FIT 5120: Support Document

Team Name: Spaghetti Marshmallow King
Team number: TA21

Team members:
Zihan Yin (34502297)
Yiyang Chen (34562109)
Klarissa Jutivannadevi (32266014)
Yu Xie (34240136)
Yean Yee Tan (32025181)
Weiqi Xie(34009000)

TABLE OF CONTENTS

1. Introduction

1.1. Purpose & Scope

PlantX is a practical, Australia-focused service that supports home gardeners with clear, location-relevant guidance. This **Support Document** is written for sponsor operators and admins: it equips you to **run, monitor, back up, restore, and update** the service independently without developer support from the student team. It outlines day-to-day and week-to-week procedures, data refresh responsibilities, access and security expectations, and the change-control steps required to introduce future improvements while protecting availability, privacy, and licensing obligations.

1.2. Audience

This Support Document is for sponsor personnel who operate PlantX day-to-day

- **Primary readers: Sponsor Operator** (runs daily/weekly procedures) and **Sponsor Admin** (access, DNS/SSL, approvals).
- **Also relevant to: Data Steward** (data sources, ETL cadence, QA) and **Analyst/PM** (awareness of refresh cycles and status).

2. Support

2.1. Host

Overview

PlantX runs on AWS. The frontend (Vue 3 + Vite) is built to S3 and served via CloudFront (HTTPS). All API calls go through API Gateway (WAF) to AWS Lambda (Python) in a private VPC. Data is in Amazon RDS (MySQL) and S3. Private components aren't publicly exposed.

Environments & Access

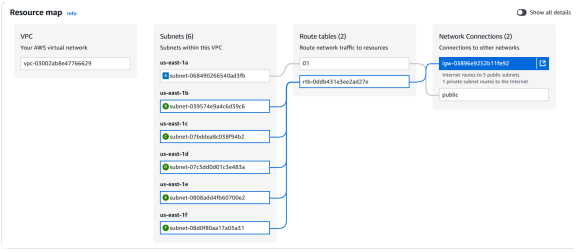
Environment	URL	Notes
Production	www.plantx.me	Public HTTPS via CloudFront.
Staging - Iteration 1	iteration1.plantx.me	Archived project of iteration 1. May be retired after project completion
Staging - Iteration 2	iteration2.plantx.me	Same as iteration 1
Staging - Iteration 3	www.plantx.me	Last iteration, final product release

Host Information:

Frontend(S3+CloudFront)	
S3 Region	us-east-1
S3 ARN	arn:aws:s3:::plantx-version2
CloudFront domain	du0rkevcjwmzy.cloudfront.net, www.plantx.me
CloudFront ARN	arn:aws:cloudfront::056000095164:distribution/E3TU0IWO3JH8DG
ACM ARN	arn:aws:acm:us-east-1:056000095164:certificate/2759a933-6422-445c-822e-7f1fa6dc307e

Backend (API gateway+ lambda)	
API type	Rest API
API ARN	arn:aws:apigateway:us-east-1::/restapis/ky21h193r2
Lambda ARN	arn:aws:lambda:us-east-1:056000095164:function:preSignedUpload, arn:aws:lambda:us-east-1:056000095164:function:test2, arn:aws:lambda:us-east-1:056000095164:function:getPlantDetail, arn:aws:lambda:us-east-1:056000095164:function:UploadToS3, arn:aws:lambda:us-east-1:056000095164:function:getStateBoundaries, Arn:aws:lambda:us-east-1:056000095164:function:recommend, Arn:aws:lambda:us-east-1:056000095164:function:diseases-query, arn:aws:lambda:us-east-1:056000095164:function:UploadToDiseaseS3, arn:aws:lambda:us-east-1:056000095164:function:getPlantsList, arn:aws:lambda:us-east-1:056000095164:function:getPlantsMapData, arn:aws:lambda:us-east-1:056000095164:function:TreeLocator, Arn:aws:lambda:us-east-1:056000095164:function:plants_disease, Arn:aws:lambda:us-east-1:056000095164:function:plant_image_query, arn:aws:lambda:us-east-1:056000095164:function:preSignedUploadDiseases,

	Arn:aws:lambda:us-east-1:056000095164:function:tsx-trend, Arn:aws:lambda:us-east-1:056000095164:function:gardening_clubs, arn:aws:lambda:us-east-1:056000095164:function:TsxTrendDocker

