Students & Companies

Implementation and Testing Document

Francesco Ostidich, Matteo Salari, Francesco Rivitti

February 16, 2025

Contents

1 Introduction

This document outlines the implementation and testing procedures that have been followed in order to develop a functioning prototype of the service described in the ”Requirements Analysis and Specification Document” and ”Design Document”.

1.1 Definitions, acronyms, abbreviations

1.1.1 Definitions

- **Internship project**: the description of the skills, technologies and roles the student will be working with during the internship, along with the set of tasks that will be covered
- **Internship advertisement**: the public post created by companies to promote available internships on the platform, aimed at attracting suitable candidates by highlighting its key aspects
- **Internship information**: general data about the (ongoing) internship, including the elapsed and remaining time, the compensation and the description of the project the student is working on
- **Enrollment request**: the submission of a student to indicate interest in a specific internship, initiating the selection process by formally applying
- **Enrollment suggestion**: the recommendation made by the platform to guide students in finding projects that best suit them
- **Custom questionnaire**: the tailored set of questions used by companies during interviews to assess a candidate fit for the internship
- **Candidate student**: a student who has applied for an internship and is currently under consideration by a company, moving forward in the selection process
- **Eligible student**: a student who meets the qualifications for an internship, making them viable candidates for recommendation and application
- **Suitable student**: a student who meets the qualifications for an internship, making them potential candidates to be recommended in the companies feed
- **Complaint**: a report submitted by a student or company to the university, regarding issues during the internship, such as unmet expectations, mistreatments, or procedural problems
- **Feedback form**: a structured form for students and companies used to provide feedback on their internship experience, enabling the platform to gather data for analysis, improvements, and recommendations

1.1.2 Acronyms

- **S&C**: Students&Companies

1.1.3 Abbreviations

- **Rn**: n-th requirement
- **KFn**: n-th key function

1.2 Revision history

- **Revised on**: February 16, 2025
- **Version**: 1.0
- **Description**: document initial release

1.3 Reference documents

- **Polimi Software Engineering 2 AY 2024/2025 assignment document**: goal, schedule and rules of the requirement engineering and design project
- **Polimi Software Engineering 2 AY 2024/2025 course slides**: the lecture slides provided during the course

1.4 Document structure

- **Chapter 1**: this section presents the problem statement and outlines the system objectives. Here are provided the essential initial resources for the readers, to facilitate a comprehensive understanding of the document.
- **Chapter 2**: here are presented all the system functions which the prototype implements.
- **Chapter 3**: this section introduces the programming languages and frameworks that have been chosen, providing comprehensive justifications for each selection.
- **Chapter 4**: this section presents the structure of the code.
- **Chapter 5**: here is provided the information regarding the testing process, specifically outlining the functions that have undergone testing and elucidating the methodology employed.
- **Chapter 6**: this section offer instructions that comprehensively guide users on how to install and run the prototype.
- **Chapter 7**: here is found an estimation of the effort spent by each group member.
- **Chapter 8**: here is provided a list of the references used in this document.

2 Product functions and requirements

2.1 Product functions

2.1.1 Key functions

KF1 - Internship project advertisements - Implemented - Companies can post detailed internship project descriptions, including tasks, required skills, technologies, and terms like compensation and benefits. This provides students clear insights about the available opportunities.

KF2 - CV upload - Implemented - Students can upload and manage their CVs, listing skills, experiences, and interests. This helps the system match students with suitable internships and allows companies to view candidate profiles.

KF3 - Interviews via custom questionnaires - Partially implemented: questionnaires come in the form of a single question with the relative answer - S&C supports the interview process by allowing companies to set up structured questionnaires, enabling them to gather specific information from candidates and assess their compatibility with the internship.

KF4 - Complaints management - Not implemented - The platform includes a complaints management feature where students and companies can report issues to the university, that can then address is accordingly, ensuring smooth and fair handling of internship-related problems.

KF5 - Recommendation system - Implemented - S&C employs a recommendation system that connects students with internships based on CV content and project requirements, using keyword matching statistical analyses, to optimize matches.

KF6 - Suggestion system - Not implemented - The platform provides guidance for students and companies on improving their CVs and project descriptions, enhancing their appeal and increasing the likelihood of successful matches.

2.2 Requirements

R1 - Implemented - The system must allow an unregistered student to sign up.

R2 - Implemented - The system must allow an unregistered company to sign up.

R3 - Not implemented - The system must allow an unregistered university to sign up.

