BORDEAUX INP Ensc

CO6SFCW0 : Communication Web

Compte Rendu de Projet

Objectif : Réaliser tout ou partie d'une application web garantissant la gestion de notes d'une école.

BOYER Simon, ELBAZ Evan, UNG Maxime

Groupe 1 - 1A

Promotion 2027

Table des matières :

Introduction :

Réalisé dans le cadre du module Communication Web à l'ENSC, ce projet consiste à développer une application web de gestion de notes scolaires, inspirée du fonctionnement de services existants comme Pronote, largement utilisés dans les établissements secondaires. L'objectif est de mettre en œuvre une chaîne complète React (Front-End) – PHP (API) – MySQL (Base de données) afin de concevoir une interface simple et fonctionnelle, permettant aux élèves de consulter leurs résultats, et aux professeurs d'accéder à l'ensemble des notes, avec des droits différenciés selon le type d'utilisateur.

Ce rapport présente dans un premier temps les fonctionnalités développées, puis détaille la structure du code en React, PHP et SQL. Il expose ensuite les choix de gestion de projet, avant de proposer un bilan global.

I - Fonctionnalités :

L'application développée propose une interface de consultation des notes scolaires, avec un accès différencié selon deux profils d'utilisateurs : élève et professeur. Elle répond aux exigences fonctionnelles du cahier des charges, en assurant une navigation claire et un affichage adapté aux droits de chaque utilisateur.

1) Sélection du profil utilisateur

L'utilisateur choisit son profil via une barre de navigation située en haut de l'interface. Deux boutons permettent d'accéder respectivement à l'espace Élève ou Professeur, conditionnant les champs de connexion et les données visibles après authentification.

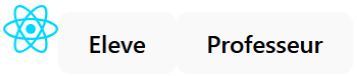

Figure 1 : Barre de navigation pour le choix du profil utilisateur

2) Connexion élève

L'élève saisit son identifiant (prénom) et son mot de passe. Après validation, il accède à ses résultats personnels. Ces derniers sont affichés sous forme d'un tableau comportant les colonnes suivantes : le nom, la matière et la note obtenue. L'élève a uniquement accès à ses propres notes en consultation.

Figure 2 : Espace de connexion et affichage des notes avec le profil élève

3) Connexion professeur

Le professeur se connecte via un mot de passe unique, commun à tous les enseignants. Aucun identifiant n'est requis. Une fois authentifié, il accède à l'ensemble des notes de tous les élèves, présentées également sous forme de tableau. Cela permet une vue d'ensemble des résultats, utile pour le suivi pédagogique.

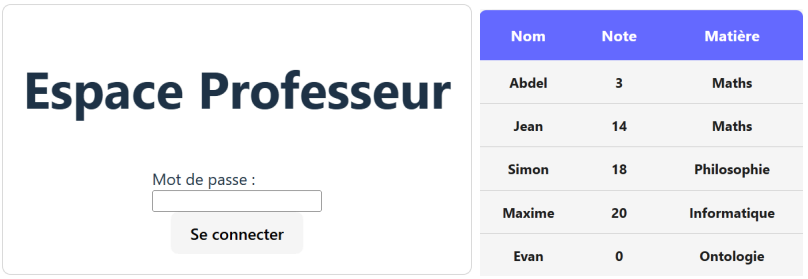

Figure 3 : Espace de connexion et affichage des notes avec le profil professeur

4) Affichage et retour utilisateur

Les résultats sont présentés sous forme de tableau à trois colonnes (Nom, Matière, Note). En cas d'absence de données ou d'erreur d'identifiants, un message d'erreur explicite est affiché. Des alertes de formulaire préviennent également l'utilisateur si des champs obligatoires ne sont pas renseignés, garantissant une interaction fluide et sans ambiguïté.

Figure 4 : Exemple de message d'erreur en cas d'identifiant non renseigné ou incorrect

II - Structure du code :

1) Structure générale

L'application repose sur une architecture à trois niveaux clairement séparés, articulée autour des fichiers `App.jsx` et `App.css`, `api.php`, ainsi que notes.sql. La logique d'ensemble suit un schéma web classique client - serveur - base de données (BDD), mis en œuvre à travers React pour le front-end, PHP

pour le traitement serveur, et MySQL pour le stockage de la BDD. Cette structuration garantit une lisibilité optimale du fonctionnement global, tout en facilitant la maintenance et les évolutions futures.

