OAKhoury Trees Presentation

Jordan Pinnick, Devan Kumar, Matthew Shi

Project Motivation

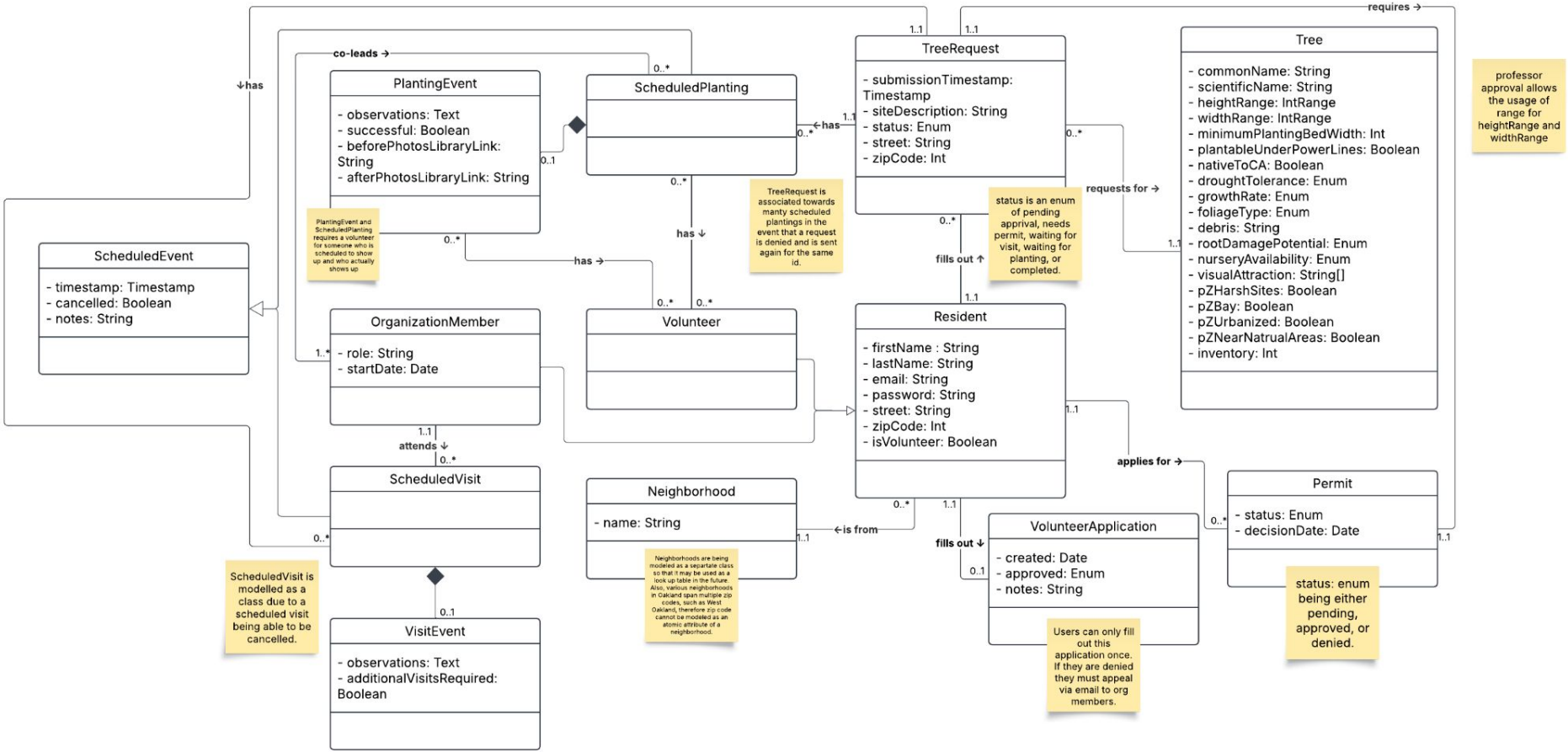
Residents who want to have a tree planted within their neighborhoods have issues requesting as the only way possible is through a slow google form where data may be mixed up and be inconsistent. Volunteers also have an unclear way to stay informed about the events occurring within their community.

This project modernizes the process for residents and volunteers through the creation of a centralized platform, allowing for people to request trees, track progress, and participate towards the planting events. The system allows for a more efficient and effective system where data is well organized and provides a clear view on how people will be able to improve their community.

