AI in Forest Management

Aliza Momysheva
Ariana Sadyr
Dulat Rakhymkul

Introduction & Problem Statement

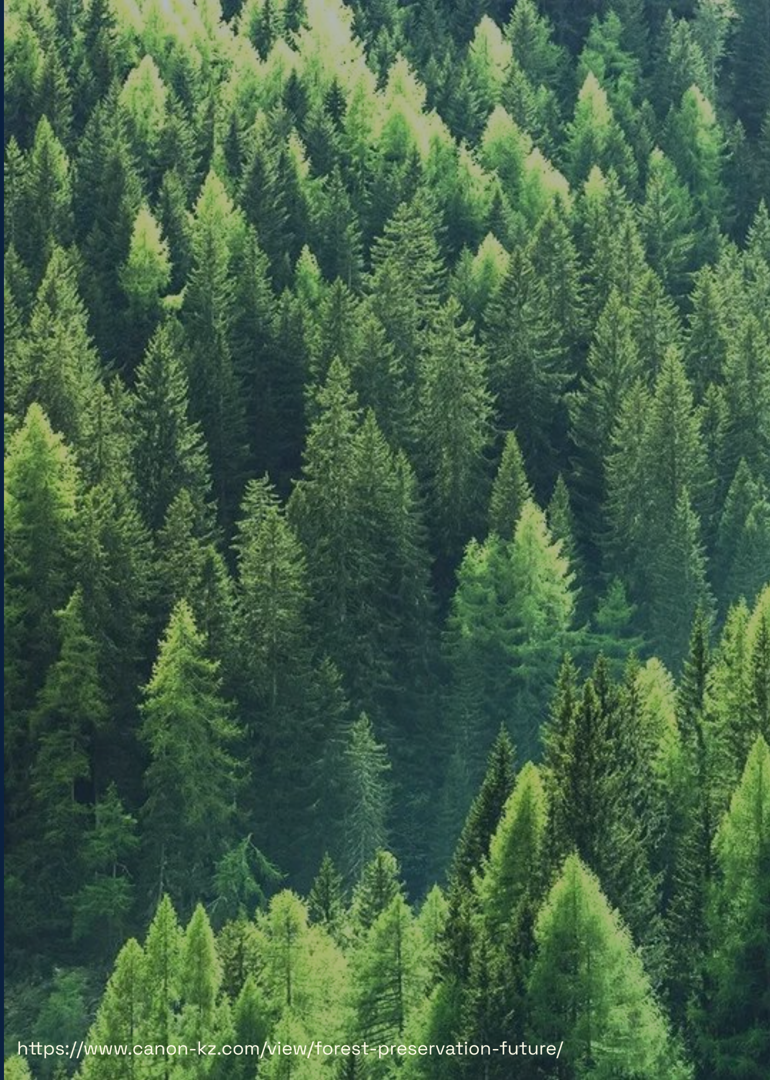
The Challenge

- Vast, remote, and under-monitored Kazakhstan forests
- No active projects in Kazakhstan
- No scalable, automated solution currently exists

Motivation

- Support ecological sustainability and climate resilience.
- Enable data-driven decisions for forest conservation.
- Bridge the data gap with an AI-powered, region-specific system.

Project Goal: Develop a mobile application that uses leveraging Satellite Imagery, Vegetative indices and a smart RAG-based chatbot (LLM) -> to monitor forest health, detect changes, and assist users in forest management tasks.



https://www.canon-kz.com/view/forest-preservation-future/

Overview of the product

Satellite-Based Forest Monitoring

- Biweekly dataset (2020–2025)
- 19 vegetative indices (e.g., NDVI, NBR, SAVI)

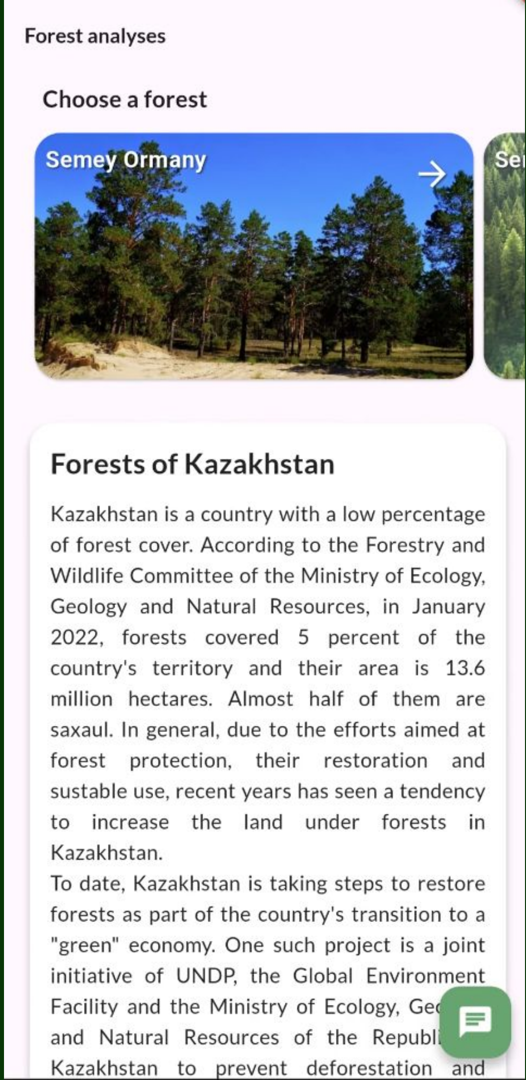
Mobile Application (Flutter + Django)
Visualizations: forest coverage masks, deforestation & fire events, time-series index graphs

