FIT 5120: Maintenance 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

This Maintenance Document sets out the practical steps and standards to keep PlantX working smoothly across handovers and future updates. It explains how to follow the current development cycle, how to package and release changes with care, and how to handle data and configuration so behaviour stays consistent. The aim is simple: maintain the quality that users already experience while avoiding regressions and unnecessary risk.

Following these guidelines helps the maintainer, developer, and sponsor technical contact work in the same way and in the same order each time. When everyone uses the same process, issues are easier to find, fixes are faster to apply, and the platform remains reliable. This consistency also supports clear sign-off and makes it easier to meet expectations on performance, security, and accessibility.

The document is a base for safe improvement. Start with the steps here to keep PlantX stable, then layer on new work in small, well-tested increments. Use this as the source of truth for routine maintenance, dependency updates, and controlled changes, so future features can be added without breaking what already works.

2. Maintenance Team Composition

Below is a **role-based** view of who maintains PlantX after handover. It's tailored to what the app actually does: a Vue/Vite frontend, serverless APIs, image upload/recognition, and a recommendation feature. Roles can be combined in small teams.

2.1. Roles, Responsibilities, Skills & Qualifications

Roles	Responsibilities	Skills & Qualification
Product Owner	● Create and prioritise the backlog ● Define requirements for fixes and enhancements ● Primary contact for stakeholders ● Ensure standards and ethical considerations are met	● Proficiency in backlog grooming and writing clear user stories/acceptance criteria ● Strong stakeholder communication and prioritisation skills ● Certification in Product Ownership/Management (e.g., SAFe PO/PM) preferred
Frontend Developer	● Maintain page structure and design ● Build user-facing features ● Ensure UI/UX is consistent and accessible ● Validate input before backend ● Optimise for mobile	● Proficiency in modern HTML, CSS, and JavaScript/TypeScrip ● Proficiency in Vue 3 and Vite (or similar SPA frameworks/tools) ● Experience with responsive design and WCAG accessibility basics ● Proficiency with Git-based

	view ● Build reusable component	workflow (GitHub/GitLab)
Backend Developer	● Maintain REST endpoints for search, image upload, image fetch; input validation and error handling ● Logging/metrics ● small performance fixes ● integrate with database and object storage	● Proficiency in Python or JavaScript for serverless APIs ● Proficiency in REST/JSON patterns and request validation ● Proficiency in SQL fundamentals and basic query tuning ● Experience with cloud SDKs and managed services is a plus
AI/ML Engineer (Recommendation & Vision)	● Own the recommendation logic and image recognition pipeline used by PlantX ● curate and version datasets ● retrain and evaluate models ● monitor inference latency and model quality ● manage safe fallbacks when models are unavailable ● document model changes	● Proficiency in Python for data and ML workflows ● Experience with ML frameworks (e.g., scikit-learn; exposure to PyTorch/TensorFlow a plus) ● Proficiency in experiment tracking and model evaluation metrics ● Experience with dataset versioning and basic MLOps hygiene
Data Scientist	● Prepare and cleanse data used by recommendations and reports; design simple visuals for stakeholders ● Validate data refreshes ● surface insights that improve model or UX quality	● Proficiency in Python (pandas/NumPy) and SQL ● Proficiency in a visualisation tool ● Proficiency in a web visualisation library (e.g. D3) ● Experience with basic statistical analysis and A/B result reading
Database Administrator	● Keep the database healthy ● manage access ● coordinate backup/restore windows with Support ● assist with dataset imports for staging and production	● Proficiency in Database Management Systems (preferably MySQL) ● Proficiency in SQL and indexing basics ● Knowledge of data models, schema design, and integrity constraints ● Experience with backup,

	● review indices for recurring slow queries	restore, and performance monitoring
Security Specialist	● Review dependency risk ● guide safe key/secret handling ● tune protective rules ● monitor and respond to security incidents.	● Proficiency in application security fundamentals and OWASP risks ● Experience with protective controls and incident response practices ● Proficiency in encryption, key management, and access control concepts ● Experience conducting basic risk assessments and mitigations
QA/Test Engineer	● Maintain test cases for UI, API, and model-assisted features ● run smoke/regression tests ● log defects ● verify fixes	● Proficiency in software testing principles and test case design ● Experience with browser and API testing tools (e.g., Postman) ● Experience creating clear bug reports and acceptance checklists ● Test automation exposure is a plus
Cloud Specialist	● Keep hosting configuration stable ● manage identities and access; tune caching and delivery ● watch cost and scaling ● support deploy paths	● Proficiency in cloud fundamentals (identity/access, networking, caching/CDN concepts) ● Experience reading service metrics and cost reports ● Experience coordinating changes across environments ● CI/CD awareness is a plus
DevOps / CI-CD Engineer	● Maintain build and deploy pipelines for frontend, APIs, and model artifacts ● manage environment variables and secrets ● enable rollbacks.	● Proficiency in CI/CD tooling and pipeline design ● Proficiency in scripting and automation for builds and deploys ● Experience with artifact/version management and environment promotion ● Experience managing secrets in pipelines
Scrum Master	● Facilitate stand-ups, planning, and retros	● Proficiency in Agile facilitation and sprint

	● track progress ● remove blockers ● keep ceremonies lightweight and on time.	ceremonies ● Experience with issue tracking tools and metrics ● Certification in Scrum (e.g., CSM or PSM) preferred
Release Manager	● Plan release windows ● coordinate go/no-go ● publish change notes ● make the rollback call within the window.	● Proficiency in release planning and stakeholder communication ● Experience using checklists and runbooks ● Experience monitoring basic health signals during releases
UI/UX Designer	● Keep design consistent; refine flows from feedback ● provide specs/assets to frontend ● support accessibility checks	● Proficiency in modern design tools and component libraries ● Experience with responsive patterns and accessibility guidelines ● Experience handing off designs and tokens to engineers
Technical Writer	● Keep Maintenance and Support docs aligned ● record what changed and why ● update links and placeholders	● Proficiency in concise, plain-language documentation ● Experience with versioned docs and changelogs ● Attention to detail and structure
Support Liaison	● Bridge incidents from operations to maintenance ● capture context and steps to reproduce ● confirm fixes meet operator needs	● Proficiency in incident communication and triage ● Experience gathering logs/symptoms for engineers ● Familiarity with ticketing and status updates

2.2. Essential vs Optional Roles

If, after takeover, the project is in a steady state with only routine upkeep, some roles remain essential while others are optional. Below explains why.

Essential for continuity:

- **Product Owner**
  Keeps a single, prioritised backlog and approves scope so changes stay aligned with sponsor needs. Without this, even small fixes stall or drift.

- **Frontend Developer**
  Maintains the UI, wiring to the API base and handling user-facing issues. Small UI fixes and accessibility touch ups are regular maintenance work.
- **Backend Developer**
  Look after search, image upload, and image fetch endpoints. Minor logic fixes, validation, and logging are routine and time sensitive.
- **AI/ML Engineer**
  Monitors model quality and inference health, retrains when data shifts, and keeps safe fallbacks in place. This can be part time but is needed to keep recommendations credible.
- **Data Scientist**
  Ensures the data feeding the models and reports is clean, current and representative. Validates refreshes, tracks key metrics, and flags drift so the AI/ML Engineer knows when retraining is warranted.
- **Database Administrator**
  Manages access, imports small datasets, and watches basic performance. Ensures recoverability stays intact through simple checks.
- **QA/Test Engineer**
  Verifies fixes and runs smoke/regression on iteration builds. Prevents regressions reaching users during routine updates.
- **Cloud Specialist / DevOps**
  Keeps environments, access and deployments working. Small pipeline or configuration issues can block all other work.

Optional (needed when maintenance is in a larger scale):

- **Security Specialist**
  Useful for formal reviews and incidents, but routine dependency checks and basic controls can be covered by DevOps and Backend if change volume is low.
- **Scrum Master**
  Helpful for larger teams. In a small post-handover team, the Product Owner can facilitate short stand ups and planning.
- **UI/UX Designer**
  Valuable for bigger UI changes. If maintenance is minor, Frontend can apply small tweaks using existing patterns.
- **Release Manager**
  Helpful during busy release periods. For light maintenance, DevOps and Product Owner can coordinate releases using a short checklist.
- **Technical Writer**
  Excellent for polished documentation. If change is minimal, engineers can update the docs briefly at the end of each iteration.
- **Support Liaison**
  Bridges day-to-day operator issues to the maintenance team, ensuring clear reproduction steps, priority, and feedback loop.

These disciplines becomes essential when operations and maintenance are run by different teams or when incident volume is non-trivial.

3. Development Cycle

We followed an Agile rhythm across **three iterations**. At the start of each iteration we agreed on a small set of website features and fixes, then tracked work on a **Kanban board** in **LeanKit**. We used **Git** for version control, kept branches short, and promoted builds through CI/CD so changes moved from iteration sites to production in a controlled way. We met regularly, held brief daily stand-ups to surface blockers, and recorded decisions on the relevant task or pull request. Keep this flow: it maintains steady delivery and protects reliability.

3.1. Approach

Use this section to run the process. Plan small, shippable tasks, keep work-in-progress low, and always leave a clear trail.

- **Planning:** pick a **small set of tasks** per iteration with one-line outcomes and simple acceptance notes.
- **Stand-ups:** 10 minutes max to surface blockers and move tasks.
- **Reviews:** one reviewer per change.
- **Required artefacts:** Backlog item → **Task** (risk, complexity, acceptance notes) → PR (linked, small diff, screenshot or API sample) → **change note** after release

Task Template

- Title: short action (e.g., “Reduce image preview timeout”)
- Outcome: user-visible result in one sentence
- Risk: Low / Medium / High
- Complexity/Uncertainty/Effort: clothes sizing chart (XS to XL)
- Acceptance notes: bullet list of behaviours to verify
- Owner / Reviewer: role or name

3.2. Workflow

Move work through these columns using the entry/exit criteria to keep flow clear.

Column	Entry Criteria	Exit Criteria	Primary Owner
Backlog	● Idea captured ● One-line outcome written	● Prioritised with risk/complexity set	● Product owner
To Do	● Selected for the iteration ● Acceptance notes added	● Developer starts work	● Product owner
Doing	● Branch created ● Work in progress	● PR opened and linked to the task	● Developer
Done	● Continuous integrity check pass ● At least one review complete	● Ready for acceptance review	● Developer
Reviewed & Accepted	● Behaviour matches acceptance notes ● Change note recorded	● Eligible for production release	● QA tester

Risk and Complexity & Uncertainty & Effort (CUE) Rubric

Level	Choose When	Extra Controls
Risk: Low	● Copy, minor styling, trivial bug	● 1 reviewer
Risk: Medium	● Normal API or UI change	● 1 reviewer + iteration smoke
Risk: High	● Auth, uploads, recommendation, or error-prone code ● Heavy code risk of timeout	● Pair of 2 reviewers + rollback preparation
CUE: XS-S	● Small, well understood	● Normal review
CUE: M	● Moderate scope or new topic encountered	● Reviewer confirms test notes
CUE: L-XL	● Cross component or model work	● Pairing recommended + second review

3.3. Pairing Recommendation

Pair selectively to reduce risk and share knowledge. Always follow with a brief third-party review.

When to pair	Who pairs	Why it helps
High-risk or 4-5 point task	Backend ←→ QA	Build tests while shaping the change; catch edge cases early
New maintainer onboarding	Frontend ←→ Backend	Share end-to-end context on a small task and reduce rework
Recommendation or image model change	AI/ML ←→ Data Scientist	Align data prep, metrics, and acceptance thresholds before enabling
Performance issue in search/uploads	Backend ←→ Cloud/DevOps	Correlate logs, throttles, and cache to isolate the bottleneck
UI issue tied to an API contract	Frontend ←→ Backend	Trace request/response to fix mapping or the contract quickly

4. Product

4.1. Product Features

This section gives a quick, scenario-style view of what each epic enables so you can understand user intent and keep behaviour consistent as you evolve the product. The epics below come from Iterations 1–3.

