Examen National de Fin De Formation
Session de Juin 2025
Examen de Fin de Formation (Epreuve de Synthèse)

Eléments de correction

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

Variante	**1**	**Durée :**	**4h00**	**Barème**	**/100**

Consignes et Précisions aux correcteurs :

Veuillez respecter impérativement les consignes suivantes :

- **Les éléments de correction sont élaborés à titre indicatif,**
- **Eviter de sanctionner doublement le stagiaire sur les questions liées,**
- **Pour toutes les questions de synthèse et de compréhension le correcteur s'attachera à évaluer la crédibilité et la pertinence de la réponse du stagiaire. Et à apprécier toute réponse cohérente du stagiaire,**
- **Le stagiaire n'est pas tenu de fournir des réponses aussi détaillées que celles mentionnées dans le corrigé,**
- **En cas de suspicion d'erreur au niveau du corrigé, prière de contacter la Division de Conception des Examens.**

Détail du Barème :

N° Des Dossiers	Travaux à réaliser	Barème
Partie Théorique		
Dossier 1	Création d'une Application Cloud native	**8 pts**
Dossier 2	Préparation d'un projet web	**6 pts**
Dossier 3	Approche Agile	**07 pts**
Dossier 4	Gestion de données NOSQL	**11 pts**
Dossier 5	Web Dynamique PHP	**08 pts**
	Total Partie Théorique	**/40 points**
Partie Pratique		
Dossier 1	Gestion des données MySQL	**12 pts**
Dossier 2	Développement Front End	**24 pts**
Dossier 3	Développement Back End	**24 pts**
	Total Partie Pratique	**/60 points**
	Total Général	**/100 points**

Partie Théorique (40 pts)

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

1- Quels sont les composants principaux de Docker?

Les composants incluent Docker Engine, Docker Hub (registre pour les images), et Docker Compose (orchestration multi-conteneurs).

2- Comment installer Express.js ?

```
npm install express
```

3- Comment afficher les images Docker ?

```
docker images
```

4- Comment fonctionne RabbitMQ ?

RabbitMQ est un broker de messages mentionné dans le contexte des microservices. Il facilite la communication en mettant en file d'attente les messages et en les livrant aux services appropriés.

Préliminaire :

La société « **FastFoodDelivery**» fournit à ses employés des vélos électriques pour assurer la livraison des colis auprès de ses clients. Pour garantir la continuité et la qualité de ses services, elle a décidé de mettre en place un système de suivi de l'état des batteries de ces vélos. Ce système permettra d'anticiper le remplacement des batteries avant qu'elles ne se détériorent ou tombent en panne.

Velo(id, matricule, #id_batterie, date_derniere_maintenance, date_prochaine_maintenance)
Batterie(id,numero_serie, capacite, sante_batterie, nombre_cycles, statut)
Changement(id, #id_velo, #id_ancienne_batterie, #id_nouvelle_batterie, date_changement, #id_technicien,raison)
Technicien(id, nom, telephone, email, specialite)

NB :*La clé primaire est écrite en souligné, la clé étrangère est écrite en #*

La clé primaire de la table 'Batterie' est présente deux fois comme clé étrangère dans la table 'Changement' : id_ancienne_batterie, id_nouvelle_batterie

Dossier 2 : Préparation d'un projet web (6 pts) :