R4 - Implemented - The system must allow a registered user to log in.

R5 - Implemented - The system must allow a registered user to fill in and edit its personal information.

R6 - Implemented - The system must allow a registered student to upload its CV.

R7 - Implemented - The system must allow a registered company to post an internship project.

R8 - Implemented - The system must allow a registered student to visualize a list of open internship projects.
R9 - Implemented - The system must allow a registered company to visualize a list of eligible students.
R10 - Implemented - The system must allow a registered student to make an enrollment request to an internship project.
R11 - Partially implemented - The system must allow a registered company to build custom made questionnaires.
R12 - Partially implemented - The system must allow a registered company to send questionnaires to students.
R13 - Partially implemented - The system must allow a registered student to fill in the questionnaire.
R14 - Implemented - The system must allow a registered company to accept students enrollment requests.
R15 - Implemented - The system must allow a registered student to see their ongoing internship information.
R16 - Implemented - The system must allow a registered company to see their ongoing internships information.
R17 - Not implemented - The system must allow a registered university to see their students ongoing internship information.
R18 - Not implemented - The system must allow a registered student to send complaints to the university.
R19 - Not implemented - The system must allow a registered company to send complaints to the university.
R20 - Not implemented - The system must allow a registered university to visualize complaints it received.
R21 - Not implemented - The system must allow a registered university to end an ongoing internship of its student.
R22 - Implemented - The system must allow a registered student to fill in a feedback form when the internship ends.
R23 - Implemented - The system must allow a registered company to fill in a feedback form when the internship ends.
R24 - Implemented - The system must allow a registered student to visualize a list of suggested internships.
R25 - Implemented - The system must allow a registered company to visualize a list of suggested students.
R26 - Implemented - The system must allow a registered student to be notified about recommended internship.
R27 - Not implemented - The system must allow a registered company to be notified about recommended students.

3 Development, frameworks and tools

3.1 Programming languages

3.1.1 C#

C# was selected as the backend programming language for the application, due to the following advantages.

- Platform independence: The .NET Runtime allows to create code that can run on various operating systems without modifications, ensuring a broad compatibility.
- Robust ecosystem: C# boasts a mature and extensive ecosystem, offering a wide range of libraries and frameworks. This rich set of tools provides a solid foundation for building scalable and maintainable backend solutions.
- Community support: C# enjoys strong community support, with a vast developer community contributing to its continuous improvement. Regular updates from Microsoft enhance the language reliability and security.
- Object-oriented paradigm: C# object-oriented programming (OOP) paradigm aligns well with the complexity of web applications. This approach supports the development of modular, extensible, and organized code.
- Readability and maintainability: C# emphasizes readability and maintainability, making it easy to creating backend code that is not only efficient but also easy to manage in the long term.

Furthermore, the decision to utilize C# for developing the backend of our prototype was influenced by the chance to implement ASP.NET and EF frameworks, which would expedite the development process.

3.1.2 JavaScript

JavaScript is the standard language used to develop dynamic and interactive web applications. It has been used to create responsive user interfaces and handling client-side scripting for the frontend of the web application.

For an efficient developing process, is has been also integrated Vite.js and NPM into the development workflow, optimizing the frontend development process through rapid prototyping capabilities, efficient dependency management and a streamlined build process.

3.2 Frameworks and standards

3.2.1 ASP.NET Core

The decision to employ the ASP.NET Core framework for the backend of the application was driven by several advantages.

- Comprehensive ecosystem: ASP.NET Core provides a rich ecosystem for building scalable and high-performance applications. Its modular design allows selective use of components, enhancing flexibility.
- Dependency injection: built-in support for dependency injection promotes loose coupling, improving maintainability and testability.
- Middleware pipeline: ASP.NET Core's middleware-based architecture enables clean separation of concerns, improving modularity and readability.
- Integration capabilities: the framework seamlessly integrates with various databases, cloud services, and third-party libraries, making it suitable for enterprise-level applications.
- Minimal APIs and controllers: ASP.NET Core supports both Minimal APIs for lightweight development and MVC/Web API for structured, scalable solutions.
- Cross-platform: it runs on Windows, Linux, and macOS, providing flexibility in development and production deployment.

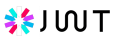
3.2.2 EF Core

The adoption of the Entity Framework Core (EF Core) for our backend is based on the following key benefits.

