PRAC 2

Software Engineering

Alejandro Pérez Bueno

Dec 19, 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 PRAC2 individually and only with the help that the professors of this subject considered appropriate, according to the FAQs about plagiarism.

Question 1

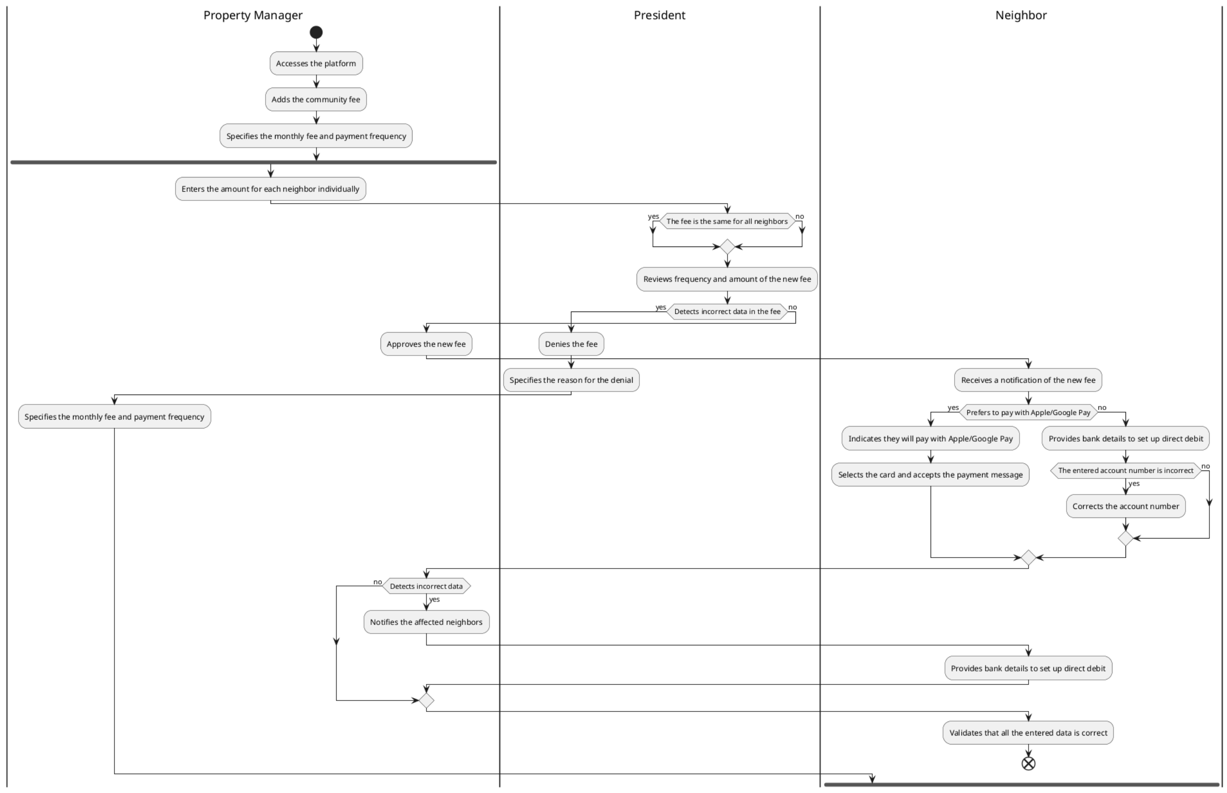
Property Manager
President
Neighbor
Accesses the platform
Adds the community fee
Specifies the monthly fee and payment frequency
Enters the amount for each neighbor individually
yes
The fee is the same for all neighbors
no
Reviews frequency and amount of the new fee
yes
Detects incorrect data in the fee
no
Approves the new fee
Denies the fee
Specifies the reason for the denial
Specifies the monthly fee and payment frequency
Receives a notification of the new fee
yes
Prefers to pay with Apple/Google Pay
no
Indicates they will pay with Apple/Google Pay
Provides bank details to set up direct debit
Selects the card and accepts the payment message
The entered account number is incorrect
no
yes
Corrects the account number
no
Detects incorrect data
yes
Notifies the affected neighbors
Provides bank details to set up direct debit
Validates that all the entered data is correct

Figure 1: Activity Diagram

Question 2

Additional Use Cases from Interviews:

