Arboracle: A Strategic and Technical Blueprint for Ecological Platform Synthesis

Section 1: The Arboracle Strategic Blueprint: A Unified Vision for Integration

This report presents a comprehensive strategic and technical blueprint for the development of the Arboracle platform. It provides an exhaustive, deeply researched plan designed to transform the initial prototype into a scalable, future-proof foundation for the Minimum Lovable Product (MLP) and the long-term Soil Grower vision. The core of this strategy is the deliberate and intelligent synthesis of existing assets—the initial Arboracle prototype, the open-source Terraware platform, and the iNaturalist ecosystem—into a new, unified, and architecturally superior system.

1.1 From Prototype to Platform: Embracing a Synthesis-Driven Approach

The journey from a promising prototype to a phenomenal platform requires a strategic shift in perspective. The task ahead is not one of mere aggregation—simply "stitching together" disparate codebases—but one of profound synthesis. We will selectively harvest the most potent concepts, robust data models, and proven technologies from each source asset to construct a new, cohesive, and far more powerful whole.
The initial Arboracle prototype, built with remarkable speed, successfully validated the core concept and demonstrated the potential for AI-driven interaction. The terraware-web project provides an invaluable asset: a battle-tested, practical, and open-source data model for the complex logistics of nursery and seed bank management. Its foundation on a PostgreSQL database signifies a commitment to data integrity and relational consistency that is essential for long-term scientific and operational rigor. Concurrently, the iNaturalist platform represents a global powerhouse of community-driven biodiversity data, offering a massive, engaged user base, a rich dataset, and a de facto taxonomic backbone for ecological projects worldwide.
Arboracle's ultimate destiny lies in the fusion of these three pillars: the operational practicality of Terraware, the community power of iNaturalist, and the unique ecological wisdom embodied in the STIM (Stasis Through Inferred Memory) research model. Achieving this fusion demands a more sophisticated architectural approach than the one envisioned in the initial prototype phase. The original plan's reliance on a Firebase-centric architecture, while excellent for rapid prototyping and certain real-time features, is insufficient to handle the relational complexity and query-intensive nature of the data models inherited from Terraware and required for deep integration with iNaturalist.
Therefore, the architectural conflict between the prototype's NoSQL model and the robust PostgreSQL foundations of Terraware and iNaturalist must be resolved. This fundamental divergence necessitates the adoption of a new, unified data strategy and a more flexible,

scalable backend architecture. This report will outline that new architecture, ensuring that the final platform is a harmonious synthesis that honors the strengths of its progenitors while transcending their individual limitations.

1.2 The "10-Hour Genesis Sprint": Redefining Speed and Success

The directive to advance the project "10x or even 100x" within a 10-hour timeframe is a powerful call for extreme efficiency and radical prioritization [User Query]. A literal interpretation—building a feature-complete, fully integrated platform in such a short period—is not feasible given the complexity of the systems involved. A "phenomenal" build, by definition, implies a system that is scalable, reliable, and future-proof, characteristics that emerge from a meticulously designed and implemented architecture, typically a multi-month endeavor.

To reconcile this ambition with technical reality, the 10-hour build constraint will be reframed as a **"10-Hour Genesis Sprint."** The objective of this sprint is not to deliver a polished user interface or a wide array of features. Instead, its singular focus is to forge the unshakeable architectural skeleton of the Arboracle platform. This approach redefines "success" for this phase from feature completion to architectural readiness.

The highest-leverage activity for the AI agent team is to concentrate on infrastructure-as-code, service scaffolding, and the configuration of the core data layers. By the end of the 10-hour sprint, the project will have a fully containerized, multi-service backend architecture, a functioning API gateway, and established connections to the primary database systems. This foundational work is the most critical and time-consuming aspect of modern application development. By automating its creation, we achieve a state where subsequent development—whether by human engineers or other AI agent teams—can begin immediately on a stable, scalable, and correctly configured foundation.

This strategic re-scoping turns an ambitious request into a tactical masterstroke. It delivers the maximum possible value within the given constraint, accelerating the entire development lifecycle by front-loading the most complex foundational work. The outcome of the Genesis Sprint will be a robust, developer-ready platform backbone, poised for the rapid implementation of business logic and user-facing features.

1.3 The Guiding Principles of Arboracle

To ensure that every technical decision aligns with the project's ambitious goals, a set of guiding principles, derived from the core brand vision and technical research, will be established. These principles will serve as the constitution for the Arboracle platform, informing its architecture, development, and evolution.

- **Future-Proof & Scalable by Design:** The architecture must be engineered from the ground up to support the long-term vision of Soil Grower. This mandates a design that embraces horizontal scalability, allowing the system to handle increasing loads by distributing work across multiple servers rather than relying on the finite capacity of a single machine. Every component will be built with the anticipation of future growth in users, data volume, and feature complexity.
- **API-First Philosophy:** All platform functionality will be developed and exposed through a secure, well-documented, and versioned Application Programming Interface (API). This approach decouples the frontend client from the backend services, enabling parallel development and providing the flexibility to build future clients (e.g., mobile applications, third-party integrations, data analysis tools) without modifying the core backend logic.

- **Data-Centricity and Knowledge as an Asset:** The platform's most valuable asset is its data. The core relational database, containing the ground truth of tree records and user contributions, and the Ecological Knowledge Graph (EKG), which models the intricate web of life, will be treated as primary, mission-critical components. All architectural decisions will prioritize data integrity, consistency, and accessibility.
- **Ethical by Design:** Ethical considerations, particularly concerning the stewardship of sensitive ecological data and Indigenous Ecological Knowledge (IEK), will be woven into the fabric of the architecture, not treated as an afterthought. Principles of data sovereignty, informed consent, and community benefit will be implemented as architectural constraints within the database schemas, access control layers, and governance protocols. This ensures that the platform operates not just as a tool, but as a trusted and respectful digital steward.

Section 2: Technical Deep Dive: Deconstructing the Terraware and iNaturalist Ecosystems

A successful synthesis requires a granular understanding of the constituent parts. This section provides a detailed technical analysis of the Terraware and iNaturalist platforms, as well as the existing Arboracle assets, to identify the specific components, patterns, and data models that will be leveraged in the unified architecture.

2.1 Analysis of Terraware (terraware-web)

The Terraware open-source project offers a robust and practical foundation for the "Arbor" component of Arboracle—the grounded, data-driven management of physical trees and seeds.

