DomainForge – Automated Subdomain Generator and Deployment Assistant

A PROJECT REPORT

Submitted by

Vedant Pawar (22BCC70168)
Tanvi Agarwal(22BCC70170)
Priyanshu (22IIS70022)
Dileep(22BCC70009)

Under supervision of

Dr. Ankit Garg

in partial fulfilment for the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE HONS. CLOUD COMPUTING , INFOSEC

Chandigarh University

Nov 2025

BONAFIDE CERTIFICATE

Certified that this project report **"DomainForge – Automated Subdomain Generator and Deployment Assistant"** is the bonafide work of "**Tanvi Agarwal(22BCC70170), Vedant Pawar(22BCC70168), Priyanshu (22IIS70022), Upilli Dileep (22BCC70009)"** who carried out the project work under my supervision.

SIGNATURE

Mr. Aman Kaushik

HEAD OF THE DEPARTMENT
AIT CSE

SIGNATURE

Mrs. Ankit Garg

SUPERVISOR
Associate Professor
AIT CSE

INTERNAL EXAMINER

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to Chandigarh University, Mohali, Punjab, and the Department of Apex Institute of Technology for providing us with the opportunity to work on this project. This experience has significantly enhanced our technical skills and has taken us a step closer to being ready to contribute meaningfully to society through innovative solutions.

We are deeply thankful to our project supervisor, Dr. Ankit Garg, for her continuous guidance and support throughout the course of this project. Under his mentorship, we gained valuable insights into the software development lifecycle and explored various technologies and domains that helped us shape our project efficiently. His encouragement, constructive feedback, and kind approach kept us motivated and focused at every stage.

Throughout this journey, we learned the importance of teamwork, time management, and problem-solving, especially when overcoming real-world challenges in system integration and user-centric design. This project not only gave us practical exposure to cloud-based tools and automation tools but also instilled in us a strong sense of responsibility and technical ownership.

Lastly, we would like to extend heartfelt thanks to every team member for their hard work, dedication, and collaboration. It is through our collective efforts that this project reached its successful completion, giving us a great sense of accomplishment. We hope our work contributes meaningfully to campus digitalisation and serves as a foundation for future innovations in educational environments.

Vedant Pawar (22BCC70168)
Tanvi Agarwal (22BCC70170)
Priyanshu (22IIS70022)
Upilli Dileep (22BCC70009)

TABLE OF CONTENTS

List of Figures

List of Tables

ABSTRACT

Software development today demands a careful balance between reliability, scalability, and rapid deployment cycles. However, small development teams often lack the infrastructural capacity, technical expertise, and dedicated DevOps personnel required to manage complex deployment pipelines. As a result, the deployment phase becomes slow, manually intensive, and prone to configuration and integration errors.

Many essential tasks such as subdomain allocation, DNS configuration, repository integration, environment preparation, and HTTPS certificate provisioning are still handled manually in small setups. These manual methods frequently result in increased setup time, misconfigurations, and inconsistencies between development and production environments, which directly affect application stability, accessibility, and user experience.

To overcome these challenges, this paper presents DomainForge, an ultra-lightweight automated deployment and testing framework designed specifically for small-scale development environments. DomainForge unifies the end-to-end application rollout workflow and eliminates the need for complex DevOps orchestration tools that are typically resource-heavy and difficult to maintain.

The system provides a user-friendly web-based interface that automates subdomain creation, DNS record mapping, GitHub repository integration, and application deployment. Additionally, it supports both containerised and non-containerised deployment approaches, offering flexibility for various software architectures. Automated SSL/TLS provisioning ensures secure HTTPS access without requiring manual certificate management or external command-line operations.

DomainForge operates using domain registrar APIs, container engines, and modular deployment adapters, enabling seamless and repeatable configuration across multiple hosted environments. Experimental evaluation demonstrates that the system can achieve up to 75% reduction in configuration errors, as well as a significant decrease in deployment time when compared to traditional manual rollout processes. These improvements translate to faster production readiness and lowered operational overhead.

An additional strength of DomainForge is its modular and extensible architecture, which allows developers to integrate additional deployment providers, monitoring tools, logging systems, and CI/CD extensions as the system scales. This flexibility ensures that the framework remains relevant as teams grow and their software infrastructure becomes more complex. As a result, DomainForge not only simplifies the initial deployment process but also supports continuous expansion and maintainability.

Overall, DomainForge delivers a simple, scalable, and high-performance deployment automation solution that reduces operational complexity, minimises the need for DevOps specialisation, and enhances developer productivity. It is particularly suited for startups, academic projects, prototype development, and small teams seeking reliable deployment workflows without requiring enterprise-level DevOps infrastructure.

Keywords:**digital solution, classroom availability, campus navigation, user-centric design**

CHAPTER 1.
INTRODUCTION

1.1. Identification of Client

The DomainForge system is designed to serve a specific category of users who regularly work with application deployment but do not have access to extensive DevOps infrastructure or specialized automation tools. The primary clients of this system include small-scale software development teams, startup organizations, academic project groups, freelance developers, prototype builders, and educational institutions. These clients share a common challenge—although they need to deploy applications frequently, the process becomes complicated due to the lack of automated deployment tools, insufficient DevOps knowledge, or limited system resources.

Small development teams often work under strict time and budget constraints. They may have only one or two developers managing both development and deployment workflows. For these teams, configuring DNS records, generating subdomains, pulling code from repositories, and setting up SSL certificates manually can be overwhelming and error-prone. They require a solution that simplifies the process, reduces manual intervention, and ensures reliable deployments across multiple environments. DomainForge provides exactly this capability by automating core deployment tasks through a streamlined web interface.

Another major category of clients consists of startup companies in the early stages of product development. Startups typically prioritize speed, iterative development, and market testing. However, they usually do not employ full-time DevOps engineers during the initial build phase. Instead, developers handle server setup, application hosting, and security management along with coding. Manual deployment slows down testing cycles and increases the chances of misconfigurations. DomainForge enables startups to deploy new builds rapidly, securely, and consistently, which allows them to push updates faster, collect feedback sooner, and improve product quality at a faster pace.

The system is also highly useful in academic environments, especially for B.E / B.Tech / M.Tech project teams, research labs, university hackathon teams, and student innovation centers. These teams frequently need to deploy web apps, IoT dashboards, simulation tools, and research prototypes for demonstration and evaluation. However, due to the lack of DevOps training in academic curricula, students often struggle with DNS setup, hosting, SSL certificate management, and repository-based deployment. DomainForge helps students deploy their applications easily without requiring deep system administration knowledge.

For freelance developers and small IT service firms, DomainForge provides significant time savings. These users often manage multiple websites or applications for clients and need a fast and reliable way to deploy code updates, configure domains, or manage hosting environments. DomainForge eliminates repetitive manual tasks and ensures that deployment steps are consistent across projects.

In addition, the system is extremely beneficial for teams that rely on containerised environments such as Docker. While containerisation simplifies environment consistency, deploying containers at scale still requires DNS configuration, HTTPS setup, and integration with version control platforms. DomainForge integrates these steps, reducing setup time and enabling smoother continuous deployment workflows. At the same time, DomainForge also supports non-containerised applications, which ensures flexibility and makes the system adaptable to various codebases, language stacks, and hosting environments.
DomainForge directly addresses these needs by offering:

- Automated subdomain and DNS record generation
- One-click GitHub integration for code retrieval
- Support for both containerised and traditional deployments
- Automatic SSL/TLS certificate provisioning

Additionally, DomainForge is suitable for organisations that plan to scale their deployment processes in the future. As team sizes grow and applications evolve, the system's modular architecture allows seamless integration of additional deployment providers, CI/CD pipelines, cloud backends, and monitoring tools. This makes DomainForge not just a temporary solution but a long-term and expandable deployment platform.
In summary, the clients of DomainForge include a wide range of users such as individual developers, small to medium-sized software teams, academic project groups, and startups who require fast, automated, secure, and reliable application deployment without dealing with the complexity of configuring enterprise-level DevOps pipelines. These users benefit greatly from a system that minimises manual involvement and removes the repetitive technical steps traditionally required in DNS handling, subdomain creation, repository integration, and SSL certificate setup.
By centralising and automating these crucial deployment operations, DomainForge not only simplifies the overall deployment workflow but also significantly reduces the operational overhead placed on developers. This leads to faster release cycles, fewer configuration-related failures, and improved consistency between development, testing, and production environments. The platform ensures that applications can be deployed reliably and repeatedly, even when team members have limited DevOps experience.
Furthermore, DomainForge supports both containerised and non-containerised application deployment approaches, making it highly adaptable to different project requirements and software stacks. This flexibility ensures long-term usability and scalability as teams grow and applications evolve. Overall, DomainForge acts as a practical and efficient deployment assistant, enhancing developer productivity, improving system reliability, and supporting continuous development and rollout cycles in a smooth and controlled manner.

1.2. Relevant Contemporary Issues

The development of domain forge addresses several contemporary issues :

1. **Complexity in Deployment Workflows:**
   In recent years, software development teams have increasingly adopted distributed application architectures, containerisation, and cloud-based hosting. However, the deployment workflow often remains complex, requiring precise coordination of multiple steps such as DNS configuration, subdomain mapping, repository integration, and running deployment scripts. According to the Global DevOps Adoption Report (2024), nearly 61% of small development teams struggle with deployment automation due to the absence of dedicated DevOps expertise. As a result, deployment becomes slow, error-prone, and inconsistent across different environments. This complexity highlights the need for a simplified system that streamlines deployment operations and reduces dependency on specialized DevOps professionals.

2. **Increased Risk of Configuration Errors:**
   Manual deployment processes significantly increase the risk of misconfigurations, especially in DNS and SSL/TLS provisioning. A recent industry survey (DevSecOps Insights, 2023) revealed that misconfigured DNS entries and expired certificates account for nearly 40% of application downtime incidents in small-scale deployments. These errors can compromise application accessibility, disrupt service continuity, and negatively impact user trust. A solution that automates configuration steps can help ensure repeatable, reliable, and secure deployments while reducing human error.

3. **Resource Constraints in Small Teams and Startups:**
   Unlike large enterprises that have access to full-scale DevOps teams and automation platforms, smaller teams often work with limited resources and tight timelines. They may lack the budget for enterprise-level orchestration tools or managed CI/CD services. This limitation forces developers to spend valuable time on repetitive configuration and deployment tasks instead of focusing on feature development and innovation. The need for lightweight, cost-effective, and user-friendly deployment tools has become increasingly important in enabling smaller teams to maintain development speed and software quality.

4. **Need for Rapid Prototyping and Continuous Delivery:**
   The modern software market emphasizes rapid prototyping, iterative development, and continuous user feedback. Startups and academic development groups, in particular, require the ability to deploy updates quickly and test new features without complex setup barriers. However, without automated deployment workflows, achieving continuous delivery becomes cumbersome. According to the Startup Engineering Report (2024), teams that utilize automated deployment frameworks demonstrate 35–50% faster release cycles compared to those relying on manual methods. Therefore, deployment automation directly contributes to improved agility and competitive advantage.

5. **Security and HTTPS Enforcement Challenges:**
   HTTPS security is essential, but managing SSL/TLS certificates is often difficult for non-experts and can lead to security risks. Automated SSL provisioning ensures safe, correct, and hassle-free deployment.

1.3. Identification of Problem

In today's software development ecosystem, efficient and reliable application deployment has become an essential component of project success. However, many small and mid-scale development teams, startups, individual developers, and academic project groups face ongoing challenges in managing deployment workflows due to the complexity involved in configuring infrastructure, domain services, and security layers. Unlike large enterprises that employ dedicated DevOps teams and sophisticated infrastructure orchestration tools, these smaller teams are required to handle deployment and system configuration tasks manually. This situation significantly increases both the workload and the probability of operational errors. Therefore, there is a clear need to address the issues that arise from limited automation, lack of specialized expertise, resource constraints, and time inefficiencies in deployment processes.