Network(VPC)	
VPC ID	vpc-03002ab8e47766629
IPv4 CIDR	172.31.0.0/16
Main network ACL	acl-04dbd0e00f7bcaec6
Resource map	

2.2. Full Stack

PlantX does **not** use a traditional LAMP server. Instead, it runs a **static frontend + serverless APIs on AWS:**

- **APIs:** All requests go through **API Gateway** (WAF enabled) to **AWS Lambda** functions inside a **private VPC**.
- **Data: Amazon RDS (MySQL)** for relational data; **S3** for images and other large objects.
- **OS:** The underlying computer is managed by AWS and based on **Linux**, but there is **no single VM to SSH into** and no Apache to restart.

Operator focus for this runtime:

- Confirm API routes are reachable and **Lambda errors/latency** remain healthy.
- Check **RDS connectivity** and watch for slow queries.
- Ensure **S3 reads/writes** succeed for uploads and image fetches.
- After a frontend release, **invalidate CloudFront** so users see the latest build.

Hosting Stack & Versions Required

Component	**Role**	**Version / Note**
Amazon S3 (origin)	Serves built frontend files	Managed service (no version)
AWS Cloudfront (CDN)	Global delivery of frontend	Managed service (no version)

AWS API Gateway	API entry point	Managed service (no version)
AWS WAF	Edge protection	Managed service (no version)
AWS Lambda runtime	Runs backend functions	Python 3.10
Amazon RDS for MySQL	Primary database	MySQL 8.0
Frontend framework	SPA used by site	Vue 3.x
Build toolchain	Build and dev server	Node.js 18 LTS + npm

2.3. AWS RDS MySQL

These instructions show how an operator connects to the **managed RDS MySQL** database for PlantX to run basic checks (e.g., verify tables, row counts) or perform an **approved** data load. You're not installing a database server since RDS is already running in AWS. Replace the placeholders with your credentials, connect over TLS, and prefer **read-only** access unless a data change has been approved.

Field	Value
Host	`database-plantx.cqz06uycysiz.us-east-1.rds.amazonaws.com`
Port	3306
Database	FIT5120_PlantX_Database
Username	zihan
Password	2002317Yzh12138.

How to connect (choose which method you are connecting with)

- **MySQL CLI**

  ```
  mysql -h database-plantx.cqz06uycysiz.us-east-1.rds.amazonaws.com \
  -P 3306 -u zihan -p --ssl-mode=REQUIRED FIT5120_PlantX_Database
  ```

- **MySQL Shell (SQL mode)**
  - mysqlsh --sql zihan@database-plantx.cqz06uycysiz.us-east-1.rds.amazonaws.com:3306/FIT5120_PlantX_Database --ssl-mode=REQUIRED
- **DSN pattern**
  - mysql://zihan:2002317Yzh12138.@database-plantx.cqz06uycysiz.us-east-1.rds.amazonaws.com:3306/FIT5120_PlantX_Database?ssl-mode=REQUIRED

2.4. Setup Environment

Installation Version
Platform tools

- Node.js 18 LTS (with npm)
- Python 3.10

- MySQL client 8.0+ (Workbench, MySQL Shell, or `mysql`)

Python packages

- torch **2.5.1 or** torch **2.0.0**
- torchvision **0.20.1 or** torchvision **0.15.0**
- transformers **4.49.0**
- pillow **11.3.0 or** Pillow **9.5.0** (use only one; names differ by case, but it's the same library)
- numpy **1.26.4 or** `numpy>=1.21.0,<2.0.0` (use one constraint; 1.26.4 satisfies the range)
- timm **0.9.12**
- boto3 **1.26.137**
- Flask **3.1.2**
- flask_cors **6.0.1**
- geopandas **1.1.1**
- mysql-connector-python **9.3.0**
- pandas **2.3.3**
- Shapely **2.1.2**

Recommendation: use **torch 2.5.1 + torchvision 0.20.1 + pillow 11.3.0 + numpy 1.26.4** unless a specific service in the repo requires the older set.

