Explication du projet

Contexte du problème :

DJEZZY, un opérateur téléphonique en Algérie, rencontre des difficultés à **détecter et gérer les erreurs** dans ses systèmes. Ces erreurs se divisent en **deux catégories** principales :

1. **Erreurs utilisateurs** :
    - Cela concerne les erreurs causées par les utilisateurs eux-mêmes, souvent dues à une mauvaise saisie ou une utilisation incorrecte des codes raccourcis (short codes).
    - Par exemple : un utilisateur tape le mauvais code, comme *719# au lieu de *710#. Ces erreurs sont généralement humaines et sont courantes dans les systèmes mobiles.
2. **Erreurs internes à DJEZZY** :
    - Ce sont des problèmes qui proviennent du système interne de DJEZZY, comme un short code qui ne fonctionne pas correctement, qui met beaucoup de temps à répondre ou qui ne répond pas du tout.
    - Par exemple, un serveur peut être saturé ou un script peut être arrêté, ce qui empêche le bon fonctionnement du service. Ces erreurs sont souvent dues à des problèmes techniques au niveau du réseau ou des serveurs.

Problématique :

Actuellement, DJEZZY ne dispose **d'aucun système automatisé** permettant de différencier et de suivre précisément ces erreurs en temps réel. Voici les problèmes clés :

1. **Manque d'automatisation** :
    - Actuellement, les erreurs sont **vérifiées manuellement**. Cela prend beaucoup de temps et peut mener à des diagnostics lents et inefficaces.
    - Sans un système d'automatisation, il est difficile de détecter rapidement l'origine de l'erreur (utilisateur ou interne), ce qui ralentit la résolution.
2. **Absence de différenciation entre erreurs utilisateurs et erreurs internes** :
    - Les erreurs ne sont pas correctement catégorisées. Par exemple, un problème lié à un code utilisateur mal tapé (erreur d'utilisateur) peut être confondu avec une erreur interne, ce qui compliquerait le diagnostic.
    - Cela rend difficile la gestion des erreurs et la mise en place d'actions correctives adaptées.
3. **Manque de réactivité proactive** :
    - DJEZZY ne dispose pas de **système d'alerte en temps réel**. Sans alertes automatiques, les problèmes peuvent persister longtemps avant que des actions ne soient entreprises.
    - Par exemple, si un short code met trop de temps à répondre (problème interne), DJEZZY pourrait ne pas s'en rendre compte à temps avant qu'un grand nombre d'utilisateurs ne soient affectés.

4. **Dépendance aux outils existants** :
    - DJEZZY utilise actuellement **ChartJS** pour la visualisation, mais il n'y a pas de système intégré pour gérer les alertes ou effectuer des analyses en temps réel.
    - Il semble également que DJEZZY utilise **Bash scripts** pour certaines opérations, mais cela manque de cohérence et d'automatisation au niveau global.

Difficultés techniques à résoudre :

1. **Intégration avec les KPIs de DPSS** :
    - Le **DPSS KPI** mesure le temps de réponse des short codes. Cette donnée est essentielle pour détecter si un problème vient d'un délai trop long dans le système.
    - L'intégration avec ces KPIs doit être réalisée de manière fluide pour que les erreurs puissent être repérées immédiatement, et non après une analyse manuelle.
2. **Analyse des causes des erreurs** :
    - Les erreurs peuvent provenir de diverses sources : des **connecteurs DPSS** défectueux, des **scripts arrêtés**, des **logs mal générés**, ou des **serveurs saturés**.
    - Identifier la cause exacte de l'erreur en temps réel et la classifier correctement est un défi majeur. Par exemple, un problème de serveur peut se manifester sous la forme d'une lenteur, mais il faut être capable de déterminer si c'est un problème technique de DJEZZY ou une erreur de l'utilisateur.
3. **Traitement des erreurs en temps réel** :
    - Le but est d'avoir un système qui soit capable de détecter les erreurs **avant que DPSS** (ou tout autre système de monitoring) ne génère des alertes internes.
    - Cela implique d'analyser les données des logs, d'appliquer des seuils de performance, et de **gérer les alertes de manière proactive**.