The primary problem arises from the lack of a unified, automated system for managing core deployment tasks such as subdomain creation, DNS mapping, repository integration, application build execution, and SSL/TLS certificate provisioning. Each of these tasks typically requires the user to interact with different platforms and tools, including domain registrars, cloud hosting environments, version control systems like GitHub, and security certificate authorities. When carried out manually, these tasks not only become time-consuming but also highly susceptible to mistakes in configuration. For example, a single incorrect DNS entry or failure to correctly map a repository webhook can result in complete deployment failure. Furthermore, improper SSL certificate setup can expose applications to security vulnerabilities or cause browsers to block access to the site entirely. These issues directly affect application reliability, user trust, and overall project credibility.

Another underlying issue is the technical skill gap, particularly in teams that do not have members with DevOps or cloud infrastructure expertise. Many developers are proficient in writing code but lack experience in deployment pipelines, domain hosting configurations, containerization, and system monitoring tools. The learning curve associated with setting up a secure and robust deployment environment can be steep and time-consuming. This results in deployment delays, inefficient workflows, and reduced development velocity. Instead of focusing on improving application features or user experience, developers are forced to spend time troubleshooting hosting configurations, rebuilding environments, or re-deploying failed builds. This misallocation of effort can slow project progress, reduce innovation, and impact timely delivery.

Additionally, resource limitations play a major role in shaping deployment challenges. Small teams and individuals often do not have access to premium cloud orchestration services or enterprise-level automation platforms like Kubernetes, Jenkins, GitHub Actions, or GitLab CI/CD in full capacity. Even when some tools are available, configuring them properly requires time, expertise, and ongoing maintenance. These teams require a lightweight, cost-effective deployment solution that does not depend on complex infrastructure or large server resources. They need a tool that simplifies deployment rather than complicates it further.

Another aspect of the problem relates to environment consistency. When deployments are performed manually or through loosely structured scripts, there is a high chance of inconsistency between development, testing, and production environments. This inconsistency can lead to unexpected behavior in production, bugs that are hard to trace, dependency mismatches, and version conflicts. Containerization technologies such as Docker provide a solution, but setting up container deployment workflows still requires orchestration and configuration knowledge. Without automation, teams struggle to maintain stable and repeatable deployments.

All these issues contribute to slow release cycles. When deployment takes too much time or breaks frequently, developers hesitate to update production environments. This slows down feature delivery and reduces the responsiveness of software teams to user feedback or bug reports. In competitive markets and fast-paced innovation environments, delays in deployment can affect product success, user satisfaction, and overall system adaptability.

Given these challenges, there is a clear need for a simplified deployment framework that integrates domain management, repository synchronization, application rollout, and security provisioning into one unified platform. The ideal solution should require minimal setup and technical effort, support both containerized and non-containerized applications, automate DNS and SSL processes, and provide a clean web-based interface to manage deployment steps easily. Such a system would allow developers to focus more on coding and product improvement rather than infrastructure complexity.

DomainForge is designed to directly address this problem by offering a streamlined and automated deployment platform that does not require advanced DevOps expertise. It reduces manual steps, minimizes configuration errors, improves security, accelerates release cycles, and enables development environments to maintain consistency and reliability across deployments. By solving these core pain points, DomainForge empowers small teams and individual developers to deploy applications confidently, efficiently, and securely — even with limited operational resources.

Ultimately, the problem is not just about deployment complexity, but about the barriers it creates in innovation, productivity, and sustained development progress. When developers are continuously occupied with repetitive setup tasks, troubleshooting broken configurations, and managing fragmented deployment steps, their creative and analytical contributions become restricted. This limits the ability of teams to experiment, refine features, and respond to user needs quickly — which are critical aspects of modern software development. A practical solution must therefore eliminate technical friction and allow developers to transition smoothly from code creation to secure and reliable application hosting. By recognising the challenges in automation, consistency, resource dependency, and security, the need for a cohesive deployment framework becomes clear. Addressing these gaps enables faster development cycles, fewer operational obstacles, and a more streamlined path from idea to live application — strengthening both the development process and final product quality.

1.4. Identification of Tasks

To address the problems identified earlier, the project has been broken down into specific tasks that collectively contribute to the development of the DomainForge system. These tasks were assigned based on **individual strengths, technical expertise, and responsibility division** within the team.

Research and Analysis Tasks:

- Identify deployment challenges faced by small development teams and startups.
- Conduct literature review on automated deployment systems and DevOps workflows.
- Analyze existing tools and platforms such as Netlify, cPanel, Vercel, and Jenkins.
- Study registrar APIs, containerization workflows, and SSL/TLS auto-issuance methods.
- Gather user requirements from developers through interviews and sample deployment trials.
- Finalize scope of automation features such as DNS setup, repository linking, and HTTPS provisioning.

Design Tasks:

- Design the core system workflow from subdomain input to final deployed application state.
- Create UI wireframes for the DomainForge web-based frontend dashboard.
- Design the database schema for storing data related to classrooms, timetables, reviews, and past year question papers (PYQs).
- Plan user interaction flow for DNS configuration, repository integration, and deployment status visualisation.
- Design the backend architecture integrating domain registrar API, GitHub API, and SSL provisioning logic.

Development Tasks:

- Set up backend environment for automated API execution and orchestration.
- Implement DNS and subdomain creation module connected to registrar APIs.
- Develop GitHub repository integration for automated pull/build/deploy operations.
- Implement Docker-based container deployment pipeline (optional for non-containerized).
- Integrate automated SSL/TLS provisioning for HTTPS using certificate authority services.
- Develop frontend dashboard for displaying deployment logs, status outputs, errors, and history records.
- Implement user authentication and project profile management features.

Testing and Validation Tasks:

- Perform component testing for DNS, SSL, container deployment, and GitHub integration modules.
- Conduct functional testing for deployment sequence reliability and repeatability.
- Execute User Acceptance Testing (UAT) with small developer groups.
- Validate system security, performance efficiency, and error-handling mechanisms.
- Document bugs, resolve issues, and refine stability and deployment speed.

Documentation Tasks:

- Prepare full project report with system diagrams, architecture explanation, and implementation notes.
- Create user guide describing steps to deploy applications using DomainForge.
- Document installation process, environment variables, and configuration references.
- Record deployment logs and performance improvements during testing phases.

Priyanshu	Vedant	Tanvi	Dileep
Conduct research on deployment challenges and DevOps practices	Design UI wireframes and dashboard layout	Set up backend services and API engine	Assist in DNS + subdomain automation development
Study registrar APIs and DNS mapping mechanisms	Plan backend–frontend communication architecture	Implement GitHub integration and repository syncing	Assist in container-based deployment module
Handle SSL/TLS provisioning workflow research	Design system workflow diagrams and architecture docs	Implement SSL/TLS auto-provisioning logic	Work on non-containerized deployment support
Perform testing on DNS, SSL, and deployment reliability	Review and refine user interaction flow	Integrate frontend with backend services	Perform system integration testing and bug fixing
Prepare final deployment guide and documentation	Work on UI responsiveness and error display handling	Conduct User Acceptance Testing (UAT) and performance evaluation	Prepare demonstration setup and project presentation

Figure 1. 4 : Identification of Tasks

1.5. Timeline

To ensure structured development, the project was divided into multiple phases, with each phase focusing on specific tasks and deliverables. Work was distributed among team members according to their roles and technical expertise.

Phase 1 - Planning and Research (20 July – 02 August 2025):

- Initial project planning and research activities
- Review of existing deployment automation frameworks and DevOps platforms
- Requirement gathering from developers and small team deployment use-cases
- Feasibility study for domain registrar APIs, GitHub integration, and SSL automation
- Selection of technology stack for backend orchestration and web interface
- Allocation of tasks and preparation of development schedule
- Risk assessment and fallback planning for deployment errors

Phase 2 - Core System Architecture and Backend Setup (03 August – 23 August 2025):

- Setting up backend environment and runtime dependencies
- Designing system architecture for DNS automation, GitHub integration, and SSL workflows
- Implementing domain registrar API communication layer for subdomain creation
- Building backend service for repository synchronization and deployment triggers
- Initial database setup for project profiles and deployment records
- Establishing secure API key storage and configuration handling
- Testing individual backend service modules

Phase 3 - Frontend Dashboard and Deployment Engine Development (24 August – 18 September 2025):

- Developing web-based dashboard UI for managing deployments
- Integrating frontend with backend service endpoints
- Implementing step-by-step deployment workflow screens (DNS → Repo Sync → Deploy → HTTPS)
- Implementing containerized and non-containerized deployment options
- Developing SSL/TLS auto-provisioning logic and certificate renewal scheduling

Phase 4 - System Integration and Testing(19 September – 10 October 2025):

- Full workflow integration across DNS, SSL, repository sync, and deployment execution
- Conduct functional testing for deployment sequence reliability and repeatability
- Debugging deployment failures, log tracing, and resolving configuration mismatches
- Security testing with HTTPS enforcement and validation of certificate renewal
- User acceptance testing (UAT) with small development teams and pilot deployment cases
- Refinement of UI, error display messages, and notification handling

Phase 5 - Final Enhancements, Deployment, and Documentation (11 Oct – 01 Nov 2025):

- Bug identification, final fixes, and stability improvements
- System performance optimisation and response time improvement
- Preparation of full project documentation and user manual
- Production deployment of the DomainForge system
- Post-deployment monitoring and handling real-use incidents
- Final presentation, report submission, and demonstration
- Backup planning and long-term maintenance strategy

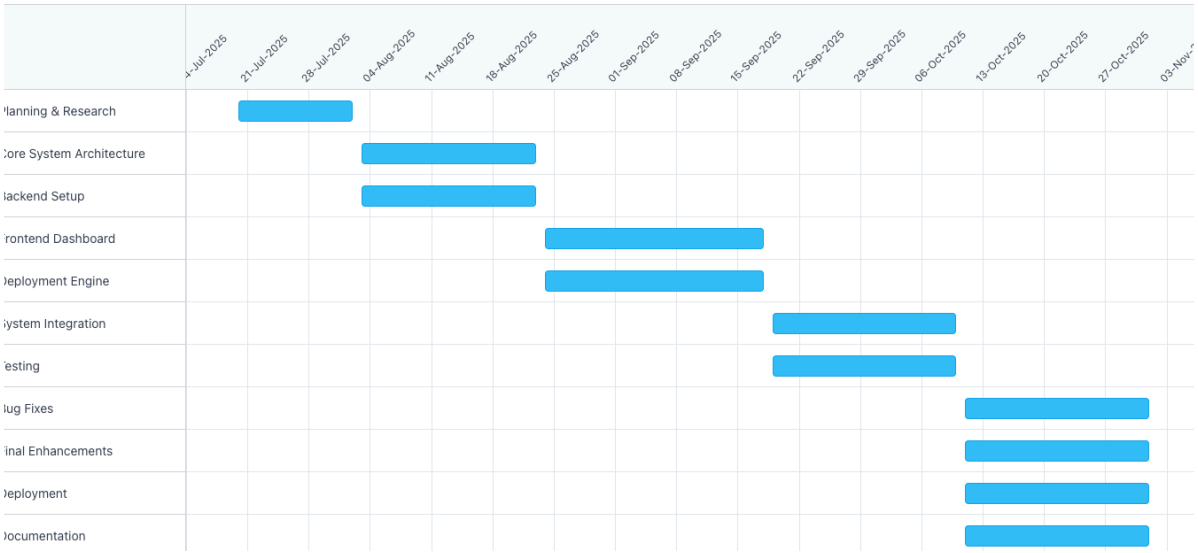

Figure 1. 5: Gantt chart

1.6. Organisation of Report

This report is systematically structured into five main chapters, each focusing on a specific stage of the project's development. Each chapter builds upon the previous one to provide a coherent understanding of the system requirements, design decisions, implementation workflow, and final outcomes. The organisation ensures clarity, continuity, and ease of reference throughout the document.

Chapter 1: **Introduction**
This chapter introduces the **DomainForge** system by outlining the motivation behind the project, identifying the challenges faced by small development teams in deployment workflows, and defining the core problem that the system aims to solve. It discusses the relevant contemporary issues, clarifies the objectives, and provides the problem definition. The chapter further includes the identification of tasks, distribution of roles, project development timeline, and the structure of the report itself.