2.4.1. Frontend (Vue/Vite)

1. Install
   `npm ci`
2. Run locally
   `npm run dev`
3. Build (for S3/CloudFront)
   `npm run build`

2.4.2. Python/ML & data environment

1. Create and activate a virtual environment
   - `python3.10 -m venv .venv`
   - For Mac/Linux:
     - `source .venv/bin/activate`
   - For Windows:
     - `.venv\Scripts\activate`
2. Install packages
   - **Recommended**

     ```
     pip install \
       torch==2.5.1 torchvision==0.20.1 \
       transformers==4.49.0 pillow==11.3.0 numpy==1.26.4 \
       timm==0.9.12 boto3==1.26.137 \
       Flask==3.1.2 flask_cors==6.0.1 \
       geopandas==1.1.1 mysql-connector-python==9.3.0 \
       pandas==2.3.3 Shapely==2.1.2
     ```
   - Using older compatibility

     ```
     pip install \
     ```

```
torch==2.0.0 torchvision==0.15.0 \
transformers==4.49.0 Pillow==9.5.0 "numpy>=1.21.0,<2.0.0" \
timm==0.9.12 boto3==1.26.137 \
Flask==3.1.2 flask_cors==6.0.1 \
geopandas==1.1.1 mysql-connector-python==9.3.0 \
pandas==2.3.3 Shapely==2.1.2
```

3. Back up

We store our code progress on GitHub. GitHub keeps the full change history, supports protected branches/tags, and lets us roll back to any known good commit. Because PlantX is a static frontend + serverless APIs, we clone locally (or in CI), build, and publish build artifacts to S3/CloudFront. This preserves auditability (what commit is live), and every build can be re-created or rolled back by checking out its tag.

Out GitHub (HTTPS): https://github.com/ychen266/2025-08-SDG13-Plant-X-Website.git

To clone to your local directory:

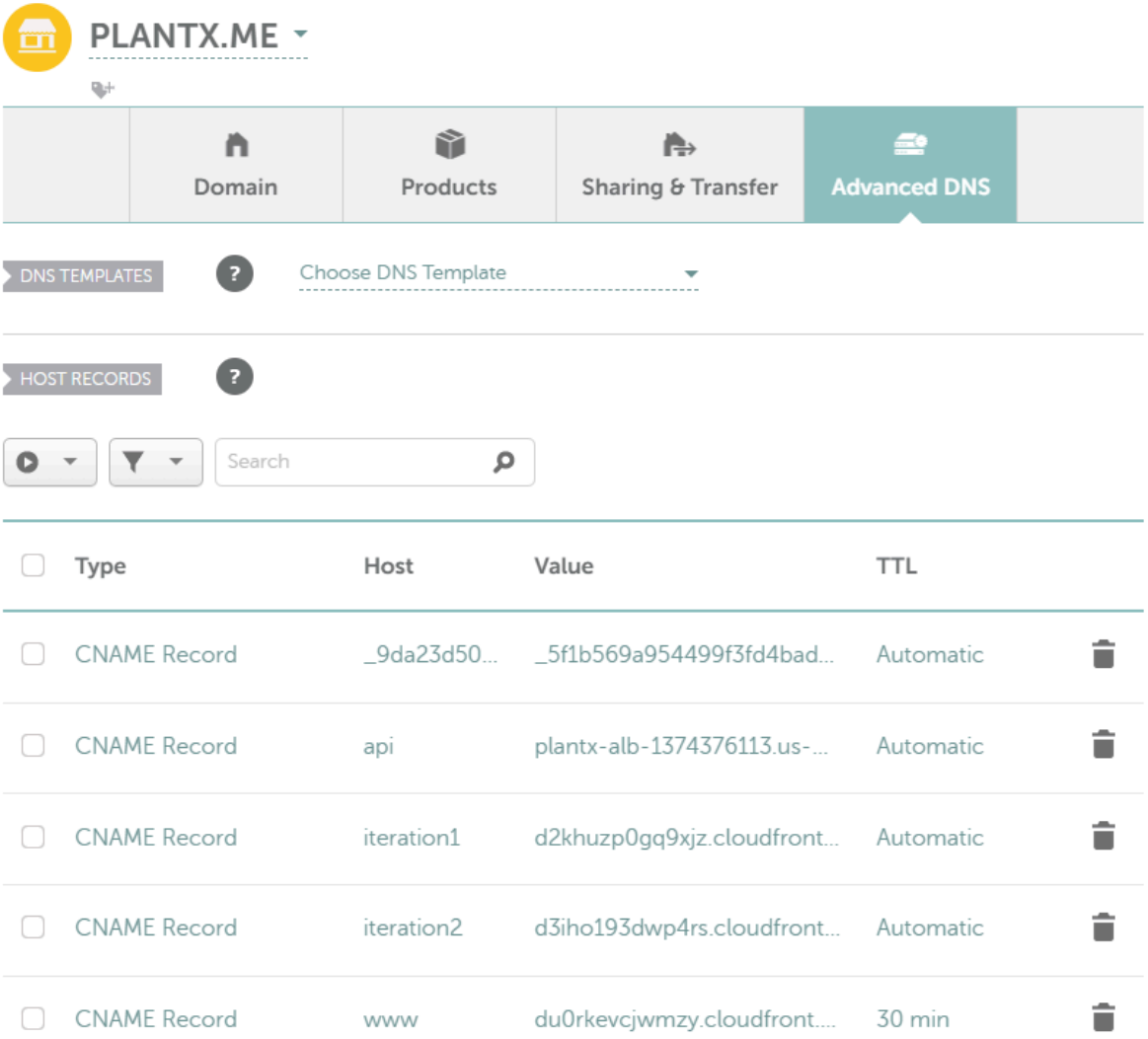
1. Create a new folder to store the project
    - `mkdir -p ~/plantx && cd ~/plantx`
2. Clone the repo by HTTPS
    - `git clone https://github.com/ychen266/2025-08-SDG13-Plant-X-Website.git`

