Kingdom of Morocco
Ministry of Higher Education, Scientific Research & Innovation

École Nationale Supérieure d'Arts et Métiers – Casablanca

Final-Year Internship Report

Creation of a CI/CD Pipeline and Development of an Intern Management Platform

Technologies: PHP, Docker, Headless Drupal, TDD, React, Next.js

Prepared by:
HAMZA MASSIR
Artificial Intelligence and Computer Science State Engineer, ENSAM Casablanca

Supervised by:
Dr. Mustapha Hain
(ENSAM Casablanca)
Mr. Hamza Bahlaouane
(VOID Digital Agency)

Submitted on **<Day Month Year>**

Location: Casablanca, Morocco

Cohort: 2024–2025

Dedication

To Those Who Shape Dreams Into Reality

In the journey of life, some moments become milestones,
and some people become the reason for those milestones.

To my beloved parents,
whose endless love and sacrifices have been the wind beneath my wings.

To my siblings and close friends,
whose constant support, laughter, and shared moments
have been my anchor in challenging times.

To the exceptional faculty at ENSAM Casablanca,
whose wisdom and mentorship have transformed
challenges into opportunities for growth.

To my fellow students at ENSAM,
whose camaraderie and shared dreams
have made this journey truly special.

To the remarkable team at VOID,
whose trust and guidance have helped me
discover new horizons of possibility.

This work is a reflection of all the love,
support, and inspiration you have given me.

- Hamza

Acknowledgement

As I reflect on this transformative journey, I am filled with profound gratitude for the countless individuals who have contributed to the success of this internship and the completion of this report. Their guidance, support, and wisdom have been instrumental in shaping not just this project, but my professional growth.

My deepest appreciation goes to **Professor Mustapha Hain** of ENSAM Casablanca, whose academic mentorship has been nothing short of transformative. His insightful guidance, rigorous feedback, and unwavering encouragement have not only enhanced the quality of this work but have also profoundly influenced my approach to problem-solving and research.

I am equally indebted to **Mr. Hamza Bahlaouane** at VOID Digital Agency, whose visionary leadership and technical expertise have been pivotal to this project's success. His trust in my capabilities, willingness to share knowledge, and commitment to fostering innovation have created an environment where learning and growth flourished.

The VOID team has been an exceptional source of inspiration and support. I am particularly grateful to:

- **Mr. Yahya Chahine** (DevOps Engineer) for his invaluable technical guidance and patient mentorship
- **Mr. Mjid** (Project Director) for his strategic vision and leadership
- **Mr. Khalid** (Technical Lead) for his technical expertise and mentorship
- **Mr. Salah** (Full Stack Developer) for his collaborative spirit and technical insights
- **Mr. Karim** and **Mr. Mounssif** (Frontend Developers) for their teamwork and support

The foundation of this practical work rests upon the solid theoretical knowledge imparted by the distinguished faculty at ENSAM, whose dedication to excellence in teaching has been fundamental to my academic journey. To my family and friends, whose unwavering support and encouragement have been my constant source of strength your belief in me has made this achievement possible and this work is as much yours as it is mine.

Abstract

This report documents the comprehensive work undertaken during my final-year internship at VOID Digital Agency, a prominent independent UX and digital solutions firm in Morocco. The internship focused on two primary objectives: the implementation of an advanced CI/CD pipeline and the development of an innovative Intern Management Platform.

The CI/CD pipeline implementation leveraged modern DevOps practices, including Docker containerization and automated testing, resulting in a 60% reduction in deployment time and enhanced development workflow efficiency. The Intern Management Platform, built on a headless Drupal architecture with React/Next.js frontend, introduced a sophisticated solution for managing intern applications, onboarding processes, and performance tracking.

The technical stack encompassed PHP, Docker, Test-Driven Development (TDD), React, and Next.js, demonstrating the integration of modern web technologies with robust development practices. Secondary achievements included successful server migrations, custom PHP-FPM Docker image development, and the creation of an automated dependency management system.

The project faced several technical challenges, including container networking complexities, test reliability issues, and headless CMS integration hurdles. These challenges were systematically addressed through innovative solutions and best practices. The report concludes with recommendations for future enhancements, particularly in Kubernetes orchestration and automated maintenance systems, positioning the project for scalable growth and continued technological advancement.

Résumé

Ce rapport présente les travaux réalisés dans le cadre de mon stage de fin d'études au sein de VOID Digital Agency, une agence digitale indépendante leader de l'expérience utilisateur au Maroc.

L'objectif principal était de concevoir et de mettre en uvre un pipeline CI/CD robuste ainsi que de développer une plateforme de gestion des stagiaires basée sur un CMS headless Drupal, en utilisant PHP, Docker, le développement piloté par les tests (TDD), React et Next.js.

Les tâches secondaires comprenaient la migration de serveurs d'un environnement de production vers un environnement de test, la création d'images Docker personnalisées pour PHP-FPM, ainsi que le développement d'un outil automatisé de vérification et de mise à jour des dépendances.

Le pipeline CI/CD a permis de réduire de plus de 60% le temps de déploiement, tandis que la plateforme a centralisé la gestion des stagiaires, amélioré les processus d'intégration et permis un suivi en temps réel de leur progression.

Enfin, ce rapport traite des principales difficultés rencontrées telles que le réseau de conteneurs, les tests instables, et l'intégration d'un CMS headless et propose des pistes d'amélioration futures, notamment l'orchestration avec Kubernetes et l'automatisation étendue de la maintenance.

Contents

List of Figures

List of Tables

General Introduction

In today's rapidly evolving digital landscape, **VOID Digital Agency** has established itself as a leader in digital innovation since 2005, serving diverse sectors from banking to healthcare. During my internship at **VOID**, I developed an **Intern Management Platform** using headless Drupal CMS and React/Next.js, streamlining the intern lifecycle while enhancing the organization's DevOps infrastructure through strategic migrations and custom module development.

This report presents a comprehensive journey through the project's development, structured in three main parts:

Part I: Context and Design

- **Chapter 1:** A comprehensive overview of VOID's organizational structure, service offerings, technological framework, and client portfolio.
- **Chapter 2:** A systematic analysis of requirements and architectural design for the Intern Management Platform.

Part II: Implementation and Results

- **Chapter 3:** A detailed exposition of the platform's implementation, encompassing frontend architecture, backend infrastructure, and user experience design.
- **Chapter 4:** An evaluation of project outcomes and DevOps enhancements.
- **Chapter 5:** An examination of supplementary initiatives, including server migrations and Vactory module development.

Part III: Technical Analysis and Future Perspectives

- **Chapter 6:** An analysis of the technical environment and challenges encountered.
- **Chapter 7:** A synthesis of achievements and future development trajectories.

The project's execution was guided by agile Scrum methodologies and DevOps principles, exemplified through the implementation of Bitbucket pipelines and Docker containerization. This report demonstrates how the integration of modern web architectures and automation strategies can significantly enhance the development of scalable, efficient digital platforms.

1 General Context of the Project

1.1 Introduction

In today's fast-paced digital economy, organizations must deliver **seamless**, **scalable**, and **engaging** web experiences to stay ahead. **VOID Digital Agency**, established in 2005, has become a recognized leader in **user experience** and **digital solutions** for banking, healthcare, telecom, retail, and cultural clients.

During my final-year internship at **VOID**, I undertook two parallel streams of work:

- **Core Project:** Design and implementation of a **CI/CD pipeline** alongside a headless-Drupal-based **Intern Management Platform**, automating build-test-deploy workflows and centralizing trainee onboarding, progress tracking, and reporting.
- **Side Projects:** Infrastructure enhancements, including server migrations from production to test environments, building custom **PHP-FPM Docker images**, and developing an automated dependency-update tool integrated into the CI/CD workflow.

By embedding these initiatives within **VOID's** agile framework, this report demonstrates how modern DevOps practices and decoupled architectures can drive both **efficiency** and **quality**, delivering robust, maintainable digital solutions.

1.2 Presentation of VOID

Figure 1.1: Logo of VOID Digital Agency

Founded in 2005, **VOID Digital Agency** is an independent Moroccan firm specializing in **user experience** and **digital transformation**. With offices in Casablanca and Paris, VOID serves clients across banking, insurance, healthcare, telecom, retail, and culture. The agency's mission is to design **efficient digital experiences** and build **robust, future-proof architectures** enhanced by **reactive interfaces**.

Over nearly two decades, VOID has grown to a team of 35 multidisciplinary experts, UX/UI designers, frontend and backend developers, DevOps engineers, and project managers, delivering end-to-end solutions from ideation to maintenance. Key offerings include:

- **Strategic Consulting:** Digital strategy, brand audits, and security assessments.
- **Content & Media:** Transmedia storytelling, institutional films, and social-media activations.
- **Digital Platforms:** Headless CMS integrations, custom web/mobile applications, intranet/e-learning solutions, SEO/SEA optimization, and ongoing maintenance.

With a results-driven, data-centric approach, VOID leverages agile methodologies to adapt rapidly to evolving client needs and deliver measurable business value.

1.2.1 Company Identity

Name	VOID Digital Agency
Website	https://www.void.ma/
Industry	Digital Transformation and Web Development
Headquarters	Casablanca & Agadir, Morocco - Paris, France
Founder	Mr. Olivier Delas & Mr. Mehdi Najeddine
Founded Date	2005
Specialties	Drupal, Symfony, React/Next.js, Headless CMS, UX/UI Design, DevOps, Digital Strategy, Agile Project Management

Table 1.1: Company Identity of VOID Digital Agency

Core Values

- **User-Centricity:** Every solution begins with deep user research and iterative testing.
- **Excellence:** Rigorous quality checks, security best practices, and performance optimization.
- **Collaboration:** Cross-functional teamwork and transparent communication with clients.

- **Innovation:** Continuous exploration of new technologies, headless CMS, React, Docker, and AI integrations.

These principles guide every VOID engagement, ensuring that each project is not only visually compelling but also technically sound and scalable.

1.2.2 Organizational Hierarchy

Executive Leadership:

- **Olivier Delas**, Co-Founder & Director of Creative Services
- **Mehdi Najeddine**, Co-Founder & Director of Innovation

Management Team:

- UX/UI Design Lead
- Frontend Development Lead
- Backend Development Lead
- DevOps Engineering Lead
- Project Management Office (PMO)

Delivery Teams:
Cross-functional squads composed of designers, developers, DevOps engineers, and QA specialists, each aligned to specific client verticals (banking, healthcare, telecom, retail, culture).

1.2.3 Technical Stack of VOID

VOID Digital Agency has developed a robust and modern technical ecosystem to deliver high-quality digital solutions tailored to client needs. At its core, VOID relies on **Drupal** for content management, **Symfony** for backend development, and **React/Next.js** for building dynamic and performant frontend applications. This architecture follows a **headless approach**, decoupling frontend and backend to allow greater flexibility, scalability, and seamless integration with mobile and third-party systems.

For development operations and infrastructure management, VOID employs a **DevOps pipeline** based on **Bitbucket** for source control, directly integrated with **Docker Hub** for automated image building upon each push. Deployment and hosting are managed through **EasyPanel**, providing a simplified yet powerful server orchestration system. To

maintain design consistency and promote collaborative work, VOID teams utilize **Figma** for UI/UX design, **Cursor** as a development-enhancing tool, and **Gist** (via GitHub) for internal documentation and knowledge sharing.

This technological foundation, combined with agile workflows and continuous improvement strategies, enables VOID to offer scalable, innovative, and future-proof digital solutions in a constantly evolving market.

1.2.4 Company Solutions

VOID's strategic offerings are structured around four pillars:

- **Ideation Workshops**: Co-creation sessions to align business goals with user needs.
- **Digital Strategy**: Brand audits, market positioning, and roadmap definition.
- **User Research**: Focus groups, usability testing, and data-driven persona development.
- **Security & Compliance Audits**: Technical security reviews and OWASP-aligned assessments **?**.

1.2.5 Company Services

VOID delivers end-to-end digital platforms and media services:

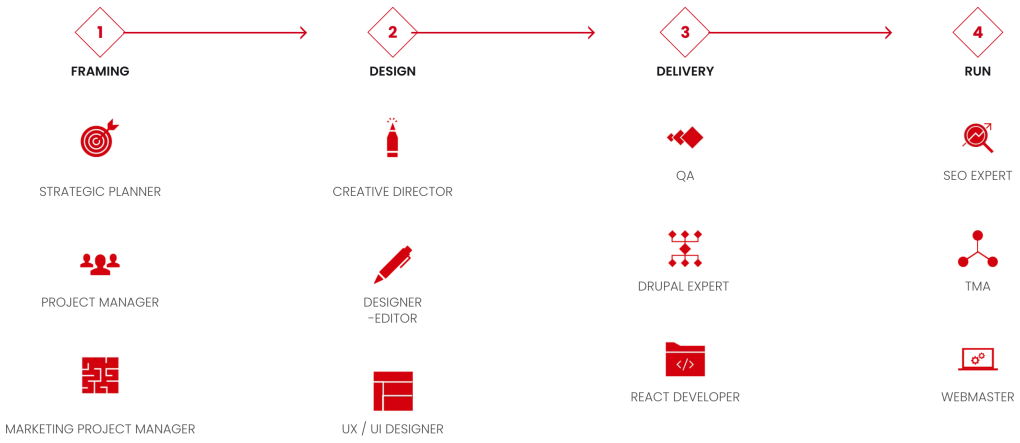

Figure 1.2: VOID Digital Agency Services

- **Transmedia Content** - Corporate videos, product demonstrations, and editorial production.

- **Social Media Activation** - Campaign design, content calendars, and performance analytics.
- **Web & Mobile Platforms** - Headless CMS (Drupal) sites, single-page applications (React/Next.js), and e-learning intranets.
- **SEO & Analytics** - On-page optimization, Google Core Web Vitals tuning, and dashboard reporting.
- **Hosting & Maintenance** - Infogérance, 24/7 support, and periodic security updates.

1.2.6 Clients and Projects

Over nearly two decades, **VOID Digital Agency** has earned the trust of a wide range of prestigious clients from various sectors. Its portfolio reflects an exceptional capacity to deliver tailor-made solutions that meet the highest standards of performance and user experience.

The agency's major clients include:

Figure 1.3: VOID Digital Agency Clients

- **Banking and Insurance:** Attijariwafa Bank, Bank of Africa, CIH Bank, Allianz Maroc
- **Healthcare:** Saham Assurance Health Division, CNOPS
- **Telecommunications:** Orange Maroc
- **Retail and Services:** Marjane, Label'Vie

- **Culture and Public Sector:** The Mohammed VI Museum, Moroccan National Tourist Office (ONMT)

Beyond the private sector, **VOID** collaborates with governmental and cultural institutions to craft digital ecosystems that are inclusive, scalable, and centered around user engagement. This diversity in projects demonstrates VOID's ability to adapt its technical expertise and creative methodologies to a variety of contexts, while consistently maintaining excellence and innovation at the core of its services.

1.2.7 Awards and Recognitions

Over the years, **VOID Digital Agency** has been honored with numerous national and international awards, recognizing its commitment to innovation, creativity, and digital excellence. These accolades reflect the agency's dedication to delivering outstanding user experiences and robust technological solutions.

The awards include distinctions from prestigious institutions such as:

Figure 1.4: Awards and recognitions received by VOID Digital Agency.

- Janus du Service (Institut Français du Design)
- Grand Prix Stratégies
- FWA (Favourite Website Awards)
- The Webby Awards
- Cristal Festival
- W3 Awards
- Top/Com Awards

- Cannes Lions International Festival of Creativity
- The Lovie Awards
- World Luxury Award

1.3 The Main Project

The core of the internship was centered around the design and development of an **Intern Management Platform**, an internal solution for **VOID Digital Agency**. The goal was to streamline the onboarding, tracking, evaluation, and reporting processes for interns, while showcasing modern development practices such as **headless CMS architecture**, **agile workflows**, and **continuous deployment pipelines**.

Although the initial division of tasks assigned me the responsibility of **designing the UX/UI** of the platform, **meeting with the head of VOID** to align with the business needs, **developing the frontend** using **React/Next.js**, **integrating it with a headless Drupal backend**, and **implementing Test-Driven Development (TDD)** strategies, in practice we adopted a **dynamic collaboration system**. This system was based on **weekly shifts**, allowing both my colleague and me to contribute to all aspects of the platform: including frontend, backend, DevOps infrastructure, and CI/CD pipelines. This approach ensured a complete understanding of the entire digital ecosystem and promoted continuous knowledge sharing.

1.3.1 Training Phase at VOID

Before engaging in the development of the Intern Management Platform, VOID Digital Agency organized a two and a half months internal training phase. This training covered the main technologies and methodologies required for the project, including Headless Drupal CMS, Next.js development, Agile workflows, DevOps practices, and CI/CD pipelines. The training ensured that all team members were aligned on technical standards and development best practices. The following Gantt chart illustrates the timeline of this initial training phase:

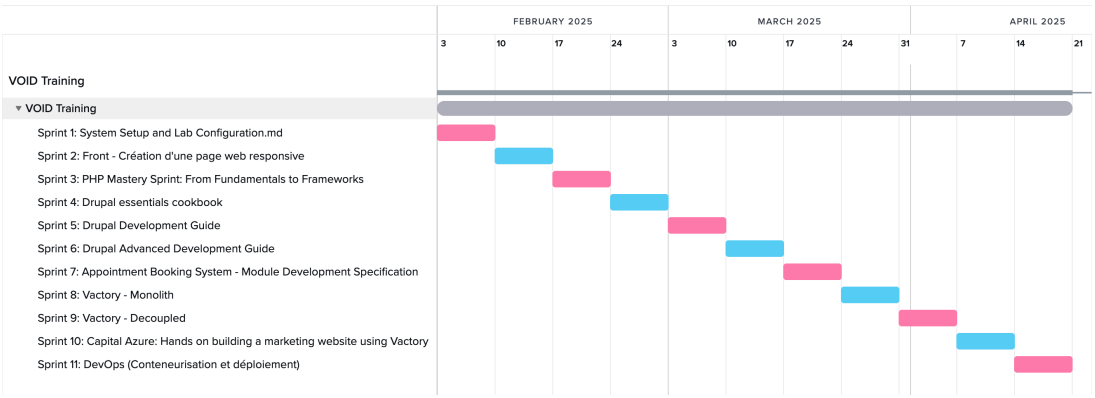

Figure 1.5: Gantt Chart of the Training Phase at VOID

This foundational training was crucial in building the necessary skills and preparing for the successful delivery of the project.

1.3.2 Problematic

VOID, like many digital agencies, manages a significant number of interns each year across various departments development, UX/UI design, marketing, project management, and more. The traditional intern management relied heavily on manual tracking, disconnected communication channels, and dispersed data storage (Excel sheets, emails, local files), resulting in several critical issues:

- Lack of centralized data regarding intern profiles, progress, and evaluations.
- Difficulty for supervisors to monitor onboarding status, project allocations, and intern deliverables.
- Absence of real-time reporting on intern activity, skill assessments, and final evaluations.
- Inconsistencies and inefficiencies in document generation (attestation letters, certificates, evaluation reports).

