Universidad Nacional de Colombia - Sede Bogotá
Facultad de Ingeniería
Ingeniería de Software II
Dominio del proyecto

Integrantes:
Yamid Alfonso Gonzalez Torres
Nelson Ivan Castellanos Betancourt
Maria Luisa Bautista Arango
Andres Felipe Ramirez Montana
Manuel Eduardo Diaz Sabogal
Eduard Harvey Patino Balaguera

ISIS Domain Documentation

1. System Overview

Sector

The ISIS platform operates within the Colombian agricultural sector. Its primary objective is to provide farmers, merchants, distributors, and other stakeholders with digital tools that facilitate the management of agricultural products, crop information, and market intelligence.

System Description

ISIS is a web-based agricultural information and management system that enables users to interact with products, crops, alerts, and real-time sector data.
Its core functionalities include:

- **Agricultural Marketplace:**
  Authenticated users can create, update, and delete agricultural product listings, which are published within the platform's internal marketplace.
- **Crop and Product Management:**
  Farmers can register and manage crop records and products available for sale or exchange.
- **Georeferenced Alert Module:**
  The platform features a geolocation-based alert system linked to Colombian departments and municipalities. Moderators publish alerts based on verified external information (news, reports, articles), including supporting images and categorized classifications.
- **Authentication and Security:**
  Access is managed through Google OAuth2, while internal service communication is secured using JWT, ensuring integrity, confidentiality, and controlled access across modules and users.

2. Stakeholders

Entities with vested interest in the system's functionality:

- Farmers
- Merchants
- Distributors
- General users within the agricultural sector
- Platform moderators
- System administrators
- Government institutions related to agriculture
- External technology providers (Google OAuth2, Cloudflare S3)

3. Actors

Roles that directly interact with ISIS:

- **Registered User (Google OAuth2):**
  Accesses the marketplace, views alerts, and browses products and crops.
- **Seller:**
  Publishes and manages agricultural products and crop-related information.
- **Buyer:**
  Searches, consults, and acquires marketplace products.
- **Moderator:**
  Creates and publishes georeferenced alerts with images and categorical metadata.
- **Administrator:**
  Manages platform users, permissions, configuration, and global system settings.
- **External Services:**
  Google OAuth2, JWT authentication services, and cloud storage providers.

4. Domain Context

Colombia's agricultural sector is shaped by its extensive biodiversity, varied climate conditions, and wide range of cultivated products. Despite its economic relevance, the sector faces major structural challenges, including:

- Limited access to updated market information
- Weak integration among producers, distributors, and buyers
- Scarce technological adoption
- Insufficient visibility for rural stakeholders
- Limited information about environmental or logistical risks affecting crops

To address these challenges, the ISIS platform was conceived as an integrated digital ecosystem that provides farmers and related actors access to:

- Real-time market data
- Direct commercial channels
- A community-oriented environment for sharing alerts, news, and field insights

The platform's name, **ISIS**, is inspired by the Greek goddess associated with agriculture and fertility. Its mission is to strengthen the connection between rural and urban communities, improve visibility for agricultural producers, and reduce historical inequalities affecting the sector.

ISIS supports the commercialization of agricultural goods by enabling farmers to provide complete and reliable product information, while also offering tools to document and manage their cultivated products. This enhances the digital presence of producers of all scales—small, medium, and large.

Furthermore, the platform informs producers about risks or phenomena that may affect their crops, fostering engagement and building a collaborative agricultural network.

Future development plans include:

- Integration with national payment systems
- A personalized notification engine
- Improved transactional processes to ensure faster, secure, and centralized interactions

5. System Model

Entities

- Alerts
- User
- Crops
- Posts
- Department
- Municipality
- Products
- User Profile
- Alert Categories
- User Groups (used to assign moderation privileges)

6. Relationships

1. Geographical Relationships

- **Department → Municipality (1:N):**
  Each department contains multiple municipalities; each municipality belongs to a single department.
- **Municipality → Entities:**
  The municipality is the spatial reference point for:
  - User profiles
  - Alerts
  - Marketplace posts

2. User and Profile Relationships

- **User ↔ Profile (1:1):**
  Each authenticated user has one corresponding user profile with extended attributes (bio, image, role).
- **User ↔ Groups (M:N):**
  Users may belong to multiple groups (e.g., Farmers, Moderators), and groups include multiple users.

3. Marketplace (Posts)

- **User → Posts (1:N):**
  A seller may publish multiple posts; each post is owned by a single seller.
- **Post → Category (N:1):**
  Each post belongs to one product category.
- **Post → Images (1:N):**
  A post can include multiple associated images.

