

Projet d'analyse des vulnérabilités d'une application web

4 année Cybersécurité

2024/2025

Encadré par

Mr. Mehdi Tmimi

Préparée par :

Wissal BOUTAYEB

Table des Matières

1. Introduction

1.1 Présentation du projet

1.2 Objectifs

2. Analyse du Code

2.1 Présentation des fonctionnalités de l'application

2.2 Architecture de l'application

2.3 Vulnérabilités et faiblesses identifiées

2.3.1 Authentification et gestion des mots de passe

Failles dans la gestion des mots de passe

2.3.2 Gestion des sessions

2.3.3 Validation et assainissement des entrées utilisateur

Absence de validation des entrées utilisateur

2.3.4 Gestion des fichiers statiques et contrôle d'accès

Risques liés à l'accès aux fichiers protégés

2.3.5 Gestion des erreurs et journalisation

2.3.6 Logique de gestion des tâches

3. Développement des Fichiers Statiques

3.1 Dossier Public

3.1.1 login.html - Conception et fonctionnalités

3.1.2 register.html - Conception et fonctionnalités

3.2 Dossier Privé/Protégé

3.2.1 index.html - Vue d'ensemble du tableau de bord

3.2.2 dashboard.html - Fonctionnalités de gestion des tâches

3.3 Mise en page et style CSS

3.4 Implémentation JavaScript

3.4.1 Utilisation de l'API **Fetch** pour l'intégration backend

3.4.2 Manipulation du **DOM** pour les mises à jour dynamiques

4. package et bibliothèque installer

Conclusion

6.1 Résumé des problèmes identifiés

6.2 Bilan des améliorations proposées

Introduction

1.1 Présentation du projet

Ce projet consiste en une application web simple développée en **Node.js**, simulant un système de gestion de tâches à petite échelle. L'application inclut des fonctionnalités telles que l'inscription et la connexion des utilisateurs, la gestion des tâches et la gestion des sessions utilisateur. Elle utilise des fichiers statiques pour servir des pages publiques (login et inscription) et privées (tableau de bord et gestion des tâches) L'objectif principal de ce projet est de fournir une plateforme d'apprentissage où les utilisateurs peuvent identifier des vulnérabilités de sécurité et des problèmes de stabilité intentionnellement intégrés dans le code.

1.2 Objectifs

Les objectifs de cette analyse sont multiples :

- **Identifier les vulnérabilités de sécurité** : on est censée d'Analyser le code pour détecter les failles de sécurité telles que les problèmes d'authentification, les failles XSS (Cross-Site Scripting), les injections SQL, ou encore les mauvaises pratiques de gestion des sessions.
- **Améliorer la fiabilité de l'application** : par Identification des scénarios d'erreurs non pris en charge et proposer des solutions pour renforcer la robustesse du système.
- **Proposer des améliorations** : Recommandation des mesures de sécurité et des optimisations pour rendre l'application plus sécurisée et plus stable.
- **Créer des fichiers statiques modernes** : on est censée Développer des pages HTML et CSS conviviales, accompagnées de scripts JavaScript dynamiques pour une meilleure expérience utilisateur.

Création des fichiers HTML nécessaires pour les dossiers publics et privés/protégés :

- **login.html** : Une page pour connexion.
- **register.html** : Une page d'inscription avec un formulaire.
- **index.html** : La page principale (tableau de bord) pour les utilisateurs connectés.
- **dashboard.html** : Une page dédiée à la gestion des tâches.
- **Intégration d'un tableau de bord fonctionnel avec deux sections** :
 - Une section d'ajout de tâches via un formulaire.
 - Une section d'affichage des tâches sous forme de tableau récupéré depuis le serveur.Utilisation uniquement **JavaScript Vanilla**, l'API Fetch et la manipulation du DOM pour implémenter les fonctionnalités.
- **Documenter les Vulnérabilités et les Erreurs**
 - Identifier et expliquer chaque problème détecté.
 - Proposer des solutions pour atténuer les impacts des vulnérabilités et des erreurs.