- **Technology Stack:** Terraware employs a modern, decoupled technology stack. The frontend is built primarily with TypeScript (over 87%) and likely uses the React framework, as suggested by the use of tools like Craco for configuration overrides. The backend is a distinct service that communicates with the frontend via an API, and the entire system is designed for containerized development using Docker. Crucially, the database is PostgreSQL, evidenced by the presence of PL/pgSQL code and instructions for connecting to a PostgreSQL client. This choice of a powerful, open-source relational database is ideal for the structured data Arboracle will manage.
- **Key Features & Data Models:** The platform's core purpose is to provide seed inventory management and monitor the physical infrastructure of seed banks. This focus translates into a well-defined and practical data model. The PostgreSQL schema will contain entities such as SeedBank, Accession (a collection of seeds from a single species at a specific time and place), Species, and StorageLocation. Adopting and extending this schema will save an immense amount of time and effort in data modeling, providing a battle-tested structure for the core operational data of Arboracle.
- **Architectural Pattern:** Terraware is built on a classic client-server model, with a clear separation of concerns between the frontend application and the backend services. It utilizes Docker for creating a consistent local development environment and Keycloak for authentication and authorization. This pattern is inherently scalable and aligns perfectly with the microservices architecture proposed for Arboracle.

The primary value of Terraware is its pragmatic, real-world-tested foundation for arboricultural data management. Its PostgreSQL schema and decoupled architecture provide a significant head start for building the operational backbone of Arboracle.

2.2 Analysis of iNaturalist (inaturalist)

iNaturalist provides the "Oracle" component of Arboracle—the connection to a vast, living oracle of community-sourced biodiversity data and taxonomic wisdom. Understanding its architecture is key to defining our integration strategy.

- **Technology Stack:** The main iNaturalist platform is a monolithic application built with Ruby on Rails. The backend logic is written in Ruby, and the frontend is a combination of server-rendered views using Haml and dynamic client-side functionality using JavaScript. Like Terraware, its database of record is PostgreSQL, which confirms the choice of this database technology as the correct path forward for Arboracle.
- **API Ecosystem:** Direct integration with the Rails monolith is both infeasible and undesirable due to its complexity and tight coupling. Therefore, the iNaturalist API is the sole and essential point of integration. The platform provides multiple API versions, each with distinct characteristics:
    - **API v1:** This is the current, stable, and most widely used version. It is a Node.js-based API that provides comprehensive access to iNaturalist data, including observations, taxa, and users. It supports authenticated POST requests, which is critical for our goal of propagating Arboracle observations back to the iNaturalist database. Its primary drawback is that responses can be verbose, often returning all available fields for an object.
    - **API v2:** This version is still in development but represents the future direction of the iNaturalist API. Its most significant advantage is that it allows clients to specify exactly which data fields they require in a response, leading to smaller payloads and more efficient data transfer. While we will build our initial integration against the stable v1 API, the design of our dedicated iNaturalist Integration Service will be modular, allowing for a straightforward upgrade to v2 as it matures.
- **Data Models & Standards:** iNaturalist's data models are the de facto standard for community science biodiversity data. The core entities are Observation, Taxon, User, Identification, and Project. The iNaturalist API will serve as the primary, authoritative source for all taxonomic information within Arboracle. By fetching Taxon data directly from iNaturalist, we ensure that our platform's species information is always aligned with a global, curated, and community-vetted standard.

2.3 Analysis of Existing Arboracle Assets

The initial materials developed for Arboracle provide the project's unique vision and intellectual core.

- **Arborcast Implementation Plan:** This 6-month roadmap details a phased rollout of an AI-powered podcast and conversational agent named "Bodhi". It outlines a clear progression from a basic MVP to a sophisticated AI platform with user accounts, dynamic voice generation (using Google Cloud TTS), and LLM integration (using Gemini). Its primary technical contribution is the detailed plan for leveraging Firebase services, including Hosting, Storage, Firestore, Authentication, and Cloud Functions. As previously noted, while Firebase is excellent for certain tasks, its NoSQL database (Firestore) is not suitable as the primary data store for the full Arboracle vision. However, the plan's use of Firebase Authentication and Cloud Functions remains highly relevant and will be incorporated into the unified architecture.

- **STIM Research Model:** The "Stasis Through Inferred Memory" model is the project's most unique and valuable asset. It proposes a novel AI memory architecture inspired by biomimicry and grounded in ecological principles called the "Truths of Nature and Survivability." Concepts within STIM, such as "Interconnectedness Linking," the "Ecological Knowledge Graph (EKG)," and a "Mycelial Network Architecture," are not just abstract ideas; they map directly and powerfully to the technical capabilities of a graph database. This research provides the theoretical mandate for implementing the EKG as the intelligent core of the platform.
- **Brand Master Document:** This document provides the soul and personality of the Arboracle brand. It defines the mission, core values, and visual identity. Critically, it introduces the "Botanical Family of AI Personalities" (e.g., Bodhi, Quercus, Prunus) and the gamified "Soil Grower" user classification system. These brand concepts will be translated into concrete technical implementations: the AI personalities will become configurable routing layers or specialized models within the AI service, and the Soil Grower classifications will be implemented as roles and attributes within the User Service and database.

Table 1: Comparative Platform Analysis

The following table synthesizes the analysis of the three source platforms, highlighting the key technical attributes that inform the proposed unified architecture. The critical conflict at the database layer becomes immediately apparent, justifying the need for a new hybrid data strategy.

Feature	Initial Arboracle Prototype	Terraware (terraware-web)	iNaturalist (inaturalist)
Vision/Purpose	AI-driven ecological content & conversational guidance.	Practical seed inventory & nursery management.	Global community science & biodiversity data collection.
Frontend Tech	HTML, CSS, JavaScript.	TypeScript, React.	Haml, JavaScript (Server-Rendered).
Backend Tech	Firebase Cloud Functions (Node.js).	Separate API Service (Containerized).	Monolithic Ruby on Rails application.
Database	**Firestore (NoSQL)**	**PostgreSQL (Relational)**	**PostgreSQL (Relational)**
Authentication	Firebase Authentication.	Keycloak.	Custom Rails-based authentication.
Key Data Models	podcasts, users, knowledgeBase.	SeedBank, Accession, Species, StorageLocation.	Observation, Taxon, User, Identification.
API	Custom Cloud Function endpoints.	Internal REST API.	Public REST API (v1, v2).
Licensing	Proprietary.	Apache-2.0 (Open Source).	MIT (Open Source).

Section 3: The Unified Arboracle Architecture: A

Scalable, Future-Proof Foundation

To achieve the synthesis of practicality, community, and wisdom, Arboracle requires a new, purpose-built architecture. This section presents the technical blueprint for that architecture, designed from the outset for scalability, flexibility, and long-term evolution into the Soil Grower platform. It is a modern, robust design that adopts industry best practices for building complex, data-intensive applications.

