Project on earthquake monitoring dashboard

Program:

```
import math
import threading
from datetime import datetime, timezone
from typing import List, Tuple
import json
import os
from pathlib import Path
import numpy as np


import requests
import tkinter as tk
from tkinter import ttk

import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg


# =========================
# Configuration Constants
# =========================

USGS_URL = (

"https://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/2.5_day.geojson"
)

REQUEST_TIMEOUT_SECONDS = 10
MAX_TABLE_ROWS = 50

WINDOW_WIDTH = 1000
WINDOW_HEIGHT = 600

FIGURE_SIZE = (6, 6)

APP_DATA_DIR = Path(os.getenv("APPDATA", "."))
APP_DATA_DIR.mkdir(parents=True, exist_ok=True)
HISTORY_FILE = APP_DATA_DIR / "earthquake_history.json"
```

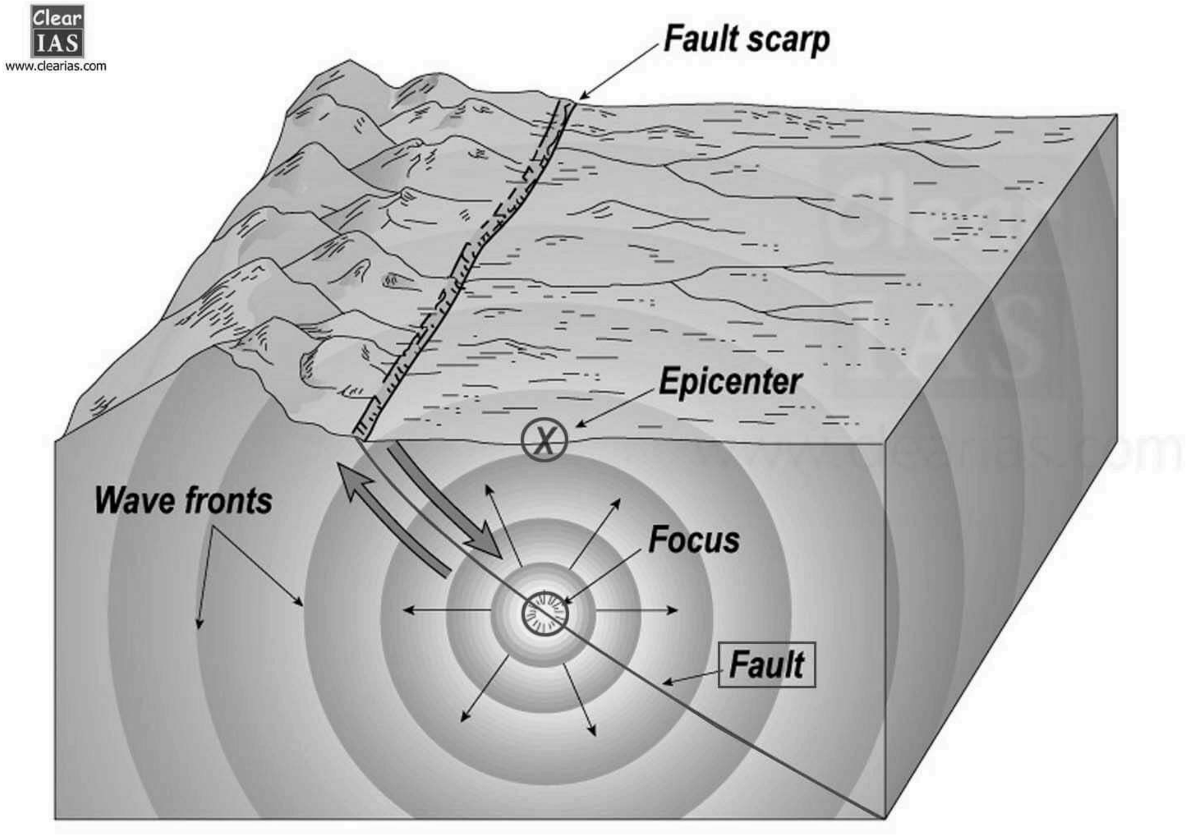
```python
# =========================
# Data Access Layer
# =========================

def fetch_earthquake_features() -> List[dict]:
    """
    Fetch earthquake data from USGS.
    Raises RuntimeError on failure.
    """
    try:
        response = requests.get(USGS_URL, timeout=REQUEST_TIMEOUT_SECONDS)
        response.raise_for_status()
        data = response.json()
        return data.get("features", [])
    except requests.RequestException as exc:
        raise RuntimeError("Failed to fetch earthquake data") from exc


# =========================
# Data Processing
# =========================

def parse_earthquakes(
    features: List[dict],
) -> Tuple[List[tuple], List[datetime], List[float], List[float]]:
    """
    Convert raw feature data into table rows and plotting series.
    """
    rows = []
    times = []
    magnitudes = []
    energies = []

    for feature in features:
        properties = feature.get("properties", {})
        geometry = feature.get("geometry", {})

        magnitude = properties.get("mag")
        coordinates = geometry.get("coordinates", [])

        if magnitude is None or len(coordinates) < 3:
            continue
        depth_km = coordinates[2]
        timestamp_ms = properties.get("time")
        place = properties.get("place", "Unknown location")

        if timestamp_ms is None:
            continue

        time_utc = datetime.fromtimestamp(
            timestamp_ms / 1000.0,
            tz=timezone.utc
        )

        rows.append(
            (
                time_utc,  # keep full datetime for sorting
                time_utc.strftime("%H:%M:%S"),
                f"{magnitude:.1f}",
                f"{depth_km:.1f}",
                place[:40],
            )
        )

        times.append(time_utc)
        magnitudes.append(magnitude)

        # Relative seismic energy (logarithmic, not absolute joules)