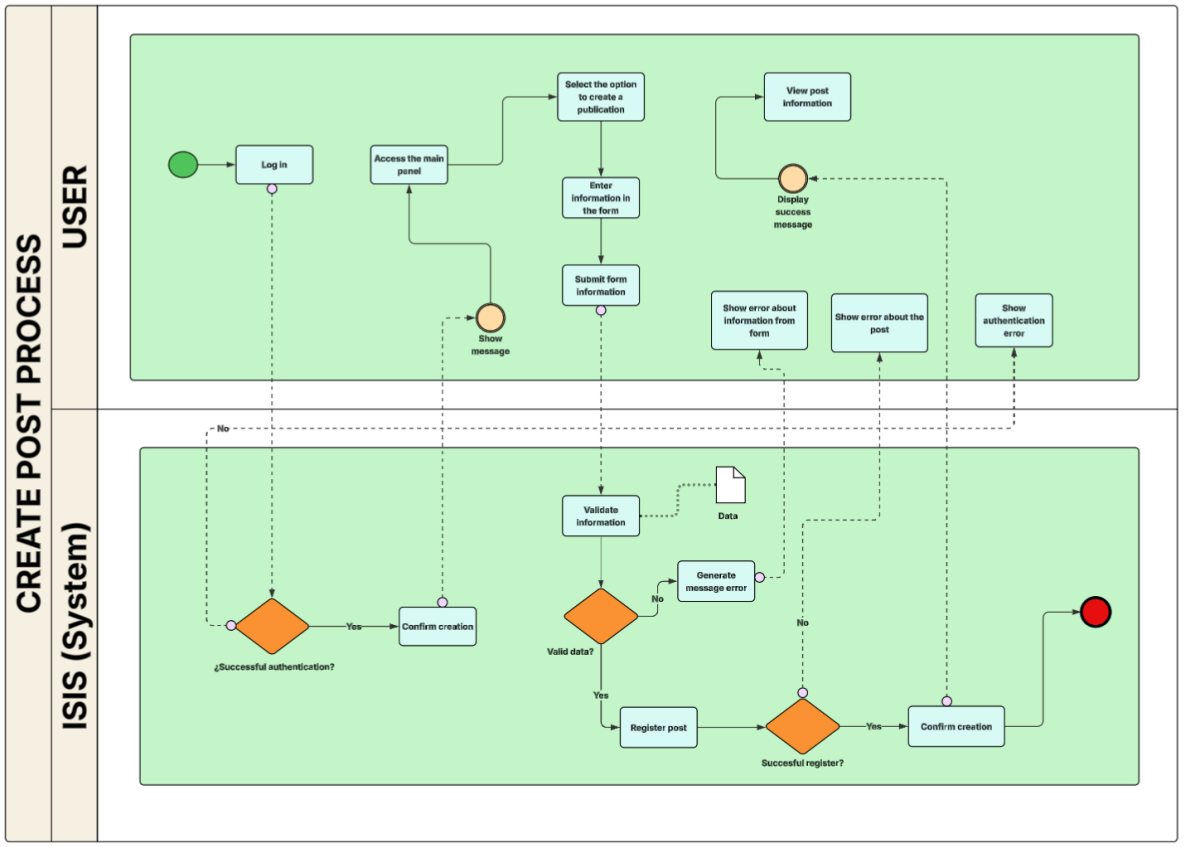
4. Agricultural Management (Crops)

- **User → Crops (1:N):**
  A farmer can register multiple crops; each crop belongs to one user.
- **Crop → Product (N:1):**
  Each crop is tied to a base product type.

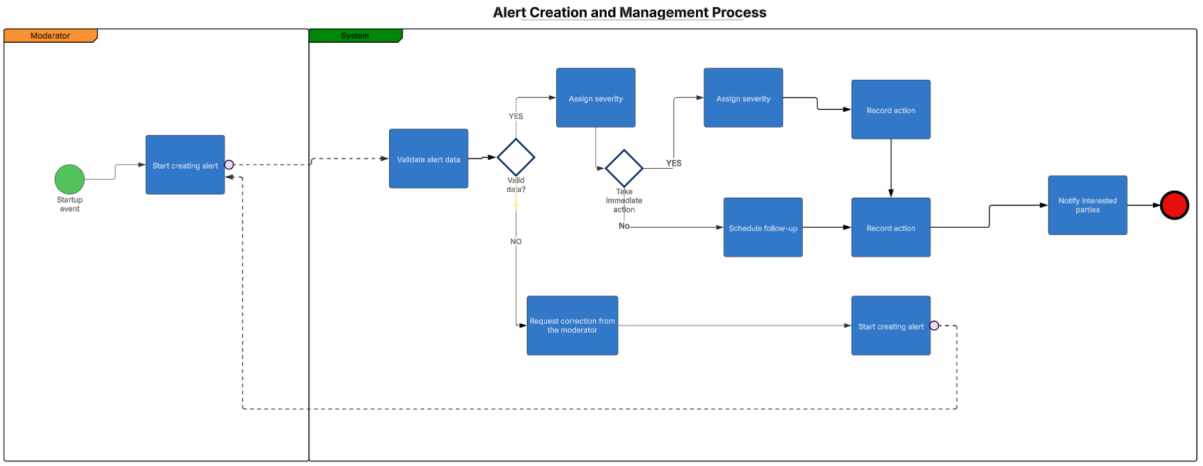
5. Alerts

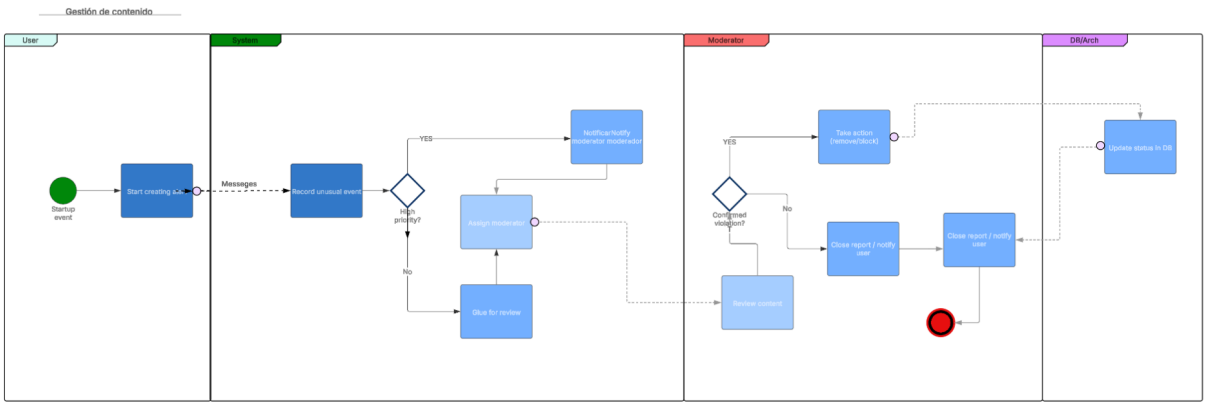
- **Moderator → Alerts (1:N):**
  A moderator may publish multiple alerts.
- **Alert → Category (N:1):**
  Each alert belongs to a specific alert category.
- **Alert → Images (1:N):**
  Alerts may contain one or more supporting images.