Fournir un Tableau de Bord Fonctionnel

- Implémenter une interface utilisateur intuitive permettant l'ajout et l'affichage des tâches, entièrement développée avec **JavaScript Vanilla** et stylisée avec un design moderne en CSS.

Analyse du Code

2.1 Présentation des fonctionnalités de l'application :

L'application est un système de gestion de tâches qui offre les fonctionnalités suivantes :

Inscription et Connexion des utilisateurs : Permet aux utilisateurs de créer un compte et de se connecter avec un nom d'utilisateur et un mot de passe.

Gestion des tâches : Les utilisateurs connectés peuvent ajouter, afficher et supprimer des tâches.

Gestion des sessions : L'application utilise des sessions pour authentifier les utilisateurs et protéger les routes sensibles.

Accès aux fichiers statiques : Les fichiers publics (login et inscription) sont disponibles à tous, tandis que les fichiers protégés nécessitent une authentification.

2.2 Architecture de l'application :

L'application est structurée comme suit :

Routes publiques :

POST /register : Inscription d'un nouvel utilisateur.

POST /login : Authentification d'un utilisateur existant.

GET /public/login.html et GET /public/register.html : Fournissent les pages de connexion et d'inscription.

Routes protégées :

GET /dashboard : Tableau de bord pour les utilisateurs authentifiés.

POST /add : Ajout d'une tâche.

GET /tasks : Récupération des tâches d'un utilisateur.

DELETE /tasks/:id : Suppression d'une tâche par ID.

GET /logout : Déconnexion de l'utilisateur.

2.3 Vulnérabilités et faiblesses identifiées :

2.3.1 Authentification et gestion des mots de passe

Problème : Les mots de passe des utilisateurs sont stockés en clair dans le fichier JSON (`database.json`). En cas de fuite de données, cela expose directement les mots de passe.

```
data.users.push({ username, password })
```

Risk et Impact : Risque élevé de compromission des comptes.

Solution proposée : Utilisation d'une bibliothèque comme **bcrypt** pour hacher les mots de passe avant de les stocker avec vérification des mots de passe lors de la connexion en comparant le hash stocké avec celui généré à partir de l'entrée de l'utilisateur.

Exemple de l'utilisation de la bibliothèque **bcrypt** :

```
const bcrypt = require('bcrypt');

const hashedPassword = await bcrypt.hash(password, 10);

data.users.push({ username, password: hashedPassword });
```

Lors de la connexion, on doit vérifier le mot de passe en utilisant la fonction **bcrypt.compare()**.

2.3.2 Manque de validation des entrées utilisateur :

Problème : Les données envoyées par les utilisateurs (nom d'utilisateur, mot de passe) ne sont pas validées.

Register route:

```
app.post('/register', async (req, res) => {

  const {username, password} = req.body;

  try{

    const data = await readData();

    const userExists = data.users.find((u) => u.username === username);

    if (userExists) {

      return res.status(400).send('User already exists');

    } data.users.push({ username, password });

    await writeData(data);

    res.status(201).send('User registered successfully');

  } catch (err) {

    return res.status(500).send('Server error');

  }

});
```

Risque et Impact : on peut créer ou insérer des utilisateurs avec des noms invalides ou malveillants. Cela peut exposer à des attaques par injection (par exemple, en injectant du code JavaScript ou SQL dans les champs (**Utilisateur,password**)).

Solution Proposée : Utilisation des outils comme **express-validator** pour valider les entrées :

express-validator: est une bibliothèque Node.js utilisée pour valider et normaliser les données entrées par les utilisateurs dans les applications Express.js. Elle permet de :

1. **Valider les données :** Vérifie si les entrées respectent des règles

2. **Sanitiser les données** : Nettoie les entrées pour éviter les attaques comme l'injection des codes malveillantes SQL ou XSS.
3. **Simplifier la gestion des erreurs** : Génère des messages d'erreur détaillés pour guider les utilisateurs.