<table>
<tr><th colspan="2">Epic</th><th colspan="2">Feature</th><th>Acceptance criteria</th></tr>
<tr><td>1.</td><td rowspan="3">Understand PlantX and how to use the website

As a novice gardener, I want to understand the platform's vision, so that I can see how my gardening choices connect to broader environmental goals.

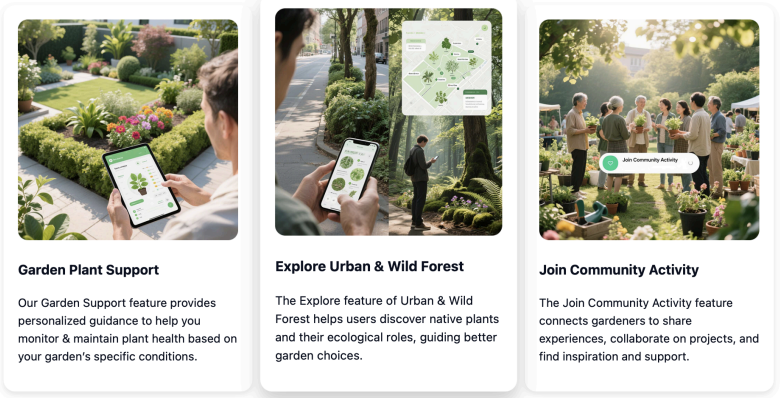
Epic Description: Give new users a fast understanding of PlantX and a clear starting point, with concise guidance and direct links to core tools.</td><td>1.1.</td><td>Clear homepage introduction
As a gardener, I want to see a clear introduction on the homepage, so that I immediately understand what the platform offers.

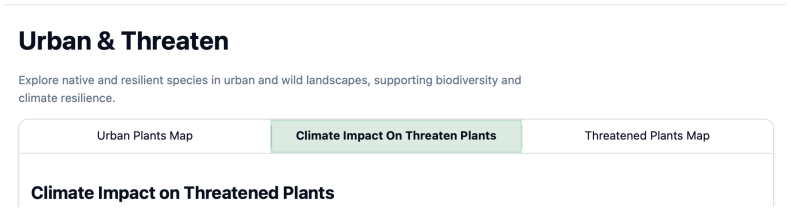
Scenario:
A first-time visitor opens home on mobile. Above the fold they see the headline "Keep your garden thriving under changing climate" with 2–3 sentences explaining what PlantX does and why it matters. A prominent Explore button is visible below the text.</td><td>A short welcome sets context
● Given I open the homepage,
● when I first visit,
● then I should see a brief summary of the platform's purpose and benefits.

Developer Guidelines
● Keep the summary short, friendly, and easy to scan (2–3 sentences).
● Make the Explore button easy to spot and use; it should move people forward without interrupting them.
● Use everyday language; avoid technical terms on this first screen.
● Ensure the same welcome appears clearly on both phone and desktop.</td></tr>
<tr><td colspan="3">The main page on top showing short information of our page and how they can explore

Plant'X
Keep Your Garden Thriving Under Changing Climate in Australia
Caring for plants is more than a hobby. With proper sunlight, soil, water, and compost, gardens thrive and bring joy. From city balconies to wild landscapes, plants connect us to nature—urban greenery builds resilience, and wild plants preserve biodiversity. Tending our gardens means caring for the planet.
Explore

  </td></tr>
<tr><td>1.2.</td><td>Highlight key features with cards
As a user, I want the</td><td>Each card explains a single action
● Given I am on the homepage,</td></tr>
</table>

			homepage to highlight key features of the platform, so that I know where to start exploring. **Scenario:** Below the welcome text, the visitor sees clear cards.Each card includes a short one-line description and a main button (for example, **Open Search**). The visitor clicks **Open Search** on the **Find Plants** card and is taken straight to the search page, where the text box is ready to type in. If they go back, they return to the same place on the homepage.	• **when** I look at the feature cards, • **then** each card has a title, description, and a main button that states the action **Clicking a card takes me straight there** • Given I am on the homepage, • when I click a feature card, • then I am directed to the matching page or tool. **Developer Guidelines** • Write action labels that tell people exactly what will happen (for example, “Open Search”). • Keep card text short and consistent so choices are easy to compare. • After someone clicks a card, place them at the start point of that page so they can begin right away. • Make it easy to come back and choose another card without losing their place.
			The screenshot shows user about to click on the ‘Explore Urban & Wild Forest’  Once clicked, it directs to the Urban & Threatened Plant page	



1.3.

Learning more on plant for higher climate awareness

As an urban gardener, I want an overview section that explains sustainable gardening and environmental benefits, so that I understand how my actions contribute to conservation.

Scenario:
From the homepage, the visitor selects **Learn More**. A simple view appears with short messages and a picture (for example, “Plants are suffering from excessive heat” with a one-sentence explanation). The visitor can move to the next message, go back to the previous one, close the view, and open it again later from the same link

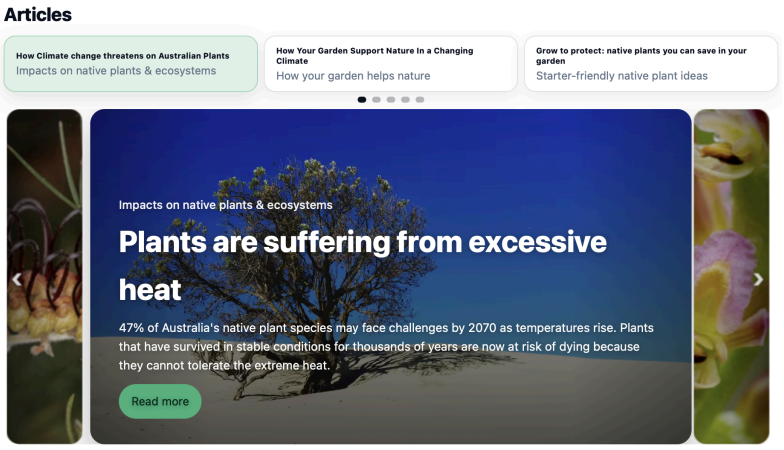
A quick information connecting gardening to climate

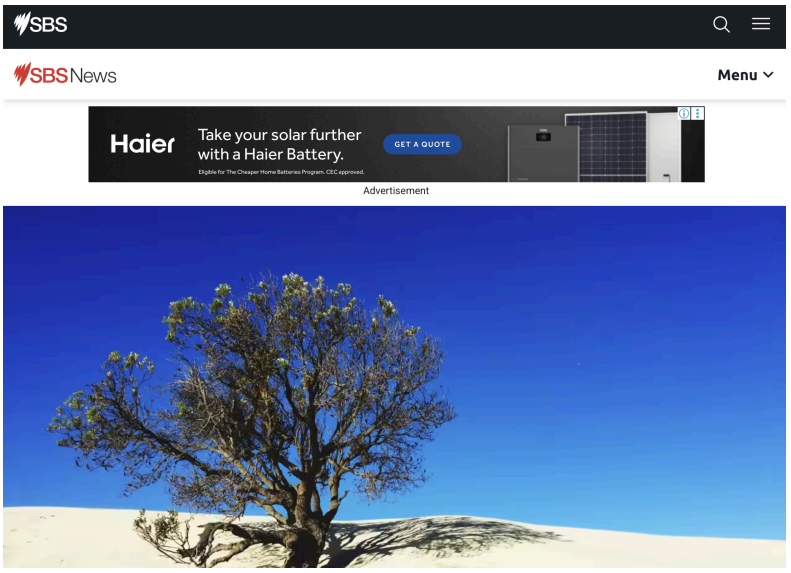
- **Given** I click **“Learn More”** (or a similar link),
- **when** the section opens,
- **then** I see a short, clear message that introduces how plants are affected by the changing climate.

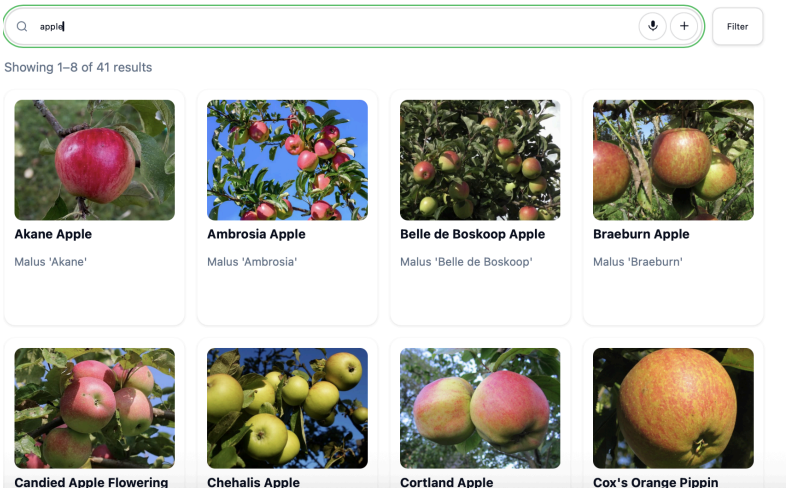
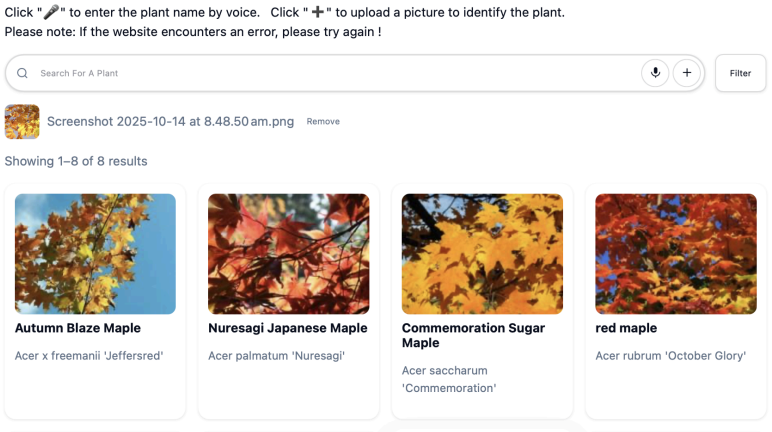
Developer Guidelines

- Keep each message brief and easy to understand; focus on one idea per message.
- Let people control the pace: simple Next, Previous, and Close actions that work on both phone and desktop.
- Make the Learn More link easy to find again so visitors can return whenever they need a reminder.
- Use plain language that connects everyday gardening actions to climate in a practical way.