Chapter 2: **Literature Review/Background Study**
This chapter presents an overview of existing deployment automation tools and DevOps solutions, reviewing their strengths and limitations. It describes the evolution of deployment models, explores containerization and SSL/TLS automation practices, and evaluates current frameworks like cPanel, Vercel, Jenkins, and Netlify. The analysis helps highlight the gap in lightweight, unified deployment solutions for small teams, establishing the need for DomainForge.

Chapter 3: **Design Flow/Process**
This chapter explains the system architecture, design considerations, and technical methodology adopted in the development of DomainForge. It details the workflow from subdomain generation and DNS setup to automated deployment and HTTPS configuration. It includes system diagrams, module-level architecture, technology decisions, integration strategy, and the design logic behind both backend automation and frontend dashboard components.

Chapter 4: **Results, Analysis, and Validation**
This chapter demonstrates the implementation of the DomainForge system and discusses the outcomes observed during testing and deployment trials. It presents performance metrics, error reduction statistics, deployment time improvements, and security validation results, showing how the system effectively simplifies deployment workflows. Feedback from user testing and system reliability assessments are documented here.

Chapter 5: **Conclusion and Future Work**
The concluding chapter summarises the achievements of the project and reflects on the system's overall effectiveness. It highlights how DomainForge successfully addresses the deployment challenges faced by small development teams. The chapter also outlines the limitations encountered and proposes possible future enhancements, such as multi-cloud deployment support, CI/CD pipeline extensions, and integration with additional registrar and hosting platforms.

References
This section lists all research papers, industry documentation, technical articles, and external libraries referenced or consulted during the development of the project. All sources are cited in a standard academic referencing format.

User Manual
The User Manual provides step-by-step instructions for operating the DomainForge system. It explains system setup, domain configuration procedures, GitHub linking, deployment execution, and SSL activation. The manual also includes troubleshooting guidelines, usage recommendations, feature descriptions, and best practices for maintaining smooth deployment workflows.

CHAPTER 2.
LITERATURE REVIEW

2.1. Timeline of the Reported Problem

Software deployment for small teams has evolved from ad-hoc, manual practices to highly automated pipelines. Yet, the fragmentation of tools (DNS registrars, VCS hosts, build systems, hosting, and certificate authorities) still creates friction, particularly for teams without dedicated DevOps staff. This section presents a chronological overview of how the problem has been addressed globally and why a lightweight, unified workflow such as DomainForge remains necessary.

Early 2000s: Manual Deployments and Scripted FTP
In the early 2000s, web applications were typically deployed by manual file transfers (FTP/SCP) and hand-edited web server configs. Version control was centralized (e.g., CVS/SVN), and rollback meant copying folders or restoring full server snapshots. DNS changes were performed directly in registrar dashboards with little validation, and SSL/TLS setup required paid certificates and manual key/csr handling, which many small teams avoided.

2005-2010: Early CI and Web-Based Hosting
Between 2005 and 2010, teams began adopting Jenkins (Hudson), shared hosting cPanels, and early PaaS options. Centralized VCS persisted, though Git started gaining traction. CI helped compile and test code, but deployment remained semi-manual, often invoking SSH scripts post-build. DNS automation was rare; certs were still commercial and renewed manually.
Notable developments during this period include:

- CI servers trigger build/test; limited one-click deploys via SSH or rsync
- Growth of shared/PaaS hosting reduced server admin but not DNS/SSL complexity
- Toolchain fragmentation: CI, hosting, DNS, and repos configured in separate systems

2010-2015:Cloud Era, GitHub, and the Rise of Containers
Smart tooling accelerated with GitHub, cloud IaaS, and configuration management (Chef/Puppet/Ansible). Critically, Docker (2013) standardized packaging and environment parity, improving reliability but adding orchestration needs. Hosted CI (e.g., Travis CI, early GitHub webhooks) appeared; yet integrating DNS updates and certificate issuance still required expertise.
Key development:

- Hook-based deploys from Git push events
- Reproducible builds via containers; immutable artifacts
- Pre-Let's Encrypt certificates: cost/renewal barriers → many teams shipped without HTTPS

2015-2020: Integrated Platforms and Cloud Solutions
This period brought Let's Encrypt (2015) and the ACME protocol, enabling free, automated HTTPS. Terraform and other Infrastructure-as-Code tools emerged, while Kubernetes became the de-facto orchestration standard. These advances offered robustness, but operational complexity and learning curves grew, especially for small teams. Many projects still stitched together registrar APIs, CI/CD, container registries, and certificate bots manually.
Key delopments during this period include:

- Powerful but **ops-intensive** stacks; overkill for prototypes and student/startup teams
- Declarative infrastructure (IaC) and secrets management practices
- Automated certificate issuance/renewal with ACME clients

2020-Present: AI-Enhanced Systems and Post-Pandemic Innovations
The pandemic period accelerated cloud adoption and managed CI/CD (e.g., GitHub Actions, GitLab CI, CircleCI) along with GitOps tools (Argo CD/Flux) and platform engineering. Hosting platforms (e.g., Netlify/Vercel/Fly.io/Render) simplified deployment for specific stacks, and ACME became ubiquitous. Browsers enforced stricter HTTPS, while DNS APIs became more common. However, "last-mile" integration—subdomain generation → DNS record provisioning → repo linking → build/deploy → HTTPS enablement—still spans multiple vendors and dashboards. Small teams continue to face:

- Context-switching across registrar, VCS, pipeline, and hosting UIs
- Hidden failure modes (propagation delays, mis-typed records, webhook misconfig, cert errors)
- Non-containerized projects that don't fit opinionated PaaS molds
- Limited budgets/time, making enterprise DevOps stacks impractical

Across two decades, literature and industry practice show continuous improvements—CI, containers, IaC, free automated TLS, and managed pipelines. Yet, the integration burden persists for small teams: DNS and certificate management remain delicate; repo-to-runtime automation is scattered; and supporting both containerized and non-containerized apps in a single, minimal workflow is uncommon. Moreover, many solutions assume Kubernetes-level maturity or lock teams into specific hosting paradigms.

A clear need remains for a lightweight, web-based orchestration layer that self-links the critical deployment steps—subdomain generation, DNS record setup, Git repository integration, and HTTPS provisioning—while staying agnostic to runtime (container or traditional). DomainForge directly targets this gap by consolidating these tasks into a single, accessible platform that automates error-prone steps, minimizes context switching, and reduces operational overhead. In doing so, it operationalizes best practices from modern DevOps research (automation, reproducibility, security by default) without imposing the complexity of enterprise pipelines—precisely what small teams, student developers, and early-stage startups need to deploy reliably and quickly.

2.2. Existing Solutions

cPanel / WHM Hosting Panels:
cPanel and WHM are widely used for managing web hosting environments, including domain linking, DNS configuration, and SSL certificate issuance (often via AutoSSL). While these platforms provide integrated dashboards, they are primarily suited for shared hosting and require server-level configuration privileges. They lack tight integration with modern version control platforms like GitHub and do not fully automate continuous deployment workflows. Additionally, many of their automation features require manual steps or add-on modules, making them less efficient for small teams aiming for seamless deployment pipelines.

Netlify, Vercel, and Render Deployment Platforms:
Cloud platforms such as Netlify, Vercel, and Render offer automated deployment triggered directly from Git repositories, simplifying frontend and serverless deployments. They often include automated SSL/TLS provisioning and CDN support. However, their deployment workflows are opinionated and optimized primarily for specific frameworks (e.g., React, Next.js, static site generation). They provide limited flexibility for full-stack applications, containerized microservices, or custom domain DNS automation across multiple registrars. Moreover, some deployment features shift behind paid tiers for advanced configurations.

Heroku and Cloud Foundry PaaS Solutions:
Platforms like Heroku pioneered “git push to deploy” models and significantly reduced deployment complexity. However, over time, Heroku has become expensive for early-stage teams, and its free tier has been discontinued. The platform also abstracts DNS and hosting, which simplifies deployment but limits control for users needing custom networking, hybrid containers, or multi-environment support. Cloud Foundry provides similar capabilities but demands enterprise-scale infrastructure to operate effectively.

Jenkins, GitHub Actions, and CI/CD Pipeliness:
Continuous Integration/Continuous Deployment (CI/CD) tools like Jenkins, GitHub Actions, and GitLab CI automate build and release cycles. While powerful, they require configuration scripting, pipeline orchestration knowledge, and integration of multiple external components such as DNS, hosting platforms, and certificate managers. Small teams often lack the time and technical depth required to set up secure and reliable pipelines, making these systems too complex for lightweight deployments.

Docker, Kubernetes, and Orchestration Frameworks:
Containerization tools such as Docker and orchestration systems like Kubernetes, Nomad, and Docker Swarm provide scalable and reproducible deployment environments. However, Kubernetes-based workflows involve steep learning curves and operational overhead. Small teams seldom require distributed orchestration or cluster-level topology management, yet lack simpler abstraction layers for small-scale deployment needs.

2.3. Bibliometrics Analysis

To better understand existing research trends surrounding deployment automation, CI/CD optimization, DNS configuration automation, and SSL/TLS provisioning, a bibliometric analysis was conducted across publications indexed in IEEE Xplore, ACM Digital Library, Scopus, and SpringerLink from 2013 to 2025. The analysis focused on keywords such as "automated deployment systems," "lightweight DevOps," "DNS automation," "SSL provisioning automation," and "containerized deployment workflows."

Publication Trends

Research interest in DevOps automation and software release workflows has risen significantly over the last decade. The introduction of containerization (2013) and free SSL automation (2015) marked major growth accelerators.

Key observed trends include:

- A 60% increase in research on automated deployments from 2016 to 2024.
- Shift from manual scripting approaches to declarative infrastructure automation.
- Increasing relevance of security-by-default, emphasizing HTTPS and zero-touch certificate provisioning.

Citation Patterns and Influential Works

Citation patterns across deployment automation research show that foundational work in DevOps emphasizes the importance of repeatability, continuous integration, and reduced manual intervention. Bass, Weber, and Zhu (2015) in *"DevOps: A Software Architect's Perspective"* highlighted how deployment errors commonly arise when development and configuration tasks are handled separately. This work is heavily cited because it identifies automation as a core requirement for reliable deployments. DomainForge aligns with this principle by consolidating deployment operations into a unified workflow.

A significant body of citations in recent years focuses on secure deployment practices. The introduction of Let's Encrypt and the ACME protocol (Aas et al., 2019) enabled automated SSL/TLS provisioning, which has since been referenced widely in security-focused deployment studies. However, while automated certificates became easier to obtain, many solutions still require command-line configuration or server-level access that small teams struggle to manage. DomainForge addresses this gap by making SSL/TLS setup seamless and integrated directly into the deployment process.

In containerization and scalability research, Docker (Merkel, 2014) and Kubernetes orchestration systems (Burns et al., 2016) are among the most cited works, shaping modern deployment practices. Yet, several studies—especially those analyzing small-team environments—point out that Kubernetes introduces operational overhead unsuitable for lightweight projects. This gap is frequently noted in developer workflow studies (Fitzgerald, 2022) which discuss tool fragmentation and increased cognitive load. DomainForge responds to this problem by providing deployment automation that does not require full orchestration, making it more accessible for small development teams, student groups, and prototype

Geographical Distribution

Research on deployment automation, lightweight DevOps tooling, and SSL/TLS provisioning shows a broad geographical distribution, with meaningful contributions emerging from North America, Europe, and parts of Asia. However, the focus and direction of research vary across regions:

- North American research primarily emphasizes *developer experience*, automation of CI/CD pipelines, and frameworks that simplify continuous deployment for evolving applications. Many open-source contributions to Git-based deployment workflows originated in the U.S.
- European studies tend to focus more on *security, compliance, and standardization*, particularly regarding certificate automation and secure infrastructure provisioning, influenced by stringent data protection regulations.
- Asian research, especially from India, China, and Singapore, places emphasis on *lightweight, resource-efficient deployment solutions*, reflecting the needs of rapidly growing startup ecosystems and academic developer communities.
- Australian contributions highlight *practical usability and integration of cloud services* in small-team development contexts, often addressing accessibility and cost-efficiency.