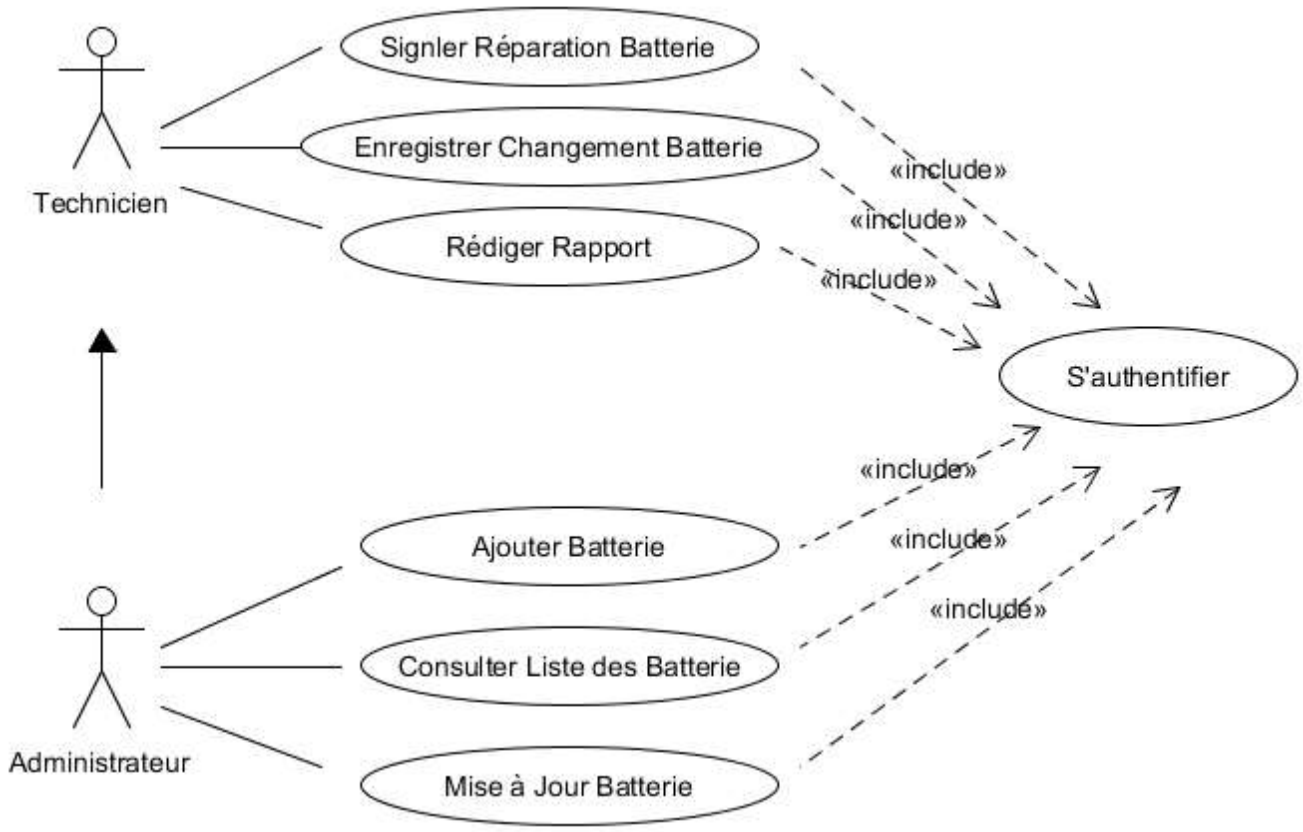
Donner le diagramme de cas d'utilisation correspondant au système suivant :(6 pts)

Le système sera utilisé par deux types d'utilisateurs : **l'administrateur** et **le technicien**

Le technicien peut **signaler** qu'une batterie a été réparée, **enregistrer** les remplacements de batteries et **rédiger** des rapports techniques pour les interventions effectuées

L'administrateur en plus des fonctions de technicien peut gérer les batteries en les **ajoutant** au système, **consulter** la liste des batteries pour surveiller leurs états et planifier des interventions, et **mettre à jour** leurs informations en cas de changement.

L'accès au système est sécurisé, et chaque utilisateur doit s'authentifier pour accéder à ses fonctionnalités respectives.

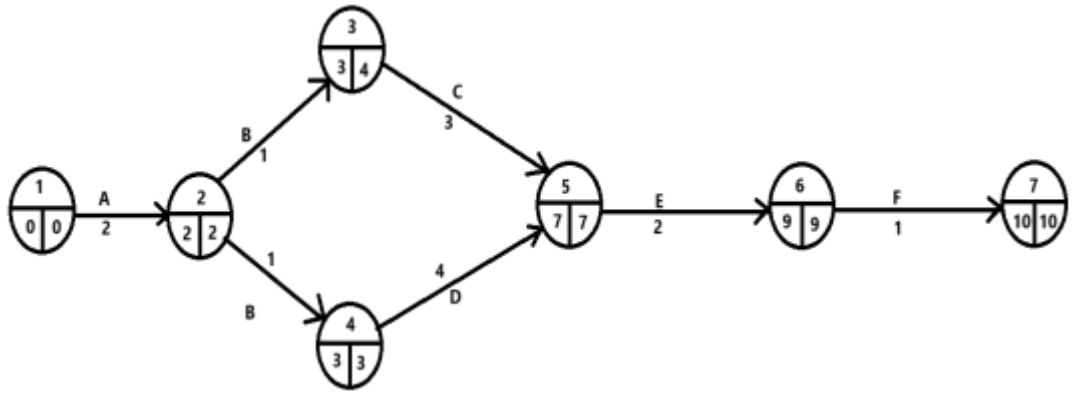


Dossier 3 : Approche Agile (07 pts)

Soit la liste des tâches suivantes :

Tache	Durée En jours	Tâches Antérieurs
A	2	-
B	1	A
C	3	B
D	4	B
E	2	C, D
F	1	E

1- Dresser le diagramme PERT (3pts)



2- Trouver le Chemin critique (1pts)

A → B → D → E → F.

3- Déterminer la durée d'exécution du projet (1pts)

10 jours

4- La tache C a pris 5 jours au lieu de 3 jours prévus : (2pts)

a. Déterminer le nouveau chemin critique

b. Déterminer la nouvelle durée du projet

Le nouveau chemin critique devient : **A → B → C → E → F**.