Le fichier `App.jsx` constitue le point d'accès principal au front-end. Il coordonne l'affichage des composants de l'application et la gestion des interactions utilisateur, notamment via les hooks React (`useState`, `useEffect`). Il organise la navigation, la saisie des informations de connexion et l'interrogation de l'API pour récupérer les données, sans implémenter directement de logique métier ou de traitement de données. Le fichier `App.css`, lié au composant principal, gère la mise en forme générale de l'application. Il fournit une présentation visuelle cohérente et lisible, sans influer sur le comportement fonctionnel de l'application.

Le fichier `api.php` fait office d'interface entre le front-end et la BDD. Il reçoit les paramètres passés via l'URL en utilisant la méthode GET, établit la connexion MySQL avec la BDD, effectue les requêtes SQL appropriées en fonction du type d'utilisateur, puis renvoie les résultats au format JSON. Ce fichier assure ainsi la logique de traitement serveur et la sécurisation des accès aux données.

Enfin, le fichier `notes.sql` définit la structure de la table Notes ainsi que son contenu initial. Il centralise les informations nécessaires à l'affichage (nom, matière, note) et à la gestion des droits d'accès (mot de passe).

2) Front en REACT

a) Navigation et Affichage conditionnel

Tout d'abord afin d'avoir un visuel d'entrée sur le site, nous avons utilisé un `useEffect` afin d'afficher un message de bienvenue sur notre site à l'utilisateur.

Ensuite nous avons utilisé un state `page` pouvant contenir deux valeurs `Eleve` et `Professeur`. En fonction de la valeur contenu dans ce state, un formulaire différent est proposé. Pour comprendre plus en détail ce processus, détaillons le fonctionnement de chaque fonction.

D'abord `action1(x)` permet de mettre le state `page` à la valeur `x`. Ici, le `x` sera `Eleve` ou `Professeur`.

Ensuite la fonction `Navbar` est une balise de passage. Elle reçoit `{action1}` en props et le transmet à `action2` tout en définissant un label comme étant `Eleve` ou `Professeur`. Ensuite `action2` passe dans `action3`.

C'est maintenant au tour du composant `Bouton` d'entrer en jeu. Celle-ci reçoit `action3` et définit la fonction `action4` comme `action3(label)`. Or, `action3` fait référence à `action2` qui fait référence à `action1` et donc à `setPage`.

Donc `action4` définit le state `page` à la valeur `label`, i.e. soit `Eleve` soit `Professeur`. Donc en résumé jusqu'à présent, lorsque l'on clique sur le bouton `Eleve` ou `Professeur` (cf `Navbar`), le state `page` se met à la valeur `Eleve` ou `Professeur`.

À partir de là, le composant `Contenu` reçoit `{page}` en props et exerce une condition sur `page`. Si `page` vaut `Eleve` (donc si l'utilisateur a cliqué sur le bouton `Eleve`) alors le formulaire pour élève est affiché. De même, si `page` vaut `Professeur` (donc si l'utilisateur a cliqué sur le bouton `Professeur`) alors le formulaire pour professeur est affiché.

b) Interrogation de l'API

Enfin, une structure permet d'interroger l'API et de récupérer dynamiquement les données depuis le serveur. Ce mécanisme est géré par la fonction `fetchFormulaire(identifiant, motDePasse)` dans le composant principal App. Cette fonction construit une URL d'interrogation en fonction du type d'utilisateur (`Eleve` ou `Professeur`) et y encode les identifiants transmis via le formulaire. Pour un élève, les deux champs `identifiant` et `motDePasse` sont inclus dans l'URL, tandis que pour un professeur seul le mot de passe est transmis.

L'appel se fait via `fetch()` en JavaScript, qui envoie une requête HTTP GET vers l'API codée en PHP. Le résultat attendu est une réponse au format JSON. Cette réponse est ensuite analysée avec `response.json()` et utilisée pour mettre à jour le `state data` avec `setData(data)`. Ces données sont ensuite affichées sous forme de tableau HTML dans la partie inférieure de la page.