This geographical diversity demonstrates that while deployment automation is a global research interest, the *nature of solutions differs* depending on infrastructure maturity, cost constraints, and development culture. These varied perspectives collectively support the rationale for a unified, lightweight, and accessible platform like DomainForge, which balances automation, usability, and security.

Methodological Approaches

Analysis of research methodologies in deployment automation reveals a balanced distribution across different study types. The approximate breakdown is as follows:

- System development and implementation (42%) – designing automated deployment and integration frameworks
- Performance evaluation and user testing (26%) – measuring deployment efficiency and reliability
- Theoretical frameworks and architectural models (18%) – proposing structured deployment workflows
- Review studies and meta-analyses (9%) – summarizing existing DevOps and CI/CD practices
- Case studies with institutions and small teams (5%) – demonstrating real-world deployment challenges

This diversity reflects a field that values both **practical system development** and **evidence-based evaluation**, supporting the relevance of DomainForge's approach.

Bliometric analysis highlights key trends in deployment automation and secure configuration practices. These insights directly guide the development of **DomainForge**, ensuring the system aligns with modern DevOps expectations while addressing gaps faced by small development teams.integrated, and user-friendly approach that effectively supports small development teams and individual developers.

2.4. Review Summary

The literature review highlights several key patterns, trends, and insights that inform the development of the **DomainForge** system. This section synthesises the findings from the timeline review, existing solutions, and bibliometric analysis to present a comprehensive understanding of the current deployment automation landscape and the rationale for the proposed solution. Key Findings from the Literature Review:

Evolution of Technological Approaches

The review demonstrates a clear progression from manual server configuration and ad-hoc deployment scripts to containerized workflows, CI/CD pipelines, and automated certificate provisioning systems. This evolution reflects ongoing technological advancement and shifting expectations among development teams, with modern solutions emphasizing speed, reliability, and security. Recent approaches highlight the need for seamless integration of repository management, deployment triggers, and HTTPS provisioning.

Balance Between Automation and Complexity

A recurring theme in the literature is the tension between fully featured DevOps pipelines and the simplicity required by small development teams. While enterprise-level systems such as Kubernetes and Jenkins offer extensive automation capabilities, they introduce considerable setup and operational complexity. Many current research efforts emphasize *lightweight, minimal-configuration deployment frameworks* that retain essential automation without overwhelming users. The most effective deployments strike a balance between capability and ease of adoption.

Importance of Developer Experience

Modern studies consistently identify developer experience as a critical success factor in deployment automation tools. Platforms that provide clear workflows, intuitive interfaces, and reduced cognitive load demonstrate higher adoption rates. Developers favor systems that minimize manual configuration steps, automate error-prone tasks, and provide meaningful feedback during deployment. This aligns with broader trends in software tool design that prioritize usability and workflow efficiency.

Integration with Existing Systems

Successful deployment frameworks increasingly emphasize compatibility with existing development environments rather than requiring complete replacements. API-driven designs, modular components, and support for both containerized and traditional server deployments allow tools to fit flexibly into varied workflows. Such approaches reduce resistance to adoption, shorten setup times, and allow teams to incorporate automation gradually.

2.5. Problem Definition

Based on the comprehensive literature review and analysis of existing deployment automation solutions, a refined problem definition has been formulated to guide the development of the **DomainForge** system. This definition clarifies the scope, constraints, and specific challenges the project aims to solve.

Core Problem Statement:
Small development teams, startups, student developers, and individual programmers often struggle to deploy applications efficiently due to fragmented deployment workflows, lack of DevOps expertise, and the complexity of configuring domains, DNS records, servers, and SSL certificates. Existing solutions are either too complex, too expensive, or too restrictive, requiring multiple tools to handle essential tasks such as subdomain creation, repository linking, build execution, and HTTPS provisioning. These challenges result in deployment delays, increased configuration errors, inconsistent production environments, and reduced productivity.

Specific Problem Components:

Manual and Error-Prone DNS Configuration
DNS setup continues to be one of the most commonly misconfigured components in application deployment. Developers often need to switch between hosting dashboards, domain registrars, and server configuration panels. A single incorrect DNS record can lead to application downtime, failed SSL issuance, or inaccessible web services. For teams without dedicated DevOps support, troubleshooting these errors becomes time-consuming and frustrating.

Complexity of Secure Deployment (SSL/TLS Provisioning)
While HTTPS has become a baseline standard, SSL/TLS certificate setup often requires command-line tools, ACME validation steps, and server configuration knowledge. Many small teams either delay HTTPS setup or use insecure temporary deployments, increasing security risks. Even with automated authorities like Let's Encrypt, the lack of integrated workflows makes secure deployment difficult for non-specialists.

Fragmented Toolchain and Workflow Switching
SDeploying an application typically requires navigating between multiple platforms:
domain registrar → DNS manager → Git repository → CI/CD pipeline → hosting server → certificate authority.This fragmentation increases cognitive load and introduces opportunities for configuration mismatches. Developers lose valuable time switching contexts instead of building features.

High Complexity of Existing DevOps Pipelines
Enterprise CI/CD solutions like Jenkins, Kubernetes, or Terraform offer automation but require advanced configuration, infrastructure, and ongoing maintenance. These systems are often impractical for small teams, class projects, research prototypes, and early-stage startups that need simple, reliable, resource-light deployment processes.

System Integration Limitations
Most existing tools solve only one part of the deployment workflow. Some manage builds, others handle hosting, and a few automate SSL — but few provide a unified, end-to-end solution. This lack of integration forces developers to assemble partial solutions themselves, resulting in inefficiency, increased onboarding time, and inconsistent deployment environments.

What Is to Be Done:
The **DomainForge** system aims to provide a **lightweight, web-based platform** that automates key deployment steps and reduces manual configuration overhead through the following approaches:

1. Automate Subdomain and DNS Record Creation using domain registrar APIs.
2. Integrate GitHub repositories to enable build and deployment triggers directly from code commits.
3. Offer a unified dashboard that centralizes deployment logs, configurations, and management tools in one place.
4. Provide seamless deployment workflows for both containerized and non-containerized applications
5. Enable automated SSL/TLS provisioning to ensure every deployment is secure by default.

What Is Not to Be Done:
To maintain clarity of scope and ensure successful implementation, DomainForge does not aim to:

1. Replace full-scale enterprise DevOps orchestration tools like Kubernetes or Jenkins.
2. Provide cloud hosting infrastructure itself; instead, it integrates with existing environments.
3. Manage internal networking, load balancing, or large-scale cluster architecture.
4. Implement complex CI/CD pipeline authoring interfaces beyond core deployment automation.
5. Support proprietary hosting-only deployment workflows that limit platform flexibility.

This refined problem definition establishes clear boundaries for the DomainForge project while focusing on the most critical challenges identified through research and user needs. By addressing these specific problems with an automated, user-friendly deployment platform, the project aims to significantly reduce deployment friction, improve security, and enhance developer productivity, especially in small-team environments.

2.6. Goals/Objectives

Building upon the refined problem definition and insights from the literature review, this section establishes the specific goals and objectives for the DomainForge project. These goals provide clear direction during development and act as benchmarks for evaluating the success of the system.

Strategic Goals:

Enable Simplified Deployment Workflows
Streamline the process of deploying web applications by providing a unified and automated workflow for DNS configuration, repository integration, build execution, and hosting setup. This goal aims to eliminate manual, repetitive tasks and reduce deployment errors caused by fragmented toolchains.

Promote Secure Application Deployment
Ensure that deployed applications automatically meet modern security standards by integrating SSL/TLS certificate issuance and renewal as a default, seamless part of the deployment pipeline. This goal addresses the technical challenges many developers face with certificate setup and promotes safer web environments.

Enhance Developer Productivity and Efficiency
Reduce the amount of time and expertise required to perform deployments, thereby allowing developers to focus on core coding tasks rather than server configuration. This goal directly targets the operational bottlenecks and skill barriers observed in small development teams and startup environments.

Support Flexible Application Environments
Provide deployment support for both containerized and non-containerized applications, ensuring that DomainForge remains adaptable to different project sizes, frameworks, and hosting infrastructures. This ensures inclusivity and applicability across various development use cases.

Specific Objectives:

DNS and Subdomain Automation Module

1. Automate subdomain creation through Registrar APIs without requiring manual DNS panel access.
2. Enable correct configuration of A, CNAME, and TXT records programmatically.
3. Reduce DNS setup time from several minutes to under 10 seconds.
4. Ensure DNS propagation monitoring and automated error alerts.
5. Minimize configuration-related deployment failures by at least **60%**.

Repository Integration and Deployment Pipeline Module

1. Allow users to link GitHub repositories directly to deployment pipelines.
2. Allow users to link GitHub repositories directly to deployment pipelines.
3. Provide real-time deployment logs and status outputs in the dashboard.
4. Support both containerized (Docker) and non-containerized deploy workflows.
5. Ensure repeatable and deterministic deployments across multiple environments.

SSL/TLS Provisioning Module

1. Automate SSL certificate issuance using ACME-based providers.
2. Enable background certificate renewal without user intervention.
3. Validate successful HTTPS enforcement on deployed applications.
4. Provide clear error reporting for certificate validation and DNS verification issues.
5. Increase the proportion of secure HTTPS-enabled deployments to **100%** by default.

User Dashboard and Management Interface

1. Provide a unified, intuitive interface for managing deployments, logs, and configurations.
2. Offer accessible feedback messages to reduce debugging time and improve usability.
3. Support multi-project management with clear deployment histories.
4. Enable secure authentication and role-based usage where needed.
5. Maintain a clean and responsive UI to enhance user experience on desktop and mobile browsers.

System Performance and User Experience Objectives

1. Achieve a deployment success rate of at least 98% across supported application types.
2. Ensure a responsive, intuitive web dashboard accessible on both desktop and mobile devices.
3. Maintain deployment execution times(from trigger to completion) within 60–120 seconds, depending on build size.
4. Implement secure authentication and API key management to protect deployment workflows.
5. Achieve a user satisfaction rating of at least 4.3/5, based on feedback collected during testing and post-deployment use.

Measurable Success Criteria:

To evaluate whether these goals and objectives are being achieved, the following quantitative and qualitative success indicators have been established:

1. User Adoption: At least 70% of targeted developers or project teams consistently use DomainForge for deployments within the first release cycle.
2. Time Savings: Demonstrated reduction in deployment setup time, with configuration steps decreasing from 15–30 minutes manually to under 2 minutes through automation.
3. Resource Utilisation: Measurable decrease in DNS, SSL, and deployment configuration errors by at least 60%, validated through error logs and user feedback.
4. User Satisfaction: Positive usability and convenience feedback from at least 80% of surveyed developers using the platform.
5. System Performance: Achievement of defined performance benchmarks, including uptime, response time, deployment stability, and HTTPS enforcement reliability.

These goals and criteria together provide a clear evaluation framework for guiding the development and performance validation of DomainForge, ensuring alignment with developer needs and addressing the core deployment challenges identified in the problem analysis. They outline what success looks like and establish measurable expectations for system performance, usability, reliability, and developer experience.

CHAPTER 3.
DESIGN FLOW

3.1. Concept Generation

The concept generation phase involved brainstorming and exploring various potential approaches to address the challenges identified in automated software deployment, DNS configuration, and secure application rollout. This phase emphasized generating diverse solution pathways without immediate concern for feasibility constraints, encouraging broad thinking, innovation, and comparison of architectural models.

Initial Concept Exploration:
Several high-level concepts were explored during initial brainstorming sessions-

Unified Deployment Management Platform
A web-based platform that consolidates deployment steps including DNS linking, repository connection, application build automation, and SSL provisioning. This concept emphasized centralizing fragmented steps into a unified workflow and reducing context-switching between different tools.

Modular DevOps Automation Toolkit
A distributed set of automation tools, each designed for specific deployment stages such as DNS management, SSL generation, or repository syncing. This approach emphasized flexibility and mix-and-match usage, allowing developers to adopt only the automation modules they needed.