RAG-Based AI Chatbot (Retrieval Augmented Generation)
Provides context-aware responses based on real project data for user assistance

Full Software Stack

Backend: Django REST API
Frontend: Flutter (cross-platform)
Storage: Google Drive + SQLite
Tools: Google Earth Engine, LangChain, Rasterio

Main page of the application



Dataset Preparation

Sentinel-2 and LANDSAT imagery were preprocessed through cloud filtering, temporal interpolation for gap filling, and reprojection to ensure consistency across datasets. Forest and burned area masks were created using NDVI and ΔNBR thresholds, with separate classification for clouds and water to reduce errors.

Study Area

- Four Forests from North and East Kazakhstan
- Timeline: 2020-2025

Data Sources

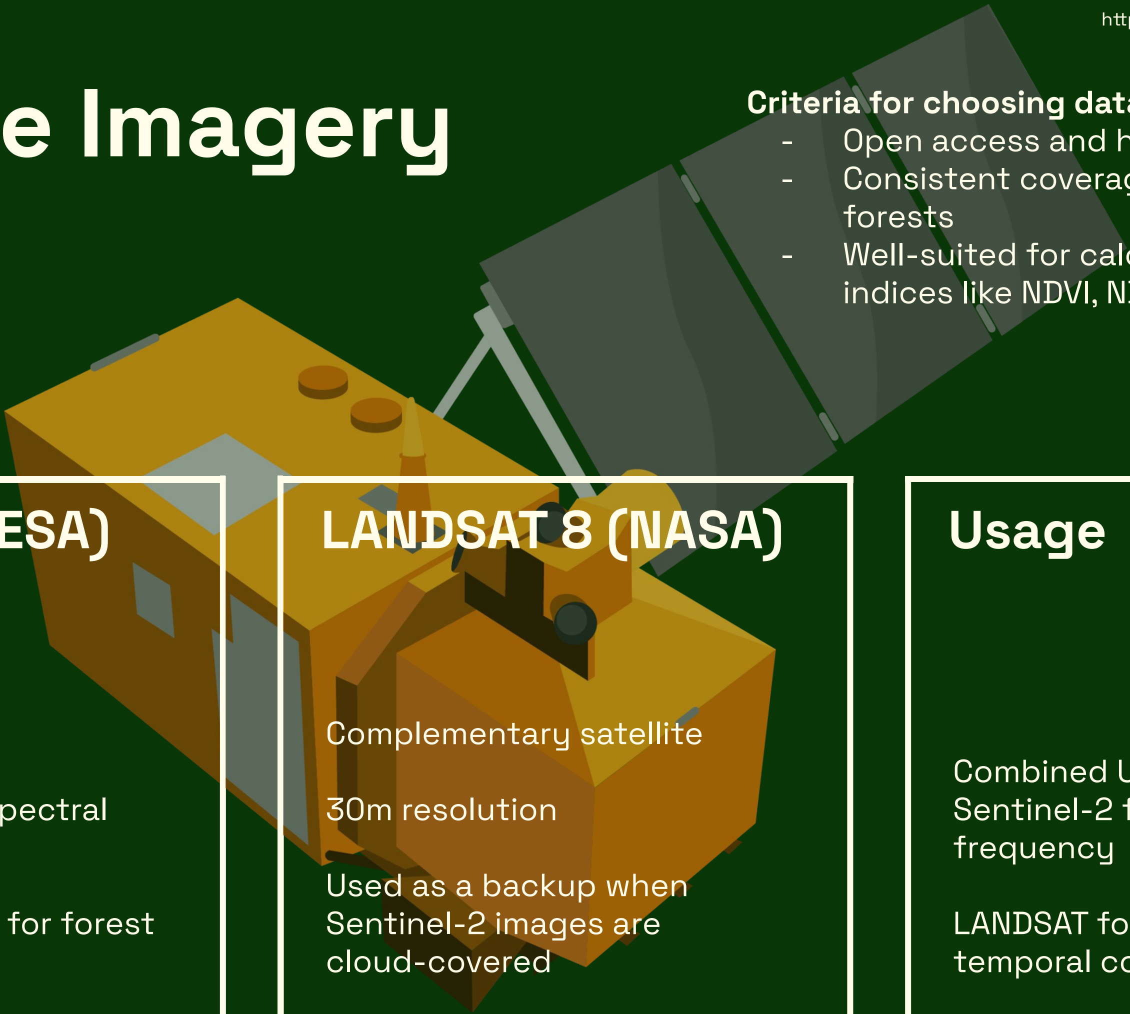
- COPERNICUS/S2_SR collection
- LANDSAT/LC08/C02/T1_L2 collection

Data Preprocessing

- Index Calculation
- Cloud Filtering
- Gap Filling

Sentinel-2 image:
https://blogs.fu-berlin.de/reseda/sentinel-2/

Satellite Imagery



Criteria for choosing data source:

- Open access and high availability
- Consistent coverage over Kazakhstan forests
- Well-suited for calculating vegetative indices like NDVI, NBR, NDWI, etc.

