Software Requirements Specification and Design Document

for

PageForge

Version 2.0 Approved

Submitted by:

Aaryan Gupta
Praveen Parakh
Ayush Kukreti
Abhishek Kumar
Shubham Kumar

Submitted to:

Rekha R
Assistant Professor

Revision History

S.No	Name	Date	Reason For Changes	Version
1	PageForge Team	January 29, 2026	Initial Draft	1.0
2	PageForge Team	February 26, 2026	Added 4.0 Software Design and 7.0 Traceability	2.0

Abstract

PageForge is a self-hosted static site deployment platform designed to provide developers and teams with full control over their deployment infrastructure. This document serves as the Software Requirements Specification (SRS) and Design Document for PageForge, covering the functional and non-functional requirements, external interface specifications, system architecture, detailed software design, database schema, security design, and traceability of requirements to design components. It is intended to guide the development, testing, and maintenance of the platform, ensuring that all stakeholders have a clear and comprehensive understanding of the system's scope, capabilities, and constraints.

Contents

1 Introduction

1.1 Purpose

This Software Requirements Specification (SRS) document describes the complete functional and non-functional requirements for **PageForge**, a self-hosted static site deployment platform, version 1.0. PageForge is inspired by **Cloudflare Pages** but takes a fundamentally different approach to build execution: where Cloudflare Pages spins up full virtual machines to build static sites, PageForge uses lightweight **Docker containers**, resulting in significantly faster build start-up times and lower resource overhead. PageForge enables developers to build, deploy, and serve static websites from Git repositories or ZIP archives with automated build pipelines, custom domain support, and GitHub integration — all running entirely on the developer's own infrastructure. This document covers the entire system including the web dashboard, REST API, build worker, and infrastructure components.

1.2 Document Conventions

- **Bold text** is used for emphasis and key terms.
- `Monospace text` is used for code, file paths, API endpoints, and technical identifiers.
- Requirements are identified with unique tags: `FR-XX` for functional requirements, `NFR-XX` for non-functional requirements.
- Use cases are identified as `UC-XX`.
- Design artifacts are identified as `D-CL-XX` (class diagrams), `D-SQ-XX` (sequence diagrams), `D-DB-XX` (database), `D-SEC-XX` (security).
- Priority levels: **High** (essential for core functionality), **Medium** (important but not blocking), **Low** (nice-to-have).
- Higher-level requirement priorities are inherited by their child requirements unless explicitly overridden.

1.3 Intended Audience and Reading Suggestions

This document is intended for:

- **Developers:** Sections 3–6 for external interfaces, system features, design, and architecture. Start with Section 5.1 (System Architecture) for an overview.
- **Project Managers:** Sections 1–2 for scope and overview, Section 7 for traceability.
- **Testers:** Sections 4.1–4.11 for feature requirements and stimulus/response sequences, Section 5.2 for sequence diagrams.
- **Evaluators:** Section 5 (Software Design) for architecture, class diagrams, sequence diagrams, database schema, and security design. Section 7 for requirement-to-design traceability.
- **End Users:** Sections 1.4, 2.1–2.3 for product scope and user classes.

1.4 Product Scope

PageForge is a self-hosted alternative to Cloudflare Pages, Vercel, and Netlify, focused exclusively on static site deployment. Unlike Cloudflare Pages which uses virtual machines for builds, PageForge leverages Docker containers for faster build initialization and lower overhead. It provides:

- A web dashboard for managing projects, deployments, environment variables, and custom domains.
- Git-based deployment from GitHub repositories (public and private) with OAuth integration.
- ZIP file upload as an alternative deployment source.
- Dockerized build pipelines with configurable build commands, resource limits, and security isolation.
- Automatic static site serving via Caddy reverse proxy with dynamic route management.
- S3-compatible artifact storage via MinIO.
- Multi-user authentication with email/password credentials, OTP-based email verification during registration, and OTP-based password reset.
- Real-time build log streaming via WebSocket and Redis Pub/Sub.

The platform is designed for individual developers or small teams who want full control over their deployment infrastructure without vendor lock-in.

1.5 References

1. Next.js 14 Documentation: https://nextjs.org/docs
2. NextAuth.js v5 (Auth.js) Documentation: https://authjs.dev
3. MongoDB/Mongoose Documentation: https://mongoosejs.com/docs/
4. BullMQ Documentation: https://docs.bullmq.io/
5. Docker Engine API: https://docs.docker.com/engine/api/
6. Caddy Server Admin API: https://caddyserver.com/docs/api
7. MinIO JavaScript Client: min.io/docs/.../minio-javascript.html
8. GitHub OAuth Documentation: https://docs.github.com/en/apps/oauth-apps
9. Roger S. Pressman, *Software Engineering: A Practitioner's Approach*, Appendix A.
10. IEEE Std 830-1998, *IEEE Recommended Practice for Software Requirements Specifications*.

2 Overall Description

2.1 Product Perspective

PageForge is a new, self-contained product that fills the gap between fully managed deployment platforms (Cloudflare Pages, Vercel, Netlify) and manual server configuration. While platforms like Cloudflare Pages rely on virtual machines for build isolation, PageForge uses Docker containers which offer faster cold-start times and more efficient resource utilization. It is designed to run entirely on a user's own infrastructure — a single Linux server with Docker — eliminating dependency on third-party hosting services.

The system consists of four major subsystems:

1. **Web Application (Next.js 14):** Dashboard UI and REST API server.
2. **Build Worker:** Background job processor that executes builds in Docker containers.
3. **Infrastructure Services:** MongoDB, Redis, MinIO, and Caddy running as Docker containers.
4. **External Integrations:** GitHub OAuth and GitHub API for repository access.

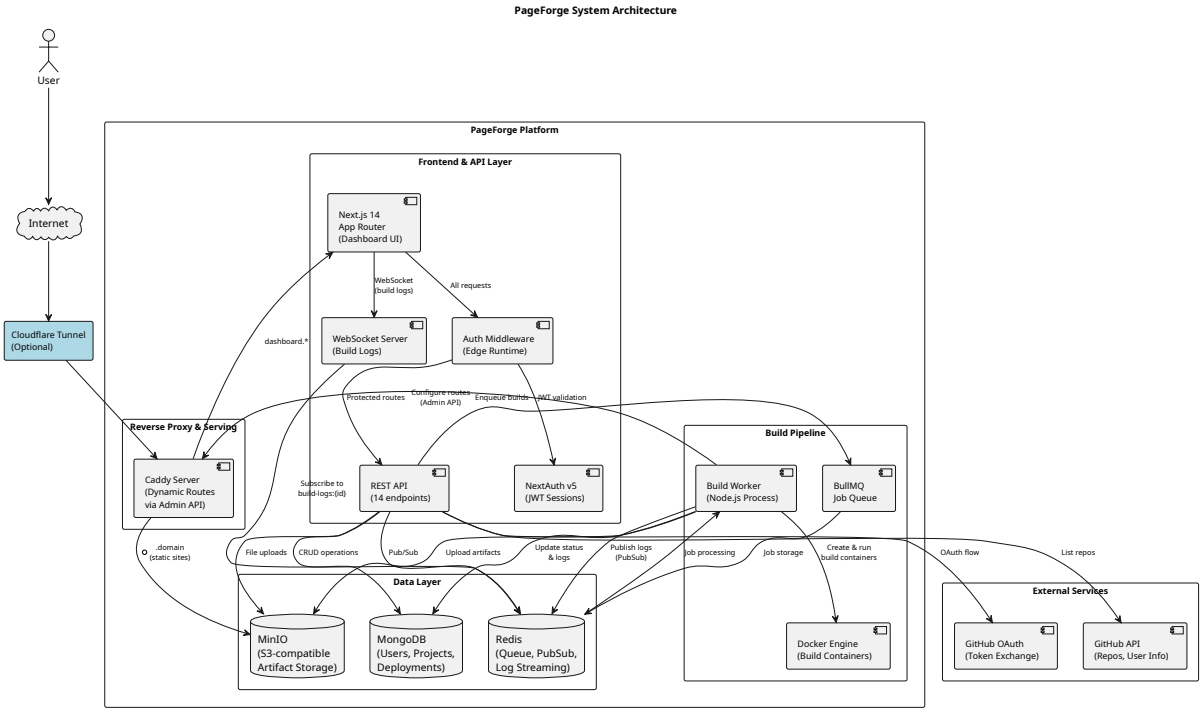

Figure 1: PageForge System Architecture Overview

2.2 Product Functions

The major functions of PageForge are:

- **User Authentication:** Registration with email OTP verification, login (email/-password), forgot password via OTP, session management, logout.

- **GitHub OAuth Integration:** Connect/disconnect GitHub account, import private repositories.
- **Project Management:** Create, read, update, delete projects with Git or ZIP sources.
- **Deployment Pipeline:** Trigger builds, execute in Docker containers, stream logs, store artifacts.
- **Environment Variables:** Configure build-time environment variables per project.
- **Custom Domains:** Add custom domains, verify via DNS CNAME, configure routing.
- **Static Site Serving:** Serve deployed sites via Caddy with automatic subdomain routing.
- **Build Log Viewer:** Real-time streaming of build output with stdout/stderr separation.

2.3 User Classes and Characteristics

1. **Developer (Primary User):** Technical users who deploy static websites. They interact with all features: project creation, deployment, domain management, and GitHub integration. Expected to understand Git, build tools (npm/yarn), and DNS configuration.
2. **Administrator:** The person who installs and maintains the PageForge server. Responsible for Docker, DNS, and infrastructure configuration. May also be a developer user.
3. **End Visitor:** Visitors who access the deployed static sites. They interact only with the served content through Caddy, not with the PageForge dashboard.

2.4 Operating Environment

- **Server OS:** Linux (tested on Arch Linux; compatible with any Linux distribution with Docker).
- **Runtime:** Node.js 20+ (LTS), Docker Engine 24+.
- **Database:** MongoDB 7.0+.
- **Cache/Queue:** Redis 7.0+.
- **Object Storage:** MinIO (S3-compatible).
- **Reverse Proxy:** Caddy 2.x with JSON admin API.
- **Package Manager:** pnpm 9+ with workspace support.
- **Browser:** Modern browsers (Chrome 90+, Firefox 90+, Safari 15+, Edge 90+).
- **Optional:** Cloudflare Tunnel for public access without port forwarding.

2.5 Design and Implementation Constraints

- **Monorepo Structure:** pnpm workspace with three packages (`apps/web`, `apps/worker`, `packages/shared`).
- **TypeScript Only:** All code must be written in TypeScript with strict type checking.
- **Edge Runtime Compatibility:** Middleware must not import Node.js-only modules (Mongoose, bcrypt) due to Next.js Edge Runtime restrictions.
- **Docker Dependency:** Build execution requires Docker Engine access via socket (`/var/run/docker.sock`).
- **Static Sites Only:** Only static site output (HTML/CSS/JS) is supported; no server-side rendering or serverless functions in deployed sites.
- **GitHub OAuth Scope:** The `repo` scope grants read and write access; GitHub does not offer a read-only scope for private repositories via OAuth Apps.

2.6 User Documentation

- `README.md` — Installation and quick start guide.
- `.env.example` — Environment variable reference with descriptions.
- In-app contextual help text on forms and configuration screens.

2.7 Assumptions and Dependencies

- Docker Engine is installed and accessible via the Docker socket.
- MongoDB, Redis, and MinIO are running (provided via `docker-compose.yml`).
- The server has internet access for cloning Git repositories and pulling Docker images.
- DNS for the base domain (`*.itspraveen.dpdns.org`) is configured with wildcard resolution pointing to the server.
- For GitHub OAuth: a GitHub OAuth App is created with correct client ID, secret, and callback URL.
- The `node:20-alpine` Docker image is available (pulled automatically on first build).
- For custom domains: users can configure CNAME records on their DNS provider.

3 External Interface Requirements

3.1 User Interfaces

UI-01 The web dashboard shall be a Next.js 14 App Router application served on port 3000, using TailwindCSS for styling with a dark mode default theme.

UI-02 The dashboard shall provide the following primary screens: Login, Register (with OTP verification step), Forgot Password (three-step flow), Dashboard (project list), Project Detail (with Overview, Deployments, Environment, Domains, and Settings tabs), New Project Wizard (three-step), Deployment Detail (with build log viewer), and Settings (GitHub integration).

