Examen National de Fin de Formation
Session de Report 2025
Examen de Fin de Formation (Epreuve de Synthèse)

Secteur :	**Digital et Intelligence Artificielle**	**Niveau :**	**Technicien Spécialisé**
Filière :	**Développement Digital option Web Full Stack**		

Variante	**V1**	**Durée :**	**4H00**	**Barème**	**100**

Consignes et Conseils aux candidats :
- Toutes les réponses devront être justifiées avec le détail des calculs qui doit être indiqué sur la copie ;
- Apporter un soin particulier à la présentation de votre copie

Document(s) et Matériel(s) autorisés :
- Les documents ne sont pas autorisés ;
- Calculatrice simple (non programmable) autorisée.

Détail du Barème :

N° Des Dossiers	Travaux à réaliser	Barème
Partie 1 : Théorie		
Dossier 1	Création d'une application Cloud native	8
Dossier 2	Préparation d'un projet web	8
Dossier 3	Approche agile	14
Dossier 4	Gestion de données NOSQL	10
	Total Partie 1 : Théorie	**/40points**
Partie 2 : Pratique		
Dossier 1	Gestion de données (MYSQL)	10
Dossier 2	Développement front-end	24
Dossier 3	Développement back-end	26
	Total Partie 2 : Pratique	**/60points**
	Total Général	**/100points**

Préliminaire :

GeoRelief Solutions, société spécialisée dans le développement des systèmes informatiques propose une solution innovante pour assurer le suivi des événements catastrophiques, des zones touchées, des infrastructures endommagées, des actions mises en place, ainsi que des donations reçues. Ce système permet une gestion structurée et cohérente de toutes ces informations grâce à un schéma relationnel soigneusement conçu.

Les principales tables et leurs colonnes sont les suivantes :

- **Catastrophes** (id, nom, type, date, description, statut)
- **Zone_Affectées** (id, nom_zone, nom_region, superficie_touchee, coordonnees_gps)
- **Catastrophe_Zone_Affectee** (#catastrophe_id, #zone_affectee_id, gravité)
- **Infrastructures** (id, type, niveau_degat, cout_estime_reconstruction, statut_reconstruction,#zone_affectee_id)
- **Actions** (id, date_debut, date_fin, montant, type, description, #infrastructure_id)
- **Donations** (id, date, montant, #action_id, #donateur_id)
- **Donateurs** (id, nom, prenom, date_naissance, tel, email)

Les valeurs possibles pour certains champs sont les suivantes :

Champ	Valeurs possibles
Type (Catastrophes)	*tremblement_de_terre, inondation, vague_de_chaleur*
Gravité (Catastrophe_Zone_Affectee)	*faible, moderee, elevee*
Statut (Catastrophes)	*ouverte, en_cours, terminee*
Status Reconstruction (Infrastructures)	*non_reparee, en_cours, terminee*
Type (Actions)	*evacuation, reconstruction, aide_humanitaire*

Partie Théorique : (40pts)

Dossier 1 : Création d'une application Cloud native (8pts)

1. Quels sont les principaux avantages du cloud native ? **(3 pts)**
2. Quels sont les systèmes d'exploitation compatibles avec Docker ? **(3 pts)**
3. Pourquoi on utilise RabbitMq ? **(2 pts)**

Dossier 2 : Préparation d'un projet web (8pts)

1. A partir du schéma relationnel précédent, réaliser le MCD (modèle conceptuel de données). **(4 pts)**
2. Réaliser le diagramme de cas d'utilisation concernant les acteurs suivants :
   - Donateur : qui peut renseigner ses informations personnelles, saisir et suivre ses donations.
   - Responsable : qui valide les donations, et gère les différentes actions.