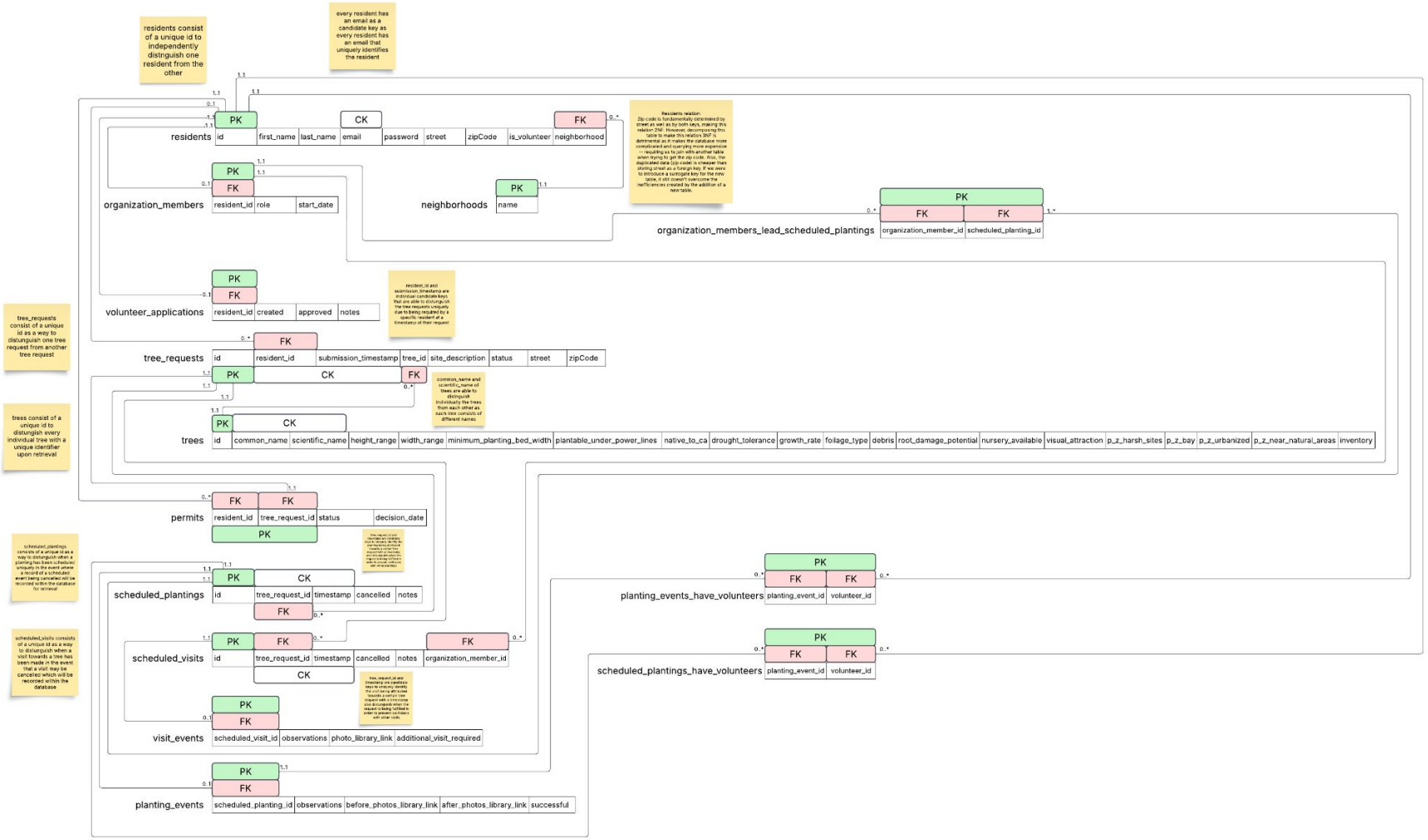
System Description

The system is able to allow residents to register as a user. Users will be able to request for a tree and are also able to send requests to become volunteers. Administrators who are members of the organization are able to approve of these requests for a tree and volunteer. Administrators are also able to view the details for a tree request as well as manage the planting events. This allows for them to keep track of all the data concisely rather than having a google form with disorganized data.