UI-03 All forms shall display inline validation errors, loading spinners during submission, and toast notifications for success/failure outcomes.

UI-04 The OTP verification screens shall present six individual digit input boxes with auto-advance on input, paste support for the full code, and auto-submit when all digits are filled.

UI-05 The build log viewer shall display real-time streaming output with visual distinction between `stdout` (default), `stderr` (red), and `system` (blue/muted) messages, with auto-scroll to the latest entry.

UI-06 All pages shall implement proper loading, error, and empty states. The dashboard shall display a centered empty-state prompt when no projects exist.

UI-07 The interface shall be usable on modern desktop browsers (Chrome 90+, Firefox 90+, Safari 15+, Edge 90+). Mobile-responsive layout is not a v1.0 requirement.

3.2 Hardware Interfaces

PageForge does not interface directly with any specialized hardware devices. The system requires:

- A Linux server with network access (for serving the dashboard, APIs, and deployed sites).
- Access to the Docker Engine socket (`/var/run/docker.sock`) for creating and managing build containers.
- Sufficient disk space for MongoDB data, Redis persistence, MinIO artifact storage, and Docker images.

3.3 Software Interfaces

SI-01 MongoDB 7.0+: Primary data store for users, projects, and deployments. Accessed via Mongoose ODM over the MongoDB wire protocol (default port 27017). Used for persistent storage of all application state and build log history.

SI-02 Redis 7.0+: Used for three purposes: (1) BullMQ job queue for build jobs, (2) Pub/Sub channels for real-time build log streaming (`build-logs:{deploymentId}`), and (3) temporary storage of OTPs (5-minute TTL) and pending registration data (10-minute TTL). Accessed via `ioredis` client over the Redis protocol (default port 6379).

SI-03 MinIO (S3-compatible): Object storage for build artifacts and uploaded ZIP files. Accessed via the MinIO JavaScript SDK using S3-compatible API (default port 9000). Artifacts are stored under `artifacts/{deploymentId}/` and uploads under `uploads/{slug}/{filename}`.

SI-04 Docker Engine 24+: Build execution environment. Accessed via the Docker Engine API over the Unix socket (`/var/run/docker.sock`) using the `dockerode` library. The system creates ephemeral containers from the `node:20-alpine` image, executes build commands, extracts artifacts, and destroys containers.

SI-05 Caddy 2.x: Reverse proxy and static file server. Configured dynamically via the Caddy JSON Admin API (default `http://localhost:2019`). PageForge adds and removes routes to map project subdomains and custom domains to MinIO artifact paths.

SI-06 GitHub API v3 (REST): Used for two integrations: (1) OAuth App flow for user authorization (`github.com/login/oauth/authorize` and `github.com/login/oauth/access_token`), and (2) fetching user profile and repository listings via `api.github.com`. Requires a registered GitHub OAuth App with `repo` scope.

SI-07 SMTP Server: Used for sending OTP emails during registration and password reset. Accessed via `nodemailer` over SMTP protocol. Configurable via `SMTP_HOST`, `SMTP_PORT`, `SMTP_USER`, `SMTP_PASS`, and `SMTP_FROM` environment variables. Currently configured with Brevo (Sendinblue) SMTP relay.

SI-08 Node.js 20+ Runtime: Both the web application and worker process run on Node.js. The web app uses the Next.js custom server (`server.ts`) and the worker runs as a standalone Node.js process.

3.4 Communications Interfaces

CI-01 HTTP/HTTPS: All client-to-server communication uses HTTP (development) or HTTPS (production via Caddy TLS termination). The REST API follows standard HTTP methods (GET, POST, PATCH, PUT, DELETE) with JSON request/response bodies.

CI-02 WebSocket: Real-time build log streaming uses WebSocket connections at `ws://{host}/ws/logs/{deploymentId}`. The WebSocket server runs alongside the Next.js custom server and bridges Redis Pub/Sub messages to connected browser clients.

CI-03 DNS: Custom domain verification requires CNAME DNS record resolution. The system resolves CNAME records for user-provided domains and compares against the expected target (`{slug}.{PAGEFORGE_DOMAIN}`).

CI-04 Cloudflare Tunnel (Optional): For public access without port forwarding, the system supports Cloudflare Tunnel (`cloudflared`) to expose the server securely. This is an optional deployment configuration, not a system requirement.

4 System Features

4.1 User Registration

4.1.1 Description and Priority

Allows new users to create an account with name, email, and password. Email address is verified via OTP before account creation (see Section 4.3). **Priority: High.**

4.1.2 Stimulus/Response Sequences

1. User navigates to `/register`.
2. User enters name, email, and password.
3. System validates input and checks email uniqueness.
4. System sends a 6-digit OTP to the provided email.
5. User enters the OTP on the verification screen.
6. System verifies OTP and creates user account with hashed password.
7. System redirects to login page.

4.1.3 Functional Requirements

FR-01 The system shall provide a registration form accepting name, email, and password.

FR-02 The system shall validate that all fields are non-empty and email is in valid format.

FR-03 The system shall reject registration if the email is already registered (HTTP 409). Note: this intentionally reveals email existence to provide clear user feedback; the anti-enumeration requirement FR-91 applies only to the password reset flow.

FR-04 The system shall hash passwords using bcrypt with 12 salt rounds before storage.

FR-05 The system shall require email OTP verification before finalizing account creation (see Section 4.3).

FR-06 The system shall store the user in MongoDB with `createdAt` and `updatedAt` timestamps only after successful OTP verification, and return HTTP 201.

FR-07 The system shall redirect authenticated users away from the registration page to the dashboard.

4.2 User Login (Credentials)

4.2.1 Description and Priority

Allows registered users to authenticate with email and password. **Priority: High.**

4.2.2 Stimulus/Response Sequences

1. User navigates to `/login`.
2. User enters email and password.
3. System verifies credentials against stored hash.
4. System issues JWT session token (30-day expiry).
5. System redirects to dashboard.

4.2.3 Functional Requirements

FR-08 The system shall provide a login form accepting email and password.

FR-09 The system shall authenticate users via NextAuth.js Credentials provider.

FR-10 The system shall compare submitted password against stored bcrypt hash.

FR-11 The system shall issue a JWT token containing user ID, name, and email upon successful authentication.

FR-12 The system shall set the JWT session cookie with a maximum age of 30 days.

FR-13 The system shall display "Invalid email or password" for failed authentication (without revealing which field is incorrect).

FR-14 The system shall redirect authenticated users away from the login page.

4.3 Email OTP Verification (Registration)

4.3.1 Description and Priority

Verifies a user's email address during registration by sending a one-time password (OTP) to the provided email. The user must enter the correct OTP to complete account creation. This ensures only valid, accessible email addresses are registered. **Priority: High.**

4.3.2 Stimulus/Response Sequences

1. User fills in the registration form (name, email, password) and submits.
2. System validates input and checks email uniqueness.
3. System generates a 6-digit OTP and sends it to the provided email.
4. User enters the OTP on the verification screen.
5. System verifies the OTP.
6. On success, the system creates the user account and redirects to login.

4.3.3 Functional Requirements

FR-15 The system shall send a 6-digit OTP to the user's email after initial registration form submission, before creating the account.

FR-16 The system shall generate a cryptographically random 6-digit OTP.

FR-17 The system shall store the pending registration data and OTP temporarily until verification completes or expires.

FR-18 The system shall expire OTPs after 5 minutes.

FR-19 The system shall invalidate an OTP after successful verification (single-use).

FR-20 The system shall reject expired or invalid OTPs with an appropriate error message.

FR-21 The system shall create the user account only after successful OTP verification.

FR-22 The system shall allow the user to request a new OTP if the previous one expired.

4.4 GitHub OAuth Connection

4.4.1 Description and Priority

Allows users to connect their GitHub account for importing private repositories. **Priority: High.**

4.4.2 Stimulus/Response Sequences

1. User navigates to `/settings` and clicks "Connect GitHub".
2. System redirects to GitHub OAuth authorization page.
3. User authorizes PageForge with `repo` scope.
4. GitHub redirects back with authorization code.
5. System exchanges code for access token and stores it.
6. User is redirected to settings with success status.

4.4.3 Functional Requirements

FR-23 The system shall redirect to GitHub OAuth with `client_id`, `redirect_uri`, `scope=repo`, and `state=userId`.

FR-24 The system shall exchange the authorization code for an access token via GitHub's token endpoint.

FR-25 The system shall fetch the GitHub user profile (ID, username) using the access token.

FR-26 The system shall store the access token, GitHub ID, username, and connection timestamp on the User document.

FR-27 The system shall store the GitHub access token with `select: false` (never exposed in API responses).

FR-28 The system shall provide a "Disconnect GitHub" action that removes all GitHub fields from the User.

FR-29 The system shall provide a status endpoint returning connection state, username, and connection date.

FR-30 The system shall handle OAuth errors (user denial, invalid code) gracefully with redirect to settings page with error parameter.

4.5 Project Management

4.5.1 Description and Priority

Allows users to create, view, update, and delete static site projects. **Priority: High.**

4.5.2 Stimulus/Response Sequences

1. User creates a project via the 3-step wizard (name, source, build config).
2. System generates a unique slug and stores the project.
3. User can view project list on the dashboard.
4. User can update project settings (name, source, build commands).
5. User can delete a project.

4.5.3 Functional Requirements

FR-31 The system shall provide a 3-step project creation wizard: (1) Name & Source Type, (2) Source Configuration, (3) Build Configuration.

FR-32 The system shall auto-generate a URL-safe slug from the project name.

FR-33 The system shall support two source types: `git` and `zip`.

FR-34 For Git sources, the system shall accept repository URL, branch, and provider (GitHub or Other).

FR-35 When GitHub is connected, the system shall display a searchable repository picker populated from the GitHub API.

FR-36 The system shall store default build configuration: `installCommand` ("npm install"), `buildCommand` ("npm run build"), `outputDirectory` ("dist").

FR-37 The system shall scope all projects to the authenticated user via `userId`.

FR-38 The system shall enforce unique slugs across all projects.

FR-39 The system shall support updating project name, source config, and build config via PATCH.

FR-40 The system shall support deleting a project via DELETE.

FR-41 The system shall list only projects belonging to the authenticated user.

FR-42 The system shall accept an optional Personal Access Token (PAT) per project for Git authentication.

FR-43 The system shall support removing a project-level PAT via `removeGitToken` flag.

4.6 Deployment Pipeline

4.6.1 Description and Priority

Executes builds in isolated Docker containers and serves the resulting static assets. **Priority: High.**

4.6.2 Stimulus/Response Sequences

1. User clicks "Deploy" on a project.
2. System creates a deployment record (status: `queued`).
3. System enqueues a build job via BullMQ/Redis.
4. Worker picks up the job, creates a Docker container, and executes the build.
5. Worker streams build logs to Redis Pub/Sub and MongoDB.
6. On success, worker uploads artifacts to MinIO and configures Caddy route.
7. Deployment status updates to `ready` or `failed`.

4.6.3 Functional Requirements

FR-44 The system shall create a deployment record with status `queued` and a source snapshot.

FR-45 The system shall resolve Git authentication tokens in priority order: project PAT ¿ user GitHub OAuth token ¿ none (public repos).

FR-46 The system shall enqueue the build job to BullMQ via Redis.

FR-47 The worker shall create a Docker container using the `node:20-alpine` image.

FR-48 The worker shall configure container resource limits: memory (512MB default), CPU (1 core default).

FR-49 The worker shall set `no-new-privileges` security option on containers.

FR-50 The worker shall optionally use gVisor runtime (`runsc`) when `GVISOR_ENABLED=true`.

FR-51 The worker shall inject environment variables from the project configuration into the container.

FR-52 The worker shall inject `GIT_AUTH_TOKEN` as an environment variable and configure Git credential helper.

FR-53 The worker shall automatically install `git` in the container if not present.

FR-54 The worker shall clone the Git repository (or extract ZIP) and execute build commands.

FR-55 The worker shall parse Docker multiplexed stream format (8-byte header) for stdout/stderr separation.

FR-56 The worker shall sanitize all secrets from build log output.

FR-57 The worker shall publish log entries to Redis Pub/Sub channel `build-logs:{deploymentId}`.

FR-58 The worker shall persist log entries to the deployment's `buildLogs` array in MongoDB.

FR-59 The worker shall extract the build output directory as a tar archive from the container.

FR-60 The worker shall upload the extracted build artifacts to MinIO under the path `artifacts/{deploymentId}/`.