3) API en PHP

a) Initialisation

L'API commence par l'instruction `header("Access-Control-Allow-Origin: *")`, qui autorise les requêtes provenant d'autres origines, notamment depuis le front-end. Les paramètres transmis dans l'URL (`identifiant` et `motDePasse`) sont ensuite récupérés à l'aide de la superglobale `$_GET` et stockés dans les variables `$identifiant` et `$motDePasse`. Leur présence détermine le traitement qui sera effectué.

b) Connexion à la base de données (BDD)

La connexion à la BDD est établie via l'objet `$bdd = new PDO()`. Deux configurations sont prévues : une connexion principale via zzz et une connexion en localhost en commentaire pour un usage hors-ligne en cas d'impossibilité de connexion au zzz. Le bloc `try`/`catch` assure la gestion des erreurs et renvoie un message JSON explicite en cas d'échec de connexion.

c) Traitement des requêtes

Le traitement des requêtes repose sur une structure conditionnelle :

- Si `$identifiant` est vide et que `$motDePasse` correspond au mot de passe enseignant (`"prof123"`), une requête est exécutée via `$bdd->query('SELECT * FROM Notes')` pour extraire toutes les lignes de la table Notes.
- Si les deux paramètres sont fournis, une requête est utilisée via `prepare('SELECT * FROM Notes WHERE name = :identifiant AND mdp = :motDePasse')`, suivie de `execute(['identifiant' => $identifiant, 'motDePasse' => $motDePasse])`, pour extraire uniquement les résultats de l'élève identifié.
- En cas d'identifiants invalides, un message d'erreur est généré avec `json_encode()`.

d) Réponse

Pour les 2 profils, les résultats sont récupérés avec `fetchAll(PDO::FETCH_ASSOC)` et stockés dans `$resultats`. Toutes les réponses sont encodées en JSON via `json_encode()`, ce qui permet une exploitation directe par le front-end en React. Le script renvoie également des messages d'erreur en JSON en cas d'identifiants incorrects ou d'erreur de connexion à la BDD.

4) Base de données (BDD) en SQL

Pour la base de données nous avons créé la table "notes" que nous avons défini avec plusieurs attributs : "id" de type INT pour l'identifiant de l'élève, "name" de type TEXT pour le nom de l'élève, "subject" de type TEXT pour la matière, "note" de type DOUBLE pour la note, et "mdp" de type TEXT pour le mot de passe de l'élève. Tous ces attributs doivent être non-nuls. Nous avons aussi défini l'attribut "id" comme clé primaire de la table "notes", s'auto-incrémentant à chaque ligne. Quant aux mots de passe, nous les avons construits sur le modèle "nom+id".

Ensuite pour remplir la table, nous avons choisi des attributs plus ou moins aléatoires jusqu'à avoir cinq élèves différents.

III - Gestion de Projet :

1) Planification et outils

Le projet s'est déroulé sur une période de trois semaines et demie, entre le 23 avril et le 18 mai. Le diagramme de Gantt reflétant notre progression est présenté en tableau 1.

Travail à faire	S17 (23/04 - 27/04)	S18 (18/04 - 04/05)	S19 (05/05 - 11/05)	S20 (12/05 - 18/05)
Séances de TP encadrées				
Idéation structure de l'application				
Création de la base de donnée en SQL				
Structuration de la chaîne React-API-BDD				
Codage du Front-end en React				
Codage de l'API en PHP				
Assemblage complet et tests				
Rédaction du rapport				

Tableau 1 : Diagramme de Gantt du projet

Les séances de TP encadrées ont servi de points de rendez-vous pour la coordination du groupe. Elles ont été utilisées pour faire le point sur les avancées individuelles en testant ensemble les différents modules et organiser la suite du travail. En dehors de ces séances, chaque membre a travaillé en autonomie tout en communiquant régulièrement via WhatsApp pour partager les avancées et répartir les tâches. Un Google Drive commun a été utilisé pour centraliser les fichiers du projet, les codes sources, ainsi que les documents du rapport.

2) Répartition des tâches

La répartition des tâches a reposé sur les compétences individuelles de chacun tout en favorisant une collaboration transversale.

- La base de données a été codée par Evan, avec la participation du groupe pour la conception, notamment pour déterminer la structure et les valeurs initiales.
- Le Front-end a principalement été développé par Simon et Evan. Maxime a complété cette partie en gérant l'intégration avec l'API.
- L'API a été développée par Maxime, avec l'appui du groupe pour la définition des paramètres et des scénarios de requêtes.

Les fonctionnalités et l'architecture globale ont été discutées en commun. Tous les membres ont également participé aux tests, à la vérification de la cohérence entre les différentes couches de l'application et au debugging final.

Bilan :

Ce projet a permis de concevoir une application web fonctionnelle de consultation de notes scolaires, en mobilisant l'ensemble des compétences introduites dans le module Communication Web. La chaîne complète React – API PHP – base de données MySQL a été implémentée avec succès, garantissant l'interopérabilité des trois couches. L'application permet la sélection du profil utilisateur, la connexion sécurisée (avec identifiant et mot de passe pour l'élève, mot de passe unique pour le professeur), ainsi que l'affichage des résultats sous forme de tableau, avec un retour adapté en cas d'erreur.