        energies.append(10 ** (1.5 * magnitude))

    return rows, times, magnitudes, energies


def load_history() -> dict:
    if not HISTORY_FILE.exists():
        return {}

    try:
        with HISTORY_FILE.open("r", encoding="utf-8") as f:
            return json.load(f)
    except (json.JSONDecodeError, OSError):
        return {}
```

```python
def save_history(history: dict) -> None:
    try:
        with HISTORY_FILE.open("w", encoding="utf-8") as f:
            json.dump(history, f)
    except OSError:
        pass


# =========================
# UI Layer
# =========================

class EarthquakeDashboard:
    def __init__(self, root: tk.Tk) -> None:
        self.history = load_history()
        self.root = root
        self._configure_window()
        self._build_layout()
        self.refresh_async()

    def _configure_window(self) -> None:
        self.root.title("Live Earthquake Dashboard")
        self.root.geometry(f"{WINDOW_WIDTH}x{WINDOW_HEIGHT}")

    def _build_layout(self) -> None:
        self._build_top_bar()
        self._build_main_area()

    def _build_top_bar(self) -> None:
        frame = tk.Frame(self.root)
        frame.pack(fill=tk.X, padx=5, pady=5)

        tk.Button(
            frame,
            text="Refresh Data",
            command=self.refresh_async,
        ).pack(side=tk.LEFT)

        self.status_label = tk.Label(frame, text="Loading...")
        self.status_label.pack(side=tk.RIGHT)

    def _build_main_area(self) -> None:
        main = tk.Frame(self.root)
        main.pack(fill=tk.BOTH, expand=True)

        self._build_table(main)
        self._build_charts(main)

    def _build_table(self, parent: tk.Widget) -> None:
        frame = tk.Frame(parent)
        frame.pack(side=tk.LEFT, fill=tk.Y)

        self.tree = ttk.Treeview(
            frame,
            columns=("time", "mag", "depth", "place"),
            show="headings",
            height=25,
        )

        for column in ("time", "mag", "depth", "place"):
            self.tree.heading(column, text=column.capitalize())

        self.tree.pack(fill=tk.Y)

    def _build_charts(self, parent: tk.Widget) -> None:
        frame = tk.Frame(parent)
        frame.pack(side=tk.RIGHT, fill=tk.BOTH, expand=True)

        self.figure, (self.ax_mag, self.ax_energy) = plt.subplots(
            2,
            1,
            figsize=FIGURE_SIZE,
        )

        self.canvas = FigureCanvasTkAgg(self.figure, master=frame)
        self.canvas.get_tk_widget().pack(fill=tk.BOTH, expand=True)

    # ========================
    # Refresh Logic
    # ========================

    def refresh_async(self) -> None:
        threading.Thread(
            target=self._refresh_data,
            daemon=True,
        ).start()
```