FR-61 The worker shall configure a Caddy route mapping `{slug}.{domain}` to the MinIO artifact path.

FR-62 The worker shall update deployment status through the lifecycle: `queued` → `building` → `uploading` → `ready` (or `failed`).

FR-63 The worker shall remove the Docker container after build completion (success or failure).

FR-64 The system shall list deployments for a project sorted by creation date (newest first).

FR-65 The system shall provide a deployment detail view with build logs and status.

4.7 Environment Variables

4.7.1 Description and Priority

Allows users to configure build-time environment variables per project. **Priority: Medium.**

4.7.2 Stimulus/Response Sequences

1. User navigates to the Environment tab on a project page.
2. User adds, edits, or removes key-value pairs.
3. User saves changes.
4. Variables are injected into Docker containers during deployment.

4.7.3 Functional Requirements

FR-66 The system shall store environment variables as an array of {`key, value, encrypted`} objects on the Project document.

FR-67 The system shall provide GET and PUT endpoints for environment variables at `/api/projects/{slug}/env`.

FR-68 The system shall replace all environment variables atomically on PUT (full replacement, not merge).

FR-69 The system shall inject all project environment variables into the Docker build container.

FR-70 The system shall mask environment variable values in the UI.

4.8 Custom Domain Management

4.8.1 Description and Priority

Allows users to map custom domains to their deployed projects. **Priority: Medium.**

4.8.2 Stimulus/Response Sequences

1. User adds a custom domain on the Domains tab.
2. System provides a CNAME target for DNS configuration.
3. User configures CNAME record at their DNS provider.
4. User clicks "Verify" to trigger DNS verification.
5. System verifies CNAME and configures Caddy route for the custom domain.

4.8.3 Functional Requirements

FR-71 The system shall accept custom domain input and validate format.

FR-72 The system shall generate a CNAME target of {`slug`}.{`PAGEFORGE_DOMAIN`}.

FR-73 The system shall store domains as embedded documents with `domain`, `cnameTarget`, `verified`, and `verifiedAt` fields.

FR-74 The system shall verify custom domains by resolving CNAME records via DNS and comparing to the expected target.

FR-75 The system shall configure a Caddy route for verified custom domains pointing to the project's artifacts.

FR-76 The system shall support removing custom domains, including cleanup of Caddy routes.

FR-77 The system shall display domain verification status (Pending/Verified) in the UI.

4.9 ZIP File Upload

4.9.1 Description and Priority

Allows users to upload pre-built static site archives as an alternative to Git-based deployment. **Priority: Medium.**

4.9.2 Stimulus/Response Sequences

1. User selects "ZIP Upload" as the source type during project creation.
2. User uploads a ZIP file via the project page.
3. System stores the file in MinIO.
4. User triggers a deployment which uses the uploaded ZIP.

4.9.3 Functional Requirements

FR-78 The system shall accept ZIP file uploads via multipart form data at `/api/upload/{slug}`.

FR-79 The system shall store uploaded files in MinIO at `uploads/{slug}/{filename}`.

FR-80 The system shall update the project's `zipFileName` field after successful upload.

FR-81 The system shall use the uploaded ZIP as the source for deployment builds.

4.10 Build Log Viewer

4.10.1 Description and Priority

Provides real-time build log streaming during deployments. **Priority: Medium.**

4.10.2 Stimulus/Response Sequences

1. User navigates to a deployment detail page.
2. System establishes WebSocket connection to `ws://{host}/ws/logs/{deploymentId}`.
3. Build logs are streamed in real-time from Redis Pub/Sub.
4. After build completes, historical logs are available from MongoDB.

4.10.3 Functional Requirements

FR-82 The system shall provide a WebSocket endpoint at `/ws/logs/{deploymentId}` for live log streaming.

FR-83 The WebSocket server shall subscribe to Redis Pub/Sub channel `build-logs:{deploymentId}`.

FR-84 The system shall forward log messages (type, stream, line, timestamp) to connected WebSocket clients.

FR-85 The system shall also poll the deployment API every 2 seconds as a fallback for log updates.

FR-86 The system shall display logs with visual distinction between stdout, stderr, and system messages.

FR-87 The system shall auto-scroll to the latest log entry.

FR-88 The system shall display deployment status (queued, building, uploading, ready, failed) with colored badges.

4.11 Forgot Password (OTP-Based Password Reset)

4.11.1 Description and Priority

Allows users to reset their password by verifying their identity via a one-time password sent to their registered email address. **Priority: High.**

4.11.2 Stimulus/Response Sequences

1. User clicks "Forgot Password" on the login page.
2. User enters their registered email address.
3. System sends a 6-digit OTP to the email.
4. User enters the OTP on the verification screen.
5. System verifies the OTP and presents the password reset form.
6. User enters and confirms a new password.
7. System hashes the new password and updates the user record.
8. System redirects to the login page with a success message.

4.11.3 Functional Requirements

FR-89 The system shall provide a "Forgot Password" link on the login page navigating to `/forgot-password`.

FR-90 The system shall accept an email address and send a 6-digit OTP to it if the email is registered.

FR-91 The system shall not reveal whether an email is registered or not during the password reset flow (to prevent user enumeration via the forgot-password endpoint).

FR-92 The system shall generate a cryptographically random 6-digit OTP with a 5-minute expiry.

FR-93 The system shall invalidate the OTP after successful verification (single-use).

FR-94 The system shall present a new password form only after successful OTP verification.

FR-95 The system shall validate that the new password meets minimum requirements.

FR-96 The system shall hash the new password using bcrypt with 12 salt rounds and update the user record.

FR-97 The system shall require the user to log in again after a successful password reset. Since JWT sessions are stateless, existing tokens remain valid until their natural 30-day expiry; the password change itself prevents further logins with the old password.

FR-98 The system shall redirect to the login page after successful password reset.

5 Software Design

5.1 System Architecture

PageForge follows a **modular monorepo architecture** with clear separation between the web application, build worker, and shared types. The system uses an **event-driven build pipeline** with BullMQ for job queuing and Redis Pub/Sub for real-time log streaming.

5.1.1 Architecture Overview

The system is divided into the following layers:

1. **Presentation Layer:** Next.js 14 App Router with React components, TailwindCSS styling, and client-side hooks (`useFetch`, `useMutation`).
2. **API Layer:** RESTful API with 14 route handlers, JWT authentication middleware, and ownership-scoped data access.
3. **Business Logic Layer:** Auth config, GitHub OAuth flow, project/deployment management, build queue management.
4. **Worker Layer:** BullMQ worker process, Docker executor, artifact manager, build logger.
5. **Data Layer:** MongoDB (Mongoose ODM), Redis (queue + pub/sub), MinIO (S3 artifact storage).
6. **Infrastructure Layer:** Caddy (reverse proxy + static serving), Docker Engine (build isolation), Cloudflare Tunnel (optional public access).

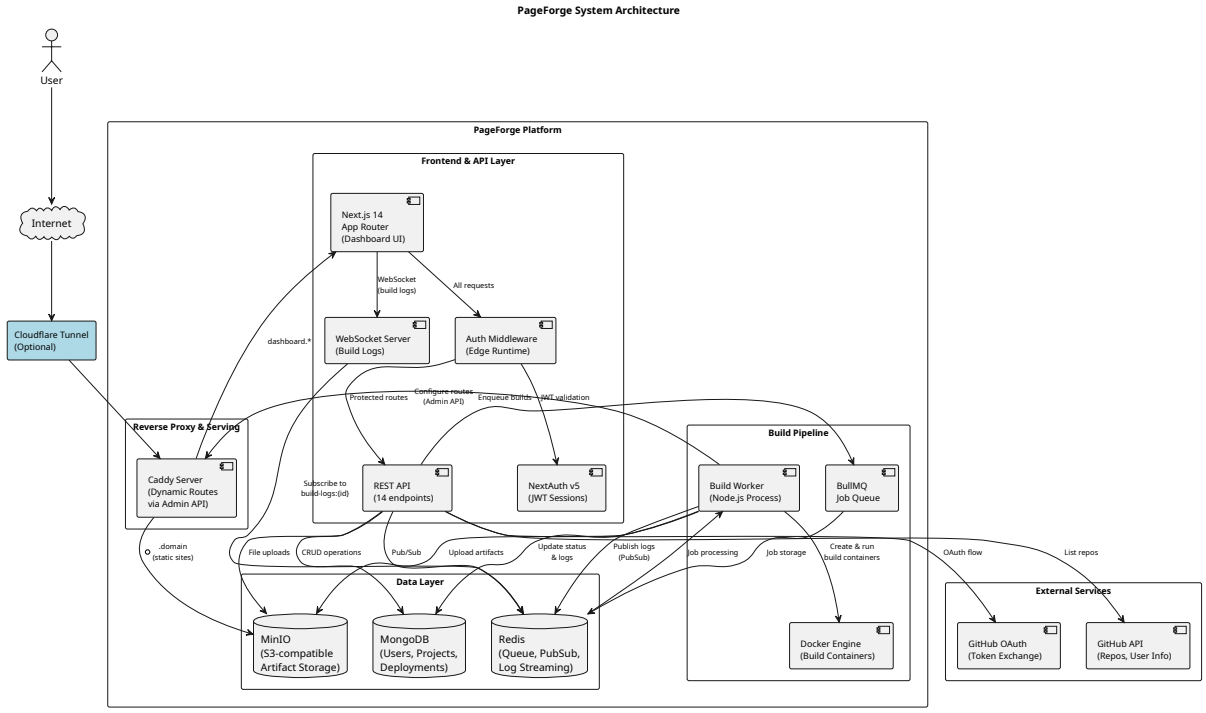

Figure 2: System Architecture Diagram

5.1.2 Monorepo Structure

```
PageForge/
  apps/
    web/          # Next.js 14 dashboard + API (Port 3000)
    worker/       # BullMQ build worker (background process)
  packages/
    shared/       # Shared TypeScript types and constants
  infra/
    docker-compose.yml  # MongoDB, Redis, MinIO, Caddy
    caddy.json          # Caddy server configuration
    init-minio.sh       # MinIO bucket initialization
```

5.1.3 API Endpoints

Method	Endpoint	Description
POST	`/api/auth/register`	User registration
GET/POST	`/api/auth/[...nextauth]`	NextAuth.js session management
POST	`/api/auth/otp/send`	Send OTP (registration or password reset)
POST	`/api/auth/otp/verify`	Verify OTP code
POST	`/api/auth/forgot-password`	Reset password after OTP verification
GET	`/api/auth/github`	Initiate GitHub OAuth flow

Method	Endpoint	Description
DELETE	`/api/auth/github`	Disconnect GitHub account
GET	`/api/auth/github/callback`	GitHub OAuth callback handler
GET	`/api/auth/github/status`	GitHub connection status
GET	`/api/github/repos`	List GitHub repositories
GET/POST	`/api/projects`	List / create projects
GET/PATCH/DELETE	`/api/projects/[slug]`	Get / update / delete project
GET/POST	`/api/projects/[slug]/deployments`	List / trigger deployments
GET	`/api/projects/[slug]/deployments/[id]`	Get deployment details
GET/PUT	`/api/projects/[slug]/env`	Get / set environment variables
GET/POST	`/api/projects/[slug]/domains`	List / add domains
POST/DELETE	`/api/projects/[slug]/domains/[domain]`	Verify / remove domain
POST	`/api/upload/[slug]`	Upload ZIP file

5.2 Detailed Design

5.2.1 Class Diagrams

The class diagram shows all major classes across the six modules: Authentication, Project Management, Deployment, Build Worker, Infrastructure, and GitHub Integration.