11 jours

Dossier 4 : Gestion de données NOSQL (11 pts)

1) Créer une base de données appelée **FastFoodDelivery** et une collection nommée **batteries** (1pt)

```
use FastFoodDelivery
db.createCollection("batteries")
```

2) Insérer les documents suivants dans la collection batteries : (2pts)

```
[
    {"_id":"1", "numéro_série":"s123", "capacité":"50%",
    "santé_batterie":"bonne", "nombre_cycles":500,"statut":"en service"},

     {"_id":"2", "numéro_série":"s345", "capacité":"70%",
     "santé_batterie":"moyenne", "nombre_cycles":600, "statut":"en panne"}
]
```

```
db.batteries.insertMany
([
{"_id":"1", "numéro_série":"s123", "capacité":"50%", "santé_batterie":"bonne",
"nombre_cycles":500, "statut":"en service"},
 {"_id":"2","numéro_série":"s345","capacité":"70%",
"santé_batterie":"moyenne", "nombre_cycles":600, "statut":"en panne"}
])
```

3) Ecrire une requête qui Modifier le champ **santé_batterie** à « faible » pour toutes les batteries dont le statut est **en panne** avec une **capacité** inférieure à 50% (2 pts)

```
db.batteries.updateMany
(
    { "statut": "en panne", "capacité": { "$lt": "50%" } },
     { "$set": { "santé_batterie": "faible" } }
 )
```

4) Afficher les attributs **numéro_série**, **capacité** et **statut** des batteries dont le nombre de cycle est entre 600 et 1000 classées par capacité décroissant (2 pts)

```
 db.batteries.find
(
   { "nombre_cycles": { "$gte": 600, "$lte": 1000 } },
   { "_id": 0, "numéro_série": 1, "capacité": 1, "statut": 1 }
).sort({ "capacité": -1 })
```

5) Afficher la moyenne des nombres de cycle des batteries par statut (2pts)

```
db.batteries.aggregate
([
   { $group: { _id: "$statut",
                 moyenne_cycles: { $avg: "$nombre_cycles" }
               }
   }
])
```

6) Supprimer les batteries dont le statut est en panne avec une capacité inférieure à 20% (2pts)

```
db.batteries.deleteMany( { "statut": "en panne", "capacité": { "$lt": "20%" } } )
```

Dossier 4 : Web Dynamique PHP (08 pts)

1) Donner le code PHP de création de la classe Batterie avec le champ capacité définit privée (1 pts)

```
<?php
        class Batterie
        {
           public $id;
           public $numero_serie;
           private $capacite;
           public $sante_batterie;
           public $nombre_cycles;
           public $statut;
        }
?>
```

2) Implémenter un getter **getCapacité** et un setter **setCapacité** pour la propriété capacité (1pts) :

```
        <?php
        class Batterie
        {
           public $id;
           public $numero_serie;
           private $capacite;
           public $sante_batterie;
           public $nombre_cycles;
           public $statut;
           //Question 2
           public function getCapacité()
           {return $this->capacite;}
           public function setCapacité($capacite)
           {$this->capacite=$capacite;}
        }
        ?>
```

3) Ajouter à la classe Batterie une méthode **verifierEntretienRequis**, Cette méthode doit vérifier si l'entretien est requis pour la batterie.
   Retourner **True** (entretien requis) si :
   La capacité est inférieure ou égale à 30%, ou Le nombre de cycles est supérieur ou égal à 800.

Sinon, retourne **False** (entretien non requis). (2pts)

```php
<?php
class Batterie
{
   public $id;
   public $numero_serie;
   private $capacite;
   public $sante_batterie;
   public $nombre_cycles;
   public $statut;
   //Question 2
   public function getCapacité()
   {return $this->capacite;}
   public function setCapacité($capacite)
   {$this->capacite=$capacite;}
   //Question 3
   public function verifierEntretienRequis()
   {
      if($this->capacite<="30%" and $this->nombre_cycles>=800 )
      {return true;}
      else {return false;}
   }
}
?>
```

4) Créer un tableau de 2 objets Batteries avec des données de votre choix (2pts)

```php
<?php
//Question 4
include "Batterie.php";
//Déclaration d'un tableau
$tab_batteries=array();
//Instancier un premier objet Batterie
$b1= new Batterie();
$b1->id=1;
$b1->numero_serie="n12";
$b1->setCapacité("20%");
$b1->sante_batterie="Bonne";
$b1->nombre_cycles=1000;
$b1->statut="En Service";
//Instancier un deuxième objet Batterie
$b2= new Batterie();
```

```
$b2->id=1;
$b2->numero_serie="n12";
$b2->setCapacité("20%");
$b2->sante_batterie="Bonne";
$b2->nombre_cycles=800;
$b2->statut="En Service";
//Ajouter les deux objets au tablaeu
array_push($tab_batteries,$b1);
array_push($tab_batteries,$b2);
?>
```