3.1 Architectural Philosophy: A Decoupled, API-First Microservices Model

The unified Arboracle platform will be constructed as a collection of independent, specialized, and loosely coupled services that communicate with each other and with client applications through a central API Gateway. This microservices architecture is the industry-standard approach for building applications that are complex, scalable, and maintainable over the long term.

This architectural choice offers several decisive advantages over a monolithic approach:

- **Scalability:** Each microservice can be scaled independently based on its specific workload. For example, if the platform experiences a surge in image uploads, the media-service can be allocated more resources (horizontal scaling) without impacting the performance of the user-service or the iNaturalist-integration-service. This granular control is essential for both performance and cost-efficiency.
- **Technological Flexibility:** While the initial implementation will utilize a unified Node.js/TypeScript stack for development velocity, the microservices pattern allows for technological heterogeneity in the future. If a specific task, such as complex ecological modeling, is better suited to a different language like Python, a new service can be written in Python and integrated seamlessly into the ecosystem.
- **Maintainability and Development Velocity:** Small, focused services with well-defined responsibilities are easier for development teams (whether human or AI) to understand, update, test, and debug. This separation of concerns allows for parallel development on different parts of the system, significantly increasing overall development speed.
- **Resilience and Fault Isolation:** In a distributed system, the failure of one non-critical service does not need to cause a catastrophic failure of the entire application. For instance, if the iNaturalist-integration-service temporarily fails due to an external API issue, the core functionality of managing a user's own tree inventory can remain fully operational.

3.2 The Hybrid Data Layer: PostgreSQL as the Source of Truth, Firebase for Speed

The fundamental conflict between the prototype's NoSQL database and the relational databases of Terraware and iNaturalist is resolved not by choosing one over the other, but by employing a hybrid strategy that leverages both platforms for their unique strengths.

- **PostgreSQL - The System of Record:** A robust, open-source PostgreSQL database will serve as the single source of truth for all core Arboracle data. This includes user profiles, tree and seed records, inventory data, and locally created observations. PostgreSQL's relational nature, support for complex queries, and transactional integrity (ACID compliance) are non-negotiable for maintaining the scientific and operational rigor of the platform's data. The database will be managed within a Docker container for consistency across development and production environments.

- **Firebase - The Application Accelerator:** While not the primary database, specific Firebase services will be integrated to handle tasks for which they are exceptionally well-suited:
    - **Firebase Authentication:** This service will manage the entire user lifecycle, including sign-up, login, password resets, and session management. It is a highly secure, scalable, and battle-tested solution that provides out-of-the-box support for email/password, social providers (Google, etc.), and phone authentication. Offloading this complex and critical function to Firebase saves significant development time and reduces security risks.
    - **Future Real-time Features:** As Arboracle evolves, services like Firestore or the Realtime Database can be strategically employed for features that require low-latency, real-time updates, such as a user-to-user messaging system or live notifications on collaborative projects. This can be done without compromising the integrity of the core relational data stored in PostgreSQL.
- **The Integration Pattern:** The two systems will be cleanly integrated using a standard, secure pattern. Firebase Authentication will issue JSON Web Tokens (JWTs) to authenticated clients (e.g., the web application). The client will then include this JWT in the Authorization: Bearer <token> header of every subsequent request to the Arboracle API Gateway. The API Gateway will be responsible for validating the JWT's signature and claims before forwarding the request to the appropriate downstream microservice. This pattern effectively decouples our application logic from the authentication provider, ensuring a secure and modular system.

3.3 The Microservices Ecosystem

The Arboracle platform will be composed of an initial set of core microservices. This list represents the foundation for the Minimum Lovable Product, with the flexibility to add more specialized services in the future.

- **API Gateway:** This is the single, unified entry point for all client requests. It acts as a reverse proxy, routing incoming traffic to the appropriate internal service. Its key responsibilities include:
    - **Routing:** Mapping public API endpoints (e.g., /api/v1/users/me) to internal service addresses (e.g., user-service:3001/me).
    - **Authentication:** Intercepting every request to validate the JWT from Firebase Auth, ensuring no unauthenticated traffic reaches the internal services.
    - **Rate Limiting & Security:** Protecting backend services from abuse and denial-of-service attacks.
    - **Request Aggregation:** Potentially combining data from multiple services into a single response for the client, simplifying frontend logic.
    - *Technology:* Express Gateway, Apollo Gateway, or a custom Express.js/Node.js application.
- **User Service:** This service is the authority on user data within the Arboracle ecosystem. It will manage user profiles, application-specific preferences (like chosen AI personality), and the "Soil Grower" classification and progress tracking. It will maintain its own set of tables in the primary PostgreSQL database and will listen for events from Firebase

Authentication (e.g., via a webhook or Cloud Function trigger) to create a corresponding user profile in its own database when a new user signs up.

- **Tree & Inventory Service:** This is the functional heart of the "Arbor" side of the platform. It will manage all data related to the physical trees, seeds, and inventory items that users track. The data models and API endpoints for this service will be heavily inspired by the practical, field-tested schema of Terraware. It will handle all CRUD (Create, Read, Update, Delete) operations for these core entities.
- **iNaturalist Integration Service:** This service acts as an abstraction layer, or "anti-corruption layer," for all interactions with the external iNaturalist API. It will be the only part of the system that communicates directly with api.inaturalist.org. Its responsibilities include fetching taxonomic data to enrich local records, searching iNaturalist for observations based on user queries, and, critically, formatting and posting new observations created within Arboracle back to a user's iNaturalist account. Isolating this external dependency in a dedicated service makes the system more resilient and easier to maintain or upgrade (e.g., from iNaturalist API v1 to v2).
- **EKG & AI Service (The "Oracle"):** This service encapsulates the platform's intelligence. It will host the Neo4j graph database that forms the Ecological Knowledge Graph (EKG) and expose the "Ask Bodhi" conversational AI functionality. It will receive natural language queries, translate them into Cypher queries to run against the EKG, and use a Large Language Model (LLM) to synthesize the retrieved graph data into a coherent, helpful response.
- **Media Service:** This service will handle all aspects of media management. When a user uploads an image of a tree or an observation, this service will be responsible for processing the upload, generating thumbnails or different resolutions, and storing the file in a scalable object storage solution like AWS S3, Google Cloud Storage, or MinIO. It will return a unique identifier or URL for the media, which will then be associated with the relevant record in the tree-inventory-service or ekg-service.