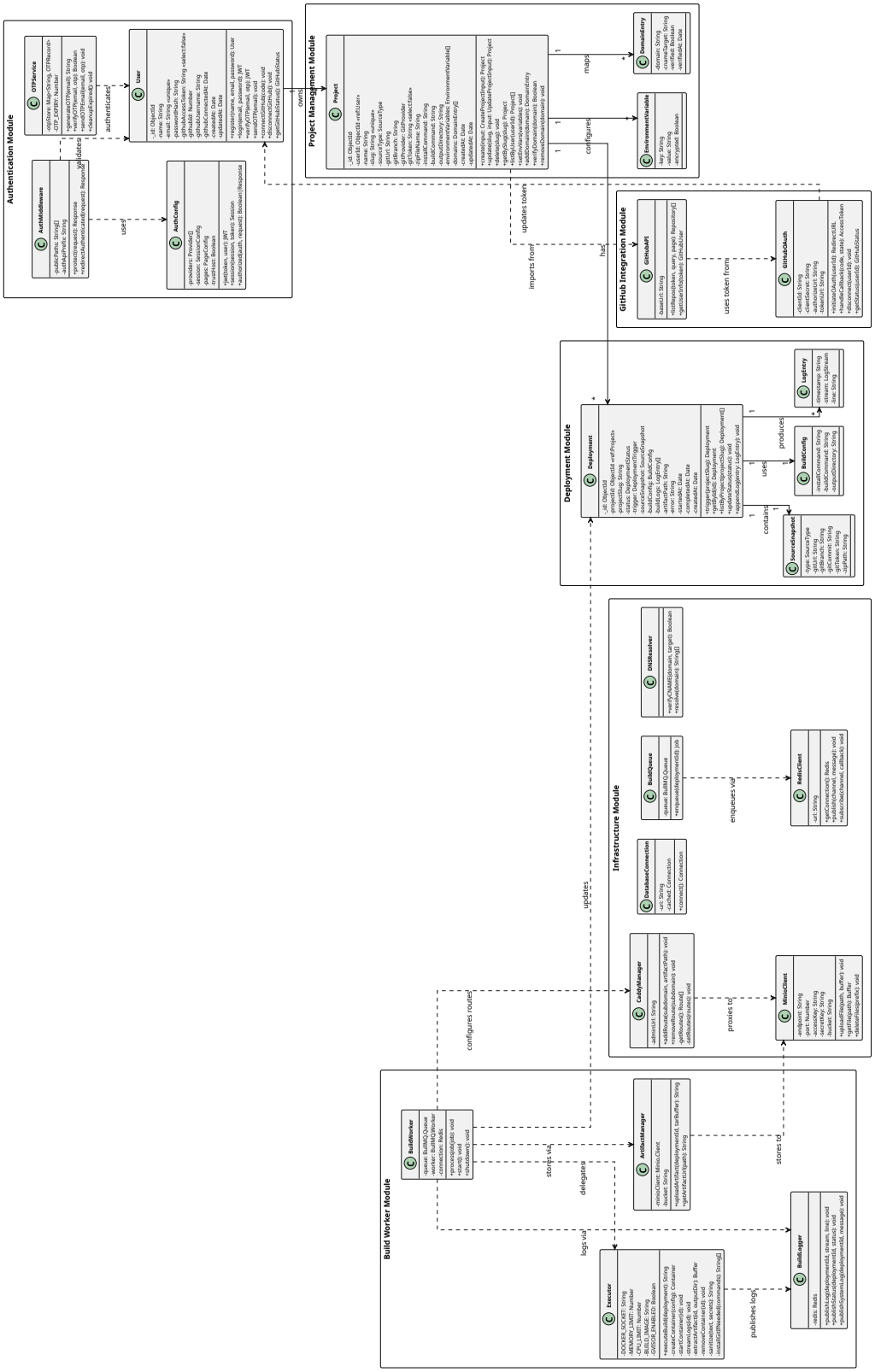

Figure 3: Complete Class Diagram (D-CL-01)

Key Design Decisions:

- **User** class owns Projects (1-to-many), and Projects own Deployments (1-to-many).
- **EnvironmentVariable**, **DomainEntry**, **SourceSnapshot**, **BuildConfig**, and **LogEntry** are embedded documents (not separate collections) for performance and atomicity.
- **Sensitive fields** (`passwordHash`, `githubAccessToken`, `gitToken`) use Mongoose `select: false` and are stripped in `toJSON`/`toObject` transforms.
- **BuildWorker** delegates to **Executor** (Docker), **ArtifactManager** (MinIO), and **BuildLogger** (Redis) following the single-responsibility principle.
- **OTPService** manages OTP generation, verification, and email delivery for registration email verification and password reset.

5.2.2 Sequence Diagrams

SD-01: User Registration

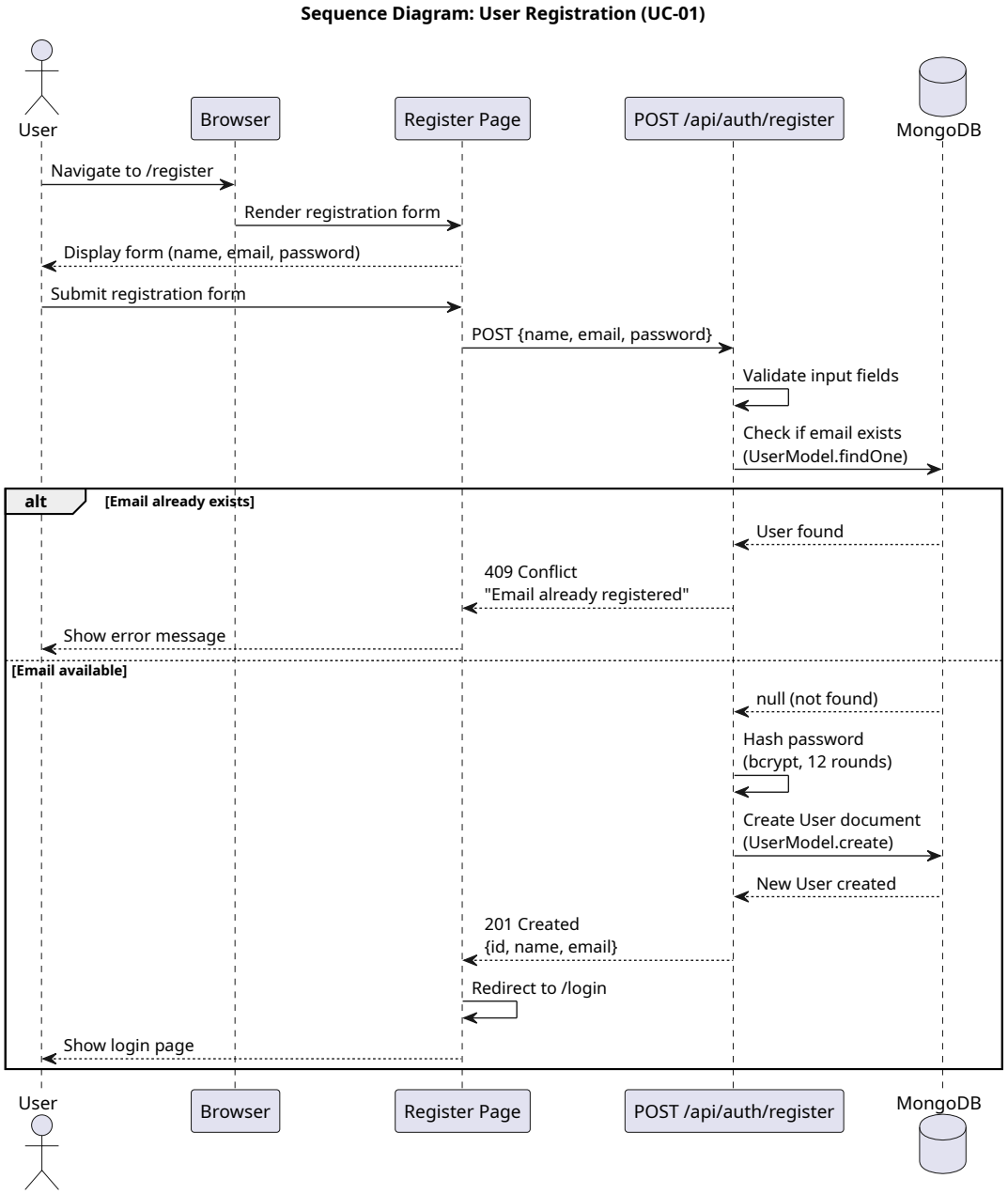

Figure 4: Sequence Diagram: User Registration (D-SQ-01)

SD-02: User Login (Credentials)

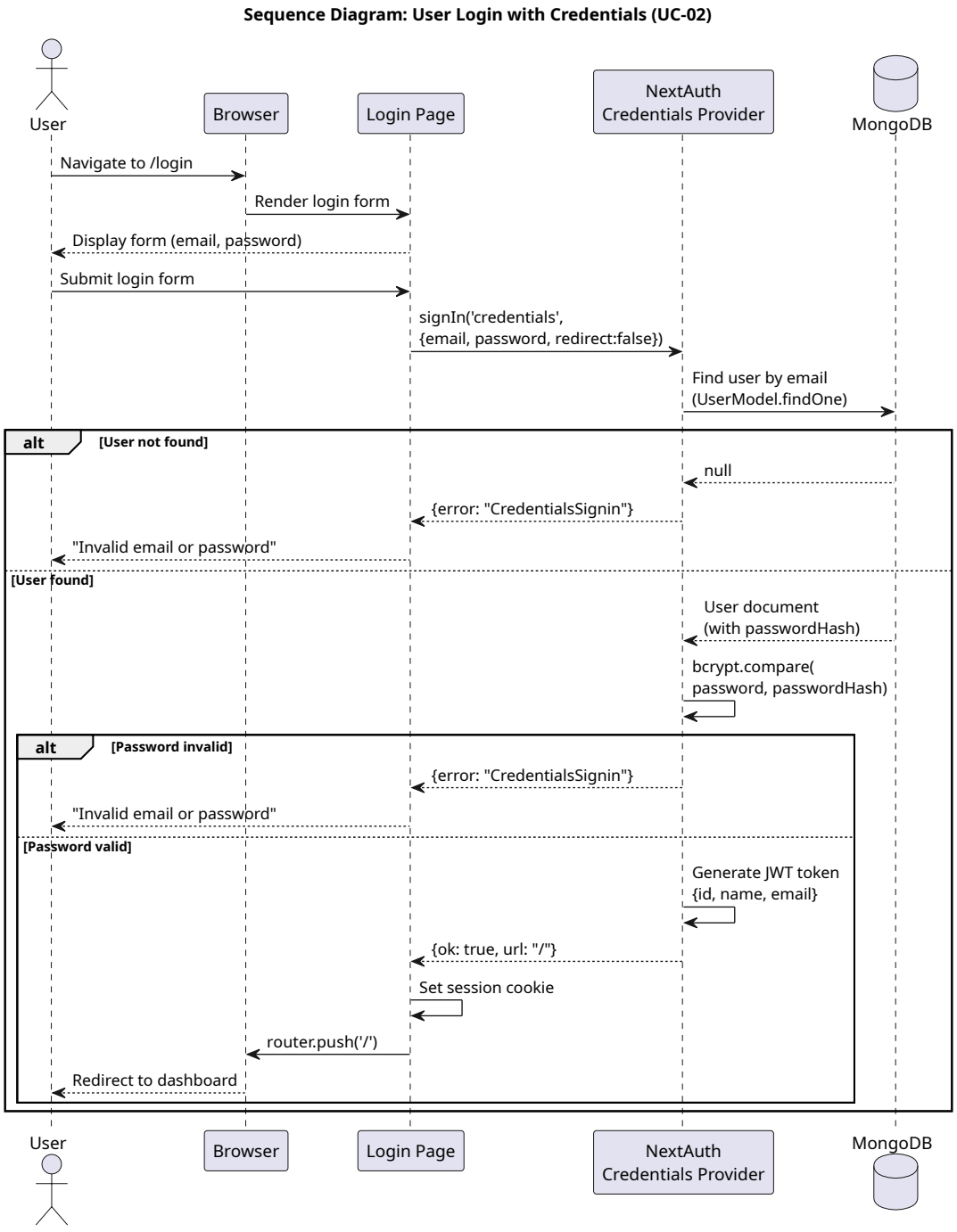

Figure 5: Sequence Diagram: User Login with Credentials (D-SQ-02)

SD-03: Email OTP Verification (Registration & Password Reset)