Objectifs principaux à atteindre :

1. **Automatiser la détection et la classification des erreurs** :
    - Créer un système qui peut **distinguer automatiquement les erreurs utilisateurs des erreurs internes** à partir des logs et des KPIs.
2. **Mettre en place des alertes proactives** :
    - Définir des **seuils d'alerte** (par exemple, seuil 1 = email, seuil 2 = SMS, etc.) qui permettent à DJEZZY d'être informé avant que l'erreur n'ait des conséquences graves sur les utilisateurs.
3. **Gérer les erreurs en temps réel** :
    - Offrir une solution qui puisse détecter et **agir sur les erreurs dès qu'elles apparaissent**, sans attendre un traitement manuel.

Résumer le problème en quelques points :

- **Manque de différenciation** entre erreurs utilisateurs et erreurs internes.
- **Difficulté à traiter les erreurs de manière proactive** et en temps réel.
- **Dépendance à des processus manuels** lents pour diagnostiquer et résoudre les problèmes.
- **Absence de système d'alerte automatisé** permettant une prise en charge rapide des incidents.

Division du projet en grandes parties

Voici une proposition de **découpage en quatre grandes parties**, avec les objectifs et les livrables de chacune :

1. Phase 1 – Collecte et centralisation des données (Logs & KPIs)

🎯 **Objectif :**

Mettre en place la **base technique** du système : collecter les données d'erreurs depuis les différentes sources et les centraliser.

🔧 **Tâches principales :**

- Identifier toutes les **sources de données** (fichiers logs, DPSS KPI, bases internes, scripts Bash, etc.).
- Établir des **scripts d'extraction et de collecte** automatique (par exemple en Bash, Python ou via Filebeat).
- Centraliser ces données dans un **système unifié** :
    - soit dans une base de données (PostgreSQL, Elasticsearch, etc.),
    - soit dans un outil de monitoring (ELK, Prometheus, etc.).
- Normaliser les logs (un format commun : timestamp, code erreur, source, description, etc.).

💡 **Livrables :**

- Un pipeline de collecte automatisée.
- Une base de données centralisée contenant les logs et les KPIs DPSS.

2. Phase 2 – Analyse et classification des erreurs

🎯 **Objectif :**

Distinguer **les erreurs utilisateurs** des **erreurs internes** et comprendre leurs causes.

🔧 **Deux approches possibles :**

- **Approche algorithmique (règles fixes)** :
    - Exemple : si l'erreur est liée à un code erroné (*719# au lieu de *710#), on la classe comme "erreur utilisateur".
    - Si le temps de réponse du short code dépasse un seuil, on la classe comme "erreur interne".
- **Approche IA (machine learning)** :
    - Entraîner un modèle à partir d'un historique d'erreurs pour qu'il apprenne à reconnaître le type d'erreur.
    - Exemple : utilisation d'un modèle de classification supervisée (Random Forest, XGBoost, etc.).

💡 **Livrables :**

- Un module de **classification automatique** des erreurs.
- Une base de données enrichie avec les catégories d'erreurs ("Utilisateur" / "Interne").

3. Phase 3 – Détection et alertes en temps réel

🎯 **Objectif :**

Automatiser la **surveillance** et la **réaction** en cas de dépassement de seuils ou de détection d'anomalies.

🔧 **Tâches principales :**

- Définir des **seuils d'alerte** :
    - Seuil 1 : notification par **email**.
    - Seuil 2 : notification par **SMS**.
    - Seuil 3 : déclenchement d'une **alerte critique** à l'équipe technique.
- Développer un **système de monitoring** (Prometheus, Zabbix, Grafana…).
- Détecter les **anomalies** (pannes de connecteurs, scripts arrêtés, saturation serveur…).
- Créer un **mécanisme proactif** : anticiper les erreurs avant que DPSS ne signale un incident.

💡 **Livrables :**

- Système d'alertes automatisé en temps réel.
- Notifications par email/SMS configurées.
- Journal des alertes envoyées.