- **Neighbour:** *Submit a Proposal* - This use case allows a Neighbour to submit a proposal for discussion or voting to the community. This functionality is mentioned by Juanma as a way to handle issues too complex for the online forum.
- **President:** *Change Community President* - Ariadne specifically mentions the need to change the community president in the application, especially for communities with rotating presidents.
- **Property Manager:** *Generate Community Financial Reports* - Juanma highlights the need for professional property managers to have access to financial data and reports. This use case addresses that need.
- **Director:** There is no mention of a "Director" role in the provided interview transcripts. It's impossible to define a relevant use case without understanding this role's responsibilities.
- **Anonymous User:** *View Public Community Information* - While Juanma emphasizes data protection, it is plausible that some community information, such as meeting dates or public announcements,

could be accessible to anonymous users to promote transparency.

UML Use Case Diagram:

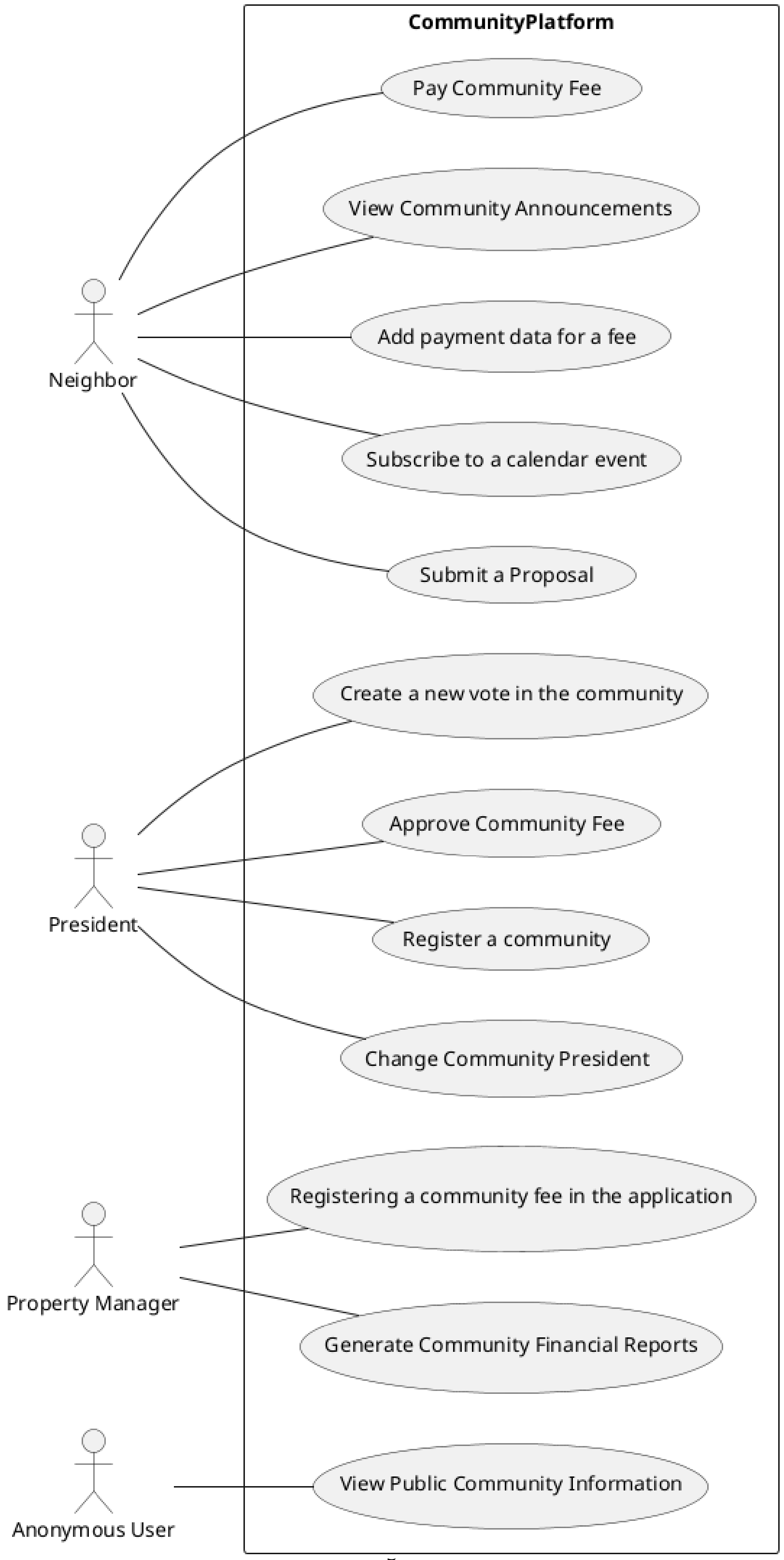
CommunityPlatform
Pay Community Fee
View Community Announcements
Add payment data for a fee
Subscribe to a calendar event
Submit a Proposal
Create a new vote in the community
Approve Community Fee
Register a community
Change Community President
Registering a community fee in the application
Generate Community Financial Reports
View Public Community Information
Neighbor
President
Property Manager
Anonymous User