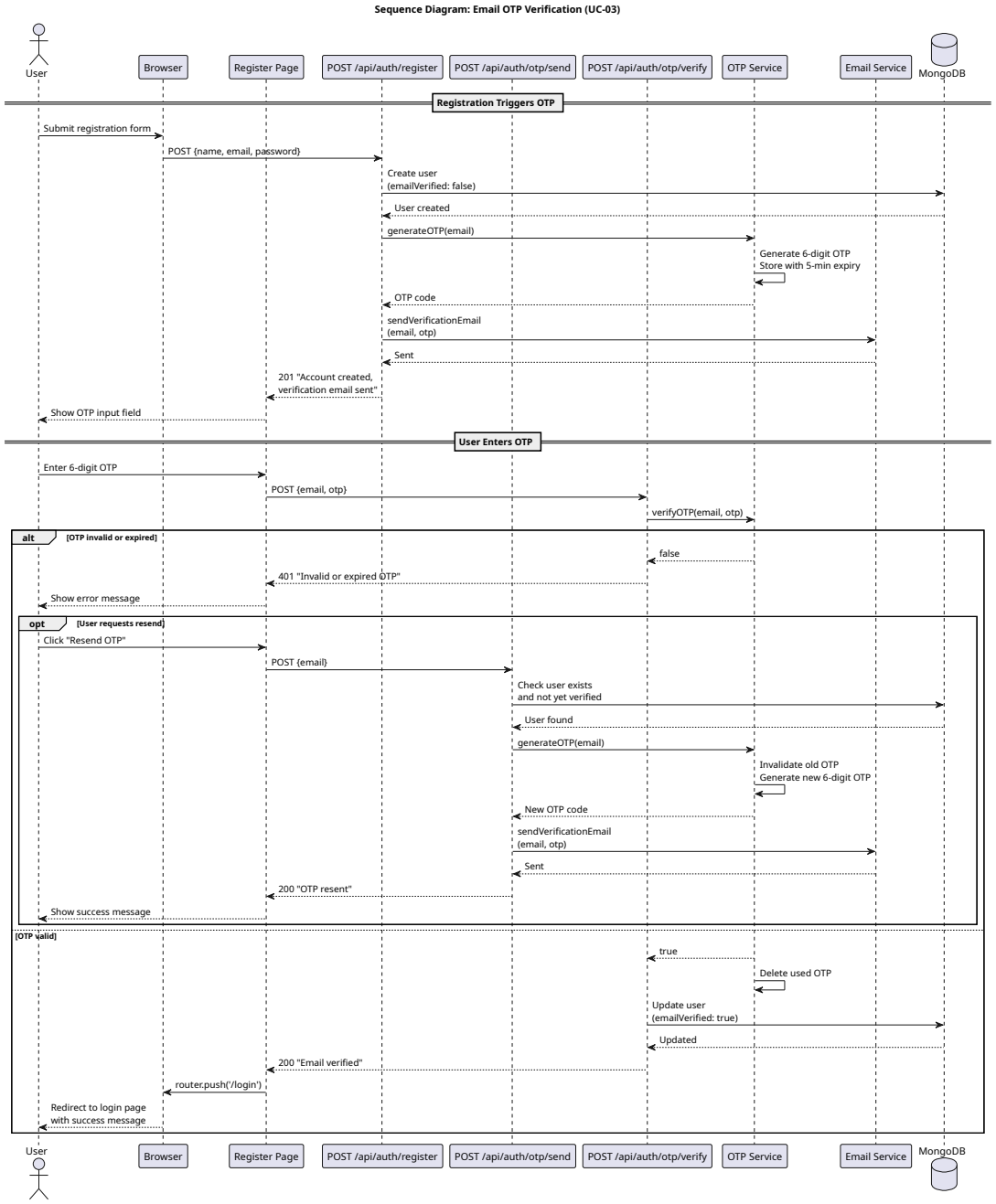

Figure 6: Sequence Diagram: Email OTP Verification (D-SQ-03)

SD-04: Create New Project

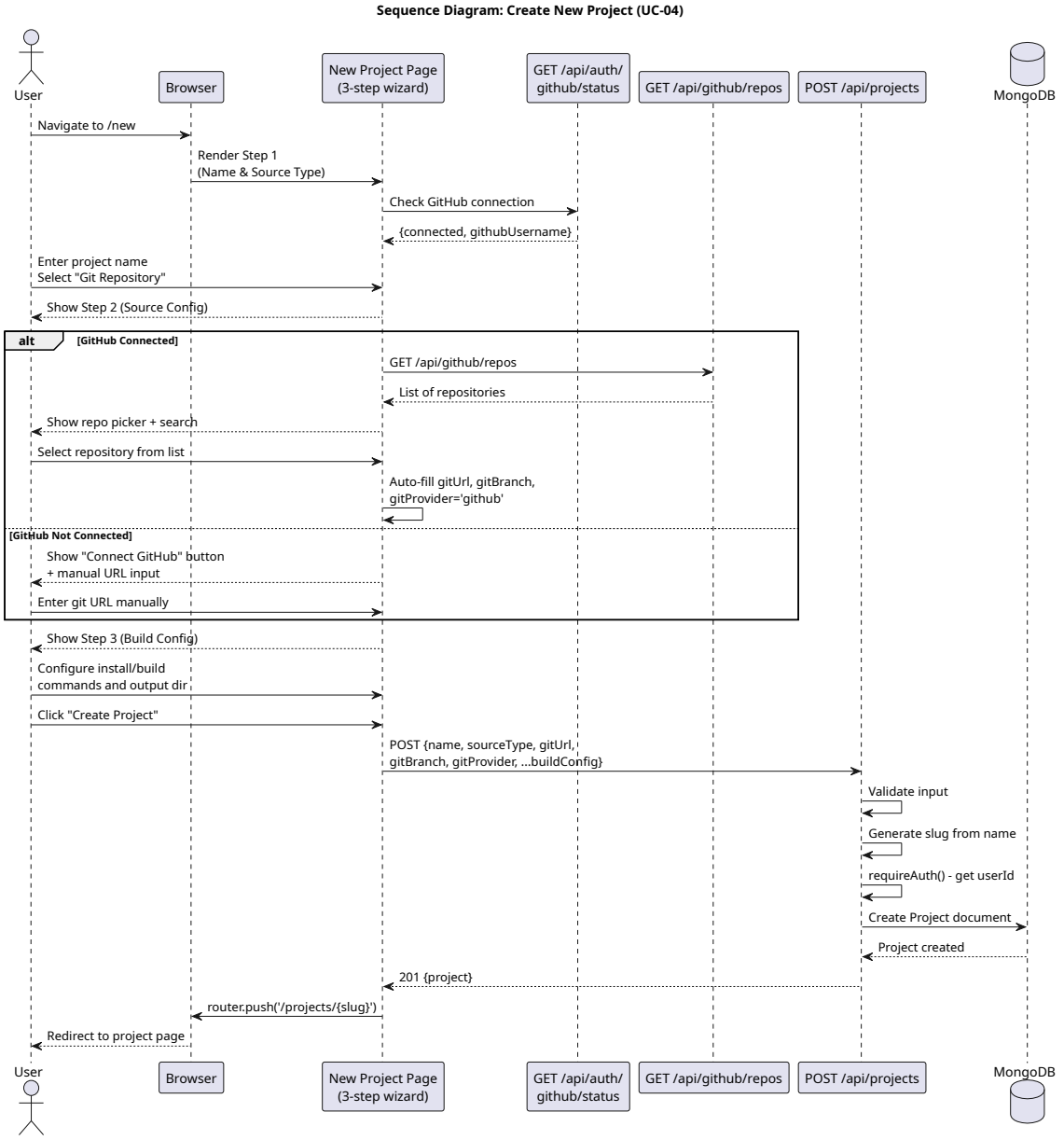

Figure 7: Sequence Diagram: Create New Project (D-SQ-04)

SD-05: Trigger Deployment and Build Process

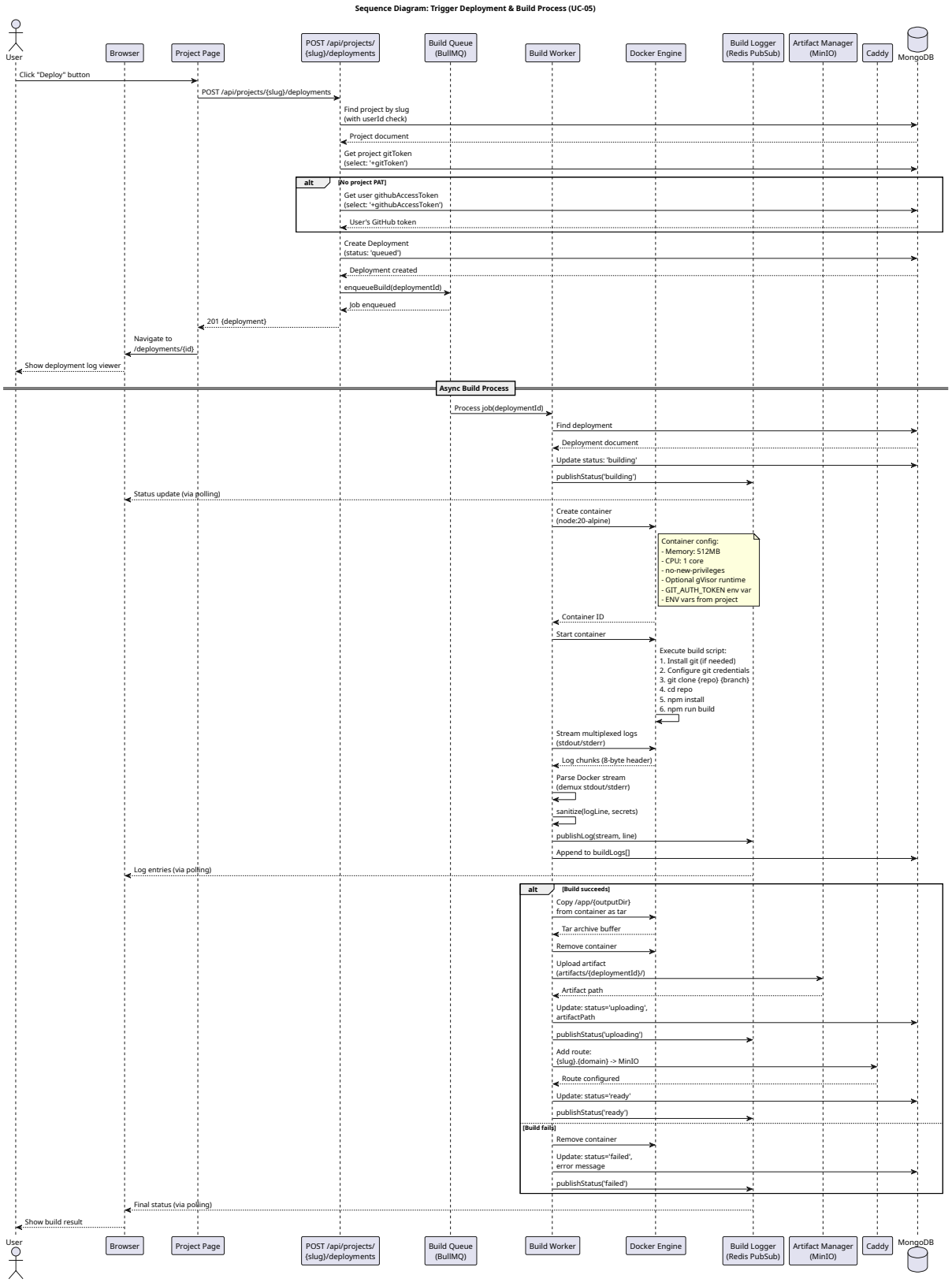

Figure 8: Sequence Diagram: Deployment and Build Process (D-SQ-05)

