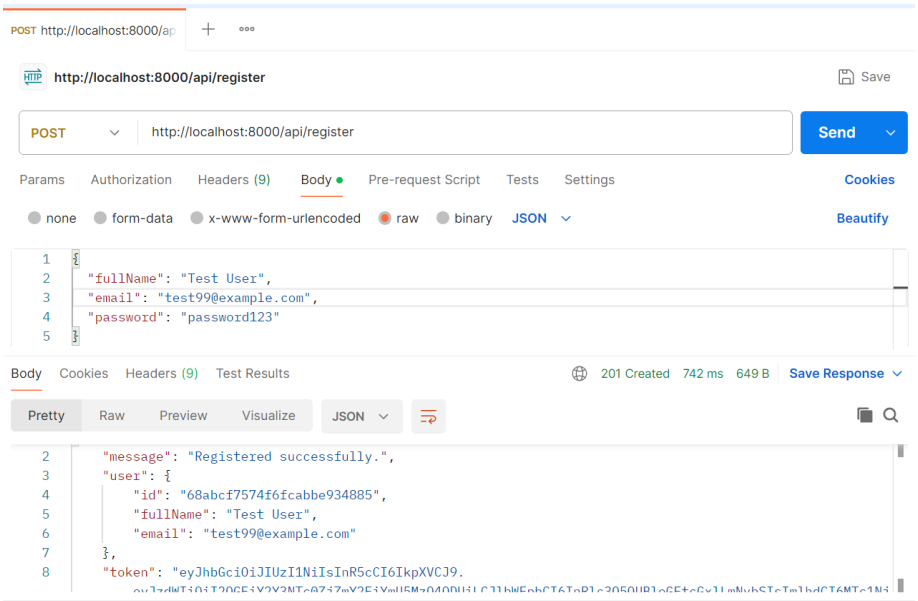


Figure 5.1: Test Evidence: Test ID : API-001

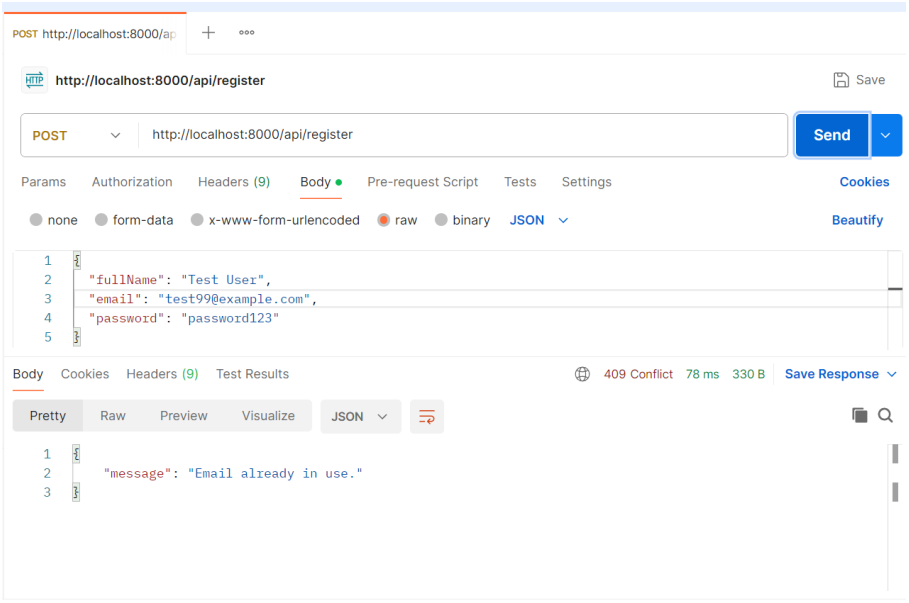


Figure 5.2: Test Evidence: Test ID : API-002

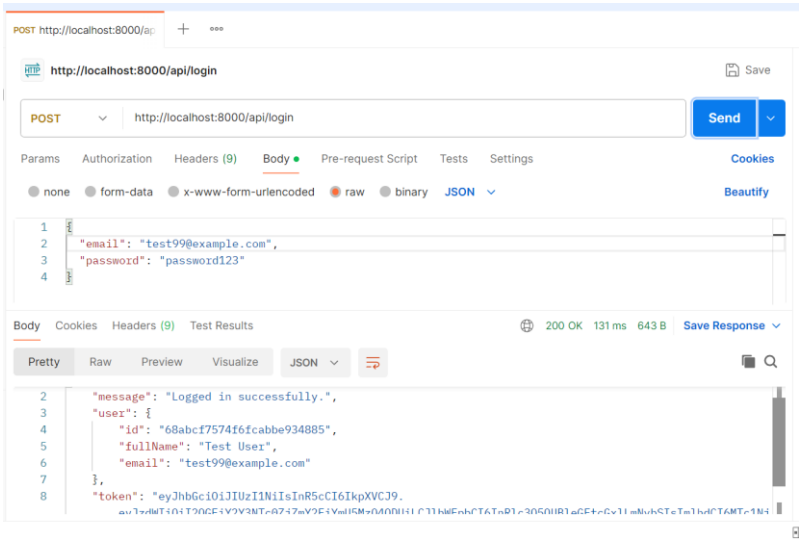


Figure 5.3: Test Evidence: Test ID : API-003

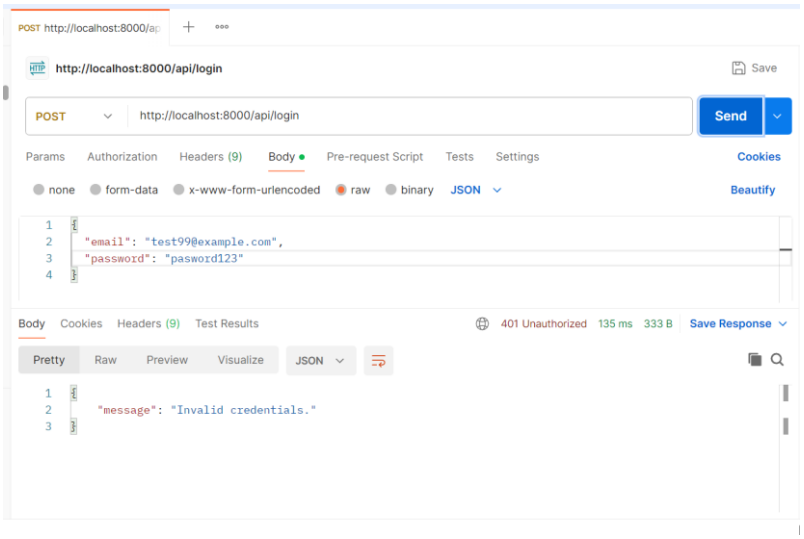


Figure 5.4: Test Evidence: Test ID : API-004

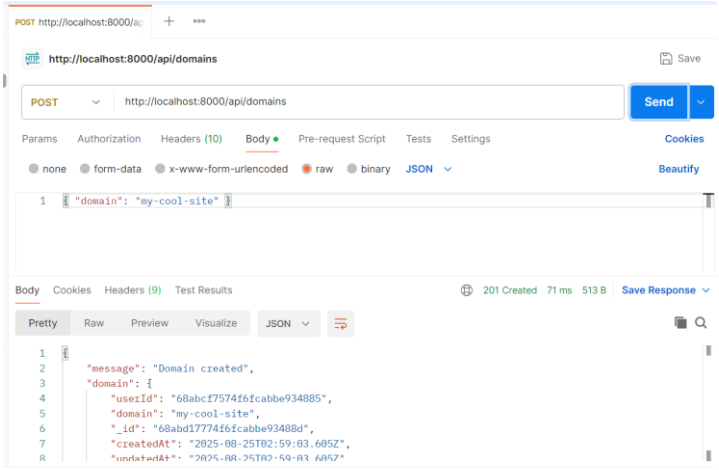


Figure 5.5: Test Evidence: Test ID : API-005

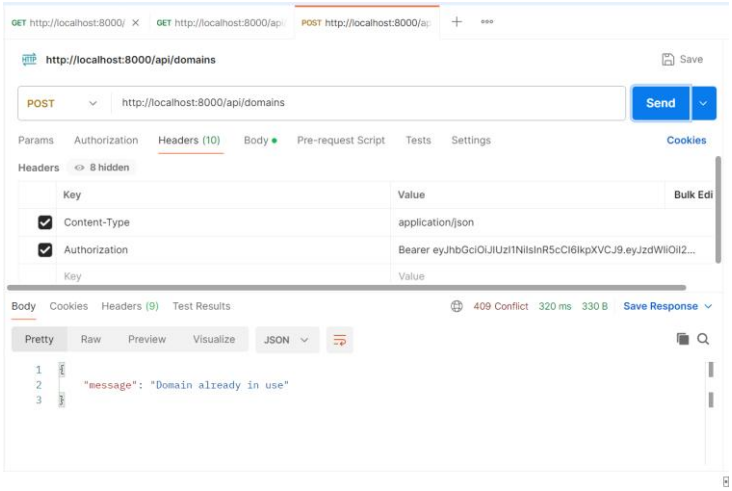


Figure 5.6: Test Evidence: Test ID: API-006

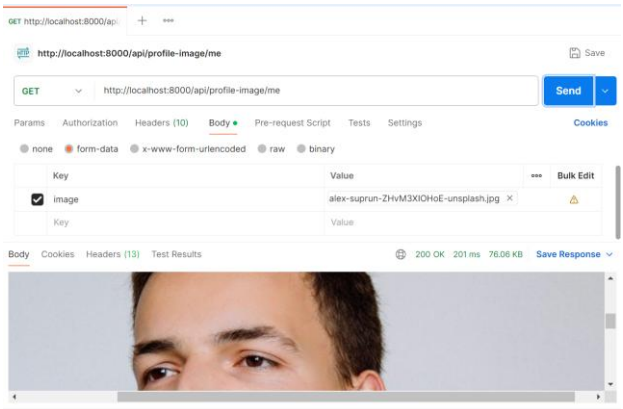


Figure 5.7: Test Evidence - Test ID - API-007

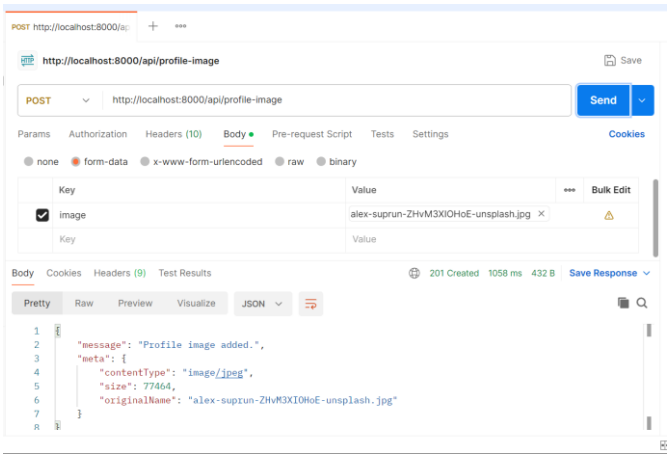


Figure 5.8: Test Evidence; Test ID - API-008

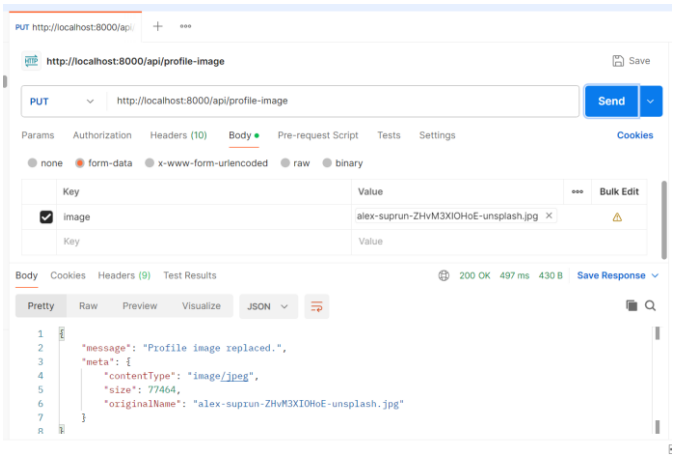


Figure 5.9: Test Evidence - Test ID - API-009

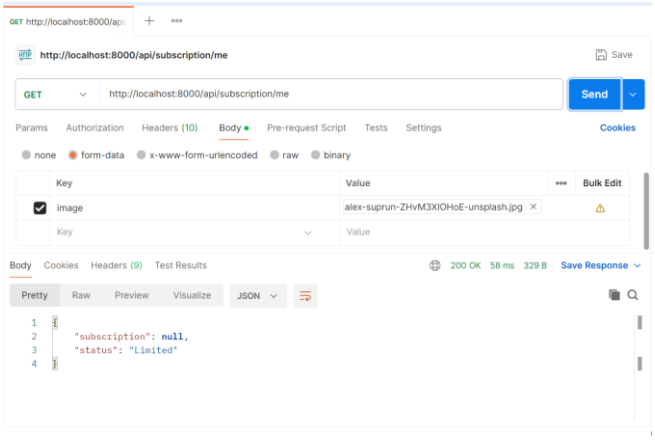


Figure 5.10: Test Evidence - Test Id: API-010

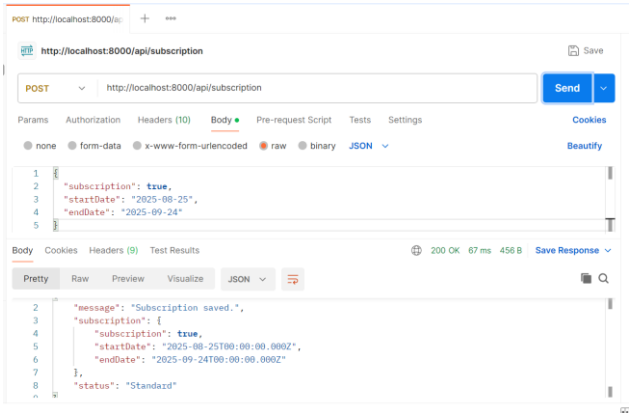


Figure 5.11: Test Evidence - TestID - API-011

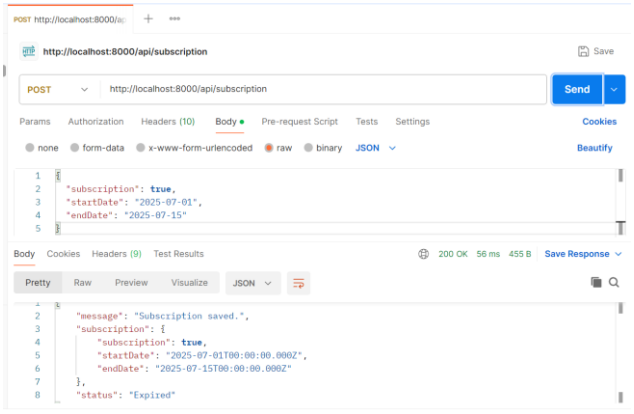


Figure 5.12: Test Evidence - Test ID: API-012

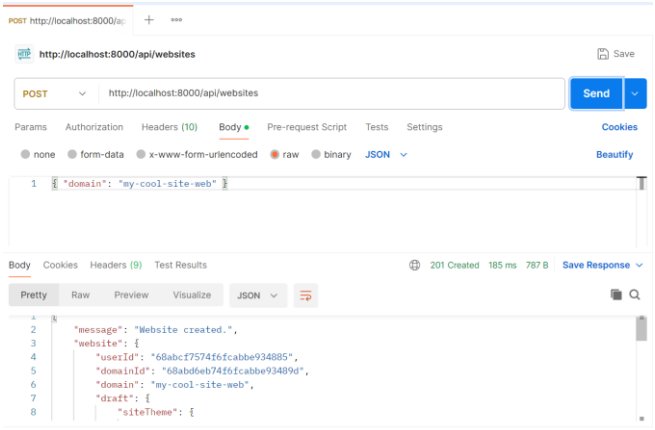


Figure 5.13: Test Evidence - Test ID: API-013