Table 2: Proposed Microservices Architecture

Service Name	Description/Responsibilities	Primary Technology	Data Store(s)
API Gateway	Single entry point for all clients. Handles request routing, authentication (JWT validation), and rate limiting.	Node.js (Express Gateway)	None
User Service	Manages user profiles, preferences, and "Soil Grower" status. Syncs with Firebase Auth.	Node.js (Express)	PostgreSQL
Tree & Inventory Service	Manages CRUD operations for trees, seeds, and inventory. Based on Terraware data models.	Node.js (Express)	PostgreSQL

Service Name	Description/Responsibilities	Primary Technology	Data Store(s)
iNaturalist Integration Service	Manages all communication with the external iNaturalist API (v1/v2). Propagates observations.	Node.js (Express)	None (stateless proxy)
EKG & AI Service	Hosts the Ecological Knowledge Graph (EKG) and the "Ask Bodhi" conversational AI.	Node.js (Express)	Neo4j (Graph Database)
Media Service	Handles upload, storage, processing, and retrieval of all user-generated media (images, audio).	Node.js (Express)	Object Storage (S3, GCS)

Section 4: The Ecological Knowledge Graph (EKG): Architecting the Brain of Arboracle

The true differentiator for Arboracle is its intelligence. This section details the plan for translating the sophisticated theoretical concepts of the STIM research model into a practical, powerful, and ethically sound AI core: the Ecological Knowledge Graph (EKG).

4.1 From STIM to EKG: A Practical Implementation

The STIM research model provides the perfect theoretical justification for the use of a graph database. Its core concepts of "Interconnectedness Linking," a knowledge base structured as a "Mycelial Network," and the desire to model a complex web of relationships are precisely the problems that graph databases are designed to solve. The EKG is the tangible, engineered implementation of STIM's vision.

- **Technology Choice: Neo4j:** To build the EKG, we will utilize Neo4j, a market-leading, highly performant, and scalable graph database. The rationale for this choice is multi-faceted:
  - **Native Graph Processing:** Neo4j stores data as a graph, meaning relationships are first-class citizens. Queries that traverse these relationships (e.g., "Find all fungi that have a symbiotic relationship with oak trees observed in this region") are orders of magnitude faster and more intuitive to write than the equivalent complex, multi-table JOINs in a relational database.
  - **Scalability and Enterprise Readiness:** Neo4j is proven to scale to billions of nodes and relationships, making it suitable for the long-term Soil Grower vision. It offers enterprise-grade features for security, clustering, and high availability.
  - **Rich Ecosystem:** Neo4j is supported by a robust ecosystem, including the Cypher query language, official drivers for Node.js and other languages, and visualization tools. Most importantly, it includes the **Graph Data Science (GDS)** library, a powerful suite of algorithms for tasks like community detection, link prediction, and

    centrality analysis, which will be invaluable for future ecological modeling.
  - **Direct Precedent:** The case study of Basecamp Research provides a compelling and direct precedent. They successfully used Neo4j to build the world's largest knowledge graph of Earth's natural biodiversity, containing over 5 billion biological relationships, demonstrating the platform's suitability for this exact domain.

4.2 EKG Schema Design: Modeling the Web of Life

The EKG will be designed using the Labeled Property Graph (LPG) model native to Neo4j. This model is both flexible and expressive, consisting of nodes (entities), relationships (connections), and properties (attributes on nodes and relationships).

- **Core Node Labels:**
  - Tree: Represents an individual, physical tree being tracked by a user in the Arboracle system. Properties will include a unique ID (uuid), speciesName, datePlanted, geospatialLocation (GeoJSON point), and other attributes derived from the Terraware data model.
  - Taxon: Represents a formal taxonomic entity (e.g., a species, genus, family). Key properties will be taxonId (the stable identifier from iNaturalist), scientificName, and commonName. This node serves as the canonical representation of a species, populated via the iNaturalist API.
  - Observation: Represents a record of a species occurrence. This could be an observation of a companion plant or animal associated with a tracked Tree, or an observation imported from iNaturalist. Properties will include observationId, observedAt, location, and qualityGrade.
  - EcosystemComponent: A highly flexible node label for representing any other interacting entity in the ecosystem. This could be further specialized with labels like Fungus, Insect, Bird, SoilMicrobe, or AbioticFactor (e.g., soil type, water source).
  - Trait: Represents a specific functional or morphological characteristic of a Taxon, such as DroughtTolerant, NitrogenFixing, or PollinatorAttractor.
  - KnowledgeSource: A critical node for data provenance and ethical governance. It represents the origin of a piece of information, with potential labels like iNaturalistObservation, UserSubmission, ScientificPaper, or IEKHolder.
- **Key Relationship Types:**
  - IS_A: The fundamental link connecting an instance to its classification. For example: (tree:Tree)-->(taxon:Taxon {scientificName: 'Quercus robur'}).
  - INTERACTS_WITH: The core relationship for modeling the ecosystem. It is highly flexible and can have a type property to specify the nature of the interaction, such as 'SYMBIOTIC', 'PREDATOR', 'POLLINATOR', or 'COMPETITOR'. Example: (taxon:Quercus_robur)-->(fungus:Fungus {name: 'Mycorrhiza'}).
  - HAS_OBSERVATION: Connects a Tree or Taxon to associated Observation nodes.
  - HAS_TRAIT: Links a Taxon to its known Trait nodes.
  - SOURCED_FROM: A crucial relationship for ethical data tracking. Every node or relationship containing factual information will be connected to a KnowledgeSource node. Example: (interaction:INTERACTS_WITH)-->(source:IEKHolder {name: 'CommunityName'}).

4.3 Integrating Biodiversity Data Standards

To ensure the EKG is scientifically valid, interoperable, and not an isolated data silo, it will be grounded in established global biodiversity data standards.

- **Darwin Core (DwC):** Darwin Core is the essential vocabulary for sharing biodiversity data. We will explicitly map the fields received from the iNaturalist API to their corresponding DwC terms and use these as the property keys on our Taxon and Observation nodes. For example, the name field from the iNaturalist taxon response will be stored as the dwc:scientificName property in our EKG. The observation date will be dwc:eventDate, and coordinates will be dwc:decimalLatitude and dwc:decimalLongitude. This semantic alignment ensures that data exported from Arboracle is immediately understandable and usable by the broader scientific community.
- **Ecology Ontologies:** To describe habitats, environments, and functional traits with precision, the EKG will incorporate terms from well-established ecology ontologies. The **Environment Ontology (ENVO)** provides a standardized vocabulary for describing biomes, environmental features, and materials. For instance, instead of a free-text habitat property, a Tree node can be linked to an ENVO term, such as environment: 'ENVO:01000219' for a temperate broadleaf forest. Similarly, trait ontologies can be used to formally define the Trait nodes. This use of ontologies adds a rich, machine-readable semantic layer to the graph, enabling more powerful and precise queries.