   ***Remarque** : Toutes les fonctionnalités précédentes nécessitent obligatoirement une authentification.*

   (4 pts)

Dossier 3 : Approche agile (14pts)

1. Quelle est la différence entre une **user story** et une **tâche** ? **(2 pts)**
2. Selon le préliminaire, donner deux **users stories** à ajouter dans le **Backlog produit. (2 pts)**
3. Soit le diagramme de Gantt suivant (Figure 1):

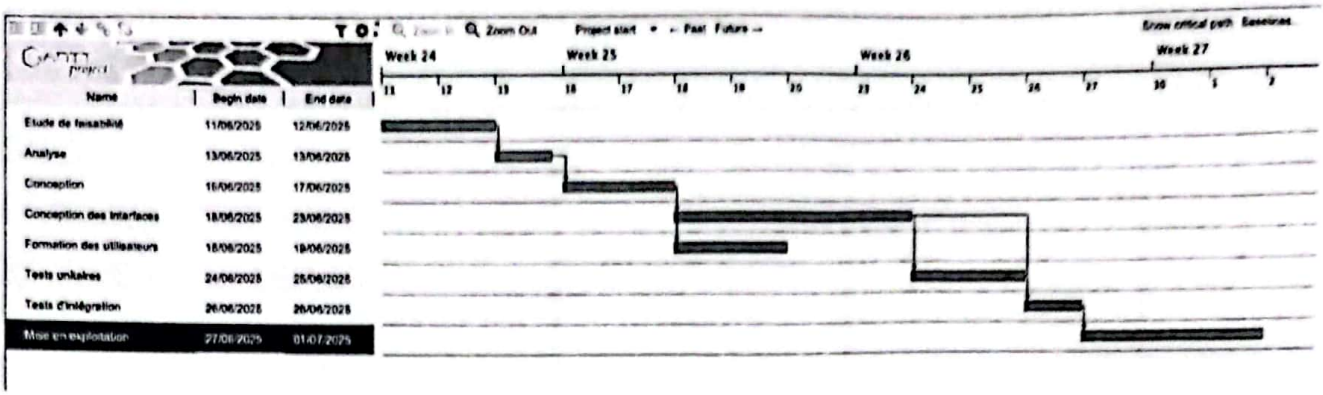

Figure 1

En supposant que l'étude de faisabilité démarre le Mercredi **11 juin 2025,** et que la personne responsable du projet ne travaille pas les weekends :

a- Calculer la durée ouvrable du projet. **(2 pts)**

b- Indiquer le chemin critique. **(2 pts)**

c- Le responsable du projet sera en formation du **26/06/2025 au 01/07/2025,** indiquer la date de fin de projet dans ce cas . **(2 pts)**

4. Que font les commandes git suivantes : **(2 pts)**
   a. `git checkout -b branche1`
   b. `git push`
5. Que signifient les résultats suivants :
   - Reliability : A
   - Security : B
   - Maintainabilty : B

Obtenus lors de l'analyse de qualité du code source de notre projet sur SonarQube **(2 pts)**

Dossier 4 : Gestion de données NOSQL (10pts)

1. Créez une base de données nommée **GestionCatastrophes** et une collection appelée **zones_affectees.** Insérez-y le document suivant : **(2pts)**

```
{
"_id": "alhaouz1",
"nom_zone": "La province d'Al Haouz",
"nom_region": "Marrakech-Safi",
"superficie_touchee": 35.2,
"coordonnees_gps": { "latitude": 31.5107, "longitude": -8.5124 },
"infrastructures": [
  {
   "type": "Pont",
   "niveau_degat": "élevé",
   "cout_estime_reconstruction": 1500000,
   "statut_reconstruction": "non_reparee"
  },
  {
   "type": "Route principale",
   "niveau_degat": "modéré",
   "cout_estime_reconstruction": 800000,
   "statut_reconstruction": "en_cours"
  }
 ]
}
```

On suppose que la collection **zones_affectees** contient plusieurs documents similaires.