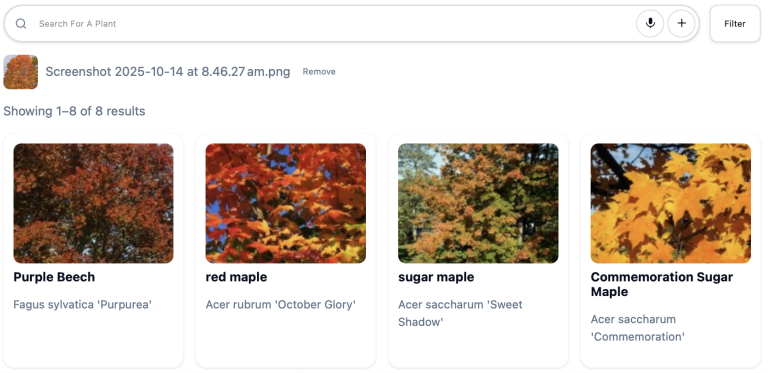
Information section showing relevant articles



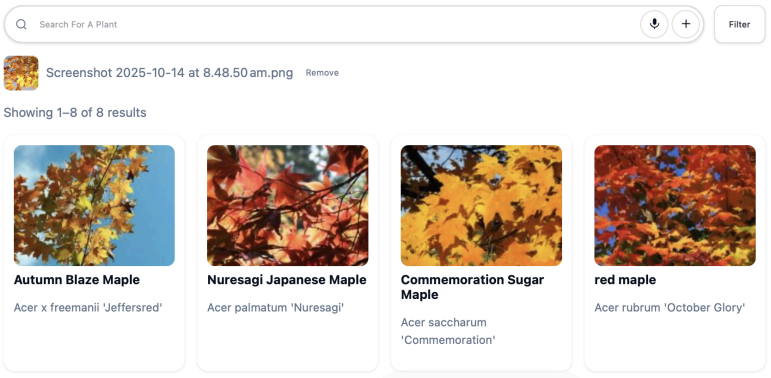
			Once clicked, it will redirect us to the actual article to read more on the information 	
2.	**Explore plant information quickly and learn more** *As a home gardener, I want to easily explore plant information so that I can learn about garden plants and understand their characteristics.* Epic Description: This epic lets people discover plants by typing, speaking, or uploading a photo, and then open a clear details page. The goal is fast discovery with simple next steps, using the same style of results no matter how the search starts.	2.1.	**Three simple ways to search for plant easily** *As a user, I want to see a clear introduction on the homepage, so that I immediately understand what the platform offers.* **Scenario** On the **Search** page, a user types “lavender” and presses **Search** (or Enter) and sees matching plants. Another taps the **Mic** button, says “rosemary,” and is shown the same style of results. A third clicks **Upload photo**, selects **rosemary.jpg**, and gets likely matches. All three see names and small images in a consistent list.	**Through text search** ● **Given** I am on the search page, ● **when** I type the name of a plant and press **Search** (or Enter), ● **then** I see a list of matching plants. **Through audio/microphone** ● **Given** I am on the search page, ● **when** I tap the **microphone** button and say a plant name, ● **then** I see a list of matching plants **Through image** ● **Given** I am on the search page, ● **when** I upload a valid photo of a plant, ● **then** I see a list of possible plant matches. **Developer Guidelines** ● Keep the three search methods easy to discover and use; all should feel equal. ● Show results in the **same layout** for text, voice, and photo so people don’t

				have to relearn the page. • Give clear, friendly messages when input isn't understood or a photo isn't suitable, and guide the person to try another method. • Keep any waiting time obvious and short with simple progress cues (for example, "Searching…"). • Avoid overwhelming the person—start with a tidy list and let them choose one result to go deeper.
			View of search using audio and text search  Uploading image is shown to be working:  	
		2.2.	**View plant details** *As an urban gardener, I want to view details of a plant I found, so that I can learn more about its characteristics.*	**Look at the selected plants information** • **Given** I see a list of possible matches, • **when** I click one, • **then** I am redirected to the **plant detail page**

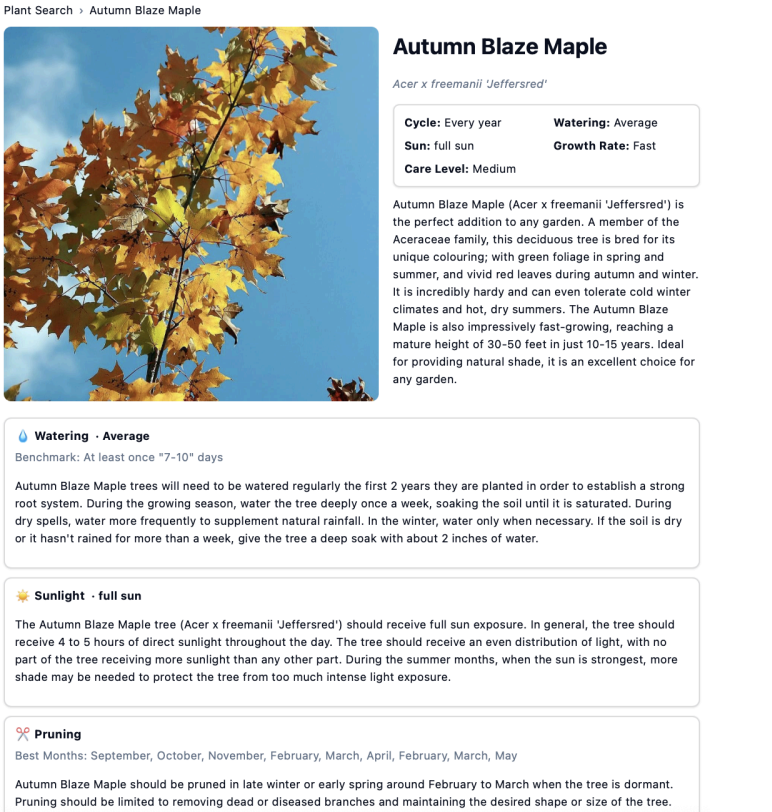
			Scenario From a list of results, a user chooses **Lavender (Lavandula)**. The **Plant details** page opens and shows the plant name, a picture (or placeholder), and key information such as light, water, and soil needs. If the user goes back, they can pick another result	containing plant information **Found plants related to the search** ● **Given** I see a list of results, ● **when** the system displays them, ● **then** I should see plants that **closely match my input** **Developer Guidelines** ● Make the details page easy to read: plant name at the top, key facts first, helpful picture if available. ● Keep names and pictures in the details page consistent with the list so people trust what they clicked. ● Provide a simple way to go back to the results and continue exploring without losing their place. ● Use plain, garden-friendly language so information is clear to non-experts.
			Using 2 images of maple:   Both give us results of maple, which is consistent. They also give a few matches that are closely related. Hence why although both are maple, the search comes a bit different while still relating to the maple family.	



Click " 🎤 " to enter the plant name by voice. Click " ➕ " to upload a picture to identify the plant.
Please note: If the website encounters an error, please try again !



When any of it is clicked, it shows a clear information of that plant



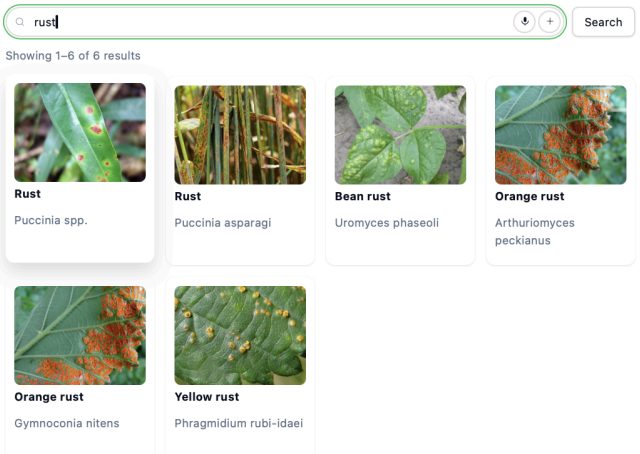
<table>
<tr>
<td>3.</td>
<td>Identify and understand plant disease

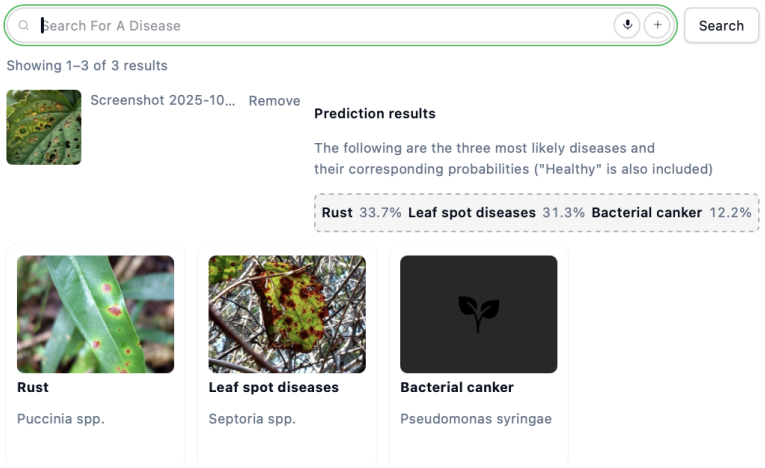
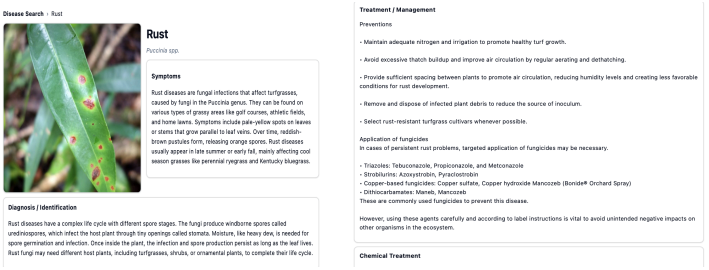
As a gardener, I want tools to help me identify and understand plant diseases, so that I can take proper actions to maintain plant health.

Epic Description
This epic helps people look up plant diseases by name or photo, then open a clear disease details view that explains symptoms and practical next steps.</td>
<td>3.1.</td>
<td>Search plant diseases
As a gardener, I want to search for plant diseases, so that I can find the relevant ones easily.

Scenario
On the Disease Tools page, a gardener types “rust” into the search box and sees matching diseases in a short list. Another gardener doesn’t know the name, so they click Upload photo, select a leaf picture, and the page returns likely disease matches they can choose from.</td>
<td>Type to search plant disease
● Given I know the disease name,
● when I type it into the search box,
● then I see a list of matching plant diseases

Upload image to search plant disease
● Given I don’t know the name,
● when I upload a plant photo,
● then I see the most likely disease of the plant I uploaded

Developer Guidelines
● Keep both entry points easy to find (type or upload) so gardeners can choose what suits them.
● Show results in a consistent list regardless of method, so people don’t have to relearn the page.
● Offer clear, friendly messages when nothing matches the search.
● Keep the flow quick and simple; let people pick a result and move on to details without distractions.</td>
</tr>
<tr>
<td colspan="3"></td>
<td colspan="2">When the keyword rust is used, it shows different type of rust disease
 
</td>
</tr>
</table>

			When uploading an image, a diagnosis would be chosen based on the most similar output to the image uploaded. 
		3.2	**View plant disease details** *As a gardener, I want to view plant disease details, so that I can understand its symptoms and possible solutions.* **Scenario** From the results list, the gardener selects Rust. A Disease details page opens showing a short description, symptoms to look for, and possible solutions such as care steps and prevention ideas. The gardener can return to the list to compare another result **Open a clear, useful details page** ● **Given** I see a list of possible disease matches, ● **when** I select one, ● **then** I am taken to a **disease detail page** that shows **symptoms** and **possible solutions**. **Developer Guidelines** ● Make the details page easy to read: name first, symptoms next, solutions close by. ● Use plain, practical language so gardeners know what to check and what to try. ● Keep names and any images consistent with the search results, and make it simple to go back and compare other matches.
			When clicked on the disease page (Rust), it shows the details of the symptoms and how to manage them 

<table>
<tr><td rowspan="2">4.</td><td rowspan="2">Location-based plant recommendations

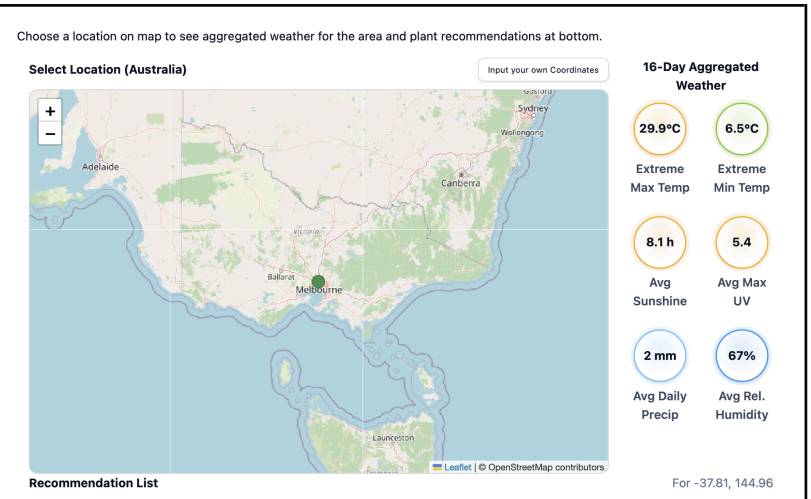
As a gardener, I want to get plant recommendations based on my location's weather conditions, so that I can choose plants that are more suitable for my environment.

