Technical Report: ReliefTrack Implementation

Domain: Cloud-Based Disaster Management & Resource Allocation

1. Technical Architecture & Design

The ReliefTrack system follows a **Three-Tier Architecture** deployed on the Amazon Web Services (AWS) cloud platform.

1.1 Hybrid Database Strategy (The "Smart Proxy")

Unlike standard applications that use one database, this project implements a specialized `db.js` proxy.

- **Relational Data (MySQL):** Used for "Families" and "Inventory" where data consistency and complex relationships are required.
- **NoSQL Data (Amazon DynamoDB):** Used for "Transactions." This ensures that even if thousands of people receive aid simultaneously, the logging system never slows down the main application.

1.2 Component Breakdown

- **Frontend:** Built with React.js and Tailwind CSS. It uses the `GeminiAssistant.tsx` component to interface with Google's Generative AI.
- **Backend:** A Node.js Express server (`server.js`) that acts as the API Gateway for the frontend.
- **Security:** Implemented via AWS Security Groups, allowing traffic only on specific ports (3000 for Web, 3306 for Database).

---

2. AWS Implementation Steps (Deployment Workflow)

Step 1: Compute Setup (EC2)

We initialized an **Amazon EC2 Instance** (Ubuntu/Linux). We configured the **Security Groups** to allow inbound traffic for the backend API.

- *Action:* SSH into the instance using the `.pem` key.
- *Action:* Install Node.js and Npm on the server.

Step 2: Database Provisioning

1. **MySQL:** Installed locally on the EC2 instance to handle core relational tables (`families`, `inventory`).
2. **DynamoDB:** Created a table named `transactions` in the `us-east-1` region. We defined a `Partition Key` as `id`.

Step 3: IAM Configuration

To allow the EC2 instance to "talk" to DynamoDB without hardcoding secret keys, an **IAM Role** was created with `AmazonDynamoDBFullAccess` and attached to the EC2 instance.

Step 4: Environment Configuration

The `.env` file was configured on the server to point to the local MySQL instance and the AWS SDK region.

Code snippet
```
DB_HOST=localhost
AWS_REGION=us-east-1
DYNAMODB_TABLE=transactions
```

---

3. Implementation Logic (Code Snippets)

3.1 The DB Proxy Logic

In `db.js`, we intercept SQL queries. If the query contains the word "transactions," the system redirects the data to AWS DynamoDB instead of MySQL:

```
if (lowerSQL.includes("transactions")) {
    // Redirect to DynamoDB PutCommand
    await ddbDoc.send(new PutCommand({ TableName: "transactions", Item: item
}));
}
```

3.2 AI Integration

The `GeminiAssistant.tsx` uses the `@google/genai` library to provide context-aware help. For "Seekers," the AI is instructed to be compassionate; for "Staff," it is instructed to be efficient and data-oriented.

---

4. Testing & Results

4.1 Unit Testing

We verified that:

1. Registering a family in the **Staff Portal** creates a row in **MySQL**.
2. Distributing a ration creates a record in **Amazon DynamoDB**.
3. The **Seeker Portal** correctly reflects the balance after a transaction.

4.2 Scalability Testing

By offloading transactions to DynamoDB, the MySQL database remains lightweight, preventing system crashes during high-traffic relief events.

---

5. Conclusion & Future Work

The ReliefTrack project successfully demonstrates how AWS Cloud services can be combined with AI to solve humanitarian logistics. **Future Scope:** * Integrating **AWS S3** for storing identity document photos of families.

- Using **AWS Lambda** to send automated SMS alerts to families when new stock arrives.