Figure 2: Use Case Diagram

Question 3

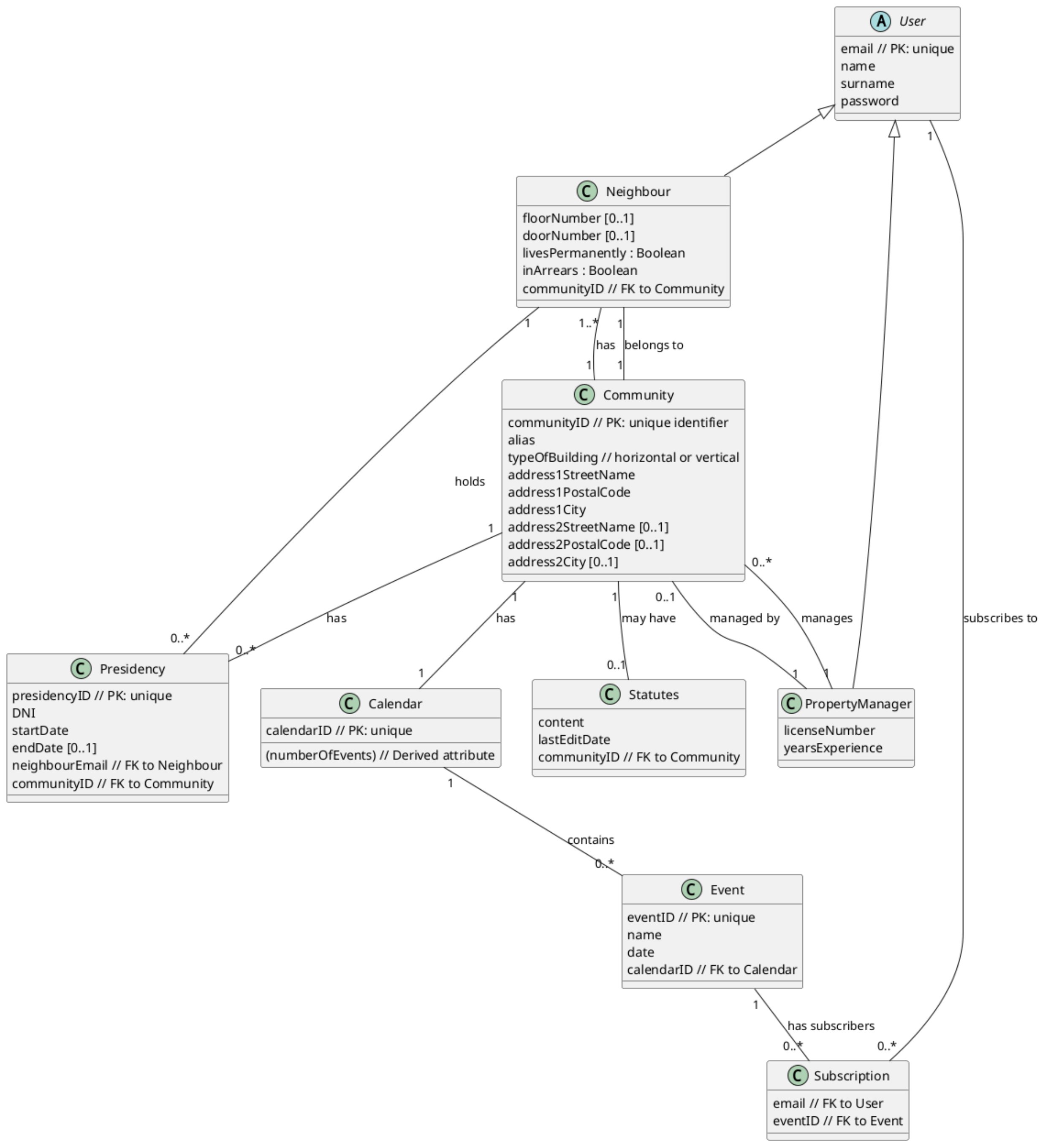

Figure 3: Class Diagram

Keys:

- **User**: `email` (Primary Key, unique across all users)
- **Neighbour**: Inherits `email` from `User`
- **PropertyManager**: Inherits `email` from `User`
- **Community**: `communityID` (Primary Key, unique identifier)
- **Presidency**: `presidencyID` (Primary Key, unique for each presidency)
- **Calendar**: `calendarID` (Primary Key, unique for each calendar)
- **Event**: `eventID` (Primary Key, unique for each event)
- **Subscription**: Composite key of `email` and `eventID` (each subscription is unique per user-event pair)
- **Statutes**: Associated with `communityID` (one set of statutes per community)

Integrity Constraints:

1. **Unique Email**: The `email` attribute must be unique for each `User` in the system.
2. **Neighbour Constraints**:
   - A `Neighbour` must belong to one and only one `Community` (`Neighbour.communityID` is a mandatory foreign key to `Community.communityID`).
   - `floorNumber` and `doorNumber` are optional attributes.
   - `livesPermanently` is a boolean indicating if the neighbour permanently resides in the property.
   - `inArrears` is a boolean indicating if the neighbour is behind on payments.
3. **President Constraints**:
   - A `Neighbour` must have `livesPermanently` set to `true` to become a `President`.
   - The application must prevent a neighbour from becoming president if they do not live permanently in the building.
   - `Presidency` records link `Neighbour` and `Community` with `startDate` and `endDate` to store the history of presidents.
   - At any given time, there can be at most one active presidency (`endDate` is `null`) per `Community`.

4. **Community Constraints**:
   - **Address**:
     - At least one address (`address1StreetName`, `address1PostalCode`, `address1City`) must be provided.
     - If a second address is provided (`address2StreetName`, etc.), it must be different from the first address.
   - `typeOfBuilding` must be either "horizontal" or "vertical".
   - A `Community` may optionally have a `PropertyManager`.
5. **PropertyManager Constraints**:
   - A `PropertyManager` can manage multiple `Communities`.
   - If a `Community` has a `PropertyManager`, it must be linked appropriately.
6. **Calendar and Events Constraints**:
   - Each `Community` has one `Calendar`.
   - The `Calendar` contains multiple `Events`.
   - All users (`Neighbours` and `PropertyManagers`) linked to a `Community` can subscribe to `Events`.
   - The number of events in a `Calendar` is a derived attribute calculated from the count of `Events` associated with it.
7. **Subscription Constraints**:
   - A `User` can subscribe to multiple `Events`.
   - An `Event` can have multiple `Users` subscribed to it.
   - The `Subscription` relationship ensures that users receive notifications about events they are interested in.
8. **Statutes Constraints**:
   - A `Community` may optionally have `Statutes`.
   - If `Statutes` are present, they must include `content` and `lastEditDate`.
   - Only one set of `Statutes` per `Community`.

9. **General Constraints**:
   - When adding an address, mandatory fields are `streetName`, `postalCode`, and `city`.
   - The `endDate` in `Presidency` is optional and is set when a presidency ends.
   - All data must comply with data integrity and validation rules (e.g., dates should be valid, boolean values properly set).

Derived Information:

- **Number of Events in a Calendar**: The `numberOfEvents` attribute in `Calendar` is derived by counting the total number of `Events` linked to that `Calendar`.

