Wildfire Detection Project — README

Contents

1 Introduction

The Wildfire Detection Project is a comprehensive system for detecting wildfires using a combination of a web application and machine learning modules. The project comprises three primary components:

- **Back_End** — A Python-based API (using Flask or FastAPI) for handling server-side logic and database interactions.
- **Front_End** — A modern web interface built with technologies such as React, Vite, and Tailwind CSS.
- **Machine_Learning** — A module responsible for training and inference of machine learning models used in wildfire detection.

As illustrated in Figure 1, the **Front_End** sends images (IMG) to the **Back_End**, which processes requests and interacts with the **DB** (database). The **AI Model** receives these images for wildfire detection, returning results back to the **Back_End**, which then relays the outcome to the user interface. This architecture streamlines the workflow of uploading, analyzing, and storing wildfire detection data.

This document provides an extensive overview of the project's directory structure, describes the purpose of each folder, and details the setup and installation instructions for all components.

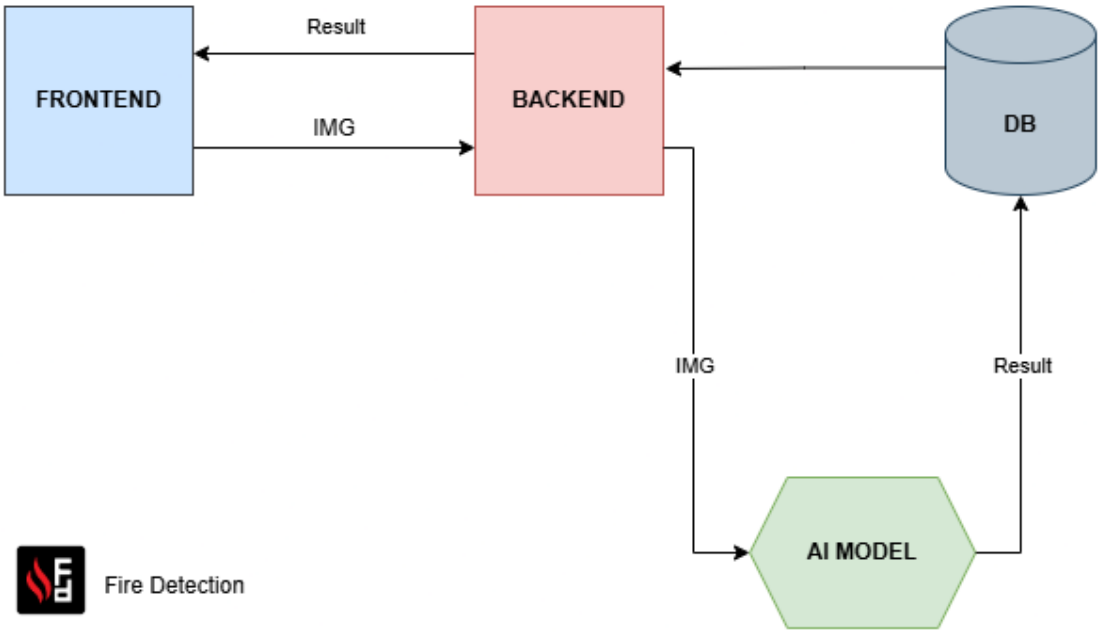

Figure 1: High-level architecture of the Wildfire Detection system

2 Project Root

The project root contains the following files and directories:

- **docker-compose.yml** - YAML configuration file for Docker Compose.
- **.env** - Environment variables file.
- **requirements.txt** - Lists dependencies for the project.
- **package-lock.json** - Lock file for package management.
- **.gitignore** - Specifies files and directories to be ignored by Git.
- **.gitattributes** - Configures Git attributes for the repository.
- **LICENSE** - License file for the project.
- **machine_learning/** - Directory for machine learning components.
- **database_creation/** - Directory related to database creation.
- **.git/** - Git repository folder.
- **front_end/** - Directory for front-end development.
- **database/** - Directory for database files and scripts.
- **.venv/** - Virtual environment directory.

3 Directory Structure Overview

The repository is organized into several directories and files to separate concerns and ease development:

- **.git/** — Git repository metadata and version control.
- **.venv/** — Virtual environment for managing dependencies.

4 Component Breakdown

4.1 Back_End

Overview

The back-end is built using Python (Flask or FastAPI) and is responsible for handling API routes, configuration, and database operations.

Structure

- `instance/` — Stores instance-specific configurations.
- `uploads/` — Stores files uploaded by users.
- `app.py` — Main application entry point.
- `config.py` — Configuration settings.
- `models.py` — Database models.
- `routes.py` — API routes.
- `requirements.txt` — List of Python dependencies.
- `database_creation/` — Scripts and tools for setting up the database.
- `docker/` — Docker configuration files and deployment scripts.

4.2 Database

The database directory contains the following:

- **.dockerignore** - Specifies files to be ignored by Docker.
- **.env** - Environment variables for database configuration.
- **Dockerfile** - Dockerfile for setting up the database container.
- **init.sql** - SQL script for initializing the database.

4.3 Front_End

Overview

The front-end provides a user-friendly interface for interacting with the wildfire detection system. It is built using modern web development technologies.