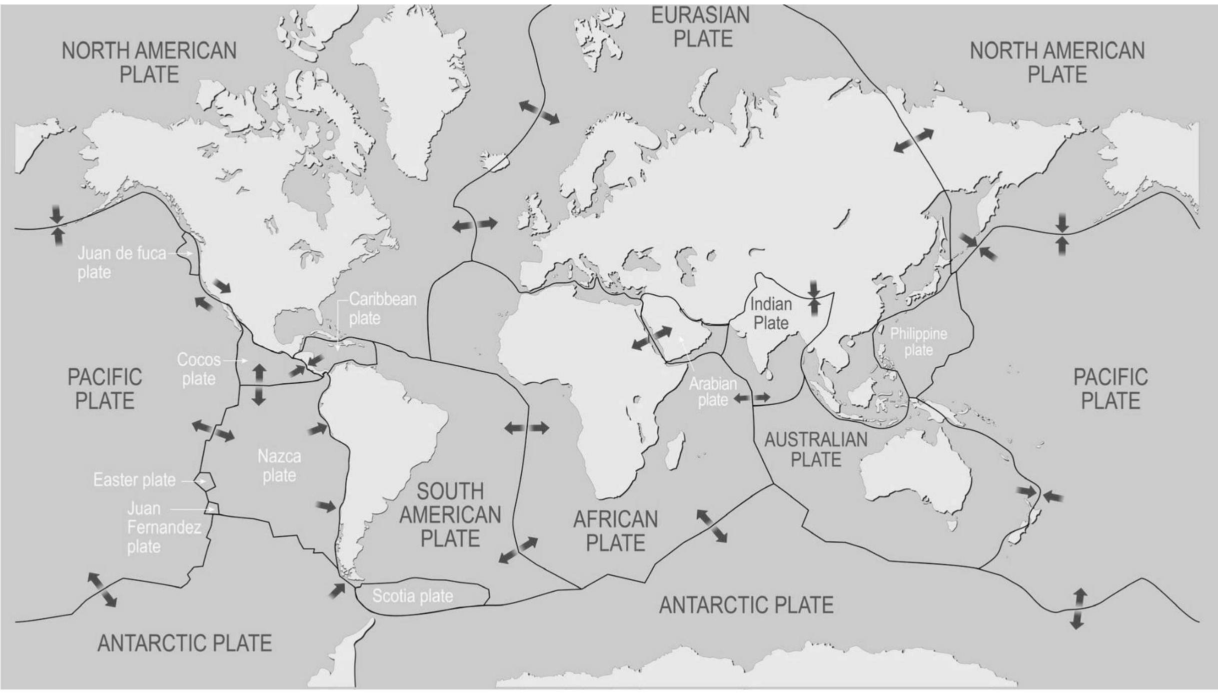
```
def _refresh_data(self) -> None:
    try:
        features = fetch_earthquake_features()

        for feature in features:
            event_id = feature.get("id")
            if event_id:
                self.history[event_id] = feature

        save_history(self.history)

        all_features = list(self.history.values())

        rows, times, mags, energies = parse_earthquakes(all_features)

        self.root.after(
            0,
            self._update_ui,
            rows,
            times,
            mags,
            energies,
        )
    except RuntimeError:
        self.root.after(
            0,
            lambda: self.status_label.config(
                text="Failed to fetch data"
            ),
        )

def _update_ui(
    self,
    rows: List[tuple],
    times: List[datetime],
    mags: List[float],
    energies: List[float],
) -> None:
    self.tree.delete(*self.tree.get_children())

    rows.sort(key=lambda r: r[0], reverse=True)
```

```
        for row in rows[:MAX_TABLE_ROWS]:
            _, time_str, mag, depth, place = row
            self.tree.insert("", tk.END, values=(time_str, mag, depth,
place))

        self.ax_mag.clear()
        self.ax_mag.scatter(times, mags, alpha=0.7)
        self.ax_mag.set_title("Magnitude vs Time (UTC)")
        self.ax_mag.set_ylabel("Magnitude")

        self.ax_energy.clear()

        if energies:
            emin = min(energies)
            emax = max(energies)

            if emin > 0:
                log_bins = np.logspace(
                    math.log10(emin),
                    math.log10(emax),
                    30
                )
                self.ax_energy.hist(energies, bins=log_bins)
                self.ax_energy.set_xscale("log")
                self.ax_energy.set_title("Energy Distribution (Log Scale)")
                self.ax_energy.set_xlabel("Energy Index")
                self.ax_energy.set_ylabel("Count")
            else:
                self.ax_energy.set_title("Energy Distribution (No Data)")

        self.figure.tight_layout()
        self.canvas.draw()

        self.status_label.config(
            text=f"Last updated:
{datetime.now(timezone.utc).strftime('%H:%M:%S')} UTC"
        )


# ========================
# Entry Point
# ========================
def main() -> None:
    root = tk.Tk()

    def on_close() -> None:
        root.quit()
        root.destroy()

    root.protocol("WM_DELETE_WINDOW", on_close)

    EarthquakeDashboard(root)
    root.mainloop()


if __name__ == "__main__":
    main()
```