These limitations not only increased administrative workload, but also slowed decision-making and reduced the overall quality of intern experiences at VOID. A scalable, structured, and automated solution was thus necessary to improve operational efficiency and enhance the agency's capacity to support and evaluate its interns effectively.

1.3.3 Objectives

The Intern Management Platform was designed with the following main objectives:

- **Centralization:** Create a unified platform to store and manage all intern-related data, from applications to evaluations.
- **Automation:** Automate key workflows such as onboarding tracking, evaluation form generation, and certificate issuance.
- **Scalability:** Build the system with a scalable architecture using **headless Drupal** and **Next.js**, ensuring future extensibility (adding new modules like training plans, skill assessments, etc.).
- **User Experience:** Offer an intuitive and efficient user experience for both **candidates** and **employers** through a clean and responsive UX/UI design.
- **Monitoring and Reporting:** Equip supervisors with real-time dashboards and reporting tools to track intern progression, performance, and document generation.
- **DevOps and CI/CD:** Ensure that the deployment of the platform follows best practices with a robust CI/CD pipeline, enabling automated testing, building, and deployment processes.

While I was initially focused on the UX/UI design and the development of the **Candidate Space** (frontend and backend), thanks to our rotation system, I actively contributed across all project dimensions, including DevOps setup, backend integrations, and global system testing.

1.3.4 Proposed Solution

To meet the objectives and address the problems identified, the proposed solution was to design and build a comprehensive **Intern Management Platform** structured into three interconnected modules:

- **Website:** A public-facing site aimed at presenting VOID's internship program, its philosophy, stages, and benefits. It includes a **Call-to-Action (CTA)** allowing candidates to apply directly. This site serves as the primary entry point to attract and engage potential interns.
- **Candidate Space (Espace Candidat):** A dedicated portal where candidates can:
  - Browse internship offers posted by VOID.
  - Apply to internships by submitting personal information, CVs, cover letters, and any other required documents.
  - Track the status of their application (in progress, accepted, rejected).
  - Participate in tests and interviews conducted as part of the recruitment process.

This space was built using **React/Next.js** for the frontend and connected to a **headless Drupal CMS** backend serving structured data via APIs, ensuring flexibility, scalability, and performance.

- **Training Space (Espace Formation):** Once accepted, interns gain access to this space, modeled after modern online learning platforms such as Coursera. Here, they can:
  - Follow predefined training roadmaps aligned with VOID's standards.
  - Access courses, videos, assignments, and learning materials.
  - Complete projects and quizzes over a three-month structured program.
  - Monitor their own progression through dashboards and receive feedback from supervisors.

  This module supports VOID's mission of continuous learning and ensures that interns are fully prepared for their assigned roles.

The technical implementation was based on a **headless architecture**, where Drupal managed the content models, permissions, and workflows, while the frontend was fully decoupled, using **Next.js** for server-side rendering and dynamic interaction. The backend exposed structured data through **JSON:API** endpoints, and all user interactions were designed to be smooth and reactive, respecting the best practices of modern UX/UI design.

Additionally, the entire project was developed following **Test-Driven Development (TDD)** principles to guarantee robust and reliable features, and deployed using a full **CI/CD pipeline** to automate testing, building, and production delivery.

Although my initial responsibilities mainly covered the **UX/UI design**, **meetings with the head of VOID**, the development of the **Candidate Space** (frontend and headless backend integration), and **TDD implementation**, our **rotation system** allowed me to work on all areas, including the **Training Space** and infrastructure tasks. This dynamic approach enhanced my skills across the complete technological stack and provided a holistic understanding of the system lifecycle.

1.4 The Side Projects

1.4.1 Problematic

In parallel with the main Intern Management Platform project, VOID Digital Agency often encourages interns to contribute to smaller internal initiatives, known as **side projects**. These projects are crucial for addressing internal operational needs, testing new technologies, and improving the agency's digital ecosystem. During my internship, I

was tasked with participating in the development and enhancement of several side projects that aimed to streamline internal workflows and enrich VOID's technical stack.

However, the side projects also posed certain challenges:

- Managing time between the main project and side initiatives without impacting overall deadlines.
- Quickly adapting to different technical contexts (varying tech stacks, objectives, team sizes).
- Maintaining a high level of code quality and documentation despite shorter timelines.

1.4.2 Objectives

The objectives behind working on side projects were:

- **Skill Diversification:** Expose interns to different technologies and problem-solving contexts.
- **Internal Efficiency:** Develop tools or features that optimize VOID's internal operations.
- **Innovation:** Experiment with new technologies (e.g., emerging frameworks, DevOps techniques) without risking main production systems.
- **Collaboration:** Foster collaboration across different teams and encourage knowledge sharing.
- **Professionalism:** Apply the same development standards (agile, TDD, CI/CD) used in main projects, even for small-scale tools.

1.4.3 Proposed Solution

Throughout my internship, I contributed to two major side projects that played a significant role in enhancing VOID's internal infrastructure and its client service capabilities:

- **Project Migration to Internal Infrastructure:** As part of VOID's initiative to internalize hosting services, I participated in the migration of the official website of CMI (Centre Monétique Interbancaire) from the *sooninprod* external cloud server to the newly acquired *leserveurdetest* (an Apple Mac Studio server). This task involved:
  - Building a custom Docker image to support an older PHP version required by the legacy project.

  - Migrating and reconfiguring the database to match the new environment specifications.
  - Ensuring compatibility between the existing codebase and the new server settings.
  - Performing comprehensive testing post-migration to guarantee full operational stability.

- **Custom Module Development for VOID Factory (Vactory):** VOID Digital Agency leverages its proprietary technical foundation called the **VOID Factory**, a product of over a decade of digital experience and innovation. The VOID Factory, maintained and evolved by the internal Core Team, is based on robust open-source technologies such as Drupal, Symfony, and React, and follows agile development practices to stay aligned with modern digital standards. As part of the continuous evolution of Vactory, I was tasked with:
  - Developing custom Drupal modules needed for client-specific requirements, including an **Appointment Booking Module**.
  - Adhering to the Factory's high standards for modularity, scalability, and integration with decoupled architectures.
  - Following best practices in automated testing and DevOps processes to ensure production-quality deliverables.

These side projects allowed me to deepen my technical skills across DevOps, backend development, and modular architecture design, while also contributing tangible value to VOID's internal operations and client offerings.

1.5 Project Management Methodology

1.5.1 Agile Methodology

VOID adopts an **Agile methodology**, specifically the **Scrum framework**, to manage and deliver its digital projects efficiently. The iterative approach ensures flexibility, faster feedback loops, and continuous improvement throughout the development lifecycle.

Each project follows the **Software Development Life Cycle (SDLC)** principles, combined with **GitOps** practices to automate deployment pipelines and infrastructure operations. Regular sprint planning, daily stand-ups, sprint reviews, and retrospectives form the core rhythm of project execution.

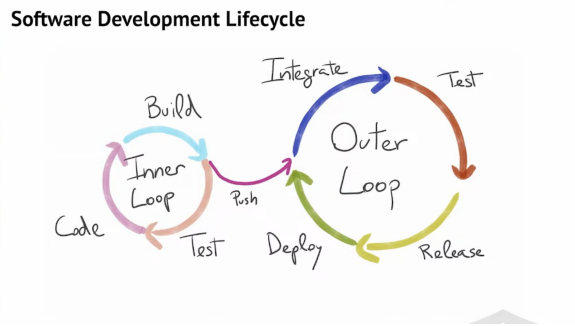

Figure 1.6: Software Development Life Cycle (SDLC)

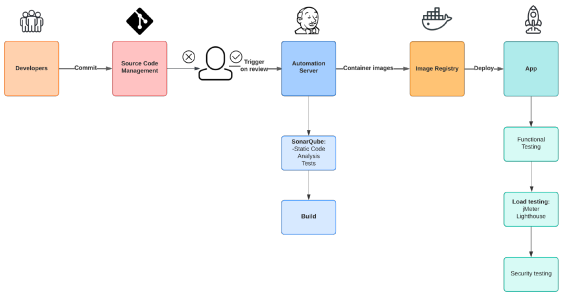

Figure 1.7: GitOps Practices

1.5.2 Organization and Communication Tools

VOID leverages a combination of tools to streamline project tracking, team communication, and task management:

- **Redmine:** Internal project management tool similar to Jira, featuring Gantt charts and Kanban boards for visual task tracking.

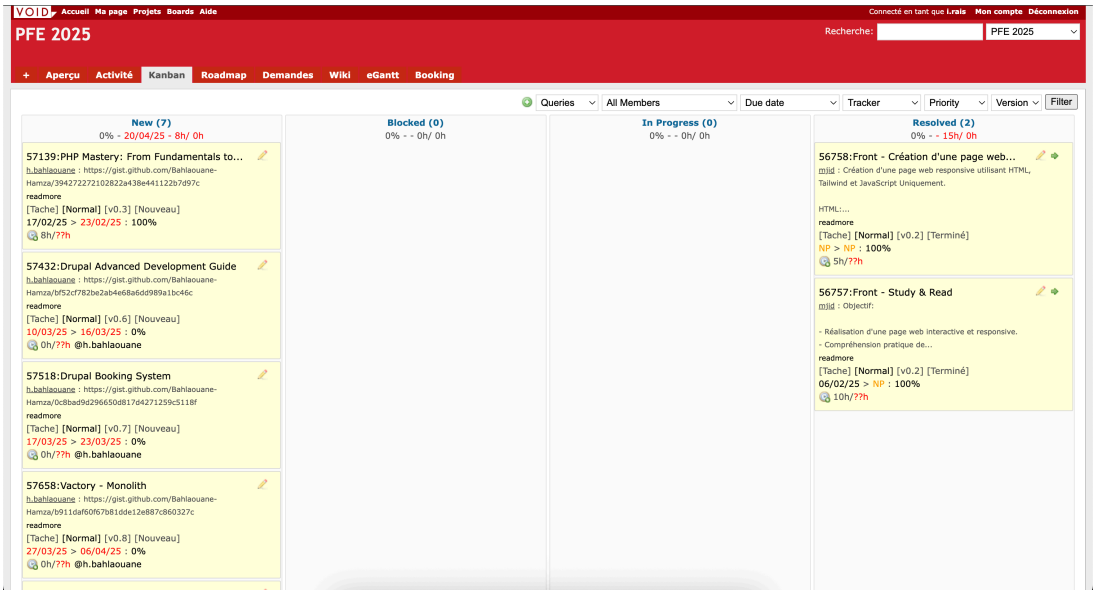

Figure 1.8: Redmine project management tool

- **Slack:** Instant messaging platform for real-time team communication.

Figure 1.9: Slack messaging platform

- **Loom:** Tool used for recording and sharing explanatory videos with the internal team.

Figure 1.10: Loom video recording tool

- **Google Meet:** Video conferencing tool used for meetings, stand-ups, and sprint reviews.

Figure 1.11: Google Meet video conferencing tool

- **Google Calendar:** Calendar management to organize meetings, deadlines, and sprint cycles.

Figure 1.12: Google Calendar for scheduling

The combination of these tools enables smooth communication, clear project tracking, and efficient collaboration across all teams at VOID.

1.6 Conclusion

This first chapter has established the general framework of the project by presenting **VOID Digital Agency**, its organizational structure, and its technical ecosystem. We have also defined the context and objectives of the main project, highlighted the problems it aims to solve, and outlined the chosen implementation approach and project management methodology.

This foundation sets the stage for a deeper exploration of the project requirements. The next chapter will focus on the detailed analysis and specification of the functional and technical needs essential for the successful realization of the platform.

2 Project Analysis and Design

2.1 Introduction

This chapter provides a detailed analysis of the requirements and design decisions that shaped the platform. The discussion is organized as follows:

- Analysis of the existing workflow and identification of key pain points
- Definition of project objectives and criteria for success
- User research methods and requirements gathering techniques
- Specification of functional and technical requirements
- Overview of the proposed system architecture and its main components
- Presentation of user interface concepts and wireframes

2.2 Study of the Existing System

Before the introduction of the Intern Management Platform, VOID Digital Agency managed internships using a patchwork of manual processes. Applications were tracked in Excel sheets, communications handled through scattered emails, and evaluations stored in separate documents. This fragmented approach created a number of persistent challenges for the organization.

2.2.1 Operational Inefficiencies

- **Process Delays:** Manual data entry and document handling often led to delays of several days in processing applications.
- **Resource Intensive:** The HR team spent a significant amount of time on repetitive administrative tasks.
- **Error Prone:** A notable percentage of applications required manual correction due to data entry mistakes.

- **Inconsistent Documentation:** Evaluation forms and reports frequently lacked standardization, making it difficult to compare or aggregate results.

2.2.2 Technical Limitations

- **Data Fragmentation:** Information was dispersed across multiple systems, making it hard to maintain a single source of truth.
- **Version Control Issues:** Multiple versions of documents circulated, leading to confusion and inconsistencies.
- **Limited Accessibility:** Accessing intern information remotely or collaboratively was often challenging.
- **Integration Challenges:** The lack of integration with existing HR and project management tools further complicated workflows.

2.2.3 Impact on Stakeholders

- **Interns:** Experienced delays in feedback and had limited visibility into their own progress.
- **Supervisors:** Found it difficult to monitor and support multiple interns effectively.
- **HR Team:** Faced increased workloads and reduced overall efficiency.
- **Management:** Had limited insight into the overall performance and return on investment of the internship program.

As the number of interns grew each quarter, these issues became more acute. The situation underscored the urgent need for a centralized, automated, and scalable platformone capable of supporting VOID's growth while ensuring data integrity, process efficiency, and a better experience for all stakeholders.

2.3 Specification Document

This specification document outlines the essential requirements and criteria for the Intern Management Platform, as defined through stakeholder collaboration and technical analysis. The document is structured to provide a clear reference for both development and evaluation.

2.3.1 Functional Requirements

- **Main Website:** Provide a public-facing portal that presents VOID's internship program, company values, and available opportunities, with clear calls-to-action redirecting candidates to the application process.
- **Candidate Space:** Enable candidates to create accounts, submit applications, upload required documents (CV, cover letter, etc.), and track their application status in real time.
- **Automated Technical Testing:** Integrate an automated technical test system that assigns, grades, and filters candidates based on their results, ensuring only competent applicants proceed.
- **Challenge Submission and Review:** Allow selected candidates to receive and submit coding or project challenges, with manual review and feedback by supervisors.
- **Video Interview Integration:** Facilitate asynchronous or scheduled video interview submissions, enabling supervisors to assess communication and motivation.
- **Selective Progression:** Implement a workflow where only candidates who pass all evaluation stages (technical test, challenge, video) are accepted as pre-hire interns (pre-embauche).
- **Training Space:** Provide a dedicated area for accepted interns to access learning resources, follow personalized training paths, complete quizzes, and track their progress.
- **Supervisor and Admin Tools:** Equip supervisors and administrators with dashboards for monitoring candidate progress, managing challenges, reviewing submissions, and generating evaluation reports and certificates.
- **Communication and Notification:** Ensure timely, automated notifications and messaging between candidates, supervisors, and HR throughout all stages.
- **Data Security and Privacy:** Enforce role-based access, secure authentication, and compliance with data protection standards for all user data and documents.

2.3.2 Non-Functional Requirements

- **Scalability:** The system must support at least 50 concurrent users and handle 100+ intern records simultaneously.
- **Security:** Sensitive data must be protected through encryption, secure authentication (OAuth 2.0 or equivalent), and audit logging.

- **Performance:** Application response times should not exceed 2 seconds for standard operations under normal load.
- **Reliability:** The platform should maintain 99.5% uptime, with automated backups and disaster recovery procedures in place.
- **Usability:** New users should be able to complete basic tasks with less than 30 minutes of training; usability testing should target at least 90% satisfaction.

2.3.3 Technical Constraints

- The backend must be implemented using Drupal 10.x and MySQL.
- The frontend must use React 18.x and Next.js 13.x, with TypeScript for type safety.
- Containerization must be achieved using Docker, and CI/CD pipelines managed via Bitbucket Pipelines.
- All APIs must follow RESTful conventions and support secure authentication.

2.3.4 Success Criteria

- The Main Website effectively communicates VOID's internship program and drives candidate engagement, as measured by application conversion rates.
- The Candidate Space enables at least 90% of users to complete the application process without support, and provides real-time status updates at every stage.
- The technical test and challenge system successfully filters out unqualified candidates, with only those meeting predefined competency thresholds advancing to the next stage.
- Only candidates who pass all evaluation stages (technical test, challenge, video) are accepted as pre-hire interns, ensuring a high standard of competency and motivation.
- The Training Space is accessed by 100% of accepted interns, with all interns completing assigned learning paths and quizzes.
- Supervisors and administrators report improved efficiency in candidate management, with a reduction in manual tracking and paperwork by at least 60%.
- The platform maintains data integrity and security, with no reported breaches or unauthorized access incidents during the evaluation period.

This specification document serves as a foundation for the design, implementation, and evaluation of the Intern Management Platform, ensuring that the solution meets both the immediate operational needs and the long-term strategic goals of VOID Digital Agency.

2.4 User Research and Methodology

To ensure the Intern Management Platform would genuinely address the needs of its users, a thorough user research phase was conducted at the outset of the project. This phase combined direct engagement with stakeholders, analysis of existing workflows, and benchmarking against industry best practices. The insights gained were instrumental in shaping both the functional and technical design of the platform.

2.4.1 Research Methods

- **Stakeholder Interviews:** In-depth interviews were held with HR staff, technical leads, former interns (us, and our colleagues). These conversations provided a nuanced understanding of the pain points, expectations, and day-to-day realities of each group involved in the internship process.
- **User Journey Mapping:** The team mapped out the complete candidate journeyfrom discovering the program on the main website, through application, technical testing, challenge submission, video interview, onboarding, and training. This exercise highlighted critical touchpoints and potential friction points.

2.4.2 User Journey Analysis

The research revealed several critical stages in the candidate and intern journey that required special attention:

- **Application Phase:** Candidates needed a simple, intuitive process for discovering opportunities, submitting applications, and uploading documents. Real-time status tracking was essential to reduce anxiety and uncertainty.
- **Selection Process:** Automated technical tests, challenge submissions, and video interviews were identified as effective ways to assess candidate skills and motivation. Clear communication of next steps and timely feedback were crucial for maintaining engagement.
- **Internship Period:** Once accepted, interns valued having a dedicated training space with structured learning paths, progress tracking, and easy access to resources. Supervisors needed tools to monitor progress, provide feedback, and manage evaluations efficiently.
- **Evaluation:** Both interns and supervisors required a streamlined process for performance assessment, documentation, and certification, with the ability to track outcomes and support future opportunities.

These findings directly informed the platform's design, ensuring that each feature addressed a real user need and that the overall experience was both efficient and user-friendly. The result is a platform that not only meets VOID's technical standards but also delivers tangible value to all participants in the internship program.