5) Parcourir le tableau de la question 4 pour afficher le nombre de batterie nécessitant un entretien en utilisant la méthode **verifierEntretienRequis** (2 pts)

```
<?php
//Question 4
include "Batterie.php";
//Déclaration d'un tableau
$tab_batteries=array();
//Instancier un premier objet Batterie
$b1= new Batterie();
$b1->id=1;
$b1->numero_serie="n12";
$b1->setCapacité("20%");
$b1->sante_batterie="Bonne";
$b1->nombre_cycles=1000;
$b1->statut="En Service";
//Instancier un deuxième objet Batterie
$b2= new Batterie();
$b2->id=1;
$b2->numero_serie="n12";
$b2->setCapacité("20%");
$b2->sante_batterie="Bonne";
$b2->nombre_cycles=800;
$b2->statut="En Service";
//Ajouter les deux objets au tablaeu
array_push($tab_batteries,$b1);
array_push($tab_batteries,$b2);
//Question 5
$nb_batterie_a_entretenir=0;
for ($i=0;$i<count($tab_batteries);$i++)
{
   if($tab_batteries[$i]->verifierEntretienRequis())
```

```
    {
        $nb_batterie_a_entretenir++;
    }
}
print("Le nombre de Batterie à entretenir = $nb_batterie_a_entretenir");
?>
```

Partie Pratique (60 pts)

Dossier 1 : Gestion des données MySQL (12 pts)

En se basant sur le schéma de base de données ci-dessus, écrire les scripts MySQL qui répondent aux questions suivantes :

1- Ecrire le script de la création de la table '**Vélo**'. **(2pts)**

```
CREATE TABLE Velo (
    id INT PRIMARY KEY AUTO_INCREMENT,
    Matricule VARCHAR(50) NOT NULL UNIQUE,
    id_batterie INT,
    date_derniere_maintenance DATE,
    date_prochaine_maintenance DATE,
    FOREIGN KEY (id_batterie) REFERENCES Batterie(id)
);
```

2- Modifier la table **'Vélo'** en ajoutant la colonne **'nb_batteries'** qui est de type entier positif et non nul (2pts)

```
ALTER TABLE Velo
ADD COLUMN nb_batteries INT UNSIGNED NOT NULL;
```

3- Ecrire un trigger **TR1** qui contrôle la valeur de la colonne **'sante_batterie'** qui doit être un nombre compris entre 0 et 100, lors de l'ajout d'une nouvelle batterie. **(2pts)**

```
DELIMITER $$

CREATE TRIGGER TR1 BEFORE INSERT ON Batterie
FOR EACH ROW
BEGIN
    IF NEW.sante_batterie < 0 OR NEW.sante_batterie > 100 THEN
```

```
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'La santé de la batterie doit être comprise
entre 0 et 100.';
    END IF;
END$$

DELIMITER ;
```

4- Ecrire un trigger **TR2,** qui modifie la valeur de la colonne statut de 'En Utilisation' à 'Retiré'. Lors d'un ajout dans la table **'Changement' (2pts)**

```
DELIMITER $$

CREATE TRIGGER TR2 AFTER INSERT ON Changement
FOR EACH ROW
BEGIN
    UPDATE Batterie
    SET statut = 'Retiré'
    WHERE id = NEW.id_ancienne_batterie;
END$$

DELIMITER ;
```

5- Ecrire une fonction **FCT1** qui retourne le nombre total de batteries consommées par un vélo dont l'id est passé en paramètre **(2pts)**

```
DELIMITER $$

CREATE FUNCTION FCT1(id_velo INT) RETURNS INT
DETERMINISTIC
BEGIN
    DECLARE total_batteries INT;
    SELECT COUNT(*) INTO total_batteries
    FROM Changement
    WHERE id_velo = id_velo;
    RETURN total_batteries;
END$$

DELIMITER ;
```

6- Gestion des utilisateurs/Rôles : **(2pts)**
   a. Créer le rôle **'RoleTechnicien'**

b. Donner les droits d'ajout, suppression et modification sur les tables '**Vélo'** et '**Changement'** au rôle 'Role*Technicien'*
c. Créer l'utilisateur 'ali' avec le mot de passe 0000
d. Attribuer le rôle 'RoleTechnicien' à l'utilisateur 'Ali' déjà crée

```
CREATE ROLE RoleTechnicien;

GRANT INSERT, DELETE, UPDATE ON Velo TO RoleTechnicien;
GRANT INSERT, DELETE, UPDATE ON Changement TO RoleTechnicien;

CREATE USER 'ali' IDENTIFIED BY '0000';

GRANT RoleTechnicien TO 'ali';
```