Epic Description
This epic helps people click a spot on the map, see a simple weather outlook, and get plants that fit those conditions. It also lets them open a clear care guide so they can grow the recommended plants with confidence.</td><td rowspan="2">4.1.</td><td>Get recommendation from the map
As a gardener, I want to click on a location on the map and get plant recommendations, so that I know which plants can grow well under my local weather conditions.

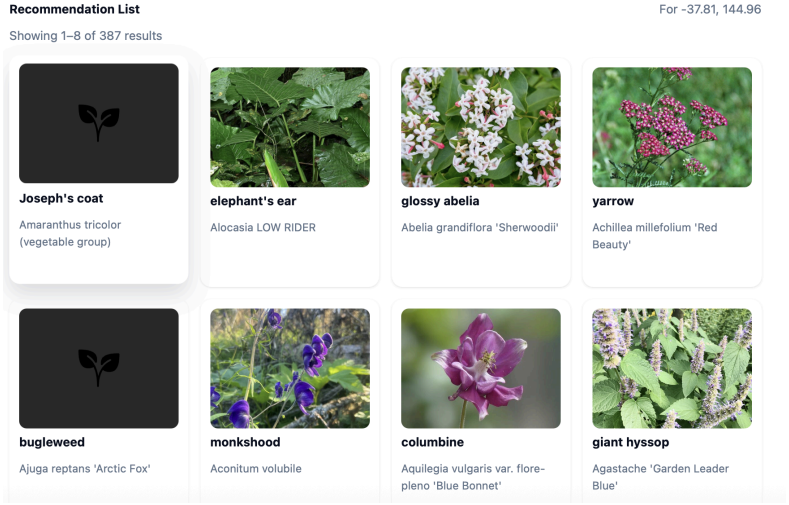
Scenario
On the Recommendations page, a gardener clicks a point on the map near their suburb. The page shows a short 16-day weather summary for that location (for example, typical highs/lows and rainfall). Under the summary, a row of recommended plant cards appears that match those conditions.</td><td>Shows details on weather prediction on selected location
● Given the user clicks on a coordinate on the map,
● when the system queries the weather service,
● then it displays an aggregated summary of the next 16 days' weather for that location.

Turn the predictions of the location into a recommendation for plants to grow
● Given the aggregated weather summary is retrieved,
● when the system compares the weather with plant growth conditions,
● then it recommends plants whose growing requirements match the forecasted weather.

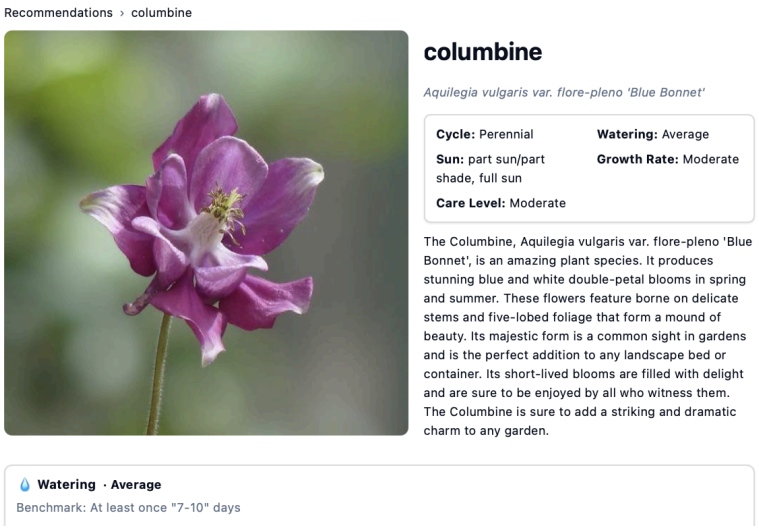
Developer Guidelines
● Keep the weather summary short and clear (clear numbers and words, not raw data dumps).
● Make the recommendations easy to follow: show plant name, a small image, and a simple reason it fits (for example, "handles low rainfall").
● Allow the gardener to pick another spot on the map and see the summary and recommendations update quickly.
● Use wording that helps decision-making (for example, "good fit for the next two weeks"), not technical terms.</td></tr>
<tr><td colspan="2">Shows the pin location and the weather prediction across that location</td></tr>
</table>

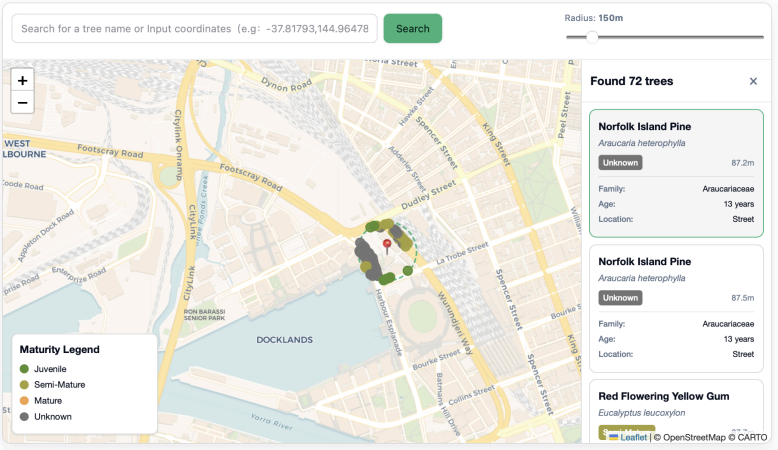
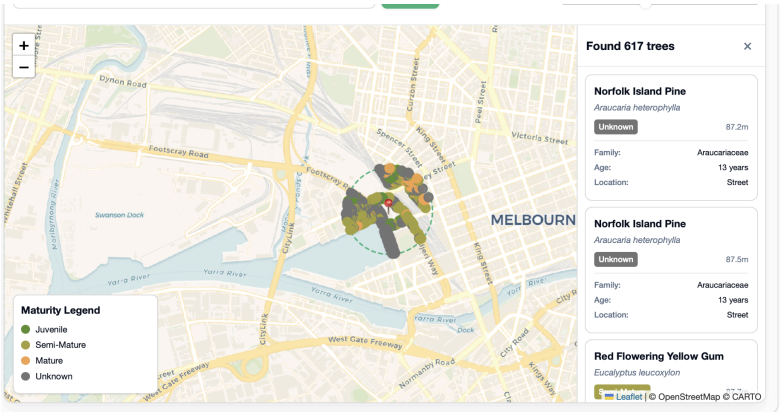


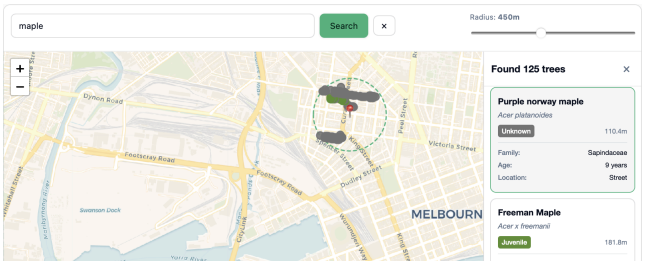
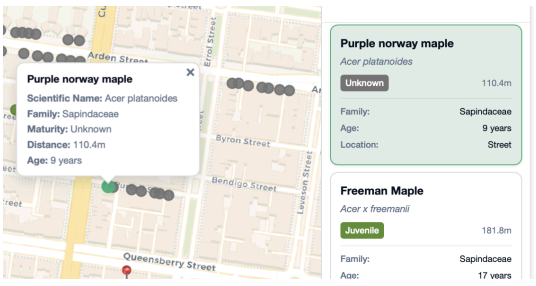
Provide recommendation list of plants that user can grow based on the location they live in



		4.2.	**Open plant details and care guides** *As a gardener, I want to view details and care guides for recommended plants, so that I know how to grow them properly.* **Scenario** After viewing the recommended plants, the gardener clicks the card for Rose. A plant details page opens that shows a description, basic characteristics (light, water, soil), and a	**From a card to a clear care guide** ● **Given** the user wants to view details and care guides for recommended plants, ● **when** the user clicks a **recommended plant card**, ● **then** the system redirects to the **plant's description, characteristics, and care guide**. **Developer Guidelines** ● Make the details page easy to follow: name first, key facts next, then a step-by-step care guide.

			care guide with plain steps.	• Keep the look and wording consistent with what was shown on the recommendation card so trust is maintained. • Provide a simple way to go back to the recommendations and pick another plant without losing place. • Write care steps in everyday language so new gardeners can apply them immediately.
			Shows the plants information that user clicked on 	
5.	**Explore nearby plants and trees** *As a gardener in Australia, I want to explore plants and trees around me so that I can learn more about my environment and appreciate biodiversity.* **Epic Description** This epic lets people open a map, view nearby species	5.1.	**Explore a map of nearby plants and trees** *As a gardener in Australia, I want to explore a map of nearby plants and trees, so that I can learn about the species in my area.* **Scenario** A gardener opens the map page and chooses a location (for example, by allowing the browser to use their current location or by clicking a spot). The map updates to show plant and tree	**Understand local species at a glance surrounding you** • **Given** I open the map view • **When** the website receives the location I chose • **Then** I should see a visualisation of plant density or species in my area, with a legend that explains the markers or colours **Developer Guidelines** • Keep the map easy to read at common zoom levels; avoid clutter and make the legend clear • Make it simple to change

	and density, and search for a plant by name to see where it occurs locally. It aims to build awareness of local biodiversity and encourage care for the environment.		presence in that area using clear markers or shading, along with a simple legend explaining what the colours mean.	location (allow current location or a click/tap on the map) and refresh the view quickly • Use plain labels and short descriptions so people can understand what the map shows without expert knowledge • Encourage exploration by making it easy to move the map and discover nearby areas
			Tree is shown within the radius of the pinned (chosen) location  Using the slider  We create a new radius around the map and detect the trees around the new radius 	
		5.2.	**Search for plants by name to see where**	**Find where certain plant occurs near me**

			they exist locally As a gardener in Australia, I want to search for plants by name, so that I can find where they exist locally. **Scenario** On the same map page, the gardener types the name of a plant (for example, “Wattle”) into a search box. The map highlights regions or coordinates where the plant is present and the legend updates to show what the highlight means. The gardener can clear the search to return to the general view.	• **Given** I search for an urban plant • **When** the website gets the plant name • **Then** it should highlight the regions or coordinates where the plant is present on the map and explain the highlight in the legend **Developer Guidelines** • Make the search bar easy to find and use; show clear matches and avoid overwhelming the user • Keep highlights obvious but not confusing; ensure the legend tells the user what the highlight means • Allow clearing the search and returning to the normal map view without losing place • Use simple, location-friendly wording so people understand what is shown and how to explore further •
			Searching for ‘Maple’ gives us any maple tree grown around the radius  When any selected plant is clicked, it shows the detail of that plant  	

<table>
<tr>
<td>6.</td>
<td>Threatened plants map
As a concerned Australian gardener, I want to view the location of threatened plants across Australian states, so that I can understand where they are distributed and learn more about them.