Sentinel-2 (ESA)

Main satellite

10m resolution, 13 spectral bands

Biweekly data used for forest health analysis

LANDSAT 8 (NASA)

Complementary satellite

30m resolution

Used as a backup when Sentinel-2 images are cloud-covered

Usage

Combined Usage
Sentinel-2 for detail and frequency

LANDSAT for gap filling and temporal consistency

Vegetative Indices

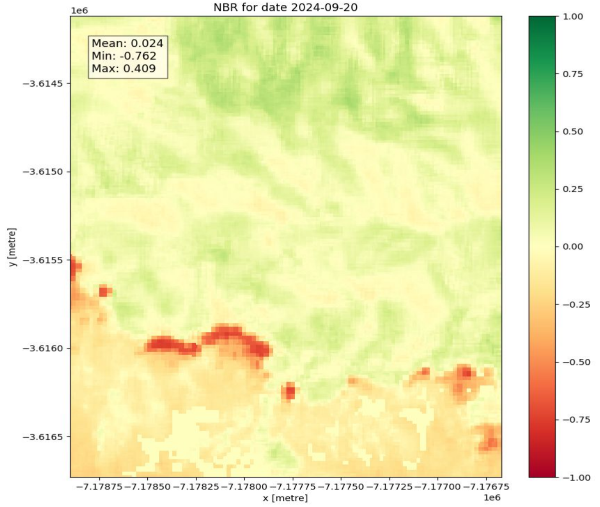
- Allow large-scale **monitoring of forest health.**
- Detect **deforestation**, **burn severity**, **moisture stress**, and **vegetation vigor**.
- Support **data-driven decisions** for sustainable forest management.

How We Use Them

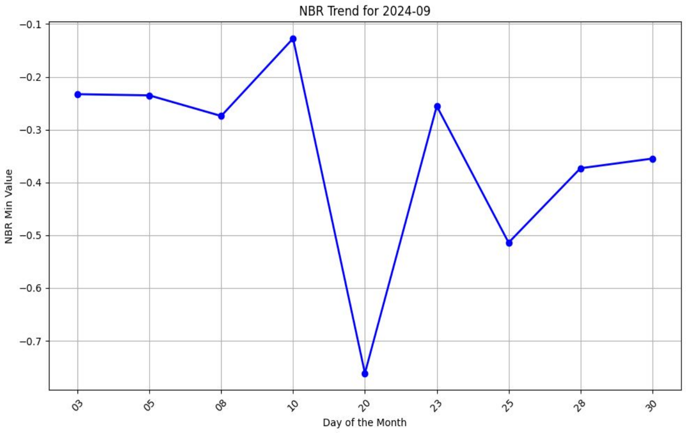
- Derived from **spectral bands** in satellite imagery.
- Used to generate **forest masks** (NDVI), track **temporal changes**, and fuel chatbot insights.

Examples of some vegetative indices (19 total)

Index	Full Name	Primary Use	Formula
NDVI	Normalized Difference Vegetation Index	General vegetation health and greenness	(NIR – R) / (NIR + R)
NBR	Normalized Burn Ratio	Burn severity and fire detection	(NIR − SWIR) ⁄ (NIR + SWIR)
SAVI	Soil Adjusted Vegetation Index	Vegetation in areas with exposed soil	(NIR − R) ⁄ (NIR + R + L) × (1 + L)



NBR Index Visualization



Forest Masks

Detection Method

Uses NDVI to detect vegetation:

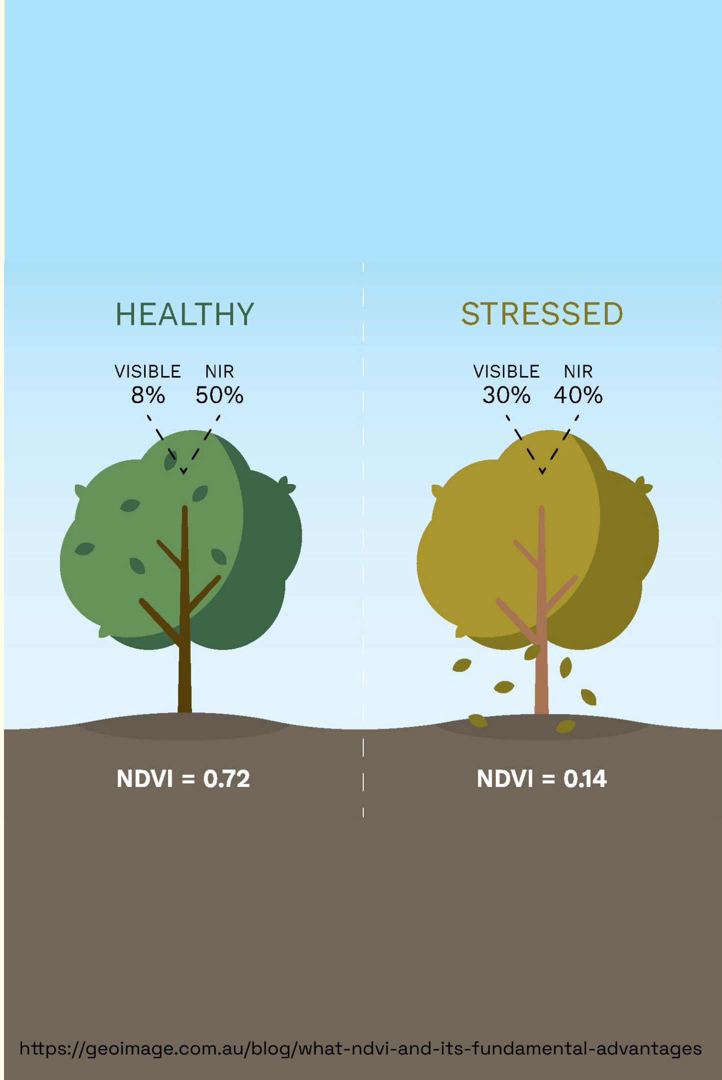
$$NDVI = (NIR - Red) / (NIR + Red)$$

Applied to Sentinel-2 and LANDSAT images

Classification Threshold

- Forest identified with NDVI between 0.2 and 0.3
- Based on observed vegetation patterns and research references

Three-class mask: 1 → Forest 2 → Cloud or Water 0 → Non-forest
Stored as binary mask images in the database



Burned Area Masks

Detection Method

Based on Normalized Burn Ratio (NBR):

$$NBR = (NIR - SWIR) / (NIR + SWIR)$$

Calculates change over time using:

$$\Delta NBR = NBR\_post - NBR\_pre$$

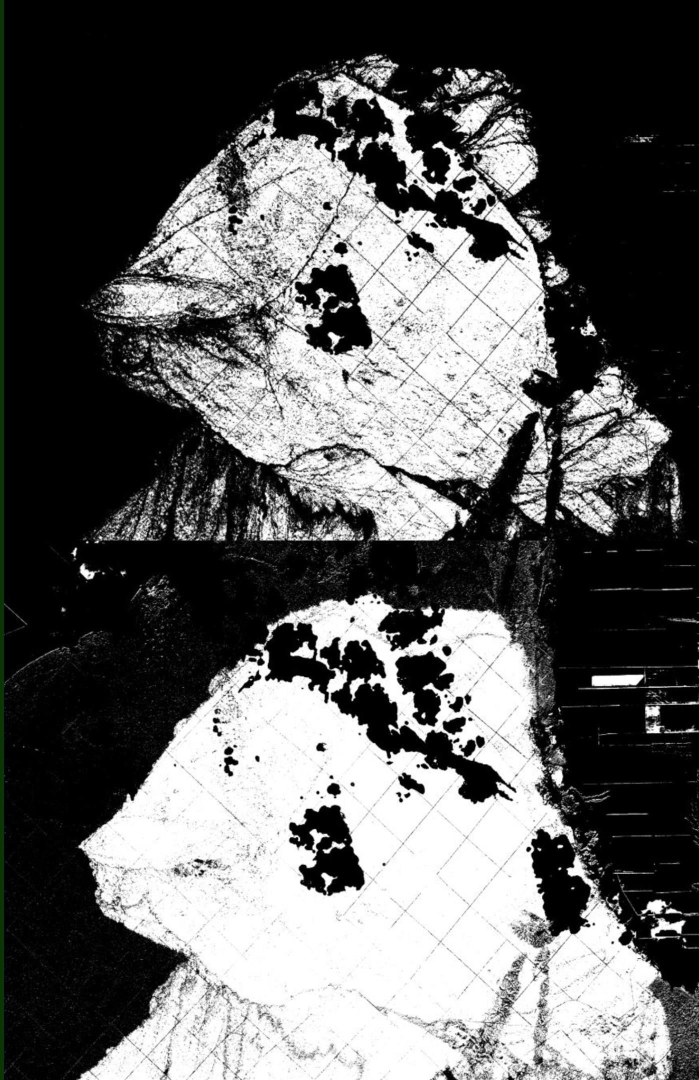
Threshold for Burn Detection

- Burned areas identified where $\Delta NBR < -0.10$

Binary mask: 1 = Burned 0 = Unburned

Applied to analyze **Semey Ormany** fire event because it has most of the historical data

Example of burned area masks



https://www.vecteezy.com/png/56486375-wildfire-action-in-forest-nature-scene-dramatic-view

Deforestation Masks

How It Works

- Compares two forest masks to detect land cover change
- Uses pixel-wise comparison with NumPy to highlight deforested areas