Écrivez les requêtes MongoDB nécessaires pour :

2. Afficher les zones de la région **Marrakech-Safi** ayant au moins une infrastructure en cours de reconstruction, en ne montrant que **nom_zone, superficie_touchee,** et les **infrastructures** correspondantes. **(2pts)**
3. Réaliser une requête qui permet d'afficher :
   - Le nom de la région (nom_region)
   - Le nombre total d'infrastructures en cours de reconstruction de cette région

   pour chaque **région**, ayant le statut_reconstruction égal à **"en_cours"**. **(2pts)**
4. Pour toutes les zones où au moins une infrastructure a "statut_reconstruction" égal à **"non_reparee"**, ajouter le champ **statut_global** avec la valeur **"en attente"**. **(2pts)**
5. Exporter tous les documents de la collection **zones_affectees** dans un fichier JSON nommé **zones_affectees.json. (2pts)**

Partie Pratique : (60pts)

Dossier 1 : Gestion de données (MYSQL) (10pts)

En utilisant le schéma relationnel de la partie préliminaire :

1. Créer une fonction nommée **fn_get_montant_total_reconstruction(zone_id INT)** qui retourne le coût total estimé pour la reconstruction des infrastructures dans une zone donnée. **(2pts)**
2. Créer une fonction nommée **fn_get_ecart_financement(zone_id INT)** qui utilise la fonction **fn_montant_total_reconstruction (de la question 1)** pour calculer et retourner la différence entre le coût total estimé pour la reconstruction d'une zone et le total des donations reçues pour cette zone. **(3pts)**
3. Créer une procédure nommée **ps_update_statut_catastrophe(catastrophe_id INT)** qui utilise la fonction fn_get_ecart_financement (**de la question 2**). Cette procédure met à jour le ***statut*** de la catastrophe à "terminee" si l'écart de financement est inférieur ou égal à 0. **(3pts)**
4. Créer un trigger nommé **tr_after_donation_insert** qui, après l'insertion d'une donation dans la table Donations, vérifie si le montant est supérieur à 100 000. Si c'est le cas, insérer un message dans la table ***Logs (champs : id, action_id, donateur_id, montant, message)***
   sous la forme : **"Donation importante : montant X pour l'action Y"**, où X est le montant et Y l'identifiant de l'action. **(2pts)**

Dossier 2 : Développement front-end (24pts)

On suppose qu'on dispose de l'API suivante :

Méthode HTTP	Get
URL de l'api	http://localhost:8000/donateursdata
Résultat	[{ "id": 1, "nom": "MAHDI", "prenom" : "Amine", "image" : "https:/url/image1.jpg", "email" :"mahdiamine@mail.com", "donations": [{ "date": "2025-05-12", "montant": 5000, "action" : "Reconstruction des habitations" }, { "date": "2025-06-11", "montant": 6000, "action" : "Réhabilitation des écoles" }] }, ...]

1. Créer le composant **App.jsx** qui appelle l'API(*http://localhost:8000/donateursdata*) et stocke les données dans le tableau **DonateursData (3 pts)**

2. Ajouter dans le composant **App.jsx** la route vers le composant **listeDonateurs** qui reçoit le tableau DonateursData comme props. **(2 pts)**

3. Créer le composant **listeDonateurs** comportant les fonctionnalités suivantes (voir figure 2):

   a- Options de recherche des donateurs par nom, action. **(2 pts)**

   b- Option d'affichage de tous les donateurs. **(2 pts)**

   c- Affichage des résultats de recherches comme représenté dans la figure 2. **(3 pts)**