Dossier 2 : Développement Front End (24 pts)

Soit l'état initiale du ***store*** redux :

```
InitialState = {
velo :{
  id: 14,
  Matricule: "1 A 4321",
  Date_derniere_maintenance: "01/06/2025",
  Date_prochaine_maintenance: "01/08/2025",
  BatterieUtilisee: {
         Id : 135,
         Capacite : 73,
         Numero_serie : "BAT-202",
         sante_batterie: 95,
         nombre_cycles: 2127,
         statut: "En Utilisation",
          }
,Techniciens :[ {id :…,Nom :…}, {id :…,Nom :…},…],
,Batteries :[ {id :…,Numero_serie :…}, {id :…, Numero_serie:…},…],
,StatutBatterie :[ "En Stock","En Utilisation","Retiré"]
};
```

Remarque : Il n'est pas demandé à vous de créer le store Redux (actions, reducer, store).

1. Ecrire la composante **DetailsVelo.js** qui lit les informations du depuis le store Redux -En utilisant useSelector- et les affiche comme indiqué dans l'image ci-dessous. **(4pts)**

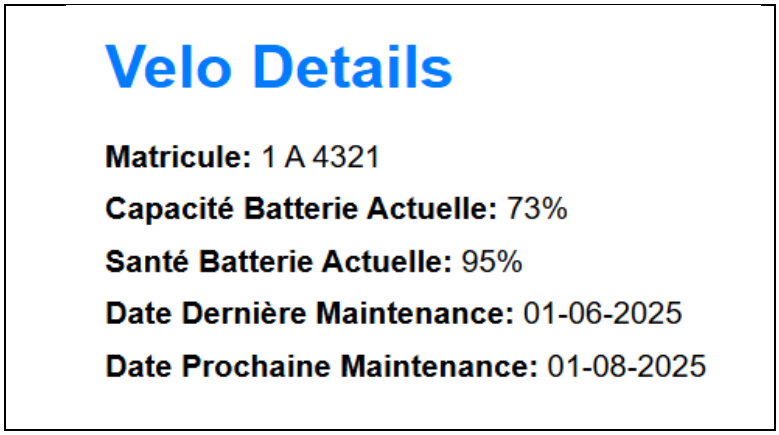


Figure 1- Détails d'un Vélo (Composante DetailsVelo.js)

```
import React from 'react';
import { useSelector } from 'react-redux';

const AfficherDetailsVelo = () => {
  const velo = useSelector(state => state.velo);

  return (
    <div>
      <h1>Détails du Vélo</h1>
      <p>ID: {velo.id}</p>
      <p>Matricule: {velo.Matricule}</p>
      <p>Date de dernière maintenance:
{velo.Date_derniere_maintenance}</p>
      <p>Date de prochaine maintenance:
{velo.Date_prochaine_maintenance}</p>
      <h2>Batterie Utilisée</h2>
      <p>ID: {velo.BatterieUtilisee.Id}</p>
      <p>Capacité: {velo.BatterieUtilisee.Capacite}</p>
      <p>Numéro de série: {velo.BatterieUtilisee.Numero_serie}</p>
      <p>Santé de la batterie:
{velo.BatterieUtilisee.sante_batterie}%</p>
      <p>Nombre de cycles: {velo.BatterieUtilisee.nombre_cycles}</p>
      <p>Statut: {velo.BatterieUtilisee.statut}</p>
    </div>
  );
};

export default AfficherDetailsVelo;
```

2. Ecrire le code de la composante **AjouterChangement.js** (voir figure 2) qui ajoute un changement de la batterie dans la base de données à travers l'API et en utilisant les données du sotre pour remplir le formulaire d'un nouveau changement de batterie de l'API Suivante : **(6pts)**

Méthode http	POST
URL de l'API	http://localhost:8000/api/AjouterChangement

Structure Objet	{
	" id_velo":...,
	"id_ancienne_batterie":...,
	"id_nouvelle_batterie":...,
	"date_changement":...,
	"id_technicien":...,
	"raison":...
	}