BPMN:



BPMN subprocesses:



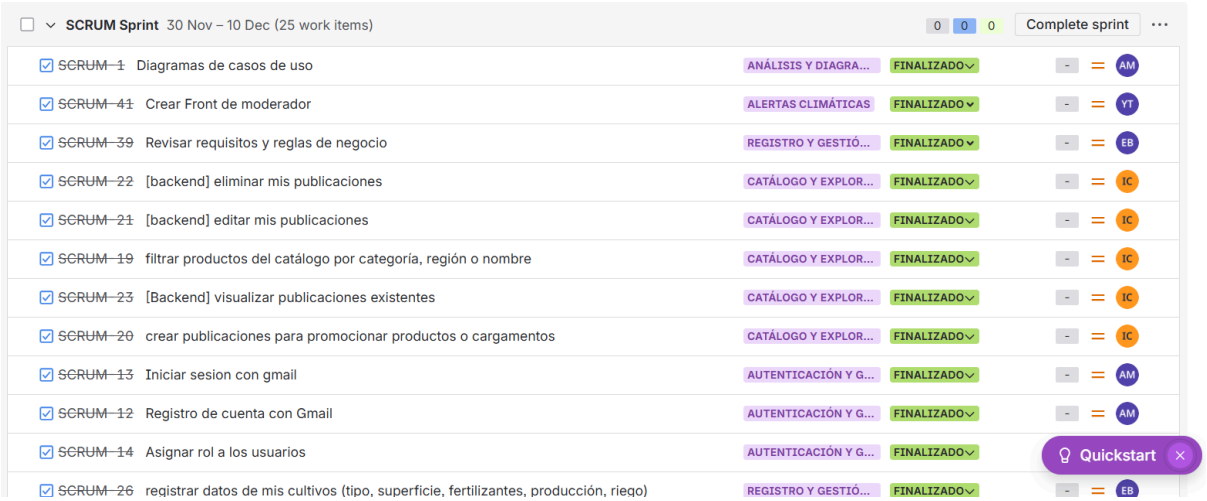


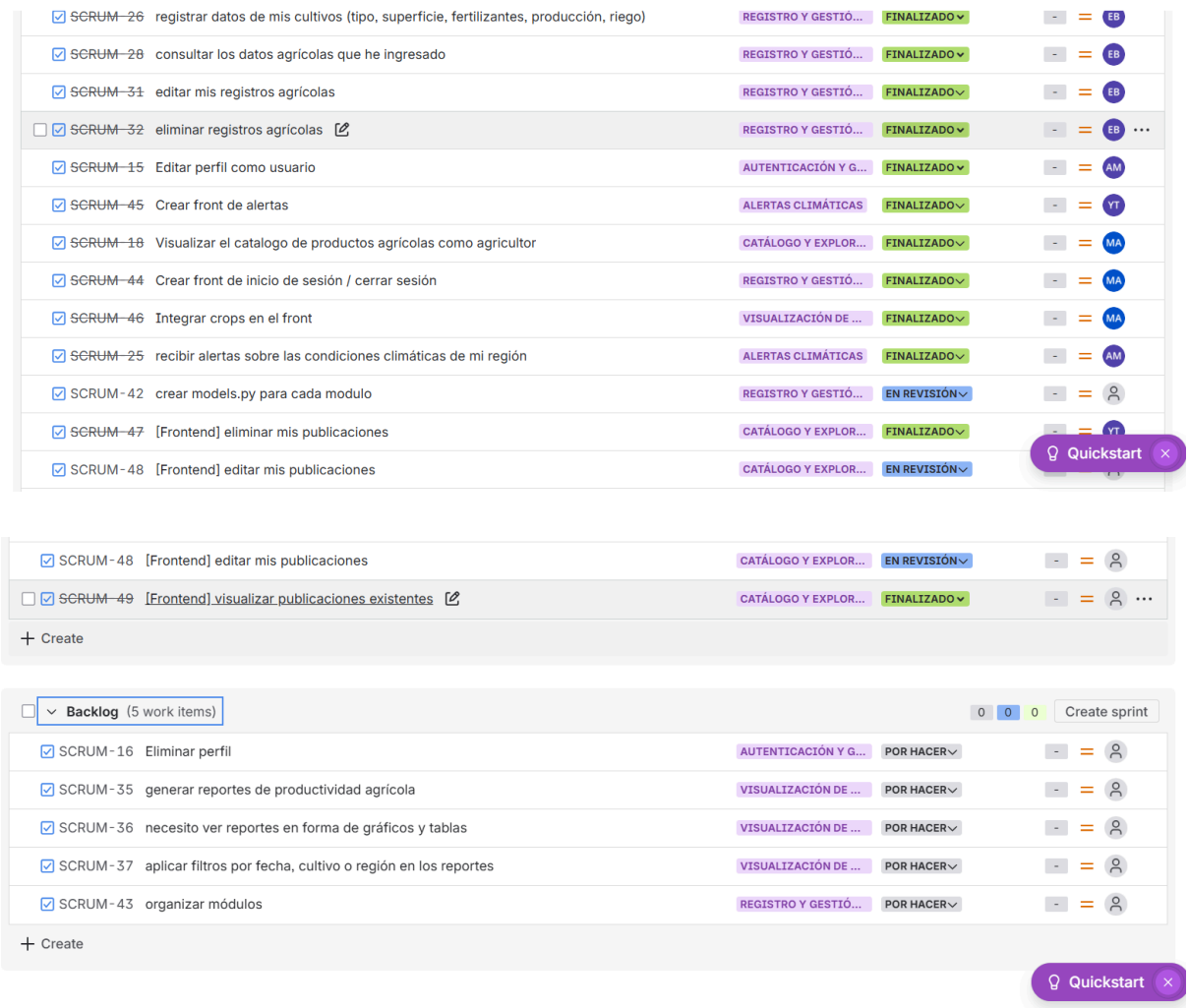
7. Business Rules

1. Only authenticated users may access full system functionality; the landing page is publicly accessible.
2. A user must complete their profile before accessing additional features.
3. Sellers must provide valid contact information before publishing posts.
4. Posts belong exclusively to the seller who created them.
5. Only active posts may be deleted.
6. All publication data must be truthful and free of inappropriate language.
7. Users may not modify other users' profiles.
8. Crop-related data belongs solely to its owner.
9. Agricultural data must be validated (non-negative values, correct formats, etc.).
10. Before agricultural data is displayed publicly, it must be anonymized through the domain service.

Sprint's

In this section we did a scrum sprint and continuously created backlogs





As we can see, although we were missing a backlog and two scrums, we did enough scrums to create a minimum viable product and succeed using the agile scrum methodology.

Design patterns

To ensure a solid software structure in the project, we employed various design patterns, including:

1. **Layered Architecture**
    - **Type**: Architectural
    - **Location**: Entire project
    - **Benefit**: Separation of concerns. This pattern helps to organize the system into layers, each with its specific responsibility, promoting cleaner code and better maintainability.
2. **DDD (Domain-Driven Design)**

- **Type**: Architectural
- **Location**: Models, signals, serializers
- **Benefit**: Focus on the domain. It emphasizes the domain model as the central focus of the design, aligning the software design with business requirements.

3. **Repository**
    - **Type**: Data
    - **Location**: Django ORM
    - **Benefit**: Data access abstraction. This pattern provides a way to access data without exposing the underlying storage mechanisms, simplifying database operations.
4. **Strategy**
    - **Type**: GOF (Gang of Four)
    - **Location**: CustomSocialAccountAdapter
    - **Benefit**: Flexibility in authentication. This pattern allows the system to support different authentication methods, making it adaptable to various scenarios.
5. **Observer (Signals)**
    - **Type**: GOF (Gang of Four)
    - **Location**: Django signals
    - **Benefit**: Event decoupling. The observer pattern helps in decoupling event producers and consumers, allowing for easier maintenance and expansion of the system's behavior without modifying core components.
6. **DTO (Data Transfer Object)**
    - **Type**: Data
    - **Location**: Serializers
    - **Benefit**: Clean data transfer. This pattern is used to transfer data between processes or layers in a system without exposing internal details, ensuring a clean and well-defined contract.
7. **OAuth 2.0**

- **Type**: Security
- **Location**: django-allauth
