Flood Monitoring System: Key Components, APIs, and Structure

# 1. AWS Package Module

To handle the system on AWS, here are the services you’ll need:

* **Amazon EC2 (Elastic Compute Cloud)**: Host your Django application and APIs.
* **Amazon RDS (Relational Database Service)**: You can use PostgreSQL or MySQL to manage structured data. AWS RDS handles backups, scaling, and replication.
* **Amazon S3 (Simple Storage Service**): For storing large datasets (water level logs, reports, GIS data, etc.) that need to be easily accessible and scalable.
* **Amazon SNS (Simple Notification Service)**: For SMS and email notifications to authorities when water levels exceed thresholds.
* **Amazon Lambda**: For running serverless functions triggered by certain events, such as analyzing the water level and sending notifications.
* **AWS IoT Core**: For handling real-time data from rigs (IoT devices) measuring the river’s water levels.
* **AWS CloudWatch**: For monitoring the performance of your application and services, and setting up alarms if anomalies in system performance occur.
* **Amazon Polly**: To generate spoken alerts for authorities (optional).
* **AWS API Gateway**: To manage and scale the API connections between rigs, the server, and the Django backend.

Estimated pricing will depend on the usage:  
- EC2: Costs vary based on instance type (starting from $0.01/hour).  
- RDS: Starts from $0.02/hour for basic usage.  
- S3: Storage pricing based on amount of data.  
- SNS: $0.00645 per SMS.  
- **IoT Core**: Pricing based on messages exchanged. A rough monthly estimate could be $100–300, but you can optimize depending on exact resource usage.

# 2. Modules and API Structure (Django)

Here’s how the system’s architecture can be broken down:

## Rigs API

The rigs send water level data every few minutes via an API endpoint. A Django REST API handles this, authenticating the rig, accepting the data, and storing it.

Fields: timestamp, water\_level, location\_id, rig\_id, rainfall\_data.

## Prediction Module

Use machine learning libraries in Django (such as scikit-learn for linear regression) to model future water levels based on past data. The model runs on incoming data and forecasts trends.

API: GET /api/predictions/

Fields: location\_id, predicted\_level, time.

## Alert Module

An alerting system that checks if water levels exceed thresholds and triggers AWS SNS for SMS alerts. This can be implemented in Django as a background task using Celery.

API: GET /api/alerts/

Fields: alert\_id, location\_id, status, sms\_sent.

## GIS Module

Integration with a GIS mapping system to visualize water levels and predicted floods. You can use Django to query GIS data from Redis and render the results via a mapping library like Leaflet.

API: GET /api/map/

Fields: location\_id, water\_level, depth, flood\_zone.

## Subscription Management

Users subscribe to flood alerts for specific rivers/regions. You can integrate a Django subscription app (e.g., django-payments or django-stripe for MPESA and banks).

API: POST /api/subscription/

Fields: user\_id, location\_id, plan, payment\_method.

## User Authentication

Django’s built-in django-allauth can handle user registration, login, and role-based access for different users (e.g., local authorities).

API: POST /api/login/

API: POST /api/register/

Fields: username, email, password.

# 3. System Structure (APIs and Functionalities)

You will need the following key functionalities, grouped by the system features:

## Database:

* MongoDB: For storing raw water level data and real-time logs.
* PostgreSQL or MySQL: To store user information, subscriptions, and processed data.
* Redis (GIS): To store geospatial data for rapid access and querying.

## Flood Monitoring:

API from the rig for data input.

Data processing using linear regression or other algorithms.

Predictions and visualizations tied into the GIS module.

## Payment:

Integration with MPESA and banks (Equity, Coop, KCB) for user subscriptions. Use APIs provided by each payment service.

## SMS:

* AfricasTalking for SMS alerts to authorities and subscribers.
* AWS SNS - Notification service

## GIS:

Mapping water levels using Redis for geospatial queries.

Display real-time updates on a web interface with Django’s template engine or a front end framework like React.

## Analysis:

Analyze trends in water levels and rainfall patterns to improve prediction accuracy.

## Prediction:

Predict future water levels based on real-time and historical data. This can involve machine learning algorithms using libraries like TensorFlow or scikit-learn.

## Invoicing:

Generate and send invoices for subscription payments using Django’s template rendering system and automated emails.

## Subscription Management:

A system where users subscribe to water level alerts and manage their preferences.

# Suggested Tools and Libraries:

* Django REST Framework: For building APIs.
* Celery: For background task management (e.g., sending alerts).
* Redis: For caching GIS queries and rapid data access.
* Leaflet or Mapbox: For displaying GIS maps.
* AfricasTalking API: For SMS notifications.
* MPESA/Banks API: For payment integration.