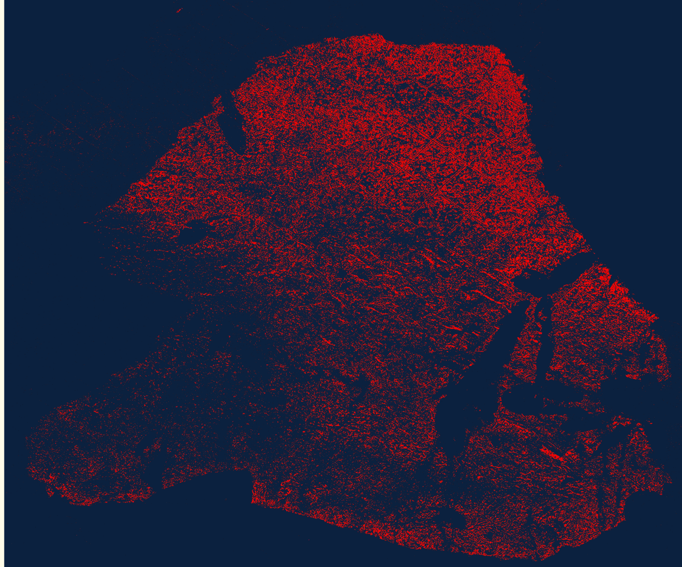
Deforested areas identified by calculating:

$$result = max(mask1, mask2) * 255$$

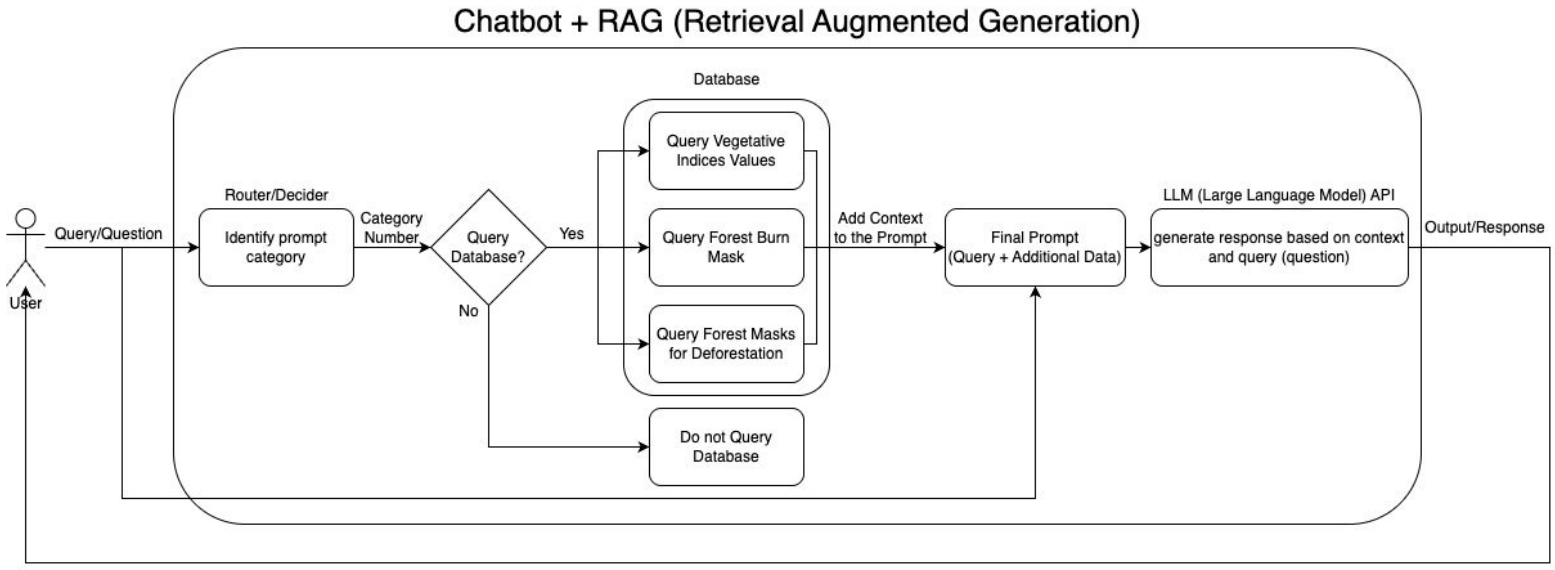
Why real time calculation?

- In case new forest masks added into database
- Expensive to store all combinations of forests masks for deforestation masks

Key Advantage: Enables real-time tracking of forest loss with high spatial resolution



Example of a deforestation mask



RAG-based Chatbot

Features: Uses structured queries instead of vector similarity for higher precision. Integrates with our forest masks and index datasets for context-aware answers. Delivers clear, domain-specific insights

Three-layer architecture:

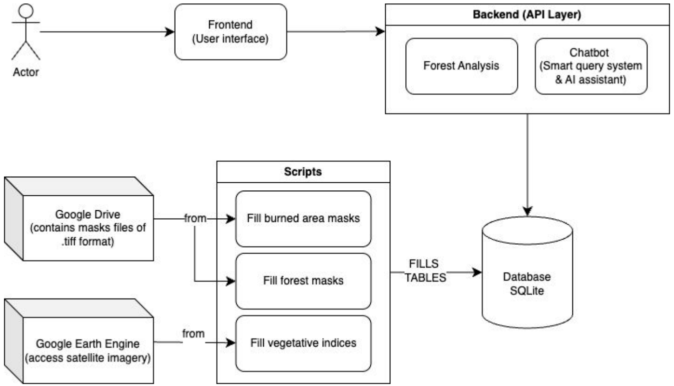
Router: Interprets user queries and selects relevant data