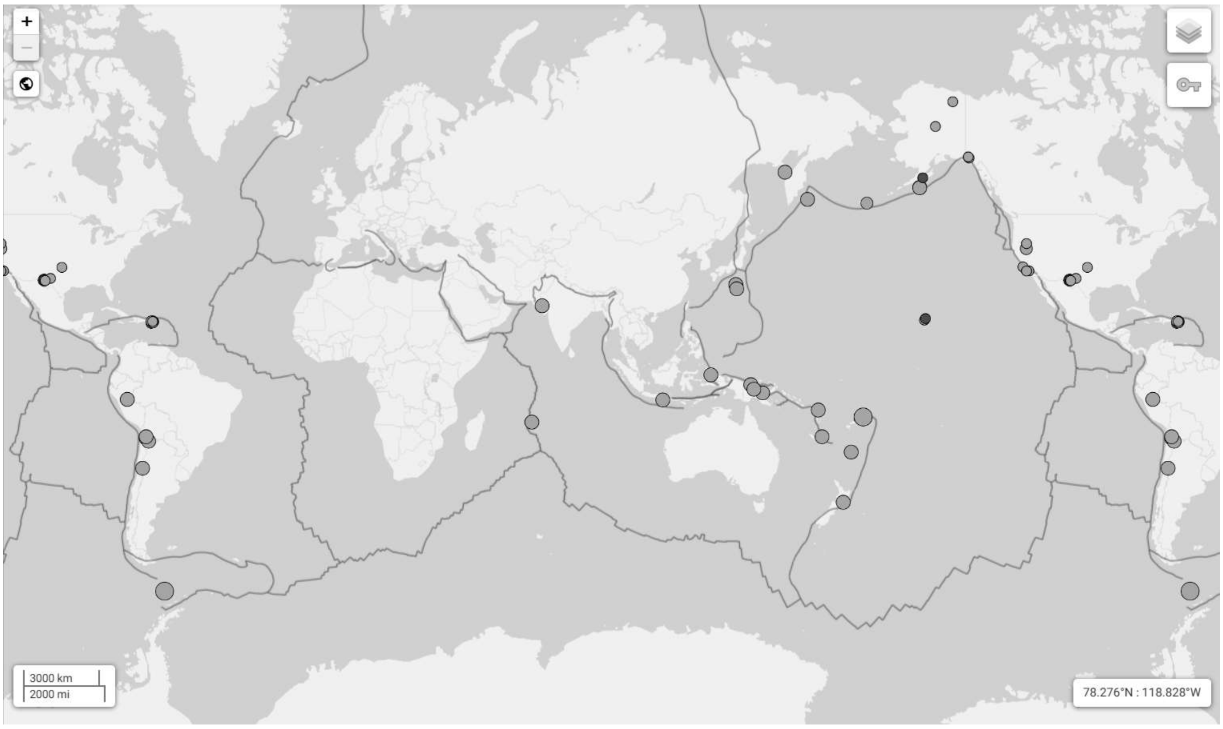
Representation of depth:



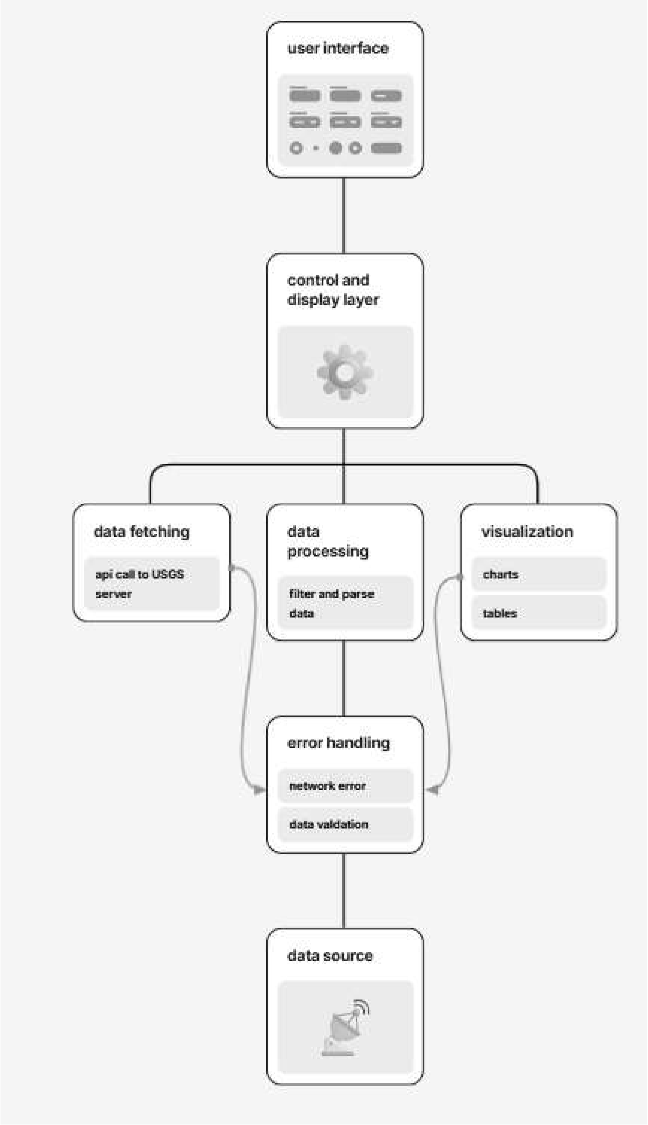
Tectonic plates



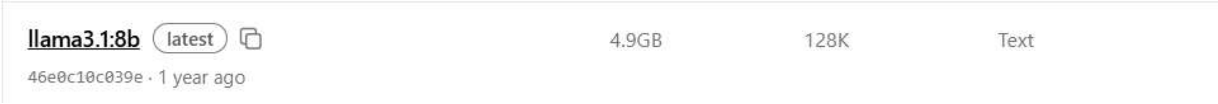
This is the data to prove that most earthquakes emerge from edges of tectonic plates



This is the sequence of functions performed in background



Url of the source https://www.usgs.gov/
My github repo https://github.com/0xmanidev/earthquake-dashboard
Local ai that used as help while building



llama3.1:8b latest 4.9GB 128K Text
46e0c10c039e · 1 year ago

External libraries used

- requests
- matplotlib
- numpy

Package that is used to compile the exe
Pyinstaller
Package manager
Pip
Python version
Python3.14

Learnt things

Types of seismic waves

- **P-waves (Primary)**
  Fastest. Compressional. First to arrive. Travel through solids and liquids.

- **S-waves (Secondary)**
  Slower. Shear motion. Only move through solids.

- **Surface waves**
  Slow, messy, destructive. These wreck buildings and ruin days.

Uses and classification of ai modals and classes and libraries used to directly access system i.e,os library