Sur le plan collaboratif, le projet a été mené de façon rigoureuse et progressive. Chaque membre s'est investit dans une ou plusieurs briques du système tout en veillant à la compréhension mutuelle des parties développées. Cette dynamique a favorisé une réelle montée en compétence collective, conformément aux attendus pédagogiques du module.

L'application, bien qu'elle réponde uniquement à la partie consultation du cahier des charges, constitue une base solide et évolutive. À l'avenir, des fonctionnalités supplémentaires pourraient être envisagées, telles que l'ajout ou la modification de notes par les professeurs, une gestion différenciée par matière ou groupe, ou encore un système de notifications pour les élèves. Ces évolutions pourraient enrichir l'outil tout en approfondissant l'apprentissage des concepts abordés.

Annexes :

1) Code du Front en React

```
import { useState, useEffect } from 'react';
import reactLogo from './assets/react.svg';
import './App.css';

// Composant Bouton
function Bouton(props)
{
  const action4 = () => {props.action3(props.label);}; // Appelle la
fonction passée en props avec le label (Élève ou Professeur)
  return (<button onClick={action4}>{props.label}</button>);
}

// Composant Navbar
function Navbar(props)
{
  return (
    <div>
      <img

src="https://static.vecteezy.com/ti/vecteur-libre/p1/14861990-creation-de-lo
go-sp-initiale-sp-lettre-logo-design-monogramme-vector-design-pro-vecteur-ve
ctoriel.jpg"
        alt="Logo SuperPronote"
        width="200"
        height="auto"
      />
      <div>
        <Bouton label="Eleve" action3={props.action2} />
        <Bouton label="Professeur" action3={props.action2} />
      </div>
    </div>
  );
}

// Composant Contenu : Affiche le formulaire selon la page actuelle (Élève
ou Professeur)
function Contenu(props) {
```

```
  if (!["Eleve", "Professeur"].includes(props.pageActuelle))
    {return null;}

  const [identifiant, setIdentifiant] = useState('');
  const [motDePasse, setMotDePasse] = useState('');

  const soumissionFormulaire = (event) => {
    event.preventDefault(); // Empêche le rechargement complet de la page
    if ((props.pageActuelle === "Eleve" && identifiant && motDePasse) ||
        (props.pageActuelle === "Professeur" && motDePasse))
          {props.actionFormulaire(identifiant, motDePasse);} // Envoie les
identifiants au serveur
    else
      {alert('Veuillez remplir tous les champs.');}
  };


  if (props.pageActuelle === "Eleve")
  {
    return (
      <div className="page">
        <div className="contenu">
          <h1>Espace Élève</h1>
          <form onSubmit={soumissionFormulaire}>
            <div className="champ">
              <label>Identifiant :</label>
              <input type="text" value={identifiant} onChange={(event) =>
setIdentifiant(event.target.value)} required />
            </div>
            <div className="champ">
              <label>Mot de passe :</label>
              <input type="password" value={motDePasse} onChange={(event) =>
setMotDePasse(event.target.value)} required />
            </div>
            <button type="submit">Se connecter</button>
          </form>
        </div>
      </div>
    );
  }
  else if (props.pageActuelle === "Professeur")
    {
```

```
      return (
        <div className="page">
          <div className="contenu">
            <h1>Espace Professeur</h1>
            <form onSubmit={soumissionFormulaire}>
              <div className="champ">
                <label>Mot de passe :</label>
                <input type="password" value={motDePasse} onChange={(event)
=> setMotDePasse(event.target.value)} required />
              </div>
              <button type="submit">Se connecter</button>
            </form>
          </div>
        </div>
      );
    }
    else
    {
      return (
        <div className="page">
          <p>Page inconnue</p>
        </div>
      );
    }
  }


// Composant principal App
function App() {
  const [data, setData] = useState([]);
  const [page, setPage] = useState("");
  const [message, setMessage] = useState("");

  useEffect(() => {setMessage("Bienvenue sur SuperPronote");}, []);

  // Fonction pour changer de page
  function action1(x)
    {setPage(x);}

  // Met à jour le titre de la page quand "page" change
  useEffect(() => {document.title = `La page : ${page}`;}, [page]);
```

```
   // Fonction pour envoyer les identifiants à l'API et récupérer les
données
    const fetchFormulaire = (identifiant, motDePasse) => {

      //let url = "http://localhost:8080/api.php"; // Interrogation API
localhost
      let url = "https://mung001.zzz.bordeaux-inp.fr/api.php"; //
Interrogation API zzz
      if (page === "Eleve")
        {url +=
`?identifiant=${encodeURIComponent(identifiant)}&motDePasse=${encodeURICompo
nent(motDePasse)}`;}
      else if (page === "Professeur")
        {url += `?motDePasse=${encodeURIComponent(motDePasse)}`;} // pas
d'identifiant pour le prof

      console.log("URL envoyée à l'API :", url);
      fetch(url)
        .then(response => response.json())
        .then(data => {
          if (data.error)
            {alert(data.error);}
          else
          {setData(data);}
        })
        .catch(error => {
          console.error('Erreur:', error);
          alert('Erreur serveur.');
        });
    };

  return (
    <>
      <Navbar action2={action1} />
      {page === "" && <h1>{message}</h1>}
      <Contenu pageActuelle={page} actionFormulaire={fetchFormulaire} />
      { ["Eleve", "Professeur"].includes(page) && (
      <table>
        <thead>
          <tr>
            <th>Nom</th>
            <th>Note</th>
            <th>Matière</th>
```

```
          </tr>
        </thead>
        <tbody>
          {Array.isArray(data) && data.length > 0 ? (
            data.map((item) => (
              <tr key={item.id}>
                <th>{item.name}</th>
                <th>{item.note}</th>
                <th>{item.subject}</th>
              </tr>
            ))
          ) : (
            <tr>
              <td colSpan="3">Aucune donnée à afficher</td>
            </tr>
          )}
        </tbody>
      </table>
      )}
    </>
  );
}

export default App;
```