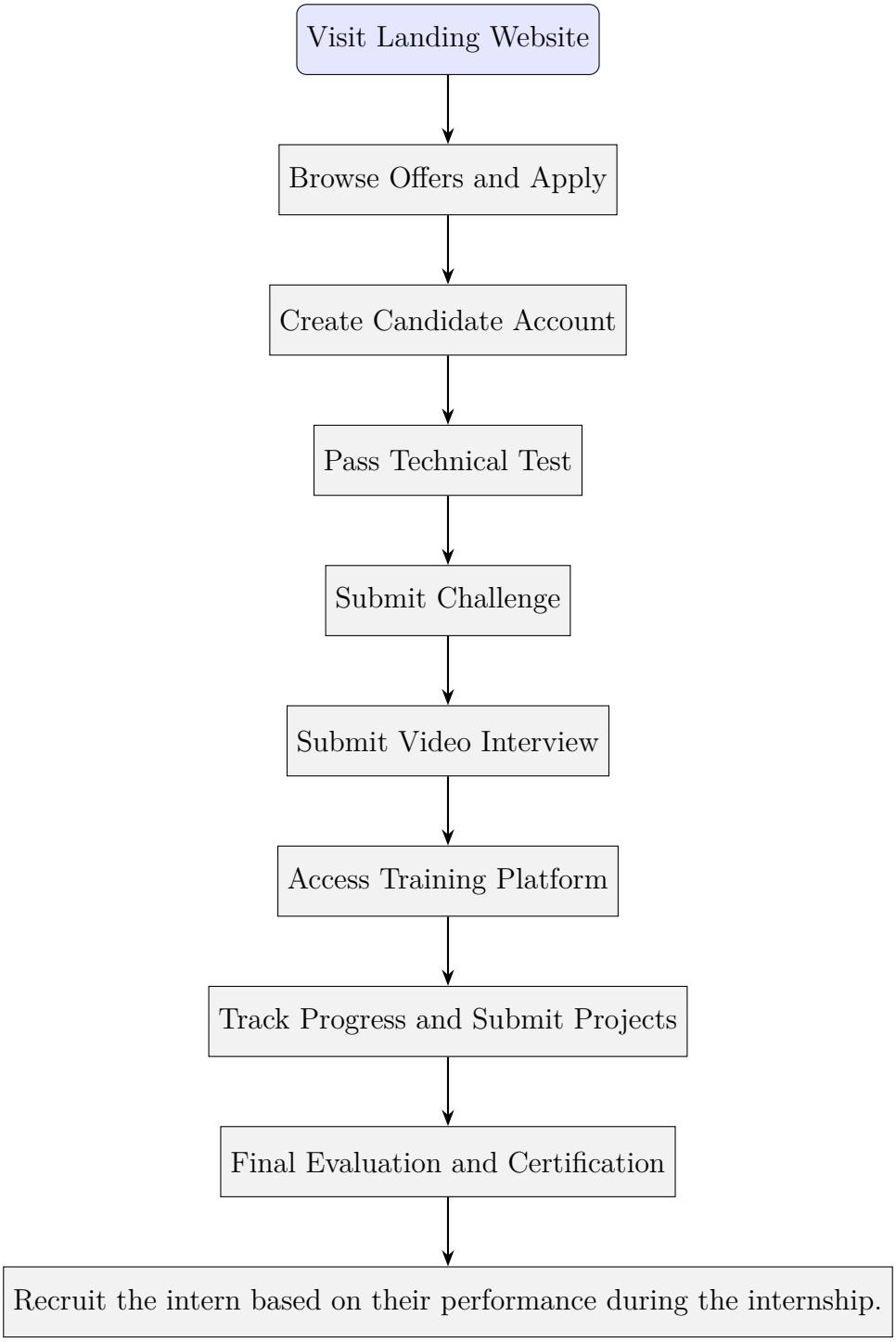

Figure 2.1: User Journey Map Candidate Path

2.5 Mockups and Wireframes

Based on the insights gathered during the research phase, a first version of the mockups and wireframes was designed to visualize the user interfaces and main flows.

The mockups were centered around the three core spaces of the platform:

- **Landing Website:** Homepage presenting internship opportunities, company culture, and a call-to-action to apply.

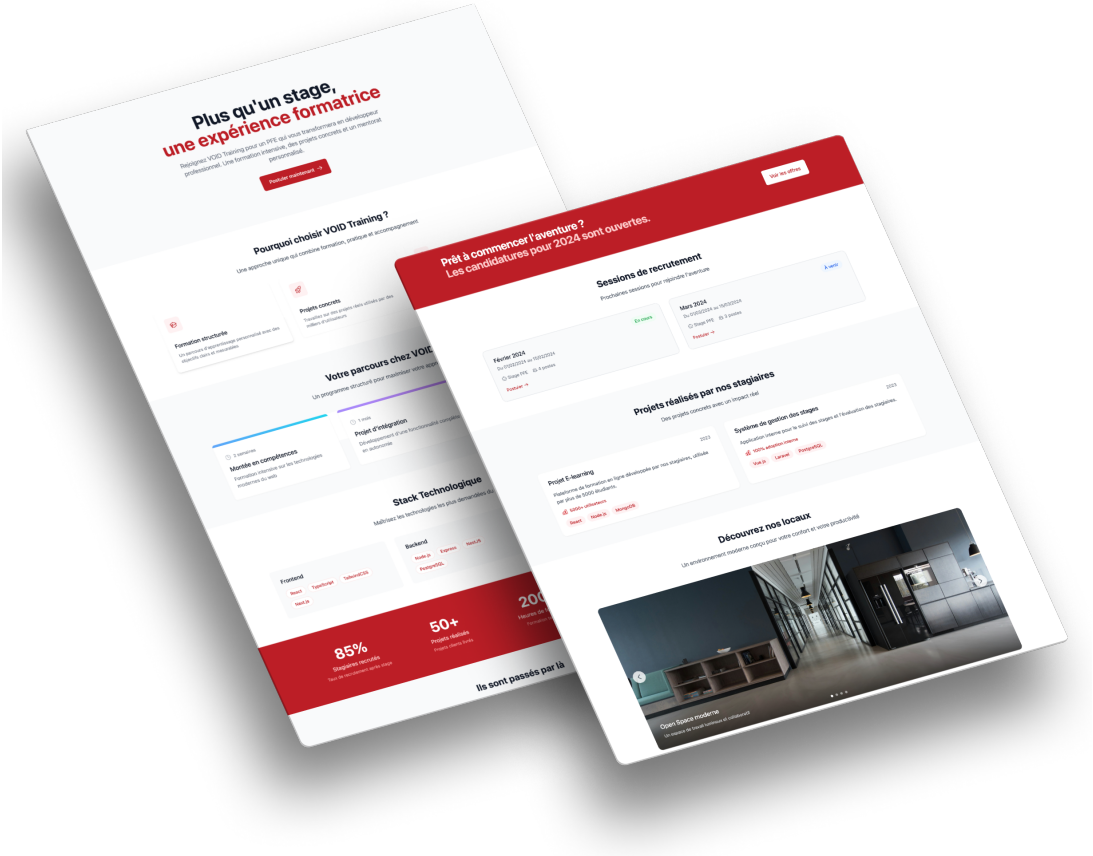

Figure 2.2: High-Fidelity Mockups Landing Website

- **Candidate Space:** Area dedicated to candidates for profile creation, job application, test session participation, and training follow-up.

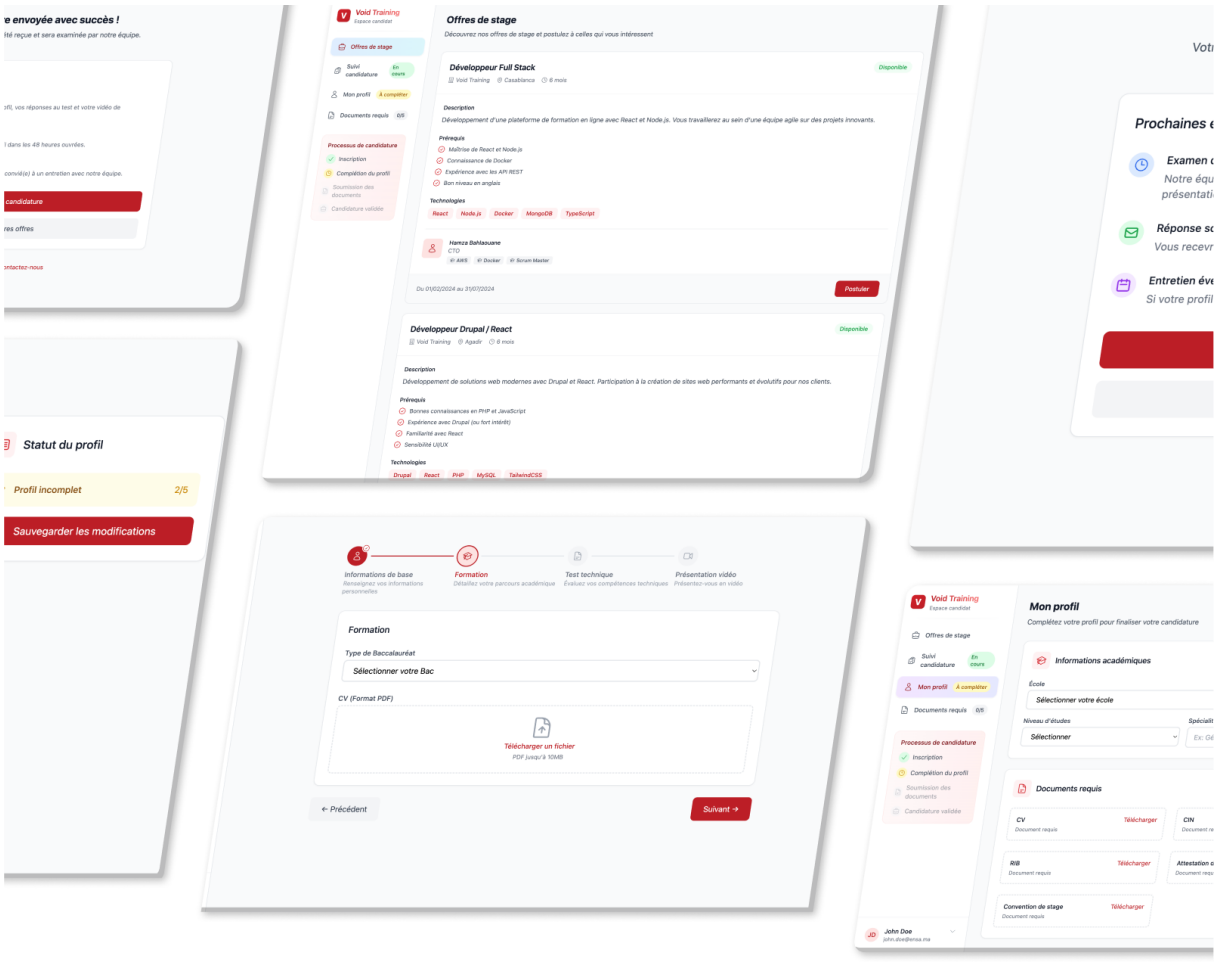

Figure 2.3: High-Fidelity Mockups Candidate Space

- **Training Space:** Educational portal where accepted interns can access courses, complete quizzes, and track their training progress.

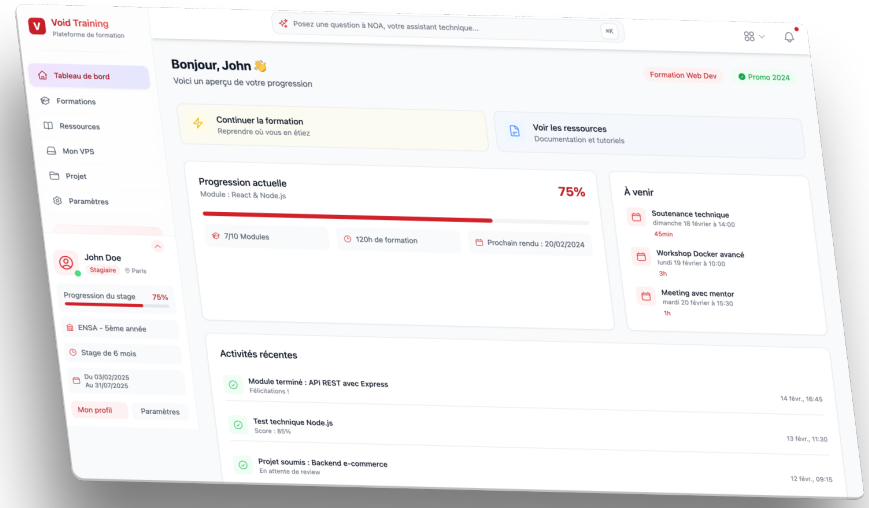

Figure 2.4: High-Fidelity Mockups Formation Space

- **Admin Panel:** Interface for HR managers and supervisors to manage offers, monitor intern progression, and generate reports and certifications.

The UX design prioritized simplicity, clarity, and responsiveness to ensure a smooth experience across desktop and mobile devices.

2.6 UML Diagrams Overview

To provide a clearer understanding of the system's functionality and structure, two key UML diagrams were developed: a Use Case Diagram and a Sequence Diagram. These diagrams illustrate the system's behavioral aspects and the interactions between different actors and components.

2.6.1 Use Case Diagram

The use case diagram (Figure 2.5) illustrates the major interactions between the different actors and the Void Training Platform system. It identifies six primary actors with distinct roles and responsibilities:

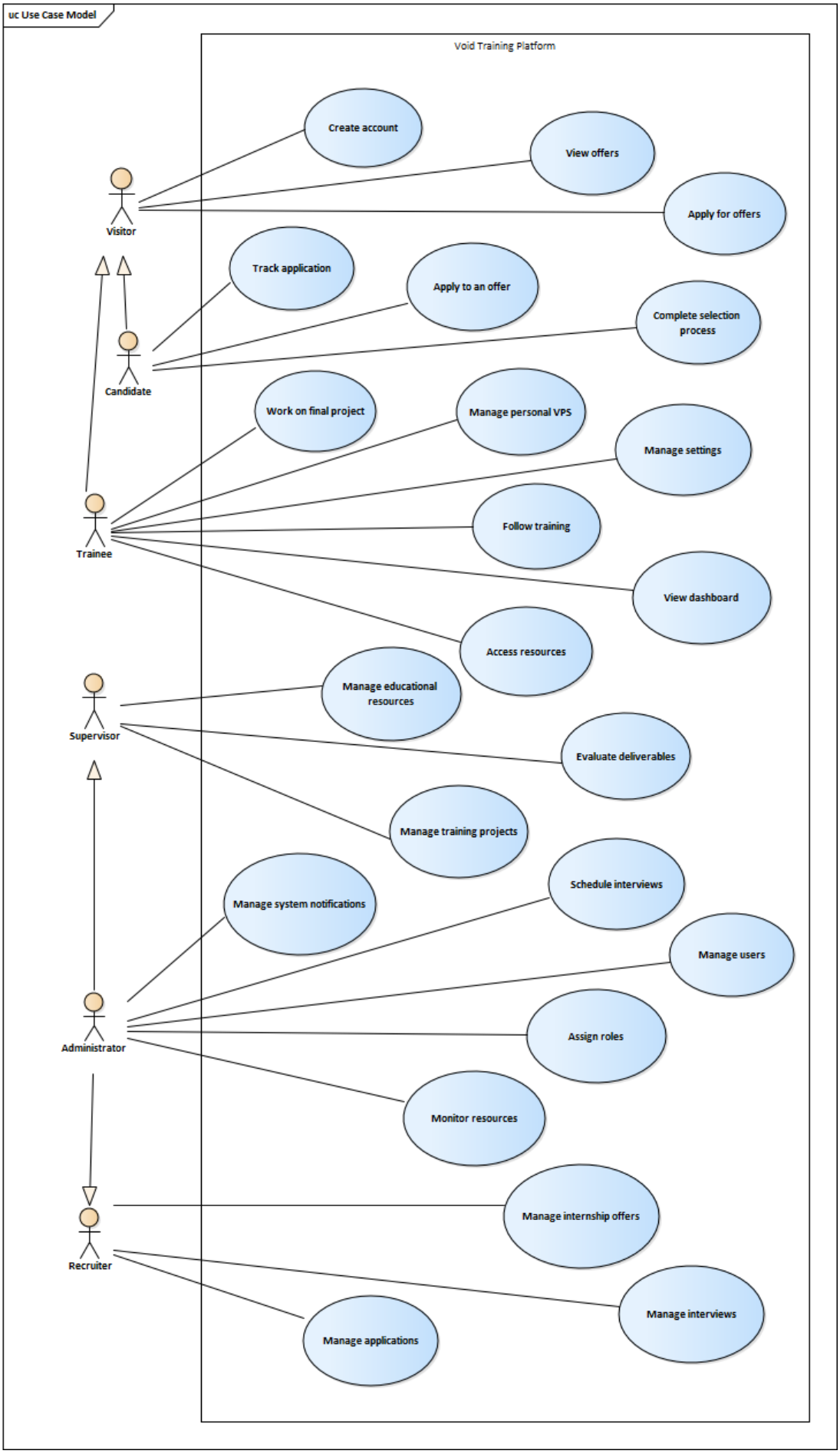

Figure 2.5: Use Case Diagram for the Intern Management Platform

2.6.2 Sequence Diagram

The sequence diagram (Figure 2.6) represents the detailed interaction flow of the candidate application process, showing the communication between different system components throughout the recruitment pipeline. This diagram illustrates the technical architecture and data flow from the candidate's initial application submission to the final decision notification.