- Database independence: EF Core provides a standardized way to interact with databases, enabling easy switches between different database providers (e.g., SQL Server, PostgreSQL, MySQL) without major code changes.
- Object-relational mapping (ORM): EF Core's ORM capabilities simplify the mapping of C# objects to database entities, reducing the complexity of database interactions and speeding up development.
- Portability: EF Core supports multiple database providers, enhancing portability and minimizing vendor lock-in, allowing the application to run on various relational databases.
- Transaction management: EF Core offers built-in support for transaction management, ensuring data consistency and integrity by managing database transactions efficiently.

In this implementation, the DBMS specifically utilized is MySQL, but EF Core would allow to switch to other relational databases with minimal effort and minimal changes in code, for maintaining the benefits of ORM.

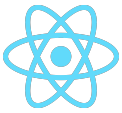
3.2.3 JWT Authentication



JSON Web Token (JWT) authentication was chosen as the preferred method for securing the web application, due to its numerous advantages and compatibility with our development stack.

- Stateless authentication: JWT is a stateless authentication mechanism, meaning that the server does not need to store any session information. This aligns well with the RESTful architecture of the application, allowing for scalability and easy maintenance.
- Token-based approach: JWT uses a token-based approach where a compact, URL-safe string is generated, containing information about the user's identity and additional claims. This token is then sent with each request, eliminating the need for continuous authentication checks against a database.
- Cross-Origin Resource Sharing (CORS) Compatibility: JWT is well-suited for applications with cross-origin requests, providing a secure way to include user authentication information in HTTP requests across different domains.
- Decentralized authorization: JWT allows for decentralized authorization by embedding user roles and permissions directly into the token. This decentralization enhances scalability and reduces the need for frequent database queries during authorization checks.
- Interoperability: JWT is a standardized, open-source format (RFC 7519), ensuring interoperability across different platforms and technologies. This makes it a suitable choice for our diverse tech stack involving C#, ASP.NET, and React.
- Expiration and refresh mechanism: JWT supports token expiration, enhancing security by limiting the lifespan of tokens. Additionally, a refresh mechanism can be implemented to obtain new tokens without requiring the user to re-enter credentials, providing a seamless user experience.

3.2.4 React



For the frontend development, React was chosen for the following reasons.

- Component-based architecture: React's component-based architecture promotes reusability and modularity, making it easier to manage complex user interfaces.
- Virtual DOM: React's use of a virtual DOM enhances performance by minimizing the need for direct manipulation of the actual DOM, resulting in faster updates and improved user experience.
- Declarative syntax: the declarative syntax of React simplifies the process of building interactive user interfaces, making the code more readable and easier to understand.
- Community and ecosystem: React has a vibrant and extensive community, along with a vast ecosystem of libraries and tools, which facilitates rapid development and problem-solving.

3.2.5 Vite.js

For the frontend build process, Vite.js was chosen for the following reasons.

- Lightning-fast development server: Vite leverages native ES modules and an optimized dependency pre-bundling process, providing near-instant cold starts and fast hot module replacement (HMR).
- Optimized build process: using Rollup under the hood, Vite produces highly optimized production builds with efficient code splitting and tree shaking.
- Framework-agnostic flexibility: Vite supports multiple frontend frameworks, including React, the one we used, as it is highly adaptable.
- Rich plugin ecosystem: Vite benefits from a growing ecosystem of plugins, including Rollup-compatible extensions, allowing for enhanced customization.

3.3 Tools

3.3.1 Nginx

Although not strictly necessary, and outside of the project implementation scope, for a secure and best-practice production deployment (beyond localhost), Nginx plays a key role in request routing and security.

- CORS elimination: as a reverse proxy, Nginx serves frontend and backend from the same origin, removing the need for CORS policies.
- HTTPS enablement: Nginx handles TLS termination, securing traffic with HTTPS and offloading SSL processing from backend services.
- Request routing: it efficiently directs incoming requests to the correct backend services based on URL paths or subdomains.
- Load balancing: if needed, Nginx distributes traffic across multiple instances, improving availability and performance.
- Static content hosting: it serves static assets efficiently, reducing backend load and improving response times.

Integrating Nginx ensures a secure, scalable, and streamlined production environment.

3.3.2 GNU Make

During the project developing process, Make has been used to automate build and run commands of all the necessary applications, like PDFLaTeX, .NET, NPM, Docker, and more. It streamlines workflows, ensuring consistency and simplifying complex tasks. By packaging commands together, it provides a unified setup for all developers, allowing everyone to use the correct commands without needing to learn their inner workings.