SD-06: GitHub OAuth Connection

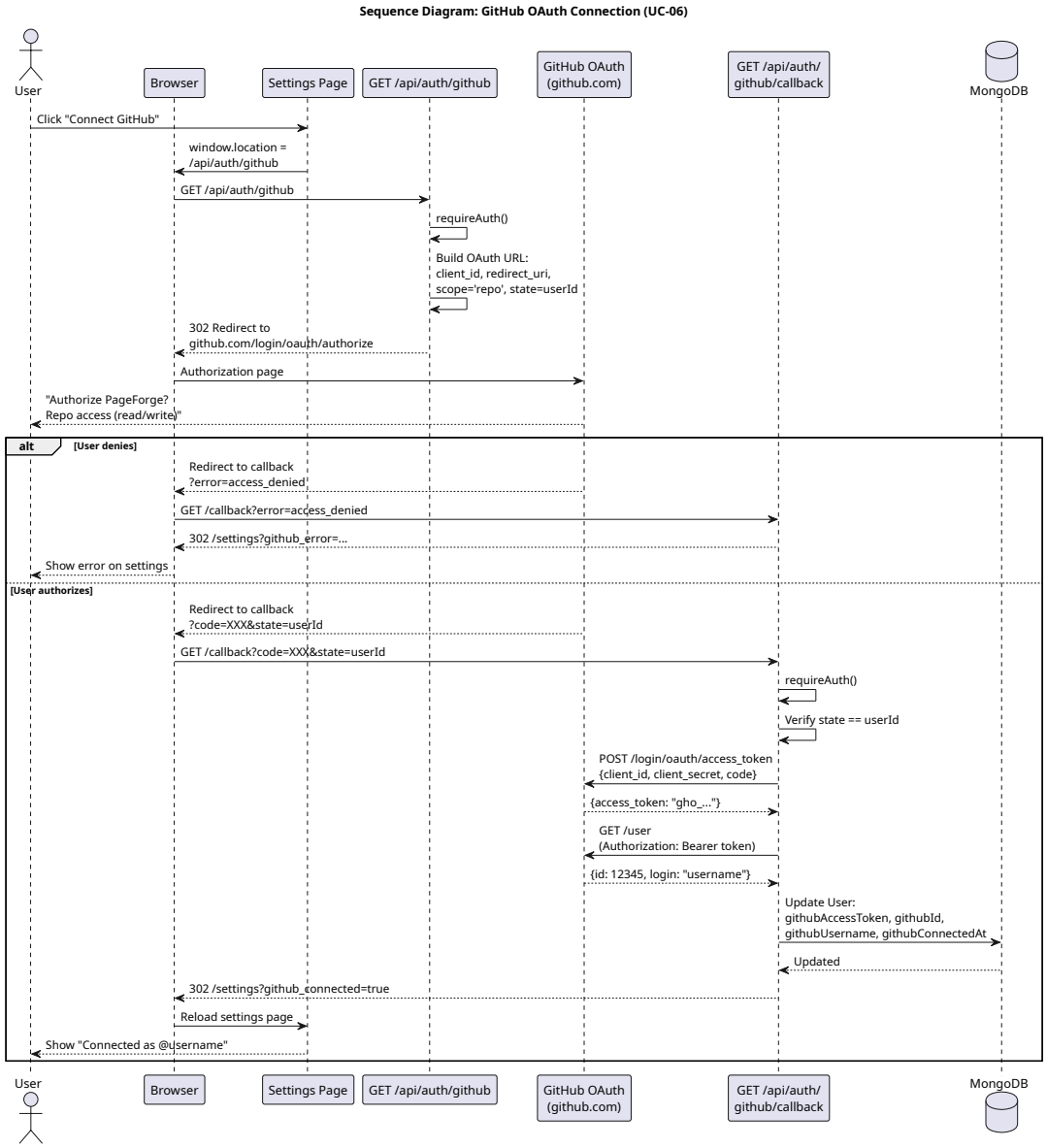

Figure 9: Sequence Diagram: GitHub OAuth Connection (D-SQ-06)

SD-07: Custom Domain Management

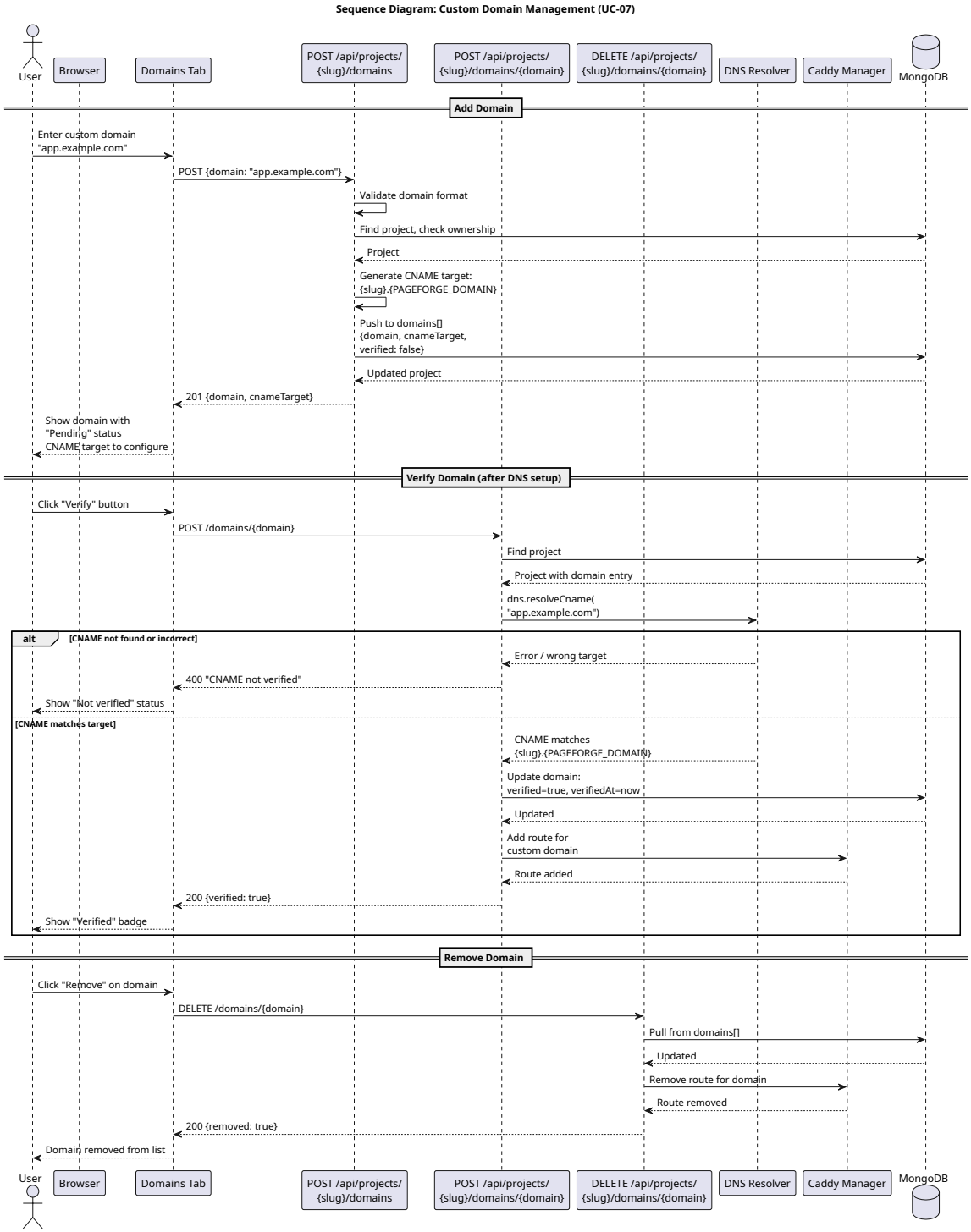

Figure 10: Sequence Diagram: Custom Domain Management (D-SQ-07)

SD-08: ZIP Upload and Deploy

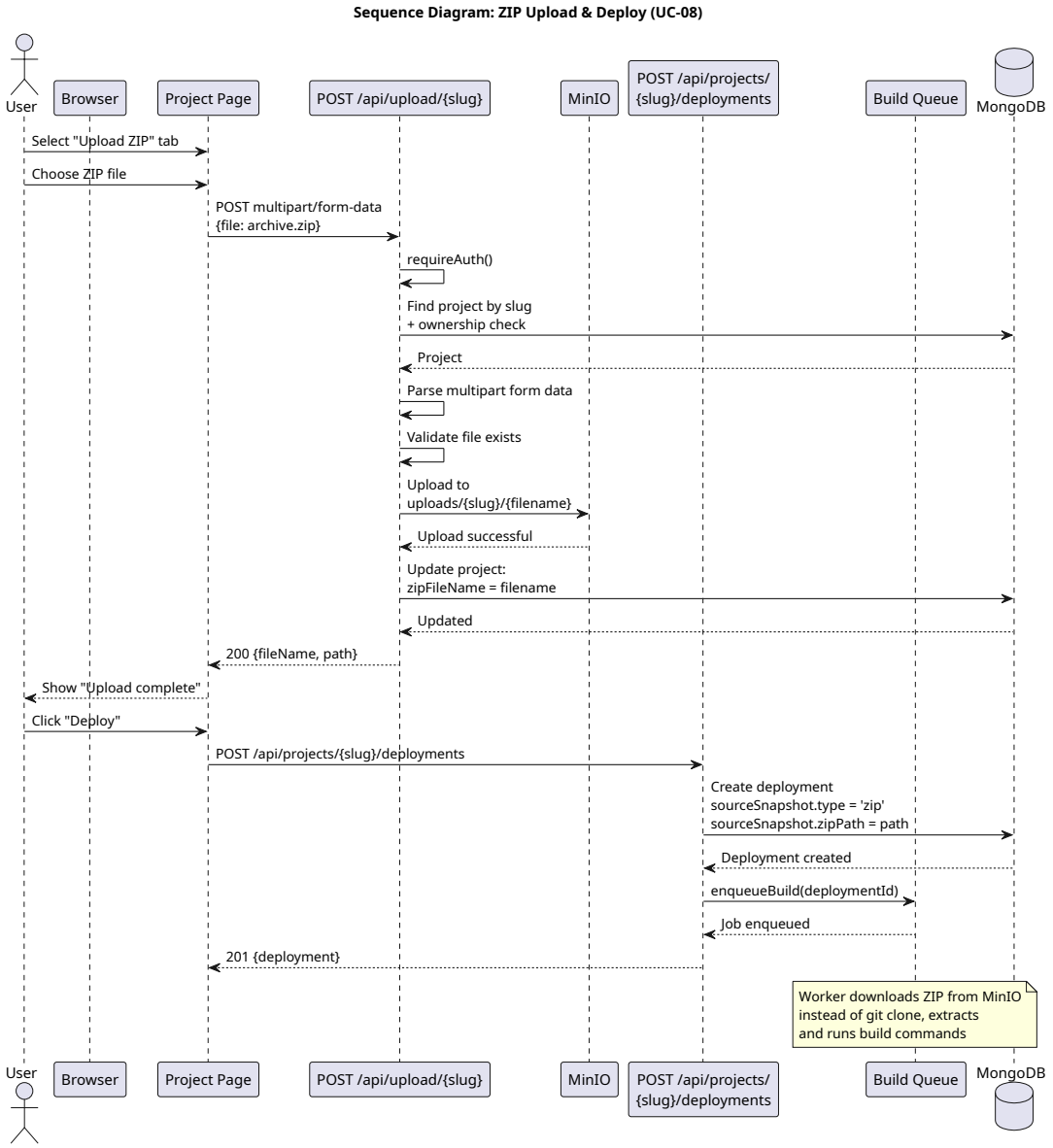

Figure 11: Sequence Diagram: ZIP Upload and Deploy (D-SQ-08)