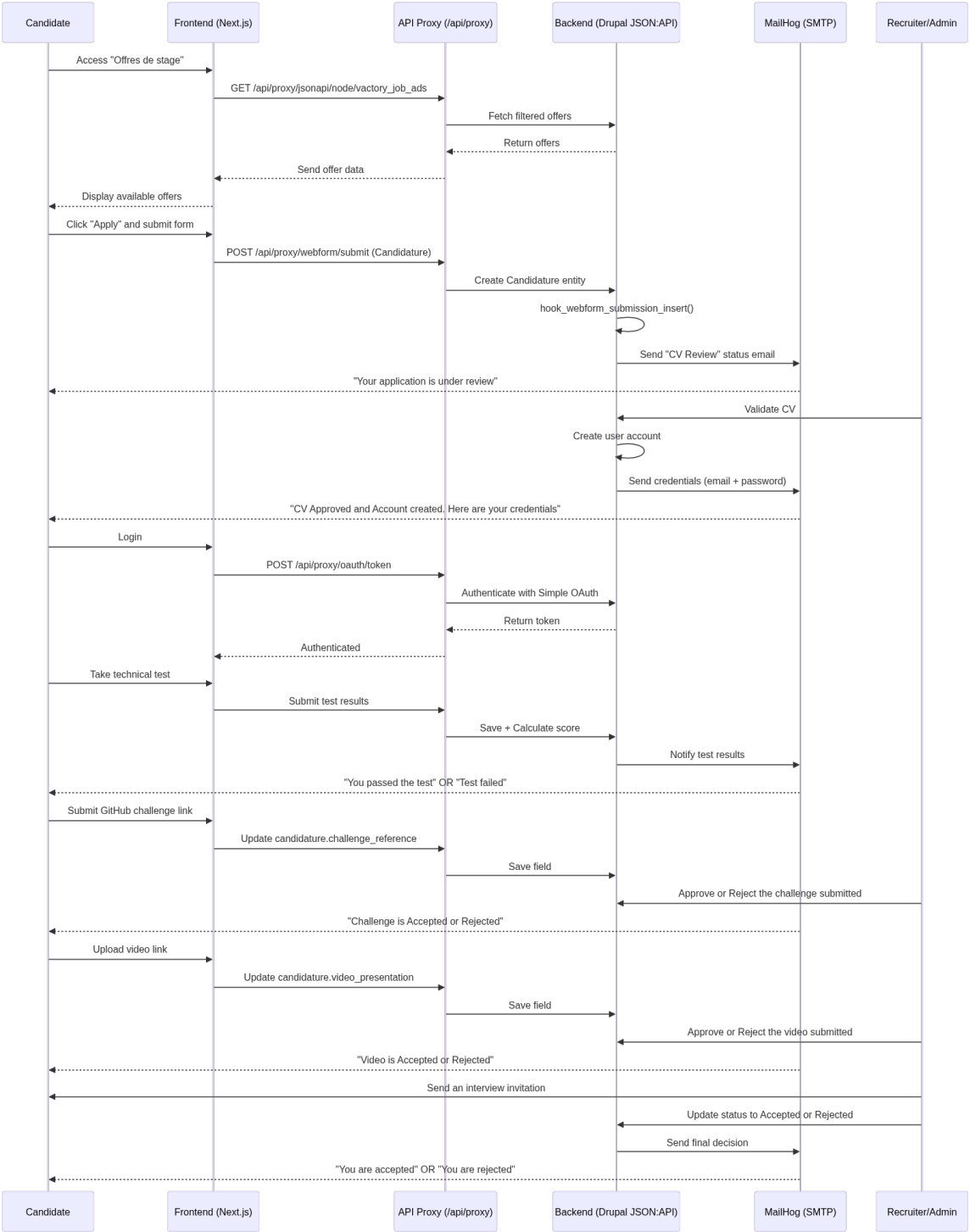

Figure 2.6: Sequence Diagram - Candidate Application Process Flow

2.6.3 Class Diagram

The class diagram (Figure 2.7) presents the complete data model and entity relationships within the Intern Management Platform. This diagram illustrates the core entities, their attributes, and the relationships between different components of the system, providing a comprehensive view of the platform's structural design.

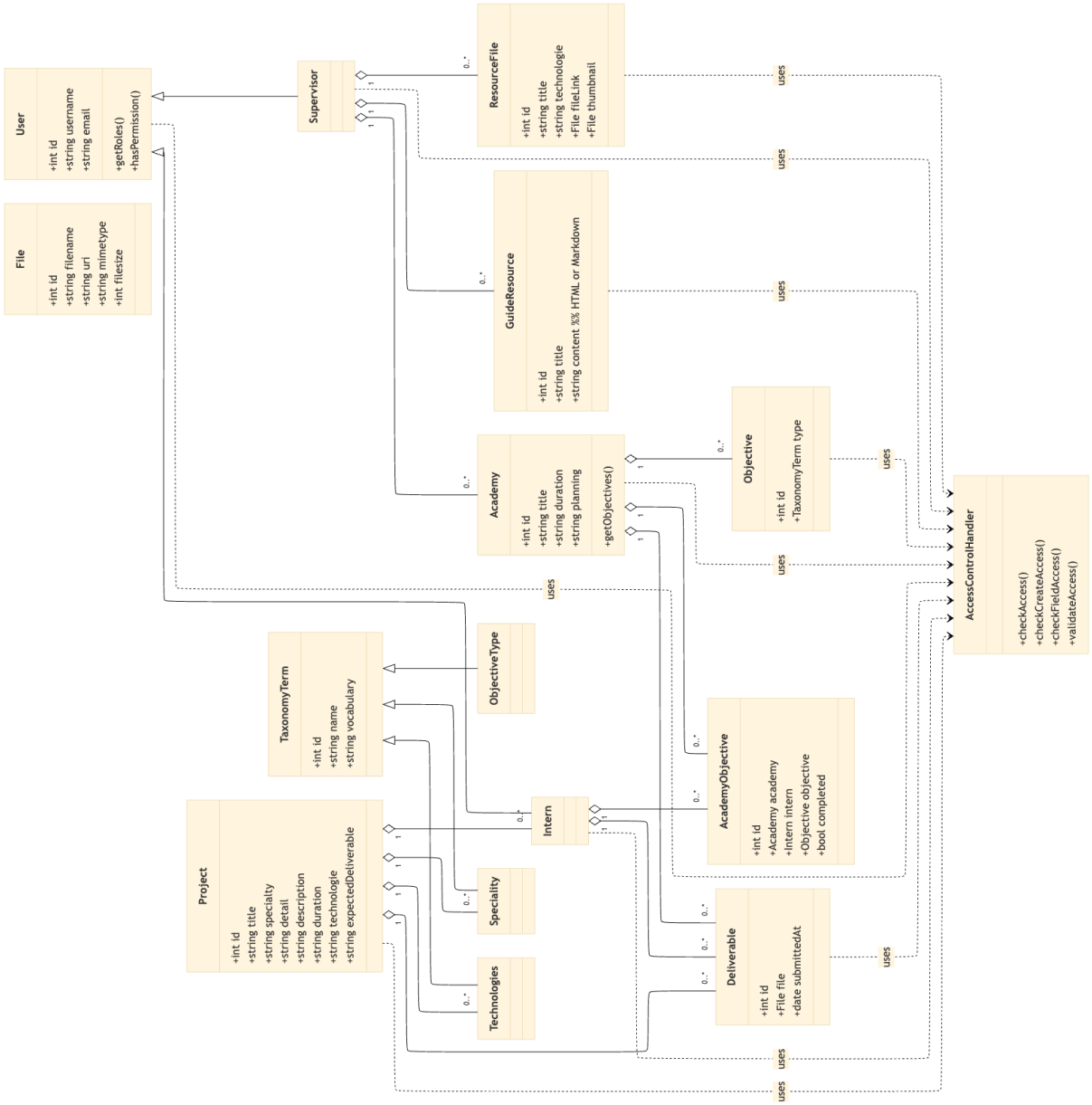

Figure 2.7: Class Diagram - System Entity Relationships and Data Model

2.6.4 System Architecture Diagram

2.7 Conclusion

This chapter laid the foundation for the technical and functional understanding of the Intern Management Platform. By outlining the problem, conducting a user journey analysis, identifying system functionalities, and modeling the architecture using UML diagrams, the analytical groundwork was established.

This phase not only clarified user needs but also guided design choices that ensured alignment with business goals and technical feasibility. The next chapter will delve into the practical implementation of these components, covering the architecture setup, technologies used, integration strategies, and component development.

3 Main Project Implementation

3.1 Introduction

This chapter details the implementation and development phase of the Intern Management Platform. The development process followed an agile methodology, with one-week sprint cycles with the encadrant and two-week sprint cycles with the CEO & CTO. The implementation phase spanned over six weeks, focusing on delivering a robust, scalable, and user-friendly platform.

3.2 Project Timeline

The implementation phase was structured into six focused sprints, each targeting a specific set of deliverables and building incrementally on the previous work. The following summary reflects the actual project flow as visualized in the Gantt chart (Figure 3.1):

- **Sprint 1: Setup** Initial setup of the Vactory distribution and Drupal environment, along with the DevOps stack (OrbStack, Docker, EasyPanel).
- **Sprint 2: Main Website** Development of the main website, including widget creation (Hero, FAQ, CTA, Process), job ads content types, taxonomies, and job offers listing modules.
- **Sprint 3: Training Platform (Backend)** Implementation of backend features including training entities, project modules, resources management, access control, status taxonomy, and automated progress tracking.
- **Sprint 4: Training Platform (Frontend)** Development of the training platform frontend, featuring the training space interface, resource library, status and objective trackers, project submission system, and progress monitoring dashboard.
- **Sprint 5: Admin Tools and Integration** Development of admin dashboards, iframe review tools, role and permission setup, and the email notification system. Integration of all modules and finalization of admin features.
- **Sprint 6: Testing, Documentation, and Deployment** Comprehensive testing and debugging (frontend and backend), CI/CD pipeline setup, Docker Compose finalization, and preparation of final documentation and delivery.

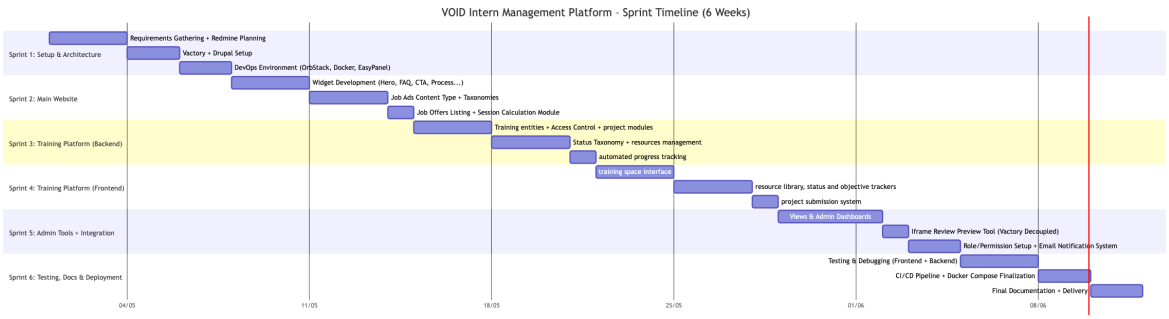

Figure 3.1: Project Gantt Chart: VOID Intern Management Platform Sprint Timeline (6 Weeks)

3.3 Programming Languages and Frameworks

3.3.1 Programming Languages

- **PHP:** Used for its robust ecosystem, versatility in web development, and strong server-side data processing capabilities. PHP was also required because Drupal, our chosen CMS, is built on it.

Figure 3.2: PHP Logo

- **HTML5:** Essential for structuring web pages, providing semantic markup, and enabling modern web features. HTML5 is a standard in all web development projects.

Figure 3.3: HTML5 Logo

- **JavaScript:** Necessary for creating interactive and dynamic user interfaces. JavaScript is fundamental for client-side logic in modern web applications.

Figure 3.4: JavaScript Logo

3.3.2 Frameworks and Libraries

- **Next.js:** Chosen for its flexibility, server-side rendering (SSR), and SEO optimization. Next.js allows us to fine-tune the frontend and improve both performance and search engine visibility.

Figure 3.5: Next.js Logo

- **Tailwind CSS:** Selected to unify design conventions between frontend developers and UX/UI designers, simplify responsive design, and accelerate interface development.

Figure 3.6: Tailwind CSS Logo

- **Drupal:** The agency specializes in Drupal, which is a strong choice for its modular, Symfony-based architecture. Drupal offers full control over backend and back office content integration, making it superior to many other CMS options for our needs.

Figure 3.7: Drupal Logo

3.3.3 Database and Caching

For data management and performance optimization:

- **MySQL:** Primary relational database for structured data storage.

Figure 3.8: MySQL Logo

- **Redis:** Used for in-memory caching to improve application performance.

Figure 3.9: Redis Logo

- **Memcached:** Used alongside Redis for distributed caching and performance.

Figure 3.10: Memcached Logo

- **Mailhog:** Used for email testing in development environment.

Figure 3.11: Mailhog Logo

3.4 Development Tools

Our development environment was chosen to maximize productivity, collaboration, and consistency across the team.

- **Visual Studio Code:** Selected for its lightweight nature, rich extension ecosystem, and strong support for web technologies, making it ideal for both frontend and backend development.

Figure 3.12: Visual Studio Code Logo

- **Docker:** Used to containerize all services, ensuring that development, testing, and production environments are consistent and portable across different machines.

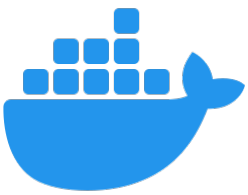

Figure 3.13: Docker Logo

- **Orbstack:** Chosen to simplify Docker management on macOS, providing an easy-to-use interface for handling containers, improving developer workflow, and enabling

fast code synchronization between the host and containers without the need for additional tools like Mutagen.

Figure 3.14: Orbstack Logo

- **Git:** Used for version control, enabling efficient collaboration, code history tracking, and branching workflows.

Figure 3.15: Git Logo

- **Bitbucket:** Chosen as our Git hosting platform, similar to GitHub, for managing private repositories and supporting team-based workflows.

Figure 3.16: Bitbucket Logo

- **Docker Hub:** Used for building, storing, and distributing Docker images, streamlining deployment and integration processes.

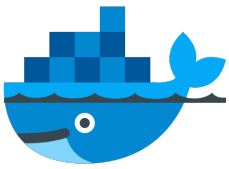

Figure 3.17: Docker Hub Logo

- **EasyPanel (Staging Server):** Used as a graphical interface to manage containers and services on the staging server, making deployment, monitoring, and pre-production validation easier and more efficient.

Figure 3.18: EasyPanel Logo

- **Nginx:** Used as a web server and reverse proxy to efficiently route frontend and backend traffic.

Figure 3.19: Nginx Logo

3.5 Development Environment

Our development environment is built for consistency, scalability, and security. We use Orbstack to manage all containers, making setup and code syncing easy.

3.5.1 Backend Containers

The backend is structured as follows:

- **Nginx + PHP-FPM + XHProf Container:** Runs the Drupal code, serves the application, handles all web requests, and provides profiling/debugging with XHProf. Code is synced from the host using Orbstack.
- **Memcached Container:** Connected to the main container for backend caching, improving speed and performance.
- **MySQL Container:** Stores all structured data for Drupal, connected internally to the main container and PhpMyAdmin.
- **PhpMyAdmin Container:** Provides a GUI for managing the MySQL database.
- **Mailhog Container:** Used for email testing. Drupal is configured to send emails via SMTP to Mailhog, which catches them for testing.

- Ports are exposed as follows: Drupal (8080), PhpMyAdmin (8085), Mailhog (8002).
- Database and Memcached containers are only accessible via the internal Docker network for security.

3.5.2 Frontend Containers

The frontend setup includes:

- **Node.js Container:** Runs the Next.js application.
- **Redis Container:** Used for caching and boosting frontend performance.
- **Redis Commander Container:** Allows management of Redis cache via terminal or GUI.

The Next.js frontend communicates with the backend, but a reverse proxy is set up so the backend is not directly exposed. All API requests go through the frontend, making the backend invisible to the outside.

3.5.3 Frontend-Backend Decoupling

The platform's architecture is fully decoupled Richardson (2018). The frontend and backend run as separate services, communicating only through JSON:API. Drupal's native support for this standard meant we didn't have to build custom endpoints for every entitycontent, taxonomies, users, and files were all exposed automatically, with permissions handled by Drupal itself. for example:

- Content Types: `/jsonapi/node/{type}`
- Taxonomies: `/jsonapi/taxonomy_term/{vocabulary}`
- Users: `/jsonapi/user/user`
- Files: `/jsonapi/file/file`

Widget-Based Component Binding

To empower non-technical users, we built a widget system in Drupal. Each widget comes with a **screenshot**, a **.twig** template for monolithic deployments, and a **settings.yml** for the decoupled frontend. On the React side, a webpack plugin scans for ***Widget.jsx** files, reads their metadata, and links them to backend widgets. This setup allows for lazy loading and keeps the bundle size in check.

Route Translation and UUID Resolution

SEO-friendly URLs are a must. Using Drupal's Pathauto, every alias (like /internship-offers) is mapped to a UUID and content type. The frontend then fetches the right data via JSON:API, so routing stays flexible and isn't tied to backend structure. Here is an example of a route translation:

1. The alias (e.g., `/internship-offers`) is passed to a translation endpoint.
2. A response returns the `UUID` and `content type`.
3. The UUID is used to fetch detailed JSON:API data for rendering.

3.5.4 Secure API Proxying

Direct backend exposure is risky. All API calls go through a Next.js proxy endpoint, which lets us tweak headers, strip cookies, and add caching or rate limits if needed. This extra layer keeps things secure and gives more control over traffic.

3.5.5 Caching Architecture

Performance matters, especially as the platform grows. We set up Memcached on the Drupal side for dynamic content and Redis on the frontend for frequently accessed collections and sessions. Redis Commander makes it easy to inspect and debug cache contents.

3.5.6 Mail Testing Infrastructure

Testing email flows without spamming real users is crucial. MailHog catches all outgoing mail in the dev stack, so we can check templates and triggers safely.

3.5.7 Component Development with Storybook

Storybook runs in its own container, letting us and the team build and preview UI components in isolation. Mocked backend data and stakeholder previews help us catch design issues early.

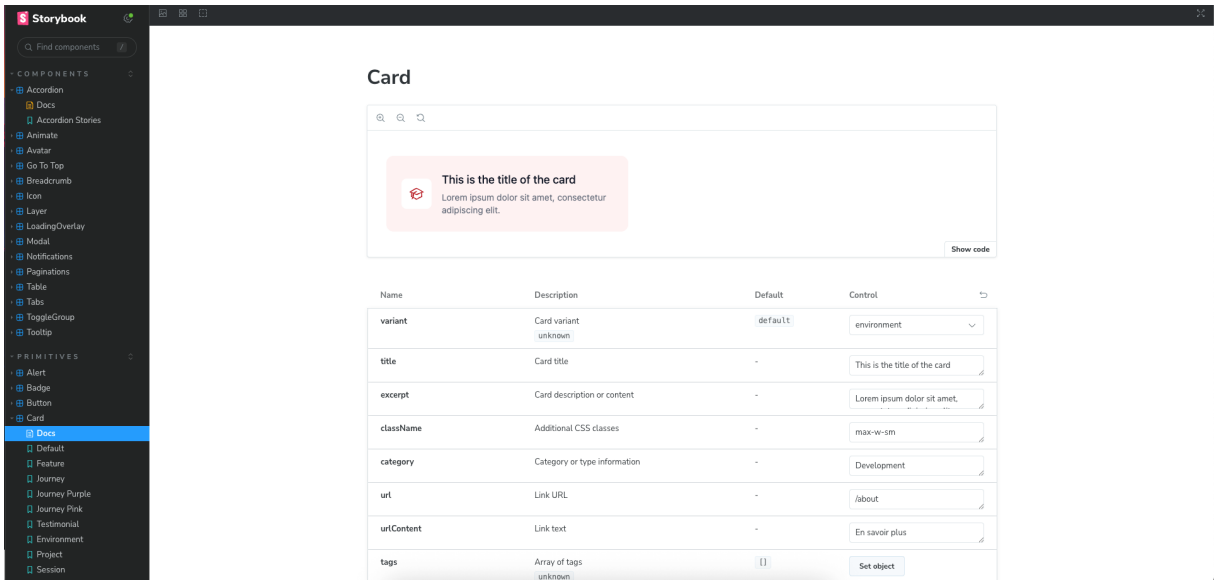

Figure 3.20: Storybook UI

3.5.8 Routing Strategy

We opted for Next.js Pages Router over the newer App Router. The Pages Router is stable, well-documented, and better supported by the modules we needed. Data fetching is straightforward, and the learning curve is lower for new contributors.