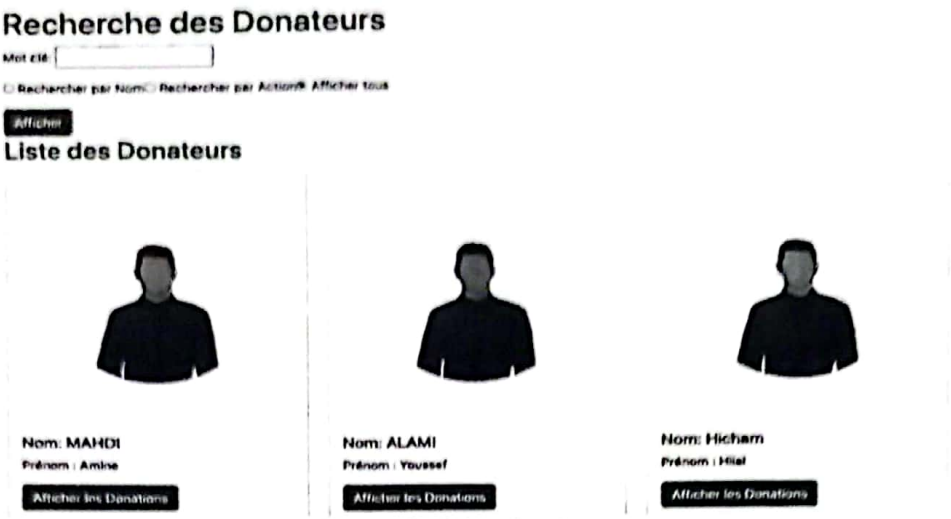

Figure 2 : Liste des donateurs

4. Créer le composant **Details** qui affiche les donations du donateur cliqué (Afficher les donations) dans le composant précédent , on affiche également le montant total de ses donations (voir figure 3) **(3 pts)**

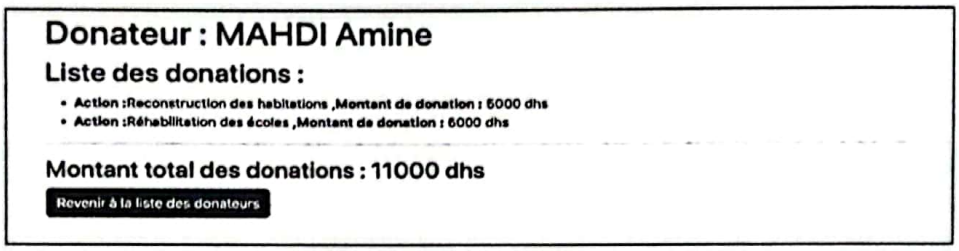

Figure 3 : Détails du Donateur

5. En utilisant Redux ou Redux Toolkit , soit l'état initiale suivant :

const **initialState**={

listeDonateurs :[ ]

}

Créer le Reducer **DonateurReducer** contenant les actions suivantes **(3 pts)**:

- **AddDonateur** : Permettant d'ajouter un nouveau donateur dans l'état initial
- **DeleteDonateur** : Permettant de supprimer un donateur de l'état initial

6. Créer le **store (2 pts)**

7. Créer le composant **Donateur** qui permet de lister les donateurs du store en utilisant les actions **AddDonateur** et **DeleteDonateur** voir figure 3 **(4 pts)**

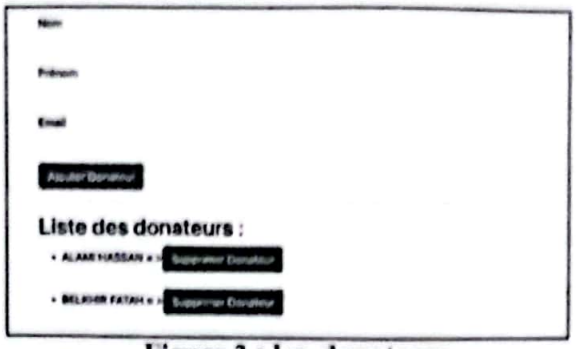

Figure 3 : les donateurs

Dossier 3 : Développement back-end (26pts)

En utilisant le schéma relationnel de la partie préliminaire réaliser ce qui suit :

1.1. Complétez le tableau suivant en fournissant la description et rôle pour chaque commande :**(1pt)**

Questions	Commande Artisan	Description / Rôle
a.	php artisan serve	.....
b.	php artisan make:seeder CatastropheSeeder	.....
c.	php artisan route:list	.....
d.	php artisan db:seed --class=CatastropheSeeder	.....