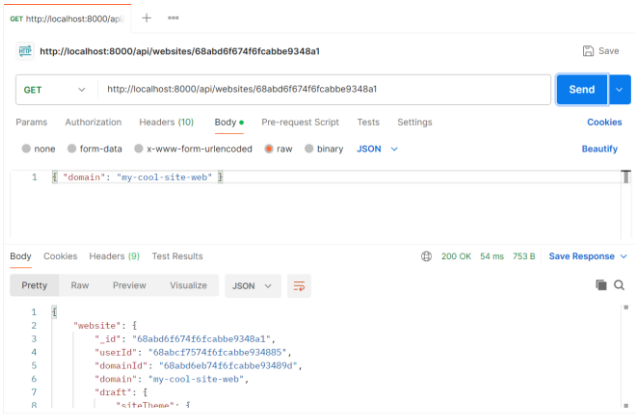


Figure 5.14: Test Evidence - Test ID: API-014

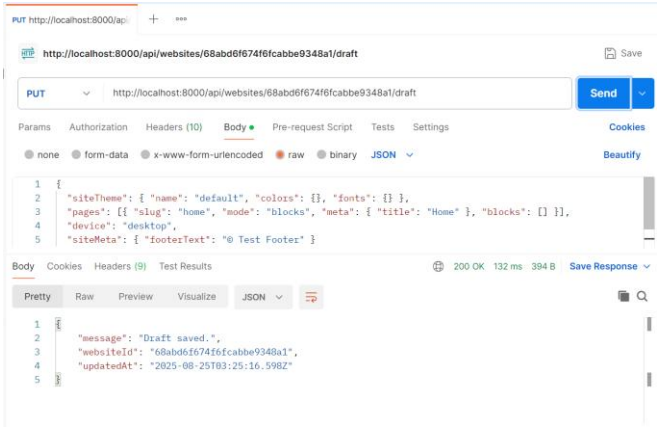


Figure 5.15: Test Evidence - Test ID: API-015

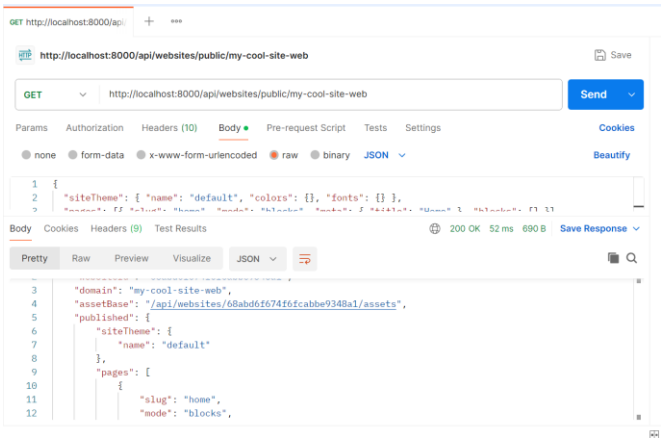


Figure 5.16: Test Evidence - Test ID: API-016

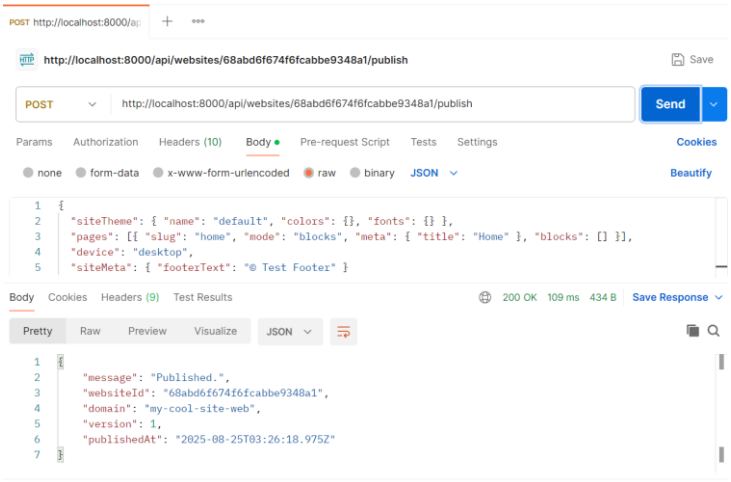


Figure 5.17: Test Evidence - Test ID: API-017

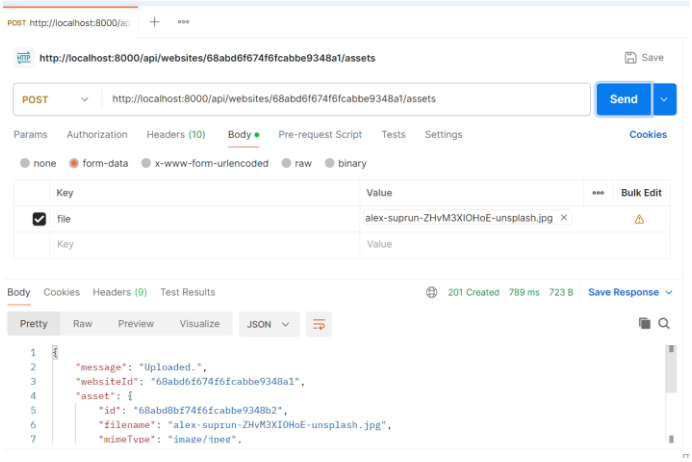


Figure 5.18: Test Evidence - Test Id: API-018

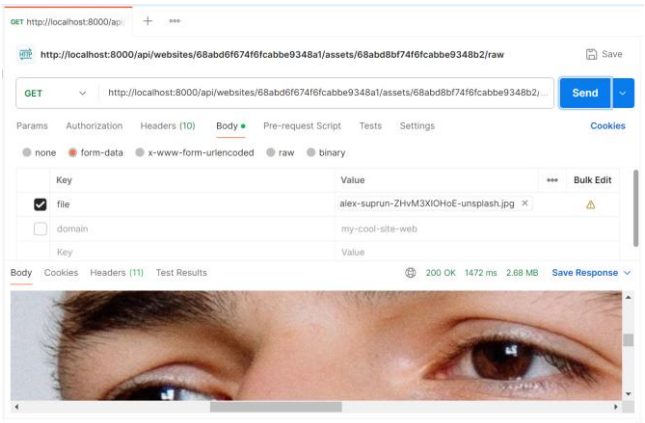


Figure 5.19: Test Evidence - Test ID: API-020

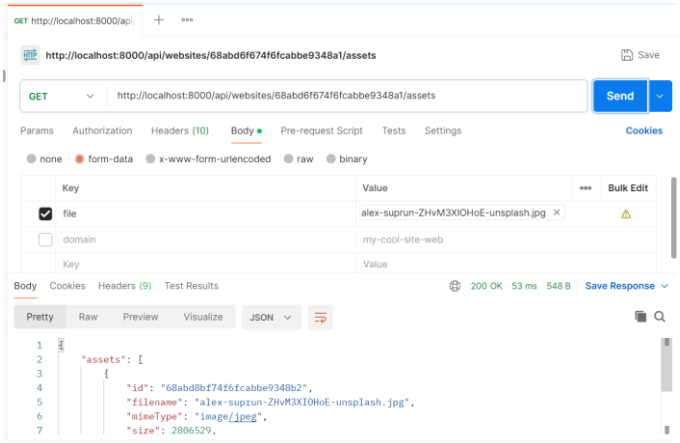


Figure 5.20: Test Evidence - Test ID: API-021

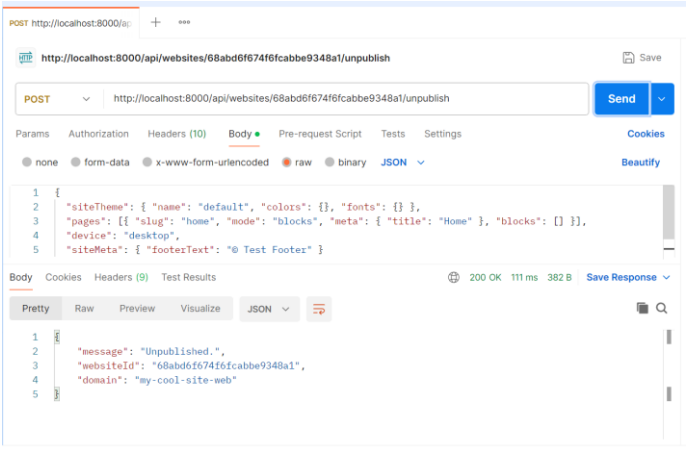


Figure 5.21: Test Evidence - Test ID: API-023

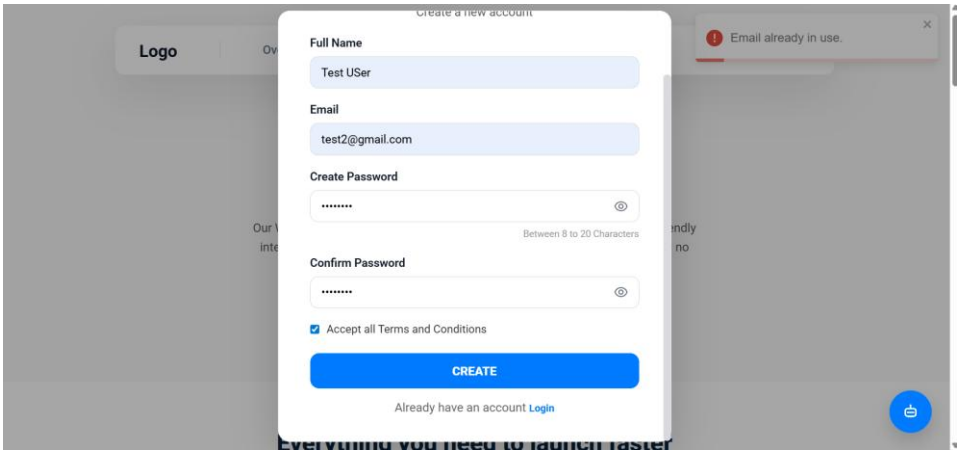


Figure 5.22: Test Evidence - UT-001

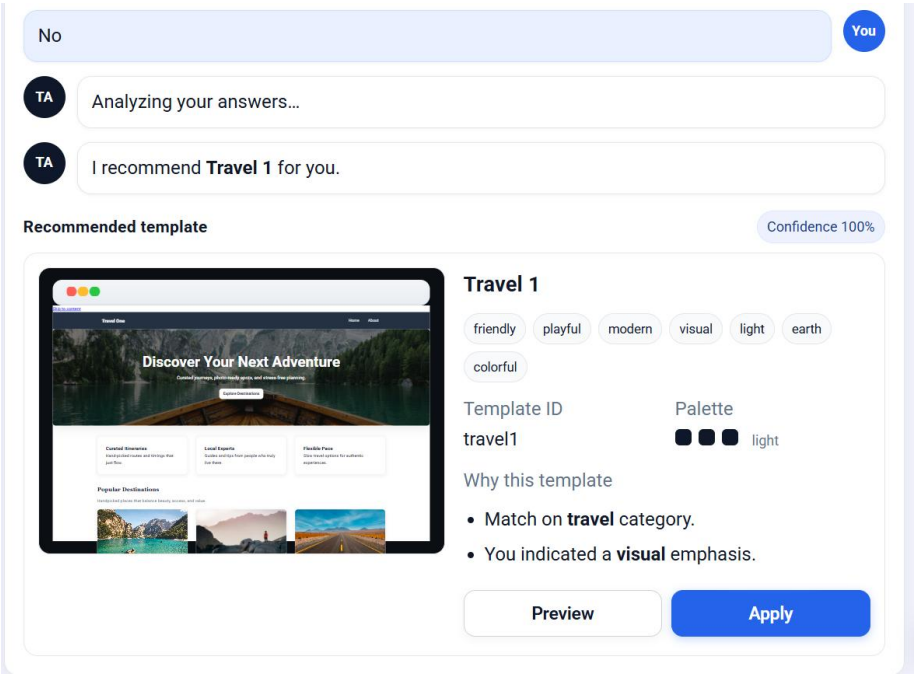


Figure 5.23: Test Evidence - UT-002

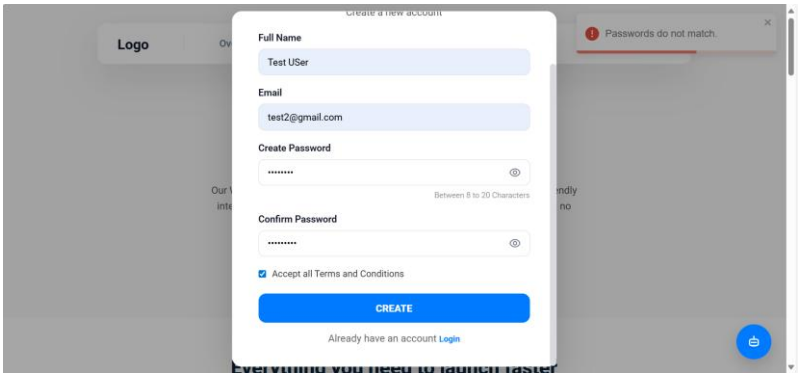


Figure 5.24:Test Evidence: UT-003

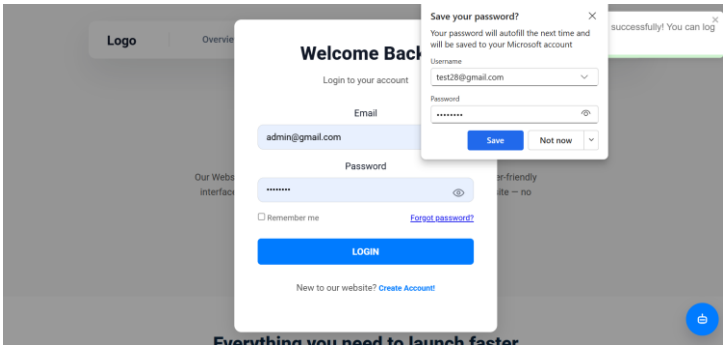


Figure 5.25: Test Evidence: UT-004

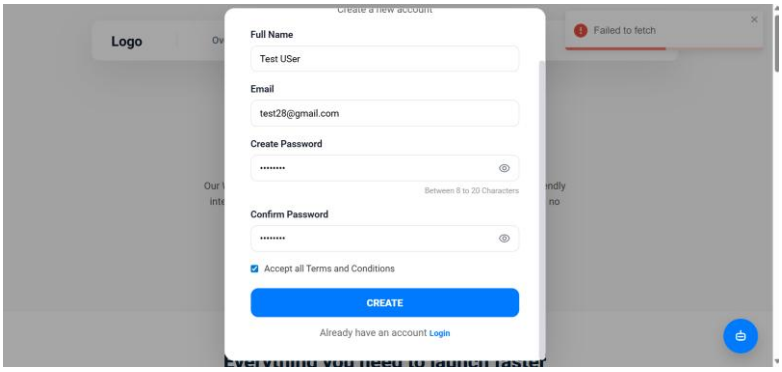


Figure 5.26: Test Evidence - UT-005

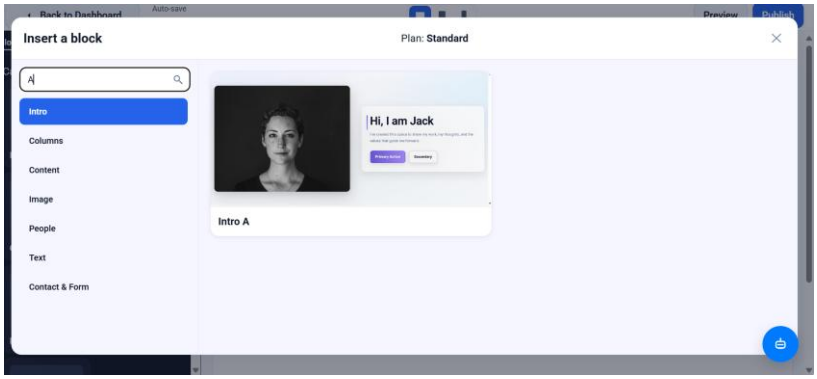