3.5.9 Development Architecture Summary

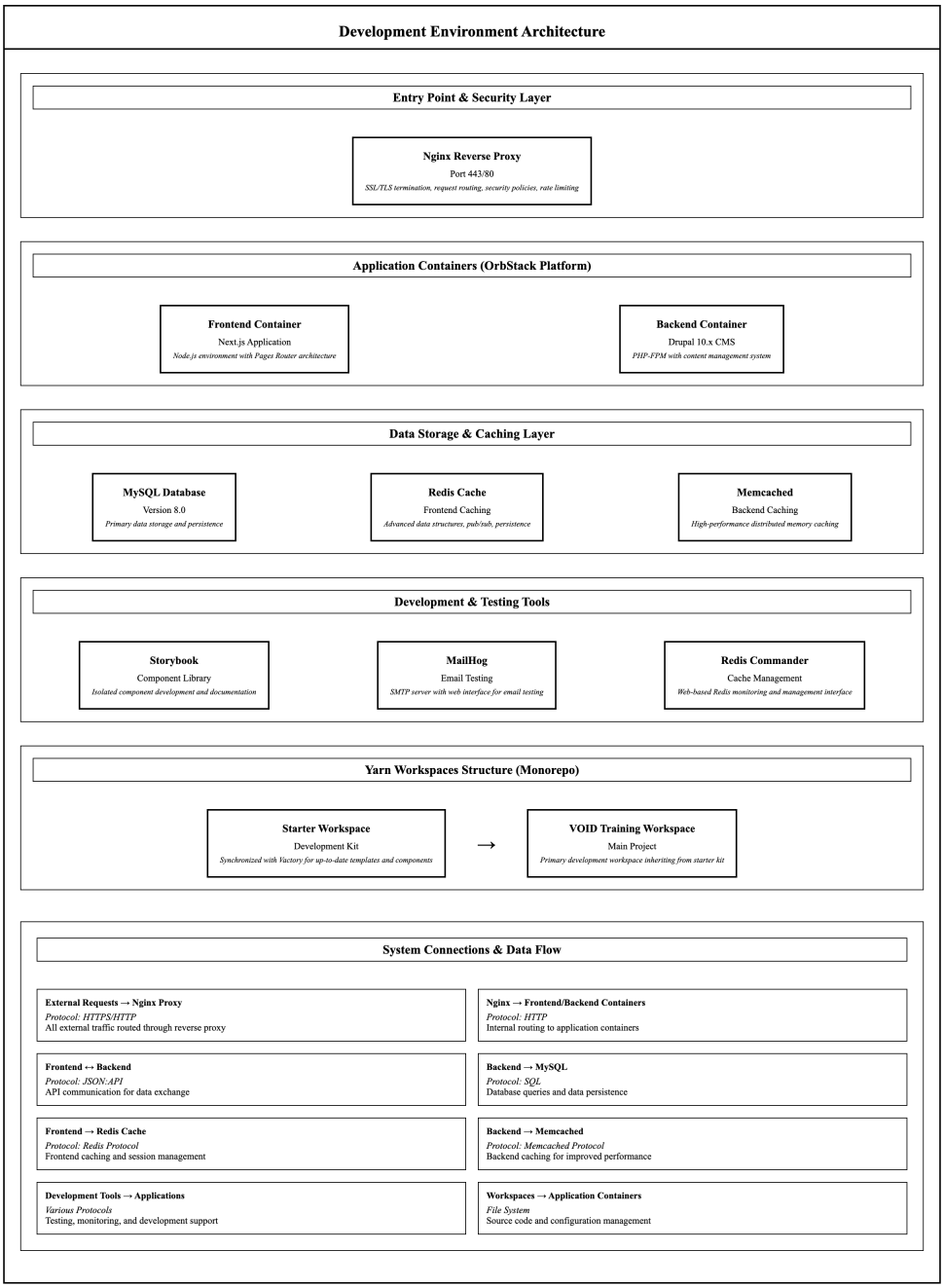

Figure 3.21: Architecture of the Development Environment

This environment made collaboration smooth, kept deployments predictable, and let us adapt quickly to new requirements. Containerization, decoupling, and the right tools meant fewer headaches and more time spent building features that matter.

3.6 Frontend Implementation

The frontend of the Intern Management Platform was developed using the **Next.js** framework, leveraging its server-side rendering (SSR) capabilities, file-based routing system, and built-in API support. The goal was to deliver a performant, SEO-friendly interface that provides a seamless user experience across three core user roles: candidates, interns, and administrators.

3.6.1 Architecture and Structure

The frontend project follows a clean modular structure with reusable components and page-level organization:

- `pages/` - Top-level routes such as `/`, `/candidat`, `/formation`, and `/admin`.
- `components/` - Shared UI elements (Buttons, Forms, Cards, Modals).
- `services/` - API communication layer via `axios`.
- `hooks/` - Custom React hooks for state and side-effects.
- `styles/` - SCSS/TailwindCSS files for component styling.

3.6.2 UI Design System with Storybook

A key part of frontend development was the creation of a **centralized UI library** using **Storybook**. Storybook allowed us to document and preview each component in isolation, and served both as a development playground and as a design reference for VOID's team.

- Each UI element (e.g. button, card, alert, input field) was developed as a standalone React component.
- Props, variants, and themes were fully documented inside the Storybook environment.
- Figma design references provided by VOID designers were used to match styles and spacing precisely.
- This also enabled us to demonstrate and validate components with stakeholders before integration.

Insert a Storybook interface screenshot here:

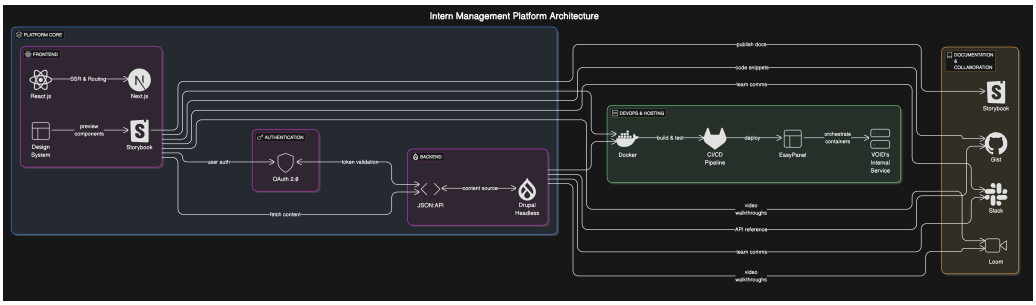

Figure 3.22: Storybook UI Library for the Platform

3.6.3 Component Implementation Strategy

The platform was designed with a mobile-first, responsive mindset, making heavy use of TailwindCSS utility classes for consistency and performance. Components were built with reusability and flexibility in mind:

- Form elements are reactive, accessible (ARIA-compliant), and support validation.
- Each page reuses blocks from the UI library to reduce redundancy.
- Skeleton loaders and placeholder states ensure a smooth experience even on slow connections.

3.6.4 Routing and Navigation

Next.js routing was used to create logical and clean URL structures. For example:

- `/` - Homepage with general internship info.
- `/candidat/login`, `/candidat/profile` - Candidate-specific routes.
- `/formation/modules`, `/formation/quiz` - Learning environment for interns.
- `/admin/offers`, `/admin/evaluations` - Administrative back-office.

3.7 Backend Implementation

3.7.1 Introduction

The backend of the Intern Management Platform is built on Drupal 10, using VOID Agency's Vactory distribution. We chose this stack for its modularity and JSON:API support, which enables seamless frontend integration. The backend consists of three main components: the Main Website, Candidate Space, and custom modules for recruitment automation.

3.7.2 Drupal Architecture and API Strategy

What is Drupal?

Drupal is an open-source content management system (CMS) used to build and manage websites. It provides a flexible back office for content creation, user management, and site configuration.

Our Use of Drupal

In our project, we use Drupal mainly as a backend service. The Drupal back office is used for content management, while the frontend is built separately. We do not use Drupal's default theming or page rendering for the public site.

Exposing Content with JSON:API

To connect Drupal with our frontend, we enabled and configured the **JSON:API** module. This module allows us to expose Drupal content, users, menus, and other entities as RESTful API endpoints. These endpoints are configurable and can be consumed directly by the frontend application.

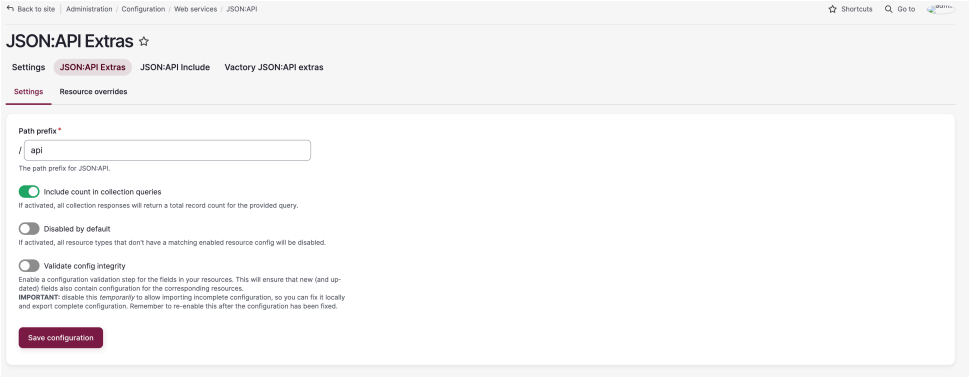

Figure 3.23: JSON:API endpoint configuration with path prefix /api

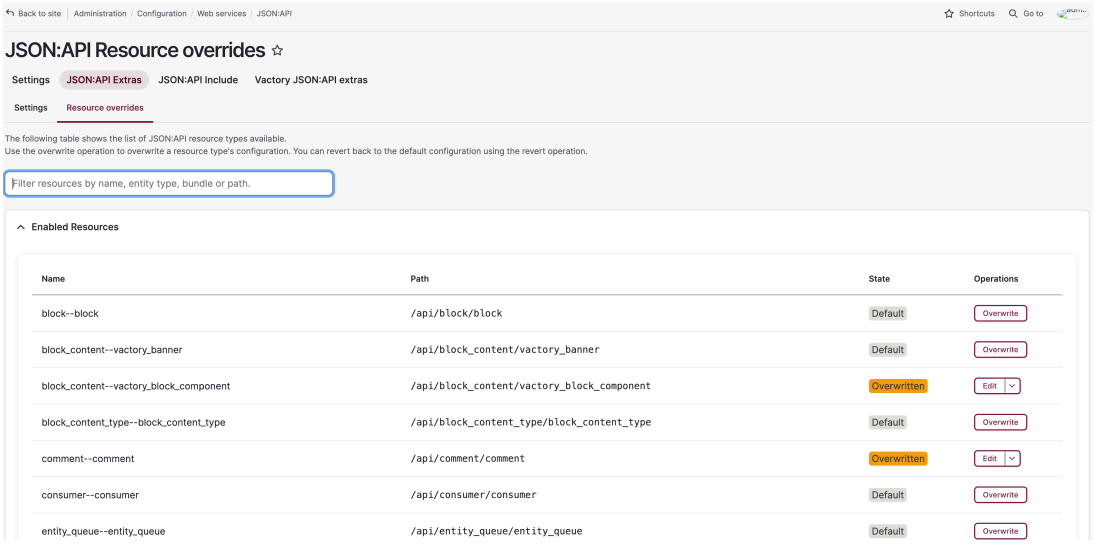

Figure 3.24: Exposed JSON:API resources in Drupal

Custom API Endpoints for Menus and Translations

For essential features like menus and translations, we configured two custom API endpoints:

- `/api/_translations` Returns translation data for the site.
- `/api/_menus?menu_name={main,footer,...}` Returns menu structures for navigation.

Building Website Structure with Dynamic Fields

To enable flexible page construction, we use the **Dynamic Field** module and a modular development approach. Website componentssuch as headers, footers, and card listsare implemented as reusable Drupal entities, which we refer to as *widgets*. These widgets are defined in the codebase and can be placed visually within page layouts using the Drupal back office.

The typical workflow is as follows:

1. Create widget templates in the codebase, specifying the data to be exposed.
2. Use the Drupal interface to assign widgets to specific page regions.
3. The JSON:API module automatically exposes the page structure and widget data as API endpoints.
4. Consume these endpoints to render the final user interface.

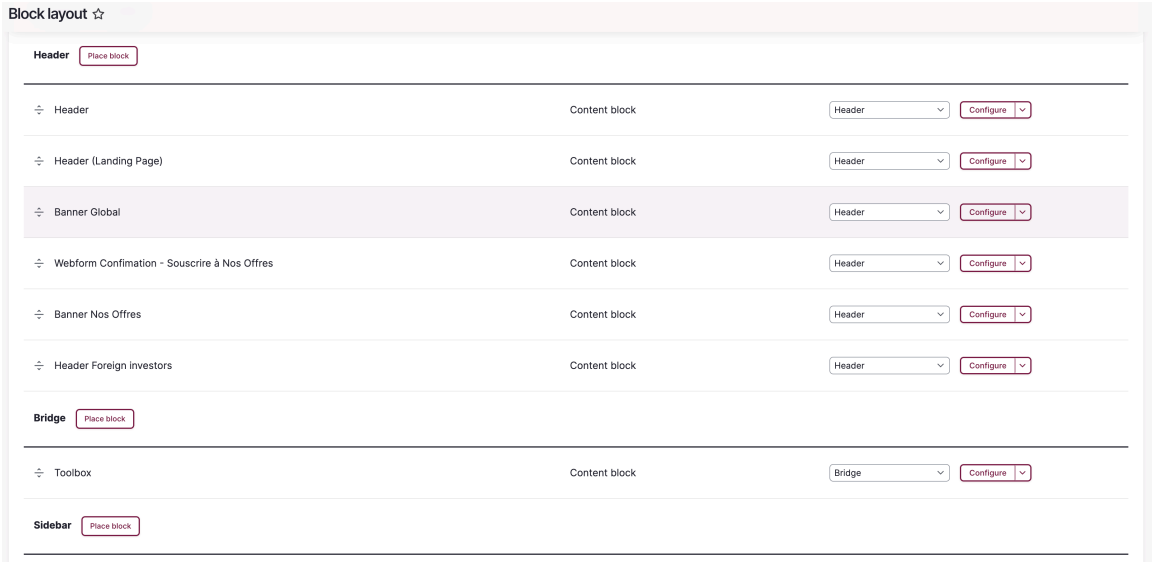

Figure 3.25: Visual placement of widgets (blocks) in the Drupal back office

This decoupled approach allows for rapid content updates and consistent data delivery to the frontend, while maintaining a clear separation between content management and presentation.

3.7.3 Performance Optimization with Memcached

To enhance backend response time and reduce database load, we implemented Memcached as our caching solution. In our Docker-based architecture, Memcached runs as a dedicated container, communicating directly with Drupal through our internal Docker network.

Memcached improves our platform's performance by:

- Caching frequently accessed data in memory (entities, views, routes)
- Reducing database queries and disk I/O operations
- Providing near-instant access to cached content
- Enabling horizontal scaling of the caching layer

This caching strategy is particularly effective for our recruitment platform, where multiple users frequently access the same content like job listings and candidate profiles. During peak recruitment periods, Memcached helps maintain fast response times even under high traffic loads.

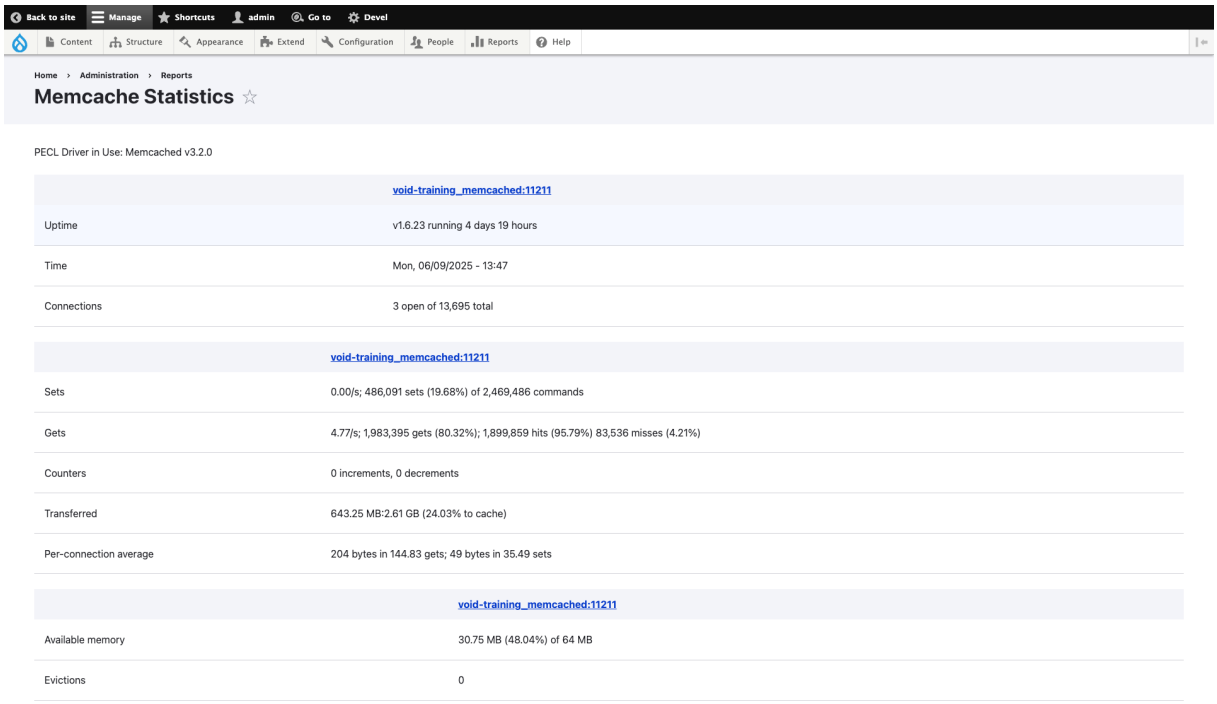

Figure 3.26: Memcached UI

3.7.4 Email Debugging with MailHog

For local development and QA testing, we integrated MailHog as a Docker container to capture and inspect outgoing emails. This containerized approach provides a safe environment for email testing without affecting real users.

The MailHog container provides:

- A web interface (accessible at port 8025) for viewing captured emails
- An SMTP server (on port 1025) that intercepts all outgoing emails
- Real-time email capture and inspection capabilities

We configured Drupal to use MailHog's SMTP server through the Symfony Mailer module, which offers better reliability than PHP's native mail() function. This setup allows us to monitor all system emails, including:

- Account creation notifications
- Interview scheduling emails
- Evaluation report deliveries
- Password reset requests

The containerized approach makes it easy to spin up the email testing environment in any development or QA instance, ensuring consistent email debugging capabilities across the team.