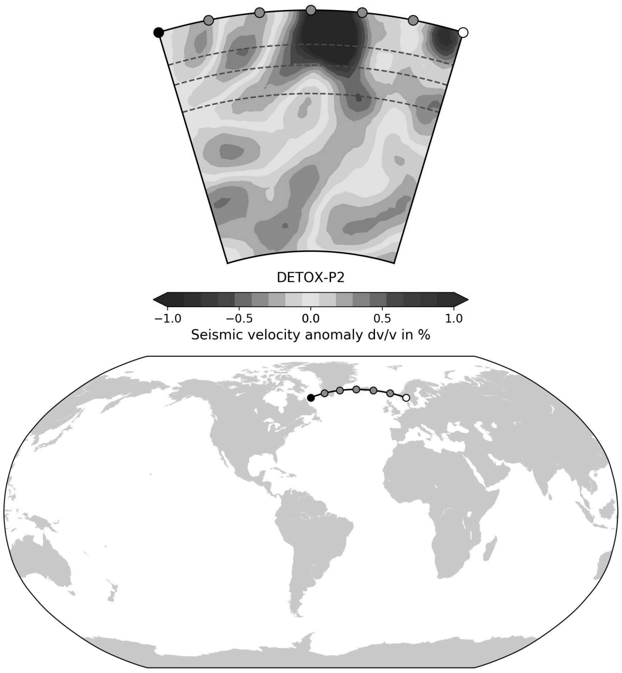
Epicenter: where the quake started

Depth: shallow quakes do more damage

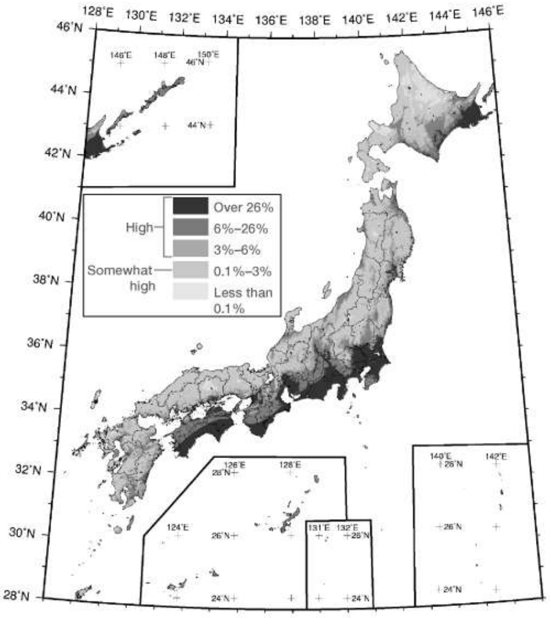
Magnitude: how much energy was released

Fault type: strike-slip, normal, reverse

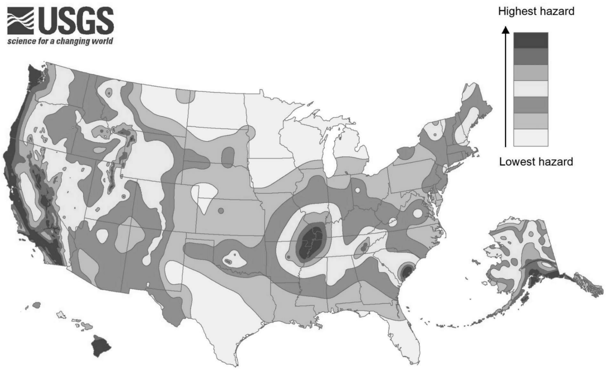
Aftershock patterns: stress redistribution, not chaos



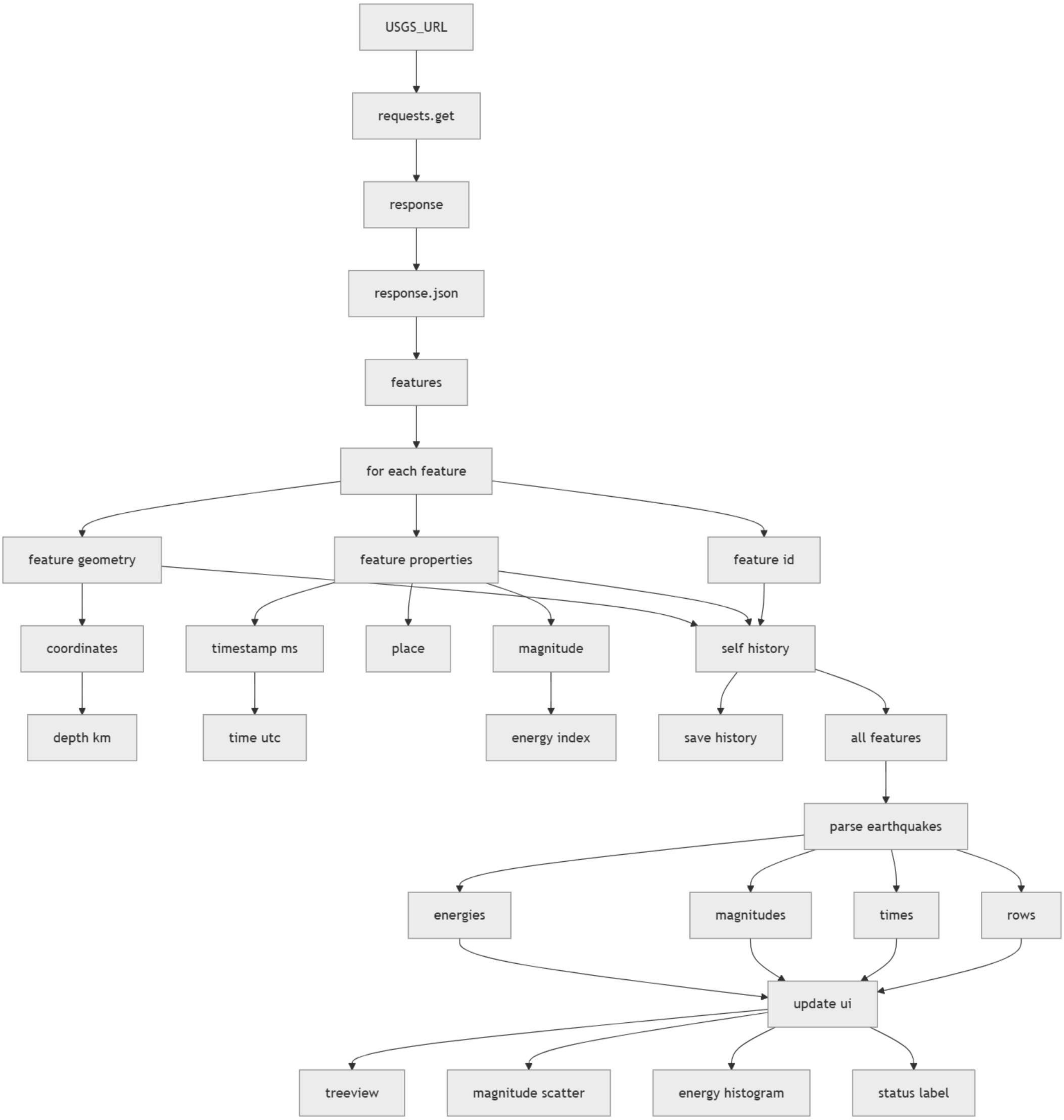
Probability of Experiencing an Earthquake of Lower 6 Intensity or Above in the Next 30 Years

