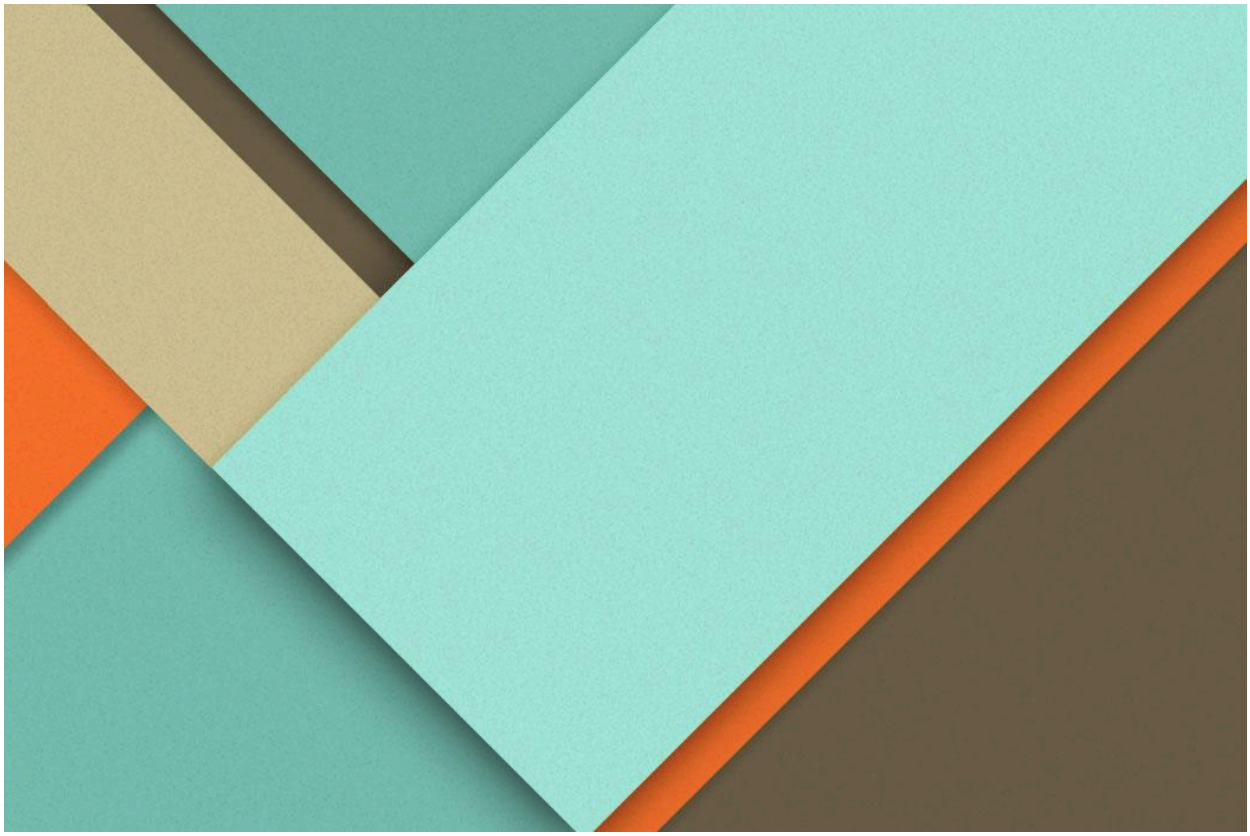

Project Description and Scope - Community Volunteer Matching Platform

20.10.2025

Dijar Hashai Fiona Kapllani Olta Gjevukaj Benita Graincaj

Project Overview

The 'Volneteer' platform is a scalable, modular volunteer-matching application designed to connect volunteers with organizations posting community service opportunities. The system manages users, opportunities, and applications, and employs smart matching algorithms to connect volunteers to suitable opportunities based on skills, interests, and availability. The architecture emphasizes scalability, memory safety, and extensibility.