https://lucid.app/lucidchart/be7d4afb-c425-4e1d-869c-bdeaad5d1f40/edit?invitationId=inv_bd4086c9-7db2-439b-920c-62f202529b7d&page=E283~sCDk0F9#



https://lucid.app/lucidchart/be7d4afb-c425-4e1d-869c-bdeaad5d1f40/edit?invitationId=inv_bd4086c9-7db2-439b-920c-62f202529b7d&page=.~P35cu.5UbE#



```sql
-- Creates tables in Postgres v.17.4

CREATE TABLE neighborhoods
(
    name VARCHAR(100) PRIMARY KEY
);

CREATE TABLE residents
(
    id           SERIAL PRIMARY KEY,
    first_name   VARCHAR(50) NOT NULL,
    last_name    VARCHAR(50) NOT NULL,
    email        VARCHAR(100) UNIQUE NOT NULL,
    password     VARCHAR(128), -- Argon-2 max hash length
    street       VARCHAR(50),
    zip_code     CHAR(5),
    is_volunteer BOOLEAN,
    neighborhood VARCHAR(100) REFERENCES neighborhoods (name) ON DELETE NO ACTION ON UPDATE CASCADE NOT NULL
);

CREATE TABLE organization_members
(
    resident_id INTEGER PRIMARY KEY REFERENCES residents (id) ON DELETE CASCADE ON UPDATE CASCADE NOT NULL,
    role        VARCHAR(50) NOT NULL,
    start_date  DATE
);

CREATE TABLE volunteer_applications
(
    resident_id INTEGER PRIMARY KEY REFERENCES residents (id) ON DELETE CASCADE ON UPDATE CASCADE NOT NULL,
    created     DATE NOT NULL,
    approved    BOOLEAN,
    notes       TEXT
);
```

```sql
CREATE TYPE tolerance AS ENUM ('moderate', 'high', 'very high');
CREATE TYPE rate AS ENUM ('slow', 'moderate', 'fast', 'very fast');
CREATE TYPE foliage AS ENUM ('deciduous', 'drought-deciduous', 'evergreen', 'semi-evergreen', 'late-deciduous');
CREATE TYPE root_damage AS ENUM ('low', 'moderate', 'high');
CREATE TYPE nursery_availability AS ENUM ('low', 'moderate', 'high');
CREATE TABLE trees
(
    id                          SERIAL PRIMARY KEY NOT NULL,
    common_name                 VARCHAR(100) UNIQUE,
    scientific_name             VARCHAR(100) UNIQUE NOT NULL,
    height_range                int4range,
    width_range                 int4range,
    minimum_planting_bed_width  INTEGER,
    plantable_under_power_lines BOOLEAN,
    native_to_ca                BOOLEAN,
    drought_tolerance           tolerance,
    growth_rate                 rate,
    foliage_type                foliage,
    debris                      VARCHAR(50),
    root_damage_potential       root_damage,
    nursery_availability        nursery_availability,
    visual_attraction           VARCHAR(50)[],
    pzharshsites                BOOLEAN,
    pzbay                       BOOLEAN,
    pzurbanized                 BOOLEAN,
    pznearnaturalareas          BOOLEAN,
    inventory                   INTEGER CHECK (inventory >= 0)
);

CREATE TABLE tree_requests
(
    id                  SERIAL PRIMARY KEY NOT NULL,
    resident_id         INTEGER REFERENCES residents (id) ON DELETE NO ACTION ON UPDATE CASCADE NOT NULL,
    submission_timestamp TIMESTAMP NOT NULL,
    tree_id             INTEGER REFERENCES trees (id) ON DELETE CASCADE ON UPDATE CASCADE NOT NULL,
    site_description    TEXT,
    approved            BOOLEAN,
    UNIQUE (resident_id, submission_timestamp)
);
```

```sql
CREATE TYPE status AS ENUM ('pending', 'approved', 'denied');
CREATE TABLE permits
(
    resident_id     INTEGER REFERENCES residents (id) ON DELETE CASCADE ON UPDATE NO ACTION,
    tree_request_id INTEGER REFERENCES tree_requests (id) ON DELETE NO ACTION ON UPDATE CASCADE,
    status          status NOT NULL,
    decision_date   DATE,
    PRIMARY KEY (resident_id, tree_request_id)
);

-- Abstract table to hold scheduled_events
CREATE TABLE scheduled_events
(
    event_id        SERIAL PRIMARY KEY NOT NULL,
    tree_request_id INTEGER REFERENCES tree_requests (id) ON DELETE CASCADE ON UPDATE NO ACTION NOT NULL,