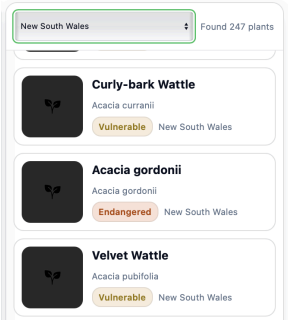
Epic Description
This epic lets people open a dedicated map of threatened plants, filter by state, see markers on the map with a matching list of plant cards, and open details to learn more.</td>
<td>6.1.</td>
<td>Filter threatened plants by state

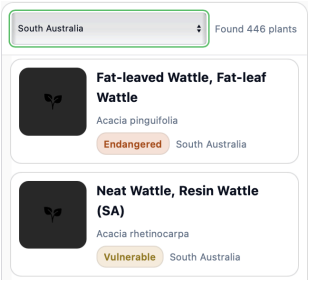
As a user, I want to filter threatened plants by state, so that I can focus on the plants relevant to my region.

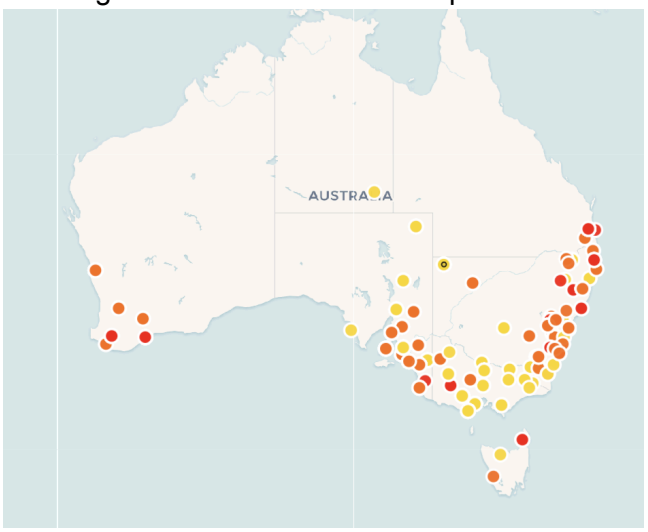
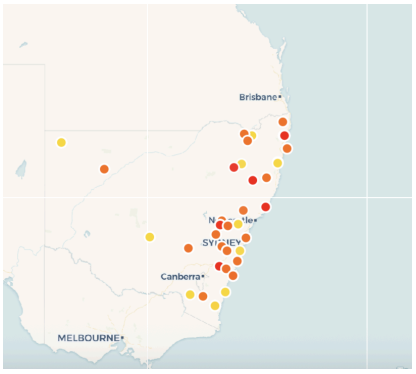
Scenario

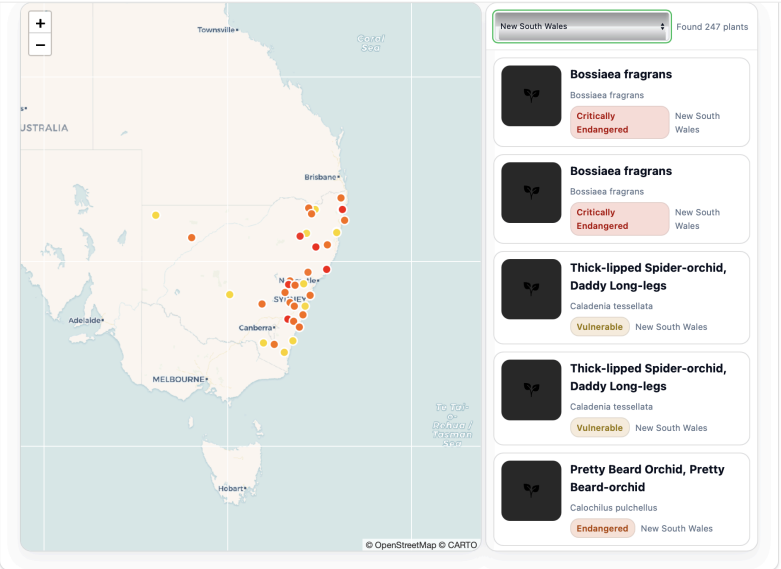
On the threatened plants map page, a user opens a simple state filter (for example, a dropdown) and picks Victoria. The map updates to show only the threatened plants in Victoria. The list beside the map updates to match.</td>
<td>Showing threatened plants on certain state
● Given I am on the threatened plants map view
● When I select a state filter
● Then the map should only display threatened plants located in that state

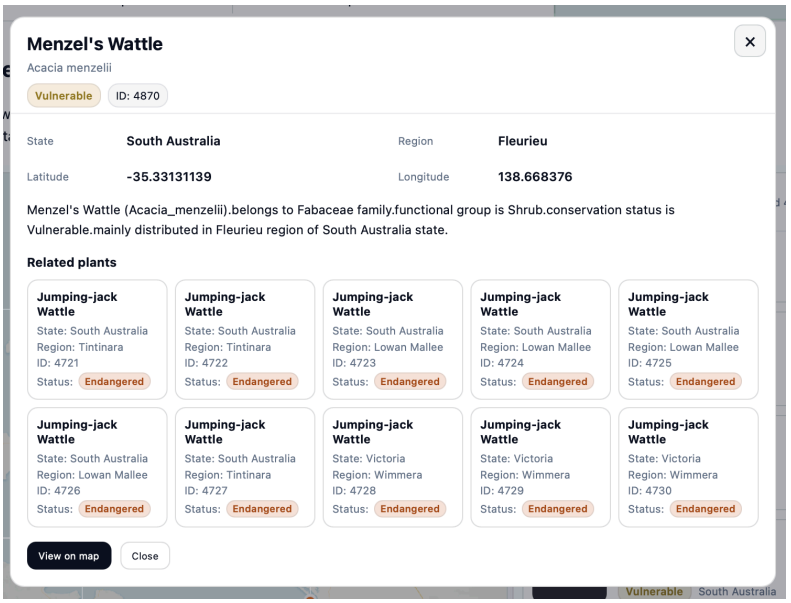
Showing the large scale of plants displayed on map
● Given I apply a state filter
● When the filter is active
● Then the card list should also update to only show plants from that state

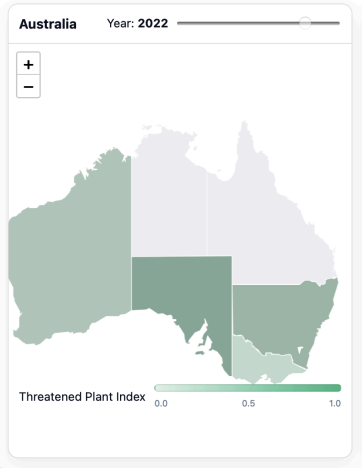
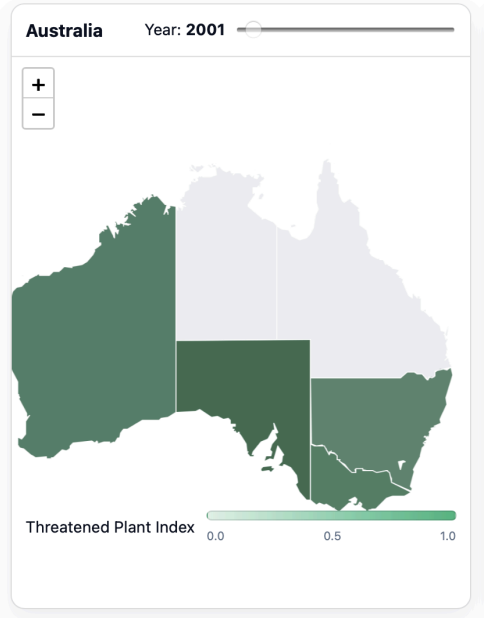
Develop Guidelines
● Keep the state filter easy to find and change at any time
● Make the map and card list move together so people always see the same set
● Use plain state names and simple wording so the purpose of the filter is obvious</td>
</tr>
<tr>
<td colspan="3"></td>
<td colspan="2">The threatened plant shown with ‘NSW’ filter
 

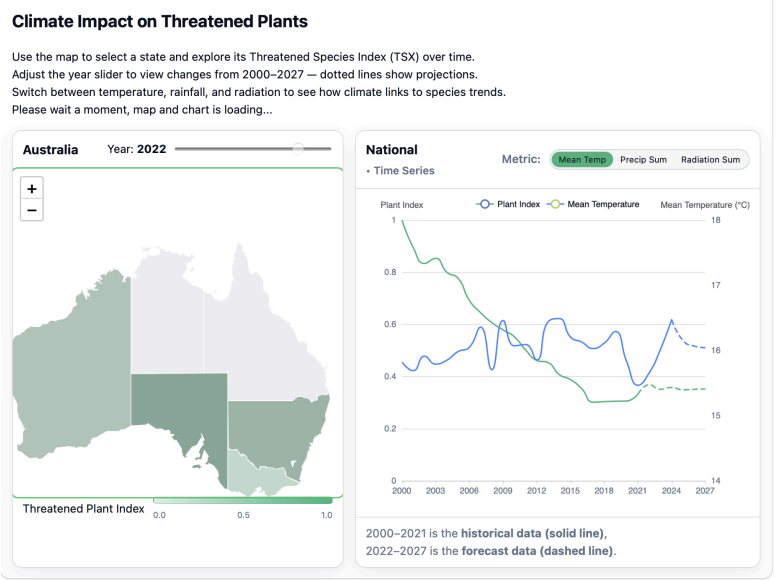
The threatened plant shown with ‘SA’ filter
 
</td>
</tr>
</table>

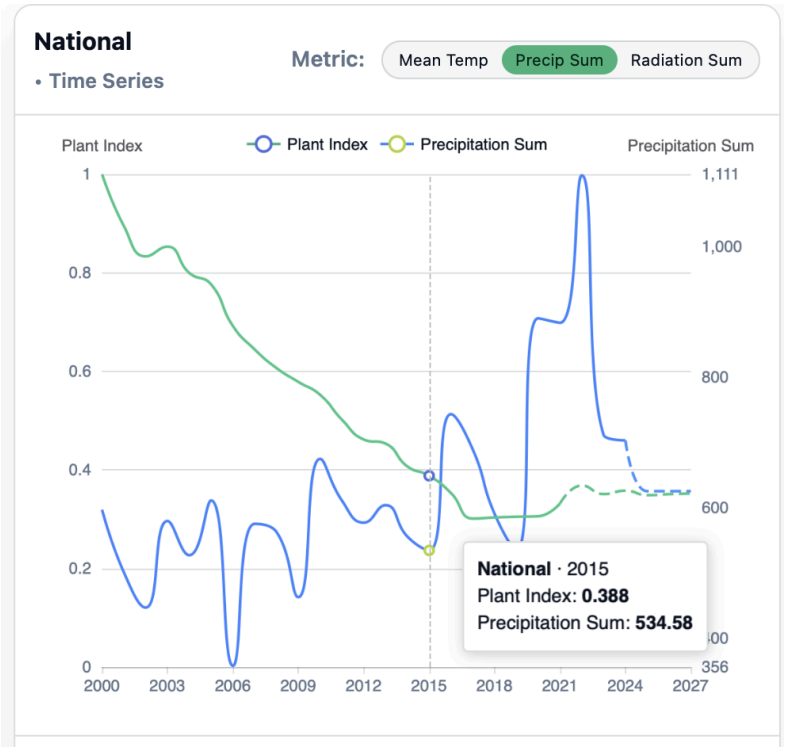
		6.2.	**Show threatened plants on an interactive map** *As a user, I want to see threatened plants displayed on an interactive map, so that I can visualise their distribution.* **Scenario** When the page loads, the map appears and plant markers fade in as the data finishes loading. The user can pan and zoom to explore.	**See markers that match real locations** ● Given I am on the threatened plants map view ● When the data is loaded ● Then I should see plant markers on the map corresponding to their location **Developer Guidelines** ● Make the first view useful at a national level and allow smooth zooming ● Keep markers readable and avoid clutter at common zoom levels ● Provide a short loading cue so users know when the data is ready
			Showing the distribution on the map of all Australia  When filtered to NSW 	