- **Benefit**: Secure social login. OAuth 2.0 provides a standardized and secure way to authenticate users using third-party providers, enhancing the system's security and user convenience.

8. **JWT (JSON Web Tokens)**
   - **Type**: Security
   - **Location**: rest_framework_simplejwt
   - **Benefit**: Stateless authentication. JWT allows the system to authenticate users without storing session data on the server, enabling scalable, stateless authentication.
9. **REST API**
   - **Type**: Web
   - **Location**: DRF Controllers
   - **Benefit**: Standard and scalable API. REST APIs provide a flexible and standardized approach for communication between services, enabling scalability and simplicity in interactions.
10. **Active Record**
    - **Type**: Persistence
    - **Location**: Django ORM
    - **Benefit**: Simplicity of data access. Active Record is a pattern that allows objects to manage their own database records, simplifying the interaction with the database and reducing the need for complex queries.

language

1. **User:** Principal entity of the system representing any individual authenticated through the external provider Google OAuth 2 under OAuth 2.0 protocols. Each user has a globally unique identifier and an internal JWT access token that allows interaction with backend services. The User role acts as a generic superset within the domain. From this entity, specialized profiles are derived (Farmer, Buyer, Seller, Moderator, Administrator).

2. **User profile:** A specialized projection of the user that encapsulates attributes and relevant information to grant certain privileges defined by the backend (e.g., moderator status). On the frontend, it organizes this information for each publication.

3. **Agricultural product:** A domain entity representing a marketable item within the marketplace. An agricultural product encapsulates structured attributes such as name and its unique identifier (id). Each product is associated with a user, meaning the associated user assumes the role of seller. In the system, products are indexed in an internal search engine with filtering capabilities by region, crop type, and price range.

4. **Crop:** An entity linked to the user profile that models an agricultural production unit. A crop includes attributes such as fertilizers, creation date, crop type, production, crop area, and crop start date. This data is used to increase digital visibility for producers within the platform.

5. **Alerts:** A module responsible for generating, classifying, and distributing alerts related to the agricultural environment. Alerts function as an aggregate managed by the Moderator, who creates instances based on validated external information (news, articles, climate reports, etc.). Categories include alert type (fire, weather, flooding), descriptive content, geolocation by department, multimedia support, moderator user ID for future validations, and issue date.