Figure 2- Ajout d'une changement d'une batterie

- Tous les champs sont obligatoires
- Les valeurs du champs (vélo, ancienne batterie,liste batteries,liste techniciens) sont lues depuis le store redux

```
import React, { useState } from 'react';
import { useSelector } from 'react-redux';
import axios from 'axios';

const AjouterChangementBatterie = () => {
  const { velo, Batteries, Techniciens } = useSelector(state => state);
  const [formData, setFormData] = useState({
    id_velo: velo.id,
    id_ancienne_batterie: velo.BatterieUtilisee.Id,
    id_nouvelle_batterie: '',
    date_changement: '',
    id_technicien: '',
    raison: ''
  });

  const gererChangement = (e) => {
    setFormData({ ...formData, [e.target.name]: e.target.value });
  };

  const soumettreFormulaire = async (e) => {
```

```
    e.preventDefault();
    try {
      const response = await
axios.post('http://localhost:8000/api/AjouterChangement', formData);
      console.log('Changement ajouté:', response.data);
      // Mettre à jour le store Redux si nécessaire
    } catch (error) {
      console.error('Erreur lors de l\'ajout du changement:', error);
    }
  };

  return (
    <div>
      <h1>Ajouter un Changement de Batterie</h1>
      <form onSubmit={soumettreFormulaire}>
        <label>
          Nouvelle Batterie:
          <select name="id_nouvelle_batterie"
value={formData.id_nouvelle_batterie} onChange={gererChangement}>
            {Batteries.map(batterie => (
              <option key={batterie.id}
value={batterie.id}>{batterie.Numero_serie}</option>
            ))}
          </select>
        </label>
        <label>
          Date de Changement:
          <input type="date" name="date_changement"
value={formData.date_changement} onChange={gererChangement} />
        </label>
        <label>
          Technicien:
          <select name="id_technicien" value={formData.id_technicien}
onChange={gererChangement}>
            {Techniciens.map(technicien => (
              <option key={technicien.id}
value={technicien.id}>{technicien.Nom}</option>
            ))}
          </select>
        </label>
        <label>
          Raison:
          <input type="text" name="raison" value={formData.raison}
onChange={gererChangement} />
        </label>
        <button type="submit">Ajouter</button>
      </form>
    </div>
  );
};
```

```
export default AjouterChangementBatterie;
```

3. Créer la composante **ListeBatteries**.js qui lit la liste des batteries depuis le store redux et l'affiche selon le schéma ci-dessous : **(6pts)**

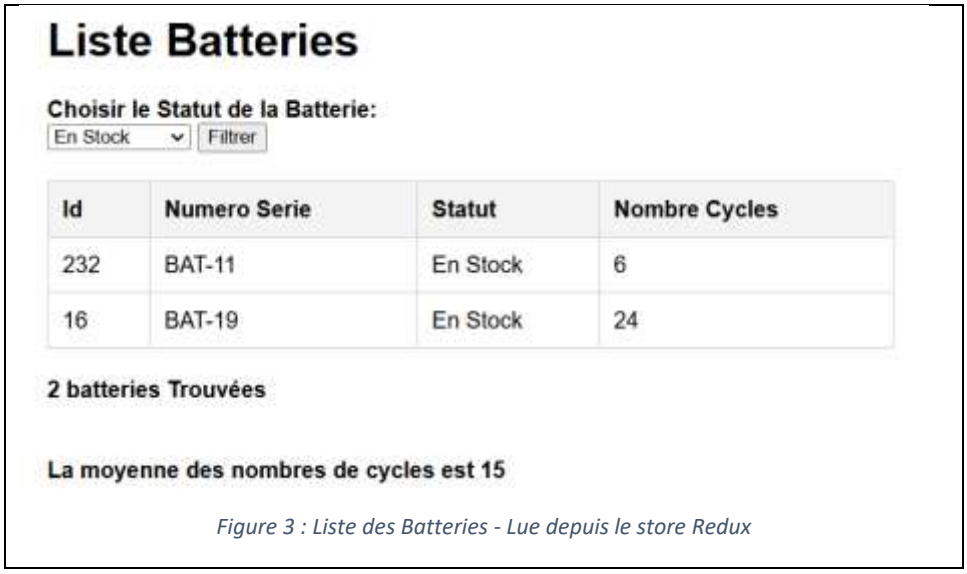


Figure 3 : Liste des Batteries - Lue depuis le store Redux

-La liste déroulante (select) lit les données depuis le store redux (StatutBatterie).

-Apres clique sur le bouton 'Filtrer' Les données sont filtrées selon l'option choisie au niveau de la liste déroulante.

-En Bas du tableau, on affiche le nombre total d'éléments trouvés ansi que la moyenne des nombres de cycles

```
import React, { useState } from 'react';
import { useSelector } from 'react-redux';

const AfficherListeBatteries = () => {
  const Batteries = useSelector(state => state.Batteries);
  const [filtre, setFiltre] = useState('');

  const filtrerBatteries = filtre ? Batteries.filter(batterie =>
batterie.statut === filtre) : Batteries;

  const totalElements = filtrerBatteries.length;
  const moyenneCycles = filtrerBatteries.reduce((somme, batterie) => somme +
batterie.nombre_cycles, 0) / totalElements || 0;

  return (
    <div>
      <h1>Liste des Batteries</h1>
      <select onChange={(e) => setFiltre(e.target.value)}>
        <option value="">Tous</option>
        <option value="En Stock">En Stock</option>
        <option value="En Utilisation">En Utilisation</option>
        <option value="Retiré">Retiré</option>
      </select>
      <button onClick={() => setFiltre('')}>Filtrer</button>
```

```
      <table>
        <thead>
          <tr>
            <th>ID</th>
            <th>Numéro de Série</th>
            <th>Statut</th>
          </tr>
        </thead>
        <tbody>
          {filtrerBatteries.map(batterie => (
            <tr key={batterie.id}>
              <td>{batterie.id}</td>
              <td>{batterie.Numero_serie}</td>
              <td>{batterie.statut}</td>
            </tr>
          ))}
        </tbody>
      </table>
      <p>Total: {totalElements}</p>
      <p>Moyenne des cycles: {moyenneCycles.toFixed(2)}</p>
    </div>
  );
};

export default AfficherListeBatteries;
```