4.4 Ethical by Design: Incorporating Indigenous Ecological Knowledge (IEK)

The integration of Indigenous Ecological Knowledge is not a technical problem to be solved, but a sacred trust to be honored. The architecture must be built from the ground up on a foundation of deep respect, informed consent, and unwavering support for Indigenous data sovereignty.

- **Implementation Strategy:**
    - **OCAP® Principles as Architectural Mandates:** The design will be guided by the First Nations principles of Ownership, Control, Access, and Possession (OCAP®). The KnowledgeSource and SOURCED_FROM components of the EKG are the primary mechanisms for this. Any data point (a node or relationship) identified as IEK will be immutably linked to a KnowledgeSource node representing the specific knowledge holder or community. This creates an unbreakable chain of provenance.
    - **Data Sovereignty through Architectural Controls:** Access to IEK-tagged data will be governed by rules embedded directly into the EKG & AI Service. These are not merely application-level permissions that a future developer might forget to check; they are fundamental architectural constraints. Queries for data linked to a specific IEK source will be programmatically filtered or rejected unless the request meets the access protocols defined by that source community.
    - **Cultural Protocols via Digital Labeling:** We will investigate and implement frameworks like **Local Contexts** (tklabels.org) and the **Traditional Knowledge (TK) Labels**. These systems provide digital tags that can be applied to data, clearly communicating the specific cultural protocols that govern its access, use, and circulation. These labels will be stored as properties on the KnowledgeSource nodes in the EKG, making the protocols machine-readable and enforceable by the API.
    - **Preservation of Context:** IEK will never be homogenized or stripped of its context. The graph structure allows us to store not just the "what" of the knowledge, but also

the "who," "how," and "why." The relationships, stories, and cultural significance associated with a piece of knowledge are as important as the factual assertion itself and will be preserved within the graph.

Table 3: Ecological Knowledge Graph (EKG) Core Schema

Component Type	Label/Type	Key Properties / Description	Example
Node	Tree	uuid: Unique ID within Arboracle speciesName: string geospatialLocation: GeoJSON	A specific oak tree planted by a user in their backyard.
Node	Taxon	taxonId: iNaturalist ID dwc:scientificName: string dwc:kingdom, dwc:phylum, etc.	The species *Quercus robur* (English Oak).
Node	Observation	observationId: iNaturalist or Arboracle ID dwc:eventDate: datetime dwc:decimalLatitude: float	A user's photo of a specific beetle on their oak tree.
Node	EcosystemComponent	name: string type: e.g., 'Fungus', 'Insect', 'Bird'	A node representing the *Mycorrhiza* fungus group.
Node	KnowledgeSource	name: string (e.g., 'iNaturalist', 'Community X') type: 'API', 'IEKHolder', 'Publication' accessProtocol: TK Label URL	A node representing the traditional knowledge of a specific Indigenous community.
Relationship	IS_A	Connects an instance (Tree) to its classification (Taxon).	(Tree {uuid: 'abc'})-->(Taxon {taxonId: 123})
Relationship	INTERACTS_WITH	type: 'SYMBIOTIC', 'PREDATOR', etc. description: text	(Taxon {name: 'Oak'})-->(EcosystemComponent {name: 'Mycorrhiza'})
Relationship	SOURCED_FROM	citation: string accessDate: datetime	(INTERACTS_WITH relationship)-->(KnowledgeSource {name: 'Community X'})

Section 5: The 10-Hour Genesis Sprint: A Prioritized

Action Plan for the AI Agent Team

This section provides the direct, prescriptive, and ruthlessly prioritized action plan for the AI agent team's 10-hour build. The focus is exclusively on establishing the foundational architecture outlined in Section 3. User-facing features are explicitly out of scope for this sprint.

5.1 Sprint Objective & Definition of Done

- **Primary Objective:** To initialize, configure, containerize, and deploy a fully functional, multi-service backend architecture. This includes a central API gateway, live and connected PostgreSQL and Neo4j database instances, and the initial scaffolding for all core microservices.
- **Definition of Done:** The sprint is considered complete when the following conditions are met:
    1. The entire system can be started locally with a single docker-compose up --build command.
    2. All services (api-gateway, user-service, etc.) are running in their respective Docker containers without errors.
    3. The PostgreSQL and Neo4j databases are running, accessible, and have their data persisted in Docker volumes.
    4. A developer can obtain a valid JWT from Firebase Authentication.
    5. Using this JWT, the developer can make an authenticated API call to a protected endpoint on the api-gateway (e.g., GET /api/v1/users/me).
    6. The gateway successfully validates the JWT and proxies the request to the user-service.
    7. The user-service successfully queries the PostgreSQL database and returns the correct user data.
    8. The master README.md file contains clear, accurate instructions for achieving the above steps.

5.2 Hour-by-Hour Breakdown

This schedule is designed to be executed sequentially by the AI agent team. Each block represents a set of focused tasks with clear deliverables.

- **Hours 0-1: Project Initialization & Monorepo Setup**
    - **Tasks:** The first hour is dedicated to setting up the project's structure. This involves creating the main GitHub repository named arboracle-platform. A monorepo structure will be initialized to manage the multiple service packages efficiently. This can be achieved using tools like pnpm workspaces or Lerna. A root packages or services directory will be created to house the code for each microservice. Top-level configuration files, including a comprehensive .gitignore (ignoring node_modules, .env files, build artifacts, etc.) and a prettierrc file to enforce consistent code formatting, will be established.
- **Hours 1-3: Containerization & Database Setup (Docker)**
    - **Tasks:** This block focuses on creating the operational environment. A master docker-compose.yml file will be created at the project root. This file will define the core infrastructure services: a postgres service using the official PostgreSQL image,

a neo4j service using the official Neo4j image, and a pgadmin or adminer service for easy visual inspection and administration of the PostgreSQL database. Docker volumes will be configured for both postgres and neo4j to ensure that all data is persisted on the host machine, even when containers are stopped and restarted. Initial database migration scripts will be written using a library like node-pg-migrate. The first migration will create the users table with essential fields (id, uid (from Firebase), email, created_at).