2) Code de l'API en PHP

```
<?php
header("Access-Control-Allow-Origin: *");

// Récupération des paramètres GET facultatifs (issus du formulaire React)
$identifiant = $_GET['identifiant'] ?? null;
$motDePasse = $_GET['motDePasse'] ?? null;

// Connexion à la base de données version full localhost
/*$host = 'localhost';
$dbname = 'notes';
$username = 'root';
$password = '';*/
```

```php
// Connexion à la base de données version zzz
$host = 'localhost';
$dbname = 'mung001';
$username = 'mung001';
$password = 'sibdd@B_INP27';

try
    {$bdd = new PDO('mysql:host=' . $host . ';dbname=' . $dbname .
';charset=utf8', $username, $password);}
catch (Exception $e)
    {die(json_encode(['error' => 'Erreur de connexion à la base de
données']));}

// Page professeurs : si seul le mot de passe est fourni et correspond à
celui des professeur
if (!$identifiant && $motDePasse === 'prof123')
{
    $requete = $bdd->query('SELECT * FROM `Notes`');
    $resultats = $requete->fetchAll(PDO::FETCH_ASSOC);
    echo json_encode($resultats);
}

// Page élèves : si identifiant et mot de passe sont fournis et
correspondent à celui d'un élève de la BDD
elseif ($identifiant && $motDePasse)
{
    $requete = $bdd->prepare('SELECT * FROM Notes WHERE name = :identifiant
AND mdp = :motDePasse');
    $requete->execute([
        'identifiant' => $identifiant,
        'motDePasse' => $motDePasse
    ]);

    $resultats = $requete->fetchAll(PDO::FETCH_ASSOC);

    if ($resultats)
        {echo json_encode($resultats);} // Envoie des résultats si présents
    else
        {echo json_encode(['error' => 'Identifiant ou mot de passe
incorrect']);}

}
```

```
else
    {echo json_encode(['error' => 'Identifiant ou mot de passe
incorrect']);}
?>
```

3) Code de la Base de Donnée en SQL

```
--
-- Structure de la table `Notes`
--

CREATE TABLE `Notes` (
  `id` int NOT NULL AUTO_INCREMENT,
  `name` text NOT NULL,
  `subject` text NOT NULL,
  `note` double NOT NULL,
  `mdp` text NOT NULL,
  PRIMARY KEY (id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;

--
-- Déchargement des données de la table `Notes`
--

INSERT INTO `Notes` (`ID`, `Name`, `Subject`, `Note`, `mdp`) VALUES
(1, 'Abdel', 'Maths', 3, 'Abdel1'),
(2, 'Jean', 'Maths', 14, 'Jean2'),
(3, 'Simon', 'Philosophie', 18, 'Simon3'),
(4, 'Maxime', 'Informatique', 20, 'Maxime4'),
(5, 'Evan', 'Ontologie', 0, 'Evan5');
```