		6.3.	**See a list of plant cards next to the map** *As a user, I want to see plant cards listed alongside the map, so that I can browse and select plants easily.* **Scenario** On the right side of the screen, a scrollable list shows plant cards that match what is shown on the map. As the user filters or moves the map, the list stays in sync.	**Browse plants alongside the map** ● Given threatened plants are displayed on the map ● When I look to the right side of the screen ● Then I should see a scrollable list of plant cards matching the displayed plants **Developer Guidelines** ● Keep the list simple and fast to scan (name and a small image) ● Ensure the list stays in sync with the map and any active filter ● Allow easy scrolling without losing view of the map
			Plants list is shown next to the map for better representation and view  	
		6.4.	**Open the threatened plant's details from a card** As a user, I want to click on a plant card and see that specific plant information, so	**Go from card to details** ● Given I am viewing the card list ● When I click on a card ● Then I would see a pop up window that shows the information of that plant **Developer Guidelines** ● Make the card click clearly

			that I can learn more about it. **Scenario** From the card list, the user clicks a plant card and is taken to that plant's detail page with its description and information.	open the right details page ● Keep names, images, and key facts consistent with what was shown on the card ● Provide a simple way to return to the map and continue browsing
			Showing the plant details and the similar plant surrounding it 	
7.	**Climate trends and impact** As a gardener, I want to view a choropleth map that can interact from the past to the future, so that I can compare their spatial patterns across states and years. **Epic Description** This epic lets people switch a map between two views (plant index and		**Interactive Chloropleth Map** *As a gardener, I want to view a choropleth map that can interact from the past to the future, so that I can compare their spatial patterns across states and years.* **Scenario** On the climate view, a gardener sees a map of Australia with a year selector for Plant Index. They choose a year, tap the toggle, and the map recolors each state to	**Year Slider of interactive map** ● Given I am on the map view ● When I move the year slider ● Then the map should recolor each state according to the selected dataset for the chosen year **Developer Guidelines** ● Keep the toggle easy to find; use plain labels so people understand what is being shown ● Keep the legend short and clear so colors make sense at a glance ● Make changing the year or toggle feel instant so

	temperature) for a chosen year, then pick a state to see a simple trend chart over time. It focuses on clear comparisons that build climate awareness without technical jargon.		match the selected data. A short legend explains what the colors mean.	comparisons are easy
			Map view of plant index in year 2022  Changing the year to 2001 will update the map color based on the plant index  	
		7.2.	**View historical trends for a selected state** *As a user, I want to see historical trends for a selected state, so that I can explore how its plant index and temperature have changed over time.*	**A simple chart that compares two lines over time** ● Given I select a state from the map or filter ● When I load its chart ● Then I should see a line graph showing both plant index and temperature (or other weather variables) across years. **Developer Guidelines**

			Scenario The gardener clicks a state on the map (or chooses it from a simple list). A small panel opens with a line chart showing the state's Plant Index and Temperature across years on the same axes so they can compare trends over time.	• Keep the chart easy to read with clear labels and a short caption that explains the takeaway • Use consistent units and wording with the map so people connect the two views • Let people change the state quickly and keep the chart updated without starting over
			The line chart when min temp is chosen  When switched to precipitation sum (together with the details when the cursor touches)	

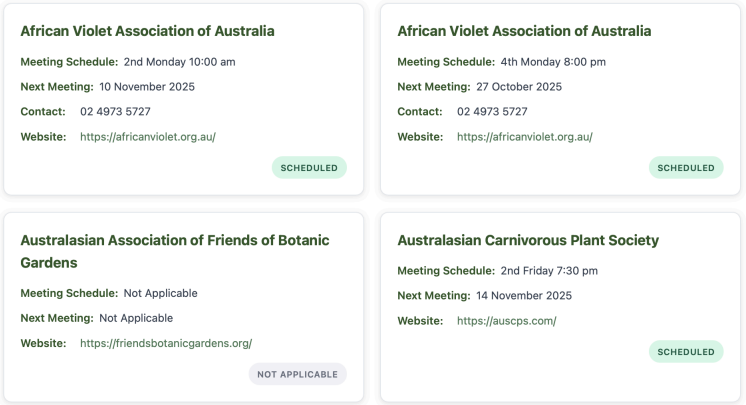
<table>
<tr><td></td><td></td><td></td><td colspan="2"> 
</td></tr>
<tr><td>8.</td><td>Explore plant clubs
As a gardening enthusiast, I want to explore plant clubs, so that I can connect with the gardening community.

Epic Description
This epic lets people browse a simple list of clubs with names, meeting schedules, and contact details (when available), filter by region/state, and see who is meeting today so they can join in.</td><td>8.1</td><td>View club listings

As a user, I want to view a list of plant clubs with their names, contact details, and meeting schedules, so that I can learn about available clubs.

Scenario
On the community page, a gardener scrolls through a clean list of plant clubs. Each club shows its name, a short note about its recurring meeting schedule, and contact details if they are provided.</td><td>Know what clubs exist and how to reach them
• Given I am on the community page
• When I view the club listings
• Then I should see the club name, recurring meeting schedule, and contact information

Developer Guidelines
• Keep each club entry short and easy to scan (name first, then schedule, then contact)
• Show contact details only when provided; avoid empty labels
• Use plain wording for schedules (e.g., “First Saturday each month, 10am”)
• Make the list comfortable on phone and desktop (no tiny tap targets)</td></tr>
<tr><td colspan="3"></td><td colspan="2">Each card shows important information of every gardening club in Australia, with the detail that user might potentially need</td></tr>
</table>



8.2.

Filter clubs by region/state

As a user, I want to filter clubs by region/state, so that I can focus on clubs in my area.

Scenario
On the same page, the gardener opens a simple region/state filter and picks their area. The list refreshes to show only clubs from that region. They clear the filter to see all clubs again

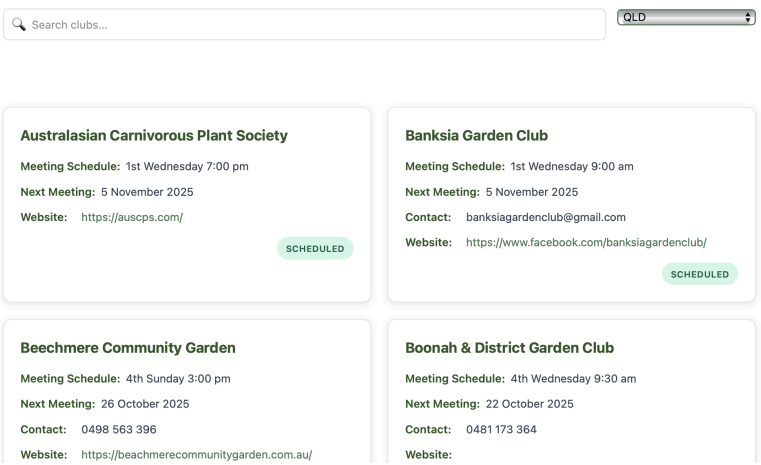
Focus on clubs near me

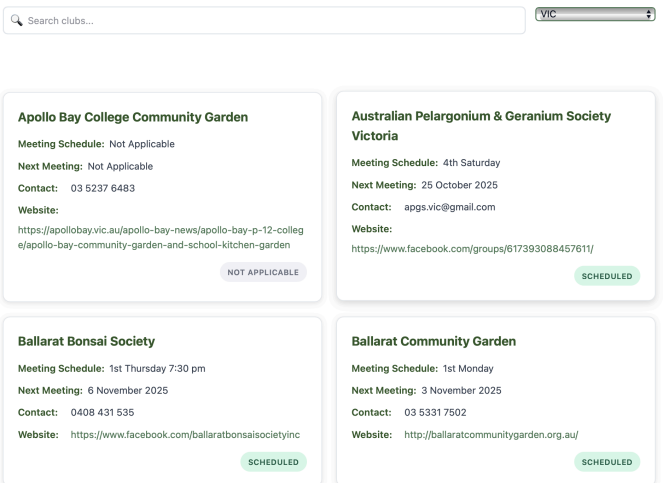
- **Given** I am on the community page
- **When** I select a region/state from the filter
- **Then** I should only see clubs located in that region

Developer Guidelines

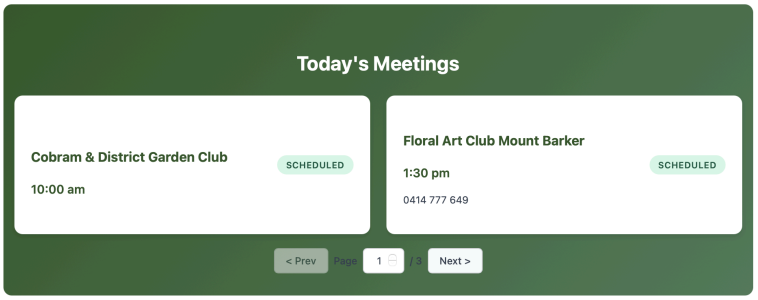
- Place the filter where it is easy to find and change
- Make it clear when a filter is active and simple to clear it
- Keep the list and filter in sync so results always match the selection

Filtering `QLD` and `VIC` state would yield different club, based on their location





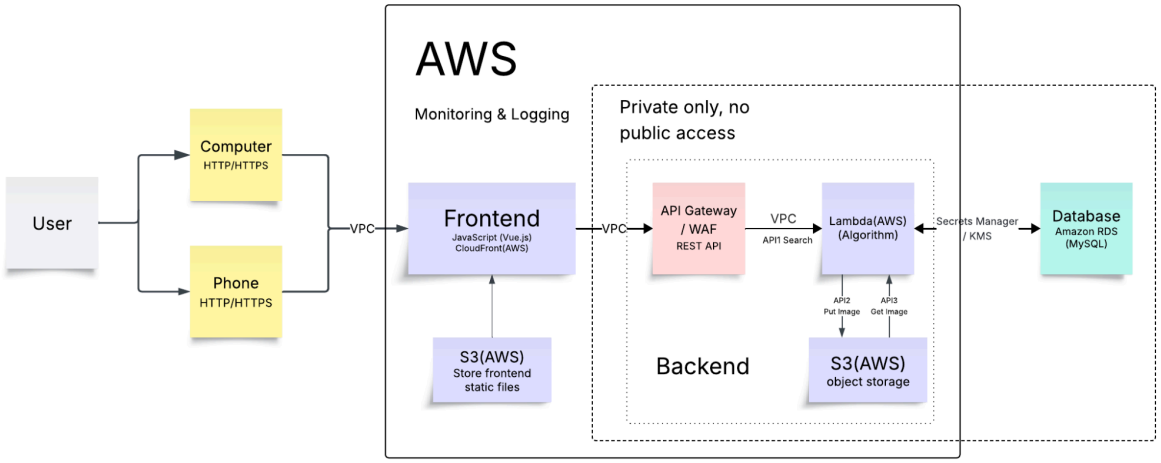
		8.3.	**See who have meeting today** As a user, I want to see which clubs are meeting schedules, so that I can join the one that suits my time **Scenario** The page shows a small “Today’s Meetings” panel. If any clubs have a recurring schedule that matches today, those clubs appear in the panel with their meeting time. If none match, the panel shows a friendly message instead.	**Today’s clubs at glance** ● **Given** I am opening the community page to see clubs ● **When** a club’s recurring schedule matches today ● **Then** the system should list that club under “Today’s Meetings” **No meetings view** ● **Given** no clubs match today’s pattern ● **When** the page loads ● **Then** the “Today’s Meetings” panel shows “No meetings today” **Developer Guidelines** ● Keep the “Today’s Meetings” panel visible and simple; don’t overcrowd it ● Use the local date so “today” matches the user’s region ● Make each listed club easy to recognise and find in the main list if the user wants more details.
			A calendar view that will show today’s meeting schedule, and which club is doing the meeting		

			 When no club has their meeting scheduled today, then it will show a “No meeting today” message.