SD-09: Environment Variable Management

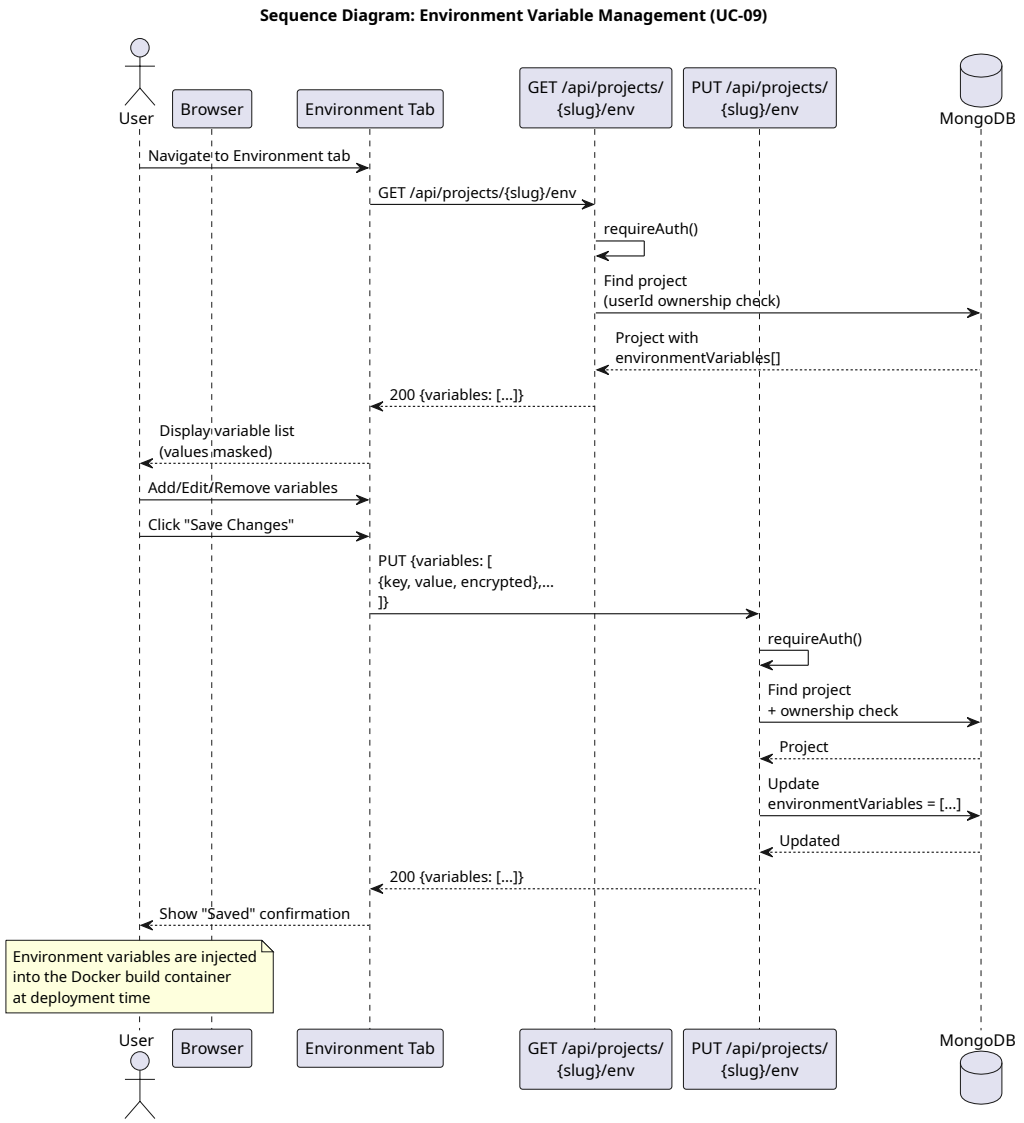

Figure 12: Sequence Diagram: Environment Variable Management (D-SQ-09)

SD-10: Authentication Middleware Flow

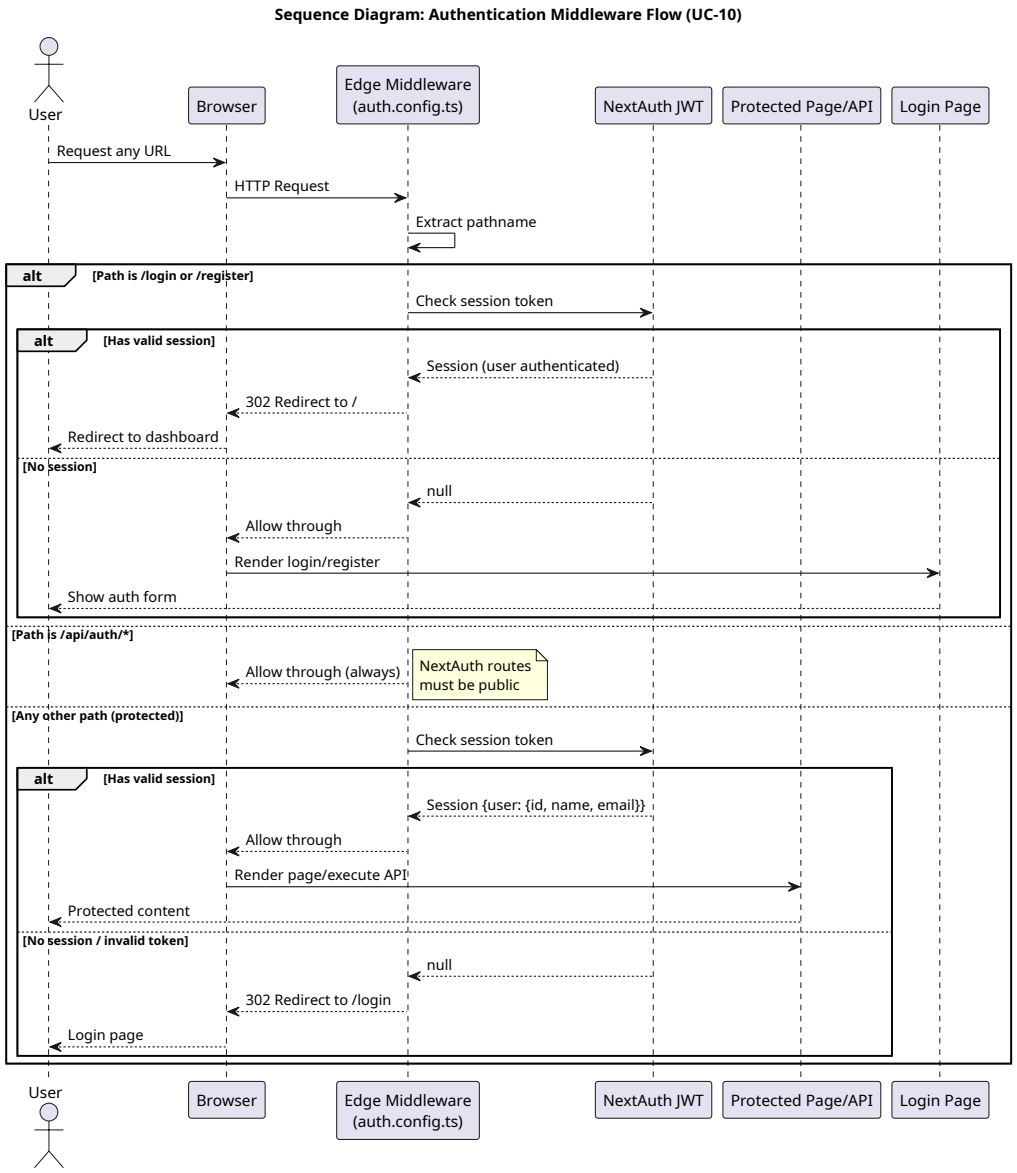

Figure 13: Sequence Diagram: Authentication Middleware Flow (D-SQ-10)

5.3 Database Schema

PageForge uses MongoDB as its primary database with Mongoose as the ODM. The schema consists of three collections with embedded sub-documents.

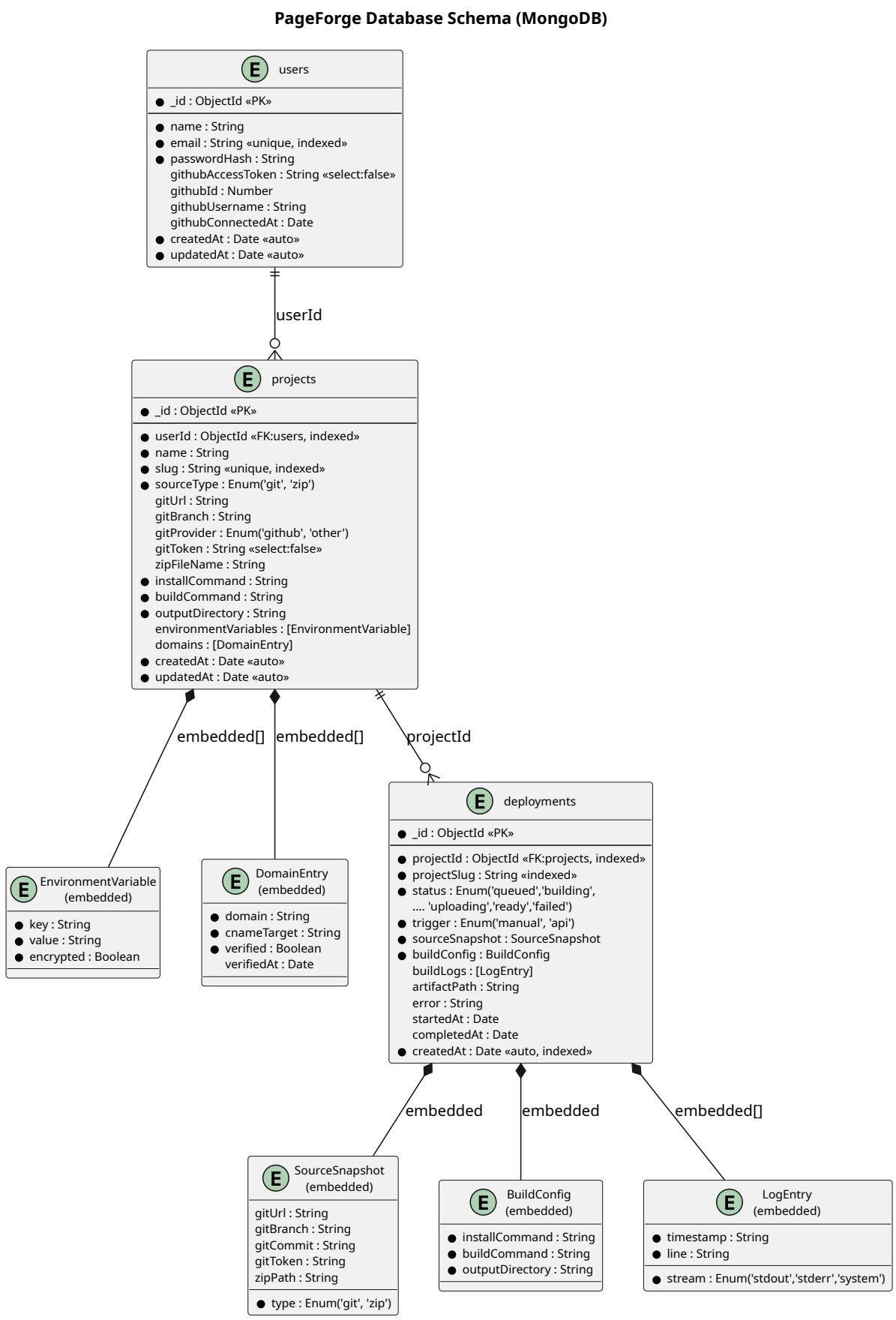

Figure 14: Database Entity-Relationship Diagram (D-DB-01)

5.3.1 Collection: `users`

Field	Type	Constraints	Description
_id	ObjectId	PK, auto	Primary key
name	String	required	User display name
email	String	required, unique, indexed	Login identifier (lowercase, trimmed)
passwordHash	String	required, select:false	bcrypt hash (12 rounds)
githubAccessToken	String	select:false	GitHub OAuth token
githubId	Number		GitHub user ID
githubUsername	String		GitHub login name
githubConnectedAt	Date		When GitHub was connected
createdAt	Date	auto	Mongoose timestamp
updatedAt	Date	auto	Mongoose timestamp

5.3.2 Collection: `projects`

Field	Type	Constraints	Description
_id	ObjectId	PK, auto	Primary key
userId	ObjectId	required, indexed, ref:User	Owner reference
name	String	required	Project display name
slug	String	required, unique, indexed	URL-safe identifier
sourceType	String	required, enum:[git,zip]	Deployment source
gitUrl	String		Repository URL
gitBranch	String	default: “main”	Branch to build
gitProvider	String	enum:[github,other]	Git hosting provider
gitToken	String	select:false	Per-project PAT
zipFileName	String		Uploaded ZIP filename
installCommand	String	default: “npm install”	Install step
buildCommand	String	default: “npm run build”	Build step
outputDirectory	String	default: “dist”	Build output path
environmentVariables	Array	embedded	Build-time env vars
domains	Array	embedded	Custom domain entries
createdAt	Date	auto	Mongoose timestamp
updatedAt	Date	auto	Mongoose timestamp

5.3.3 Collection: `deployments`

Field	Type	Constraints	Description
_id	ObjectId	PK, auto	Primary key
projectId	ObjectId	required, indexed, ref:Project	Parent project
projectSlug	String	required, indexed	Denormalized slug
status	String	required, enum	Build lifecycle state
trigger	String	required, enum:[manual,api]	How deployment started
sourceSnapshot	Object	required, embedded	Frozen source config
buildConfig	Object	required, embedded	Frozen build commands

Field	Type	Constraints	Description
buildLogs	Array	embedded	Log entries array
artifactPath	String		MinIO artifact location
error	String		Error message on failure
startedAt	Date		Build start time
completedAt	Date		Build end time
createdAt	Date	auto, indexed	Creation timestamp

5.3.4 Indexes

- `users.email` — unique index for login lookup.
- `projects.slug` — unique index for URL routing.
- `projects.userId` — index for user-scoped queries.
- `deployments.projectId` — index for project-scoped queries.
- `deployments.projectSlug` — index for slug-based lookups.
- `deployments.createdAt` — index for chronological sorting.