Structure

- `node_modules/` — Contains installed dependencies.
- `public/` — Static files (images, icons, etc.).
- `src/` — Source code of the front-end application.
- `index.html` — Main HTML file.
- `package.json` — Lists project dependencies and scripts.
- `package-lock.json` — Dependency tree lock file.
- `powershell_launch_frontend.ps1` — PowerShell script to launch the front-end.
- `tailwind.config.js` — Tailwind CSS configuration.
- `vite.config.js` — Vite configuration.
- `.gitignore` and `eslint.config.js` — Files to enforce code quality and version control settings.

4.4 Machine Learning

The machine learning directory contains the following:

- **app/** - Application-related scripts and modules.
- **graphs/** - Visualization and plotting scripts.
- **instance/** - Configuration and instance-specific files.
- **models/** - Trained models and model-related scripts.
- **processed_images/** - Preprocessed images used in training.
- **test_images/** - Images used for model testing.
- **training_models_scripts_kaggle/** - Scripts related to training models from Kaggle datasets.
- **uploads/** - Directory for uploaded datasets or images.
- `preprocessing.py` — Python script for preprocessing data.
- `config.py` — Configuration script for machine learning settings.
- `run.py` — Main script for running machine learning models.

Additional Structure Details

- `app/` — Main application logic.
- `graphs/` — Visualizations and performance plots.
- `instance/` — Instance-specific configurations.
- `models/` — Trained machine learning models.
- `scripts/` — Data processing and utility scripts.

- `test_images/` — Test dataset images.
- `config.py` — Configuration settings.
- `link_to_preprocessed_dataset.txt` — Link to the preprocessed dataset.
- `powershell_launch_backend.ps1` — PowerShell script to launch the ML service.

5 Additional Files and Tools

The repository also contains several additional files and folders that support development and deployment:

- `powershell_commands/` — PowerShell scripts for automation and system configuration.
- `.gitattributes` — Git attribute definitions (e.g., line endings, file types).
- `.gitignore` — Lists files and directories to be ignored by Git.
- `.Rhistory` — Stores command history for R (if applicable).
- `docker_app.exe` — Executable related to Docker deployments.
- `launch_log.txt` — Log file tracking application launch details.
- `LICENSE` — The projects license file.
- `package-lock.json` — Ensures consistent package installations.

6 Running the App

The architecture uses Docker Compose to manage a multi-container setup, which includes a backend, frontend, and database. Each component runs inside a separate container as defined in the `docker-compose.yml` file.

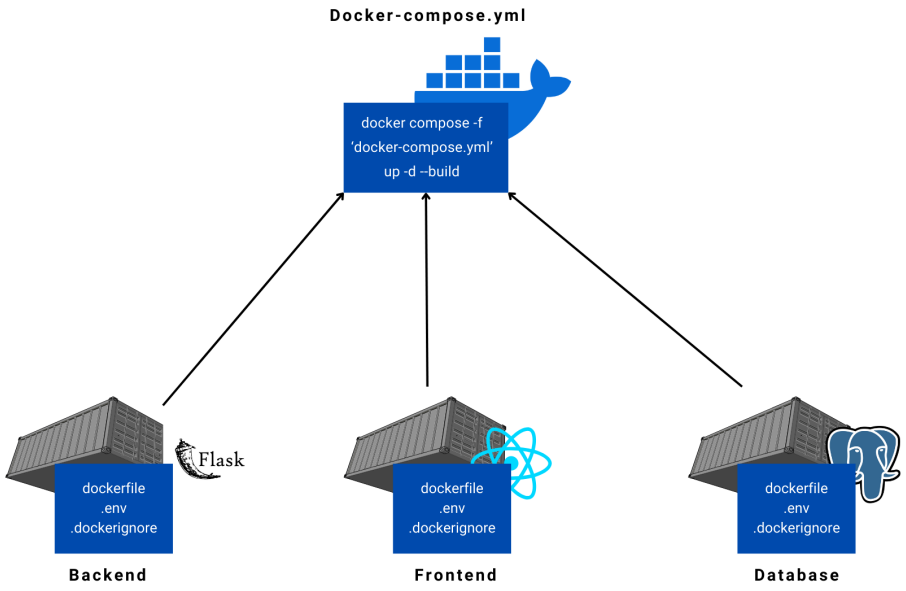

Figure 2: Docker Compose architecture for the application

To build and run all services, execute the following command:

```
docker compose -f 'docker-compose.yml' up -d --build
```

After building, you can start the containers using:

```
docker-compose up --build
```

To stop the running containers, use:

```
docker compose down
```

For restarting containers when they are already built, run:

```
docker-compose up
```

6.1 Installation and Running in Docker

This subsection outlines the complete installation and running process for the Docker containers. Ensure Docker is installed and configured on your system before executing the commands above.

Note: The entire process is expected to take approximately 20 minutes on a standard system.

7 Summary

This document provides a comprehensive overview of the Wildfire Detection Project. It details the directory structure, the purpose of each module (Back_End, Front_End, and Machine_Learning), and the necessary steps for setup and deployment. Whether you are managing server-side logic, front-end development, or machine learning workflows, this guide should assist you in understanding and deploying the project effectively.