Context Builder: Extracts and formats info from the SQLite database

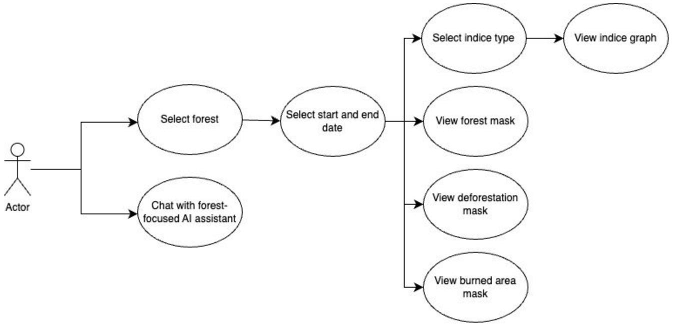
LLM API Layer: Sends enhanced prompt to GPT-4 and returns response

Backend

- **Framework:** Django + Django REST Framework – powers the API and handles user requests.
- **Data Sources:** Google Earth Engine used for real-time satellite data processing.
- **Geospatial Tools:** Rasterio to process and analyze .tif satellite images locally.
- **External APIs:**
  - OpenAI – for chatbot responses.
  - Google Drive – for storing and retrieving processed files.



Use case diagrams



Monolith Architecture with Modular Apps

Forest App: Handles satellite data processing and analysis. It includes endpoints for forest masks, vegetation indices, deforestation, and burned area masks. It also manages its own database tables for request logs and result storage.

Chatbot App: Manages the AI assistant functionality. It provides endpoints for sending/receiving messages and viewing chat history, and has its own database tables for storing conversations.

Backend

The backend uses a **relational database** (SQLite)

- **Chat Messages Table:**
 Stores each message exchanged between the user and the AI assistant.
- **Forest Metadata Table:**
 Can be used to store metadata about available forests, such as names, bounding boxes, and default dates.

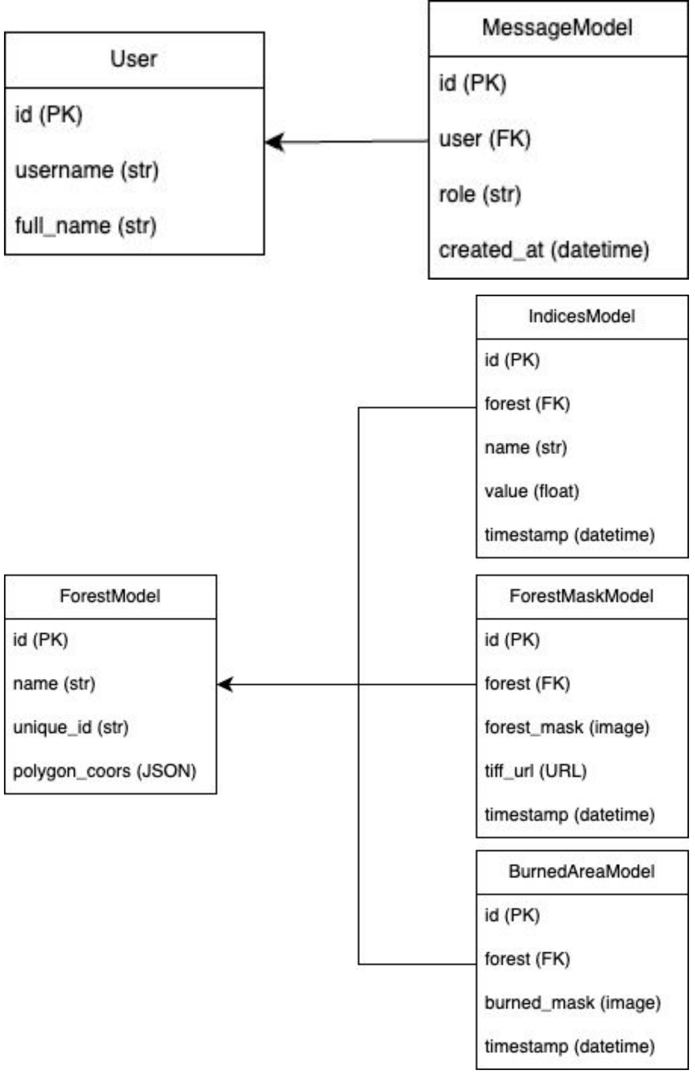
🌲 Forest Analysis API Endpoints

1. **get_forest_mask/**
 Returns the forest mask for a selected region and date — highlights vegetated areas using satellite imagery.
2. **get_forest_indices/**
 Computes and returns vegetation indices (e.g., NDVI, EVI) over a date range for a selected forest, allowing time-series analysis.
3. **get_burned_mask/**
 Detects and returns a mask highlighting burned areas within the selected region and timeframe.
4. **get_deforestation_mask/**
 Compares satellite images from two different dates and returns areas where deforestation occurred (vegetation changed to non-vegetation).

🤖 Chatbot API Endpoints

1. **chat/**
 Accepts user messages and returns responses from the forest-focused AI assistant, leveraging LLMs via LangChain.
2. **chat/history/**
 Retrieves the user's previous chatbot interactions for reference or review.