3.3.3 Docker

Docker has been integrated into the project to streamline development, deployment, and execution across different environments. Docker images have been created for both the web server and the application server. Furthermore, with the use of Docker Compose, the full deployment of the system (frontend, backend and database servers) can be runned all together with a single configuration file. It provides various benefits.

- Containerization: Docker ensures that applications run consistently across different machines by packaging dependencies and configurations into isolated containers.
- Cross-platform compatibility: it enables seamless execution across various operating systems, making development and deployment more efficient.
- Simplified dependency management: it eliminates conflicts by bundling all necessary libraries and tools within each container.

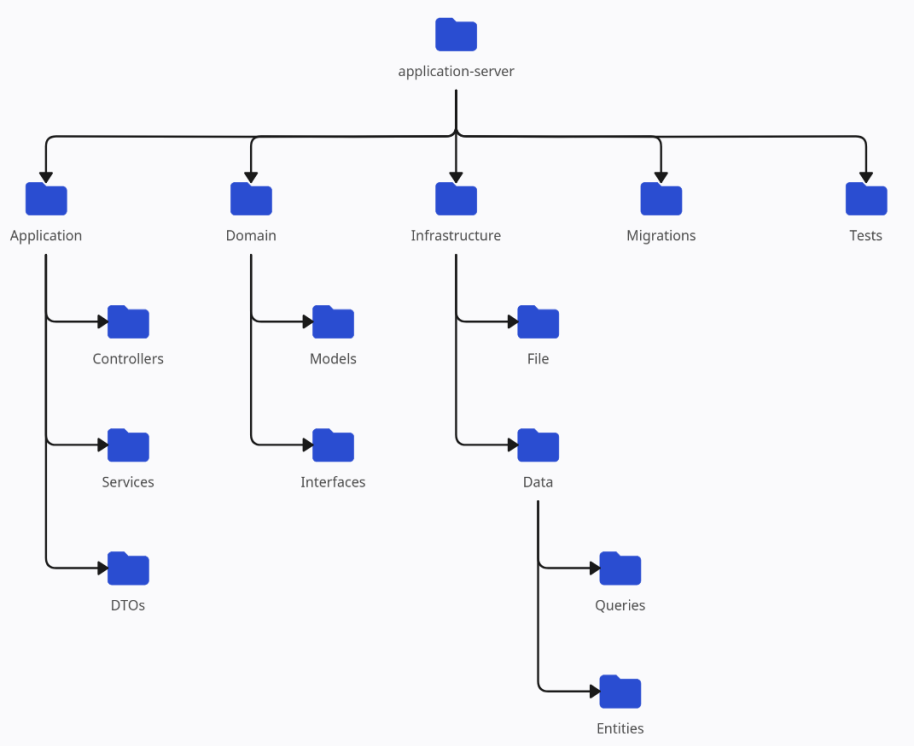
- Reproducible builds: Docker ensures that every developer and deployment instance runs the same version of the application.
- Scalability and deployment: rapid scaling and deployment is achieved by using container orchestration tools, such as Docker Compose.

By using Docker, a uniform and reliable environment in guaranteed, reducing setup complexity and enhancing maintainability.