Exemple d'utilisation de cette bibliothèque dans notre code :

```
const validator = require('validator');
```

```
app.post('/register', async (req, res) => {  
  const {username, password} = req.body;  
  if (!validator.isAlphanumeric(username) || !validator.isLength(password, { min: 8 })) {  
    return res.status(400).send('Invalid input');  
  }  
});
```

Route de connexion (POST /login) :

```
app.post('/login', async (req, res) => {  
  const { username, password } = req.body;  
  try {  
    const data = await readData();  
    const user = data.users.find((u) => u.username === username && u.password === password);  
    if (user) {  
      req.session.user = user;  
      return res.redirect('/protected/index.html');  
    } else {  
      return res.status(401).send('Invalid credentials');  
    }  
  }  
});
```

- **Problème** : Les entrées ne sont pas validées avant d'être utilisées pour rechercher dans la base de données. Cela ouvre la voie à des attaques par injection.
- **Solution proposée** : on doit valider les champs username et password avec **express-validator** (comme dans la solution précédente).

Ajout de tâches (POST /add) :

```
app.post('/add', async (req, res) => {  
  if (!req.session.user) {  
    return res.status(401).send('Unauthorized');  
  }  
  
  const { task } = req.body;
```

```

try {
  const data = await readData();
  const taskId = Date.now();
  data.tasks.push({ id: taskId, task, user: req.session.user.username });
  await writeData(data);
  res.redirect('/protected/dashboard.html');
} catch (err) {
  return res.status(500).send('Server error');
}
});

```

Problème :

- L'entrée **task** n'est pas validée avant d'être ajoutée à la liste. Cela pourrait permettre à un utilisateur malveillant de soumettre des scripts ou des caractères.

Solution proposée :

- Ajouter une validation des entrées pour les tâches (comme une vérification de longueur et de contenu).
- Utilisation d'une bibliothèque comme **DOMPurify** côté client cette bibliothèque permettent de nettoyer les données si elles sont affichées dans le DOM.

Exemple d'utilisation de DOMPurify :

```

const createDOMPurify = require('dompurify');

const window = new JSDOM("").window;

const DOMPurify = createDOMPurify(window);

app.post('/add', (req, res) => { let { task } = req.body; // Nettoyer l'entrée de la tâche pour éviter les
scripts malveillants

task = DOMPurify.sanitize(task);

```

Suppression de tâches (DELETE /tasks/:id) :

```

app.delete('/tasks/:id', async (req, res) => {

  if (!req.session.user) {

    return res.status(401).send('Unauthorized');

  }

  const id = parseInt(req.params.id, 10);

  try { const data = await readData();

    data.tasks = data.tasks.filter(t => t.id !== id);

    await writeData(data);

    res.send('Task deleted');

  } catch (err) {

    return res.status(500).send('Server error');
  }
});

```

Problème : Le paramètre id n'est pas validé avant d'être utilisé.

Solution proposée : on doit valider l'id de l'utilisateur pour s'assurer qu'il s'agit d'un entier valide

Exemple d'utilisation :

```
app.delete('/tasks/:id', [ check('id').isInt().withMessage('L'ID doit être un entier.')
```

2.3.3 Gestion des sessions :

Problème : La clé secrète utilisée dans express-session : **'mysecretkey'** est définie en clair dans le code. Ainsi que Les cookies de session n'ont pas les options de sécurité activées.

RISQUE : Si quelqu'un accède au code source, il peut compromettre la sécurité des sessions, en plus Les cookies non sécurisés sont exposés aux attaques de type **man-in-the-middle** (MITM)., cela peut impliquer Accès non autorisé aux comptes utilisateur. Compromission de données sensibles.

Solution proposée : On doit stocker la clé secrète dans une variable d'environnement tout en ajoutant les options de sécurité nécessaire pour protéger les cookies

Exemple d'implémentation :

```
const sessionSecret = process.env. SESSION_SECRET || 'fallbackSecret';

app.use(
  session({
    secret: sessionSecret,
    resave: false,
    saveUninitialized: true,
    cookie: {secure: true, httpOnly: true, sameSite: 'strict' },
  })
);
```

Cette clé (**SESSION_SECRET**) est utilisée pour signer et valider les cookies de session. Elle garantit que les cookies n'ont pas été modifiés par un tiers.