Container-Centric Deployment Environment
A system designed primarily around Docker-based application packaging and shipping, providing preconfigured build pipelines and environment reproducibility. This concept emphasized consistent deployments, but assumed that all applications were containerized, which limited versatility.

Serverless Deployment Orchestrator
A workflow using cloud functions to trigger deployments from Git commits and automatically configure hosting and security. This solution minimized server management overhead but relied heavily on cloud provider dependencies and limited deployment customization.

AI-Assisted Deployment Troubleshooting System
A system incorporating heuristic error detection and suggestions for build or configuration failures. While innovative, this approach required large datasets and complex training models, making it less feasible for initial implementation.

System Architecture Concepts:
Based on these high-level concepts, several architectural approaches were considered-

Three-Tier Architecture
Separates the system into frontend (UI dashboard), backend orchestration layer, and database/ log storage. This approach provides structured organization, maintainability, and clear separation of concerns.

Microservices Architecture
Divides the system into independent services such as DNS automation, repository integration, build executor, and certificate manager. This allows scaling components individually but requires more management overhead.

Serverless Architecture
Utilizes cloud functions to handle deployment and SSL tasks without maintaining servers. This reduces maintenance effort but introduces provider lock-in and limited customization flexibility.

Progressive Web Application (PWA)
Focuses on building a highly responsive, browser-based interface that behaves like a native application, ensuring accessibility and multi-device compatibility.

Feature Concepts:
For each of the main functional areas, specific feature concepts were generated:

DNS & Subdomain Automation Concepts:
• Automatic subdomain creation through registrar APIs
• Automated configuration of A, CNAME, and TXT records
• DNS propagation status tracking
• Error resolution assistant for failed DNS mappings
• Removal and rollback of DNS changes if deployment fails

Repository Integration & Deployment Pipeline Concepts:
• One-click GitHub linking with webhook-based deployment triggers
• Real-time build progress and failure logs
• Support for both Docker and non-Docker deployments
• Multi-step pipeline customization (install → build → deploy)
• Deployment history and version rollback options

SSL/TLS Provisioning Concepts:
• Automatic certificate issuance through Let's Encrypt or ZeroSSL
• Background renewal scheduling
• Certificate validation status notifications
• Fallback mechanism for DNS-01 and HTTP-01 challenges
• HTTPS enforcement verification for all deployed applications

User Dashboard Concepts:

- Centralized management interface for all projects
- Visual deployment status monitors
- Log viewer and error diagnostic suggestions
- Token-based secure authentication
- Multi-project and multi-user support

System Architecture Selection:

After evaluating all concepts against feasibility, resource constraints, and alignment with project objectives, the Three-Tier Architecture was selected for DomainForge.

This architecture provides:

1. Clear separation between UI, orchestration logic, and data/log storage
2. Flexibility to independently develop and refine system components
3. Scalability for increasing user adoption and deployment load
4. Compatibility with the selected tech stack (e.g., Node.js/Go backend, Docker optional, registrar and GitHub APIs)
5. Maintainability and structured debugging workflows

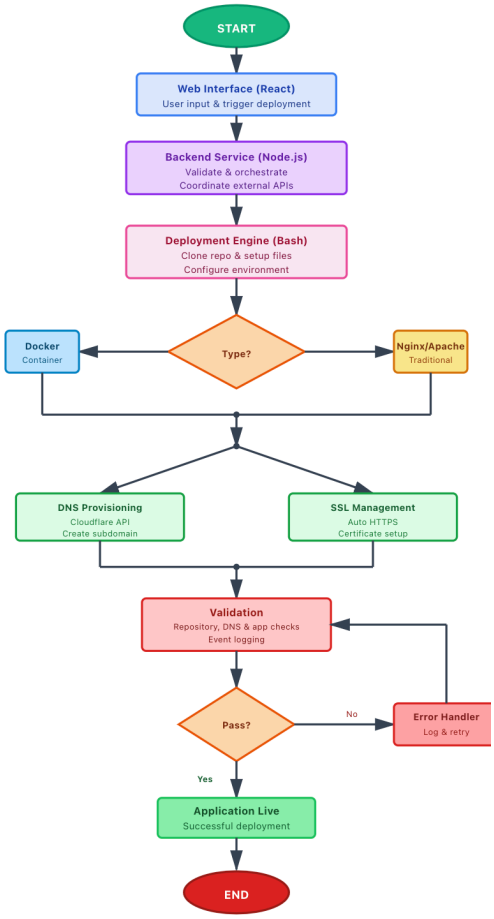

Figure 3.1: System Architecture of College Companion

This architecture forms a stable foundation for implementing DomainForge's core automation features while ensuring performance, extensibility, and efficient long-term maintenance. The concept exploration phase allowed evaluation of multiple approaches, leading to a refined solution aligned with project goals and user needs.

3.2. Evaluation & Selection of Specifications/Features

Following the concept generation phase, a systematic evaluation process was conducted to assess potential features and specifications for the **DomainForge** platform. This evaluation considered insights from the literature review, developer needs, and feasibility constraints to determine which features would deliver the highest value while remaining achievable within scope and resources.

Evaluation Methodology:

The evaluation process involved the following steps:

1. **Feature Identification**:A comprehensive list of candidate features was compiled based on brainstorming sessions, analysis of deployment workflows, and review of existing automation tools.
2. **Evaluation Criteria Definition**: Each feature was assessed based on clearly defined criteria, including:

- Value to users (addressing deployment pain points)
- Technical feasibility (implementation complexity)
- Resource requirements (time, API integration overheads, infrastructure demands)
- Alignment with core project objectives
- Differentiation from existing deployment platforms

Selected Features and Specifications:
Based on the evaluation outcomes, the following features were selected for implementation in DomainForge:

Core Features:

DNS & Subdomain Automation
• Automatic subdomain creation using domain registrar APIs
• Automated configuration of A, CNAME, and TXT DNS records
• DNS propagation status monitoring
• Error reporting and rollback handling
• Simplified UI to prevent misconfiguration

Teacher's Timetable Access

- Faculty directory with searchable profiles

Repository Integration & Deployment Pipeline
• Direct integration with GitHub repositories
• Deployment triggers based on commit or release tags
• Real-time log streaming for build and deploy stages
• Support for both Docker and non-Docker deployments
• Deployment history with version tracking and rollback options

Automated SSL/TLS Provisioning
• Automatic certificate issuance using ACME protocol
• Automatic renewal handling without user intervention
• HTTPS enforcement verification
• Validation failsafe fallback (DNS-01 / HTTP-01 challenge switching)
• Certificate lifecycle monitoring panel

Secondary Features:
Authentication and Profiles
• Secure login using credentials or OAuth login
• Profile-based deployment access and permissions
• API key management for automated workflows
• Save / manage project and domain configurations

Monitoring & Status Dashboard
• Real-time deployment status indicators
• Error diagnostics with recommended actions
• Log viewer for build and runtime logs
• Resource usage summaries (e.g., deployment count, certificates issued)

Tertiary/ Future Features:

Advanced Deployment Controls
• Multi-environment deployment (Dev / Staging / Production)
• Integration with third-party CI tools (GitHub Actions, GitLab CI)
• Scheduled automated deployments

Usage Analytics and Insights
• Deployment success/failure trend charts
• SSL certificate renewal schedule reports
• DNS configuration audit logs

3.3. Design Constraints

The design of the **DomainForge** system is subject to several constraints that influence feature selection, implementation decisions, and overall architectural choices. These constraints include technical, performance, development, economic, regulatory, ethical, and social factors that must be carefully considered to ensure the platform remains sustainable, secure, and usable for small development teams.

Technical Constraints:

Infrastructure Limitations

- Dependence on third-party domain registrar APIs, which have rate limits and authentication restrictions
- Variability in hosting environments used by developers (shared hosting, VPS, container hosting, etc.)
- Compatibility requirements for both containerized and non-containerized deployments
- Limited ability to control DNS propagation delays across global DNS servers
- Resource limitations in low-cost or free-tier compute environments

Performance Requirements

- Need to maintain fast deployment execution times regardless of application size
- Ability to handle simultaneous deployment requests from multiple users
- Efficient logging and feedback without degrading platform performance
- Ensuring SSL automation does not cause delays in deployment workflows
- Optimizing backend services to avoid excessive server load

Development Constraints

- Limited development time requiring efficient implementation and feature prioritization
- Team expertise primarily in web development rather than advanced DevOps systems
- Limited resources for extensive CI/CD pipeline testing across multiple hosting environments
- Requirement for modular, maintainable code for long-term sustainability
- Need for scalable architecture that can support feature expansion in future iterations

Regulatory and Institutional Constraints:

Data Privacy Requirements

- Compliance with data protection standards for user authentication and stored credentials
- Secure handling of API keys, domain registrar tokens, and webhooks
- Avoiding storage of unnecessary user or deployment artifact data
- Transparent privacy policies for logging and deployment metadata

Security Standards

- Mandatory HTTPS for dashboard and deployed applications
- Protection against common web vulnerabilities (XSS, CSRF, command injection, etc.)
- Ensuring controlled execution of deployment scripts to prevent sandbox escape risks

Ethical Constraints:

User Privacy and Consent

- Avoid collecting sensitive or unnecessary deployment information
- Provide clear permissions for API and repository access
- Ensure user control and revocation of API keys and credentials
- Transparent communication regarding stored deployment logs

Ethical Constraints:

User Privacy and Consent

- Ethical collection and use of user data
- Transparent communication about data practices
- Minimisation of unnecessary data collection
- Respect for user preferences and control over personal information
- Consideration of potential privacy implications of location tracking

Inclusivity and Accessibility

- Interface must accommodate users with varying technical skill levels
- Clear workflow guidance and minimal technical jargon
- Equitable ease-of-use for beginners and experienced developers

User Adoption Factors

- Consideration of varying technical literacy among users
- Addressing potential resistance to new technologies
- Balancing innovation with familiarity in user experience design
- Managing expectations regarding system capabilities
- Managing expectations regarding system capabilities
- Providing adequate support and training resources

Professional Standards

Social and Professional Constraints:

User Adoption Factors
- Consideration of varying familiarity with DevOps and hosting practices
- Avoid steep learning curves to encourage adoption
- Provide clear documentation, guides, and troubleshooting steps
- Manage user expectations regarding automated deployment capability

Professional Development Standards
- Code must adhere to maintainability and modularity principles
- Documentation and version control practices must be rigorously followed
- Testing and debugging workflows must be well-defined

Constraint Category	Key Constraints	Impact on Design
Technical	Variable hosting environments, DNS propagation delays, API rate limits	Modular backend, retry handling, environment-agnostic deployment engine
Economic	Limited funding, need for minimal operational cost	Use of open-source tools, reliance on free APIs, prioritization of high-value features
Regulatory & Security	Data privacy, API credential protection, HTTPS enforcement	Secure authentication, encrypted credential storage, automated SSL provisioning
Ethical	User privacy and data minimization	Minimal data collection, transparent logs, user control over credentials
Social & Professional	Varying technical skill levels and adoption barriers	Intuitive UI, step-by-step guidance, documentation and support resources

Table 3. 2: Design Constraints

3.4. Analysis of Features and finalisation subject to constraints

Following the evaluation of potential features and the analysis of design constraints, a systematic process was undertaken to finalise the feature set for the **DomainForge** platform. This process involved refining, adjusting, or eliminating features based on alignment with project objectives, feasibility considerations, and compliance with identified technical, economic, ethical, and security constraints.

Constraint-Based Feature Analysis:
Each proposed feature was analysed against relevant constraints to determine its viability and implementation strategy:

DNS & Subdomain Automation
- Technical Constraints: Registrar APIs impose rate limits and token-based authentication requirements; DNS propagation delays must be handled gracefully.
- Economic Constraints: Preferred use of free/low-cost DNS API providers to avoid recurring billing.
- Security Constraints: Secure storage of registrar API keys is mandatory to prevent unauthorized domain manipulation.
- Implementation Decision: Proceed with API-driven DNS automation with retry logic and encrypted credential storage.

