Flood Watch System Documentation

1. Overview

The Flood Watch System is a comprehensive web application for monitoring and forecasting flood events. It consists of a Django backend and a React frontend, containerized with Docker for easy deployment.

2. System Architecture

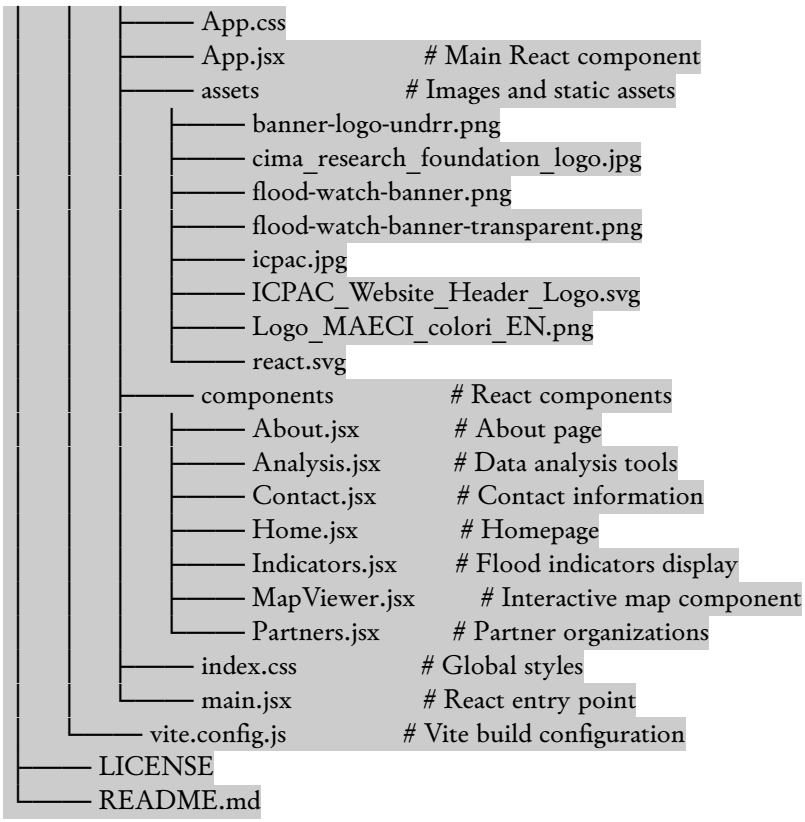
The system is divided into several containerized components:

- **Backend**: Django-based API server handling data processing, management, and analysis
- **Frontend**: React-based user interface for visualization and interaction
- **GeoServer**: Manages and serves geospatial data for visualization and analysis
- **PostgreSQL/PostGIS**: Database system with spatial extensions for storing geographic data
- **Celery**: Distributed task queue for handling asynchronous processes and background tasks
- **Redis**: In-memory data store used as a message broker for Celery and for caching

All components are containerized using Docker for ease of deployment, and communication between components occurs via RESTful APIs and message queues.

3. Directory Structure

```
.
├── backend
│   ├── Dockerfile
│   ├── flood_watch_system          # Django project directory
│   │   ├── asgi.py
│   │   ├── __init__.py
│   │   ├── settings.py             # Project configuration
│   │   ├── urls.py                 # URL routing
│   │   └── wsgi.py
│   ├── Impact                      # Main Django app
│   │   ├── admin.py                # Admin interface configuration
│   │   ├── apps.py
│   │   ├── __init__.py
│   │   ├── management              # Custom management commands
│   │   │   ├── commands
│   │   │   │   ├── import_admin1.py  # Import admin boundaries
│   │   │   │   ├── __init__.py
│   │   │   │   ├
│   │   │   │   ├── merge_jsonFiles.py
│   │   │   │   ├── syncD_shapefiles.py
│   │   │   │   ├── sync_raster.py
│   │   │   │   ├── syncS_shapefiles.py
│   │   │   │   ├── sync_tiffs.py
│   │   │   │   └── sync_timeseries.py
│   │   │   └── __init__.py
│   │   ├── migrations              # Database migrations
│   │   │   ├── 0001_initial.py
│   │   │   └── __init__.py
│   │   ├── models.py               # Database models
│   │   ├── serializers.py          # API serializers
│   │   ├── tests.py                # Unit tests
│   │   ├── urls.py                 # App-specific URL routing
│   │   └── views.py                # API views and business logic
│   ├── manage.py                   # Django management script
│   ├── requirements.txt            # Python dependencies
│   └── temp_data                   # Temporary data storage
│       └── merged_data.geojson
├── docker-compose.yml              # Docker services configuration
└── frontend
    ├── Dockerfile
    ├── eslint.config.js            # Code style configuration
    ├── index.html                  # Main HTML template
    ├── nginx.conf                  # Nginx server configuration
    ├── package.json                # Node.js dependencies
    ├── package-lock.json
    ├── public                      # Static public assets
    │   ├── GeoSFM_Forecast_Locations.geojson
    │   ├── icon.jpg
    │   └── merged_data.geojson
    ├── README.md
    ├── src                         # React source code
```


App.css
App.jsx # Main React component
assets # Images and static assets
banner-logo-undrr.png
cima_research_foundation_logo.jpg
flood-watch-banner.png
flood-watch-banner-transparent.png
icpac.jpg
ICPAC_Website_Header_Logo.svg
Logo_MAECI_colori_EN.png
react.svg
components # React components
About.jsx # About page
Analysis.jsx # Data analysis tools
Contact.jsx # Contact information
Home.jsx # Homepage
Indicators.jsx # Flood indicators display
MapViewer.jsx # Interactive map component
Partners.jsx # Partner organizations
index.css # Global styles
main.jsx # React entry point
vite.config.js # Vite build configuration
LICENSE
README.md