- **Cookie :** { secure: **true** } : permet d'assurer que les cookies ne sont pas transmis en clair sur le réseau. et permettent d'empêcher l'interception des cookies sur des connexions non sécurisées (attaque de type **man-in-the-middle**).

cookie: { httpOnly: **true** }

Description : permettent de rendre les cookies inaccessibles depuis JavaScript côté client.

- **Rôle en sécurité :** Empêcher d'ajouter ou d'injecter de code JavaScript malveillant, permet aussi d'empêcher ces scripts malveillants d'accéder aux cookies de session, ceci réduit les risques de vol de session.

cookie: { sameSite: '**strict**' }

Description : Limite les envois de cookies uniquement aux requêtes provenant du même site.

Rôle en sécurité : permettent de protéger contre les attaques **Cross-Site Request Forgery (CSRF)** tout en empêchant les cookies d'être inclus dans des requêtes provenant de sites tiers.

Le mode **strict** empêchant même l'envoi de cookies lors de navigations intersites.

2.3.4 Accès aux fichiers protégés

Problème :

Le middleware de protection des routes utilise uniquement une session pour vérifier l'authentification, sans autre couche de validation ou de restriction.

Risque : Accès non autorisé aux données ou aux fichiers protégés.

Solution proposée : on doit Ajouter une couche de vérification basée sur des **JSON Web Tokens (JWT)** pour sécuriser les routes.

Développement des Fichiers Statiques

3.1 Dossier Public

Ces deux fichiers permettent à un utilisateur de s'authentifier

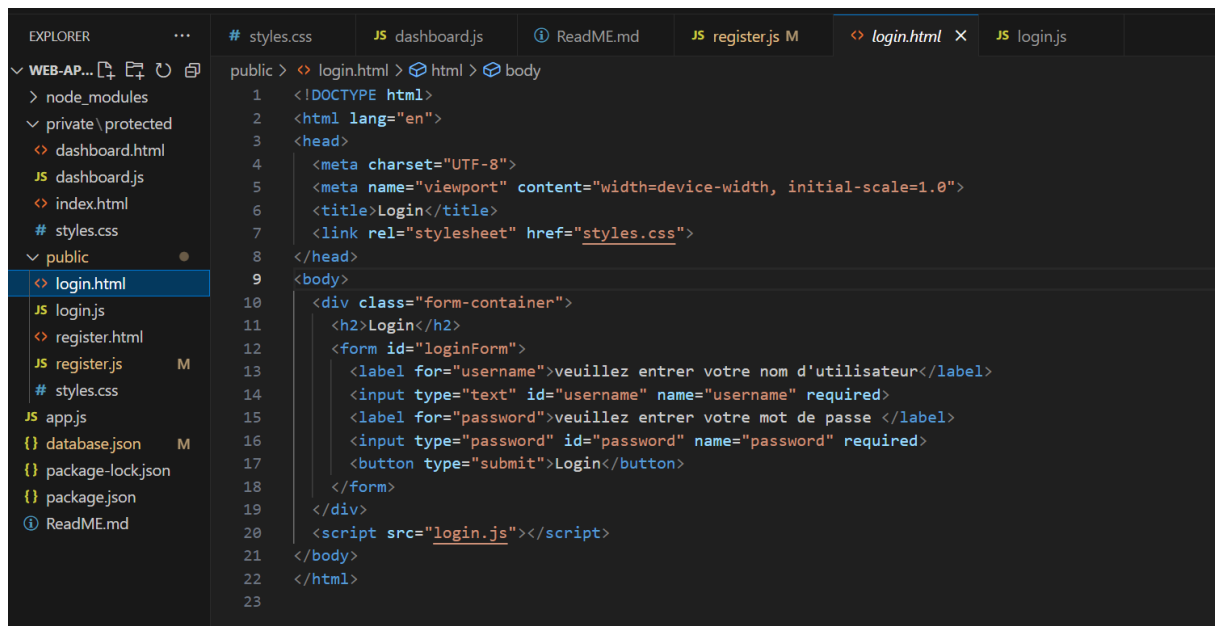
login.html : Une page de connexion conviviale.