- **Hours 3-5: API Gateway Implementation**
  - **Tasks:** The central nervous system of the architecture is built here. A new service directory, api-gateway, will be created and initialized as a Node.js/Express project. The gateway's primary function is to act as the single entry point for all client traffic. It will be configured to listen on the main application port (e.g., 8080). The most critical piece of middleware to be implemented is JWT validation. This will be accomplished using libraries like jsonwebtoken and jwks-rsa. The middleware will be configured to fetch the public signing keys from Firebase's well-known JWKS (JSON Web Key Set) endpoint, allowing it to cryptographically verify any JWT issued by Firebase Auth without needing a shared secret. Basic routing rules will be set up to proxy requests starting with /api/v1/users to the future user-service.
- **Hours 5-7: User Service Scaffolding**
  - **Tasks:** With the gateway in place, the first backend service, user-service, will be scaffolded. A new directory will be created and initialized as another Node.js/Express project. A database connection module will be implemented to handle the connection pool to the PostgreSQL container using the pg library. The core task is to create a single, protected API endpoint: GET /me. This endpoint will be protected by middleware that expects a validated JWT payload (passed from the gateway). Upon successful authentication, it will query the users table in the PostgreSQL database based on the uid from the JWT and return the corresponding user's profile information as a JSON response.
- **Hours 7-9: Scaffolding Core Services & Initial EKG Connection**
  - **Tasks:** To complete the architectural skeleton, placeholder service directories will be created for the remaining core services: tree-inventory-service, inaturalist-service, and ekg-service. Each will be initialized as a basic Node.js/Express project. Within the ekg-service, a dedicated Neo4j connection module will be created using the official neo4j-driver for Node.js. A simple health-check endpoint (e.g., GET /health) will be implemented in this service. This endpoint's sole purpose is to execute a simple Cypher query (e.g., RETURN 1) to verify that the service can successfully establish and maintain a connection to the Neo4j database container.
- **Hour 9-10: Documentation, Finalization, and Initial README.md**
  - **Tasks:** The final hour is dedicated to ensuring the project is usable and well-documented. The master README.md file will be populated with the precise setup and execution instructions generated throughout the sprint. Template environment files (.env.example) will be created for each service, clearly documenting all required configuration variables (database credentials, ports, Firebase project ID, etc.). A final end-to-end test run will be performed to ensure all services build correctly and launch without errors via the single docker-compose up command, solidifying the "Definition of Done."

Table 4: 10-Hour Genesis Sprint Task & Time Allocation

Time Block	Primary Objective	Key Tasks	Example AI Agent Prompts	Deliverable/Verification
Hour 0-1	Project Initialization	Create GitHub repo. Init pnpm monorepo. Create root configs.	"Initialize a new pnpm monorepo with a services directory. Create a root .gitignore file suitable for a Node.js monorepo."	GitHub repository exists with correct initial file structure.
Hours 1-3	Containerization	Create docker-compose.yml. Define postgres & neo4j services with persistent volumes. Write initial SQL migration for users table.	"Generate a docker-compose.yml file that runs PostgreSQL and Neo4j services. Ensure data is persisted to local volumes named pgdata and neodata."	docker-compose up starts databases. pgadmin shows users table.
Hours 3-5	API Gateway Setup	Create api-gateway service. Implement Express server. Add JWT validation middleware using jwks-rsa for Firebase.	"Write an Express.js middleware function that validates a Firebase JWT from the Authorization header using the jwks-rsa library."	Gateway runs and rejects requests without a valid JWT.
Hours 5-7	User Service Build	Create user-service. Connect to PostgreSQL. Implement a protected GET /me endpoint to fetch user data.	"Create an Express route GET /me that reads a user ID from a validated JWT, queries a PostgreSQL users table for that user, and returns the result."	Authenticated call to /api/v1/users/me returns user data.
Hours 7-9	Core Service Scaffolding	Create placeholder services. Implement Neo4j connection and	"Generate the boilerplate for an Express service named	All service directories exist. ekg-service health check returns 200

Time Block	Primary Objective	Key Tasks	Example AI Agent Prompts	Deliverable/Verification
		health check in ekg-service.	ekg-service. Include a module that connects to a Neo4j database using the official driver."	OK.
Hour 9-10	Documentation & Finalization	Populate README.md with setup instructions. Create .env.example files for all services. Final test run.	"Generate a README.md with sections for 'Prerequisites' and 'Local Development Setup', detailing the steps to clone, configure.env files, and run docker-compose up."	A new developer can clone the repo and run the system successfully.

Section 6: The Master README.md: A Living Document for Development and Deployment

This section contains the complete, formatted markdown content intended for the project's master README.md file. It is designed to be the central, authoritative source of documentation for any developer, human or AI, working on the Arboracle platform.

Arboracle

Fusing practical arboriculture with community science and ecological AI.
Arboracle is a next-generation platform designed to empower individuals and communities to cultivate thriving ecosystems. It integrates robust tree and seed inventory management with a powerful, community-driven biodiversity database and a sophisticated ecological AI, creating a comprehensive tool for environmental stewardship.

Architectural Overview

The Arboracle platform is built on a modern, decoupled microservices architecture. This design ensures scalability, resilience, and maintainability. All client applications interact with the backend through a single **API Gateway**, which is responsible for authentication, routing, and security.
The backend consists of several independent services, each with a distinct responsibility:

- **User Service:** Manages user profiles and platform-specific data.
- **Tree & Inventory Service:** Handles all data related to physical trees and seed inventories.

- **iNaturalist Integration Service:** Acts as a dedicated proxy for all communication with the external iNaturalist API.
- **EKG & AI Service:** Hosts the Ecological Knowledge Graph (Neo4j) and powers the "Ask Bodhi" conversational AI.
- **Media Service:** Manages the upload and storage of all user-generated media.

Data is persisted in a hybrid model: a central **PostgreSQL** database serves as the system of record for core relational data, while a **Neo4j** graph database powers the EKG. User authentication is handled by **Firebase Authentication**. The entire system is containerized with **Docker** for consistent development and deployment environments.

Prerequisites

Before you begin, ensure you have the following software installed on your local machine:

- (https://www.docker.com/products/docker-desktop/)
- Node.js (v18 or later recommended)
- pnpm (as the package manager for the monorepo)

You will also need a Firebase project set up for handling user authentication.

Getting Started: Local Development Setup

Follow these steps to get the entire Arboracle platform running on your local machine.