4. Phase 4 – Visualisation et reporting

🎯 **Objectif :**

Présenter les résultats de façon claire pour le suivi et la prise de décision.

🔧 **Tâches principales :**

- Développer un **dashboard dynamique** (ChartJS, Grafana, Kibana…).
- Visualiser :
    - le nombre d'erreurs par type,
    - les performances des short codes,
    - les temps de réponse,
    - les alertes actives.
- Générer des **rapports automatiques** (journaliers, hebdomadaires, mensuels).

💡 **Livrables :**

- Dashboard opérationnel intégré dans l'environnement de DJEZZY.
- Rapports PDF/Excel automatisés (suivi des erreurs, tendances, etc.).

⚙️ **Optionnel : Division en sous-projets techniques**

Selon la taille de l'équipe et les ressources disponibles, tu peux encore **diviser chaque phase** en sous-projets techniques, par exemple :

Domaine	Sous-projet	Objectif
Data Engineering	Collecte & pipeline des logs	Construire les scripts et le stockage des données
Monitoring	Mise en place de Prometheus / Grafana	Détection en temps réel des anomalies
IA / Data Science	Modèle de classification des erreurs	Distinguer erreurs utilisateurs / internes
DevOps	Automatisation des déploiements	Déployer les scripts et alertes en PROD
Frontend	Tableau de bord (ChartJS, Grafana)	Visualiser et filtrer les erreurs

Vue d'ensemble de la progression

Phase	Objectif principal	Livrable clé
1	Collecte et centralisation	Données unifiées et prêtes à analyser
2	Classification	Module qui distingue les types d'erreurs
3	Alertes en temps réel	Système d'alerte automatisé

Phase	Objectif principal	Livrable clé
4	Visualisation	Dashboard complet et rapports

Division du projet entre deux binomes

🎯 Objectif global (rappel)

Automatiser la **détection et le suivi des erreurs** dans les systèmes de DJEZZY, en distinguant :

- les **erreurs utilisateurs**,
- les **erreurs internes (administratives)**,
  et en générant des **alertes en temps réel** selon des seuils définis.

👥 Répartition du travail pour un binôme

Voici une **proposition équilibrée** et réaliste pour deux étudiants de licence :

🔷 Binôme 1 — Partie 1 : Collecte & Classification des erreurs

🎯 Objectif :

Mettre en place le **pipeline d'analyse des erreurs** — c'est-à-dire la récupération des données, leur traitement et leur classification (utilisateur / interne).

🔧 Responsabilités principales :

1. **Collecte des logs et des données KPIs**
   - Écrire des **scripts Bash** pour extraire les logs (ex : depuis /var/log/, DPSS KPI, etc.).
   - Nettoyer et uniformiser les logs dans un format commun (CSV, JSON, etc.).
   - Automatiser la collecte via cron jobs ou scripts programmés.
2. **Analyse et traitement des données**
   - Identifier les **différents types d'erreurs** et leurs signatures (mot-clés, codes d'erreur, messages spécifiques).
   - Mettre en place un **programme Python ou Bash** pour filtrer les erreurs.
3. **Classification des erreurs**
   - Définir des **règles logiques** (ou un petit modèle IA si possible) pour classer :
     - erreurs utilisateur (codes mal saisis),
     - erreurs internes (retards, échecs serveurs, scripts stoppés, etc.).

- Sauvegarder les résultats dans une base (ou fichier structuré).

💡 **Livrables :**

- Script de collecte automatique des logs.
- Module de classification automatique (règles ou IA simple).
- Documentation technique expliquant les étapes et les critères utilisés.

🧠 **Compétences mobilisées :**

- Linux / Bash scripting
- Analyse de logs
- Python (pour traitement de texte et classification)
- Notions d'IA (facultatif mais valorisant)

🔷 **Binôme 2 — Partie 2 : Alertes & Visualisation en temps réel**

🎯 **Objectif :**

Créer le **système d'alerte automatique** et un **dashboard interactif** pour visualiser les erreurs et leur évolution.