register.html : Une page d'inscription contenant un formulaire

La page **register.html** développé :

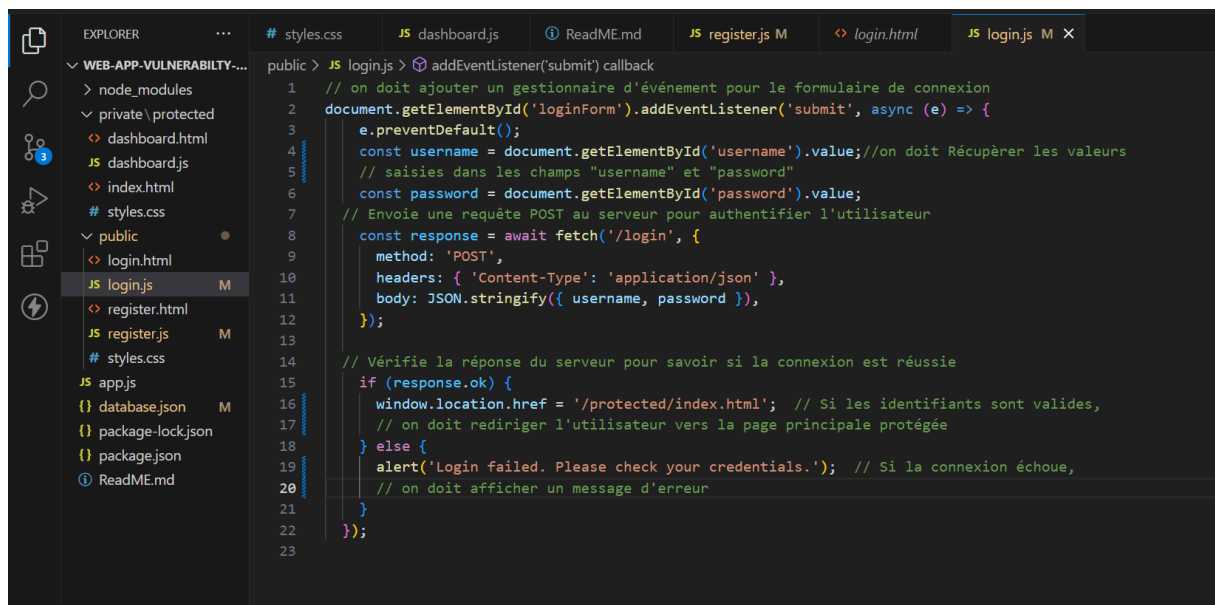
La page **register.js** développé :

La page **login.html** développé



```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Login</title>
7   <link rel="stylesheet" href="styles.css">
8 </head>
9 <body>
10  <div class="form-container">
11    <h2>Login</h2>
12    <form id="loginForm">
13      <label for="username">veuillez entrer votre nom d'utilisateur</label>
14      <input type="text" id="username" name="username" required>
15      <label for="password">veuillez entrer votre mot de passe</label>
16      <input type="password" id="password" name="password" required>
17      <button type="submit">Login</button>
18    </form>
19  </div>
20  <script src="login.js"></script>
21 </body>
22 </html>
23
```

La page **login.js** développé



```
1 // on doit ajouter un gestionnaire d'événement pour le formulaire de connexion
2 document.getElementById('loginForm').addEventListener('submit', async (e) => {
3   e.preventDefault();
4   const username = document.getElementById('username').value; // on doit récupérer les valeurs
5   // saisies dans les champs "username" et "password"
6   const password = document.getElementById('password').value;
7   // Envoie une requête POST au serveur pour authentifier l'utilisateur
8   const response = await fetch('/login', {
9     method: 'POST',
10    headers: { 'Content-Type': 'application/json' },
11    body: JSON.stringify({ username, password }),
12  });
13
14  // Vérifie la réponse du serveur pour savoir si la connexion est réussie
15  if (response.ok) {
16    window.location.href = '/protected/index.html'; // Si les identifiants sont valides,
17    // on doit rediriger l'utilisateur vers la page principale protégée
18  } else {
19    alert('Login failed. Please check your credentials.');// Si la connexion échoue,
20    // on doit afficher un message d'erreur
21  }
22 });
23
```