4. Backend System

4.1 Environment Configuration

The system uses a .env file for environment-specific configuration. Here's a sample of the environment variables used:

```
# Django Settings
SECRET_KEY=
DEBUG=
DJANGO_ALLOWED_HOSTS=
DJANGO_SETTINGS_MODULE=flood_watch_system.settings

# Database Configuration
DB_NAME=your_db_name
DB_USER=your_db_user_name
DB_PASSWORD='your_db_password'
DB_HOST=localhost
DB_PORT=5432

# CORS Configuration
CORS_ALLOWED_ORIGINS=''

# SFTP Configuration
SFTP_HOST=IP
SFTP_PORT=REMOTE_PORT
SFTP_USERNAME=remote_user_name
SFTP_PASSWORD='remotepass'
REMOTE_FOLDER_BASE='base folder'

# Redis Settings
REDIS_HOST=redis
REDIS_PORT=6379

# API Settings
API_VERSION=v1
CORS_ORIGIN_WHITELIST=''

# Docker Settings
TAG=latest
```

These environment variables control various aspects of the system including:

- Django security settings
- Database connection parameters
- Remote file transfer configuration
- Redis connection for Celery
- API versioning
- Docker image tagging

4.2 Core Django Project

The backend is built using Django and handles all data processing, including loading datasets, syncing files, and managing user access. It follows the MVC architecture pattern.

Key components include:

- **settings.py**: Contains database configuration, installed apps, middleware, and security settings
- **urls.py**: Defines API endpoints and routing
- **models.py**: Defines database schema for flood data, dams, and time series information
- **views.py**: Implements the business logic and API endpoints
- **serializers.py**: Handles data serialization/deserialization for API responses

4.2 Management Commands

The system includes several custom Django management commands for data operations:

Command	Description
import_admin1.py	Imports administrative boundary data
load_timeseries.py	Loads time series data for analysis
merge_jsonFiles.py	Merges multiple GeoJSON files
syncD_shapefiles.py	Synchronizes dam-related shapefiles
sync_raster.py	Synchronizes raster data files
syncS_shapefiles.py	Synchronizes supplementary shapefiles
sync_tiffs.py	Synchronizes TIFF image files
sync_timeseries.py	Synchronizes time series data

5. Frontend System

5.1 React Application

The frontend is built with React and Vite, providing an interactive user interface for visualizing and analyzing flood data.

Key components include:

- **MapViewer.jsx**: Interactive map visualization using web mapping libraries
- **Analysis.jsx**: Tools for analyzing flood data
- **Indicators.jsx**: Display for flood indicators and metrics
- **About.jsx**, **Home.jsx**, **Contact.jsx**, **Partners.jsx**: Informational pages

5.2 Assets and Data

The frontend includes various assets:

- Organization logos (ICPAC, UNDRR, CIMA Research Foundation)
- GeoJSON files for mapping (GeoSFM_Forecast_Locations.geojson, merged_data.geojson)
- UI elements and branding assets

6. Additional Components

6.1 GeoServer

GeoServer is used for serving geospatial data as OGC-compliant web services (WMS, WFS). It connects to the PostGIS database and handles:

- Rendering and styling of geospatial layers
- Processing geospatial queries
- Serving vector and raster data to the frontend

6.2 PostgreSQL/PostGIS

PostgreSQL with PostGIS extension serves as the primary database, handling:

- Storage of vector data (boundaries, flood zones, dam locations)
- Spatial queries and analysis
- Time series data relationships

6.3 Celery and Redis

Celery with Redis as the message broker handles asynchronous processing:

- Scheduled data synchronization from external sources
- Processing of large geospatial datasets
- Background tasks for data analysis and reporting

7. Data Flows

7.1 Data Ingestion

1. Administrative boundaries and base datasets are imported using management commands
2. Shapefiles, raster data, and time series are synchronized from external sources
3. GeoJSON files are merged to create unified datasets for visualization

7.2 Visualization Pipeline

1. Django backend processes data and exposes it through API endpoints
2. GeoServer serves spatial data as web services
3. React frontend consumes these services to render interactive maps and charts

8. Deployment

The entire system is containerized using Docker, making deployment consistent across environments.

8.1 Running the System

```
# Build and start all services
docker-compose up --build

# Run in detached mode
docker-compose up -d

# View logs
docker-compose logs -f

# Stop all services
docker-compose down
```

8.2 Development Environment

For development purposes:

1. Backend:

   ```
   cd backend
   pip install -r requirements.txt
   python manage.py migrate
   python manage.py runserver
   ```

2. Frontend:

   ```
   cd frontend
   npm install
   npm run dev
   ```

9. Maintenance

9.1 Backup Procedures

- Regular PostgreSQL database backups
- GeoServer configuration backups
- Source code version control through Git

9.2 Troubleshooting

Issue	Solution
Database connection problems	Check PostgreSQL container logs and credentials
GeoServer layer issues	Verify layer configuration in GeoServer admin
Data synchronization failures	Review management command logs for errors
Frontend display issues	Check browser console and network requests

10. Conclusion

The Flood Watch System provides a comprehensive platform for monitoring and analyzing flood risks using modern web technologies and geospatial components. Its containerized architecture ensures deployment consistency and scalability.