Note: For deployment we **build** and push artifacts to S3 (or CI does). We do **not** run `git clone` on a server.

4. Security

4.1. Domain and SSL Certificate

Only when you want to move to a new domain, visit: www.namecheap.com. Apply a new domain, then set new records with the same certificate from AWS Certificate Manager



4.2. Third-Party Security Testing

To ensure the security and reliability of PlantX, sponsor admins may engage external security testing providers to perform periodic assessments. These tests should focus on:

- **CloudFront and API Gateway exposure**: Validate WAF rules, rate limiting, and bot protection.
- **Lambda endpoints**: Confirm that input validation and authentication are enforced.
- **RDS MySQL access**: Ensure TLS is required, and credentials are rotated regularly.
- **Static frontend (Vue)**: Scan for XSS, CSRF, and dependency vulnerabilities.

Recommended tools include:

- AWS Inspector (for Lambda and RDS)
- OWASP ZAP or Burp Suite (for API fuzzing)
- Snyk or npm audit (for frontend dependency checks)

All findings should be documented and reviewed by the Sponsor Admin. Critical issues must be remediated before the next release cycle.

4.3. Application Layer Security

PlantX enforces application-level security through multiple layers:

- **Input validation:** Frontend forms and backend functions sanitize user input to prevent injection attacks.
- **Rate limiting:** API Gateway enforces request quotas to prevent abuse.
- **CORS Policy:** Only trusted domains are allowed to access API endpoints.

5. Data Management

For more details, please visit: Data Management Plan

PlantX uses a structured relational schema that supports plant search and details, disease lookup, location-based recommendations, threatened species, climate trends, urban tree data, and the community directory. Keys are simple integers, and relationships are mostly one-to-many, with a few link tables for joins. Below, each feature area lists the tables involved and the key relationships, along with the screenshots.