UML diagrams of database

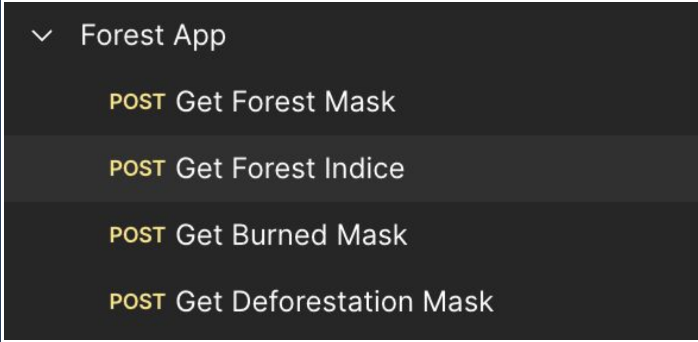


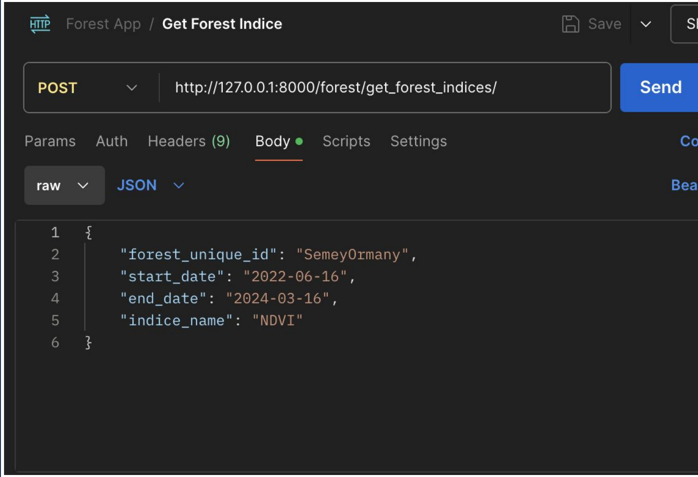
Testing

To verify that each API endpoint works correctly, we used **Postman**, a tool for sending HTTP requests to our backend. We tested endpoints like:

- GET /get_forest_mask/ – returns forest area as a visual mask
- GET /get_forest_indices/ – provides vegetation indices over time
- GET /get_burned_mask/ and /get_deforestation_mask/ – detect burned or deforested regions
- POST /chat/ – sends user messages to the chatbot and returns AI responses
- GET /chat/history/ – fetches previous user-AI interactions

We checked the response status codes, payload structure, and image data to ensure everything functioned as expected.





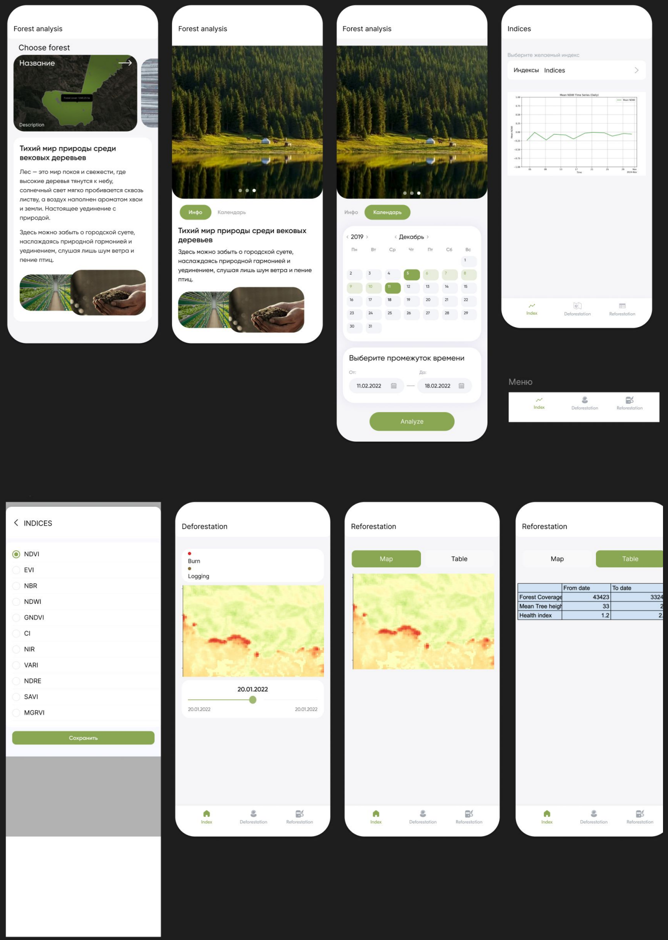
Frontend

The frontend is developed using **Flutter**, enabling a smooth, interactive experience across platforms. The interface combines UI and UX through a **swipeable image carousel** for forest previews and an **Info / Calendar toggle** that lets users switch between forest descriptions and a dynamic calendar view.

A **custom date-range picker** powered by **TableCalendar** enables users to select specific timeframes for analyzing changes in forest health. Interactive elements like **ElevatedButtons** manage state transitions, while dropdowns allow smooth navigation between months and years.

Thanks to Flutter's **hot-reload**, **Material design support**, and useful packages like google_fonts, we've created a visually rich, responsive, and user-friendly frontend—all written in **Dart** and deployable on web, Android, and iOS platforms.