Figure 5.27: Test Evidence: UT-006

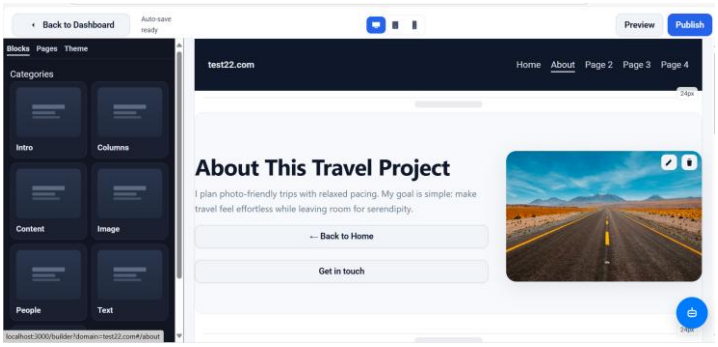


Figure 5.28: Test Evidence: UT-007

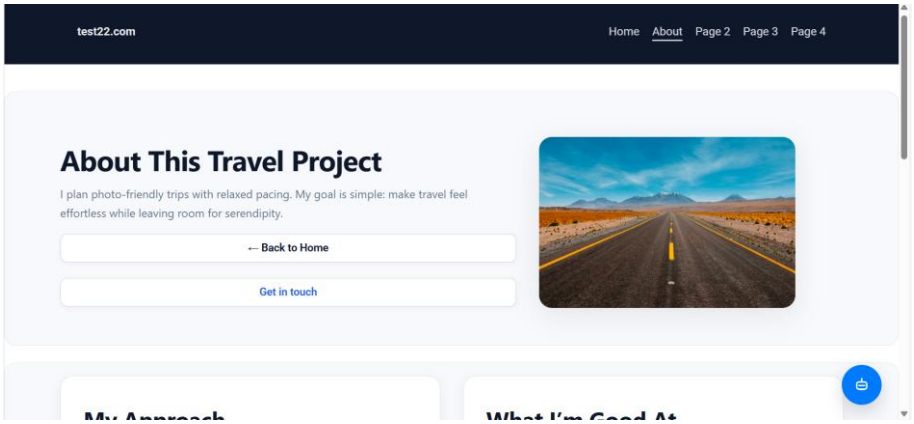


Figure 5.29: Test Evidence: UT-008

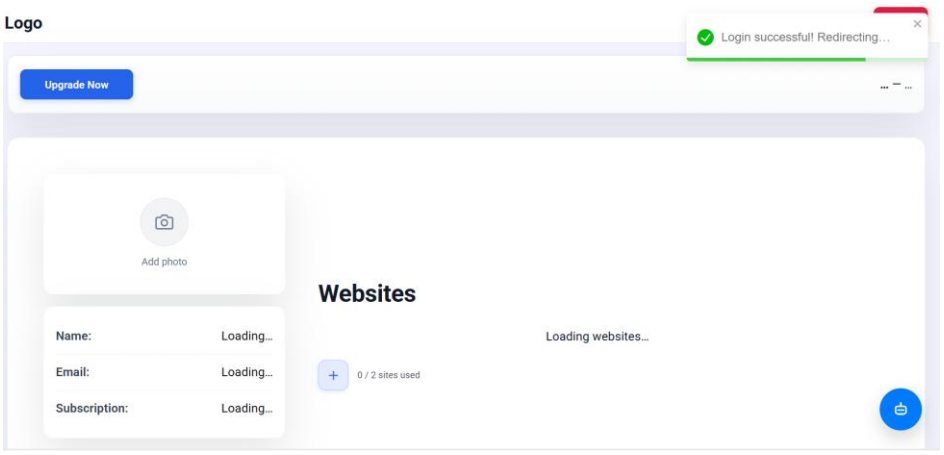


Figure 5.30: Test Evidence: UT-009

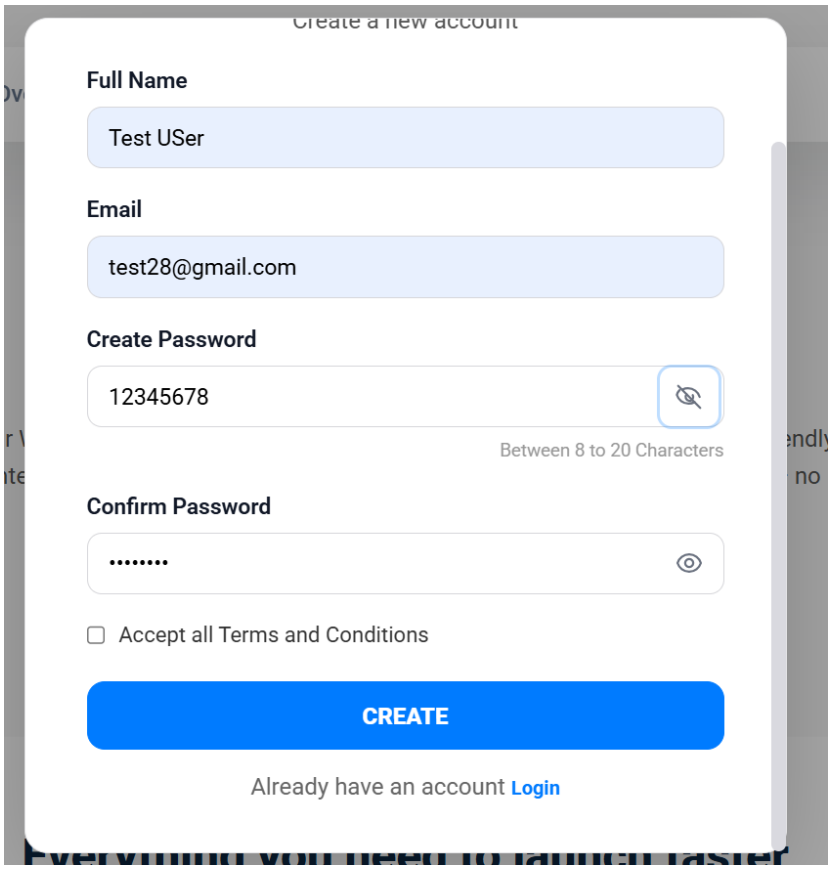


Figure 5.31: Test Evidence: UT-010

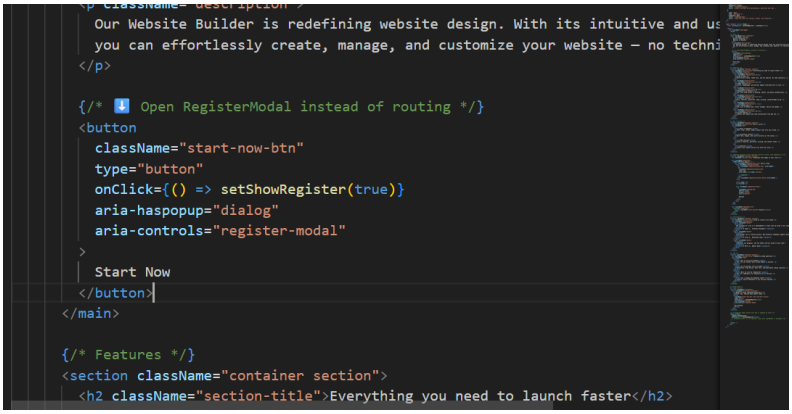


Figure 5.32: Test Evidence: UT-011

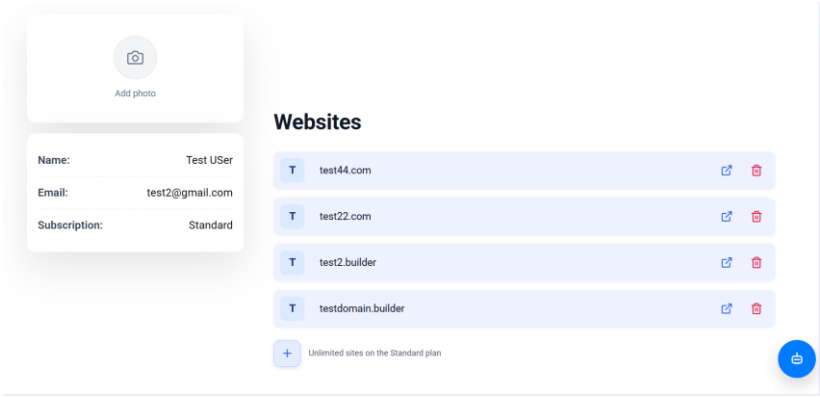


Figure 5.33: Test Evidence: UT-012

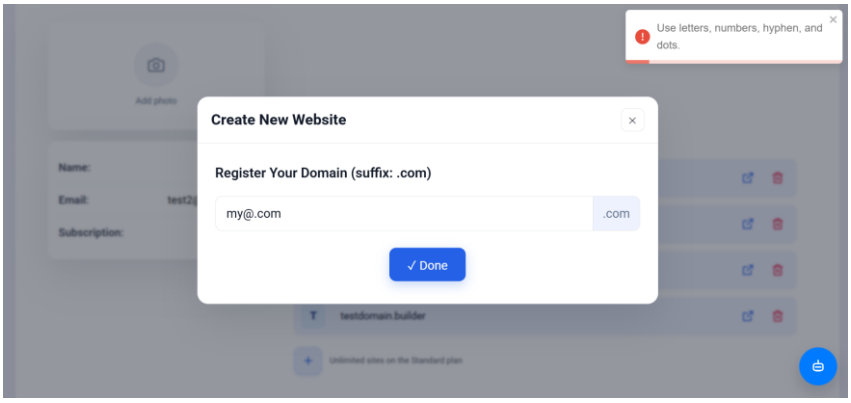


Figure 5.34: Test Evidence: UT-013

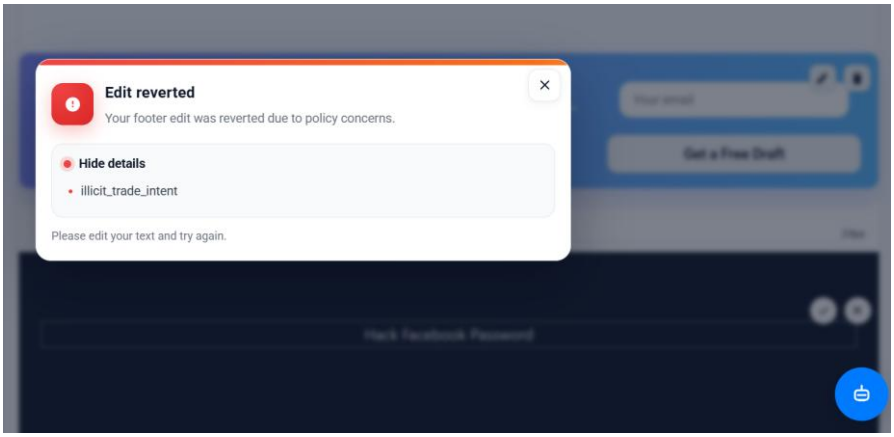


Figure 5.35: Test Evidence: UT-014

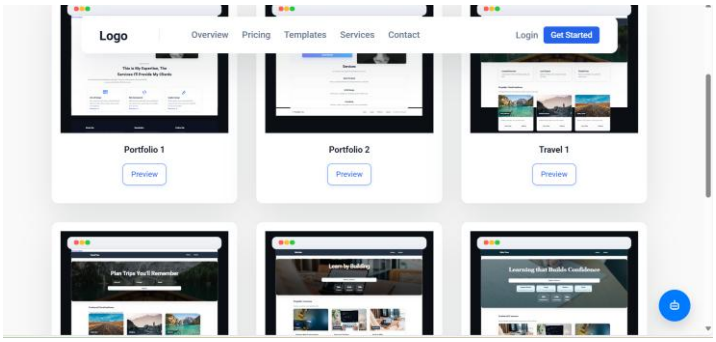


Figure 5.36: Test Evidence - UT-015

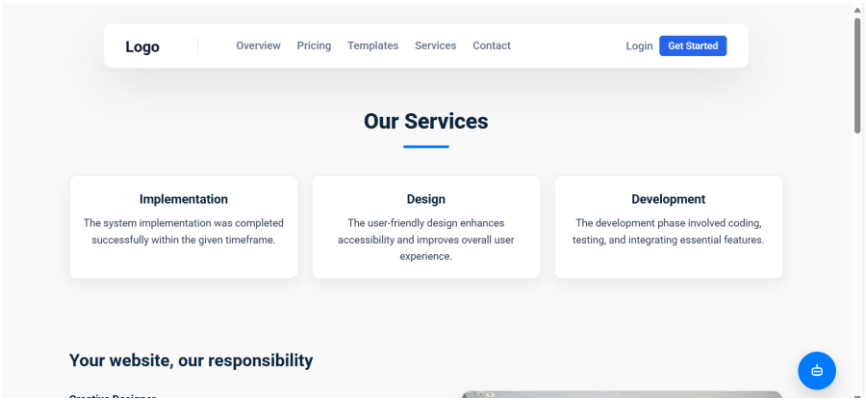


Figure 5.37: Test Evidence: UT-016

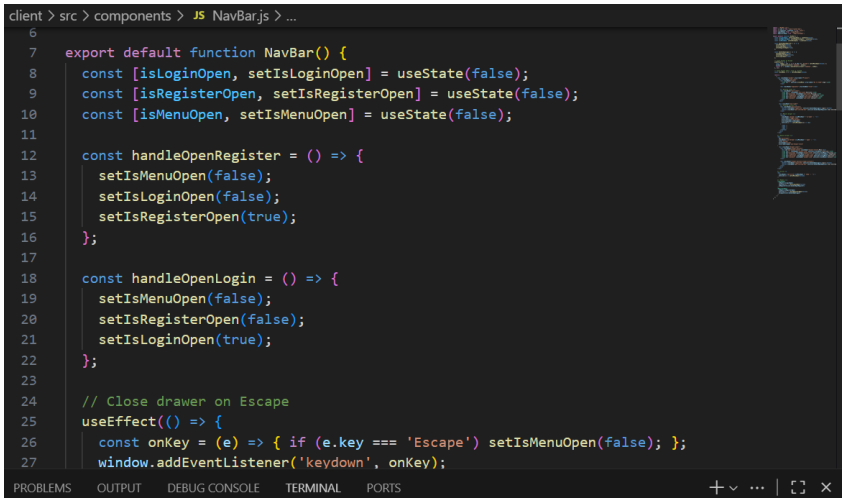


Figure 5.38: Test Evidence: UT-017

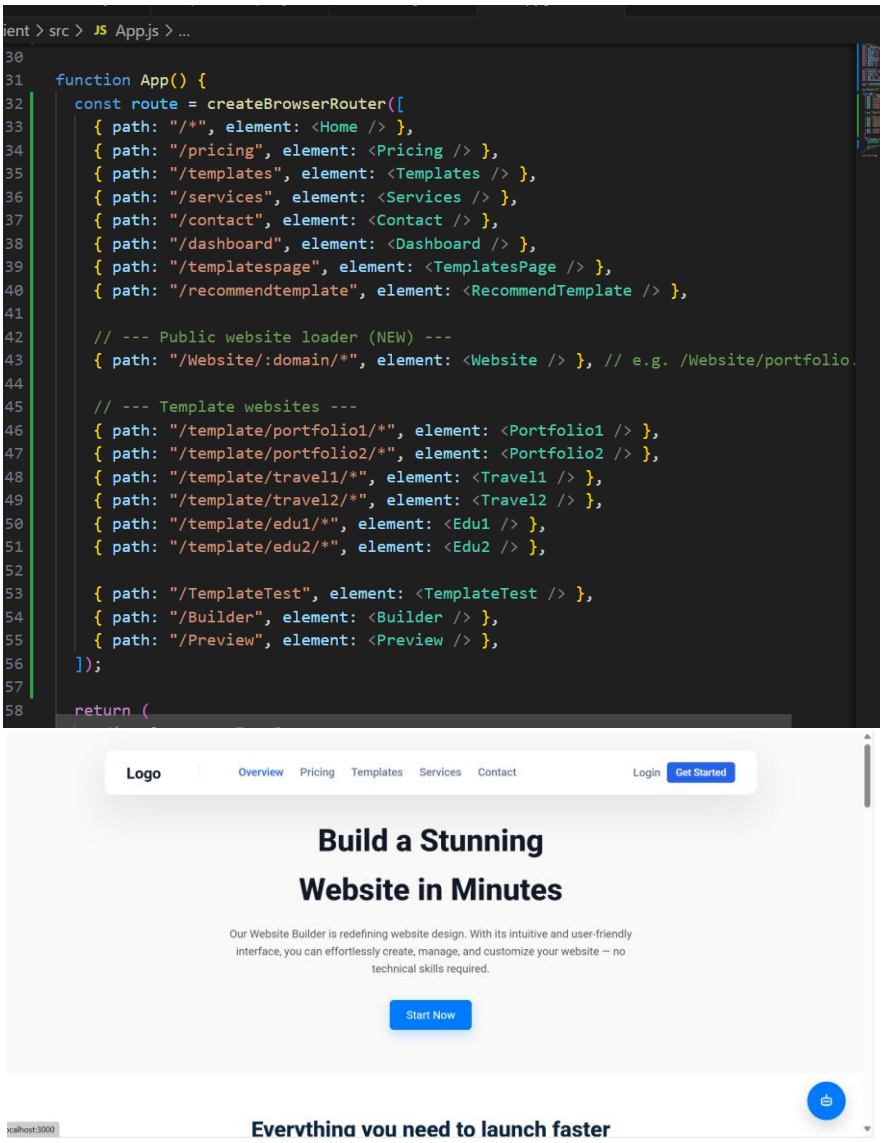


Figure 5.39: Test Evidence: IT-001

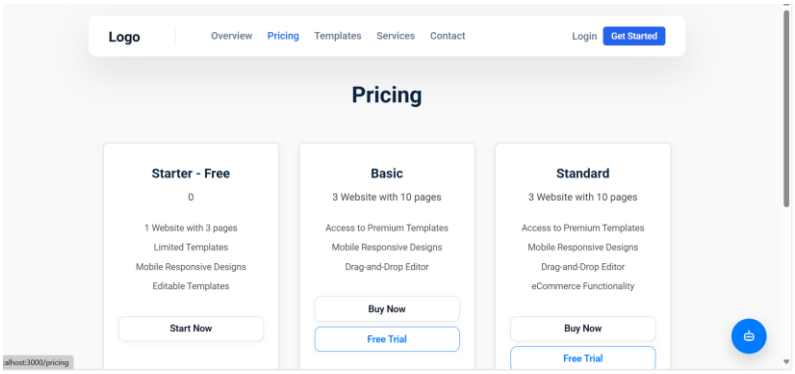


Figure 5.40: Test Evidence: IT-002

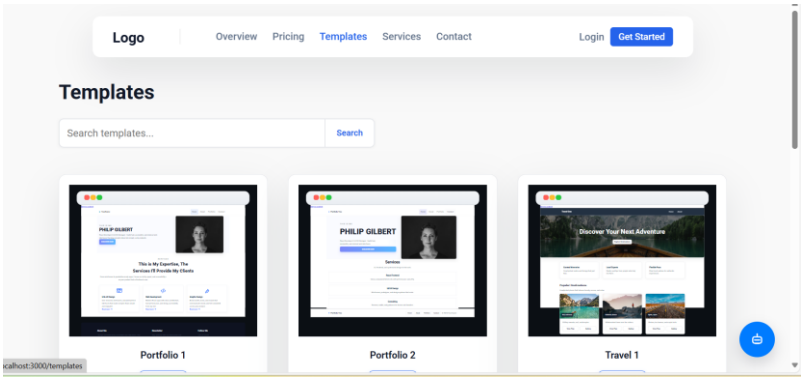


Figure 5.41: Test Evidence: IT-003