4.2. Operating Environment

Category	Description	Notes for Maintainers
Operating System	macOS 12+ / Windows 10+/ Ubuntu 22.04+	Any of these are fine for local builds and tooling
Desktop Browsers	Chrome, Edge, Firefox, Safari (latest 2 versions)	Use these for testing UI changes before release
Mobile Browsers	iOS Safari, Android Chrome (latest)	Sanity-check key pages (Home, Search, Details, Map)
Network Connection	Stable broadband with HTTPS access	Required to pull dependencies, query APIs, and view iteration/prod sites.
Technical Requirement	Node.js 18 LTS + npm; Git 2.40+; MySQL client 8.0+; AWS CLI v2; (Optional) Python 3.10+	Build/run Vue (Vite) locally; version control; run DB scripts; inspect logs/deploy via CLI; Python only if touching serverless logic.
Hardware Requirements	At least 8 GB RAM, 20 GB free disk, 2 CPU cores	Enough for local builds, package installs, and test runs. RAM and CPU cores are especially important in running machine learning model
Security Requirements	MFA on GitHub and AWS; no secrets in code; use read-only DB creds for local work	Replace secrets with environment variables; rotate keys as per sponsor policy.
Environment Variables & Secrets	API base URLs, DB connection, Weather API key, any image/vision key	Store as environment vars or in AWS Secrets Manager; example placeholders

5. System Architecture

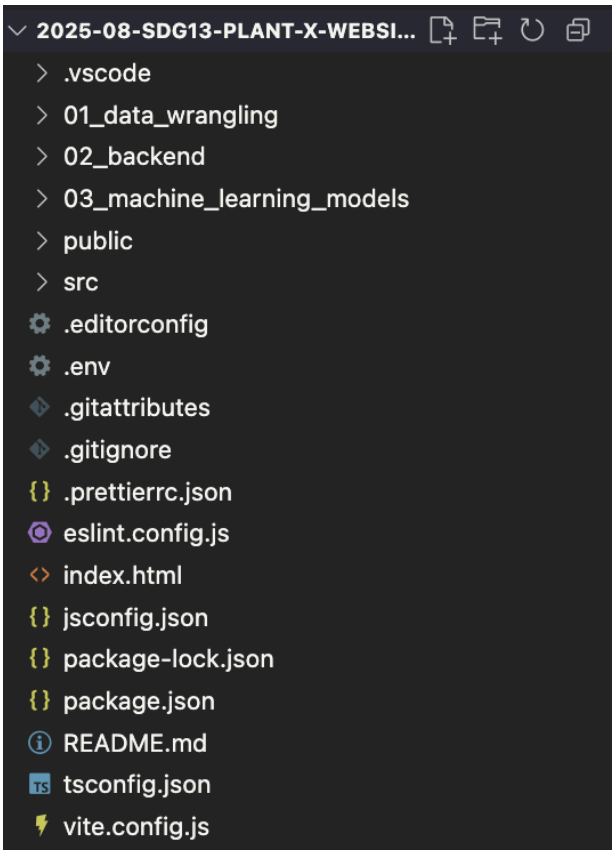


For more details, please refer to: [System Architecture Explained]

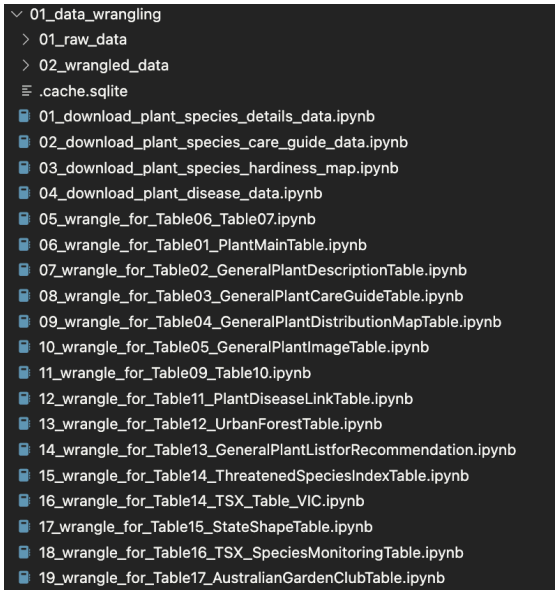
6. Maintenance Activities

6.1. Code

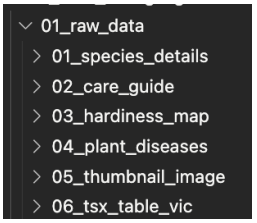
We have organised our project folder into a readable file structure for easier navigation.



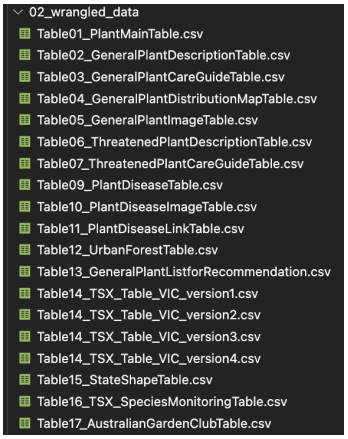
1. **01_data_wrangling:** consist of some jupyter notebook files that handle wrangling each dataset.



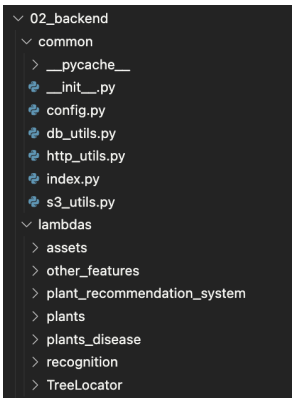
1.1. 01_raw_data: stores the collection of the open data collected prior to wrangling

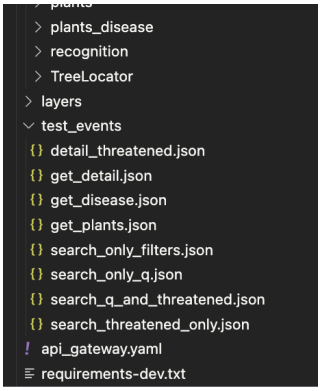


1.2. 02_wrangled_data: is just the cleaned data once we apply the raw data into the wrangled data

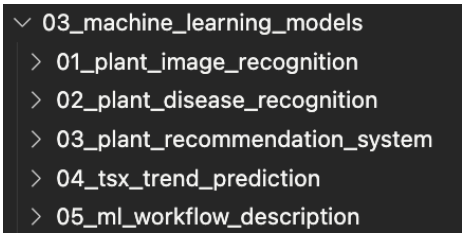


2. 02_backend: stores the working code for our project website





3. 03_machine_learning_models: each folder represents one feature ML model. 05_ml_workflow_description is specifically a documentation explaining how it works.



6.2. Data

Names	Physical access	Frequency of source updates	Frequency of iteration system updates	Granularity	Copyright/licensing details
Prenual Plant API: Species Details	API (JSON)	Periodic	On demand	High	Prenual API Terms and Use
Prenual Plant API: Care Guides	API (JSON)	Periodic	On demand	High	Prenual API Terms and Use
Prenual Plant API: Distribution/Hardiness Map	API (HTML)	Periodic	On demand	Medium	Prenual API Terms and Use
City of Melbourne: Threatened Plant Living Collection Plan	CSV	Periodic	Annually	High	Melbourne Open Data licence

City of Melbourne: Urban Forest	CSV	Periodic	Annually	High	Melbourne Open Data licence
Open-Meteo: Historical & Forecast Weather	API (JSON)	Real-time	On demand	Medium	Open-Meteo terms
Threatened Species (TSX)	CSV	Annual	On demand	Yearly index	TSX terms
Kaggle - Plant Disease Dataset	Files (JPG)	Static	As needed	High (labelled image)	Kaggle dataset terms
Australian Garden Club Directory	CSV	Ad-hoc	As needed	Organisation level	Public listings
ASGS State/Territory Digital Boundaries	SHP / GeoJSON	Periodic (ABS releases)	As needed	Region-level	ABS license (open)

For more details on the data, please refer to: Data Management Plan

6.3. Test

6.3.1. Test Plan

This Test Plan describes how we will test the AI Climate Project to make sure the system works correctly, reliably, and is easy for users to interact with. The main goal of testing is to confirm that the platform gives accurate and useful gardening recommendations based on plant and climate data, and that all parts of the system, from the data pipeline to the frontend interface, work smoothly together.

Objectives

The main objectives of our testing are to:

- Check that the data pipeline retrieves, cleans, and stores data correctly.
- Make sure the backend APIs return accurate, consistent, and complete responses.
- Verify that the frontend interface is responsive, easy to use, and accessible on different devices.
- Confirm that all components (frontend, backend, database, and AI system) integrate properly.
- Assess the overall reliability and quality of the platform under different use conditions.

In Scope

The following areas will be covered by our testing:

- **Unit Tests:** Focus on individual modules or functions (e.g., data cleaning scripts, API fetch functions) to ensure they work correctly on their own.

- **Integration Tests:** Check how different modules interact with each other (e.g., backend connecting with database, frontend communicating with backend).
- **System Tests:** Validate the complete end-to-end workflow, from data retrieval to final output shown to the user.
- **Usability Tests:** Focus on the user experience, including how easily users can navigate the system and understand the gardening recommendations.
- **Reliability Tests:** Ensure the system provides consistent and stable results, even when tested multiple times with the same data.

Out of Scope

The following areas are currently not part of the testing scope, as the project is still in development and has not yet been planned for public release. These may be included in future phases:

- Large-scale performance benchmarking (outside project scope).
- Long-term stress testing under heavy traffic or continuous usage.

Test Approach

Our team follows an **Agile and test-driven development** approach, where testing is planned and conducted throughout the development cycle. We write tests early, review them regularly, and run them frequently as features are built and updated.

Key methods used:

- **Unit Testing:** Ensures that each individual piece of code (like data transformation or API calls) works as intended.
- **Integration Testing:** Verifies that data flows correctly between components (e.g., data moves from API → database → frontend without errors).
- **System Testing:** Confirms the entire system runs smoothly as one complete product.
- **Functional Testing:** Checks that each feature meets the original requirements and behaves correctly for users.
- **Performance Testing:** Measures how the system performs in terms of speed, load time, and responsiveness.
- **Usability Testing:** Gathers feedback from testers and users to improve design, navigation, and overall user experience.
- **Reliability Testing:** Runs repeated tests to check that results remain accurate and consistent

Tools

We use the following tools for testing:

- **MySQL Workbench:** To validate data stored in the database and check query results.
- **Postman:** For testing API endpoints, input/output formats, and error handling.
- **Browser Developer Tools:** For checking frontend performance, UI responsiveness, and console errors.
- **GitHub Issues:** To record bugs, track fixes, and manage version history

- Methodology: Agile test-driven development.
- Pair Programming: All tests are written/reviewed in pairs for accuracy.
- Tools:
    - MySQL Workbench: for database validation queries.
    - Postman: for API endpoint testing.
    - Browser DevTools: for frontend testing.
    - GitHub Issues: for bug tracking and version history tracking.

Testing

- Backend: Python + Flask, MySQL.
- Frontend: Vue.js, run locally and in browser.
- Data: Cleaned JSON/CSV datasets and stored them as relational database MySQL tables.

Testing Methodologies

We applied a layered testing approach:

- **Unit Tests:** Test small, isolated functions like data cleaning or API fetching.
- **Integration Tests:** Check whether data is correctly processed and stored between the backend and the database.
- **System Tests:** Run the full process from start to finish — for example, uploading plant data, generating recommendations, and displaying results.
- **Usability Tests:** Collect feedback on the user experience, focusing on changes made after feedback (e.g., improved search bar, “no results” messages, accordion-style information sections).
- **Reliability Tests:** Make sure the recommendation and disease detection results stay consistent across repeated queries.
- **Performance Tests:** Check how quickly the system responds and whether it meets acceptable load times for maps, API calls, and AI recommendations.
- **Security and Error Handling Tests:** Test how the system responds to invalid inputs, failed API calls, or missing data without crashing.