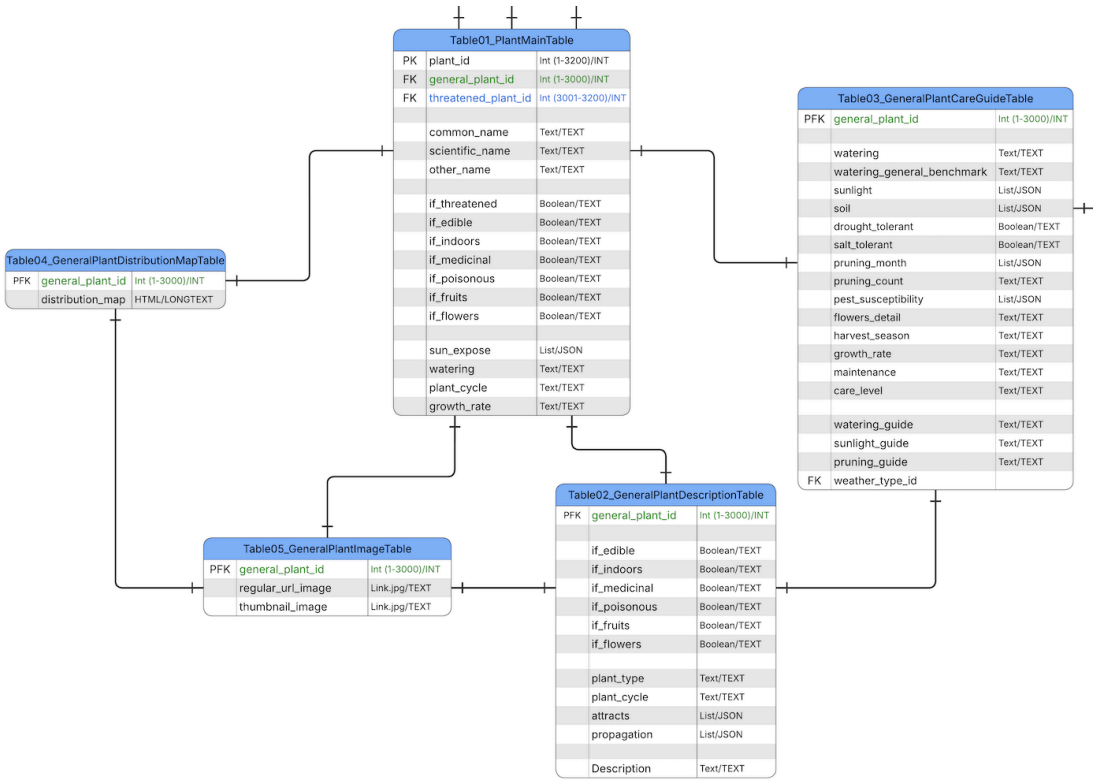
5.1. Plant Catalogue

The canonical plant record lives in **Table01_PlantMainTable** (`plant_id`, `general_plant_id`, optional `threatened_plant_id`). Rich description fields live in **Table02_GeneralPlantDescriptionTable** (e.g., if_edible, if_indoors, plant_type, Description). Care instructions are in **Table03_GeneralPlantCareGuideTable** (watering, soil, sunlight, pest notes, pruning, care_level). Images are stored in **Table05_GeneralPlantImageTable** (regular_url_image, thumbnail_image). A distribution/overview map blob is stored in **Table04_GeneralPlantDistributionMapTable** (`distribution_map`) when available.

Key Relationships

- Table01_PlantMainTable (1) → (1) Table02_GeneralPlantDescriptionTable (via `general_plant_id`)
- Table01_PlantMainTable (1) → (1) Table03_GeneralPlantCareGuideTable (via `general_plant_id`)
- Table01_PlantMainTable (1) → (many) Table05_GeneralPlantImageTable (via `general_plant_id`)
- Table01_PlantMainTable (1) → (1) Table04_GeneralPlantDistributionMapTable (via `general_plant_id`)

Screenshot



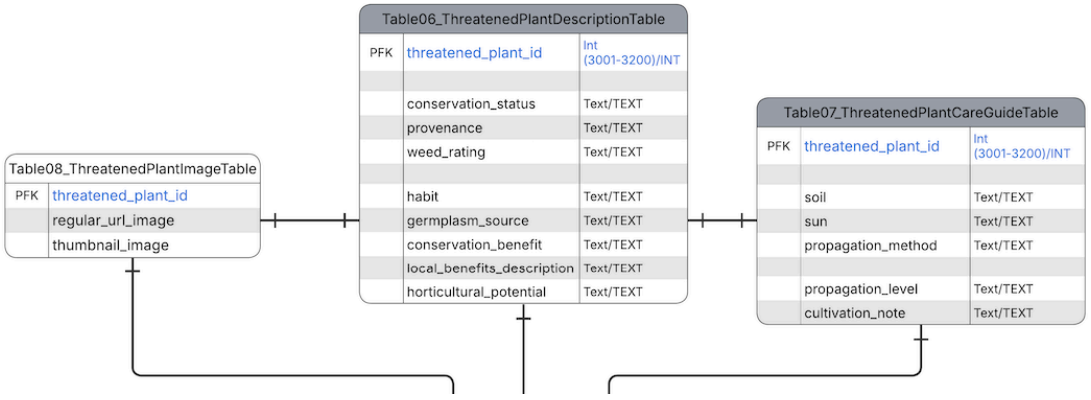
5.2. Threatened Species

Threatened species have their own extended profile in **Table06_ThreatenedPlantDescriptionTable** (status, provenance, local benefits, etc.) and specific care advice in **Table07_ThreatenedPlantCareGuideTable** (soil, sun, watering, propagation_method, cultivation_note). When a plant in Table01 is threatened, `threatened_plant_id` links to these tables for richer content.

