**Design document**

***CareNest***

*People for People NGO*

A logo of hands and flowers

Description automatically generated

|  |
| --- |
| **Date : 07.01.2025** |
| **Version : 2.1** |
| **State : Updated** |
| **Author : Bianca Cristea** |

#### Version history

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Version** | **Date** | **Author(s)** | **Changes** | **State** |
| 0.1 | 10.10.2024 | Bianca Cristea | Initial draft | Draft |
| 1 | 11.10.2024 | Bianca Cristea | Improved for version 2 | Stable version |
| 2 | 29.11.2024 | Bianca Cristea | Includes CI setup | Stable |
| 2.1 | 07.01.2025 | Bianca Cristea | Updated Diagrams | Update |

**Distribution**

|  |  |  |
| --- | --- | --- |
| **Version** | **Date** | **Receivers** |
| 1 | 11.10.2024 | People for People NGO |
| 2 | 29.11.2024 | People for People NGO |

Contents

[1. Introduction 4](#_Toc187218949)

[2. Architecture constraints 5](#_Toc187218950)

[3. C4 Model Diagrams 6](#_Toc187218951)

[3.1 C1 diagram 6](#_Toc187218952)

[3.2 C2 diagram 7](#_Toc187218953)

[3.3 C3 diagram 8](#_Toc187218954)

[3.4 C4 diagram 9](#_Toc187218955)

[4. CI setup 11](#_Toc187218956)

# Introduction

The purpose of this document is to outline the design and architecture of the *CareNest* software project. The project aims to define a software solution for the client, *People for People NGO*, whose wish is to diminish the neglect and ack of integration into society for people suffering from various affections, as well as elderly people. By detailing the design structure and principles, this document serves as a guide for the development process.

The following chapters contain the reasoning behind the decisions that were made for structuring this project, in order for the final product to prove reliability, scalability and quality. Therefore, this paper provides a comprehensive explanation of how the modern technologies and design strategies are combined so that the delivered system satisfies the client with a great experience.

# Architecture constraints

The system is build through a full-stack approach, following a set of constraints provided by the client, each of them having great advantages in the development of the project:

* React (frontend)

React provides a component based architecture for building a user interface in a single page application. This ensures a smooth experience for the users and high performance in communication with the backend

* Spring Boot + Java (backend)

Spring Boot, a Java-based framework, allows for a rapid development of the backend application and configuration of a RESTful API that communicates with the frontend

* MySQL(database layer)

The database is designed so that redundancy is reduces, providing an architecture that optimizes the performance of the application. Since the communication with the backend is done through JDBC, connection handling and queries management ensure smooth data transfers.

# C4 Model Diagrams

This chapter contains diagrams that describe the architecture of the application, starting with a high level view, and then diving into a detailed visual representation of its structure.

## C1 diagram

A diagram of a diagram

Description automatically generated

Entities in the diagram:

* **CareNest [Software System]**

This is the core system that is developed, which allows users to request the services of a caretaker, view announcements posted by medical managers, and manage their account information

* **E-mail System [External Softqare system]**

This is an external system which allows users to receive emails that automatically communicate specific information

* **Platform User [Person]**

A platform user is someone with a personal account who accesses the system to look for services provided by caretakers. They have the ability to view announcements and request caretaker services.

* **Manager [Person]**

A manager manages the system data related to announcements and employees(other managers). They have the ability to post announcements that can be viewed by platform users.

* **Caretaker [Person]**

A caretaker promotes their services on the platform, manages their account, and accepts or denies service requests from users.

## C2 diagram

A diagram of a diagram

Description automatically generated

Containers in this diagram:

* **Single Page Application (SPA) [Container: React & SpringBoot]**

This is the fronend layer of the application, with which the users have direct contact and perform various actions that are available for them. This uses API calls in order to communicate with thw web application

* **Web Application [Container: Java Spring Boot - REST]**

This is the layer that handles all the logic of the application and processes the information received form the frontend.

* + Communicates with the SPA through WebSockets in order to provide real-time updates and notifications
* **Database [Container: MySQL]**

This container represents the data storage layer of the system. It is a relational database that stores all the data received from the system and allowes it to read all the saved information

## C3 diagram

A diagram of a diagram

Description automatically generated

Key elements of the diagram:

* Single Page Application (React + Spring Boot)

The frontend (SPA) interacts with the backend by making API calls to the controllers. It’s the entry point for users accessing functionalities related to announcements, user accounts, services, and more.

* Controllers

The controllers in the diagram (e.g., AnnouncementController, UserController, RequestController) handle incoming API requests from the frontend

* Services

The service layer (e.g., AnnouncementServices, UserServices, SicknessServices) contains the business logic. Each service component works with its respective repository to perform operations like creating, updating, reading, or deleting data.

* Repositories

The repository layer (e.g., AnnouncementRepository, UserRepository, SicknessRepository) interfaces with the database. These components are responsible for reading from and writing to the database. Each repository handles a specific entity type

* Database

The database at the bottom of the diagram is the central data storage for all application entities

## C4 diagram

A diagram of a company

Description automatically generated

Key Classes and Their Relationships:

1. **User Class**
   * The user can optionally have an address
   * Each user can have one or more roles. These can be:
     + Patient
     + CareTaker
     + Manager
2. **Address Class**
   * A user can have at most one address
3. **CareTaker Class**
   * A caretaker may be specialized in one or more sicknesses
   * The availability of caretakers is modeled as an enum, and it can be part-timr or full-time
4. **Patient Class**
   * A patient suffers from one or more sicknesses
5. **Manager Class**
   * Managers can most more announcements
6. **Sickness Class**
   * Reprezents the sicknesses patients suffer from, and the ones that caretakers are specialised in
7. **Announcement Class**
   * each manager can post many announcements
8. **HireRequest Class**
   * This class represents the service request or "hire" made by a patient to a caretaker.
   * A hire request links patients to caretakers and is associated with the Sickness class, indicating that the service is related to the treatment of a specific sickness
   * Has a status that is represented as an enum (pending/accepted/denied)

# CI setup

The CI/CD pipeline for *CareNest* project automeates the process of building, testing and deploying the application to ensure reliability and efficiency.

A diagram of a diagram

Description automatically generated with medium confidence

Continuous Integration (CI)

1. Automated Build: After developers push changes to the main branch, the pipeline compiles the application using Gradle (./gradlew clean assemble).
2. Automated Testing: This stage of the pipeline runs the tests and makes sure they all pass (./gradlew test) to ensure functionality.
3. Code Quality Check: SonarQube analyses the code for quality, enforcing a quality gate. If the quality gate fails, the pipeline stops, and a report is generated.

Continuous Deployment (CD)

1. Updating docker containers: After all CI stages pass, the pipeline deletes any existing container and Docker image, builds a new Docker image, and pushes it to the Docker container.
2. Deployment: The new Docker image is deployed, and an updated container is run in the staging environment

Reports:

* If any stage fails, a failure report is generated and sent to the developers for debugging.
* If all stages succeed, the pipeline completes all its steps

A screenshot of a computer

Description automatically generated

This automated pipeline ensures an error-proof development, improves code quality, and supports rapid and reliable deployments.