Question 4

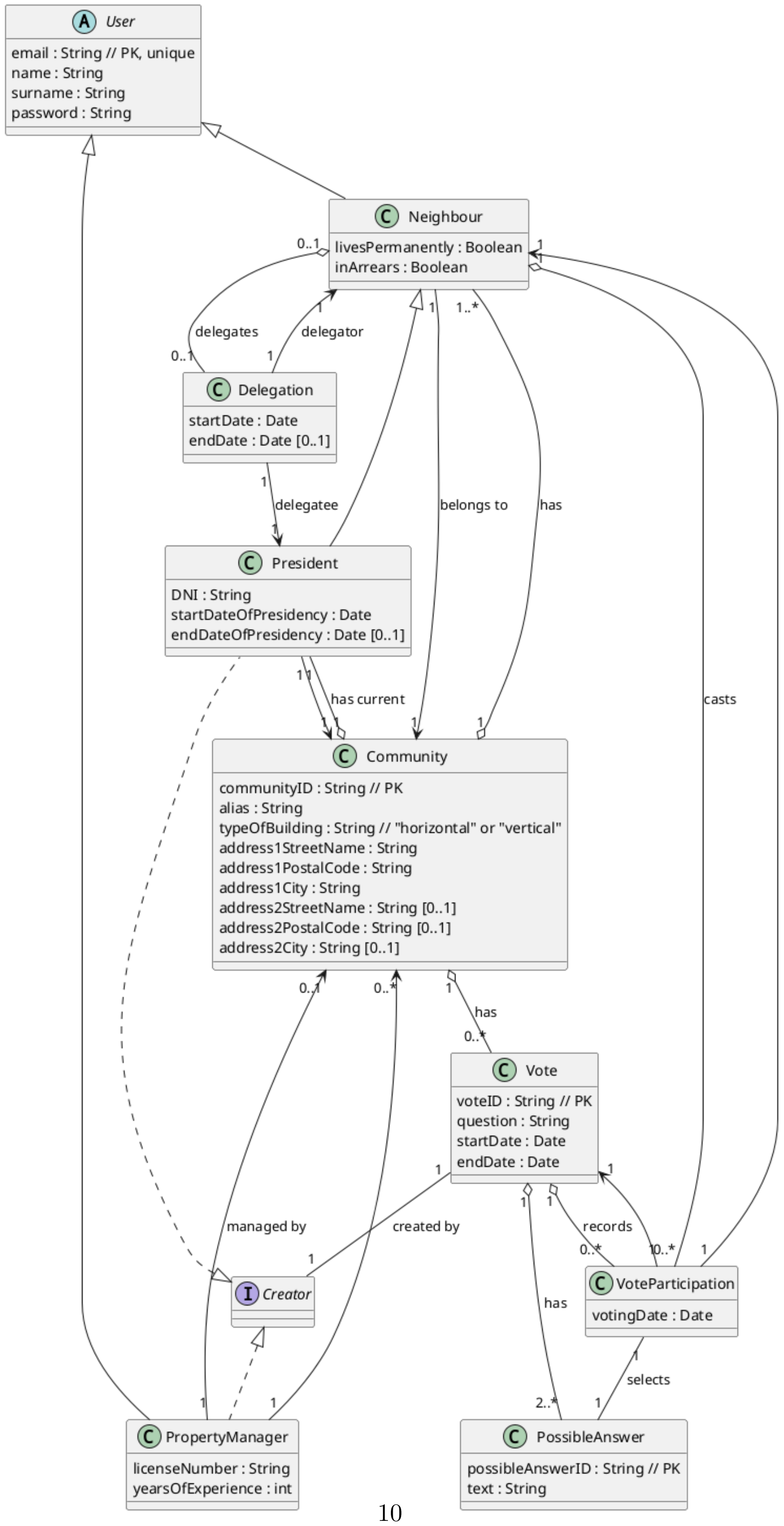

Figure 4: Class Diagram

Keys:

- **User**: `email` (Primary Key)
- **Neighbour**: Inherits `email` from **User**
- **President**: Inherits `email` from **Neighbour**
- **PropertyManager**: Inherits `email` from **User**
- **Community**: `communityID` (Primary Key)
- **Vote**: `voteID` (Primary Key)
- **PossibleAnswer**: Composite Primary Key (`possibleAnswerID`, `voteID`)
- **VoteParticipation**: Composite Primary Key (`voteID`, `neighbourEmail`)
- **Delegation**: `neighbourEmail` (Primary Key)

Integrity Constraints:

1. **Vote Creation:**
   - A `Vote` must be associated with one `Community` (`Vote` → `Community`).
   - A `Vote` is created by either a `President` or a `PropertyManager` (`Creator` interface).
   - The creator must be associated with the same `Community` where the `Vote` is being held.
2. **Possible Answers:**
   - Each `Vote` must have at least **two** `PossibleAnswers`.
   - Each `PossibleAnswer` is associated with one `Vote`.
3. **Voting Eligibility:**
   - Only `Neighbours` who are **not in arrears** (`inArrears = False`) can participate in votes.
   - `President` can participate in votes if they are not in arrears (since they are also a `Neighbour`).
   - Only `Neighbours` belonging to the specific `Community` can participate in its `Votes`.
4. **VoteParticipation:**
   - Records the `votingDate`, which must be within the `Vote`'s `startDate` and `endDate`.
   - Each `VoteParticipation` links a `Neighbour` to a `Vote`, recording their selected `PossibleAnswer`.

- A `Neighbour` can participate in a `Vote` only once; enforced by the composite primary key (`voteID`, `neighbourEmail`).

5. **Delegation:**
    - Only `Neighbours` who **do not reside permanently** in their property (`livesPermanently = False`) can delegate their votes.
    - Delegation is from a `Neighbour` to the `President` of their `Community`.
    - Delegation is valid from `startDate` to an optional `endDate`. If `endDate` is not set, the delegation is considered ongoing.
    - During the delegation period, the `Neighbour` cannot cast votes themselves; the `President` votes on their behalf.
6. **Derived Information:**
    - **Number of Votes** for a `Vote` is calculated by counting the associated `VoteParticipation` entries.
    - **Most Voted Answer(s)** are determined by tallying the `PossibleAnswer` selections in `VoteParticipation` records. In case of ties, multiple answers may be the most voted.
7. **Constraints on Dates:**
    - `Vote.startDate` and `Vote.endDate` must be valid dates with `startDate` before `endDate`.
    - `VoteParticipation.votingDate` must be between the `Vote`'s `startDate` and `endDate`.
8. **Constraints on Delegation:**
    - A `Neighbour` cannot delegate their vote if they reside permanently (`livesPermanently = True`).
    - A `Neighbour` can only delegate their vote to the `President` of their own `Community`.
    - The `Delegation` must not have overlapping delegation periods for the same `Neighbour`.
9. **Constraints on Voting:**
    - The system must prevent a `Neighbour` from voting if they have an active delegation.
    - A `Neighbour` cannot vote if they are in arrears.
10. **Multiplicity Constraints:**
    - Each `Vote` must have at least **two** `PossibleAnswers` (Multiplicity `2..*`).
    - A `Neighbour` can have at most **one** active `Delegation`.
11. **Data Integrity:**
    - All foreign keys must reference existing records (e.g., `voteID` in `VoteParticipation` must exist in `Vote`).

- The `President` must be associated with the same `Community` as the `Neighbour` in `Delegation`.
- The selected `PossibleAnswer` in `VoteParticipation` must be one of the possible answers for that `Vote`.

Derived Information Calculations:

- **Number of Votes for a Vote:**
  - Count the number of `VoteParticipation` records associated with that `Vote`.
- **Most Voted Answer(s):**
  - For each `PossibleAnswer`, count the number of times it was selected in `VoteParticipation`.
  - The answer(s) with the highest count are the most voted. In the case of equal counts, multiple answers can be the most voted.

Changes from Previous Classes:

- **Classes Reused:** `User`, `Neighbour`, `President`, `PropertyManager`, and `Community` are included from previous exercises.
- **No Attribute Changes:** The attributes from the previous classes remain unchanged for this diagram.
- **Additional Associations:** New associations have been added to connect these classes to the voting system entities (`Vote`, `PossibleAnswer`, `VoteParticipation`, `Delegation`).