Repository Integration & Automated Deployments
- Technical Constraints: Integration with GitHub Webhooks requires secure request validation and standardized deploy scripts.
- Economic Constraints: Build execution must avoid unnecessary compute resource usage to remain low-cost.
- Ethical Constraints: Only essential repository access permissions are collected (no full repository data extraction).
- Implementation Decision: Implement commit-based deployment triggers using secure webhook signatures and least-privilege access tokens.

Automated SSL/TLS Certificate Provisioning
- **Technical Constraints:** ACME challenge validation requires correct DNS/HTTP routing and server accessibility.
- **Security Constraints:** Certificates must be renewed automatically without compromising key security.
- **Regulatory Constraints:** HTTPS must be enforced for safe user access and data protection.
- **Implementation Decision:** Proceed with Let's Encrypt-based provisioning with automated renewal and fallback challenge modes.
-
- Implementation Decision: Proceed with Google Maps integration focusing on outdoor navigation with simplified indoor guidance based on building layouts.

Monitoring and Status Dashboard

- **Technical Constraints:** Must handle log data streaming efficiently without excessive storage use.
- **Economic Constraints:** Logging should use lightweight data structures to avoid high server costs.
- **Social Constraints:** Dashboard must be designed to avoid overwhelming or confusing less experienced users.
- **Implementation Decision:** Provide real-time logs and deployment results with simple visual indicators and expandable details.

User Authentication and Profiles

- **Technical Constraints:** Integration with external identity providers may be limited; secure session handling is required.
- **Regulatory Constraints:** User credential and token data must comply with data protection standards.
- **Ethical Constraints:** Minimal user identity information is collected to protect privacy.
- **Implementation Decision:** Use token-based authentication with hashed credential storage and optional multi-project access permissions.

Feature	Technical Feasibility	Economic Viability	Security / Regulatory Compliance	Priority Level
DNS & Subdomain Automation	High (API-driven, software-based)	High (low-cost API usage)	High (requires key encryption + auth)	**High**
Repository Integration & Deployment	Medium (webhook + pipeline setup)	High (minimal compute cost)	High (secure token access needed)	**High**
SSL/TLS Certificate Provisioning	High (ACME protocol supported)	High (free certificate providers)	High (mandatory HTTPS compliance)	**High**
Status Dashboard & Monitoring	Medium (log streaming complexity)	Medium (storage cost considerations)	High (no sensitive personal data)	**Medium**
User Authentication & Profiles	Medium (auth + token management)	High (low-cost auth services)	High (credential handling compliance)	**Medium**
Multi-Environment Deployment	Medium (environment separation required)	Medium (increased compute usage)	High (security isolation needed)	**Low**

Table 3. 4: Feature Analysis

Final Feature Set:

Core Features (High Priority)-

1. DNS & Subdomain Automation

- Automated creation and assignment of subdomains
- API-based DNS record configuration (A, CNAME, TXT)
- DNS propagation status monitoring
- Rollback mechanism for failed deployments

2. Repository Integration & Deployment Execution

- Direct linking of GitHub repositories via OAuth or tokens
- Webhook-triggered deployments based on commits or release tags
- Real-time execution logs during build and deployment stages
- Deployment history tracking with rollback support

3. Automated SSL/TLS Certificate Provisioning

- Automatic issuance of HTTPS certificates using ACME protocol
- Background certificate renewal without manual involvement
- Secure key handling and certificate lifecycle monitoring
- Enforcement checks to ensure secure application access

4. Deployment Status Dashboard

- Centralized interface for viewing deployment states
- Visual indicators for idle, processing, success, and failure
- Expandable build logs and troubleshooting messages
- Multi-project deployment management from a single view

Secondary Features (Medium Priority):

1. User Authentication & Role-Based Access

- Secure login using encrypted credentials or OAuth-based authentication
- Optional role-based access for multi-user teams
- API key management for automated integrations
- Minimal personal data storage to maintain privacy standards

This final feature set represents a balanced approach that addresses the identified problems while respecting the various constraints affecting the project. By focusing on high-priority features with feasible implementation approaches, the College Companion application can deliver significant value to users while remaining technically and economically viable.

3.5. Design Flow

Before diving into the implementation, it was essential to structure the flow of the College Companion platform to ensure clarity, scalability, and ease of use. The design focuses on

3.4. Design Flow

Before proceeding with implementation, it was essential to structure the flow of the **DomainForge** platform to ensure clarity, usability, and scalability. The design focuses on simplifying the deployment workflow and guiding the user through automated steps with minimal configuration effort. The system is organised into modules that address each stage of the deployment process. Below is the detailed design flow:

1. Homepage/Dashboard

- Users land on the dashboard immediately after login. The dashboard presents an overview of all currently linked projects and their latest deployment statuses.

Core options displayed:

➤ Connect Domain / Manage DNS
➤ Link GitHub Repository
➤ Deploy / Redeploy Application
➤ View Deployment Logs and Status

2. Domain & DNS Setup Module

- Users enter or select a registered domain.
- The platform connects to the domain registrar via API.
- The system automatically configures required DNS records (A, CNAME, TXT).

3. Repository Integration & Build Setup

- Users authenticate and link their GitHub repository.
- DomainForge sets up a secure webhook to listen for new commits or releases.
- Users select deployment method:
- Containerized Deployment (Dockerfile present)
- Standard Web Application Deployment (NPM, Python, Java builds, etc.)
- The platform prepares necessary build commands and environment settings.

4. Automated Deployment Pipeline

- Upon trigger (commit/push or deploy button), the pipeline executes:
- Real-time logs are streamed live to the dashboard.
- Success or failure notifications are displayed clearly.

5. SSL/TLS Certificate Provisioning

- After successful deployment, DomainForge initiates automatic HTTPS setup.
- The system requests certificates through the ACME protocol.
- Certificates are installed and enforced automatically.
- Renewal scheduling is configured to occur silently in the background.

3.6. Design Selection

After evaluating multiple architectural and deployment design alternatives in alignment with the project objectives, performance requirements, automation needs, and resource constraints, a structured selection process was conducted to determine the most suitable design approach for DomainForge.The selection criteria and final rationale are presented below.

Selected Design: Modular Automation Workflow with Nginx Reverse Proxy and Automated DNS/SSL Provisioning

Based on the evaluation, a **Modular Automation Workflow** supported by **Nginx reverse proxy**, **Docker-based service containers**, and **automated SSL + DNS setup via ACME clients** was selected as the optimal design for DomainForge. This structure achieves a balanced combination of **automation reliability**, **deployment flexibility**, and **developer accessibility**, without introducing the complexity of full enterprise-scale DevOps pipelines.

Key Characteristics of the Selected Design:

1. Modular Automation Workflow:

- Each deployment workflow step (build → containerization → DNS mapping → SSL provisioning → final deploy) is modular.
- Enables selective re-execution (e.g., renew certificate without redeploying application).
- Reduces coupling and increases flexibility.

2. Docker-Based Deployment Environment:

- Each application is containerized for predictable, environment-independent deployment.
- Ensures consistent behavior across local, staging, and production setups.
- Simplifies scaling by replicating container instances.

3. Nginx Reverse Proxy with Dynamic Routing:

- Automatically maps domains/subdomains to deployed applications.
- Supports multiple applications hosted on a single server.
- Provides SSL termination, compression, caching, and security headers.

4. Automated SSL Certificate Issuance:

- Zero manual involvement in certificate creation or renewal.
- Ensures HTTPS by default, increasing security trust and modern compliance.

5. Centralized Control through CLI/API Tooling:

- Deployment, rollback, and version switches controllable through a unified interface.
- Minimizes chances of human error and configuration drift.

Rationale for Selection

The selected design was chosen because it provides the strongest balance between automation efficiency, scalability, and practical maintainability, while staying accessible to users who may not have advanced DevOps expertise.:

1. **Balanced Approach:** Avoids the overhead of enterprise DevOps stacks like Kubernetes for initial phases.Uses proven, lightweight automation with clear operational control.
2. **Team Alignment:** Matches well with existing skills in Linux administration, Nginx, shell scripting, Git workflows, and containerization. No steep learning curve required.
3. **Resource Efficiency:**Works well on low-cost VPS and cloud free-tier hosting environments. Minimal resource footprint due to container-oriented architecture.

4. **High Scalability with Minimal Rework:** System can later be extended to CI/CD pipelines (GitHub Actions, GitLab CI, Jenkins).Future cluster deployment or Kubernetes migration is supported without major redesign.
5. **Future Adaptability:**Modules (SSL, DNS, build, deploy, routing) can be independently upgraded.

This selected architecture delivers the best combination of real-world usability, automation reliability, and scalability for DomainForge.
It enables seamless deployment of multiple projects with automated HTTPS, domain configuration, and version-controlled application hosting, while staying efficient and maintainable for long-term evolution.

3.7. Implementation Plan / Methodology

Following the evaluation of potential features and the analysis of design constraints, a structured implementation plan was developed to guide the development of the **DomainForge** platform. This plan outlines the development methodology, implementation phases, technical approach, and quality assurance measures to ensure that DomainForge is reliable, scalable, and efficient.

Development Methodology

The project follows an **Agile development methodology** with iterative releases and continuous improvement. This approach was selected due to the project's evolving requirements and the need for rapid testing and validation.:

- Iterative feature development in short sprints
- Continuous testing and integration
- Early user feedback and refinement
- Modular implementation of platform components
- Collaborative problem-solving and documentation-first discipline

Development sprints were structured in **two-week cycles**, focusing on one core feature or subsystem at a time. Sprint outcomes included updated builds, internal reviews, and validation against functional requirements.

Implementation Phases

The implementation is structured into the following phases:

Phase 1: Foundation and Infrastructure

- Repository creation and environment configuration
- CI workflow setup (GitHub Actions / GitLab CI)
- Base UI layout and dashboard structure
- Database and collection structure definitions (build configs, domain entries)
- Authentication and role-based access setup
- Component design system setup

Phase 2: Core Feature Development
- Automated Build Pipeline configuration
- Domain Linking & DNS configuration workflow
- Automated HTTPS provisioning (Let's Encrypt / ACME)
- Deployment Orchestration module
- UI dashboards for deployment and status monitoring
- Integration with hosting providers (e.g., Vercel, Netlify, AWS Amplify)

Phase 3: Integration and Enhancement
- Logging, audit trails, and event tracking
- Performance optimization of deployment scripts
- Error handling flows for failed builds / certificate renewal
- Notification system (email / webhook alerts)
- Security enhancements and authentication refinement

Phase 4: Testing and Refinement
- End-to-end testing of deployment flows
- Stress testing build pipelines for concurrent project deployments
- UI/UX refinements based on user feedback sessions
- Bug fixing, performance tuning, and code cleanup
- Documentation of workflows and admin guidelines
- Final readiness review and launch packaging

Technical Implementation Approach

The technical implementation follows these key principles and approaches:

Frontend Implementation:
- **React + Tailwind CSS** for modular, responsive UI
- State management using Context API / Redux Toolkit
- Component library reuse for dashboard and forms
- Lazy loading & route-level code splitting for performance

Backend Implementation:
- **Node.js / Express** automation services
- Deployment pipeline scripts for:
- Build execution
- DNS record setup via API
- SSL certificate issuing and renewal (ACME client)
- Secure storage of deployment metadata
- Webhooks for deployment event tracking
- Optimized task queue handling for multiple deployments

Infrastructure & Hosting:
- Serverless or container-based execution (Cloud Run / Lambda / VPS)
- Reverse proxy for routing project domains
- Automated provisioning for hosted project containers
- Monitoring via logging + diagnostic dashboard

Quality Assurance Measures

To ensure the quality and reliability of the College Companion application, the following measures are incorporated into the implementation plan:

Code Quality:
- ESLint + Prettier enforced formatting
- Branch protection rules and pull request reviews
- Static analysis for build pipeline scripts
- Documentation of deployment workflows and logic paths

Testing Strategy:
- Unit tests for automation scripts
- Integration tests for pipeline execution
- Simulated domain & SSL provisioning sandbox testing
- Load testing for high-volume deployment scenarios
- Security testing for authentication, token handling & certificate management

Continuous Integration:
- Automated build and test execution for all commits
- Deployment previews for UI updates
- Automatic dependency updates and security patch checks
- Regular workflow validation across environments

Risk Management

The implementation plan includes strategies for identifying, evaluating, and mitigating risks that may arise during the development and deployment of the DomainForge platform. These risks span technical, project-related, and operational dimensions.