1. **Clone the Repository**
   ```
   git clone https://github.com/MarvinFernie/arboracle-platform.git
   cd arboracle-platform
   ```

2. **Install Dependencies** Use pnpm to install all dependencies for all services in the monorepo.
   ```
   pnpm install
   ```

3. **Configure Environment Variables** Each service in the services/ directory requires its own .env file for configuration. Template files (.env.example) are provided in each service directory.For each service (e.g., api-gateway, user-service, etc.):
   ```
   cp services/<service-name>/.env.example
   services/<service-name>/.env
   ```
   Now, open each new .env file and fill in the required values (e.g., database passwords, Firebase project details, etc.). The default values in the docker-compose.yml file are configured to work with the example environment variables.
4. **Launch the Platform** Use Docker Compose to build and start all the services, including the databases.
   ```
   docker-compose up --build -d
   ```
   This command will:
   - Build the Docker images for each custom service.
   - Start containers for all services defined in docker-compose.yml.
   - Run the services in detached mode (-d).
5. **Verify Services are Running** You can check the status of all running containers:
   ```
   docker-compose ps
   ```
   To view the logs for a specific service:

```
docker-compose logs -f <service-name>
```
The API Gateway should now be accessible at http://localhost:8080.

Running Tests

Each service contains its own test suite. To run tests for a specific service, navigate to its directory and use the defined script:
```
cd services/<service-name>
pnpm test
```

API Documentation

The Arboracle API is documented using the OpenAPI specification. Once the platform is running, you can typically find the Swagger UI or a link to the Postman collection at the API Gateway's root endpoint or a /docs path.
To make authenticated requests, you must first obtain a JWT from your Firebase project. You can do this through your client application's login flow. Include the obtained token in the Authorization header of your API requests: Authorization: Bearer <your-firebase-jwt>

Key Data Models

- **PostgreSQL:** The primary relational database holds core data for users, trees, and inventory. The schema is managed via migration files located in the respective services. You can connect to the database using the pgadmin service defined in Docker Compose.
- **Neo4j (EKG):** The Ecological Knowledge Graph stores taxonomic data, observations, and their complex interrelationships. You can access the Neo4j Browser UI at http://localhost:7474 to explore the graph visually.

Deployment

The containerized nature of this application makes it suitable for deployment on any cloud provider that supports Docker (e.g., AWS ECS, Google Cloud Run, Azure Container Apps). A production deployment would involve:
1. Building and pushing service images to a container registry (e.g., Docker Hub, ECR, GCR).
2. Setting up managed PostgreSQL and Neo4j database instances (e.g., AWS RDS, Neo4j Aura).
3. Configuring production environment variables and secrets.
4. Deploying the services using an orchestration tool like Kubernetes or the cloud provider's native container service.

Contribution Guidelines & Roadmap

This project is in active development. Please refer to the project's issues tab for the current development roadmap and open tasks. Contributions are welcome; please follow standard fork-and-pull-request workflows.

Section 7: Beyond the Sprint: The Minimum Lovable Product (MLP) Roadmap

The successful completion of the 10-Hour Genesis Sprint provides the solid architectural bedrock upon which the Arboracle Minimum Lovable Product (MLP) can now be constructed. With the complex infrastructure in place, development can shift to a feature-focused, agile methodology. This roadmap outlines a series of targeted sprints that will build out the core user-facing functionality, adapting the vision from the original Arborcast plan to our new, more robust microservices architecture.

7.1 From Foundation to Features: An Agile Sprint Plan

The following sprint plan is a high-level guide for the next phase of development. Each sprint focuses on bringing a key component of the Arboracle vision to life by building out the functionality of one or more microservices and the corresponding frontend interfaces.

7.2 Sprint 2: Core Tree Management (2-4 Weeks)

- **Focus:** Building out the tree-inventory-service and the initial user-facing UI.
- **Features:**
    - **Backend:** Implement the full suite of CRUD (Create, Read, Update, Delete) API endpoints within the tree-inventory-service for managing Tree, Seed, and other inventory-related entities. The data models will be finalized based on the Terraware schema.
    - **Frontend:** Develop the initial React components for the user dashboard. This will include a view to list all of a user's tracked trees and a detailed view for a single tree, displaying its core attributes (species, planting date, location, etc.). A form will be created to allow users to add new trees to their inventory.

7.3 Sprint 3: iNaturalist Integration & Data Enrichment (2-4 Weeks)

- **Focus:** Activating the inaturalist-service and connecting its data flow to the EKG.
- **Features:**
    - **Backend (iNaturalist-service):** Implement the logic to call the iNaturalist API v1. A key feature will be an endpoint that allows searching for taxa by scientific or common name.
    - **Backend (ekg-service):** When a user adds a new tree, the tree-inventory-service will trigger an event. The ekg-service will listen for this event, call the inaturalist-service to get the canonical taxon data, and create/link the appropriate Tree and Taxon nodes in the Neo4j graph.
    - **Data Propagation:** Develop the crucial pipeline for propagating new observations created in Arboracle back to the user's iNaturalist account. This involves using an authenticated endpoint on the inaturalist-service that formats the data correctly and POSTs it to the iNaturalist API.

7.4 Sprint 4: The "Ask Bodhi" Experience (4-6 Weeks)

- **Focus:** Bringing the ekg-service and the conversational AI to life.
- **Features:**
    - **Frontend:** Develop the UI for the "Ask Bodhi" chat interface, including a message input field and a display area for the conversation history.
    - **Backend (ekg-service):** This is the most complex part. The service will receive a natural language question from the client. It will use an LLM (such as the Gemini API) for Natural Language Understanding (NLU) to parse the user's intent and extract key entities. It will then translate this intent into a formal Cypher query to be executed against the Neo4j EKG.
    - **Response Generation:** Once the Cypher query returns data from the graph (e.g., a list of interacting species), this structured data will be passed back to the LLM along with a prompt instructing it to synthesize the information into a fluid, helpful, and context-aware response, embodying the "Bodhi" personality.
    - **AI Personality Routing:** Implement a mechanism in the API Gateway or ekg-service to route queries to different LLM prompts or configurations based on the user's selected AI personality (Bodhi, Quercus, etc.).

7.5 Sprint 5: Community & Ecosystem Visualization (4-6 Weeks)

- **Focus:** Building features that leverage the richly interconnected data within the EKG.
- **Features:**
    - **Frontend:** Develop new UI components to visually represent a tree's ecosystem. This could be a graph visualization (using libraries like D3.js or Vis.js) that displays the Tree node and all its connected EcosystemComponent nodes based on the INTERACTS_WITH relationships in the EKG.
    - **Backend/Frontend:** Implement the full user profile pages and the "Soil Grower" classification system. The user-service will manage the logic for leveling up based on user contributions (e.g., number of trees planted, observations added). The frontend will display badges and progress indicators to gamify the user experience.