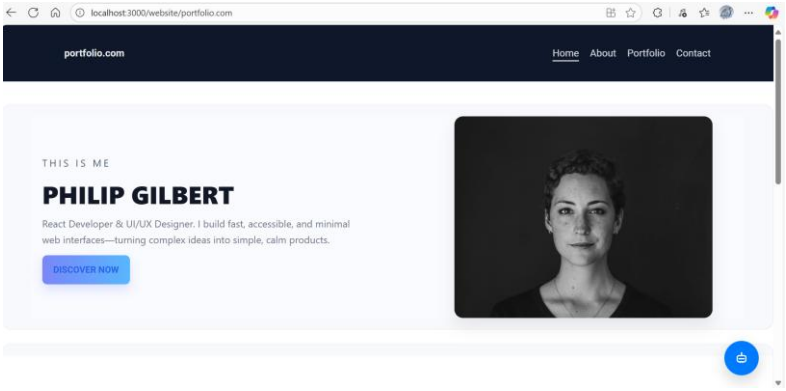


Figure 5.42: Test Evidence - IT-004

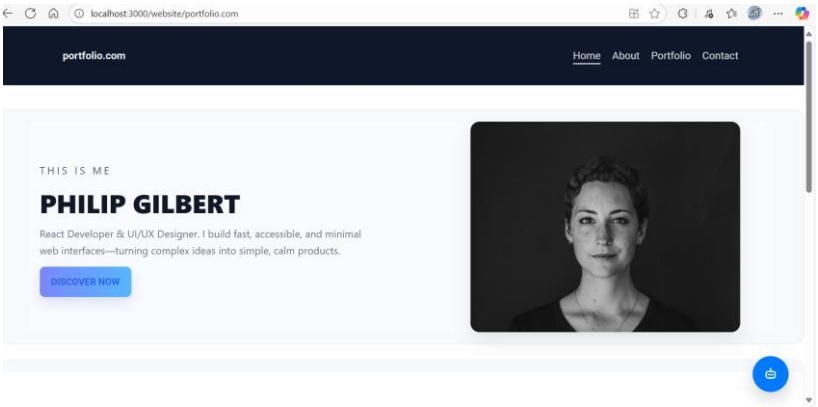


Figure 5.43: Test Evidence: IT-005

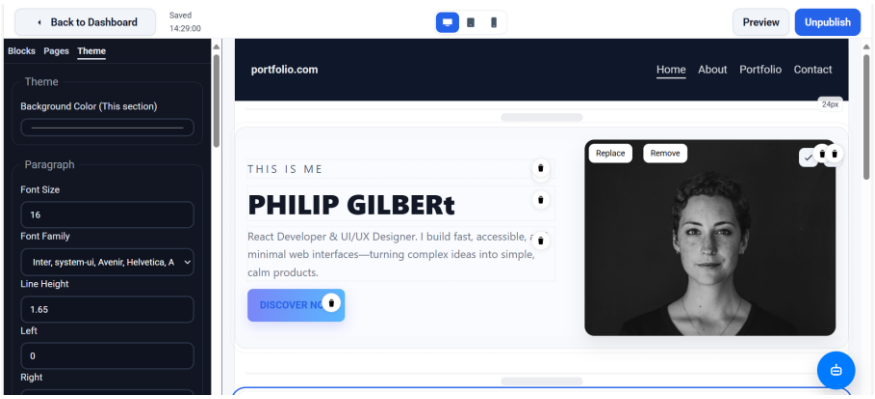


Figure 5.44: Test Evidence: IT-006

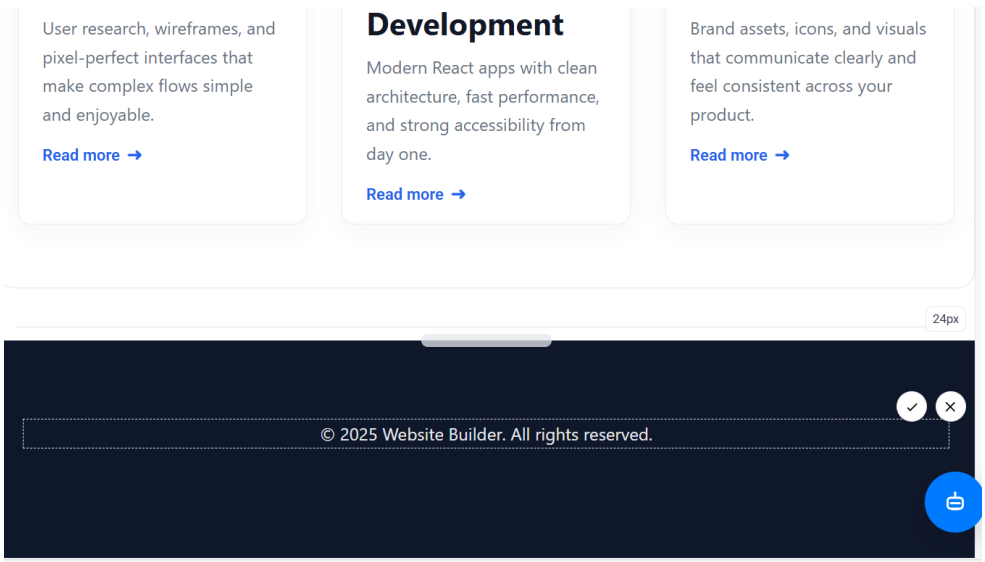


Figure 5.45: Test Evidence: UT-007

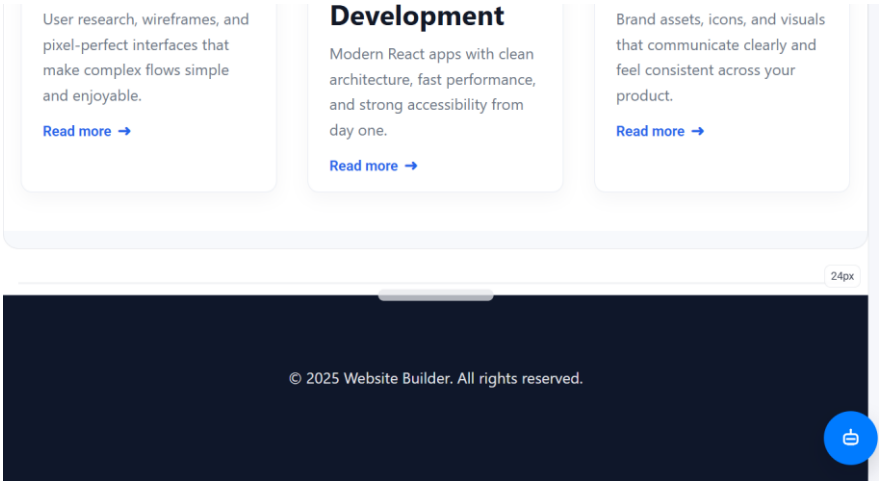


Figure 5.46: Test Evidence: IT:008



```
function openPreview() {
  // keep existing query params (?domain=..., ?template=..., etc.)
  const currentParams = new URLSearchParams(window.location.search);

  // grab token from storage
  const token = localStorage.getItem("token") || sessionStorage.getItem("token");

  // add the token into the preview URL (primary path)
  if (token) currentParams.set("auth", token);

  const url = new URL(`/preview?${currentParams.toString()}`, window.location.origir

  // open in a new tab; keep 'noopener' for safety
  window.open(url, "_blank", "noopener");

  // also broadcast the token (secondary path / redundancy)
  const bc = new BroadcastChannel("builder-preview-auth");

  const send = () => {
    if (token) bc.postMessage({ type: "token", token });
```