Design plan



Demo

Google Earth Icon:
https://icon-icons.com/icon/google-earth-engine/104576

Dataset Evaluation

Box 1: Method

- Compared forest masks with **Google Earth imagery**
- Verified match between **classified forest areas** and **visible vegetation**
- Checked **cloud and water classification** for accuracy

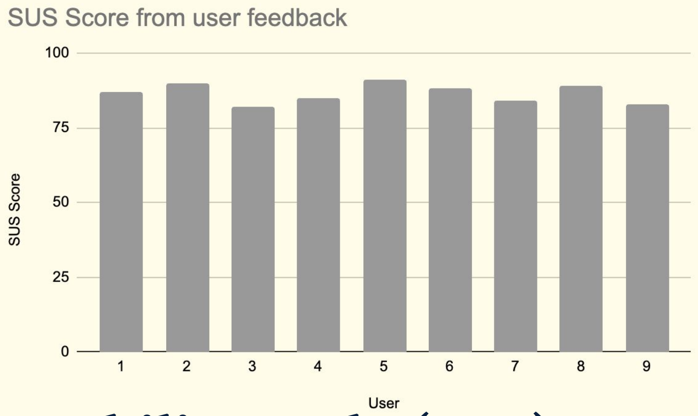
Box 2: Results

- High consistency between masks and real satellite images
- NDVI thresholding (0.2–0.3) effectively separated forest vs non-forest
- Gap-filled masks maintained reliable continuity

Box 3: Conclusion

- Forest masks are **accurate and visually validated**
- Dataset is suitable for **deforestation detection** and **index analysis**
- Supports confidence in system outputs for forest monitoring

System Performance Evaluation



System Usability Scale (SUS)

We conducted user testing with 9 participants, including computer science students and individuals from forestry and agriculture-related projects. They found the app intuitive, user-friendly, and praised its visuals, responsiveness, and chatbot functionality. The average **SUS** score was **86.6/100**, with most users rating the app highly for ease of use, clarity of visual data, and real-world applicability.

S1
I found the forest monitoring app easy to use

S2
The functions and features in the app were well integrated.

S3
I felt confident using the system without needing help.

S4
The information and visuals (like masks and graphs) were clear and understandable.

S5
I would use this kind of app in real-world forest or environmental monitoring.

System Performance Evaluation

Observations:

- Most pages load within **2–5 seconds**
- **Chatbot** responds in about 4 seconds.
- **Deforestation visualizations** 7–10 seconds (due to pixel-level comparisons of high-resolution masks).

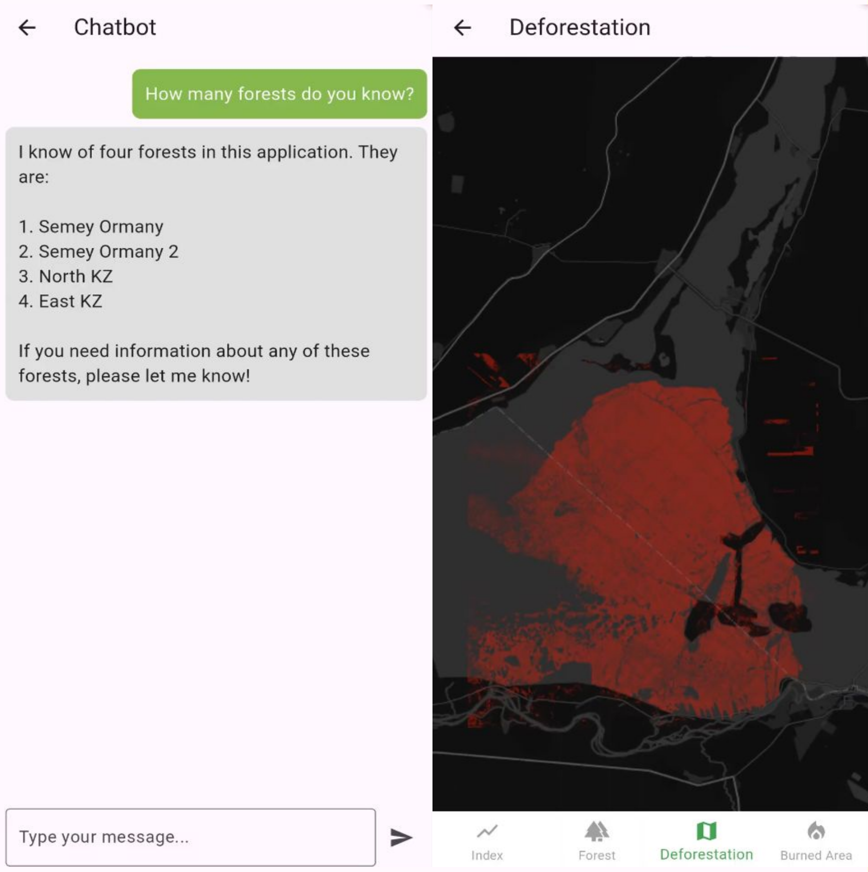
Tested with Android emulator.

Universally accepted metrics:

No universally mandated official standard for application response times

Some industry standards suggest -> up to 5 seconds is generally acceptable for most features (HeadSpin, OctoPerf).

Overall, the system is responsive, reliable, and well-optimized for real-world forest monitoring tasks.



User Interface

Future works

Expand monitoring to all of Kazakhstan’s forests.

Integrate weather data to improve analysis.

Develop predictive models for fire risk, deforestation, and vegetation change.

Deliver actionable insights for climate adaptation and forest policy.

Thank You