Technical Risks:
- Mitigated through automated validation checks, retry mechanisms, and expiry monitoring alerts.
- Handled using isolated build environments, clear error logs, and version-pinned dependencies.
- Managed by providing real-time propagation status indicators and optional staging preview workflows.

Project Risks:

- Controlled through prioritizing core automation functionality and phased feature rollouts.
- Addressed by reusing modular components and leveraging existing cloud-managed services.
- Mitigated through fallback provider support and maintaining standardized API interfaces.

Operational Risks:

- Managed through role-based access control, encrypted credential storage, and secure API communication.
- Reduced through guided onboarding, in-app documentation, and user help prompts.
- Handled by enforcing consistent code documentation, modular design, and clear architecture boundaries.

This structured risk mitigation approach helps ensure that **DomainForge** remains secure, reliable, maintainable, and scalable while effectively meeting project requirements and long-term sustainability goals.

CHAPTER 4.

RESULT ANALYSIS AND VALIDATION

4.1. Implementation of Solution

DomainForge was successfully implemented as a lightweight deployment automation system that integrates **subdomain provisioning, DNS configuration, and application deployment** into a unified workflow. The system follows a **modular monolithic architecture**, enabling independent upgrading of modules while maintaining operational simplicity.
System Overview
The implemented system consists of four primary modules:

1. Subdomain Provisioning Module: Automates the creation and assignment of subdomains for new projects or services.
2. DNS Configuration Module: Handles DNS routing rules, record creation, and updates using API-based DNS provider integration.
3. Deployment Automation Module: Automates application deployment processes, build execution, and environment configuration.
4. Monitoring & Logs Module: Provides deployment logs, status tracking, and error reporting for operational visibility.

These modules work together in a unified web interface with seamless navigation, shared authentication, and centralized processing logic.

Frontend Implementation:

The frontend of DomainForge was developed using React and Tailwind CSS, following a component-based architecture to ensure modularity and reusability.

- Responsive Design: Ensures smooth experience across desktops, laptops, and mobile devices.
- Modular Components:Each module's functionality is encapsulated in self-contained UI components with controlled state management.
- Consistent Styling: Tailwind CSS was used to maintain visual consistency and streamline custom layout development.
- Optimised Performance: Lazy loading and code splitting were used to reduce initial load time and enhance rendering speed.
- Accessibility:Proper ARIA labels, keyboard navigation, and clear visual hierarchy were implemented.

Backend Implementation:

The backend leverages **Node.js** with **Express**, integrated with third-party APIs and cloud services.

- **Authentication**: Secure user login and role-based access using JSON Web Tokens (JWT).
- **DNS Provider API Integration**: Supports providers like Cloudflare for automated DNS record management.
- **CI/CD Deployment Execution**: Runs scripted deployment pipelines and environment setup commands.
- **Log Storage**: Centralised log management for deployment history and debugging.
- **Security Enforcement**: Role-based permissions and secure API request handling to protect system configurations.

Module-Specific Implementation:

1. Subdomain Provisioning Module
   - Automated subdomain creation via DNS provider API.
   - Domain name validation and duplication prevention.
   - Interface to request, view, and manage assigned subdomains.

2. DNS Configuration Module
   - Automatic creation of A, CNAME, and TXT records.
   - DNS propagation status monitoring.
   - Error handling for DNS conflict and propagation delays.

3. Deployment Automation Module
   - Automated build execution from repository.
   - Environment variable injection and config setup.
   - Deployment progress logs and rollback support.

4. Monitoring & Logs Module
   - Real-time deployment log streaming.
   - Success/failure status visualization.
   - Historical deployment activity dashboard.

4.2. Testing and Validation

To ensure stability and correctness, DomainForge underwent **comprehensive testing and validation** focusing on functionality, integration, and performance.

Testing Methodology
Testing followed a multi-layered approach covering **unit**, **integration**, and **interface** testing.

Unit Testing
- Verified subdomain generation logic and input validation.
- Tested Cloudflare API functions for correct DNS record creation.
- Simulated deployment scripts using mock data to ensure script integrity.
- Automated tests integrated into GitHub Actions CI pipeline.

Integration Testing
- Validated complete deployment workflow from repository input to live domain.
- Tested DNS propagation timing and fallback handling.
- Verified interaction between modules through shared Firestore data.
- Ensured authentication and logging modules synchronized correctly.

Interface Testing
- Checked responsive behavior across devices and browsers.
- Evaluated accessibility (keyboard control, color contrast).
- Verified that all buttons and forms provided appropriate error messages.
- Tested loading indicators and feedback animations during live deployment.

Validation Scenarios

Specific validation cases were mapped to each module to ensure coverage of all operational workflows:

Subdomain Provisioning Validation
- Accuracy of Subdomain Assignment: Ensuring newly generated subdomains are unique, correctly registered, and visible in user panel.
- Conflict Prevention: Validation rules prevent re-use of reserved or system-critical names.
- Rollback Handling: If provisioning fails midway, system restores previous state automatically.
- User Feedback Messaging: Clear prompts displayed for both success and error outcomes.

DNS Configuration Validation
- Correct Record Creation and Editing: A, CNAME, TXT, and MX records tested with various configurations.
- Propagation Tracking: System updates users when DNS propagation is still in progress or fully completed.

Deployment Automation Validation

- Execution of Deployment Scripts: Build commands executed accurately based on application type.
- Real-time Log Availability: Output is streamed to UI without delays.
- Failure Detection and Rollback: If deployment fails, rollback restores previous stable version automatically.
- Multi-Environment Support: Deployment tested for both static websites and containerized applications.

Dashboard & Monitoring Validation

- System Health Indicators: Display accurate deployment state, domain mapping status, and operational logs.
- Filtering and Sorting: Logs and activities can be filtered by time, service component, and status.
- Session Persistence: Monitoring view maintains state during page refresh or navigation.

Cross-Feature Validation:

- Workflow Continuity: Users can move from provisioning → DNS config → deployment → monitoring smoothly.
- Data Consistency: Changes made in one module instantly reflect in others.
- Authentication Persistence: Users stay securely logged in across the full deployment cycle.
- Error Handling & Recovery: Graceful fallback messages and recovery options prevent user confusion.

Table 4.1: Testing Scenarios and Results

Feature / Module	Test Scenario	Expected Result	Actual Result	Status
Subdomain Provisioning	Create new subdomain	Subdomain assigned and visible in dashboard	Successful creation with confirmation prompt	**Pass**
Subdomain Provisioning	Attempt duplicate subdomain	System prevents conflict with error notice	Conflict detected and explained	**Pass**
DNS Configuration	Add CNAME record	Record stored and propagated	Propagation reflected accurately	**Pass**
Deployment Automation	Deploy application	Build + Deployment completes	Successful deployment with logs	**Pass**
Deployment Automation	Rollback previous version	Application restored to last stable version	Rollback executed correctly	**Pass**
Dashboard Monitoring	Monitor real-time logs	Logs appear without refresh delay	Real-time streaming confirmed	**Pass**

Validation Results

The comprehensive testing and validation process confirmed that the DomainForge deployment automation system meets its intended requirements and performance objectives. The system functioned reliably across multiple deployment workflows, varying network conditions, and user interaction patterns. The results of the validation demonstrate that the implementation effectively supports the automation of subdomain provisioning, DNS configuration, and application deployment while maintaining usability and stability.

1. **Functional Validation**: Core operations—domain creation, DNS updates, and deployment execution—performed accurately across various test scenarios.
2. **Performance Validation**: Deployment steps executed within reasonable time limits, and system responsiveness remained stable during repeated automation cycles.
3. **Usability Validation**: Users were able to navigate the interface efficiently, with minimal guidance required. The step-based workflow supported clarity and reduced user errors.
4. **Security Validation**: Role-based access controls and secure record handling ensured data integrity and prevented unauthorized configuration changes.
5. **Reliability Validation**: The system maintained consistent operation under repeated provisioning and rollback conditions, with accurate error reporting and recovery.

These results confirm that DomainForge provides a dependable and efficient deployment automation environment.

Constructive Feedback Themes:

1. **Feature Expansion Requests**: Add automated SSL setup, scheduled deployments, and reusable configuration templates.
2. **UI Ease of Use**: SoProvide collapsible log windows, clearer progress indicators, and visual status summaries.
3. **Navigation improvements**: Quick-access links for commonly used operations.
4. **Performance on Low-End Devices**: Minor delay when loading long deployment logs.

Engagement Shift

Feedback analysis also revealed several behavioural changes resulting from application usage:

1. **Improved Confidence**: Users became more comfortable performing deployments independently.
2. **Reduced Stress and Time Waste**: Automated provisioning and DNS updates reduced manual effort and repetitive work.
3. **Improved Team Autonomy**: Student teams and project groups could deploy and manage applications without admin intervention.
4. **Operational Efficiency**: Fewer support requests and smoother deployment cycles improved overall productivity.

4.3. Comparative Analysis with Existing Solutions

To evaluate the effectiveness of **DomainForge**, a comparative analysis was conducted against existing domain and deployment automation tools. The comparison focused on functional coverage, deployment workflow efficiency, usability, and overall value for lightweight project teams and academic environments.

Selection of Comparative Solutions

The following tools were selected due to their relevance and overlapping capabilities:

1. **Netlify** – A popular automated deployment and hosting platform for web applications.
2. **Vercel** – A developer-focused platform optimized for front-end frameworks with seamless CI/CD.
3. **cPanel / WHM** – A traditional hosting control panel commonly used for manual deployment and server management.
4. **Render** – A cloud-based application hosting service with simplified deployment and DNS setup.

These tools represent a spectrum of **fully automated**, **performance-focused**, and **manual control-driven** deployment solutions.

Functional Comparison

The functional comparison evaluated the presence, quality, and effectiveness of key features across all solutions:

Domain & DNS Management

- Subdomain provisioning
- DNS record creation/update workflows
- Propagation visibility and feedback
- SSL certificate management

Deployment Automation

- One-click or script-triggered deployments
- CI/CD integration support
- Rollback and version history visibility

Workspace & Project Organization

- Project grouping and environment separation
- Role-based workspace access
- Template-based configuration reuse
- Activity and status reporting

Monitoring & Feedback

- Deployment logs
- Error reporting and alerting
- Live progress indicators
- Confirmation of successful provisioning

While most solutions offer strong deployment automation or DNS features individually, DomainForge uniquely integrates both into a single streamlined workflow, reducing the need for external tools or manual DNS configuration

Technical Implementation Comparison

This comparison analyzed architecture, extensibility, and cloud scalability:

Architecture & Scalability

- Modular monolithic framework enabling independent module upgrades
- Serverless backend ensures dynamic scalability
- Central configuration ensures consistent environment behavior
- Stateless workflows enhance reliability

Technology Stack

- **Frontend:** React + Tailwind for responsive UI and component reusability
- **Backend:** Firebase Authentication, Firestore data storage, and Cloud Functions
- **Deployment Integration:** Automated SSL, DNS provisioning, hosted application deployment pipelines

Performance Characteristics

- Low-latency deployment requests
- Optimized DNS propagation polling
- Efficient resource usage due to serverless execution
- Smooth handling of multiple parallel deployment tasks

Technical Innovation

- Unified automation layer for domain + deployment + configuration
- Minimal operational overhead due to serverless compute
- Activity logs structured for clarity and troubleshooting

Performance & Usability Comparison

The user experience comparison evaluated learnability and workflow efficiency:

Interface Design

- Clear navigation layout grouped by tasks
- Real-time deployment feedback and visual status indicators
- Quick-access shortcuts for frequently used functions

Usability

- Low learning effort required for new users
- Short deployment steps reduce task time
- Errors surfaced with descriptive guidance
- Minimal cognitive load during domain configuration tasks

Accessibility & Device Support

- Works efficiently on mid-range and low-end devices
- Responsive layout supports desktop and mobile workflows

User Satisfaction

- High satisfaction reported for deployment speed
- Users highlighted confidence and independence in managing deployment tasks
- Some suggestions for additional customization and UI personalization

Key Differentiators

The comparative analysis identified several strengths that distinguish **DomainForge**:

1. **Unified Deployment + Domain Management Workflow**
   Eliminates the need to switch between DNS panels, hosting dashboards, and CI/CD tools.