Figure 5.47: Test Evidence: IT-009

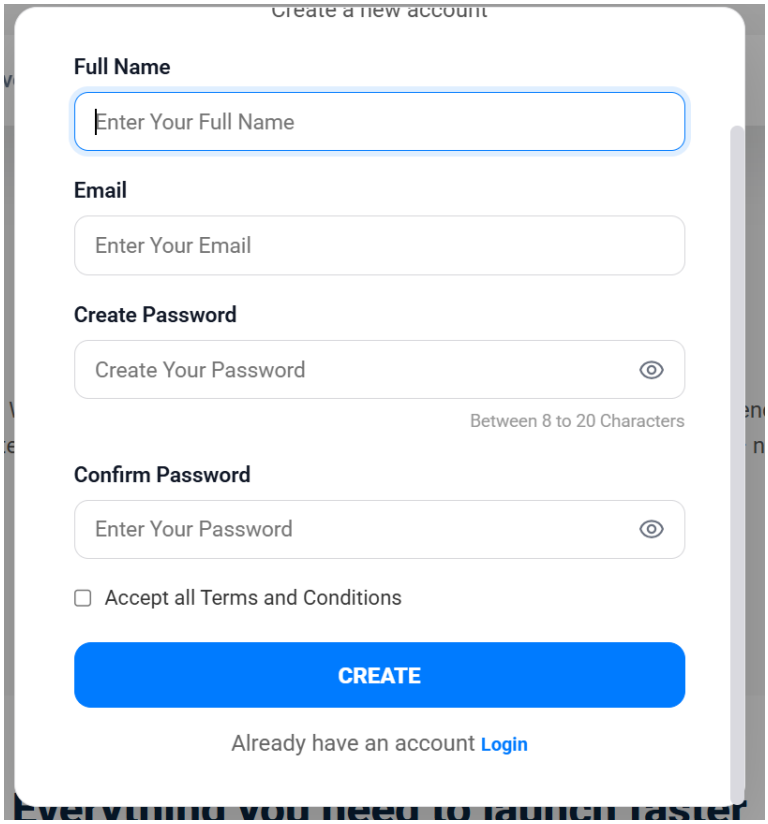


Figure 5.48: Test Evidence: IT-010

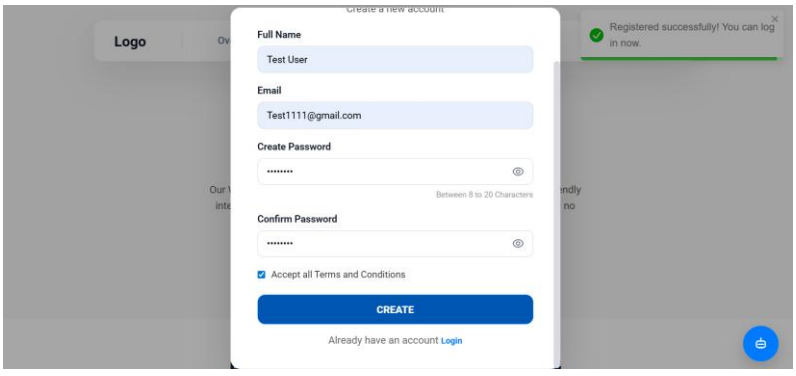


Figure 5.49: Test Evidence: IT-011

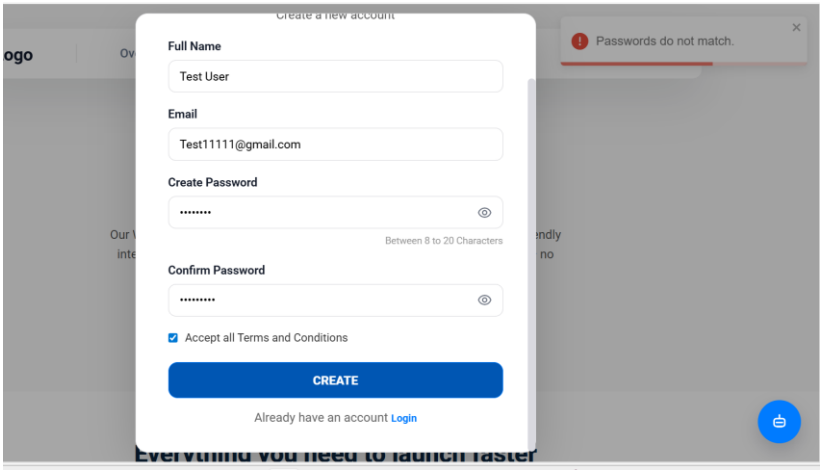


Figure 5.50: Test Evidence: IT-012

```
export default function LoginModal({ isOpen, onClose, onSuccess, onRegisterClick }) {
  const handleSubmit = async (e) => {
      const data = await fetchJSON(`${API_BASE}/api/login`, {
        body: JSON.stringify({
          password,
        }),
      });

      const storage = remember ? localStorage : sessionStorage;
      storage.setItem('token', data.token);
      storage.setItem('user', JSON.stringify(data.user || {}));

      const other = remember ? sessionStorage : localStorage;
      other.removeItem('token');
      other.removeItem('user');

      onSuccess?.({ user: data.user, token: data.token });
      toast.success('Login successful! Redirecting…');

      setTimeout(() => {
        onClose?.();
        navigate('/dashboard');
      }, 800);
    } catch (e2) {
      toast.error(e2.message || 'Login failed');
    } finally {
      setLoading(false);
    }
  };
```