Le fichier **styles.css** contient le CSS de notre **register.html** et **login.html** pages

```

public > # styles.css > 🔍 form
1  body {
2      font-family: Arial, sans-serif;
3      margin: 0;
4      padding: 0;
5      background-color: #f4f4f9;
6  }
7  .form-container, .dashboard-container {
8      max-width: 400px;
9      margin: 140px auto;
10     padding: 40px;
11     background: white;
12     border-radius: 50px;
13     box-shadow: 0 3px 4px rgba(0, 0, 0, 0.2);
14 }
15 form {
16     display: flex;
17     flex-direction: column;
18 }
19 label {
20     margin-bottom: 7px;
21 }
22 input {
23     margin-bottom: 16px;
24     padding: 12px;
25     border: 2px solid #ccc;
26     border-radius: 6px;
27 }
28 button {
29     padding: 15px;
30     background-color: #6497ce;

```

```

    border-radius: 6px;
}
button {
    padding: 15px;
    background-color: #6497ce;
    color: white;
    border: none;
    border-radius: 5px;
    cursor: pointer;
}
button:hover {
    background-color: #69a2e0;
}
table {
    width: 100%;
    border-collapse: collapse;
}
thead {
    background-color: #5e9ad9;
    color: rgb(248, 226, 226);
}
td, th {
    padding: 20px;
    border: 2px solid #ddd;
    text-align: center;
}
}

