Organ Donation Management System - Final Report

TEAM-7

IMT2021012 - Chokshi Yadav
IMT2021070 - Kevin Adesara
IMT2021081 - Gowtham Reddy
IMT2021102 - Anant Ojha

June 15, 2024

Contents

1 Introduction

The Organ Donation Management System (ODMS) developed by TEAM-7 aims to optimize organ donation processes. This document serves as the final report detailing the challenges faced during implementation in relation to the initial Software Requirements Specification (SRS).

2 Overall Description

The ODMS was designed to streamline donor registration, recipient matching, medical assessment tracking, communication management, and reporting. However, due to time constraints and technical complexities, the implemented system lacks several functionalities outlined in the initial SRS.

3 External Interface Requirements

While the system partially interfaces with various components such as user interfaces and basic software integrations, full compatibility with certain hardware and software components outlined in the SRS has not been achieved.

4 System Features

Key functionalities like donor registration, recipient matching, and patient prioritization were partially implemented. However, advanced reporting and compatibility assessments remain unimplemented due to technical constraints and time limitations.

5 Other Nonfunctional Requirements

The system partially meets nonfunctional requirements like basic security measures and usability aspects. However, due to the limited scope of implementation, aspects such as scalability and full compliance with data protection regulations are not addressed thoroughly.

6 Entities and Classes

The system comprises several entities and classes that encapsulate different aspects of our Organ Donation Management System (ODMS). Here's an overview of the key entities and classes:

6.1 Entities

6.1.1 Patient Entity

The **Patient Entity** represents individuals in need of organ transplants. It contains various attributes such as `patient_id`, `name`, `age`, `medical_condition`, `blood_type`, `antibody`, `needed_organ`, `organ_size`, `waited_time`, and `priority_score`.

6.1.2 Organ Entity

The **Organ Entity** represents available organs for donation. It includes attributes like `donor_id`, `organ`, `blood_type`, `organ_size`, and `antibody`.

6.2 Classes

6.2.1 Patient Class

The **Patient Class** encapsulates functionality related to patient management. It contains methods such as `__init__`, `__hash__`, `__eq__`, and other methods relevant to patient data management.

6.2.2 Organ Class

The **Organ Class** manages organ-related functionalities. It consists of methods like `__init__` and other methods pertinent to organ data handling.

6.3 Relations Between Classes

The **Patient Class** and **Organ Class** are interconnected within the system. Each patient instance (**Patient Entity**) may relate to an available organ instance (**Organ Entity**) based on various parameters such as compatibility in blood type, organ size, and antibody.

The relation between these classes involves matching patients in need of organs (**Patient Class**) with available organs (**Organ Class**) by assessing their attributes and compatibility. This relation is crucial for the system's functionality in efficiently pairing donors with recipients.

These entities and classes serve as the backbone of our system, allowing for organized data management and operations critical to the functionalities outlined in the Software Requirements Specification.

7 Testing

The Organ Donation Management System (ODMS) underwent thorough testing to validate its functionality and reliability. This involved two key methods:

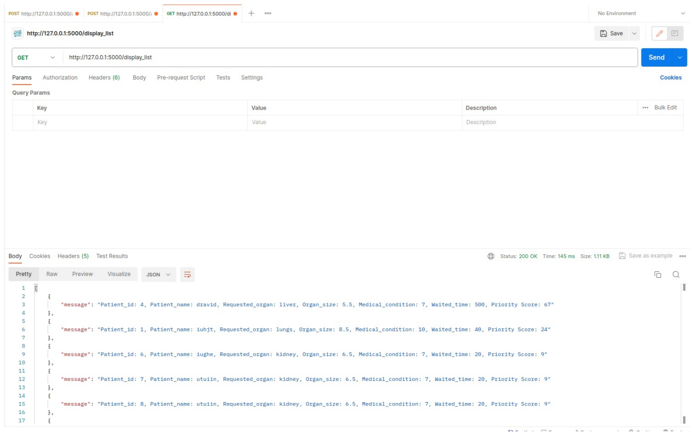
7.1 Unit Testing

Unit tests using Python's `unittest` framework were conducted to verify critical system functions and methods. Tests were designed for components like patient and donor retrieval, priority calculations, and matching criteria validation.

7.2 Functional Testing with Postman

Functional testing was performed using Postman, focusing on API endpoint testing (`/add_patient`, `/add_donor`). Various scenarios were tested to ensure proper HTTP request handling and expected responses, contributing to the system's reliability and adherence to requirements.

7.3 Postman Screenshot



http://127.0.0.1:5000/display_list

GET http://127.0.0.1:5000/display_list

Status: 200 OK Time: 145 ms Size: 1.11 KB

```
[
    {
        "message": "Patient_id: 4, Patient_name: dravid, Requested_organ: liver, Organ_size: 5.5, Medical_condition: 7, Waited_time: 500, Priority Score: 67"
    },
    {
        "message": "Patient_id: 1, Patient_name: iuhjt, Requested_organ: lungs, Organ_size: 8.5, Medical_condition: 10, Waited_time: 40, Priority Score: 24"
    },
    {
        "message": "Patient_id: 6, Patient_name: iughe, Requested_organ: kidney, Organ_size: 6.5, Medical_condition: 7, Waited_time: 20, Priority Score: 9"
    },
    {
        "message": "Patient_id: 7, Patient_name: utuiin, Requested_organ: kidney, Organ_size: 6.5, Medical_condition: 7, Waited_time: 20, Priority Score: 9"
    },
    {
        "message": "Patient_id: 8, Patient_name: utuiin, Requested_organ: kidney, Organ_size: 6.5, Medical_condition: 7, Waited_time: 20, Priority Score: 9"
    },
    {
```

Figure 1: Postman screenshot showcasing endpoint testing

These testing methods collectively ensured the robustness and reliability of the ODMS.

8 Backward Traceability of Deliverables

Throughout the project, we aimed to match our progress with what was outlined in the Software Requirements Specification (SRS). We started by breaking down the SRS into smaller tasks and milestones.

We managed to nail down some of the key requirements mentioned in the SRS, like allowing users to register as donors and basic recipient matching. However, some hurdles popped up that prevented us from checking off everything on that SRS list.

Our plan hit some unexpected snags, especially when dealing with connecting specific hardware parts and implementing complex compatibility checks. These challenges slowed down the process, and we couldn't pull off all the cool stuff we'd planned initially.

We regularly checked our progress against the SRS to see where we stood. We tweaked our plans as best we could to fit the unexpected issues, but time and resources didn't let us cover everything.

9 Proof of Concept

The implemented system represents a proof of concept demonstrating core functionalities but lacks the completeness outlined in the initial SRS. Not all features were implemented, citing technical complexities and resource constraints.

10 Conclusion

The ODMS project faced significant challenges in meeting the SRS requirements comprehensively. While the developed system demonstrates core functionalities, it falls short in several areas outlined in the initial specifications. Future iterations and enhancements are necessary to address these shortcomings and realize the system's full potential.

Github Link: `https://github.com/kadavakolla/se_project`