4. Ecrire la composante **Menu.js** qui définit les liens qui mènent vers les composantes : **(4pts)**

Lien	Composante	Titre
/AjouterChgmt	AjouterChangement.js	Nouveau Changement
/ListeBatteries	ListeBatteries.js	Liste Batteries
/DetailsVélo	DetailsVelo.js	Détails Vélo

```
import React from 'react';
import { NavLink } from 'react-router-dom';

const Menu = () => {
  return (
    <nav>
      <ul>
        <li>
          <NavLink to="/AjouterChgmt" >
            Nouveau Changement
          </NavLink>
        </li>
        <li>
```

```
          <NavLink to="/ListeBatteries" >
            Liste Batteries
          </NavLink>
        </li>
        <li>
          <NavLink to="/DetailsVelo" >
            Détails Vélo
          </NavLink>
        </li>
      </ul>
    </nav>
  );
};

export default Menu;
```

5. Ecrire le code **App.js** qui : **(4pts)**
   - Définit le routage (BrowserRouter, Routes, …)
   - Appel la composante Menu.js crée ci-dessus.
   - Integer le provider du store redux

```
import React from 'react';
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';
import { Provider } from 'react-redux';
import store from './store';
import Menu from './Menu';
import AfficherDetailsVelo from './DetailsVelo';
import AjouterChangementBatterie from './AjouterChangement';
import AfficherListeBatteries from './ListeBatteries';

const App = () => {
  return (
    <Provider store={store}>
      <Router>
        <Menu />
        <Routes>
          <Route path="/AjouterChgmt" element={<AjouterChangementBatterie
/>} />
          <Route path="/ListeBatteries" element={<AfficherListeBatteries />}
/>
          <Route path="/DetailsVelo" element={<AfficherDetailsVelo />} />
        </Routes>
      </Router>
    </Provider>
  );
};

export default App;
```

Dossier 3 : Développement Back End (24 pts)

1) Donner la commande de création de migration pour la table batteries (1pt)

```
php artisan make:migration create_batteries_table
```

2) Donner le code des méthodes ***up*** et ***down*** du fichier de migration de la table ***batteries*** en définissant le champ ***numéro_série*** comme unique et le champ ***statut*** par défaut à « En service » (2pts)

```
public function up(): void
    {
        Schema::create('batteries', function (Blueprint $table) {
            $table->id();
            $table->timestamps();
            $table->string("numero_serie",50)->unique();
            $table->string("capacité",5);
            $table->string("sante_batterie",50);
            $table->integer("nombre_cycles");
            $table->string("statut",50)->default("En Service");

        });
    }

public function down(): void
    {
        Schema::dropIfExists('batteries');
    }
```

3) Donner la commande pour appliquer la migration (1pt)

```
php artisan migrate
```

4) Donner la commande de création de migration pour une modification de structure de la table batteries (1pt)

```
php artisan make:migration add_column_to_batteries_table
ou
php artisan make:migration modify_batteries_table --table=batteries
```

5) Donner le code de la méthode up et down pour ajouter un index sur la colonne statut et changer le type de données de la colonne capacité en chaine de caractère de taille 15 (2pts)

```
public function up(): void
    {
        Schema::table('batteries', function (Blueprint $table) {
            $table->index('statut');
            $table->string('capacité',15)->change();

        });
    }


public function down(): void
    {
         Schema::table('batteries', function (Blueprint $table) {
            $table->dropIndex('batteries_statut_index');
            $table->string('capacité',5)->change();
        });
    }
```

6) Créer les modèles des tables **batteries** et **vélos** avec leurs attributs et fonctions de relations (2pts)

```
php artisan make:model Velo

class Velo extends Model
{
    use HasFactory;
    public function batterie()
    {
        return $this->belongsTo(Batterie::class);
    }
}

Php artisan make:model Batterie

class Batterie extends Model
{
    use HasFactory;
    public function velos()
    {
        return $this->hasMany(Velo::class);
    }
}
```