Figure 5.51: Test Evidence: IT-013

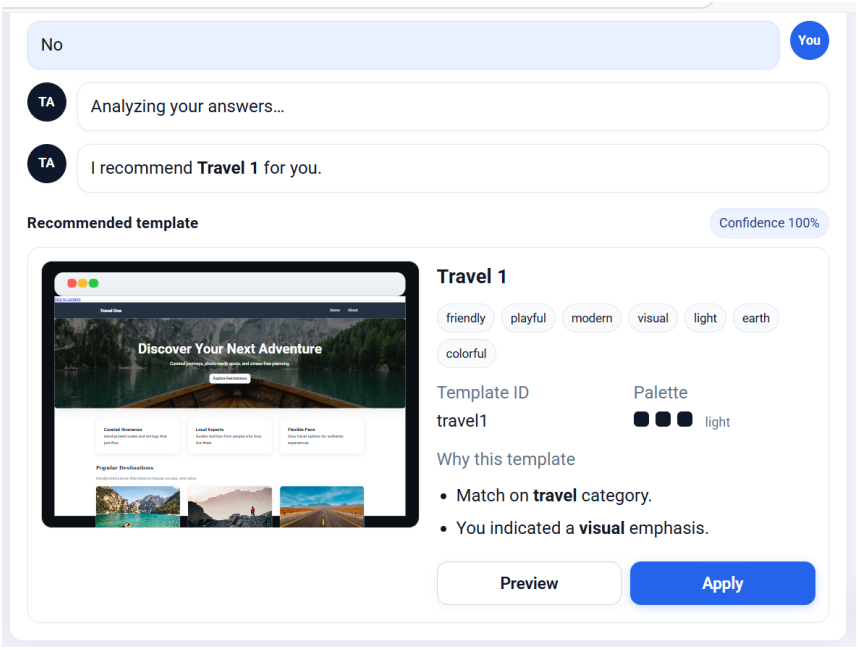


Figure 5.52: Test Evidence: IT-014

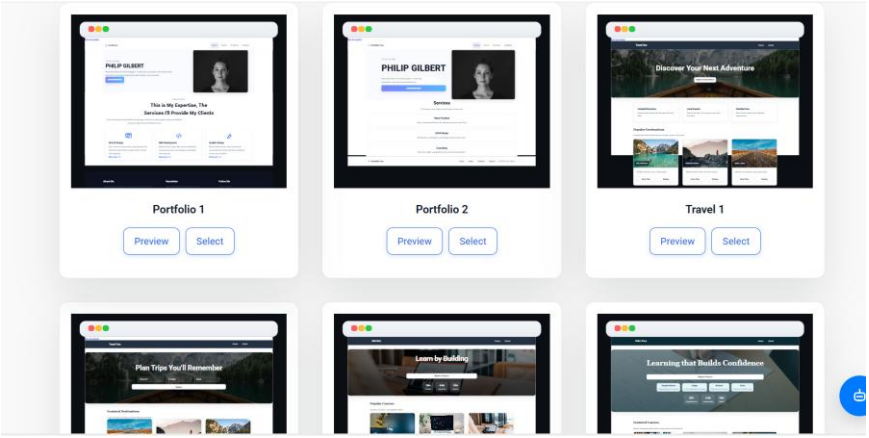


Figure 5.53: Test Evidence: IT-015

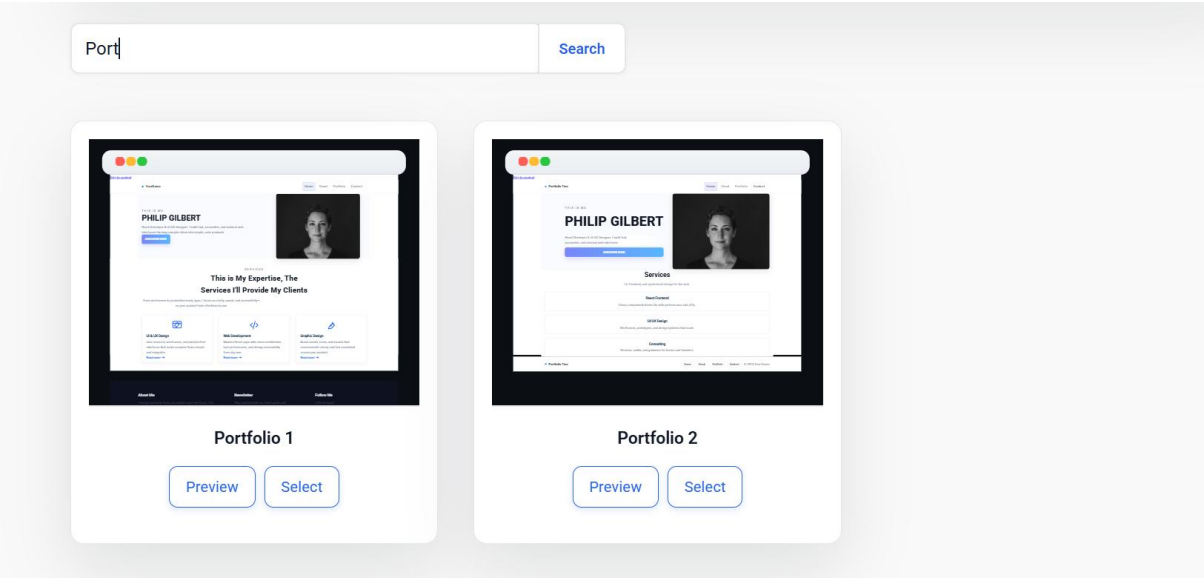


Figure 5.54: Test Evidence: IT-016

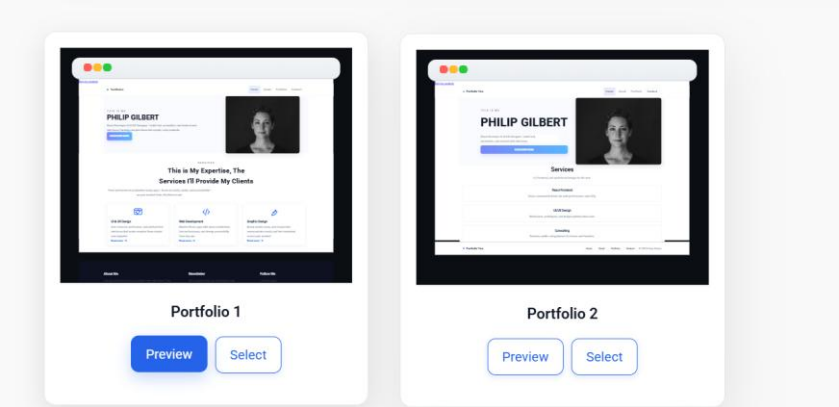


Figure 5.55: Test Evidence: IT-017

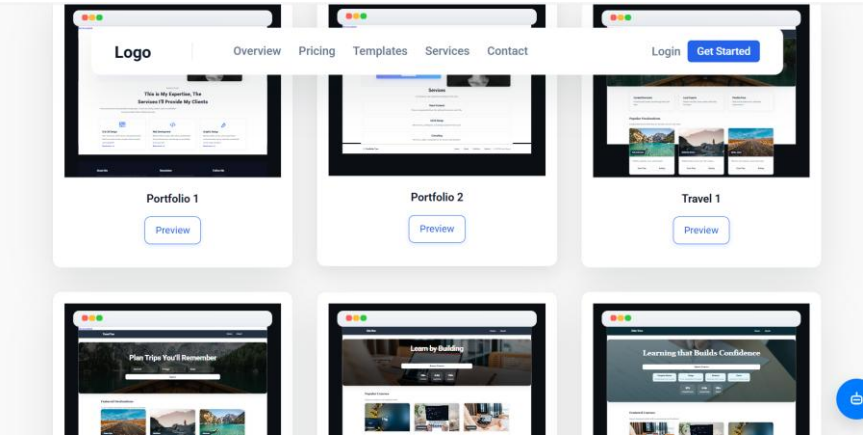


Figure 5.56: Test Evidence: IT-018

5.3 Test Evaluation

Testing covers API, unit, integration, and end-to-end layers. On 20 August 2025, the suite achieved a 100% pass rate (Unit: 120; Integration: 18; E2E: 22) in 86.7 seconds. Validated areas include authentication and dashboard (guarded routes, domain CRUD), builder workflows (template loading, device switching, edits, autosave, publish/unpublish), preview/public routing and metadata, template search/preview (new tab), and BotTalk resilience. Performance hotspots are new-tab flows and long UI sequences. Risks: window flakiness, debounce sensitivity, timing variance, and missing negative E2E. Recommendations: mock window.open; reduce debounce; disable animations; replace sleeps with state-based waits; fixture APIs; add negative E2E; enforce timing gates.

6 Conclusion

This project designed, implemented, and evaluated a no-code website builder for non-technical users. It uses an MVC architecture: React for a responsive editor/preview, Express for secure APIs/workflows, and MongoDB/Mongoose for flexible persistence, allowing independent evolution. Features include block-based drag-and-drop with auto-save and publish, a hybrid rules+TF-IDF recommender, a privacy-preserving RAG Chatbot, and text/image moderation. Validation used layered testing: automated tests passed; end-to-end checks confirmed sign-up-to-publish flows.

Technical Difficulty Encountered

The main technical challenges were reliability, safety, and scale in a no-code UX. Rich editor state had to be normalized and resized to fit document limits, with defensive upload validation. Assistance required rules-first recommendations and lexical RAG for deterministic answers; moderation combined lists, PII checks, NSFWJS and CLIP. Cross-browser/device stability and API fidelity demanded layered unit/integration/E2E tests to prevent regressions.

Future Scope

Future work: add learning-based re-ranking to the recommender; introduce team workflows (roles, reviews); support a plugins/blocks marketplace; strengthen governance with audit logs, rate limits, quotas; automate a11y/performance via WCAG checks and CI budgets; improve portability with exportable templates and self-hosted profiles; run A/B tests to optimize onboarding, fit, publish success.

Bibliography

Badman, A., & Kosinski, M. (2025, June 26). *What is a dataset?* Retrieved from IBM: https://www.ibm.com/think/topics/dataset

Bekmanova, G., Yergesh, B., Omarbekova, A., Orynbay, L., Bessembayeva, A., Kabdylova, D., . . . Sultan, B. (2024). Requirements for the Development of a Website. *IX th International Conference on Computer Science and Engineering*, 6.

Class Diagram. (n.d.). Retrieved from https://comp.ieu.edu.tr/: https://homes.izmirekonomi.edu.tr/culudagli/files/SE305/Labs/Week11/Class%20Diagrams%20Notes.pdf

Design. (2025, April 03). *Design*. Retrieved from Design: https://www.design.com/website-builder/

Dhoke, P., & Lokulwar, D. (2023). Evaluating the Impact of No-Code/Low-Code. *14th International Conference on Computing Communication and Networking Technologies (ICCCNT)*, 5.

Education and Skills Act 2008. (n.d.). Retrieved from legislation.gov.uk: https://www.legislation.gov.uk/ukpga/2008/25/contents

Federal Trade Commission. (2025, March 7). *Start with Security: A Guide for Business*. Retrieved from Federal Trade Commission Protecting America's Consumers: https://www.ftc.gov/business-guidance/resources/start-security-guide-business

General Data Protection Regulation. (n.d.). Retrieved from General Data Protection Regulation: https://gdpr-info.eu/

Intersoft Consulting. (2025, March 7). *Right to erasure ('right to be forgotten')*. Retrieved from GDPR: https://gdpr-info.eu/art-17-gdpr/

KAMOUCHI, H. E., KISSI, M., & BEGGAR, O. E. (2023). Low-code/No-code Development : A systematic. *14th International Conference on Computing Communication and Networking Technologies (ICCCNT)*, 5.

Khedekar, L., Vaidya, M., Sonwane, V., Shivade, V., Singh, V., Wasule, V., & Ganjoo, V. (2025, 18 May). *Farmer Circle:An Online Auction Website for.* Retrieved from IEEE Xplore: https://ieeexplore.ieee.org/document/10391970

Khurana, N., Bansal, M., Thakur , A., Varshitha, V. S., Tannu , & Kaur, E. (2025, April 10). *Machine Learning-Powered Real-Time.* Retrieved from IEEE Xplore: https://ieeexplore.ieee.org/document/10426464

Odoo. (2025, April 21). *Odoo*. Retrieved from Odoo: https://www.odoo.com/

Office, C. (2025, March 7). *Copyright Law of the United States (Title 17)*. Retrieved from U.S. Copyright Office: https://www.copyright.gov/title17/

PCI Security Council. (2025, May 24). *Who We Are*. Retrieved from PCI Security Council: https://www.pcisecuritystandards.org/about_us/

Sadika. (2023, September 19). *The MVC Architecture*. Retrieved from Medium: https://medium.com/@sadikarahmantanisha/the-mvc-architecture-97d47e071eb2

Scrum. (2025, March 22). *What is Scrum?* Retrieved from Scrum: https://www.scrum.org/resources/what-scrum-module

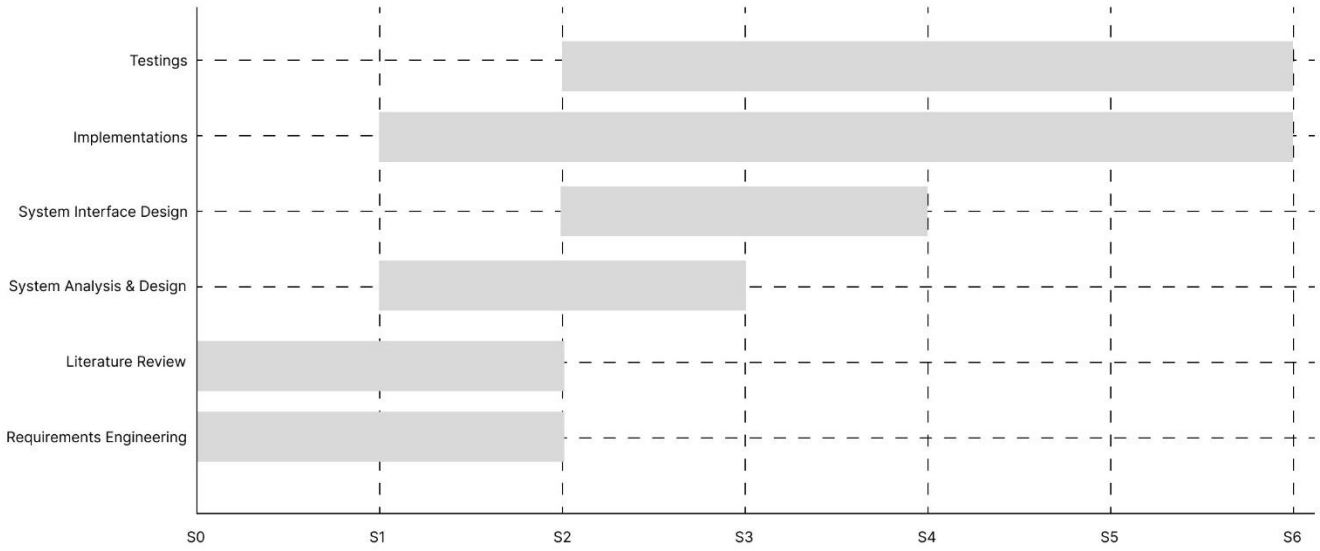
Site123. (2025, April 3). *Site123*. Retrieved from Site123: https://www.site123.com/

Velasquez, A. P., Perez-Salazar, C., Gonzalez, H., & Sanchez-Garcia, A. J. (2024). Systematic Literature Review of low-code and its future trends. *12th International Conference in Software Engineering Research and Innovation (CONISOFT)*, 8.