7.6 The Path to Soil Grower

This Minimum Lovable Product is the essential launchpad for the grander vision of Soil Grower. The microservices architecture provides the necessary flexibility to incrementally add more advanced capabilities without requiring a complete system overhaul. The path forward will involve:

- **Sophisticated Ecological Modeling:** Integrating more complex ecological models and simulators, such as TreeSim or models for species interaction networks , into a new, dedicated modeling-service.
- **Real-Time Data Integration:** Incorporating real-time sensor data (e.g., from soil moisture sensors, weather stations) via an IoT data ingestion pipeline, likely using services like AWS IoT Core or Google Cloud IoT, which would feed data into the EKG.
- **Predictive Analytics:** Leveraging the Neo4j Graph Data Science library to perform advanced analytics on the EKG, such as predicting potential pest outbreaks, identifying keystone species within a user's local ecosystem, or recommending companion plants to

enhance biodiversity.

The foundation laid in the Genesis Sprint and built upon in the MLP roadmap ensures that Arboracle is not just a product, but a scalable, evolving platform capable of realizing the full, profound vision of Soil Grower.

Works cited

1. terraware/terraware-web - GitHub, https://github.com/terraware/terraware-web 2. iNaturalist - GitHub, https://github.com/inaturalist 3. inaturalist/inaturalist: The Rails app behind iNaturalist.org - GitHub, https://github.com/inaturalist/inaturalist 4. 7 Essential Tips For Scalable Backend Architecture - Arunangshu Das, https://arunangshudas.com/blog/7-essential-tips-for-scalable-backend-architecture/ 5. Scalability:strategies For Achieving Scalability - FasterCapital, https://fastercapital.com/topics/scalability:strategies-for-achieving-scalability.html 6. Digital Platforms for Indigenous Knowledge Sharing - Prism → Sustainability Directory, https://prism.sustainability-directory.com/scenario/digital-platforms-for-indigenous-knowledge-sharing/ 7. Beyond Conservation: Working Respectfully with Indigenous People and Their Knowledge Systems, https://ipcaknowledgebasket.ca/resources/working-respectfully-with-indigenous-people-and-their-knowledge-systems/ 8. unimelb.libguides.com, https://unimelb.libguides.com/Indigenous_Knowledges_Research/Ethics#:~:text=At%20the%20heart%20of%20these,cultural%20and%20intellectual%20property%20rights. 9. terraware/web-components: Web components used in terraformation apps - GitHub, https://github.com/terraware/web-components 10. React & Rails 7.... What's the consensus & hotness? - Reddit, https://www.reddit.com/r/rails/comments/1b20yfe/react_rails_7_whats_the_consensus_hotness/ 11. inaturalist/iNaturalistAPI: Node.js API for iNaturalist.org - GitHub, https://github.com/inaturalist/iNaturalistAPI 12. iNaturalist API Example #1 / robin-song - Observable, https://observablehq.com/@robin-song/inaturalist-api-example-1 13. pyinaturalist 0.20.1 documentation, https://pyinaturalist.readthedocs.io/en/stable/ 14. API Documentation: Getting private Observation location data for own observations on localhost - General - iNaturalist Forum, https://forum.inaturalist.org/t/api-documentation-getting-private-observation-location-data-for-own-observations-on-localhost/54278 15. How to use iNaturalist API's to fetch observations like I would from web page?, https://forum.inaturalist.org/t/how-to-use-inaturalist-apis-to-fetch-observations-like-i-would-from-web-page/61294 16. API v1 vs. API v2 observation count by project not the same - General - iNaturalist Forum, https://forum.inaturalist.org/t/api-v1-vs-api-v2-observation-count-by-project-not-the-same/24394 17. API Reference - pyinaturalist 0.20.1 documentation, https://pyinaturalist.readthedocs.io/en/stable/reference.html 18. API interface backward compatibility - #2 by pisum - General - iNaturalist Community Forum, https://forum.inaturalist.org/t/api-interface-backward-compatibility/63352/2 19. Build a Scalable and Resilient Data Platform Architecture - Acceldata, https://www.acceldata.io/blog/designing-a-future-ready-data-platform-architecture 20. Integrate Postgres and Firebase to create automation - BuildShip, https://buildship.com/integrations/apps/postgres-and-firebase 21. Neo4j Graph Database Platform, https://neo4j.com/product/neo4j-graph-database/ 22. Neo4j Graph Database &

Analytics | Graph Database Management System, https://neo4j.com/ 23. Basecamp Research - Graph Database & Analytics - Neo4j, https://neo4j.com/customer-stories/basecamp-research/ 24. Data modeling - Memgraph, https://memgraph.com/docs/data-modeling 25. Graph database concepts - Getting Started - Neo4j, https://neo4j.com/docs/getting-started/appendix/graphdb-concepts/ 26. WHAT DO INTERACTION NETWORK METRICS TELL US ABOUT SPECIALIZATION AND BIOLOGICAL TRAITS?, https://interactio.org/wp-content/uploads/2014/10/bluethgen_et_al_2008.pdf 27. Frequently Asked Questions (FAQs) - GBIF.us, https://www.gbif.us/faq 28. What is Darwin Core, and why does it matter? - GBIF, https://www.gbif.org/darwin-core 29. Darwin Core - TDWG, https://www.tdwg.org/standards/dwc/ 30. Matching biodiversity and ecology ontologies: challenges and evaluation results | The Knowledge Engineering Review | Cambridge Core, https://www.cambridge.org/core/journals/knowledge-engineering-review/article/matching-biodiversity-and-ecology-ontologies-challenges-and-evaluation-results/772E301B65ED2774BB820D4148419860 31. Biodiversity and Ecology track (biodiv) - Ontology Alignment Evaluation Initiative, https://oaei.ontologymatching.org/2021/biodiv/index.html 32. What Are The Ethical Considerations When Using Indigenous Knowledge? → Question, https://lifestyle.sustainability-directory.com/question/what-are-the-ethical-considerations-when-using-indigenous-knowledge/ 33. Indigenous Research Methodologies: Research Ethics - UBC Library Research Guides, https://guides.library.ubc.ca/IndigResearch/researchethics 34. TreeSim: An object-oriented individual tree simulator and 3D visualization tool in Python, https://www.researchgate.net/publication/364603918_TreeSim_An_object-oriented_individual_tree_simulator_and_3D_visualization_tool_in_Python 35. Interaction network rewiring and species' contributions to community-scale flexibility | PNAS Nexus | Oxford Academic, https://academic.oup.com/pnasnexus/article/3/3/pgae047/7618480