6. **Authentication and Authorization:** Identity subsystem integrating Google OAuth2 for federated authentication under the OpenID Connect standard. Once authenticated, the backend generates an internal JWT token containing roles, permissions, and defined claims. This token controls access to endpoints.

7. **Marketing System / Marketplace:** A service that orchestrates interactions between Buyers and Sellers, managing the publication, search, and acquisition of agricultural products. It operates through REST APIs and an optimized query engine to facilitate product discovery by category, price, and product name.

Layer

1. **Domain layer:** This layer contains the functional core of the system and defines the rules governing the behavior of the agricultural model. It has no dependency on external frameworks.

    a. **Entidades Principales**
        i. User
        ii. User profile
        iii. Agricultural product
        iv. Crop
        v. Agricultural alert

vi. Department / Municipality
vii. Categories (Products / Alerts)
viii. Permissions groups

b. Valuables

i. Contact information
ii. Crop phenological state
iii. Price rang
iv. Inventor
v. News information (alerts)

c. Domain Rules and Invariants

i. Inventory and prices cannot be negative.
ii. A product can only be created or modified by its seller.
iii. Alerts may only be generated by users with moderator role.
iv. Agricultural data must pass format and consistency validations.
v. Profiles must be complete before enabling sensitive operations.
Maximum number of images for posts and alerts: 10.
vi. Moderators may also sell products and promote crops.
vii. All posts require seller information.
Alerts depend on the user profile's geographic information; if incomplete, alerts will not appear.

d. Domain Events

i. ProductCreated
ii. CropUpdate
iii. AlertPublished
iv. ProfileCompleted
v. CropCreated
vi. CropDeleted
vii. ProductFound

2. **Application Layer:** This layer orchestrates domain operations, coordinates between layers, and handles use cases.

Use cases:

a. User management

i. Register user
ii. UpdateProfile
iii. CompleteProfile
iv. ValidateProfileCompletion
v. GetUserInfo

b. Product Manager

i. CreateProduct
ii. UpdateProduct
iii. DeleteProduct
iv. GetProductsBySeller
v. SearchProducts
vi. ValidateInventory

c. Crop Management

i. CreateAlert (moderators only)
ii. PublishAlert
iii. FilterAlertsByLocation
iv. UpdateAlert
v. GetAvailableAlerts

d. Alert Management

i. CreateAlert (moderators only)
ii. PublishAlert
iii. FilterAlertsByLocation
iv. UpdateAlert
v. GetAvailableAlerts

Application Services:

a. UsuarioApplicationService

- registrar(data): RegistrationResult
- completarPerfil(userId, profileData): UpdateResult
- obtener(userId): UserData

b. ProductoApplicationService

- crear(userId, productData): CreationResult
- actualizar(productId, userId, updatedData): UpdateResult
- eliminar(productId, userId): DeletionResult
- obtenerPorVendedor(sellerId): ProductList
- buscar(filters): ProductList

c. CultivoApplicationService

- crear(userId, cropData): CreationResult
- actualizar(cropId, userId, updatedData): UpdateResult
- eliminar(cropId, userId): DeletionResult
- obtenerPorUsuario(userId): CropList
- actualizarEstadoFenológico(cropId, newState): UpdateResult

d. AlertaApplicationService

- crear(moderatorId, alertData): CreationResult
- publicar(alertId, moderatorId): PublicationResult
- obtenerPorUbicación(coords, radius): AlertList
- filtrarPorCategoria(category): AlertList
- actualizar(alertId, moderatorId, updatedData): UpdateResult

DTO's (Data Transfer Objects): In this part, the data is processed using serializers.

a. CreateProductDTO

- title: string
- description: string
- price: decimal
- category: string
- inventory: int
- images: List<Image> (max 10)
- location: Coordinates

b. CreateCropDTO

- name: string
- cropType: string
- sowingDate: Date
- location: Coordinates
- phenologicalState: string

c. CreateAlertDTO

- title: string
- description: string
- category: string

- geographicLocation: Department and municipality
- images: List<Image> (max 10)

d. CompleteProfileDTO

- bio: string
- location: Department and municipality
- phone: string
- profilePicture: Image

Event Handling

ProductCreatedEventHandler

- handle(event: ProductCreated): void
    - Notify users in geographic zone
    - Register in event history

AlertPublishedEventHandler

- handle(event: AlertPublished): void
    - Filter users by location
    - Register event

i.

3. **Capa de Interfaz:** Defines system entry points (REST API, frontend, etc.).

Controllers (REST API):

a. UsersController:

i. GET /alerts/
ii. GET /alerts/{id}/
iii. GET /alerts/categories/
iv. POST /alerts/create/

b. CropsController

i. GET /crops/
ii. POST /crops/

iii. GET /crops/{crop_id}/
iv. PUT /crops/{crop_id}/
v. PATCH /crops/{crop_id}/
vi. DELETE /crops/{crop_id}/
vii. GET /crops/products/
viii. GET /crops/products/{product_id}/

c. PostsController

i. Post Categories
   1. GET /posts/categories/
   2. GET /posts/categories/{slug}/

ii. Post Marketplace
   1. GET /posts/marketplace/ (listar posts públicos)
   2. GET /posts/marketplace/{id}/ (ver post público)

iii. Post Moderation
   1. GET /posts/moderation/
   2. GET /posts/moderation/pending_review/
   3. GET /posts/moderation/{id}/
   4. PUT /posts/moderation/{id}/
   5. PATCH /posts/moderation/{id}/
   6. PATCH /posts/moderation/{id}/activate/
   7. PATCH /posts/moderation/{id}/approve/
   8. PATCH /posts/moderation/{id}/reject/