Key Relationships

- Table01_PlantMainTable (many) → (1) Table06_ThreatenedPlantDescriptionTable (via `threatened_plant_id`)
- Table06_ThreatenedPlantDescriptionTable (1) → (1) Table07_ThreatenedPlantCareGuideTable (via `threatened_plant_id`)

Screenshot



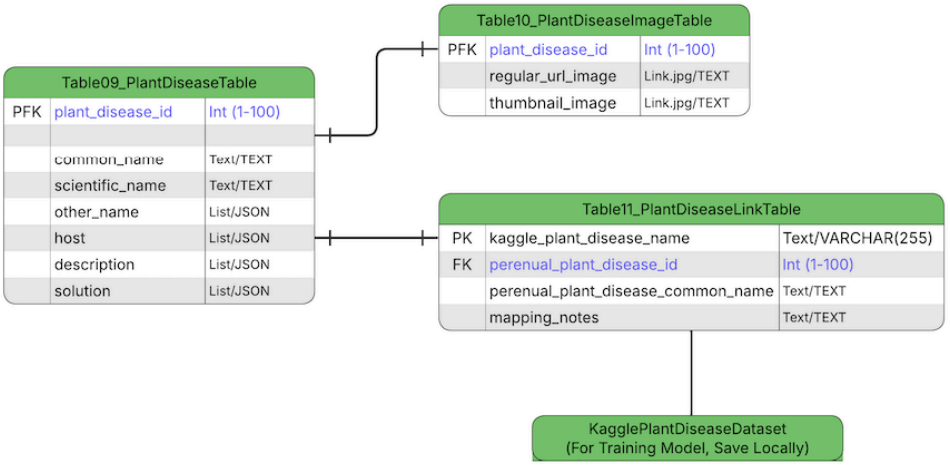
5.3. Plant Disease

Disease lookup returns rows from **Table08_PlantDiseaseTable** (common_name, scientific_name, host, description, solution). Disease images are in **Table09_PlantDiseaseImageTable** (regular_url_image, thumbnail_image). The link table **Table11_PlantDiseaseLnkTable** maps external names (e.g., Kaggle disease labels) to internal disease IDs for flexible matching.

Key Relationships

- Table08_PlantDiseaseTable (1) → (many) Table09_PlantDiseaseImageTable (via `plant_disease_id`)
- Table11_PlantDiseaseLnkTable (many) → (1) Table08_PlantDiseaseTable (via `perenual_plant_disease_id`)