Wix. (2025, April 02). *Wix*. Retrieved from Wix: https://www.wix.com/

Appendix

Appendix 1



Appendix 2

UNIVERSITY OF NORTHAMPTON
FACULTY OF ARTS, SCIENCE & TECHNOLOGY

PROJECT DIARY/LOG

Student: Aadarsha Shahi Supervisor: Nischal Khadka

Date: Feburary 28 Last diary date: 29 January

Objectives:

- ~~To~~ Formulation of aim & objectives
- Research Study
- To identify ~~rese~~ ete elicitation techiniques.
- Basic requirement specification
-

Progress:

- ~~Re~~ & Literature Review
- Formulation of Aim & Objectives
- Developed Gantt Chart
- Planning of System Architecture, database design and System Interface designing tools.

Supervisor's Comments:

- Review the existing system & identify the problem domain.
- Prepare the solution strategy of the system.
- Refinement of aims & objectives of the system.
- Literature Review

28/ February /2025

AHM/2017 BSc Computing

UNIVERSITY OF NORTHAMPTON
FACULTY OF ARTS, SCIENCE & TECHNOLOGY

PROJECT DIARY/LOG

Student: Aadarsha Shahi **Supervisor:** Nischal Khadka

Date: May 5, 2025 **Last diary date:** April 20, 2025

Objectives:

- To collect & analyze requirements from potential users.
- To conduct competitor analysis and identify prioritize features.
- To develop backend architecture of the system.
- To develop wireframes and mock-ups for the front-end system.
- To implement a secure and scalable database.
- To implement basic layout & feature of the system.

Progress:

- Identify of system requirements from interview.
- Comparable system study of four prominent existing system.
- Problem Domain Specification and solution strategy.
- Identify of functional & non-functional requirements.
- ~~Developed~~ Analyze the system & design using UML class, sequence & activity diagrams.
- System Interface designing with wirfames & mocks-up.
- Implementation of login, register and basic functionalities.

Supervisor's Comments:

- further enhancement is required on the implementation.
- Revise the system design such as class, usecase & sequence diagram.
- follow proper documentation standard on the report.
- Enhancement of the problem domain & solution strategy.

05/may/2025

UNIVERSITY OF NORTHAMPTON
FACULTY OF ARTS, SCIENCE & TECHNOLOGY

PROJECT DIARY/LOG

Student: Aadarsha Shahi **Supervisor:** Nischal Khadka

Date: July 18 2025 **Last diary date:** May 5, 2025

Objectives:

~~Front-ent~~
-> Back end System Development
-> Front end System Development
-> API testings
-> Integration of Frontend & Backend System

Progress:

-> Backend System was developed with (Model + Controller + Route)
-> Components for the font end was developed & implemented using React.js.
-> API testing carried out & error are fixed.
-> Successfully integrated partial backend systems.

Supervisor's Comments:

No progress shown after interim.

July 18/2025

UNIVERSITY OF NORTHAMPTON
FACULTY OF ARTS, SCIENCE & TECHNOLOGY

PROJECT DIARY/LOG

Student: Aadarsha Shahi **Supervisor:** Nischal Khadka

Date: 1 Aug 20, 2025 **Last diary date:** July 18, 2025

Objectives:

-> To create Templates for the System
-> To integrate ~~wed~~ website & backends
-> To create additional features, BoTalk, Recommendation, Moderation
-> To perfom Unit, Integration, E2E Testing
-> ~~To op~~

Progress:

-> Templates blocks & Template Website created for reusability
-> Website metadata saved to database successfully.
-> Additional features ~~were~~ are implemented & tested.
-> Unit, Integration testings are carrid out.
-> E2E testing perfom to visualize user journey.
-> Reflection of projects and additional information added to the projet Report.

Supervisor's Comments:

- Revise the report as per feedback received on the interim.
- Further refinement is required in the system diagram such as usecase, class and activity diagram.
- Required more template framework for implementation.

No progress shown after interim.

20 August 2025

AHM/2017 BSc Computing

Appendix 3

Website Builder with Recommendation System



- Aadarsha Shahi (23834806)

Introduction

- Websites are important for small enterprises, but time, budget, and skills are limited.
- Existing tools are too complex or too rigid, leading an inconsistent results.

Gap:	Relaying on web developers even for minor changes
Solution	A **no-code** website builder based on standard design practices.

Introduction

Background

- Audience: non-technical, small teams with limited time/budget.
- Approach: No-code workflows with standard, accessible patterns.

Rational

- Adaptive templates + Drag-and-drop feature
- No-code focus on clarity/consistency/faster setup.

Aim & Objectives

Aim: To build a website builder with prebuilt templates and drag-and-drop customization.

Objectives:

- Analysis & gather user requirements
- To conduct Comparable System Study
- Literature Review and technology selection
- Develop System Designs
- Design intuitive System Interface
- Implement template selection & in-place editing.
- Deliver drag-and-drop builder.
- Set up secure & scalable database.
- System Testing

Requirement Engineering

The process to capture, agree, and manage what a system must do and under what constraints.

Elicitation:

- **Prepare:** goals, scope, key stakeholders
- **Methods: Interview, Survey, Comparable-system study**
- **Run:** open questions, note pain points & metrics
- **Validate:** review/prototype with stakeholders, refine

Interview Plan

Objective: To understand user expectations for a website builder that supports creating personal sites easily and effectively.

Duration: 1 hour

Date: 9th April 2025

Stakeholders

- **Nora Young (Student Freelancer)**
- **Brian Carter(School Administrator)**
- **Oliver Cruz(Travel Agency Owner)**
- **Grace Mitchell(E-commerce owner)**
- **Riley West(UI/UX designer)**

Survery

Objective: To gather insights into diverse users' needs and preferences for a no-code website builder.

Google Forms (Online Survey)

Duration: 5 days

Number of Respondents Targeted:

30–50

Targeted Audiences

- Students
- School administrators
- Travel agency owners
- Small business owners
- Freelancers
- UI/UX designers

Comparable System



(Odoo, 2025)



(WIX, 2025



(SITE123, 2025)



(Design, 2025)

Literature Review

- Requirements for the Development of a Website Builder with Adaptive Design, Gulmira Bekmanova, Banu Yergesh, Assel Omarbekova, Laura Orynbay, Aiganym Bessembayeva, Dinara Kabdylova, Altanbek Zulkhazhav, Beibarys Sultan, 2024
- Systematic Literature Review of low-code and its future trends, Axel Panama Velasquez, Christian Perez-Salazar, Lizbeth A. Hernandez-Gonzalez, Angel J. Sanchez-Garcia, 2024
- Evaluating the Impact of No-Code/Low-Code Backend Services on API Development and Implementation: A Case Study Approach, Pratik Dhoke, Dr. Prasad Lokulwar, 2023
- Low-code/No-code Development: A systematic literature review, Hind EL KAMOUCHI, Mohamed KISSI, Omar EL BEGGAR, 2023

Research Methodology

- **Agile Scrum (2 week sprints): priority-led** iterations with rapid prototyping, stakeholder feedback, and incremental releases.
- **Practices:** Daily Scrum, weekly refinement.
- **Mixed methods:** surveys, interviews, and 4-system comparison using checklists/performance matrices.
- **Ethics & analysis:** GDPR-compliant, anonymized consent; surveys in Excel; interviews recorded in text.
- **Evaluation:** Balance user needs and technical performance.

System Analysis and Design

Purpose: To visualize and model system components and interactions to guide design.

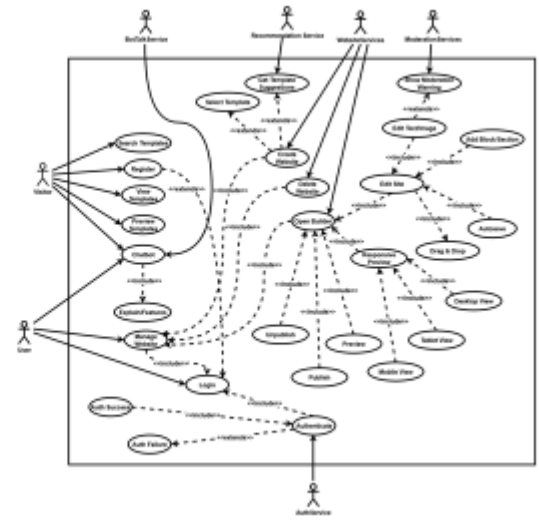
Diagram Used:

- Use Case Diagram
- Class Diagram
- Sequence Diagram
- Activity Diagram
- Database Schema

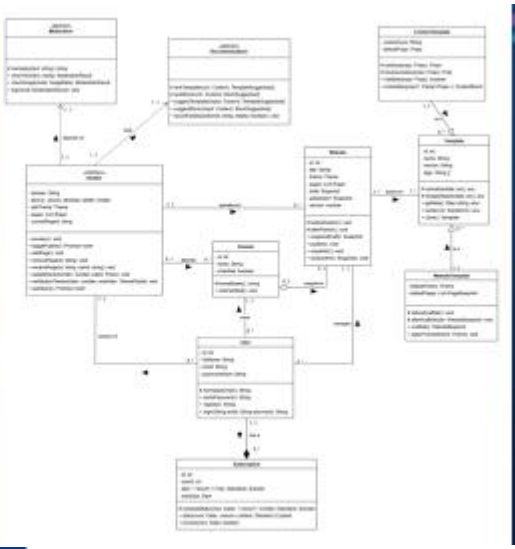
Benefits:

- Clear Communication of system design
- Helps identify system requirements and dependencies
- Guided development and testing phases