Question 5

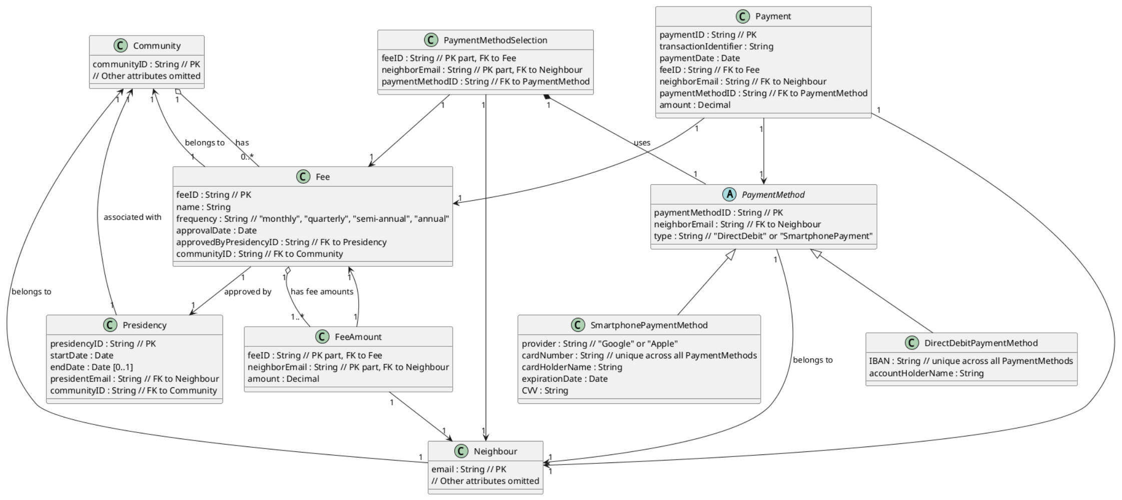
Community
communityID : String // PK
// Other attributes omitted
PaymentMethodSelection
feeID : String // PK part, FK to Fee
neighborEmail : String // PK part, FK to Neighbour
paymentMethodID : String // FK to PaymentMethod
Payment
paymentID : String // PK
transactionIdentifier : String
paymentDate : Date
feeID : String // FK to Fee
neighborEmail : String // FK to Neighbour
paymentMethodID : String // FK to PaymentMethod
amount : Decimal
belongs to
has
0..*
uses
associated with
Fee
feeID : String // PK
name : String
frequency : String // "monthly", "quarterly", "semi-annual", "annual"
approvalDate : Date
approvedByPresidencyID : String // FK to Presidency
communityID : String // FK to Community
PaymentMethod
paymentMethodID : String // PK
neighborEmail : String // FK to Neighbour
type : String // "DirectDebit" or "SmartphonePayment"
belongs to
approved by
has fee amounts
1..*
Presidency
presidencyID : String // PK
startDate : Date
endDate : Date [0..1]
presidentEmail : String // FK to Neighbour
communityID : String // FK to Community
FeeAmount
feeID : String // PK part, FK to Fee
neighborEmail : String // PK part, FK to Neighbour
amount : Decimal
SmartphonePaymentMethod
provider : String // "Google" or "Apple"
cardNumber : String // unique across all PaymentMethods
cardHolderName : String
expirationDate : Date
CVV : String
belongs to
DirectDebitPaymentMethod
IBAN : String // unique across all PaymentMethods
accountHolderName : String
Neighbour
email : String // PK
// Other attributes omitted

Figure 5: Class Diagram

Keys of Domain Classes:

1. **Community**
   - `communityID` (Primary Key)
2. **Neighbour**
   - `email` (Primary Key)
3. **Presidency**
   - `presidencyID` (Primary Key)

   - `presidentEmail` (Foreign Key to Neighbour)

   - `communityID` (Foreign Key to Community)
4. **Fee**
   - `feeID` (Primary Key)

   - `communityID` (Foreign Key to Community)

   - `approvedByPresidencyID` (Foreign Key to Presidency)
5. **FeeAmount**
   - Composite Key: (`feeID`, `neighborEmail`) (Foreign Keys to Fee and Neighbour)

   - Attributes:
     – `amount`
6. **PaymentMethod** (Abstract Class)

- `paymentMethodID` (Primary Key)
- `neighborEmail` (Foreign Key to Neighbour)
- `type` ("DirectDebit" or "SmartphonePayment")