Roles and Responsibilities

Testing Stage	**Responsible Role**	**Description**
Unit Testing	Team Developer	Develop and run unit tests to check individual modules and functions.
Code Review	Team Developer	Review code for quality, correctness, and adherence to standards.
Integration Testing	Team Tester	Test the connection between modules (e.g., API to database).
System Testing	Team Tester	Conduct full end-to-end testing across the entire system.
Alpha Testing	Team Tester	Perform internal testing before the product

		is shared with users.
Beta Testing	End Users	Provide feedback based on real-world usage scenarios.
Functional Testing	Team Tester	Verify that each function meets user requirements and behaves as expected.
Performance Testing	Team Tester	Measure performance, load time, and system responsiveness.
Usability Testing	Team Tester	Evaluate user experience, navigation flow, and interface clarity.

6.3.2. Test Cases

This document lists test cases to verify that the project functions are working as intended. Each test case includes the test purpose, input, and the expected output.

The tests cover:

- Frontend UI/UX – navigation, data display, user interactions.
- Backend/API – data retrieval, API response correctness, integration with database.
- Data Pipeline – data cleaning, validation, wrangling, storage.
- AI/Recommendation System – correctness and consistency of eco-friendly gardening suggestions.
- Non-Functional Requirements – focus on how well the system performs rather than what it does

Frontend Test

ID	Description	Input	Expected Output	Notes
TC-FE-01	Check homepage loads correctly	Open app	Homepage displays header and main navigation properly	Test on multiple browser
TC-FE-02	Click “learn more”	Click button	Shows information page of the topic I want to learn about	Check responsiveness
TC-FE-03	Browse plant page	Click on plant page	Shows the plant description properly	Look at the design and readability
TC-FE-04	Search for plant with no results	Enter “xyzabc”	Displays “no search found” message	Added after mentor feedback
TC-FE-05	Accordion display of	Expand section	Only one accordion expands at a time,	Improves usability of long

	explanation		readable UI	text
TC-FE-06	Debounced slider update (radius filter)	Drag radius slider from 1km→10km continuously	Map re-renders once after drag ends (or at configured debounce), showing correct markers for 10km	Prevents “one step late” update; verify no stale state
TC-FE-07	Live slider update (no debounce)	Drag slider slowly step-by-step	Map updates in ≤300ms per step, with no lag for previous value	Use if you chose immediate updates; measure latency
TC-FE-08	Slider drag-cancel resilience	Start dragging → release outside track	No crash; value reverts to last committed; map state unchanged	Edge case often missed
TC-FE-09	Pin re-select refresh	Click pin A → change radius → click pin B	Results refresh to B with current radius; no A-residue markers	Guards “sticky” results
TC-FE-10	Image upload timeout error UX	Upload large image with simulated slow backend	After 30s (configurable) show friendly timeout toast + retry	Mirrors your “HTTP issue” scenario
TC-FE-11	Unsupported image type	Upload `.tiff` / `.webp` if not supported	Block upload; show “file type not supported” guidance	Prevents wasted requests
TC-FE-12	Max file size enforcement	Upload image > max MB	Client rejects; shows max size and tips to compress	Security + UX
TC-FE-13	Empty state for trends	Open Trends page with no data for suburb	Clear empty-state component with next steps (choose nearby suburb)	Polished UX
TC-FE-14	Accessibility: keyboard map controls	Tab to map controls; use arrows	Controls operable via keyboard; visible focus ring	WCAG 2.1 AA
TC-FE-15	Mobile layout (map + cards)	Phone viewport	Map uses height properly; cards scroll; no overflow	Catch mobile clipping
TC-FE-16	Persistent filters on nav	Set filters → navigate to Plant page →	Filters & radius persist; results identical pre-navigation	Prevents user rework

		back		
TC-FE-17	Loading placeholders (skeletons)	Open map on slow network	Skeletons display until data arrives; no layout shift	CLS hygiene

Backend Test (API & Database)

ID	Description	Input	Expected Output	Notes
TC-BE-01	Retrieve plant data	API request	JSON response with all required fields (converted from the relational database	Validate on input and output from schema
TC-BE-02	Handle missing data	Invalid plant ID	Null response handled in proper way (does not break)	Ensure no crash
TC-BE-03	Get disease detection results	Upload plant image	JSON response with classification + confidence	
TC-BE-04	Trend Index endpoint schema	GET `/api/tsx-index?suburb=…`	JSON with `{suburb, species_id, tsx_index, date}`; types correct	
TC-BE-05	Forecast endpoint happy path	GET `/api/forecast?species_id=…&h=12`	12 monthly points with `{date, forecast, lower, upper}`	From ARIMA/SARIMAX
TC-BE-06	Forecast bad params	`h=0` or `species_id` missing	`400` with helpful error JSON	Defensive programming
TC-BE-07	Caching headers	GET same forecast twice	`Cache-Control`/ETag present; 2nd call 304 if unchanged	Perf + best practice
TC-BE-08	Pagination for dense areas	GET `/api/trees?lat,lon,radius&page=2`	Deterministic page size; stable sort; no duplicates	Large result sets
TC-BE-09	Graceful upstream failure	Simulate weather API 500	Our API returns `502` with `upstream="open-meteo"` + fallback msg	Clear provenance
TC-BE-10	DB null safety	Query species with partial records	Missing optional fields defaulted; no 500s	Matches BE-02 intent

TC-BE-11	Rate limiting	50 hits/10s from same IP	Receive `429` with retry-after	Basic abuse control
TC-BE-12	Geo query correctness	Trees with radius 5km	Results all within 5km (Haversine check)	Correct spatial filter

Data Pipeline

ID	Description	Input	Expected Output	Notes
TC-DP-01	Validate API JSON structure	Raw JSON from API	All required keys exist, values of primary data is not null	Check on robustness on the script
TC-DP-02	Clean CSV dataset	Raw threatened species CSV	No duplicate, consistent format, and stored in MySQL	
TC-DP-03	Verify distribution map	HTML file	Ensure that HTML can be correctly stored in the database for web access	
TC-DP-04	Weather data testing	API JSON	Weather table updated correctly	Helps with the recommendation system
TC-DP-05	Join plant & weather dataset	Plant Care DB + Weather DB	Combined table helps improve query for recommendation	
TC-DP-06	TSX GAM aggregation reproducibility	Same raw TSX sample twice	Identical TSX Index after rerun (seeded where applicable)	Make runs deterministic
TC-DP-07	Climate join temporal alignment	TSX monthly index + monthly climate	Join on same month/year; no cross-month leakage	Avoids look-ahead bias
TC-DP-08	Unit/scale validation	Climate JSON (°C, mm)	Units normalized; schema records unit metadata	Prevent silent errors
TC-DP-09	Outlier handling	Inject extreme climate value	Outlier flagged/trimmed per rule; logged	Document rule
TC-DP-10	Duplicate detection	TSX CSV with dup rows	Duplicates removed; count report emitted	Data quality
TC-DP-11	Foreign key integrity	Species table vs. index table	All `species_id` in index exist in species	Referential integrity

TC-DP-12	Incremental load idempotency	Run pipeline twice same day	No duplicate rows; upserts stable	Use natural/composite keys
TC-DP-13	HTML artifact storage	Save updated map HTML	DB stores latest version; previous version archived	Matches TC-DP-03
TC-DP-14	Pipeline failure alerting	Force step error	Alert/log created with step name + traceback	Operability
TC-DP-15	Backfill safety	Backfill last 24 months	Only missing months inserted; no overwrite unless flagged	Change control

AI Recommendation System

ID	Description	Input	Expected Output	Notes
TC-AI-01	Show closest match of plants	Image from user	The list of plants shown to the user should include the actual plants that the user wants to have.	Image recognition should work with high accuracy
TC-AI-02	Consistency check	Same input twice	Identical input should give the same output	Model should be stable
TC-AI-03	Adversarial robustness test	Slightly altered plant image	Correctly classified plant or nearest match	To check on accuracy
TC-AI-04	Weather-supported recommendation	Lat/lon with location with a lot of rain	Recommend water-efficient plant	Cross-check with DB for accuracy
TC-AI-05	Recommendation diversity	Lat/lon in various temperature zone	Suggest at least 3 plants	Ensure variety for user to choose
TC-AI-06	Confidence threshold fallback	Low-quality/blurred image	If `conf < τ`, show “nearest matches” + KB tips, not a hard label	Better UX than wrong label
TC-AI-07	Determinism with seed	Same image, set seed	Same top-k orderings	A/B stable
TC-AI-08	Ties & diversity	Image near two classes	Top-k not all same family; at least 2 families	Avoids monotony
TC-AI-09	Climate-aware	Lat/Lon	Recommendations	Policy encoded

	filtering	marked “high rain”	exclude drought-sensitive species (or warn)	
TC-AI-10	Seasonal windowing	Winter month context	Plants not in current planting window flagged/warned	Practical guidance
TC-AI-11	Forecast-informed risk	Forecast shows decline	UI shows “at-risk” badge; suggests resilient alternatives	Uses TC-BE-05 data
TC-AI-12	Adversarial crop/rotate	Crop 25%, rotate 90°	Classifier still returns correct/near-correct	Robustness beyond TC-AI-03
TC-AI-13	Repeatability across devices	Same photo iOS vs Android	Near-identical rankings (Δ score ≤ ε)	Platform parity

Map & Visualisation Test

ID	Description	Input	Expected Output	Notes
TC-MV-01	Start load of map	Open map view	Map loads with correct centre (Melbourne CBD)	Help user from going too far from location of focus
TC-MV-02	Usability of zoom and pan	Zoom in and out	Smooth navigation, markers should still remain visible	User don’t have to search for the same location (too quick panning can cause this)
TC-MV-03	Integration with plant information cards	Click map	Location of tree around that area should be displayed, with the proper card included	
TC-MV-04	Marker clustering density	Radius 10km dense area	Clusters appear; expanding zoom reveals members	Performance + clarity
TC-MV-05	Legend/scale correctness	Toggle layers	Legend updates; units correct (km/mm/°C)	Prevents misreadings
TC-MV-06	Bounds after search	Search a suburb	Map fits bounds of results with 10% padding	Polished feel
TC-MV-07	Stale overlay prevention	Change radius quickly twice	Only the latest overlay visible; no flicker	Cancels in-flight requests

TC-MV-08	Forecast overlay toggle	Enable “Forecast risk” layer	Shows hatch/heat overlay consistent with API data	Ties to forecasting
TC-MV-09	Card sync on marker tap	Tap marker	Right card highlights same entity; ARIA live region updates	A11y + sync state

Non-Functional Requirements

ID	Description	Input	Expected Output	Notes
TC-NF-01	P95 map render time	Cold start map load	P95 initial render ≤ 2.0s on mid-tier laptop	Measure on throttled network
TC-NF-02	Memory leak check	10 minutes of panning/zooming	Memory plateau; no continuous growth	DevTools performance tab
TC-NF-03	Cross-browser	Chrome, Safari, Firefox	Behaviour and layout consistent	Usually different: Safari
TC-NF-04	Security headers	Any page	`CSP`, `X-Content-Type-Options`, `X-Frame-Options`, `Referrer-Policy` set	Quick OWASP hygiene
TC-NF-05	Input validation fuzz	Random strings in search/IDs	400/validation messages; no server error	Prevents injection
TC-NF-06	Privacy: PII in logs	Trigger errors	Logs contain no PII/image contents	Redaction rules verified
TC-NF-07	404/500 UX	Bad route / forced server error	Themed error pages with recovery links	Cohesive product feel
TC-NF-08	Offline/spotty net	Toggle offline during fetch	Graceful degradation; retry button appears	Realistic mobile case

6.4. Troubleshoot