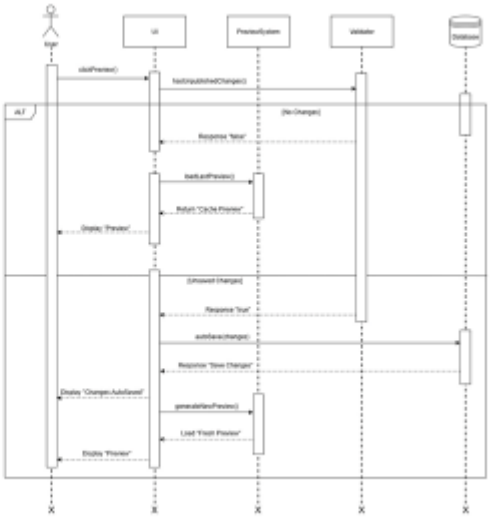
Use Case Diagram



Class Diagram



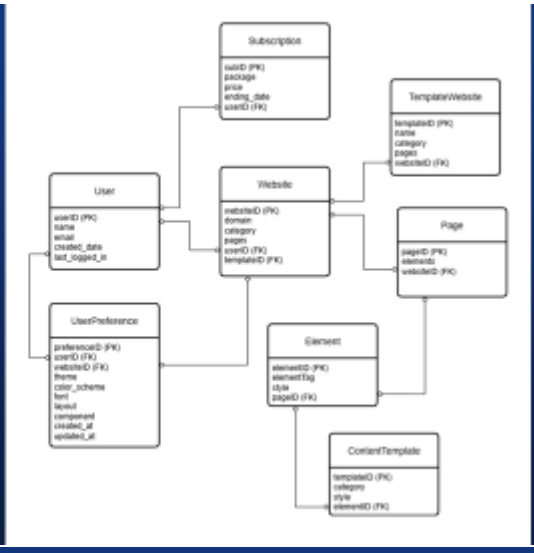
Sequence Diagram



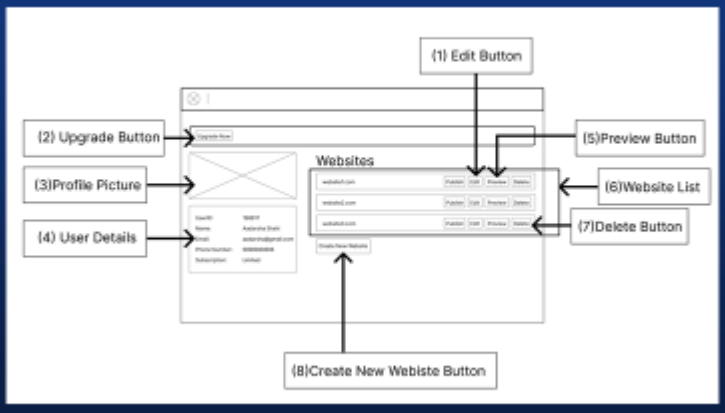
Activity Diagram



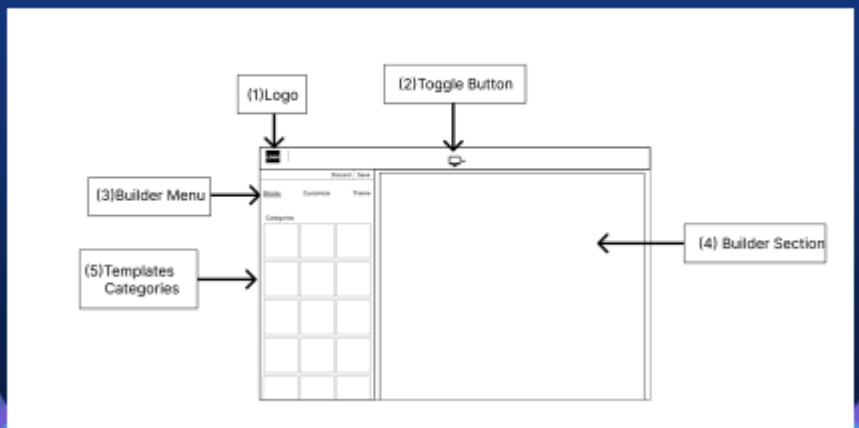
Database Schema



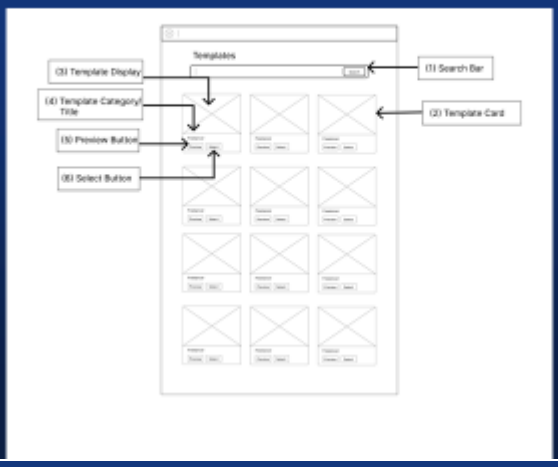
System Interface Design/ Wireframe



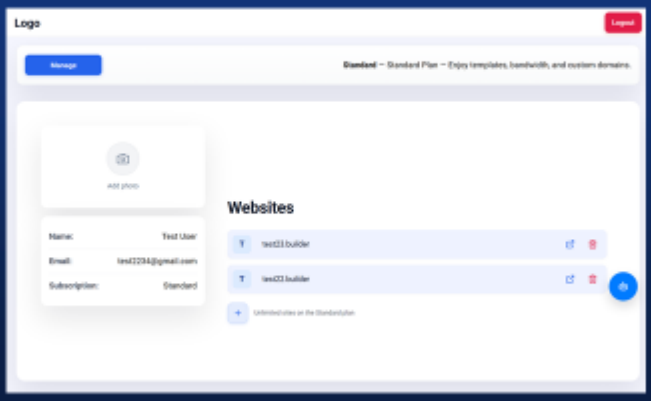
System Interface Design/ Wireframe



System Interface Design/ Wireframe



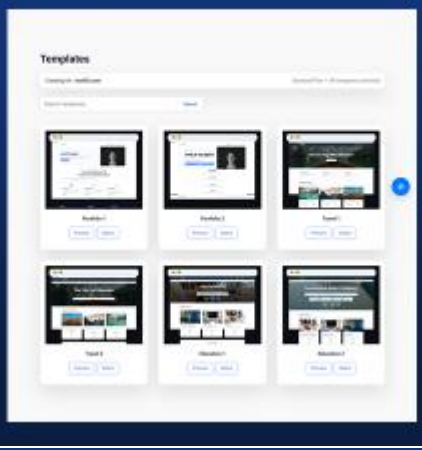
System Interface Design/ Screen Mock-up



System Interface Design/ Screen Mock-up



System Interface Design/ Screen Mock-up



Implementation

- Used MVC architectural for clean separation
- **Frontend**: React.js for the templates & drag & drop system
- **Backend:** Node.js/Express.js for APIs and Auth
- **Database**: MongoDB for users and contents

Testing

- **API**: Validate endpoints' correctness, reliability, and contracts.
- **Unit**: Validate individual functions/components work in isolation.
- **Integration**: Ensure modules work together correctly by verifying boundaries and data
- **End to End**: Validate real user journeys in a browser against a running stack.

Testing/API

Test ID	Function	Purpose	Input/Action	Expected	Actual	Status
API -003	Create Website	Create a website doc for the domain	POST {{baseUrl}}/websites | Headers: Authorization: Bearer {{token}}, Content-Type: application/json | Body: {"domain":"{{domain}}"} (or {"domainId":"{{domainId}}"})	201 Created { message:"Website created.", website:{ _id, domain, ... } } or 409 Conflict with existing website.	As Expected	Pass
API -004	Save Draft (legacy)	Persist a minimal valid draft snapshot	PUT {{baseUrl}}/websites/{{websiteId}}/draft | Headers: Authorization: Bearer {{token}}, Content-Type: application/json | Body: {"siteTheme":{"name":"default"},"pages":[{"slug":"home","mode":"raw","meta":{"title":"Home","description":"Landing"},"html":"<section> Hello </section>","css":"h1{font-weight:700}"}],"device":"desktop","siteMeta":{"footerText":"© {{domain}}"}}	200 OK { message:"Draft saved.", websiteId, updatedAt }. Server rejects oversize with 413; invalid id with 400; missing/foreign site with 404.	As Expected	Pass

Testing/API






Testing/Unit

Test ID	Function	Purpose	Input/Action	Expected	Actual	Status
UI-01	Render	Show templates website & contents	Load builder page	Builder loads template website & contents	Builder loads template website & contents	PASS
UI-02	Drag-and-Drop	Add block to canvas	Drag template onto canvas	New block appears	New block appears	PASS
UI-03	Section Click	Select section for editing	Click on section area	Section selected	Section selected	PASS
UI-04	Moderation	Block bad text edits	Try to save bad text	Warning modal appears	Warning modal appears	PASS

Testing/Unit



Testing/Unit



UI-01



UI-03



UI-02



UI-04

Testing/Integration

Test ID	Area	Purpose	Input/Action	Expected	Actual	Status
INT-001	Builder	Load site + autosave on theme change	Change theme	Autosave triggered	Autosave triggered	Pass
INT-002	Builder	Standard: footer editable & saved	Edit footer (Standard)	Save OK (moderation allow)	Save OK (moderation allow)	Pass
INT-003	Builder	Limited: footer hidden; default autosaved	Limited plan	Default footer sent	Default footer sent	Pass
INT-004	Builder	Preview opens & shares token	Click Preview	New window + token broadcast	New window + token broadcast	Pass

Testing/Integration

```
D:\NAMI\website-builder\server>npm run test:int

> website-builder@1.0.0 test:int
> vitest -c vitest.int.config.ts


 DEV  v1.6.1 D:/NAMI/website-builder/server

 ✓ tests/integration/website.int.test.js (6) 4713ms
 ✓ tests/integration/domain.int.test.js (9) 4348ms
 ✓ tests/integration/subscription.int.test.js (9) 4229ms
 ✓ tests/integration/profileImage.int.test.js (9) 4331ms
 ✓ tests/integration/user.int.test.js (7) 2806ms
 ✓ tests/integration/moderation.int.test.js (9) 1979ms
 ✓ tests/integration/reco.int.test.js (5) 2198ms

 Test Files  7 passed (7)
      Tests  54 passed (54)
   Start at  09:13:13
   Duration  9.91s (transform 1.78s, setup 5.23s, collect 690ms, tests 24.60s, environment 4ms, prepare 1.80s)


 PASS  Waiting for file changes...
       press h to show help, press q to quit
```

Testing/Integration



INT-001



INT-003



INT-002

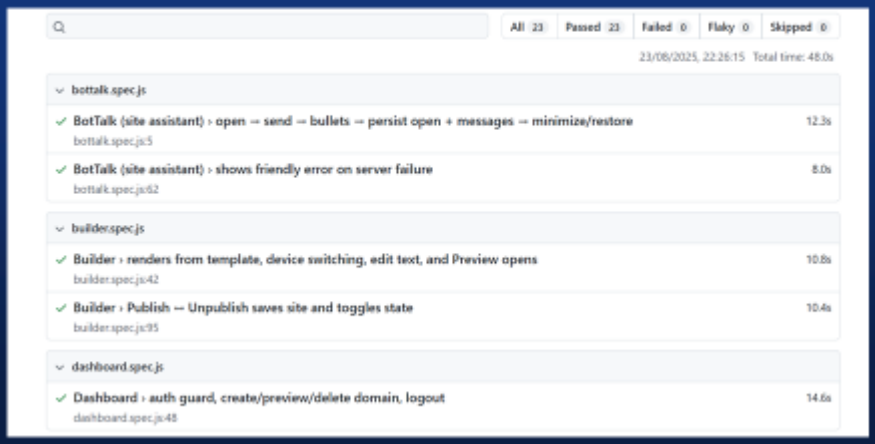


INT-004

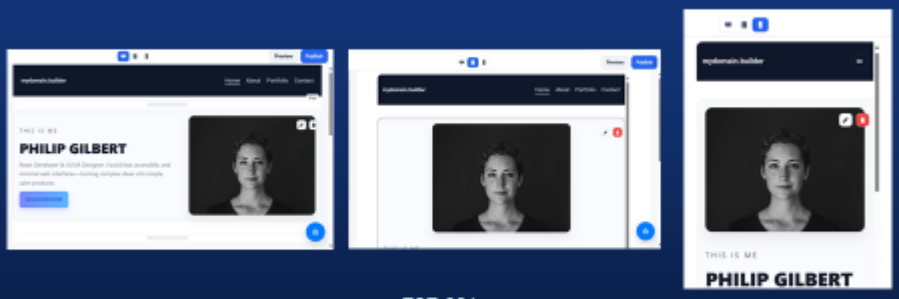
Testing/End-to-End

Test ID	Function	Purpose	Input/Action	Expected	Actual	Status
E2E-001	Builder	renders from template, device switching, edit text, and Preview opens	Open Builder with template; switch device viewport; edit text block; click Preview to verify edited content.	Template renders; viewport switches; text edit persists; Preview shows changes.	As expected.	Pass
E2E-002	Builder	Publish ↔ Unpublish saves site and toggles state	Publish from Builder; verify public site; Unpublish; verify visibility/state.	Publish exposes site; Unpublish hides/reverts; UI state toggles correctly.	As expected.	Pass
E2-003	Dashboard	auth guard, create/preview/delete domain, logout	Visit /dashboard logged out (expect redirect); login; create domain; preview; delete; logout.	Guard redirects guests; create/preview/delete succeed; logout returns to public route.	As expected.	Pass

Testing/End-to-End

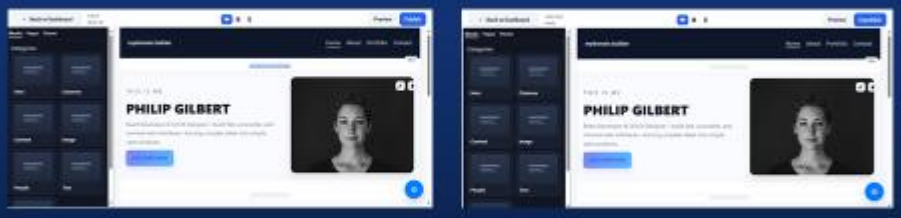


Testing/End-to-End



E2E-001

Testing/End-to-End



E2E-002

Testing/End-to-End







E2E-003



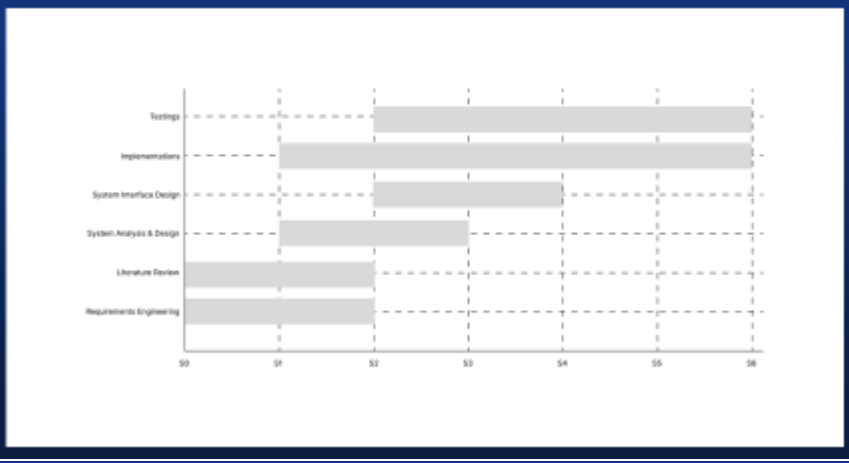
Future Expansion

- More templates: Broaden the library to meet diverse user requirements.
- E-commerce: Browse the catalog, add to cart, and complete checkout.
- Travels: Book online with live availability and instant confirmations.
- Payments for individual users: Send payouts and issue receipts per user.
- Admin dashboard: Track sites, users, and billing at a glance.
- Login/Signup (per website): Offer email/social sign-in with role-based access.
- User dashboard (per website): Manage profile, orders, and bookings in one place.

Conclusion

- Built a no-code website builder with drag-and-drop, adaptive templates, and a recommendation system to fulfill the aim.
- Delivered a scalable MERN + MVC design with clear models, flows, and schema to guide development.
- Validated quality via layered testing: API, unit, integration, and end-to-end user journeys.
- Outcome: faster site creation, lower skill barrier, consistent responsive results for small teams.

Gantt Chart



Thank You

Any Suggestions?