iv. Posts - MyListings
   1. GET /posts/my-listings/
   2. POST /posts/my-listings/
   3. GET /posts/my-listings/{id}/
   4. PUT /posts/my-listings/{id}/
   5. PATCH /posts/my-listings/{id}/
   6. DELETE /posts/my-listings/{id}/
   7. PATCH /posts/my-listings/{id}/mark_as_sold/
   8. PATCH /posts/my-listings/{id}/pause_listing/
   9. PATCH /posts/my-listings/{id}/toggle_visibility/

d. UserController

i. GET /users/all/
ii. GET /users/me/
iii. GET /users/{username}/
iv. PATCH /users/{username}/profile/

e. Users Sellers

i. GET /users/sellers/
ii. GET /users/sellers/{username}/
iii. GET /users/sellers/{username}/posts/

f. DepartmentsController

i. GET /users/departments/

Validators (Input Validation):

a. **ProductValidator**
   1. validateTitle(title): Boolean
   2. validatePrice(price): Boolean
   3. validateInventory(inventory): Boolean
   4. validateImages(images): Boolean (max 10)
   5. validateCategory(category): Boolean

b. **CropValidator**
   1. validateName(name): Boolean
   2. validateCropType(type): Boolean
   3. validateSowingDate(date): Boolean
   4. validateLocation(coords): Boolean

c. **AlertValidator**
   1. validateSeverity(severity): Boolean
   2. validateLocation(coords): Boolean
   3. validateRadius(radius): Boolean
   4. validateImages(images): Boolean (max 10)
   5. verifyModeratorRole(userId): Boolean

d. **ProfileValidator**
   - validateContactData(data): Boolean
   - validateCoordinates(coords): Boolean
   - verifyCompleteness(profile): Boolean

Templates (Frontend):

a. **Páginas React**
   i. Alerts
   ii. CreateCrop
   iii. CreatePost
   iv. EditPost
   v. Home
   vi. MyProducts
   vii. ModeratorPanel
   viii. ProductDetails
   ix. ProductsBySellers
   x. Profile
   xi. Sellers
   xii. DashboardPage
   xiii. Contactanos

4. **Infrastructure Layer:** Handles persistence, external communication, and technical details.

Data Persistence:

a. **UserRepository (interface + implementación)**
   i. gsave(user)
   ii. getById(id)
   iii. getByEmail(email)
   iv. update(user)
   v. delete(id)

b. **ProductRepository**
   i. save(product)
   ii. getBySeller(sellerId)
   iii. update(product)
   iv. delete(id)
   v. search(filters)

c. **CorpRepository**
   i. save(crop)
   ii. getById(id)
   iii. getByUser(userId)
   iv. update(crop)
   v. delete(id)

d. **CategoryRepository**
    i. getProducts()
    ii. getAlerts()

e. **DepartmentRepository**
    i. getAll()
    ii. getMunicipalities(departmentId)

Third-party Integration:

a. **AuthenticationService (Google OAuth / Allauth)**
    i. authenticate(credentials): Token
    ii. refreshToken(token): NewToken
    iii. accessToken(token): Token

b. **StorageService (AWS S3 / Cloud Storage)**
    i. uploadImage(file): URL
    ii. deleteImage(url): void
    iii. getImage(url): Bytes

Database

a. **Main Tables**
    i. users
    ii. profiles
    iii. products
    iv. product_images
    v. crops
    vi. alerts
    vii. alert_images
    viii. categories
    ix. category_products
    x. category_alerts
    xi. departments
    xii. municipalities
    xiii. permission_groups
    xiv. permissions

b. **Key Relationships**
    i. User 1:1 Profile

ii. User 1:N Products
iii. User 1:N Crops
iv. Moderator 1:N Alerts
v. Product N:N Categories
vi. Alert N:N Categories

Logging & Monitoring

a. **LoggingService**
   i. registerOperation(action, user, result)
   ii. registerError(exception, context)

b. **MonitoringService**
   i. registerMetric(name, value)
   ii. reportPerformance()

Configuration

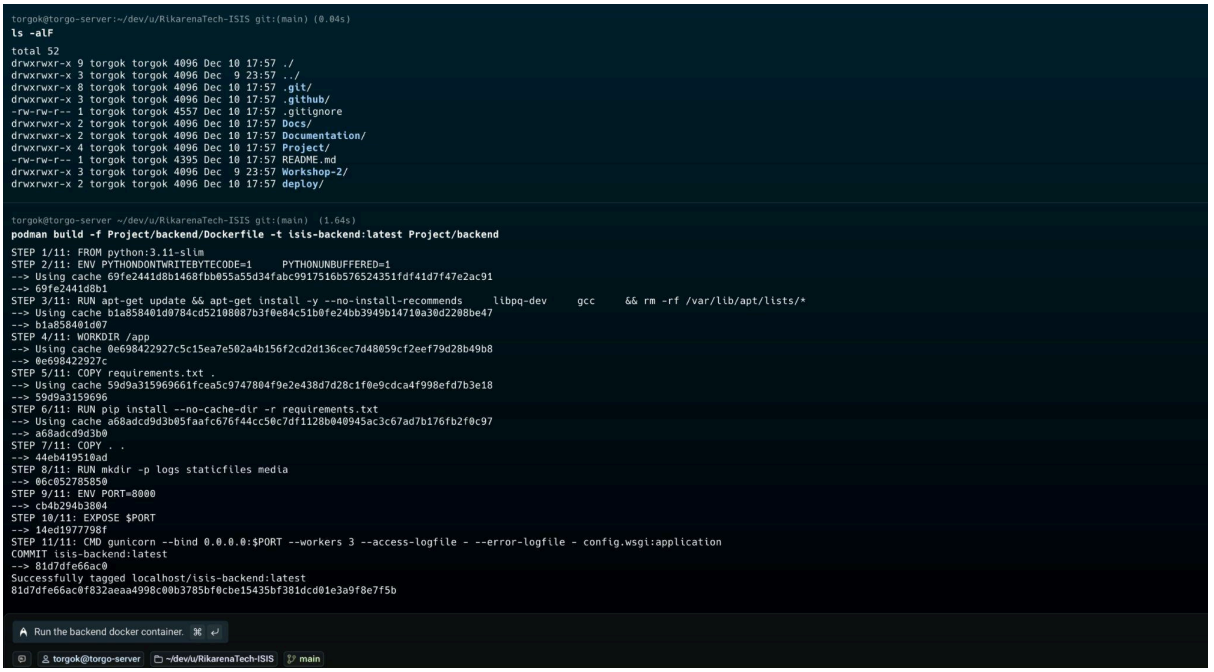
a. **DatabaseConfig**
   i. connectionString
   ii. poolSize
   iii. timeouts