    event_timestamp TIMESTAMP NOT NULL,
    cancelled       BOOLEAN,
    notes           TEXT
);

CREATE TABLE scheduled_plantings
(
    -- inherits all from scheduled_events
    PRIMARY KEY (event_id)
) INHERITS (scheduled_events);


CREATE TABLE scheduled_visits
(
    organization_member_id INTEGER REFERENCES organization_members (resident_id) ON DELETE CASCADE ON UPDATE CASCADE NOT NULL,
    PRIMARY KEY (event_id)
) INHERITS (scheduled_events);

CREATE TABLE visit_events
(
    scheduled_visit_id       INTEGER PRIMARY KEY REFERENCES scheduled_visits (event_id) ON DELETE CASCADE ON UPDATE CASCADE,
    observations             TEXT,
    photo_library_link       VARCHAR(100),
    additional_visit_required BOOLEAN
);
```

```sql
CREATE TABLE planting_events
(
    scheduled_planting_id      INTEGER PRIMARY KEY REFERENCES scheduled_plantings (event_id) ON DELETE CASCADE ON UPDATE CASCADE,
    observations               TEXT,
    before_photos_library_link VARCHAR(100),
    after_photos_library_link  VARCHAR(100),
    successful                 BOOLEAN NOT NULL
);

-- junction tables

CREATE TABLE organization_members_lead_scheduled_plantings
(
    organization_member_id INTEGER REFERENCES organization_members (resident_id) ON DELETE CASCADE ON UPDATE CASCADE,
    scheduled_planting_id  INTEGER REFERENCES scheduled_plantings (event_id) ON DELETE CASCADE ON UPDATE CASCADE,
    PRIMARY KEY (organization_member_id, scheduled_planting_id)
);

CREATE TABLE scheduled_plantings_have_volunteers
(
    planting_event_id INTEGER REFERENCES scheduled_plantings (event_id) ON DELETE CASCADE ON UPDATE CASCADE,
    volunteer_id      INTEGER REFERENCES residents (id) ON DELETE CASCADE ON UPDATE CASCADE,
    PRIMARY KEY (planting_event_id, volunteer_id)
);

CREATE TABLE planting_events_have_volunteers
(
    planting_event_id INTEGER REFERENCES planting_events (scheduled_planting_id) ON DELETE CASCADE ON UPDATE CASCADE,
    volunteer_id      INTEGER REFERENCES residents (id) ON DELETE CASCADE ON UPDATE CASCADE,
    PRIMARY KEY (planting_event_id, volunteer_id)
);


-- Get the status of a tree
-- https://neon.tech/postgresql/postgresql-plpgsql/postgresql-create-function
CREATE OR REPLACE FUNCTION get_tree_request_status(p_tree_request_id INTEGER)
    RETURNS TEXT
AS
```

```sql
-- Get the status of a tree
-- https://neon.tech/postgresql/postgresql-plpgsql/postgresql-create-function
CREATE OR REPLACE FUNCTION get_tree_request_status(p_tree_request_id INTEGER)
    RETURNS TEXT
AS
$$
DECLARE
    v_status TEXT;
BEGIN
    SELECT
        -- https://www.w3schools.com/sql/sql_case.asp
        CASE
            WHEN pe.successful IS TRUE THEN 'completed'
            WHEN ve.additional_visit_required IS FALSE THEN 'waiting for planting'
            WHEN p.status = 'approved' THEN 'waiting for visit'
            WHEN tr.approved IS TRUE THEN 'needs permit'
            WHEN tr.approved IS FALSE THEN 'denied'
            ELSE 'pending approval'
            END
    INTO v_status
    FROM tree_requests tr
             LEFT JOIN permits p
                       1<->0,n ON tr.id = p.tree_request_id
             LEFT JOIN scheduled_visits sv
                       ON tr.id = sv.tree_request_id
             LEFT JOIN visit_events ve
                       ON sv.event_id = ve.scheduled_visit_id
             LEFT JOIN scheduled_plantings sp
                       ON tr.id = sp.tree_request_id
             LEFT JOIN planting_events pe
                       ON sp.event_id = pe.scheduled_planting_id
    WHERE tr.id = p_tree_request_id;
    RETURN v_status;
END;
$$ LANGUAGE plpgsql;
```