5.4 Security Design

5.4.1 Authentication and Authorization Mechanisms

Authentication Mechanisms (D-SEC-01)

1. **Password-Based Authentication:**
   - Passwords are hashed using bcrypt with 12 salt rounds.
   - Plaintext passwords are never stored or logged.
   - Authentication is handled by NextAuth.js v5 Credentials provider.
   - JWT tokens are issued with 30-day maximum age.
   - Tokens contain user ID, name, and email (no sensitive data).
2. **OTP-Based Verification:**
   - 6-digit cryptographically random OTP generated per request.
   - OTPs expire after 5 minutes.
   - Single-use: invalidated immediately after successful verification.
   - Delivered via email to the registered address.
   - Used for two purposes: (1) email verification during registration, (2) identity verification during password reset.
3. **JWT Session Management:**
   - Strategy: JWT (stateless, no server-side session store).
   - Token stored in HTTP-only secure cookie.
   - Maximum age: 30 days.
   - Token payload: {`id, name, email`}.

Authorization Mechanisms (D-SEC-02)

1. **Route Protection via Middleware:**
   - All routes except `/login`, `/register`, and `/api/auth/*` require authentication.
   - Middleware runs in Edge Runtime for performance.
   - Unauthenticated requests are redirected to `/login`.
   - Authenticated users are redirected away from auth pages.
2. **API-Level Ownership Checks:**
   - All 8 protected API route files call `requireAuth()` which extracts `userId` from the JWT.
   - All database queries include `userId` filter to enforce ownership scoping.
   - Users can only access their own projects, deployments, and settings.
3. **GitHub OAuth Security:**
   - OAuth state parameter contains `userId` to prevent CSRF.
   - Callback verifies `state` matches the authenticated user's ID.
   - Redirect URI uses `AUTH_URL` (not request origin) to prevent open redirect.
   - `trustHost: true` in NextAuth config for reverse proxy compatibility.

5.4.2 Security Protocols

SP-01: Secret Management

- **Password hashes:** Stored with Mongoose `select: false`; stripped in `toJSON`/`toObject` transforms.
- **GitHub access tokens:** Stored with `select: false`; never exposed in API responses; only a `hasGithubToken` boolean is returned.
- **Project PATs:** Stored with `select: false`; only `hasGitToken` boolean is exposed.
- **Auth secret:** `AUTH_SECRET` environment variable for JWT signing; never exposed.

SP-02: Build Container Isolation

- Builds execute in ephemeral Docker containers destroyed after each build.
- Containers have `no-new-privileges` security option.
- Resource limits: configurable memory (default 512MB) and CPU (default 1 core).
- Optional gVisor (`runsc`) runtime for additional kernel-level sandboxing.
- Git credentials passed via environment variable (not command-line arguments) to avoid `/proc` exposure.
- Git credential helper configured inside the container to use the token without embedding in URLs.

SP-03: Log Sanitization

- The `sanitize()` function replaces all known secrets in build log output with `[REDACTED]`.
- Secrets sanitized include: `GIT_AUTH_TOKEN`, project PAT, user OAuth token, and all environment variable values.
- Sanitization is applied before both Redis publishing and MongoDB persistence.

SP-04: Reverse Proxy Security

- Caddy handles TLS termination and HTTPS enforcement.
- `trustHost: true` in NextAuth config trusts `X-Forwarded-*` headers from the proxy.
- `AUTH_URL` is used as the canonical base URL for all redirect construction (prevents localhost leakage behind proxies).
- Cloudflare Tunnel (optional) provides DDoS protection and hides origin server IP.

SP-05: Data Access Control

- All API routes extract `userId` from the JWT and filter database queries accordingly.
- No cross-user data access is possible through the API.
- Deployment artifacts are stored in MinIO with deployment-specific paths.
- Caddy routes are configured per-project to serve only the correct artifacts.

6 Other Nonfunctional Requirements

6.1 Performance Requirements

NFR-01 Dashboard pages shall load promptly without noticeable delay on a standard broadband connection.

NFR-02 API responses for CRUD operations shall feel near-instantaneous to the user under normal load.

NFR-03 Build log streaming shall deliver output to the browser with minimal perceptible latency so the user experience feels real-time.

NFR-04 The system shall support concurrent builds limited by the host machine's Docker resource availability (CPU, memory, disk I/O).

NFR-05 Build start-up time shall be significantly faster than VM-based platforms (such as Cloudflare Pages) due to the use of lightweight Docker containers.

NFR-06 GitHub repository listing shall support pagination to remain responsive regardless of how many repositories the user has.

6.2 Safety Requirements

NFR-07 Build containers shall be resource-limited to prevent denial-of-service to the host system.

NFR-08 Failed builds shall not leave orphaned Docker containers (cleanup on both success and failure).

NFR-09 Build failures shall not affect other running builds or the main application.

6.3 Security Requirements

NFR-10 All passwords shall be hashed with bcrypt (minimum 12 rounds).

NFR-11 Sensitive fields shall use Mongoose `select: false` and be stripped from API responses.

NFR-12 Build containers shall run with `no-new-privileges` security option.

NFR-13 All secrets shall be sanitized from build logs before storage and streaming.

NFR-14 JWT tokens shall expire after a maximum of 30 days.

NFR-15 API routes shall enforce ownership checks (users can only access their own data).

NFR-16 GitHub OAuth shall use the `state` parameter to prevent CSRF attacks.

6.4 Software Quality Attributes

NFR-17 Maintainability: Monorepo with shared types ensures consistency across web app and worker.

NFR-18 Portability: Runs on any Linux server with Docker; no cloud-provider-specific dependencies.

NFR-19 Usability: Dark mode default, consistent loading/error/empty states on all pages, desktop-optimized layout. Mobile-responsive layout is deferred to a future version.

NFR-20 Reliability: BullMQ provides job retry and persistence; deployments survive worker restarts.

NFR-21 Extensibility: Git provider type supports "other" for future self-hosted Git integration.

6.5 Business Rules

BR-01 Open registration: any user can create an account.

BR-02 Projects are scoped per user: users can only see and manage their own projects.

BR-03 Token priority for private repos: project-level PAT takes precedence over user-level GitHub OAuth token.

BR-04 Only the latest successful deployment is actively served via Caddy (previous deployments remain in storage).

BR-05 Custom domains require DNS CNAME verification before activation.

7 Traceability

7.1 Requirement-to-Design Mapping

Req / UC ID	Req / UC Name	Design ID(s)	Design Artifact
UC-01	User Registration	D-CL-01, D-SQ-01, D-DB-01	Class Diagram, Sequence Diagram, Database Schema
UC-02	User Login (Credentials)	D-CL-01, D-SQ-02, D-SEC-01	Class Diagram, Sequence Diagram, Security Design
UC-03	Email OTP Verification	D-CL-01, D-SQ-03, D-SEC-01	Class Diagram, Sequence Diagram, Security Design
UC-04	GitHub OAuth Connection	D-CL-01, D-SQ-06, D-DB-01, D-SEC-02	Class Diagram, Sequence Diagram, Database Schema, Security Design
UC-05	Project Management	D-CL-01, D-SQ-04, D-DB-01	Class Diagram, Sequence Diagram, Database Schema
UC-06	Deployment Pipeline	D-CL-01, D-SQ-05, D-DB-01, D-SEC-02	Class Diagram, Sequence Diagram, Database Schema, Security Design
UC-07	Environment Variables	D-CL-01, D-SQ-09, D-DB-01	Class Diagram, Sequence Diagram, Database Schema
UC-08	Custom Domain Mgmt	D-CL-01, D-SQ-07, D-DB-01	Class Diagram, Sequence Diagram, Database Schema
UC-09	ZIP File Upload	D-CL-01, D-SQ-08, D-DB-01	Class Diagram, Sequence Diagram, Database Schema
UC-10	Build Log Viewer	D-CL-01, D-SQ-05, D-SQ-10	Class Diagram, Sequence Diagrams
UC-11	Forgot Password	D-CL-01, D-SQ-03, D-SEC-01	Class Diagram, Sequence Diagram, Security Design
FR-01 to FR-07	Registration Reqs	D-SQ-01, D-DB-01	Sequence Diagram, Database Schema
FR-08 to FR-14	Login (Credentials)	D-SQ-02, D-SEC-01	Sequence Diagram, Security Design
FR-15 to FR-22	Email OTP Verification Reqs	D-SQ-03, D-SEC-01	Sequence Diagram, Security Design
FR-23 to FR-30	GitHub OAuth Reqs	D-SQ-06, D-SEC-02	Sequence Diagram, Security Design
FR-31 to FR-43	Project Mgmt Reqs	D-SQ-04, D-DB-01	Sequence Diagram, Database Schema
FR-44 to FR-65	Deployment Reqs	D-SQ-05, D-DB-01, D-SEC-02	Sequence Diagram, Database Schema, Security Design

Req / UC ID	Req / UC Name	Design ID(s)	Design Artifact
FR-66 to FR-70	Env Variables Reqs	D-SQ-09, D-DB-01	Sequence Diagram, Database Schema
FR-71 to FR-77	Domain Mgmt Reqs	D-SQ-07, D-DB-01	Sequence Diagram, Database Schema
FR-78 to FR-81	ZIP Upload Reqs	D-SQ-08, D-DB-01	Sequence Diagram, Database Schema
FR-82 to FR-88	Log Viewer Reqs	D-SQ-05, D-SQ-10	Sequence Diagrams
FR-89 to FR-98	Forgot Password Reqs	D-SQ-03, D-SEC-01	Sequence Diagram, Security Design
NFR-10	Password Hashing	D-SEC-01	Security Design
NFR-11	Sensitive Field Protection	D-SEC-01, D-DB-01	Security Design, Database Schema
NFR-12	Container Security	D-SEC-02	Security Design
NFR-13	Log Sanitization	D-SEC-02	Security Design
NFR-14	JWT Expiry	D-SEC-01	Security Design
NFR-15	Ownership Checks	D-SEC-02, D-SQ-10	Security Design, Sequence Diagram
NFR-16	OAuth CSRF Prevention	D-SEC-02	Security Design

8 Other Requirements

- **Database:** MongoDB 7.0+ with Mongoose ODM. Connection pooling with cached connection pattern.
- **Internationalization:** Not in scope for v1.0. English only.
- **Legal:** Self-hosted; no data leaves the user's infrastructure (except GitHub API calls). Users are responsible for compliance with their own policies.
- **Reuse:** The `packages/shared` module is designed for reuse across web app and worker processes.
- **Deployment:** Infrastructure services are containerized via `docker-compose.yml`. Application processes (web, worker) run natively on the host.

Appendix

Glossary

BullMQ A Node.js library for Redis-based message queues with job scheduling and retry support.

Caddy A modern web server with automatic HTTPS and a JSON-based admin API for dynamic configuration.

CNAME A DNS record type that maps one domain name to another (Canonical Name).

CSRF Cross-Site Request Forgery — an attack where unauthorized commands are transmitted from a trusted user.

Docker A platform for building, shipping, and running applications in isolated containers.

gVisor A container sandbox runtime by Google that provides additional kernel-level isolation.

JWT JSON Web Token — a compact, URL-safe means of representing claims to be transferred between two parties.

MinIO An S3-compatible object storage server designed for high-performance workloads.

Mongoose An Object Data Modeling (ODM) library for MongoDB in Node.js.

NextAuth.js An authentication library for Next.js applications (v5 also known as Auth.js).

OAuth An open standard for access delegation, commonly used to grant websites access to user information without exposing passwords.

OTP One-Time Password — a password valid for only one login session or transaction.

PAT Personal Access Token — a token used as an alternative to passwords for Git authentication.

Pub/Sub Publish/Subscribe messaging pattern where senders publish messages to channels without knowledge of subscribers.

REST Representational State Transfer — an architectural style for designing networked applications.

SPA Single-Page Application — a web application that interacts with the user by dynamically rewriting the current page.

WebSocket A communication protocol providing full-duplex communication channels over a single TCP connection.