b. **StorageConfig R2**
   i. bucketName
   ii. accessKey

Building the Backend:

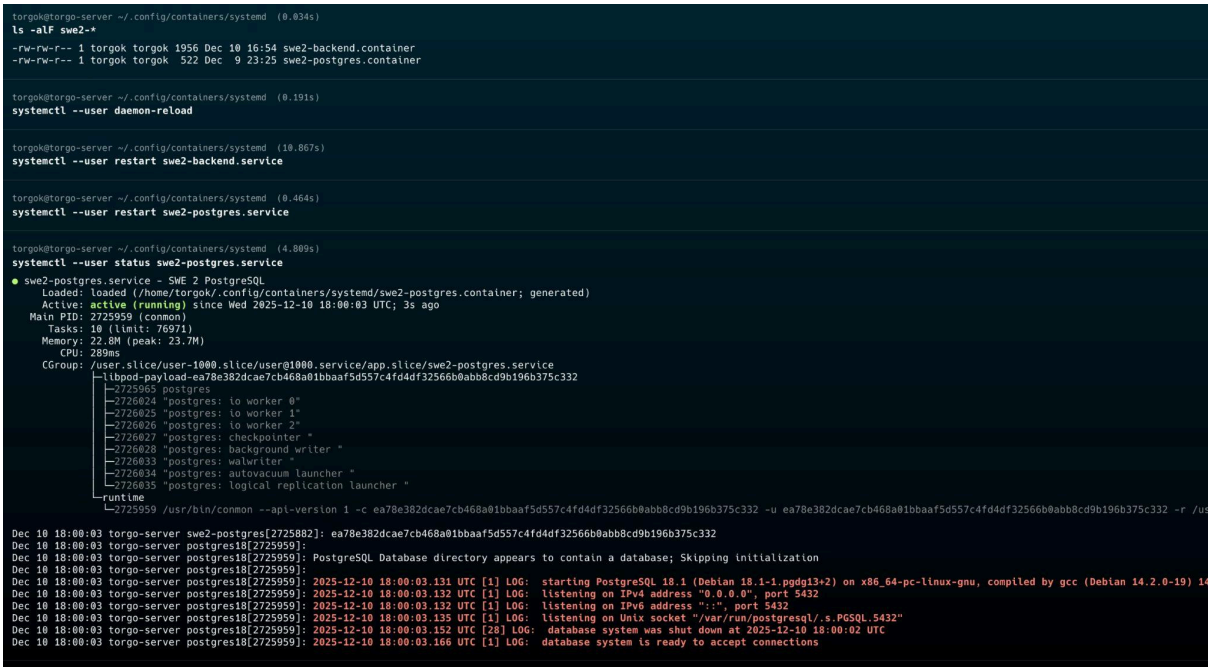
First, I ensured all dependencies were installed by running:

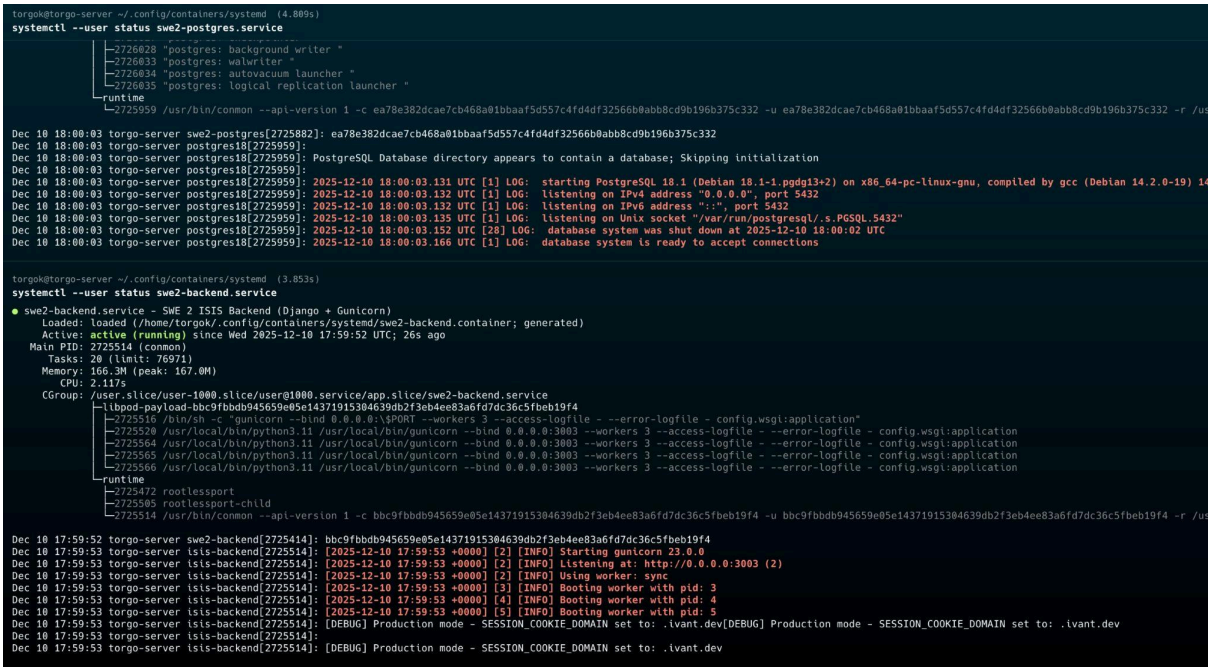
- This builds the backend images from the Dockerfile, ensuring all required dependencies and configurations are correctly integrated into the environment.