1.2. Complétez le tableau suivant en fournissant la commande correspondant à chaque description : **(1pt)**

Questions	Description / Rôle	Commande Artisan
a.	Créer une migration pour la table `actions`.	.....
b.	Appliquer toutes les migrations pour mettre à jour la base de données.	.....
c.	Supprimer le cache des routes enregistrées dans votre application.	.....
d.	Créer une classe de validation pour gérer les règles de la requête `ZoneAffecteeRequest`.	.....

2.1. Donner la commande de création du contrôleur de ressource nommée CatastropheController. **(1 pt)**

2.2. Remplissez le tableau des routes **générées automatiquement** par Route::resource pour le contrôleur CatastropheController. **(2pts)**

Méthode HTTP	URI	Action	Nom de Route

3. Créer le modèle **Catastrophe** en y ajoutant les éléments suivants: **(2pts)**

- La propriété **$table** pour définir explicitement la table associée.
- La propriété **$fillable** pour rendre modifiables les colonnes : nom, type, date, description, statut.
- Une méthode **zones()** pour établir la relation avec **ZoneAffectee** via la table pivot **catastrophe_zone_affectee**, en incluant le champ gravité.

4. Créer le modèle **ZoneAffectee** en y ajoutant les éléments suivants :**(2pts)**
   - Une méthode **catastrophes()** définissant la relation avec Catastrophe via la table pivot **catastrophe_zone_affectee.**
   - Une méthode **infrastructures()** définissant une relation avec le modèle **Infrastructure**

5.1. Dans le contrôleur **CatastropheController** : **(2pts)**
Écrire une méthode **getCatastropheWithZones ($nomCatastrophe)** qui prend comme paramètre ($nomCatastrophe) et qui retourne la catastrophe et ses zones associées, La redirection doit se faire vers la vue **catastrophes.zones**

5.2. Dans le contrôleur **ZoneAffecteeController** : **(2pts)**
Écrire une méthode **getZoneWithInfrastructures ($nom_zone)** qui prend en paramètre **($nom_zone)**et retourne la zone et ses infrastructures . La redirection doit se faire vers la vue **zones.infrastructures**

5.3. Créer la vue **catastrophes.zones** pour afficher les zones retournées par la méthode **getCatastropheWithZones,** pour chaque zone, afficher : **nom_zone, nom_region et la gravité. (1pt)**

5.4. Créer la vue **zones.infrastructures** pour afficher les infrastructures retournées par la méthode **getZoneWithInfrastructures,** pour chaque infrastructure, afficher : **type, cout_estime_reconstruction, et statut_reconstruction. (1pt)**

6.1. Créer un contrôleur nommé **CatastropheApiController** dédié à la gestion de la table Catastrophes dans la base de données. Ce contrôleur contiendra les méthodes nécessaires pour gérer les routes décrites ci-dessous. **(1pt)**

6.2. Définissez les routes API dans **`routes/api.php` : (2pts)**

Méthode HTTP	PATH	Méthode du contrôleur	Description
GET	/catastrophes	Index()	Retourne une liste de toutes les catastrophes en format JSON (Code HTTP : 200)
GET	`/catastrophes/searchByNom/{nom}`	SearchByNom($nom)	Recherche une seule catastrophe par son nom et retourne le résultat en JSON (Code HTTP : 200 ou 404 si non trouvée)
POST	/catastrophes	Store(request $request)	Ajoute une nouvelle catastrophe dans la base de données et retourne la catastrophe ajoutée en JSON (Code HTTP : 201)
DELETE	/catastrophes/{id}	Destroy($id)	Supprime une catastrophe en fonction de son ID et retourne un message de succès ou d'erreur en JSON (Code HTTP : 200 ou 404 si non trouvée)

6.3. **Pour chaque route** définie dans le fichier **routes/api.php**, donner l'implémentation de la méthode correspondante dans le contrôleur **CatastropheApiController**. Les réponses doivent être renvoyées en format JSON comme décrit dans **la question 6.2 (8pts)**