7) Donner la commande artisan de création du contrôleur **VéloController** (2pts)

```
php artisan make:controller VéloController
```

8) Dans le contrôleur **VéloController** créer la méthode **GetBatterie** qui retourne les informations d'une batterie d'un vélo dont l'id est passé en paramètre (2pts)

```
<?php
namespace App\Http\Controllers;
use Illuminate\Http\Request;
use App\Models\Batterie;

class VéloController extends Controller
{
   function GetBatterie($veloId)
   {
        $velo = Velo::with('batterie')->find($veloId);
      return $velo->batterie ;
   }
}
```

9) Créer une méthode **AfficherBatterie** qui retourner une vue nommée **InfoBatterie** avec les données de la méthode **GetBatterie** de la question 8 (2pts)

```
<?php
namespace App\Http\Controllers;
use Illuminate\Http\Request;
use App\Models\Batterie;

class VéloController extends Controller
{
   function GetBatterie($veloId)
   {
        $velo = Velo::with('batterie')->find($veloId);
      return $velo->batterie ;
   }

   function AfficherBatterie($veloId)
   {
      return view("InfoBatterie")->with("info_batterie",$this->GetBatterie($veloId));
   }
}
```

10) Créer la vue **InfoBatterie** dans **le contrôleur VéloController** qui affiche les informations de la Batterie sous forme d'un tableau comme le montre l'exemple ci-dessous avec les liens modifier et supprimer (2pts)

Id	Numéro série	Capacité	Santé batterie	Nombre cycle	Statut	Modifier	Supprimer
1	S1	80	Bonne	500	En service	Modifier	Supprimer

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <meta http-equiv="X-UA-Compatible" content="ie=edge">
   <title>Document</title>
</head>
<body>
   <table>
      <tr>
         <td>Id</td>
         <td>Numéro série</td>
         <td>Capacité</td>
         <td>Santé batterie</td>
         <td>Nombre cycle</td>
         <td>Statut</td>
         <td>Modifier</td>
         <td>Supprimer</td>
      </tr>

      <tr>
         <td>{{$info_batterie->id]}}</td>
         <td>{{$info_batterie->numero_serie}}</td>
         <td>{{$info_batterie->capacité}}</td>
         <td>{{$inof_batterie->sante_batterie}}</td>
         <td>{{$info_batterie->nombre_cycles}}</td>
         <td>{{$info_batterie->statut}}</td>
          <td>{{$info_batterie->sante_batterie}}</td>
<td><a href="modifier?id={{$info_batterie->id}}">Modifier</a></td>
<td><a href="supprimer?id={{$infobatterie->id}}">Modifier</a></td>
         </tr>
   </table>
</body>
</html>
```

11) Dans le contrôleur **VéloController** créer la méthode **SuuprimerBattrie** qui supprime une batterie dont l'id est passé en paramètre et qui redirige l'utilisateur vers la vue **Welcome** (2pts)

```
<?php
namespace App\Http\Controllers;
use Illuminate\Http\Request;
use App\Models\Batterie;

class VéloController extends Controller
{
  function GetBatterie($veloId)
   {
       $velo = Velo::with('batterie')->find($veloId);
      return $velo->batterie ;
   }

   function AfficherBatterie($veloId)
   {
      return view("InfoBatterie")->with("info_batterie",$this->GetBatterie($veloId));
   }
   function SuuprimerBattrie(Request $request)
   {
      Batterie::destroy($request->id);
      return redirect()->route('welcome')->with('success', 'Batterie supprimée avec succès');
   }
}
```

Dans ce qui suit on suppose qu'un système d'authentification est déjà installé sur le projet et que les utilisateurs sont déjà créés dans la base de données

12) Donner la commande pour créer le middleware sous le nom «**autorisationSuppression**» (1 pt)

```
php artisan make:middleware autorisationSuppression
```

13) Donner le corps de la méthode **handle** pour que le middleware **autorisationSuppression** accepte les requêtes provenant de l'utilisateur « toto » dans le cas contraire on affiche un code d'erreur 403 avec le message « vous n'êtes pas autorisé à effectuer cette opération de suppression » (2pts)

```php
<?php
namespace App\Http\Middleware;
use Closure;
use Illuminate\Http\Request;
use Symfony\Component\HttpFoundation\Response;
class autorisationSuppression
{
   public function handle(Request $request, Closure $next): Response
   {
      if(auth()->user()->name=='toto')
      {
         return $next($request);
      }
      else
      {
          abort(403, 'vous n'êtes pas autorisé à effectuer cette opération de suppression');
      }

   }
}
```

14) Donner le code de la route qui mène vers l'action de suppression de batterie en la protégeant par le middleware «**autorisationSuppression**» (2pts)

```php
Route::get('/supprimer',[VéloController::class,'SuuprimerBattrie'])
    ->middleware("autorisationSuppression");
```