3.7.5 Widget System Architecture and Discovery Process

The backbone of our content management strategy relies on Vactory's widget system, which implements dynamic, reusable content components that bridge backend configuration with frontend rendering. Understanding the widget lifecycle reveals how Drupal automatically discovers, categorizes, and exposes these components to content editors.

The widget discovery process follows a structured technical workflow:

1. **Widget Definition**: Each widget consists of a directory containing a `settings.yml` file that defines metadata, JSON:API queries, field structures, and categorization information. This YAML configuration acts as the widget's blueprint, specifying data sources, field types, and display preferences.

2. **Automatic Discovery**: During Drupal's module discovery phase, the system scans all enabled modules for widget directories. The presence of a `settings.yml` file signals to Drupal that the directory contains a valid widget definition.

3. **Plugin Registration**: Drupal's plugin system registers each discovered widget as a paragraph type, making it available through the administrative interface. The widget's metadata from the YAML file determines its category, id, and content.

4. **Frontend Mapping**: A corresponding Webpack plugin on the frontend side scans for widget files matching specific patterns (`*Widget.jsx`). This plugin extracts configuration IDs from each React component and generates a mapping file that links backend widget IDs to their frontend implementations.

5. **Template Selection Interface**: Content editors access widgets through Drupal's paragraph interface, where widgets are organized by categories (Academy, Banners, Content, Void Training, etc.). This categorization is defined in each widget's YAML configuration.

When content editors create or edit pages, they encounter a streamlined widget selection process. The template chooser displays widgets organized by functional categories, allowing editors to quickly locate appropriate components for their content needs.

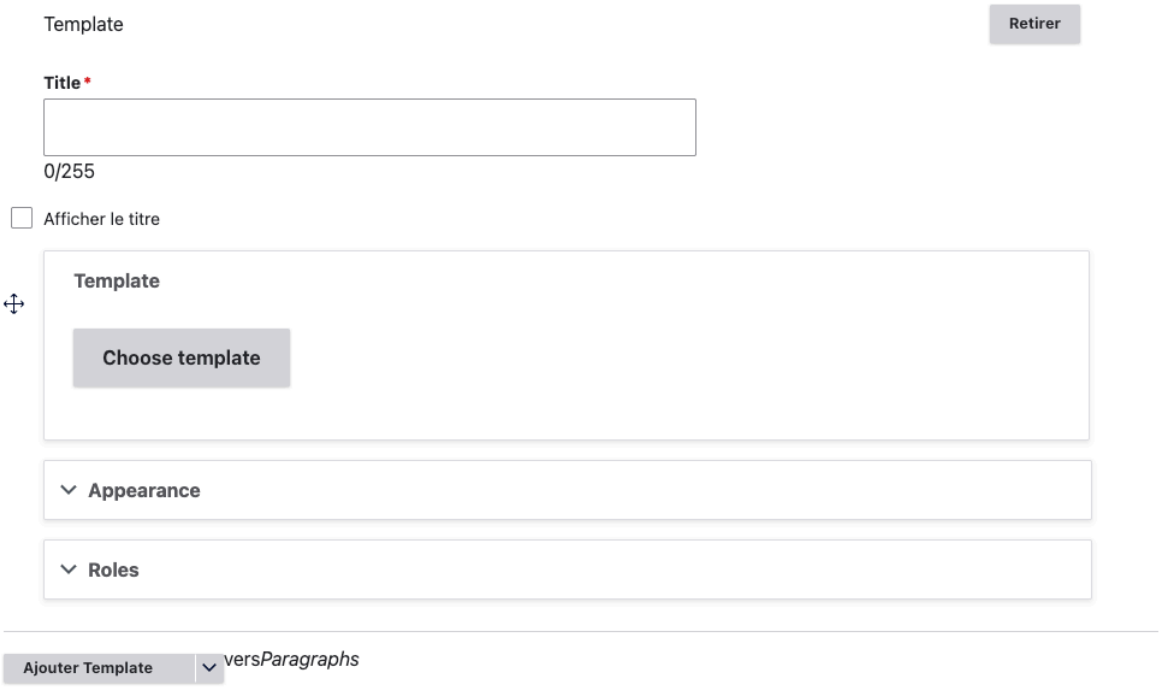

Figure 3.27: Widget Selection Interface with Category Tabs

Within each category, editors can browse available widget templates. For the Void Training project, custom widgets were developed specifically for recruitment and training content, including process flows, call-to-action sections, and testimonial displays.

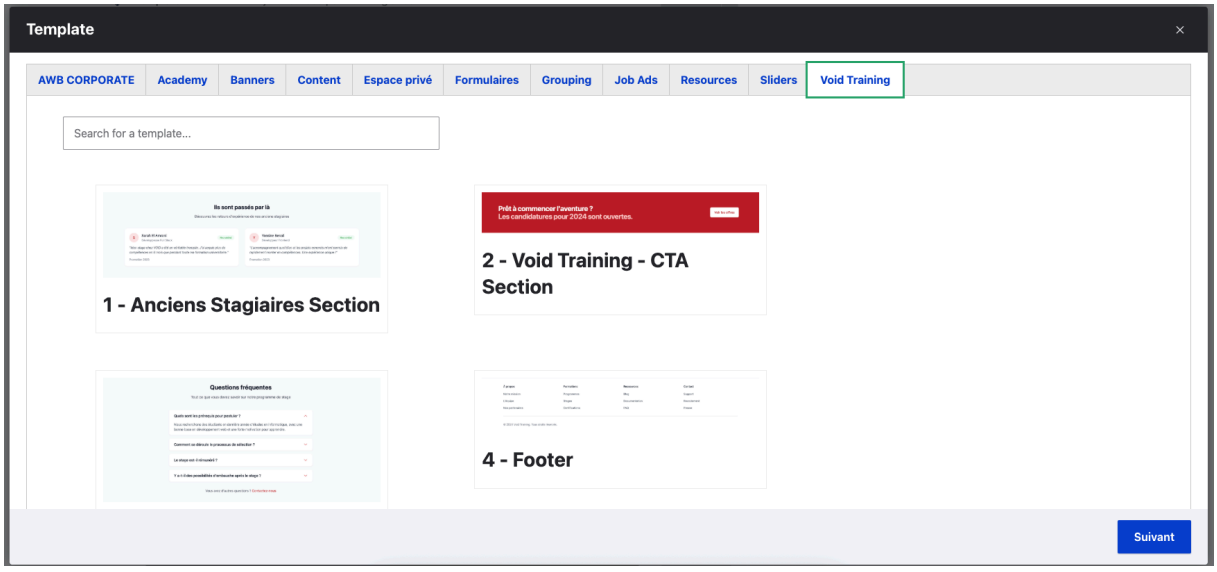

Figure 3.28: Available Widget Templates in Void Training Category

Once a widget is selected, editors access a dynamic configuration interface. Each widget exposes different field types and structures based on its YAML definition. For complex widgets like the Recruitment Process widget, editors can configure multiple components, titles, descriptions, and process steps through an intuitive form interface.

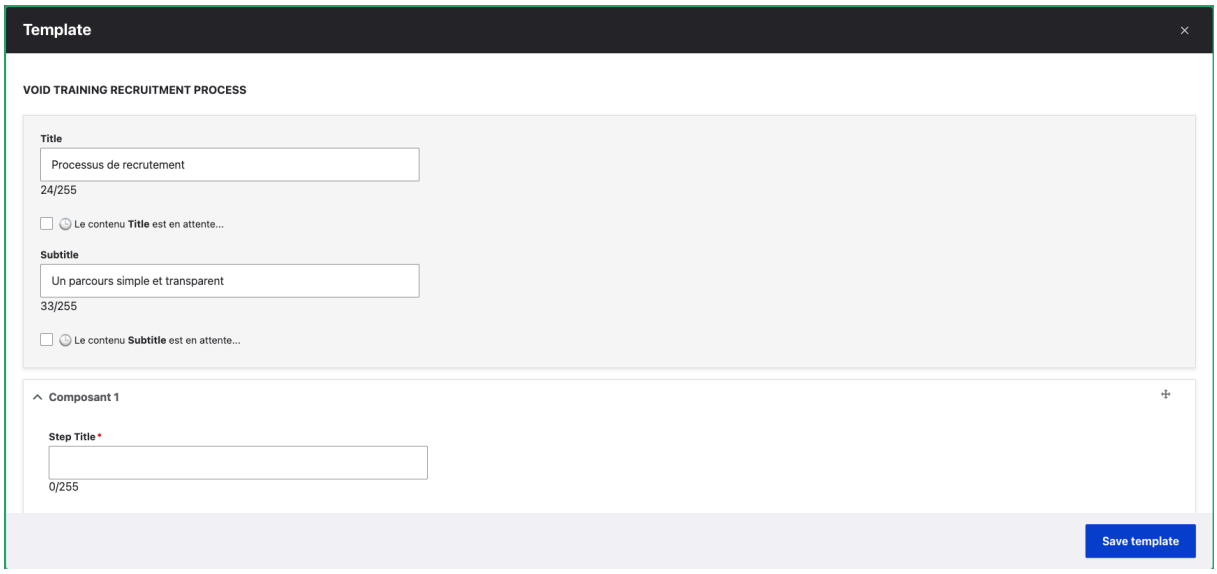

Figure 3.29: Widget Configuration Interface for Recruitment Process

3.7.6 Main Website Backend Structure

The main website of the platform acts as the public-facing interface where internship opportunities are presented and where potential applicants begin their journey. Its backend is structured using Drupal nodes and dynamic fields to allow flexible content curation by non-technical administrators.

The site is entirely modularized through the widget system offered by Vactory. Each widget corresponds to a distinct frontend component (React), and its data source is a YAML-defined configuration that maps JSON:API resources to frontend display logic.

These widgets act as atomic UI and content modules and are highly reusable across different pages. Their configuration is defined using a settings.yml file and a Twig template for non-decoupled implementations.

For this project, several key widgets were implemented specifically for the main website:

- Recruitment Process Widget: Highlights the different stages of the candidate journey.
- Training Cards Widget: Displays available learning tracks post-acceptance.
- FAQ Widget, Call-to-Action Widget, and Social Media Widget: Used to enrich content and increase engagement.

The job offer listing is based on a custom content type named vactory_job_ads, which integrates several taxonomies:

- Specialities: The specialities of the job offer.

- Internship Type: The type of internship (PFE, PFA, etc.).
- School: The school of the job offer (ENSAM, ENSA, etc.).
- Required Technologies: The technologies required for the job offer (React, Node.js, etc.).

The job offer node also supports:

- JSON:API exposure for structured field-level access.
- Automated filtering and sorting via Drupal Views and contextual arguments.
- Integration with Webforms for candidate applications.

Widgets were created to expose these structured contents:

- List of all job offers.
- List of the three most recent offers.
- Postulation form widget (Webform-based).
- Recruitment Session widget: This logic calculates ongoing and upcoming sessions based on offer dates and availability.

3.7.7 Candidate Journey and Backend Logic

Once an applicant engages with the system, a complex series of interactions are handled by a custom entity named Candidature, which was implemented via ContentEntityBase. This custom entity serves as the backbone of the candidate's progression and encapsulates all relevant data, from personal information to quiz results, challenge links, and final decisions.

Candidature Entity Fields and Relationships

- User Reference: Links the candidature to the Drupal user account.
- Personal Details: First name, last name, email, phone number, filière, and school (taxonomy reference).
- Submission Data: Linked CV (file), LinkedIn URL.
- Recruitment Artifacts: Entity references to:
  - vactory_job_ads for job offer association,
  - vactory_quiz for technical tests,
  - challenges for problem-solving assignments.

  - Video presentation.

- Scoring Data: Quiz results, cheating detection, timestamp, duration, and question count.
- Decision Support: Status (via taxonomy term), dates of internship (start/end), GitHub challenge URL, and candidate-specific dynamic fields.

Status Tracking with Taxonomy To handle multi-step evaluation workflows, a custom taxonomy named candidature_status was defined. It governs both visibility and backend logic. The statuses are as follows:

- CV Review: Initial status when candidature is submitted for CV review.
- CV Approved: The candidate's CV has been approved by the recruiter, user account created and test email sent.
- Test Approved: The candidate has passed the technical test successfully.
- Challenge Reviewing: Challenge has been completed and is being reviewed by the recruiter.
- Challenge Approved: The candidate has passed the challenge successfully.
- Video Reviewing: Video presentation is under review by the recruiter.
- Video Approved: The candidate has passed the video review successfully.
- Accepted: The candidate has been accepted for the internship.
- Rejected: The candidate has been rejected for the internship.

This taxonomy enables the frontend to dynamically render progress indicators and allows the backend to execute specific logic on status change.

3.7.8 Candidate Account Lifecycle and Automation

The platform implements full automation of candidate onboarding:

1. CV Approval: A recruiter reviews the initial application. If approved:
   - A user account is automatically generated.
   - Credentials are emailed using a dedicated mail hook.
   - The test is assigned automatically based on the selected specialization (taxonomy).
2. Quiz Completion: The candidate takes a test. The score is evaluated in real-time using custom logic. If the score passes:

- The status changes to Test Approved.
- The next step (challenge) is automatically assigned.

3. Challenge Review: The recruiter validates the GitHub-submitted project manually.
4. Video Upload: A video presentation URL is submitted and manually reviewed.
5. Final Decision: Based on recruiter evaluation, the status is set to Accepted or Rejected. On acceptance, the role is updated to stagiaire.

Each of these transitions is handled using custom entity hooks such as:

- webform_submission_insert: Triggered when a new webform submission is created, used to process initial candidate applications and trigger automated workflows.
- entity_update: Fires when any entity (like a candidature) is updated, enabling status change tracking and automated transitions between evaluation stages.
- mail: Custom mail hook for sending automated notifications, including account credentials and test assignments to candidates.
- jsonapi_entity_filter_access: Controls access to JSON:API endpoints, ensuring candidates can only view their own data while recruiters have broader access.
- views_data_alter: Modifies Views data structure to add custom filters and fields for candidate management interfaces.

3.7.9 Custom Modules

To modularize the platform, several custom modules were created:

- void_training_candidature: This module handles the core candidate application functionality. It defines the candidature entity type, implements access control rules, manages dynamic form fields, orchestrates the status workflow transitions, integrates with webforms for initial submissions, and provides custom views filters for candidate management.
- void_training_quiz: This module manages the technical assessment system. It defines the structure for quiz metadata, implements the question and answer format, establishes scoring rules and thresholds, and exposes quiz data through JSON:API for frontend consumption.
- void_training_challenge: This module handles the practical coding challenge phase. It stores challenge instructions and content, manages GitHub repository URL submissions, and provides an interface for recruiters to review and evaluate submitted challenges.

Each module follows Drupal's best practices for service injection, plugin extension, hook overrides, and configuration export.

3.7.10 Custom Admin Views for Candidature Tracking

To streamline the review process for administrators and recruiters, custom Views were created in the Drupal backoffice. These Views provide a clear, tabular overview of all candidate applications, segmented by their current phase in the recruitment pipeline.

Each phase of the candidature process (e.g., CV Review, Quiz, Challenge, Video, Final Decision) is represented as a filterable tab or column, allowing administrators to quickly assess the status of each candidate and take appropriate actions. Bulk operations and contextual links are integrated for efficient workflow management.

3.7.11 Security and Access Control

Simple OAuth was used for secure JWT-based token issuance. Private claims were modified to include role and user-specific fields.

The CandidatureAccessControlHandler class restricts access based on role, entity ownership, and operation. Only recruiters can edit or validate candidatures not belonging to them. JSON:API collection access was filtered using:

Filtering Collection per Connected User:

```
function void_training_candidature_json_api_collection_alter(...) {
  // Custom logic to expose only relevant records per token
}
```

Role-Based Access Control and Permissions To maintain security and content workflow integrity, user roles and permissions were carefully structured within the backend. Three primary roles were defined:

- Anonymous User: Has read-only access to public-facing content, such as internship listings and FAQs. No account is required to view content or initiate applications.
- Authenticated Candidate: This role is automatically assigned to candidates once their CVs are approved and accounts are programmatically created. These users can authenticate via the frontend and are allowed to view, update, and track their own candidature entity.
- Recruiter and Administrator: These roles include advanced permissions to manage job offers, moderate content, review applications, evaluate technical tests and challenges, and update candidature status. Recruiters have access to content moderation workflows and admin views, while system administrators retain broader access to site configuration and user management.

3.8 DevOps and CI/CD Pipeline Setup

To ensure smooth development, testing, and deployment workflows, a complete CI/CD (Continuous Integration and Continuous Deployment) pipeline was implemented. The main goals were automation, repeatability, and stability in moving from local development to production with minimal manual intervention.

3.8.1 DevOps Stack Overview

The stack implemented at **VOID** is centered around the following tools and services:

- **Bitbucket**: Source control system for Git repositories and CI pipeline triggers.
- **Docker & Docker Hub**: Used to containerize services and store built images.
- **EasyPanel**: Lightweight UI-based orchestration platform to manage containers internally on the VOID Mac Studio server.
- **Drush**: CLI for managing Drupal operations (cache clearing, database import/export, config sync, etc.).

3.8.2 CI/CD Workflow Description

The CI/CD pipeline was configured to perform the following steps automatically on every push to the `main` branch:

1. **Build Stage:**
    - Build Docker images for the frontend (Next.js) and backend (Drupal).
    - Run unit and integration tests inside the containers.
2. **Push Stage:**
    - Push tagged Docker images to the VOID Docker Hub registry.
3. **Deploy Stage:**
    - Pull the updated image into the EasyPanel internal server.
    - Restart containers via EasyPanel to apply the new version.

3.8.3 Bitbucket CI/CD Pipeline Script (Simplified)

```
pipelines:
  branches:
    main:
      - step:
          name: Build and Deploy
          caches:
            - docker
          script:
            - docker build -t void-candidate-frontend ./frontend
            - docker build -t void-drupal-backend ./backend
            - docker push void/void-candidate-frontend
            - docker push void/void-drupal-backend
```

3.8.4 Environment Configuration

All environment-specific secrets and settings (API keys, backend URLs, mail server configuration) were injected securely via:

- Environment variables declared in EasyPanel.
- Bitbucket repository secrets.
- Drupal configuration sync files (`config/sync/*.yml`).

3.8.5 Automated Dependency Updates

A Node.js script was introduced to automate dependency checking and alerting:

- Scan `package.json` and `composer.lock` for outdated dependencies.
- Compare with latest versions from NPM/Packagist.
- Notify the developers via Slack and optionally create a pull request.