Objectives

The main goal is to design and implement a maintainable and efficient volunteer matching application using sound software engineering principles. The application will showcase clean architecture, design patterns, and robust data management practices. Key objectives include:

- Implementing a modular architecture following SOLID principles.
- Ensuring memory stability and preventing crashes by decoupling storage and logic.
- Applying well-known design patterns for maintainability and scalability.
- Providing core functionalities such as user registration, opportunity creation, matching, and notifications.

Project Scope

The scope of this project includes the implementation of a core backend system built using JavaScript (Node.js). It will include models, repositories, and services to handle the matching process between volunteers and organizations. The frontend may be minimal, focusing mainly on backend logic and structure. The following functionalities are in scope:

- User Management (Registration, Login, Logout)
- Organization Management and Opportunity Posting
- Volunteer Profile Management
- Matching Service (Matching volunteers to opportunities)
- Notification Service (Email/In-App Notifications)
- Data Repository for persistence and memory-safe operations.

Design Patterns and Architectural Decisions

The project uses a range of design patterns to improve flexibility, scalability, and code reusability:
• Repository Pattern – Decouples data persistence from business logic, ensuring safe and efficient data handling.
• Dependency Injection – Improves testability and modularity by injecting dependencies rather than hardcoding them.
• Strategy Pattern – Used in MatchingService to allow multiple matching algorithms without modifying existing logic.
• Observer Pattern – Enables the NotificationService to handle asynchronous notifications.
• Factory and Adapter Patterns – Facilitate extensibility and integration with external systems such as email APIs.

Risk Assessment and Mitigation

Potential risks include memory overflow from unbounded in-memory data, concurrency conflicts, and dependency mismatches. Mitigation strategies include:
• Using a database-backed repository to prevent unbounded memory growth.
• Employing asynchronous processing for heavy operations such as matching.
• Implementing clear separation of concerns (SOLID principles) to isolate potential failures.
• Using pagination and data caps to limit processing loads.
• Running automated tests to detect regressions and memory leaks early.

Anticipated Design Risks & Mitigation

Risk	Impact	Mitigation
Memory overflow from unbounded in-memory repository	High	Implemented **Singleton** `DataRepository`; later migration to database possible without refactoring (DIP).
Controller–Service coupling reducing modularity	Medium	Use **Dependency Injection** in `main.js` to inject service instances.
Blocking operations during matching computation	Medium	Matching algorithm runs in linear time O(n); easily parallelizable (Strategy pattern).
Notification overload / tight coupling	Low	Observer-like architecture allows asynchronous notification handling.

Applied Design and Patterns

Pattern	Location	Purpose
Repository Pattern	`DataRepository.js`	Decouples data persistence from business logic.
Singleton Pattern	`DataRepository.js`	Guarantees single memory-safe instance.
Strategy Pattern	`MatchingService.js`	Enables future matching algorithms without rewriting code.
Observer Pattern	`NotificationService.js`	Decouples notification sending from core logic.

Dependency Injection	`main.js`	Simplifies testing and enhances flexibility.

Expected Deliverables

• Fully functional backend prototype demonstrating volunteer matching.
• Implementation of Repository, Matching, and Notification Services.
• Basic user interface or console interaction for testing.
• Documentation of design patterns, architecture, and design decisions.
• Unit and integration tests ensuring system stability.

Memory Safety and Reliability Considerations

The project architecture explicitly prevents memory leaks and crashes by:
• Avoiding global data storage and instead using repositories with cloning to prevent memory retention.
• Leveraging asynchronous non-blocking operations to maintain responsiveness.
• Using dependency injection to avoid circular dependencies.
• Employing garbage-collection-friendly data handling techniques.

Conclusion

The Volneteer platform represents a robust, scalable, and memory-safe volunteer matching system. By using structured design patterns, following software engineering best practices, and emphasizing separation of concerns, this project ensures stability, maintainability, and extensibility. The design ensures that the application can evolve over time without causing technical debt or performance degradation.