Screenshot



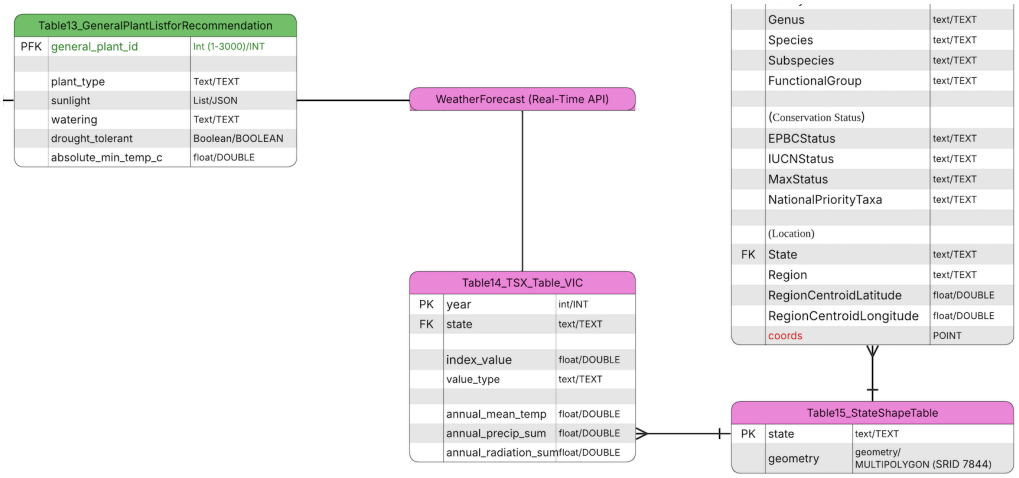
5.4. Location-Based Recommendation

Recommendations are produced by comparing local weather summaries with plant needs. The matching thresholds/derived fields live in **Table13_GeneralPlantListforRecommendation** (per `general_plant_id`: plant_type, sunlight, watering, drought_tolerant, absolute_min_temp_c). Weather input for VIC trends is captured in **Table14_TSX_Table_VIC** (year, index_value, mean/annual temps, rainfall, radiation). State boundaries for mapping are stored in **Table15_StateShapeTable** (geometry MULTIPOLYGON SRID 7844).

Key Relationships

- Table13_GeneralPlantListforRecommendation (many) → (1) Table01_PlantMainTable (via `general_plant_id`)
- Table14_TSX_Table_VIC (many) → (1) Table15_StateShapeTable (via `state`), for map join/filter by state

Screenshot



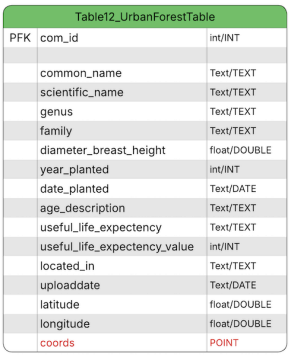
5.5. Urban & Wild Trees

City/urban tree features use **Table12_UrbanForestTable** (common_name, scientific_name, genus/species, diameter, year_planted, coords/POINT). This powers “what grows around me” and lets us plot trees on the map with basic attributes.

Key Relationships

- Table12_UrbanForestTable is standalone for points

Screenshot



5.6. Climate Trend & Species Monitoring

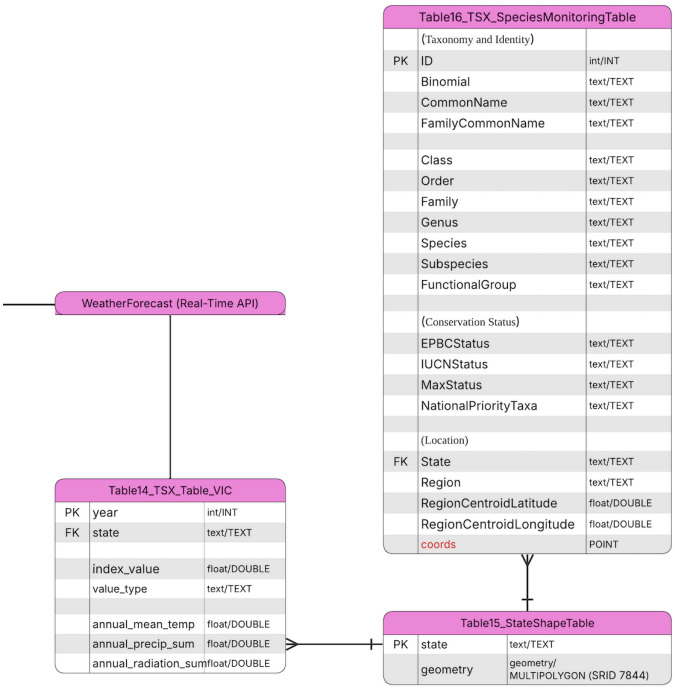
For climate-plant trend visualisations, **Table14_TSX_Table_VIC** (metrics by `year`, `state`) combines with state geometry from **Table15_StateShapeTable**. Taxonomy and conservation metadata appear in **Table16_TSX_SpeciesMonitoringTable** (Binomial, Family, Genus, Species, status fields), supplying consistent naming/status where needed.

Key Relationships

- Table14_TSX_Table_VIC (many) → (1) Table15_StateShapeTable (via `State`)

- Table16_TSX_SpeciesMonitoringTable is a taxonomy/identity reference (joins by names/IDs as needed).

Screenshot



5.7. Community Clubs

The community page reads from **Table17_AustralianGardenClubTable** (Club, Link, Meeting_day/week/hour, Contact, State, Location). This supports the listing, region/state filter, and “Today’s meetings.”

Key Relationships

- Table17_AustralianGardenClubTable (many) → (1) State

Screenshot