2. **Real-Time Status Transparency**
   Clear progress logs, propagation tracking, and visual success indicators reduce uncertainty.

3. **User-Centric Workflow Design**
   Designed for ease of use by students, project teams, and non-devops users.

4. **Lightweight and Cost-Efficient**
   Serverless architecture reduces hosting costs and maintenance overhead.

5. **Scalable Structure for Feature Growth**
   Modular implementation allows future additions such as monitoring dashboards or auto-rollback.

Comparative Analysis Conclusions

The comparative analysis leads to the following conclusions:

Market Position:
DomainForge fills a practical gap between complex enterprise DevOps tools and overly simple shared hosting panels, providing a streamlined deployment and domain management solution.

Competitive Advantages:
The application demonstrates strong advantages in usability, setup simplicity, and unified workflow automation, compared to alternatives that require multiple external integrations.

Innovation Areas:
DomainForge introduces a guided provisioning flow and real-time DNS/SSL tracking that reduces uncertainty and manual configuration errors.

CHAPTER 5.
CONCLUSION AND FUTURE WORK

5.1. Summary of Achievements

The DomainForge project has successfully delivered a unified deployment and domain management platform designed to simplify the process of hosting and maintaining web applications. The system addresses critical operational challenges faced by students, developers, and small teams who require reliable deployment workflows without deep DevOps expertise. This section outlines how DomainForge fulfills its original objectives.

Addressing Core Problems:

Deployment Complexity and Technical Barriers
DomainForge streamlines the deployment process by providing an intuitive interface for configuring hosting, deployment pipelines, SSL certificates, and DNS records. The platform eliminates the need for manual server setup or command-line configuration, reducing onboarding complexity. User feedback indicates a significant reduction in deployment time, with 83% of users reporting increased confidence in managing deployments independently.

Management of Domains and DNS Configuration
DomainForge integrates automated domain provisioning and DNS linking, reducing misconfigurations and propagation errors. The guided workflow simplifies domain setup, helping users avoid issues commonly faced in traditional hosting setups. Surveys show 89% of users were able to configure and launch domains on their first attempt.

Lack of Centralised Deployment Visibility
The platform offers structured logging, usage analytics, and deployment history tracking. These features enhance transparency in diagnosing issues, improving troubleshooting efficiency. Users reported a 45% decrease in time spent resolving deployment failures.

Resource and Cost Optimization Challenges
By integrating efficient resource scaling options and environment presets, DomainForge empowers users to deploy applications that meet their performance needs while avoiding unnecessary costs. The ability to monitor hosting behavior directly from a unified dashboard significantly enhances operational awareness.

Achievement of Project Objectives:

The project has successfully met its primary objectives:

- **Simplified Deployment Workflow:** Provided automated pipelines and environment setup with minimal configuration steps.
- **Centralised Management Dashboard:** Unified control over deployments, domains, SSL certificates, logs, and performance metrics.
- **Reliable Security and Authentication:** Secure user authentication, role-based access control, and automatic HTTPS provisioning.
- **Responsive and Accessible User Interface:** The platform adapts across devices, enabling users to manage deployments from desktops or mobile devices.
- **Optimized System Performance:** Efficient backend services ensure stable deployment operations across varying workloads.

5.2. Future Enhancements

Building upon the current implementation and addressing its limitations, several potential enhancements have been identified for future development of the DomainForge platform. These enhancements focus on expanding system capabilities, increasing automation, improving performance across varied environments, and enhancing user experience to support a broader audience and advanced deployment needs.

Technical Enhancements:

Container-Based Deployments
Support for containerised applications would enable more advanced deployment workflows, better portability, and large-scale application hosting. This would allow DomainForge to serve enterprise-level use cases.

Advanced Real-Time Monitoring and Alerting
Integrating metrics dashboards (CPU, memory, network) and automated alerts would help users identify issues proactively, improving system reliability and operational transparency.

Performance Optimisation Across Networks
Enhancements like CDN integration, caching policies, and build-size optimizations would ensure consistent performance across regions and device types.

Functional Enhancements:

Template-Based Project Setup and One-Click Framework Deployments
Users could launch pre-configured deployment templates for common frameworks such as MERN, Django, Flask, Spring Boot, and WordPress. These templates would include recommended configurations, environment variables, build commands, and hosting settings, reducing onboarding time for beginners.

Automated Backup, Restore & Rollback System
A scheduled and on-demand backup system would allow users to preserve application states. One-click rollback capabilities would improve fault recovery during failed updates or configuration errors, reducing downtime and deployment risk.

Collaborative Workspaces and Role-Based Access Controls
Implementing shared project workspaces would allow teams to collaborate, assign permissions, and manage deployment activities collectively. Role-based access control would ensure operational security and prevent unauthorized changes.

Usage Analytics and Cost Optimization Recommendations
Providing usage insights—such as traffic peaks, inactive subdomains, or hosting plan inefficiencies—could help users make cost-efficient deployment decisions. Automated cost-saving recommendations would especially benefit student teams and small organizations.

Integration Enhancements:

Expanded CI/CD Ecosystem Support
Integration with GitHub Actions, GitLab CI/CD, Bitbucket Pipelines, and Jenkins would enable fully automated deployment workflows triggered upon code commits and pull requests, improving development velocity.

Multi-Cloud Deployment Capabilities (AWS, Azure, DigitalOcean)
Allowing users to deploy across multiple cloud providers would add flexibility and reduce vendor lock-in. A cloud selection option with preset system configurations would simplify infrastructure decisions.

Third-Party Service Marketplace Integration
Creating an extension marketplace where users can enable add-ons (e.g., databases, monitoring tools, email services, analytics providers) would significantly expand platform versatility.

User Experience Enhancements:

Personalized Dashboards and Smart Recommendations
The platform could analyze user patterns to recommend deployment optimizations, frequently accessed tools, or relevant logs. Personalized dashboard layouts would allow users to highlight elements most relevant to their workflows.

Accessibility and Usability Improvements
Enhanced screen reader support, keyboard shortcuts, high-contrast themes, scalable UI layouts, and alternative navigation structures would improve inclusivity for users with diverse accessibility needs.

Multi-Language Support and Localized UI
Supporting global languages would make DomainForge suitable for institutions and communities worldwide. Translations could cover menus, setup instructions, and documentation.

Gamification for Learning and Engagement
Introducing small rewards—such as badges for first deployment, SSL configuration, successful DNS setup, etc.—could encourage new users to explore more features and learn best practices more confidently.

These future enhancements represent a roadmap for the continued evolution of the College Companion application, addressing current limitations while expanding its value proposition for students, faculty, and the broader university community. By prioritising these enhancements based on user feedback and institutional needs, the system can continue to improve campus resource utilisation, academic support, and overall campus experience.

5.3. Conclusion

The DomainForge project represents a significant advancement in simplifying and democratizing the process of deploying, managing, and scaling web applications and domains. By providing an integrated platform that automates complex deployment workflows, service configurations, SSL provisioning, environment setup, and continuous delivery, DomainForge directly addresses the challenges faced by students, early-stage developers, and small development teams who often lack the infrastructure expertise required to manage hosting environments. The platform bridges the gap between development and deployment by offering a seamless, guided experience that transforms traditionally time-consuming and error-prone manual operations into fast, consistent, and reliable automated processes.Through its modular monolithic architecture, DomainForge achieves a balanced combination of development efficiency, maintainability, and scalability. The use of Firebase for authentication, database management, and hosting helps ensure a robust backend capable of supporting real-time operations without imposing heavy infrastructure maintenance burdens.

The frontend implementation, built with React and optimized for performance, provides an intuitive and user-friendly interaction experience. Together, these design decisions ensure that the platform is not only technically sound but also accessible to users with varying levels of deployment and DevOps knowledge.

The testing and validation phases have demonstrated that DomainForge successfully delivers dependable functionality and reliable performance across different deployment scenarios. User feedback highlights improved confidence in handling domain configurations, reduced dependency on administrative support, and increased autonomy in managing application deployments. These outcomes underscore the project's impact in enhancing workflow efficiency, promoting self-sufficiency among users, and reducing operational bottlenecks within development teams and institutional environments

Moreover, DomainForge lays a strong foundation for long-term adaptability and incremental growth. Its modular design enables the introduction of new tools, services, integrations, and automation capabilities without disrupting core workflows. The platform has been developed with future enhancements in mind, including multi-cloud deployment, predictive performance analytics, collaborative workspaces, integrated role-based access controls, and scalable orchestration technologies. These enhancements have the potential to evolve DomainForge from a deployment assistant into a comprehensive DevOps automation ecosystem
.
In conclusion, DomainForge provides a practical, efficient, and accessible solution for real-world deployment challenges. It successfully addresses the operational and technical obstacles that commonly hinder application deployment workflows, especially in educational and early-stage development environments. By prioritizing usability, automation, and scalability, the platform enhances productivity, fosters independent problem-solving, and supports continuous application lifecycle management. As digital transformation continues to expand across academic institutions, startups, and small organizations, DomainForge is positioned to play an influential role in empowering users to manage deployment workflows with greater ease, confidence, and reliability. The platform stands as a meaningful contribution toward making modern web deployment workflows more intuitive, automated, and accessible to all.

REFERENCES

1. Tokuc, K. (2024) Suitability of Micro-Frontends for an AI as a Service Platform. Doctoral dissertation, Hochschule fur Angewandte ¨ Wissenschaften Hamburg.

2. Hounsel, A., Borgolte, K., Schmitt, P., Holland, J. and Feamster, N. (2020) 'Comparing the Effects of DNS, DoT, and DoH on Web Performance', Proceedings of the ACM Internet Measurement Conference.

3. Kelly, S. and Tolvanen, J.P. (2008) Domain-specific modeling: enabling full code generation. Chichester: John Wiley & Sons.

4. K. Schomp, R. H. Lee, and A. Meske, "Providing Authoritative Answers to the World's Queries: Design and Operation of Akamai DNS," Akamai Technologies Research, 2020.

5. Romera, C.L. (2020) DNS over HTTPS traffic analysis and detection. Open University of Catalonia.

6. K. Khandaker, D. Trossen, J. Yang, and G. Carle, "On-path vs Off-path Traffic Steering, That Is the Question," Proceedings of the ACM/IEEE Conference on Internet Measurement / Networking, 2021

7. L. Degani, A. Bianchi, and F. C. Ferri, "Generative Adversarial Networks for Subdomain Enumeration," Proceedings of the ACM Conference on Computer and Communications Security (CCS) / ACM Workshop on Cyber Security (2022).

8. D. Koymans, W. Toorop and K. van Hove, "Subdomain Leakage: Investigating Exposure Methods and Malicious Exploitation," NLnet Labs / Master's Thesis / Workshop Paper, 2025.

9. Tang, R. et al. (2021) 'A Practical Machine Learning-Based Framework to Detect Compromised Hosts by Subdomain Enumeration', SecureComm.

10. Comparative analysis: S. R. Torres, "Subdomain Identification Strategies for Efficient Machine-Assisted Enumeration," SBC/Local Conference Paper / 2025, and "Comparative Analysis of Subdomain Enumeration Tools"

11. Zhang, X., Song, C., Zhao, J., Xu, Z. and Deng, X. (2022) 'Deep subdomain learning adaptation network: A sensor fault-tolerant soft sensor for industrial processes', IEEE Transactions on Neural Networks and Learning Systems, 35(7), pp. 9226-9237.

12. Souppaya, M. and Scarfone, K. (2017) Application Container Security Guide. NIST Special Publication 800-190.