PRAC 1

Software Engineering

Alejandro Pérez Bueno

Nov 08, 2024

Table of Contents

Self-Responsibility Declaration

I understand that plagiarism, the use of AI or other generated content will imply that the delivered work will not be reviewed and it will be automatically assigned a grade of D. I certify that I have completed the PRAC1 individually and only with the help that the professors of this subject considered appropriate, according to the FAQs about plagiarism.

Question 1

Non-Functional Requirement 1

Category	Details
Requirement	"I believe that ensuring 99.9% availability is sufficient."
Description	The system should be available to users 99.9% of the time. This means there can be a total downtime of no more than 8 hours and 45 minutes per year.
Type	**Availability**
Stakeholder	**Salma** (Software Engineer responsible for the development of the application)

Non-Functional Requirement 2

Category	Details
Requirement	"Considering the project's characteristics, we will develop the application using a cross-platform framework, allowing us to run the application on both iOS and Android with the same code, thus speeding up development and simplifying maintenance."
Description	The application should be developed using a cross-platform framework so that the application can be deployed on both iOS and Android using the same codebase. This will reduce development time and maintenance efforts.
Type	**Portability/Compatibility**
Stakeholder	**Salma** (Software Engineer responsible for the development of the application)

Non-Functional Requirement 3

Category	Details
Requirement	"the application must comply with the WCAG 2.2 standard"
Description	The system should be accessible to users with disabilities. The Web Content Accessibility Guidelines (WCAG) 2.2 define how to make web content more accessible to people with disabilities.
Type	**Accessibility**
Stakeholder	**Rasheeda** (Business Angel and financial manager of the project)

Question 2

Here are the user stories:

- As a **resident**, I want to be able to **view a calendar of events** so that I can **stay informed about upcoming community meetings, inspections, and other important dates.**

- As a **resident**, I want to be able to **cast my vote in community polls** so that I can **have a say in decisions that affect my community**.
- As a **president**, I want to be able to **create and manage community polls** so that I can **gather feedback and make decisions based on the input of residents**.
- As a **manager**, I want to be able to **manage community fees and track payment status** so that I can **ensure that the community is financially stable.**

Question 3

Conflicting and Dependent Requirements

a) Conflicting Requirements:

- **Stakeholders:** Rasheeda (Business Angel) and Anonymous Residents
- **Conflicting Requirements:** Adding photos to advertisements vs. Setting advertisement expiration dates
  - **Rasheeda** wants the application to be accessible to people with disabilities and, therefore, requests that the application comply with the WCAG 2.2 standard. WCAG 2.2 includes success criterion 2.2.1: Timing Adjustable, which requires that users be able to disable, adjust, or extend time limits. [outside source] This means that if residents are allowed to set expiration dates for their advertisements, they must also be able to disable those expiration dates.
  - **Anonymous residents** might want to include photos in their advertisements. However, if the system allows photos, it would become more difficult to automatically moderate the content of the advertisements to ensure they do not violate the WCAG guidelines. For instance, a photo could contain flashing images or text that is illegible to people with certain visual impairments. As a result, setting expiration dates on advertisements with photos could conflict with the accessibility requirements proposed by Rasheeda.

b) Dependent Requirements:

- **Stakeholders:** Anonymous residents and Julián (professional property manager)
- **Requirements:** Registering as a resident and managing community fees.
  - **Anonymous residents** should be able to register in the application to become part of the community.
  - **Julián** suggests a feature for managing community fee payments.
  - The ability to manage community fees is dependent on users being registered residents. Only registered residents should be able to view and pay their community fees. Therefore, the "manage community fees" requirement relies on the "register as a resident" requirement.

Question 4

Here are the user roles that exist in the application based on the interviews:

- **President**
- **Resident**
- **Manager**
- **Project Executives**

Question 5

Let's describe the use case "Register a Community Fee in the Application" based on the provided interviews. The Property Manager, Julián, outlines how this process works when an association is managed by a professional property manager.

Use Case: Register a Community Fee

- **Use Case Identifier:** UC001
- **Actor Principal:** Property Manager
- **Supporting Actors:** President, Resident
- **Level:** User Goal
- **Scope:** Organization
- **Main Stage of Success:**

  1. The Property Manager initiates the fee registration process.
  2. The Property Manager enters the fee amount and payment frequency.
     - If the fee is not the same for everyone, the amount is entered individually for each resident.
  3. The Property Manager submits the fee information for validation.
  4. The President reviews the fee amounts and frequency.
  5. The President approves the fee.
  6. The system sends a notification to each resident, prompting them to select a payment method.
  7. Residents select their preferred payment method.
     - If direct debit is selected, residents enter their bank account information.
     - If Apple/Google Pay is selected, residents choose their preferred card.
  8. The Property Manager validates that all residents have entered their payment information correctly.
  9. The fee registration is finalized.

- **Alternative Scenarios:**

  1. **President Rejects Fee:** If the President finds a discrepancy in the fee amount or frequency during their review, they reject the fee and provide a reason for the rejection. The Property Manager must then modify the fee information based on the President's feedback and resubmit it for approval.

2. **Resident Enters Incorrect Bank Information:** If a resident selects direct debit and enters an incorrect bank account number, the system displays an error message due to invalid check digits. The resident must correct the bank account information before proceeding.
3. **Resident Fails to Enter Payment Information:** If a resident does not enter their payment information within a specified timeframe, the Property Manager is alerted. The Property Manager then sends a message to the affected resident(s), reminding them to provide the required payment details.

This use case description provides a comprehensive overview of the fee registration process from an organizational level. However, it's important to note that the sources do not explicitly state how the President is notified of the submitted fee or how the Property Manager is alerted if residents do not enter payment information. These details would need to be further defined in a real-world scenario.

Question 6

Here are some use cases based on the homeowner's association application interviews:

a) User-level use cases where the President is the main actor:

- **Create a calendar event:** The President schedules events in the community calendar, including meetings, inspections, and other relevant dates.
- **Initiate a community vote:** The president starts a voting process for decisions that affect the community, allowing residents to participate and have a say.

b) User-level use cases where the Neighbor is the main actor:

- **View the community calendar:** The Neighbor accesses the calendar to stay informed about upcoming events and important dates within the association.
- **Vote in a community poll:** The Neighbor participates in active polls to contribute to decision-making processes within the association.

c) Task-level use cases derived from selected user-level use cases:

For the President:

- **User-level use case:** Initiate a community vote
  - **Task-level use case 1: Define voting parameters.** The President sets the start and end dates of the vote, specifies the eligible voters, and sets up the voting options.
  - **Task-level use case 2: Publish the vote.** The President makes the vote accessible to eligible residents in the application.

For the Neighbor:

- **User-level use case:** View the community calendar
  - **Task-level use case 1: Filter calendar events.** The Neighbor chooses to view events based on categories like meetings, maintenance, or social gatherings.

- **Task-level use case 2: Set event reminders.** The Neighbor subscribes to receive notifications or reminders for specific events.

These task-level use cases break down the broader user-level use cases into more specific and manageable actions, providing a more detailed view of how users will interact with the application.