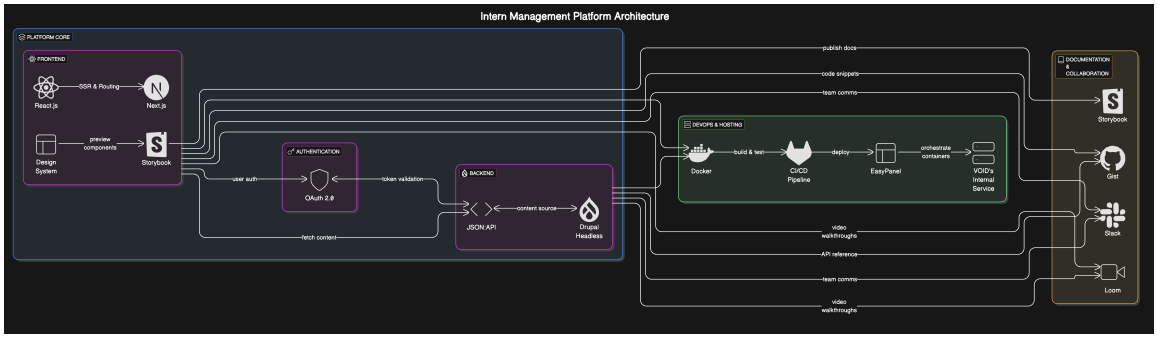

Figure 3.30: CI/CD Pipeline for the Intern Management Platform

3.8.6 Benefits Observed

The introduction of CI/CD led to the following benefits:

- **60% reduction** in deployment time.
- Elimination of manual production release errors.
- Faster testing feedback and release cycles.

You should insert a Gantt chart after this section showing the implementation schedule. Let me know if you want one auto-generated in LaTeX or TikZ.

3.9 Testing and Quality Assurance

Testing played a pivotal role in ensuring the reliability, scalability, and maintainability of the Intern Management Platform. At **VOID**, a Test-Driven Development (TDD) approach was adopted, particularly for the frontend side, to detect regressions early and maintain a high standard of code quality.

3.9.1 Testing Strategy Overview

The testing efforts were organized across different layers of the application:

- **Unit Testing:** Targeted critical logic functions (e.g., authentication, form validation, state management) using Jest for React components and PHPUnit for backend Drupal hooks.
- **Integration Testing:** Validated end-to-end flows between components such as login, offer application, and training access using testing libraries like React Testing Library.
- **Manual Functional Testing:** Performed during sprints and after deployment on EasyPanel to validate flows such as uploading documents, passing tests, and generating reports.
- **Visual Testing and QA:** Used Storybook for developing and validating UI components in isolation before integrating them into production flows.

3.9.2 Frontend Testing Tools

- **Jest:** JavaScript testing framework used for snapshot and logic tests.

- **React Testing Library:** Ensures that UI behaves as users expect through DOM-level interaction testing.
- **Storybook:** Enables developers and designers to collaborate on components independently from the app logic.

3.9.3 Backend Testing Tools

- **PHPUnit:** Unit test framework used for validating custom Drupal modules and hook implementations.
- **Drush Scripts:** Used to automate repeated testing tasks such as database resets, cache rebuilds, and configuration syncs.

3.9.4 Linting and Static Code Analysis

- `eslint` and `prettier` were integrated into the CI pipeline to maintain consistent code formatting and avoid common pitfalls in React development.
- PHPStan was used for static analysis of backend code to detect unreachable code, wrong type usage, or missing return values.

3.9.5 Example Test Case (React + Jest)

```
import { render, screen } from '@testing-library/react';
import OfferCard from '../components/OfferCard';

test('renders offer title and description', () => {
  render(<OfferCard title="UX Intern" description="Join VOID UX team!" />);
  expect(screen.getByText(/UX Intern/i)).toBeInTheDocument();
  expect(screen.getByText(/VOID UX team/i)).toBeInTheDocument();
});
```

3.9.6 Continuous Testing Integration

Tests were executed automatically during each build in the CI/CD pipeline. If any unit or integration test failed, the build was aborted and the developer was notified via Slack. This automated feedback loop ensured high code quality and regression prevention.

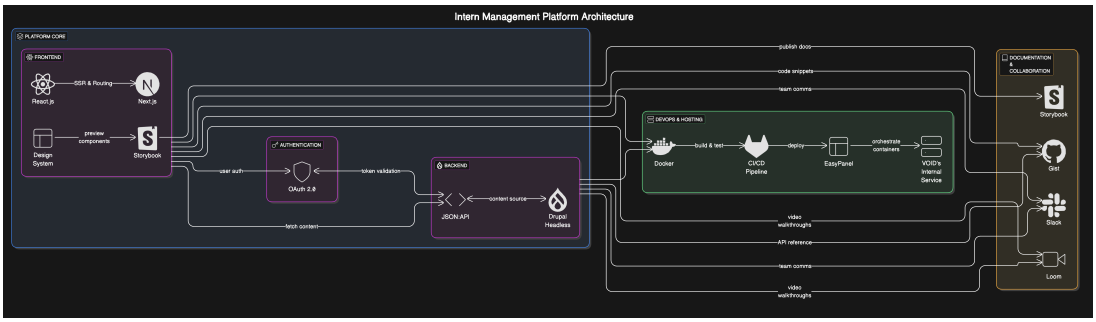

Figure 3.31: Testing Pipeline Integration in CI/CD

3.9.7 Quality Metrics

- **Code coverage:** Maintained above 80% for frontend components.
- **Bug tracking:** Performed via Redmine tickets integrated with Slack.
- **Deployment approval:** Blocked automatically if test stage fails.

4 Main Project Results & Showcase

5 Discussion and Future Improvements

5.1 Introduction

This chapter provides a comprehensive evaluation and analysis of the Intern Management Platform developed for VOID Digital Agency. The discussion focuses on the significance of the achieved results, the impact of the implemented solution, identified limitations, and potential areas for future enhancement. The analysis encompasses both technical and operational perspectives, examining how the platform addresses the initial challenges while identifying opportunities for continued improvement.

5.2 Significance of the Achieved Results

5.2.1 Process Transformation and Efficiency Gains

The implementation of the Intern Management Platform has fundamentally transformed VOID Digital Agency's internship management process. The transition from a manual, fragmented approach to a centralized, automated system represents a significant organizational advancement:

- **Application Processing**: The platform successfully handles a significant number of applications per recruitment cycle, compared to the previous manual processing that was prone to delays and human error.
- **Automated Assessment**: The integration of automatic technical test assignment and grading has significantly reduced evaluation time, allowing recruiters to focus on higher-value tasks while eliminating the cost of using external test platforms like CoderPad.
- **Standardized Workflow**: The implementation of a structured candidate journey ensures consistent evaluation criteria and fair assessment across all applicants.
- **Real-time Tracking**: Both candidates and administrators benefit from transparent status tracking, reducing inquiries and improving communication efficiency.

5.2.2 Technical Architecture Success

The chosen technical architecture has proven its effectiveness in meeting both current and anticipated future requirements:

- **Scalability**: The headless Drupal 10 backend with Next.js frontend successfully handles concurrent user loads during peak application periods.
- **Maintainability**: The component-driven development approach using Storybook has facilitated consistent UI development and simplified maintenance procedures.
- **Integration Capabilities**: The JSON:API implementation enables seamless communication between frontend and backend, supporting future expansion and third-party integrations.
- **DevOps Excellence**: The containerized deployment using Docker and EasyPanel has demonstrated reliable performance with automated CI/CD pipelines reducing deployment risks.

5.2.3 User Experience Enhancement

The platform has significantly improved the user experience for all stakeholder groups:

- **Candidate Satisfaction**: Initial feedback from candidates indicates high satisfaction with the streamlined application process, though formal metrics are still being collected.
- **Administrative Efficiency**: The recruitment team reports significant time savings in routine task management, enabling greater focus on candidate evaluation.
- **Transparency**: The improved communication system has noticeably reduced the volume of candidate inquiries.

5.3 Current Platform Limitations

5.3.1 Candidate Space Limitations

User Interface and Experience Constraints

Several areas within the candidate space require attention for optimal user experience:

- **Mobile Responsiveness**: While functional, the mobile experience could be enhanced for candidates primarily using smartphones for applications.
- **Multi-language Support**: The system is primarily designed for French-speaking candidates, limiting international applicant accessibility.

Technical Assessment Limitations

The technical testing component, while functional, presents several areas for improvement:

- **Cheat Detection Mechanisms**: The current system lacks robust anti-cheating measures such as:
  - Screen monitoring capabilities
  - Candidate webcam monitoring
  - Candidate Audio monitoring
- **Question Pool Limitations**: We currently have only 200 question in each specialization. The static question database may lead to answer sharing among candidates in subsequent recruitment cycles.
- **Challenge Pool Limitations**: We currently have only 3 challenges in each specialization. The static challenge database may lead to answer sharing among candidates in subsequent recruitment cycles.
- **Manual Challenge Review**: We currently have no automated challenge review system. The challenge review is done manually by the recruiters.
- **Manual Video Interview Review**: We currently have no automated video interview review system. The video interview review is done manually by the recruiters.

Communication and Feedback Systems

Current communication mechanisms could be enhanced:

- **Detailed Feedback**: Limited constructive feedback provided to unsuccessful candidates for future improvement.
- **Interview Scheduling**: Manual coordination required for interview scheduling, lacking integration with calendar systems, and no automatic interview scheduling system.

5.3.2 Administrative Interface Limitations

The backoffice interface presents usability challenges for non-technical administrative staff:

- **Complex Navigation**: The current Drupal administrative interface requires technical knowledge and may overwhelm non-technical users.
- **Customization Limitations**: Limited ability to customize the interface layout and workflow to match specific administrative preferences.

- **Learning Curve**: New administrators require extensive training to effectively utilize all platform features.
- **Visual Design**: The interface lacks modern, intuitive design elements that could improve user adoption and efficiency.
- **Data Export Limitations**: We only have the ability to export the data in CSV format.

5.4 Proposed Future Improvements

5.4.1 Administrative Interface Enhancement

Custom Administrative Dashboard

Development of a user-friendly administrative interface specifically designed for non-technical users:

- **Simplified Navigation**: Intuitive menu structure with role-based access control
- **Visual Workflow Management**: Drag-and-drop interface for managing candidate pipeline stages
- **Customizable Widgets**: Personalized dashboard components for different administrative roles
- **Guided Workflows**: Step-by-step wizards for complex administrative tasks

5.4.2 Enhanced Security and Anti-Cheating Measures

Advanced Proctoring System

Implementation of comprehensive examination security features:

- **Browser Lockdown**: Prevent access to external resources during assessments
- **Webcam Monitoring**: Optional video proctoring for high-stakes assessments
- **Screen Recording**: Comprehensive session recording for review purposes

5.4.3 Candidate Experience Improvements

Enhanced User Interface

Comprehensive redesign of the candidate-facing interface:

- **Progressive Web App (PWA)**: Mobile-first design with offline capability
- **Multilingual Support**: Internationalization for French, English, and Arabic languages

Advanced Assessment Capabilities

Implementation of more sophisticated evaluation methods:

- **Coding Sandbox**: Integrated development environment for practical programming assessments
- **Portfolio Review System**: Structured evaluation framework for candidate work samples
- **Video Interview Integration**: Built-in video conferencing for remote interviews

5.5 Impact Assessment and Validation

5.5.1 Quantitative Metrics

The platform's success can be measured through several key performance indicators:

- **Processing Efficiency**: Significant reduction in application processing time through automated workflows
- **System Reliability**: High uptime maintained during peak recruitment periods
- **User Adoption**: Strong administrative staff adoption following deployment
- **Cost Effectiveness**: Notable reduction in recruitment-related administrative costs

5.5.2 Qualitative Benefits

Beyond measurable metrics, the platform has delivered significant qualitative improvements:

- **Professional Image**: Enhanced VOID Digital Agency's reputation as a technology-forward organization
- **Candidate Experience**: Improved perception of the recruitment process among applicants
- **Staff Satisfaction**: Reduced administrative burden and improved job satisfaction for recruitment team
- **Scalability Foundation**: Established infrastructure for future growth and expansion

5.6 Summary

The Intern Management Platform represents a successful digital transformation initiative that has significantly improved VOID Digital Agency's recruitment processes. While the current implementation addresses the primary objectives and delivers substantial value, the identified limitations provide clear direction for future development efforts. The platform's modular architecture and solid technical foundation ensure that proposed enhancements can be implemented incrementally, allowing for continuous improvement and adaptation to evolving organizational needs.

The success of this project demonstrates the value of thoughtful system design, user-centered development approaches, and comprehensive testing methodologies. Future iterations should focus on enhancing user experience, implementing advanced security measures, and leveraging emerging technologies to maintain competitive advantage in the talent acquisition landscape.

Conclusion

The Intern Management Platform transformed how VOID Digital Agency handles recruitment. Instead of managing 600+ applications through scattered emails and spreadsheets, they now have a streamlined system that works for everyone involved. This internship experience at VOID Digital Agency provided far more than just technical skills development. Working on a real project with actual business impact taught valuable lessons about software architecture, user experience design, and collaborative development practices. The challenges facedfrom debugging complex API integrations to optimizing Docker configurationsbuilt confidence in tackling unfamiliar technologies and problem-solving under pressure. Beyond the technical achievements, this experience demonstrated the importance of understanding user needs, communicating effectively with stakeholders, and writing maintainable code that others can work with. These skills, combined with hands-on experience in modern development practices like CI/CD, provide a solid foundation for entering the professional software development field. The internship proved that good software development is about more than writing codeit's about creating solutions that genuinely help people do their jobs better and more efficiently.

List of Acronyms

AI *Artificial Intelligence*

API *Application Programming Interface*

CI/CD *Continuous Integration/Continuous Deployment*

CMS *Content Management System*

CTA *Call To Action*

CV *Curriculum Vitae*

DevOps *Development and Operations*

Drush *Drupal Shell*

GitOps *Git Operations*

OWASP *Open Web Application Security Project*

PHP-FPM *PHP FastCGI Process Manager*

PMO *Project Management Office*

QA *Quality Assurance*

SDLC *Software Development Life Cycle*

SEO/SEA *Search Engine Optimization/Search Engine Advertising*

TDD *Test-Driven Development*

UX/UI *User Experience/User Interface*

Appendix B: Glossary

Headless CMS: A content management system that provides content through APIs without a traditional frontend. The content is stored and managed in the backend but can be displayed on any device or platform through API calls.

JSON:API: A specification for building APIs in JSON format that defines how resources should be formatted, how relationships should be handled, and how data should be queried. It provides a standardized way to structure API responses.

Monorepo: A software development strategy where multiple related projects are stored in a single repository. This approach facilitates code sharing, dependency management, and coordinated development across projects.

Slug: A user-friendly URL identifier derived from a page title or content name.

UUID (Universally Unique Identifier): A 128-bit number used to uniquely identify information in computer systems. In Drupal, UUIDs are used to identify content entities across different environments.

Widget: A reusable component that encapsulates specific functionality and can be embedded in pages or other widgets. In this project, widgets bridge Drupal content management with Next.js component rendering.

Content Type: A predefined template in Drupal that defines the structure and fields for a specific type of content. Examples include "Job Offer," "Candidature," or "Quiz." Each content type can have custom fields and behaviors.

Entity: The fundamental unit of content in Drupal. Entities can be nodes (content), users, taxonomy terms, or custom entities. Each entity has fields that store data and can have relationships with other entities.

Hook: A PHP function that allows modules to interact with Drupal core and other modules. Hooks follow a naming convention (hook_event_name) and are called at specific points in Drupal's execution to allow custom functionality.

Paragraph: A Drupal entity type that allows content creators to build flexible, structured content by combining different paragraph types. Each paragraph type can have different fields and layouts.

Taxonomy: A system for classifying and organizing content using terms and vocabularies. For example, a "Specialization" vocabulary might contain terms like "Frontend Development," "Backend Development," and "DevOps."

Vocabulary: A collection of taxonomy terms used for categorizing content. Multiple vocabularies can exist for different classification systems (e.g., Tags, Categories, Specializations).

Webform: A Drupal module that provides form building capabilities, allowing administrators to create complex forms with various field types, validation rules, and submission handling logic.

AccessHandler Controller: A Drupal class responsible for determining whether a user has permission to perform specific operations (view, edit, delete) on entities. It implements access control logic based on user roles and permissions.

Webpack: A module bundler for JavaScript applications that processes and optimizes assets (JavaScript, CSS, images) for production deployment. It can also run custom plugins during the build process.

Appendix C: Technical Details

.1 Docker Configuration Files

.1.1 Frontend Development Docker Configuration

```
version: "3.4"
services:
  cache:
    container_name: "${PROJECT_NAME}_frontend_cache"
    image: redis:6.2-alpine
    restart: always
    ports:
      - "6379:6379"
    command: redis-server --save 20 1 --loglevel warning
       --requirepass eYVX7EwVmmxKPCDmwMtyKVge8oLd2t81
    volumes:
      - cache:/data

  redis-commander:
    image: rediscommander/redis-commander:latest
    restart: always
    depends_on:
      - cache
    environment:
      REDIS_HOSTS: "${PROJECT_NAME}_frontend_cache"
      REDIS_PASSWORD: eYVX7EwVmmxKPCDmwMtyKVge8oLd2t81
      HTTP_USER: root
      HTTP_PASSWORD: root
    ports:
      - 8081:8081

volumes:
  cache:
    driver: local
```

Listing 1: Frontend Development Docker Configuration

.1.2 Backend Development Docker Configuration

```yaml
version: "3.4"
services:
  php:
    container_name: "${PROJECT_NAME}_php"
    build:
      context: ./
      dockerfile: .docker/php/Dockerfile
    environment:
      IS_DOCKER: "true"
      DOCKER_DB_NAME: "${PROJECT_NAME}_db"
      DOCKER_DB_USER: "${PROJECT_NAME}_user"
      DOCKER_DB_PASSWORD: "123456"
      DOCKER_DB_HOST: "${PROJECT_NAME}_db"
      FRONTEND_URL: "http://host.docker.internal:3000"
    depends_on:
      - db
      - memcached
    volumes:
      - ./:/var/www/html
      - ./.docker/php/default.ini:/etc/php81/conf.d/custom.ini:ro
    ports:
      - "8080:80"

  db:
    container_name: "${PROJECT_NAME}_db"
    image: mysql:8.0
    environment:
      MYSQL_ROOT_PASSWORD: root
      MYSQL_DATABASE: "${PROJECT_NAME}_db"
      MYSQL_USER: "${PROJECT_NAME}_user"
      MYSQL_PASSWORD: "123456"
    volumes:
      - dbdata:/var/lib/mysql
    ports:
      - "3306:3306"

  phpmyadmin:
    container_name: "${PROJECT_NAME}_phpmyadmin"
    image: phpmyadmin/phpmyadmin:latest
    environment:
      PMA_HOST: "${PROJECT_NAME}_db"
      PMA_USER: root
      PMA_PASSWORD: root
    depends_on:
      - db
    ports:
      - "8081:80"

  memcached:
    container_name: "${PROJECT_NAME}_memcached"
    image: memcached:1.6-alpine
```

```
    ports:
      - "11211:11211"

  mailhog:
    container_name: "${PROJECT_NAME}_mailhog"
    image: mailhog/mailhog:latest
    ports:
      - "8025:8025"
      - "1025:1025"

volumes:
  dbdata:
    driver: local
```