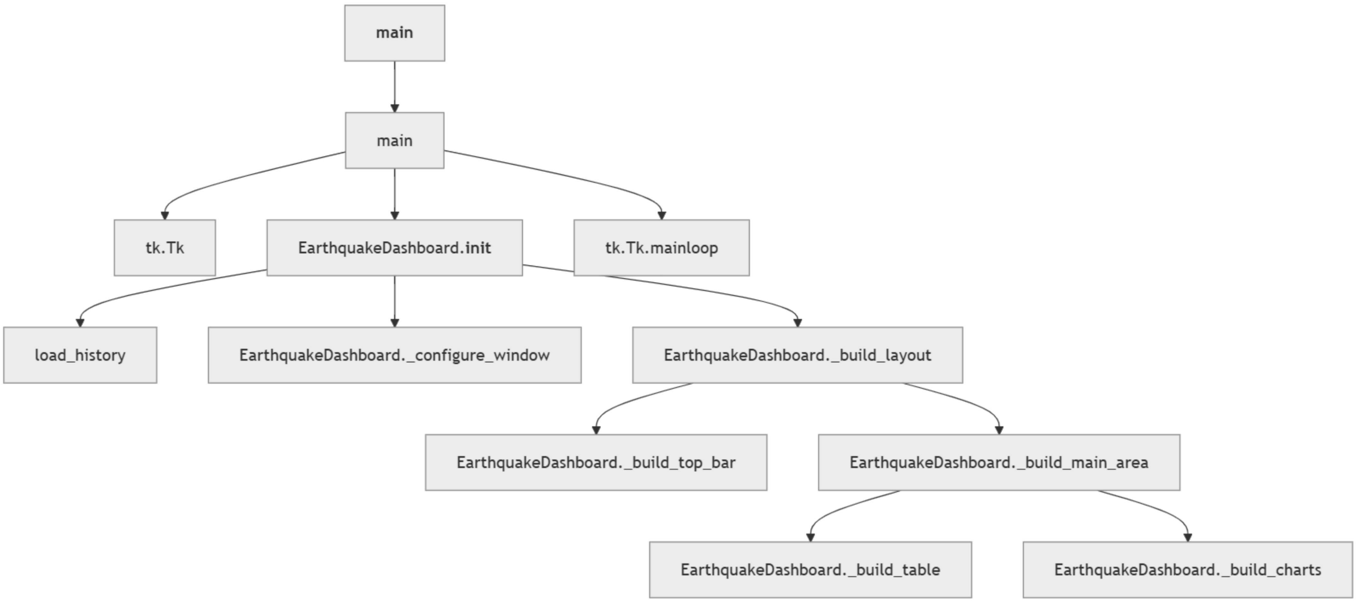


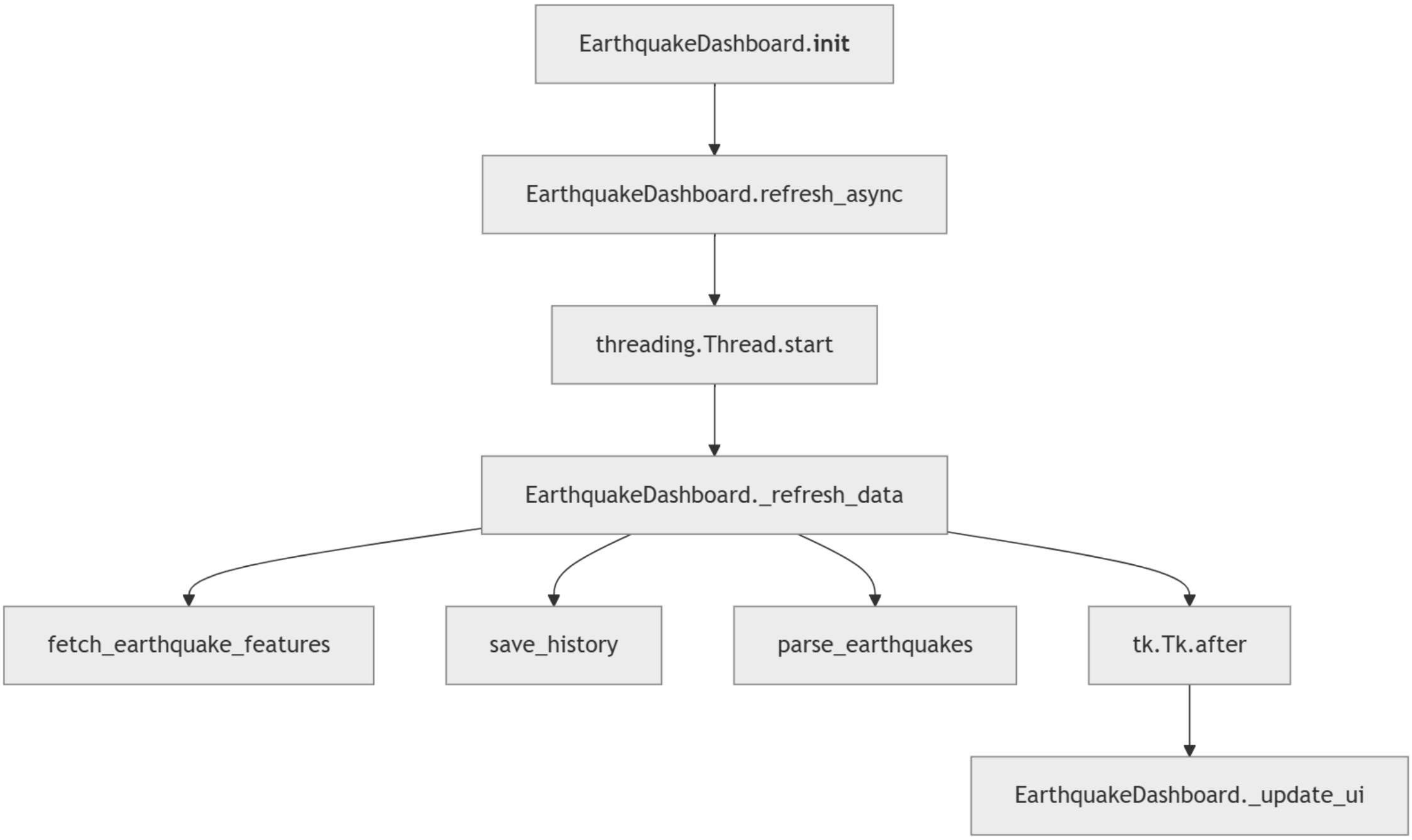
Source: Headquarters for Earthquake Research Promotion.



earthquake predictions


USGS_URL
requests.get
response
response.json
features
for each feature
feature geometry
feature properties
feature id
coordinates
timestamp ms
place
magnitude
self history
depth km
time utc
energy index
save history
all features
parse earthquakes
energies
magnitudes
times
rows
update ui
treeview
magnitude scatter
energy histogram
status label


main
main
tk.Tk
EarthquakeDashboard.**init**
tk.Tk.mainloop
load_history
EarthquakeDashboard._configure_window
EarthquakeDashboard._build_layout
EarthquakeDashboard._build_top_bar
EarthquakeDashboard._build_main_area
EarthquakeDashboard._build_table
EarthquakeDashboard._build_charts


EarthquakeDashboard.**init**
EarthquakeDashboard.refresh_async
threading.Thread.start
EarthquakeDashboard._refresh_data
fetch_earthquake_features
save_history
parse_earthquakes
tk.Tk.after
EarthquakeDashboard._update_ui

System Architecture Overview

The application follows a layered architecture to separate responsibilities and improve maintainability:

- **Data Access Layer**
  Responsible for fetching live earthquake data from the USGS API using HTTP requests.

- **Data Processing Layer**
  Filters, validates, and transforms raw earthquake data into structured formats suitable for display and visualization.

- **Persistence Layer**
  Stores previously fetched earthquake data locally in JSON format to maintain historical records and prevent data loss.

- **User Interface Layer**
  Built using Tkinter, responsible for displaying tabular data, charts, and status updates.

- **Concurrency Model**
  Uses background threads for network operations while ensuring all UI updates occur safely on the Tkinter main thread.

Threading and UI Synchronization

To maintain UI responsiveness and avoid race conditions:

- All network and data-fetching operations are executed in a background thread.

- Tkinter UI components are **never updated directly from worker threads**.

- The `root.after()` mechanism is used to safely schedule UI updates on the main thread.

This design ensures thread safety and prevents application freezes or crashes.

Error Handling and Fault Tolerance

The application includes robust error-handling mechanisms:

- Network failures during API calls are caught and handled gracefully.
- Invalid or corrupted JSON history files are safely ignored.
- User feedback is provided via status messages when data fetching fails.
- The application continues running even after recoverable errors.

Data Constraints and Assumptions

Aspect	Assumption
Magnitude	Events with missing magnitude values are ignored
Coordinates	At least depth information must be present
Energy Calculation	Uses a relative logarithmic scale, not absolute joules
History Storage	Local JSON file may be empty or corrupted