4 Source code structure

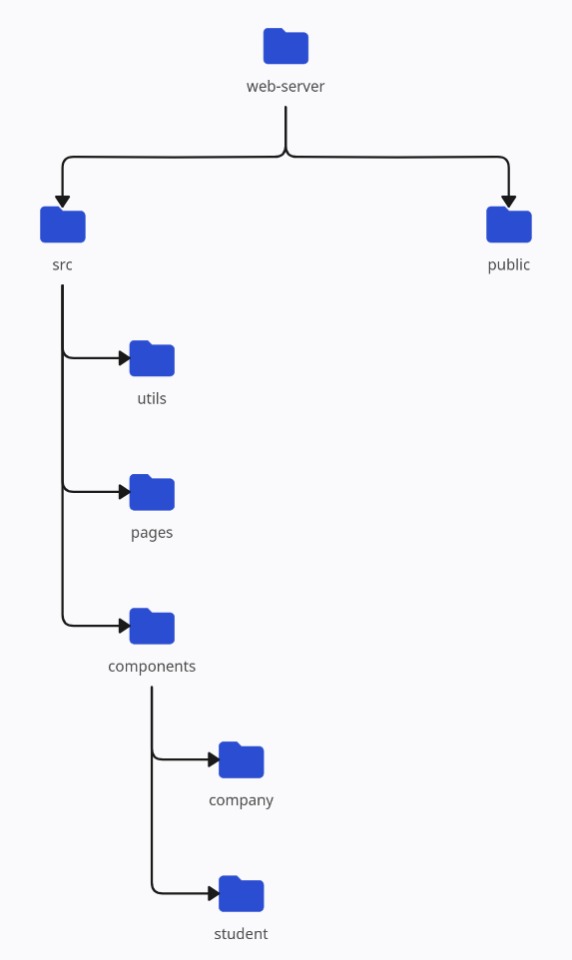
4.1 Application server

The application server project tree follows a distributed MVC approach combined with Clean Architecture principles. This structure ensures a clear separation of concerns, maintaining independent layers for application logic, domain rules, and infrastructure. By organizing the code in this way, we improve maintainability, scalability, and testability across the system.



4.2 Web server

The web server is organized with a modular structure to ensure scalability and maintainability. There are two main folders, one containing the core application logic, and the other holding static assets, supporting the separation of UI components and resources from the application's core functionality.



5 Testing

xUnit.net

As for the testing process, xUnit was used to implement comprehensive and exhaustive integration tests. These tests are designed to thoroughly evaluate the application's behavior by simulating real-world scenarios. A real, seeded in-memory database, that mirrors the production environment, was instantiated, ensuring that the tests run against a realistic data set and query behavior. Essentially, each test involves launching the entire application, allowing verifing that all components interact correctly during runtime.

Our integration tests cover the full scope of the application's logic, including database queries, business logic, and API responses. Each API endpoint is tested to ensure it responds correctly for every potential status code and response message. This exhaustive coverage ensures that all parts of the system, from data access to application responses, function as expected in a controlled test environment.

In the end, more than 200 test methods were ran and passed, for a total of almost 40 APIs.

6 Installation guide

Below are the steps to launch the system applications for local hosting.

6.1 Install Docker Compose

The installation guide for Docker Compose can be found at the following link.

https://docs.docker.com/compose/install

6.2 Download the configuration file

Download the `docker-compose.yml` file found in the root of the repository. To do so, you can simply open a terminal and run the following command.

```
curl -O https://raw.githubusercontent.com/Fostidich/\
OstidichSalariRivitti-StudentsAndCompanies/main/docker-compose.yml
```

6.3 Start the application

Stay in the folder containing the `docker-compose.yml` file that you've just downloaded.

Depending on how Docker Compose has been installed, run the corresponding command: use

```
docker compose up --pull always
```

if you installed Docker Compose as a plugin; use

```
docker-compose up --pull always
```

if you installed it as a standalone binary.

6.3.1 Troubleshoot

Since Docker Compose can be installed in multiple ways, and since slight differences can be found depending on the running OS, below is a list of possible solutions to try if the previous command fails.

- If you're using Docker Desktop, be sure that it has been previously launched.
- If on Windows, try to launch the terminal (CMD or PowerShell) with administrator rights.
- If on MacOS or Linux, try prepending a `sudo` before the previous commands.
- It may happen that the ports that have been set in the `docker-compose.yml` are already in use. If so, feel free to change them, but make sure they still match accordingly.

6.4 Browser page

Open the browser and go the the link below.

http://localhost:8080.

Mind changing the port, if you have modified it.

6.5 Stopping the container

To stop the containers, use

```
docker compose down
docker-compose down
```

depending on your installation.

6.5.1 Resources cleaning

To free up space and clean up unused Docker resources, you can run the following command, which will delete all unused containers, images, volumes, and networks.

```
docker system prune -a
```

6.5.2 Restart application

To start the server applications once more, simply repeat the commands found at point 3.

7 Effort spent

Unit	Member	Hours
Backend	Ostidich, Salari, Rivitti	120
Frontend	Rivitti	80
Testing	Ostidich, Salari	40

8 References

1. The document has been written using latex-project.org
2. The project repository has been uploaded on github.com
3. The diagrams have been created using creately.com
4. The system has been tested for public hosting capabilities on aws.amazon.com
5. C#: learn.microsoft.com/en-us/dotnet/csharp
6. JavaScript: developer.mozilla.org/en-us/docs/web/javascript
7. ASP.NET Core: learn.microsoft.com/en-us/aspnet/core
8. EF Core: learn.microsoft.com/en-us/ef/core
9. JWT Authentication: jwt.io
10. React: react.dev
11. Nginx: nginx.org
12. GNU Make: gnu.org/software/make
13. Docker: docker.com
14. xUnit: xunit.net