Listing 2: Backend Development Docker Architecture

.1.3 Frontend Production Docker Configuration

```
version: "3.4"
services:
  proxy:
    image: vactory/nginx-lua:2.4
    environment:
      RUNTIME_UPSTREAM_NEXT_JS_BACKEND:
         "${PROJECT_NAME}_frontend_app:3000"
    depends_on:
      - starter
    labels:
      - "traefik.enable=true"
      -
         "traefik.http.routers.${PROJECT_NAME}_https.rule=Host(‘${APP_DOMAIN]
      -
         "traefik.http.routers.${PROJECT_NAME}_https.tls.certresolver=k8sprej
      -
         "traefik.http.routers.${PROJECT_NAME}_https.entrypoints=websecure"
    networks:
      - traefik

  starter:
    container_name: "${PROJECT_NAME}_frontend_app"
    build:
      context: .
      dockerfile: ./.docker/Dockerfile.easypanel.nextjs
      target: runner_starter
    environment:
      REDIS_HOST: "${PROJECT_NAME}_frontend_cache"
      REDIS_PASSWORD: eYVX7EwVmmxKPCDmwMtyKVge8oLd2t81
      DRUPAL_BASE_URL: "https://api.${PROJECT_BASE_DOMAIN}"
    volumes:
      - frontendAppDB:/app/apps/${APP_NAME}/database
```

```
29      depends_on:
30        - cache
31      networks:
32        - traefik
33
34    ui:
35      build:
36        context: .
37        dockerfile: ./.docker/Dockerfile.easypanel.storybook
38        target: runner_ui
39      labels:
40        - "traefik.enable=true"
41        -
             "traefik.http.routers.${PROJECT_NAME}_ui_https.rule=Host(‘ui.${PROJ
42        -
             "traefik.http.routers.${PROJECT_NAME}_ui_https.tls.certresolver=k8s
43      networks:
44        - traefik
45
46    cache:
47      container_name: "${PROJECT_NAME}_frontend_cache"
48      image: redis:6.2-alpine
49      restart: always
50      command: redis-server --save 20 1 --loglevel warning
           --requirepass eYVX7EwVmmxKPCDmwMtyKVge8oLd2t81
51      volumes:
52        - cache:/data
53      networks:
54        - traefik
55
56    redis-commander:
57      image: rediscommander/redis-commander:latest
58      restart: always
59      depends_on:
60        - cache
61      environment:
62        REDIS_HOSTS: "${PROJECT_NAME}_frontend_cache"
63        REDIS_PASSWORD: eYVX7EwVmmxKPCDmwMtyKVge8oLd2t81
64        HTTP_USER: root
65        HTTP_PASSWORD: root
66      labels:
67        - "traefik.enable=true"
68        -
             "traefik.http.routers.${PROJECT_NAME}_redis_https.rule=Host(‘redis.
69        -
             "traefik.http.routers.${PROJECT_NAME}_redis_https.tls.certresolver=
70      networks:
71        - traefik
72
73  volumes:
74    frontendAppDB:
```

```
    driver: local
  cache:
    driver: local

networks:
  traefik:
    external: true
```

Listing 3: Frontend Production Docker Configuration

.1.4 Backend Production Docker Configuration

```
version: "3.4"
services:
  php:
    container_name: "${PROJECT_NAME}_php"
    build:
      context: ./
      dockerfile: .docker/php/Dockerfile
    environment:
      IS_DOCKER: "true"
      DOCKER_DB_NAME: "${PROJECT_NAME}_db"
      DOCKER_DB_USER: "${PROJECT_NAME}_user"
      DOCKER_DB_PASSWORD: "${DB_PASSWORD}"
      DOCKER_DB_HOST: "${PROJECT_NAME}_db"
      FRONTEND_URL: "https://${PROJECT_BASE_DOMAIN}"
      MEMCACHED_HOST: "${PROJECT_NAME}_memcached"
    volumes:
      - public_files_data:/var/www/html/sites/default/files
      - private_files_data:/var/www/html/sites/default/private
      - backup_db:/var/www/backup
    labels:
      - "traefik.enable=true"
      -
        "traefik.http.routers.drupal_${PROJECT_NAME}_https.rule=Host(‘backer
      -
        "traefik.http.routers.drupal_${PROJECT_NAME}_https.tls.certresolver=
      -
        "traefik.http.routers.drupal_${PROJECT_NAME}_api_https.rule=Host(‘ap
      -
        "traefik.http.routers.drupal_${PROJECT_NAME}_api_https.tls.certresol
      -
        "traefik.http.routers.drupal_${PROJECT_NAME}_media_https.rule=Host(
      -
        "traefik.http.routers.drupal_${PROJECT_NAME}_media_https.tls.certres
    depends_on:
      - db
      - memcached
    networks:
      - traefik
```

```
33
34  db:
35    container_name: "${PROJECT_NAME}_db"
36    build:
37      context: .
38      dockerfile: .docker/mysql/Dockerfile
39    environment:
40      MYSQL_ROOT_PASSWORD: "${DB_ROOT_PASSWORD}"
41      MYSQL_DATABASE: "${PROJECT_NAME}_db"
42      MYSQL_USER: "${PROJECT_NAME}_user"
43      MYSQL_PASSWORD: "${DB_PASSWORD}"
44    volumes:
45      - dbdata:/var/lib/mysql
46    networks:
47      - traefik
48
49  phpmyadmin:
50    container_name: "${PROJECT_NAME}_phpmyadmin"
51    image: phpmyadmin/phpmyadmin:latest
52    environment:
53      PMA_HOST: "${PROJECT_NAME}_db"
54      PMA_USER: root
55      PMA_PASSWORD: "${DB_ROOT_PASSWORD}"
56      PMA_ABSOLUTE_URI:
          "https://phpmyadmin.${PROJECT_BASE_DOMAIN}/"
57    depends_on:
58      - db
59    labels:
60      - "traefik.enable=true"
61      -
          "traefik.http.routers.${PROJECT_NAME}_phpmyadmin_https.rule=Host(`pl
62      -
          "traefik.http.routers.${PROJECT_NAME}_phpmyadmin_https.tls.certresol
63      -
          "traefik.http.routers.${PROJECT_NAME}_phpmyadmin_https.middlewares=a
64    networks:
65      - traefik
66
67  memcached:
68    container_name: "${PROJECT_NAME}_memcached"
69    image: memcached:1.6-alpine
70    networks:
71      - traefik
72
73  filebrowser:
74    container_name: "${PROJECT_NAME}_filebrowser"
75    image: filebrowser/filebrowser
76    volumes:
77      - private_files_data:/srv/private
78      - public_files_data:/srv/public
79      - backup_db:/srv/backup/database
```

```
80      labels:
81        - "traefik.enable=true"
82        -
             "traefik.http.routers.${PROJECT_NAME}_files_https.rule=Host(‘files.$
83        -
             "traefik.http.routers.${PROJECT_NAME}_files_https.tls.certresolver=l
84        -
             "traefik.http.routers.${PROJECT_NAME}_files_https.middlewares=auth"
85      networks:
86        - traefik
87
88  volumes:
89    dbdata:
90      driver: local
91    public_files_data:
92      driver: local
93    private_files_data:
94      driver: local
95    backup_db:
96      driver: local
97
98  networks:
99    traefik:
100     external: true
```

Listing 4: Backend Production Configuration

.2 Multi-Stage Docker Build Implementation

```
1  # Stage 1: Workspace preparation
2  FROM registry.access.redhat.com/ubi9/nodejs-22:9.5-1746003035 AS
       workspace
3  WORKDIR /app
4  COPY ["package.json", "yarn.lock", "./"]
5  COPY packages ./packages
6  COPY apps ./apps
7  RUN find packages \! -name "package.json" -mindepth 2 -maxdepth
       2 -print | xargs rm -rf
8
9  # Stage 2: Dependency installation
10 FROM registry.access.redhat.com/ubi9/nodejs-22:9.5-1746003035 AS
       deps
11 WORKDIR /app
12 RUN npm install yarn@1.22.19 --global
13 COPY --from=workspace /app ./
14 RUN yarn install --frozen-lockfile
15
16 # Stage 3: Application building
17 FROM registry.access.redhat.com/ubi9/nodejs-22:9.5-1746003035 AS
       builder_starter
```

```
WORKDIR /app
COPY --from=deps /app ./
COPY . .
RUN NODE_ENV=production yarn workspace ${APP_NAME} build

# Stage 4: Production runtime
FROM registry.access.redhat.com/ubi9/nodejs-22:9.5-1746003035 AS
   runner_starter
WORKDIR /app

ENV NODE_ENV=production
ENV NEXT_TELEMETRY_DISABLED=1

RUN addgroup --system --gid 1001 nodejs
RUN adduser --system --uid 1001 nextjs

COPY --from=builder_starter
   /app/apps/${APP_NAME}/.next/standalone ./
COPY --from=builder_starter /app/apps/${APP_NAME}/.next/static
   ./apps/${APP_NAME}/.next/static
COPY --from=builder_starter /app/apps/${APP_NAME}/public
   ./apps/${APP_NAME}/public

USER nextjs

EXPOSE 3000

ENV PORT=3000
ENV HOSTNAME="0.0.0.0"

CMD ["node", "/app/apps/$APP_FOLDER/server.js"]
```

Listing 5: Multi-Stage Docker Build Process

.3 Frontend Implementation Code

.3.1 Webpack Plugin Architecture

```
const path = require('path');
const fs = require('fs');

class WidgetsPlugin {
  constructor(options = {}) {
    this.options = {
      pattern: '**/*Widget.jsx',
      outputFile: 'widgets.js',
      ...options
    };
  }
```

```
12
13    apply(compiler) {
14      compiler.hooks.compilation.tap('WidgetsPlugin',
           (compilation) => {
15        compilation.hooks.additionalAssets.tapAsync('WidgetsPlugin',
             (callback) => {
16          this.generateWidgetMapping(compilation, callback);
17        });
18      });
19    }
20
21    generateWidgetMapping(compilation, callback) {
22      const glob = require('glob');
23      const widgetFiles = glob.sync(this.options.pattern, {
24        cwd: compilation.options.context
25      });
26
27      const mappings = [];
28      const imports = [];
29
30      widgetFiles.forEach((filePath) => {
31        try {
32          const fullPath =
               path.resolve(compilation.options.context, filePath);
33          const config = this.extractWidgetConfig(fullPath);
34
35          if (config && config.id) {
36            if (config.lazy === true) {
37              mappings.push(`"${config.id}": dynamic(() =>
                   import("${filePath}"))`);
38            } else {
39              const exportName = config.id
40                .replaceAll(":", "__")
41                .replaceAll("-", "_");
42              const exportNameDefault =
                   exportName.charAt(0).toUpperCase() +
                   exportName.slice(1);
43
44              imports.push(`import ${exportNameDefault} from
                   "${filePath}"`);
45              mappings.push(`"${config.id}":
                   ${exportNameDefault}`);
46            }
47          }
48        } catch (error) {
49          console.warn(`Failed to process widget file
               ${filePath}:`, error.message);
50        }
51      });
52
53      const outputContent = `
```

```
import dynamic from "next/dynamic";
${imports.join('\n')}

export const Widgets = {
  ${mappings.join(',\n  ')}
};

export default Widgets;
`;

    const outputPath = path.resolve(
      compilation.options.output.path,
      this.options.outputFile
    );

    fs.writeFileSync(outputPath, outputContent);
    callback();
  }

  extractWidgetConfig(filePath) {
    try {
      const content = fs.readFileSync(filePath, 'utf8');
      const configMatch =
         content.match(/export\s+const\s+config\s*=\s*({[\s\S]*?});/);

      if (configMatch) {
        // Simple config extraction - in a real implementation,
        // you might want to use a proper JS parser
        const configStr = configMatch[1];
        return eval(`(${configStr})`);
      }
    } catch (error) {
      console.warn(`Failed to extract config from ${filePath}:`,
         error.message);
    }

    return null;
  }
}

module.exports = WidgetsPlugin;

// Usage in webpack.config.js
const WidgetsPlugin = require('./plugins/WidgetsPlugin');

module.exports = {
  // ... other webpack config
  plugins: [
    new WidgetsPlugin({
      pattern: 'components/**/*Widget.jsx',
      outputFile: 'widgets.js'
```

```
    }),
    // ... other plugins
  ]
};
```

Listing 6: Complete Webpack Widget Plugin Implementation

.3.2 Offline Mode Implementation

```
// lib/offline-manager.js
import { createContext, useContext, useState, useEffect } from
   'react';

const OfflineContext = createContext();

export const OfflineProvider = ({ children }) => {
  const [isOffline, setIsOffline] = useState(false);
  const [cachedData, setCachedData] = useState(new Map());

  useEffect(() => {
    const handleOnline = () => setIsOffline(false);
    const handleOffline = () => setIsOffline(true);

    window.addEventListener('online', handleOnline);
    window.addEventListener('offline', handleOffline);

    // Initial state
    setIsOffline(!navigator.onLine);

    return () => {
      window.removeEventListener('online', handleOnline);
      window.removeEventListener('offline', handleOffline);
    };
  }, []);

  const value = {
    isOffline,
    cachedData,
    setCachedData
  };

  return (
    <OfflineContext.Provider value={value}>
      {children}
    </OfflineContext.Provider>
  );
};

export const useOffline = () => {
  const context = useContext(OfflineContext);
```

```
  if (!context) {
    throw new Error('useOffline must be used within
       OfflineProvider');
  }
  return context;
};

// lib/api-with-offline.js
import axios from 'axios';
import { useOffline } from './offline-manager';

class ApiWithOffline {
  constructor() {
    this.axios = axios.create({
      baseURL: process.env.NEXT_PUBLIC_API_BASE_URL
    });

    this.setupInterceptors();
  }

  setupInterceptors() {
    // Request interceptor
    this.axios.interceptors.request.use(
      (config) => {
        // Add timestamp for cache validation
        config.metadata = { startTime: new Date() };
        return config;
      },
      (error) => Promise.reject(error)
    );

    // Response interceptor
    this.axios.interceptors.response.use(
      (response) => {
        // Cache successful responses
        this.cacheResponse(response.config.url, response.data);
        return response;
      },
      async (error) => {
        const { config } = error;

        // If network error and we have cached data, return it
        if (this.isNetworkError(error) && config.url) {
          const cachedData = this.getCachedData(config.url);
          if (cachedData) {
            console.warn('Network error, serving cached data:',
               config.url);
            return {
              data: cachedData,
              status: 200,
              statusText: 'OK (Cached)',
```

```
90                headers: {},
91                config,
92                fromCache: true
93              };
94            }
95          }
96
97          return Promise.reject(error);
98        }
99      );
100   }
101
102   isNetworkError(error) {
103     return (
104       error.code === 'NETWORK_ERROR' ||
105       error.message === 'Network Error' ||
106       !error.response
107     );
108   }
109
110   cacheResponse(url, data) {
111     if (typeof window !== 'undefined' && window.localStorage) {
112       try {
113         const cacheKey = `api_cache_${btoa(url)}`;
114         const cacheData = {
115           data,
116           timestamp: Date.now(),
117           url
118         };
119
120         localStorage.setItem(cacheKey,
              JSON.stringify(cacheData));
121
122         // Also store in Redis if available
123         this.storeInRedis(url, data);
124       } catch (error) {
125         console.warn('Failed to cache response:', error);
126       }
127     }
128   }
129
130   getCachedData(url) {
131     if (typeof window !== 'undefined' && window.localStorage) {
132       try {
133         const cacheKey = `api_cache_${btoa(url)}`;
134         const cached = localStorage.getItem(cacheKey);
135
136         if (cached) {
137           const parsedCache = JSON.parse(cached);
138           const isExpired = Date.now() - parsedCache.timestamp >
              3600000; // 1 hour
```

```
139
140           if (!isExpired) {
141             return parsedCache.data;
142           } else {
143             localStorage.removeItem(cacheKey);
144           }
145         }
146       } catch (error) {
147         console.warn('Failed to retrieve cached data:', error);
148       }
149     }
150
151     return null;
152   }
153
154   async storeInRedis(url, data) {
155     try {
156       await fetch('/api/cache', {
157         method: 'POST',
158         headers: {
159           'Content-Type': 'application/json'
160         },
161         body: JSON.stringify({
162           key: url,
163           data,
164           ttl: 3600 // 1 hour
165         })
166       });
167     } catch (error) {
168       console.warn('Failed to store in Redis:', error);
169     }
170   }
171
172   async get(url, config = {}) {
173     return this.axios.get(url, config);
174   }
175
176   async post(url, data, config = {}) {
177     return this.axios.post(url, data, config);
178   }
179
180   async put(url, data, config = {}) {
181     return this.axios.put(url, data, config);
182   }
183
184   async delete(url, config = {}) {
185     return this.axios.delete(url, config);
186   }
187 }
188
189 export const apiWithOffline = new ApiWithOffline();
```

```
190
191 // components/OfflineIndicator.jsx
192 import { useOffline } from '../lib/offline-manager';
193
194 export const OfflineIndicator = () => {
195   const { isOffline } = useOffline();
196
197   if (!isOffline) return null;
198
199   return (
200     <div className="fixed top-0 left-0 right-0 bg-yellow-500
          text-black p-2 text-center z-50">
201       <span className="font-medium">
202         You're currently offline. Showing cached content.
203       </span>
204     </div>
205   );
206 };
207
208 // pages/api/cache.js (Next.js API route)
209 import Redis from 'ioredis';
210
211 const redis = new Redis(process.env.REDIS_URL);
212
213 export default async function handler(req, res) {
214   if (req.method === 'POST') {
215     const { key, data, ttl = 3600 } = req.body;
216
217     try {
218       const cacheKey =
            `offline_cache:${Buffer.from(key).toString('base64')}`;
219       await redis.setex(cacheKey, ttl, JSON.stringify(data));
220
221       res.status(200).json({ success: true });
222     } catch (error) {
223       console.error('Redis cache error:', error);
224       res.status(500).json({ error: 'Failed to cache data' });
225     }
226   } else if (req.method === 'GET') {
227     const { key } = req.query;
228
229     try {
230       const cacheKey =
            `offline_cache:${Buffer.from(key).toString('base64')}`;
231       const cached = await redis.get(cacheKey);
232
233       if (cached) {
234         res.status(200).json({ data: JSON.parse(cached) });
235       } else {
236         res.status(404).json({ error: 'Data not found in cache'
            });
```

```
237         }
238       } catch (error) {
239         console.error('Redis retrieval error:', error);
240         res.status(500).json({ error: 'Failed to retrieve cached
              data' });
241       }
242     } else {
243       res.setHeader('Allow', ['GET', 'POST']);
244       res.status(405).end(`Method ${req.method} Not Allowed`);
245     }
246 }
```

Listing 7: Offline Mode Implementation

This appendix provides all the technical implementation details that support the main chapters of this report, ensuring complete documentation of the platform's architecture and functionality.

Bibliography

Richardson, C. (2018). Microservices patterns: With examples in java. In *Manning Publications*, pages 1–520.