Starting the Services:

- Once the build process was completed:





With this, we would have the server deployed and running efficiently using Docker.

Bibliografía

1. American-Python Software Foundation. (n.d.). Python. https://www.python.org/

2. Django Software Foundation. (n.d.). Django documentation. https://docs.djangoproject.com/

3. Django REST Framework. (n.d.). Django REST framework. https://www.django-rest-framework.org/

4. django-allauth contributors. (n.d.). django-allauth documentation. https://django-allauth.readthedocs.io/

5. djangorestframework-simplejwt contributors. (n.d.). Simple JWT documentation. https://django-rest-framework-simplejwt.readthedocs.io/

6. PostgreSQL Global Development Group. (n.d.). PostgreSQL. https://www.postgresql.org/

7. Redis Ltd. (n.d.). Redis. https://redis.io/

8. Celery Project. (n.d.). Celery documentation. https://docs.celeryproject.org/

9. Amazon Web Services, Inc. (n.d.). AWS S3. https://aws.amazon.com/s3/

10. django-storages contributors. (n.d.). django-storages documentation. https://django-storages.readthedocs.io/

11. OAuth Working Group. (n.d.). OAuth 2.0. https://oauth.net/2/

12. OpenID Foundation. (n.d.). OpenID Connect. https://openid.net/

13. IETF. (n.d.). Transport Layer Security (TLS) — Internet standards. https://www.ietf.org/

14. React. (n.d.). React – A JavaScript library for building user interfaces. https://reactjs.org/

15. Vue.js. (n.d.). Vue.js. https://vuejs.org/

16. Vite. (n.d.). Vite — Next generation front-end tooling. https://vitejs.dev/

17. Webpack. (n.d.). Webpack. https://webpack.js.org/

18. Tailwind Labs. (n.d.). Tailwind CSS. https://tailwindcss.com/

19. Bootstrap. (n.d.). Bootstrap. https://getbootstrap.com/

20. Axios. (n.d.). Axios — Promise based HTTP client for the browser and node.js. https://axios-http.com/

21. Mozilla Developer Network. (n.d.). Fetch API. https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API

22. Google. (n.d.). Google Maps Platform. https://developers.google.com/maps

23. Django Software Foundation. (n.d.). GeoDjango — GIS framework for Django. https://docs.djangoproject.com/en/stable/ref/contrib/gis/

24. PostGIS Project Steering Committee. (n.d.). PostGIS — Spatial and Geographic Objects for PostgreSQL. https://postgis.net/

25. Docker, Inc. (n.d.). Docker documentation. https://docs.docker.com/

26. Docker, Inc. (n.d.). docker-compose documentation. https://docs.docker.com/compose/

27. Gunicorn contributors. (n.d.). Gunicorn — Python WSGI HTTP Server for UNIX. https://gunicorn.org/

28. Nginx, Inc. (n.d.). NGINX. https://nginx.org/

29. GitHub, Inc. (n.d.). GitHub Actions. https://docs.github.com/actions

30. pytest developers. (n.d.). pytest documentation. https://docs.pytest.org/

31. Django Software Foundation. (n.d.). Testing in Django. https://docs.djangoproject.com/en/stable/topics/testing/

32. Jest. (n.d.). Jest — Delightful JavaScript Testing. https://jestjs.io/

33. ECMAScript. (n.d.). ESLint — Pluggable JavaScript linter. https://eslint.org/

34. Prettier. (n.d.). Prettier — Code formatter. https://prettier.io/

35. Sentry. (n.d.). Sentry. https://sentry.io/

36. Prometheus Authors. (n.d.). Prometheus — Monitoring system & time series database. https://prometheus.io/

37. Grafana Labs. (n.d.). Grafana — The open observability platform. https://grafana.com/

38. Google. (n.d.). Google Identity — Google OAuth 2.0 login documentation. https://developers.google.com/identity

39. Firebase. (n.d.). Firebase Cloud Messaging. https://firebase.google.com/docs/cloud-messaging

40. Visual Studio Code. (n.d.). Visual Studio Code — Code editor. https://code.visualstudio.com/

41. Lucid Software Inc. (n.d.). Lucidchart. https://www.lucidchart.com/

42. The Markdown Guide. (n.d.). The Markdown Guide. https://www.markdownguide.org/

43. SPDX. (n.d.). SPDX — Software Package Data Exchange. https://spdx.org/licenses/