🔧 **Responsabilités principales :**

1. **Définition des seuils d'alerte**
   - Travailler avec le binôme 1 pour identifier les seuils critiques :
     - Exemple : si le taux d'erreurs internes > 10% sur 5 min → envoi d'un email.
     - Si le temps de réponse DPSS > X secondes → envoi d'un SMS.
2. **Système d'alertes automatisées**
   - Mettre en place des scripts qui **surveillent les logs traités** et déclenchent les alertes.
   - Utiliser :
     - **mailx** ou **sendmail** pour l'envoi d'emails,
     - **API SMS** (ex. Twilio, Kannel ou gateway interne Djezzy) pour les alertes SMS.
   - Enregistrer toutes les alertes dans un fichier log ou une base.
3. **Visualisation et dashboard**
   - Utiliser **ChartJS** (outil déjà utilisé par Djezzy) pour afficher :
     - le nombre d'erreurs par type,
     - les temps de réponse des short codes,

        - l'état des alertes actives.
    - Créer une **page web** simple (HTML/JS/PHP) connectée à vos données.
    - Bonus : ajouter des **filtres** (par jour, type d'erreur, code USSD…).

💡 **Livrables :**

- Système d'alerte automatique (email + SMS).
- Tableau de bord ChartJS dynamique et lisible.
- Documentation utilisateur pour le monitoring.

🧠 **Compétences mobilisées :**

- JavaScript / ChartJS / HTML / CSS
- Shell / Cron / Automatisation
- Concepts de monitoring et seuils
- Communication entre scripts et interface

🔄 **Collaboration entre vous deux**

Les deux parties sont **étroitement liées**, donc vous devrez collaborer régulièrement :

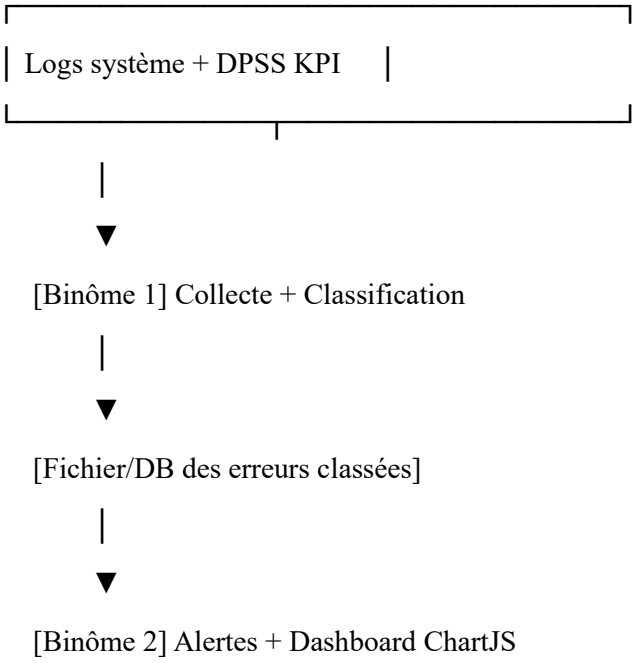
Étape	Binôme 1	Binôme 2
Collecte des logs	✅ (mise à disposition des données)	🔶 (utilise les données traitées)
Classification	✅ (production du fichier "erreurs classées")	🔶 (affichage des résultats)
Alertes	🔶 (donne les seuils d'erreurs)	✅ (implémente l'envoi et les notifications)
Visualisation	🔶 (fournit les données brutes)	✅ (développe le dashboard ChartJS)

Vous travaillez donc sur deux modules **complémentaires** :

- le premier crée et prépare les données,
- le second les exploite pour surveiller et alerter.

📦 **Architecture globale du projet**

Voici une vision simple du flux :



📅 Proposition de planning (8–10 semaines)

Semaine	Binôme 1	Binôme 2
1–2	Étude du système, collecte manuelle des logs	Étude de ChartJS et plan du dashboard
3–4	Scripts de collecte automatique	Maquette du dashboard (HTML/JS)
5–6	Module de classification automatique	Implémentation du système d'alerte
7–8	Tests et validation des données	Intégration des alertes + graphiques
9–10	Documentation & rapport	Documentation & rapport final