7. **DirectDebitPaymentMethod** (Subclass of PaymentMethod)
    - Inherits `paymentMethodID`
    - Attributes:
        - `IBAN` (Unique across all PaymentMethods)
        - `accountHolderName`
8. **SmartphonePaymentMethod** (Subclass of PaymentMethod)
    - Inherits `paymentMethodID`
    - Attributes:
        - `provider` ("Google" or "Apple")
        - `cardNumber` (Unique across all PaymentMethods)
        - `cardHolderName`
        - `expirationDate`
        - `CVV`
9. **PaymentMethodSelection**
    - Composite Key: (`feeID`, `neighborEmail`)
    - Attributes:
        - `paymentMethodID` (Foreign Key to PaymentMethod)
10. **Payment**
    - `paymentID` (Primary Key)
    - Attributes:
        - `transactionIdentifier`
        - `paymentDate`
        - `feeID` (Foreign Key to Fee)
        - `neighborEmail` (Foreign Key to Neighbour)

  - `paymentMethodID` (Foreign Key to PaymentMethod)
  - `amount`

Integrity Constraints and Business Rules:

1. **Fee Constraints:**
   - `frequency` must be one of "monthly", "quarterly", "semi-annual", or "annual".
   - A **Fee** must be associated with one **Community**.
   - The `approvalDate` must be set when the **Fee** is approved.
   - `approvedByPresidencyID` must reference a **Presidency** whose term includes the `approvalDate`.
   - The **Presidency** must be associated with the same **Community** as the **Fee**.
2. **FeeAmount Constraints:**
   - Each **FeeAmount** links a **Neighbour** to a **Fee** and specifies the `amount` they need to pay.
   - The `amount` must be greater than zero.
   - One **FeeAmount** per **Neighbour** per **Fee**.
3. **PaymentMethod Constraints:**
   - **PaymentMethod** is associated with one **Neighbour**.
   - The `IBAN` in **DirectDebitPaymentMethod** must be unique across all payment methods.
   - The `cardNumber` in **SmartphonePaymentMethod** must be unique across all payment methods.
   - For **DirectDebitPaymentMethod**, `IBAN` and `accountHolderName` cannot be null.
   - For **SmartphonePaymentMethod**, `provider`, `cardNumber`, `cardHolderName`, `expirationDate`, and `CVV` cannot be null.
   - The `provider` must be either "Google" or "Apple".
4. **PaymentMethodSelection Constraints:**
   - Each **PaymentMethodSelection** specifies the `paymentMethodID` for a **Neighbour** for a specific **Fee**.
   - The `paymentMethodID` must belong to the same **Neighbour** as specified in `neighborEmail`.

- One **PaymentMethodSelection** per **Neighbour** per **Fee**.

5. **Payment Constraints:**
   - Each **Payment** must reference a valid **Fee**, **Neighbour**, and **PaymentMethod**.
   - The `paymentMethodID` must be the one selected in **PaymentMethodSelection** for the corresponding **Fee** and **Neighbour**.
   - The `paymentDate` must be after the `approvalDate` of the **Fee**.
   - The `amount` in **Payment** should match the `amount` specified in **FeeAmount** for that **Fee** and **Neighbour**.
6. **Approval Process Constraints:**
   - A **Fee** must be reviewed and approved by the **President** before any payments can be processed.
   - The **President** approving the **Fee** must be in office during the `approvalDate`.
7. **Uniqueness Constraints:**
   - `IBAN` and `cardNumber` must be unique across all payment methods in the system.
   - **PaymentMethod** IDs are unique.
   - **Payment** IDs are unique.
8. **Association Constraints:**
   - **Neighbour** belongs to one **Community**.
   - **Presidency** is associated with one **Community** and one **President** (**Neighbour**).
   - **FeeAmount** must link **Neighbours** that belong to the same **Community** as the **Fee**.
9. **Data Integrity Constraints:**
   - All date attributes must be valid dates.
   - Non-nullable attributes must have valid data.
   - Foreign keys must reference existing records.
10. **Sequence of Operations:**
    - A **Fee** is created and then must be approved by the **President**.
    - After approval, each **Neighbour** must select a **PaymentMethod** for the **Fee**.
    - Payments can then be made using the selected **PaymentMethod**.

Derived Information:

- **Payment Amount in Payment**:
  - The `amount` in **Payment** can be derived from the `amount` specified in **FeeAmount** for the corresponding **Fee** and **Neighbour**.

  - Alternatively, it can be stored explicitly to record partial payments or adjustments.
- **Approval by President:**
  - Although the **Fee** records the `approvedByPresidencyID`, the actual **President** (the **Neighbour**) who approved the **Fee** can be derived from the **Presidency** referenced.