```

La page register.html

localhost:3000/public/register.html

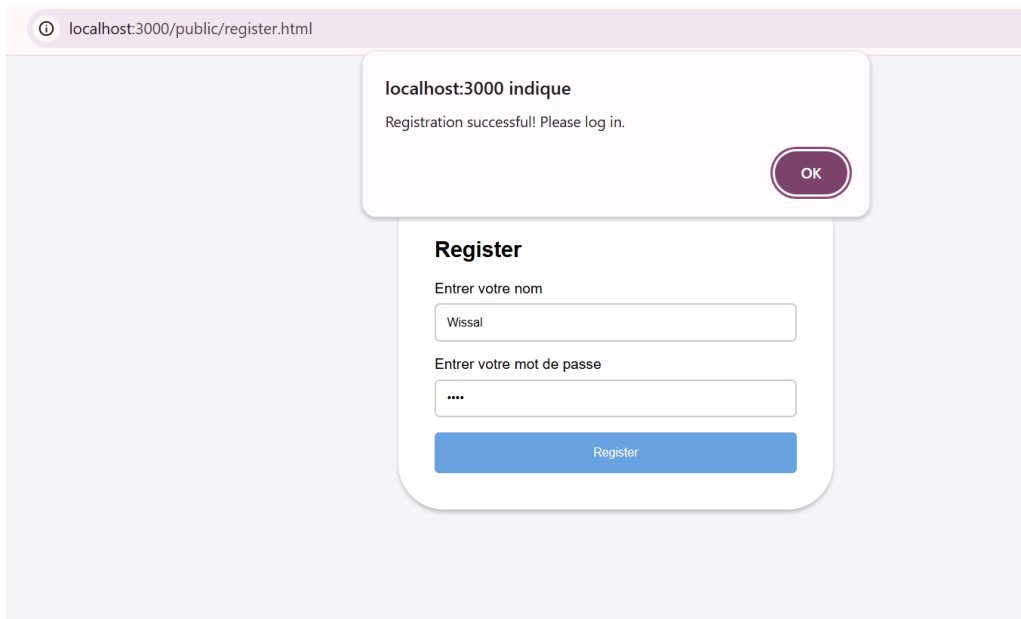
Register

Entrer votre nom

Entrer votre mot de passe

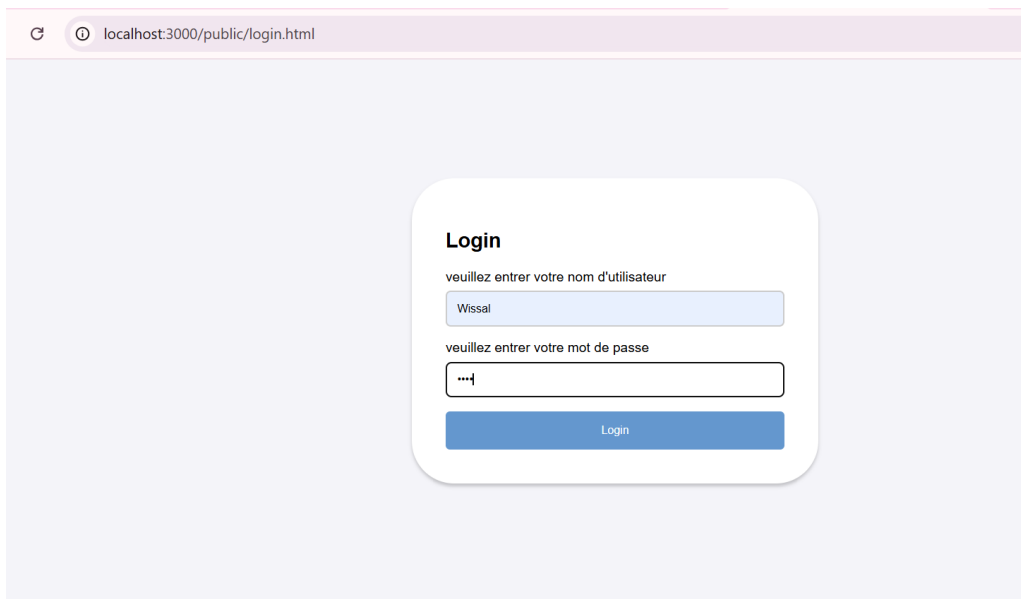
Register

L'authentification d'un nouvel utilisateur :



The screenshot shows a web browser window with the address bar displaying "localhost:3000/public/register.html". A light purple notification box at the top center contains the text "localhost:3000 indique" and "Registration successful! Please log in.", with an "OK" button. Below the notification is a "Register" form. The form has two input fields: "Entrer votre nom" with the value "Wissal" and "Entrer votre mot de passe" with masked characters "....". A blue "Register" button is at the bottom of the form.

La page **login.html**



The screenshot shows a web browser window with the address bar displaying "localhost:3000/public/login.html". The page features a "Login" form. The form has two input fields: "veuillez entrer votre nom d'utilisateur" with the value "Wissal" and "veuillez entrer votre mot de passe" with masked characters "....". A blue "Login" button is at the bottom of the form.

L'enregistrement de connexion de cette utilisateur dans la base donnée **database.json** tout en stockant les mots de passe crypté haché(en utilisant la bibliothèque **bcrypt**)

Database.json

```

{} database.json > [ ] tasks
1  {
2    "users": [
3      {
4        "username": "user1",
5        "password": "password1"
6      },
7      {
8        "username": "user2",
9        "password": "password2"
10     },
11     {
12       "username": "Maryam",
13       "password": "$2b$10$oJf7aw7oAHR6ae5hvXYqBOYHh5n0dgSk4WCHzZ/gBJPR3QKwtVSpC"
14     },
15     {
16       "username": "BOUTAYEB",
17       "password": "$2b$10$AM8Wkn..r6g/hpTdGTQewOzI.iwID0qSQzS1AyhfqD0s910gFbZky"
18     },
19     {
20       "username": "Wissal",
21       "password": "$2b$10$eb2iEt2DP0hg3VoP0dLPO.ca2xb4rv04AZwKvHavXapTohzGgaVxe"
22     }
23   ],
24   "tasks": [
25     {
26       "id": 1674220800000,
27       "task": "Buy groceries",
28       "user": "user1"
29     },
30   ]

```

3.2 Dossier Privé/Protégé

3.2.1 index.html - Vue d'ensemble du tableau de bord

3.2.2 dashboard.html - Fonctionnalités de gestion des tâches

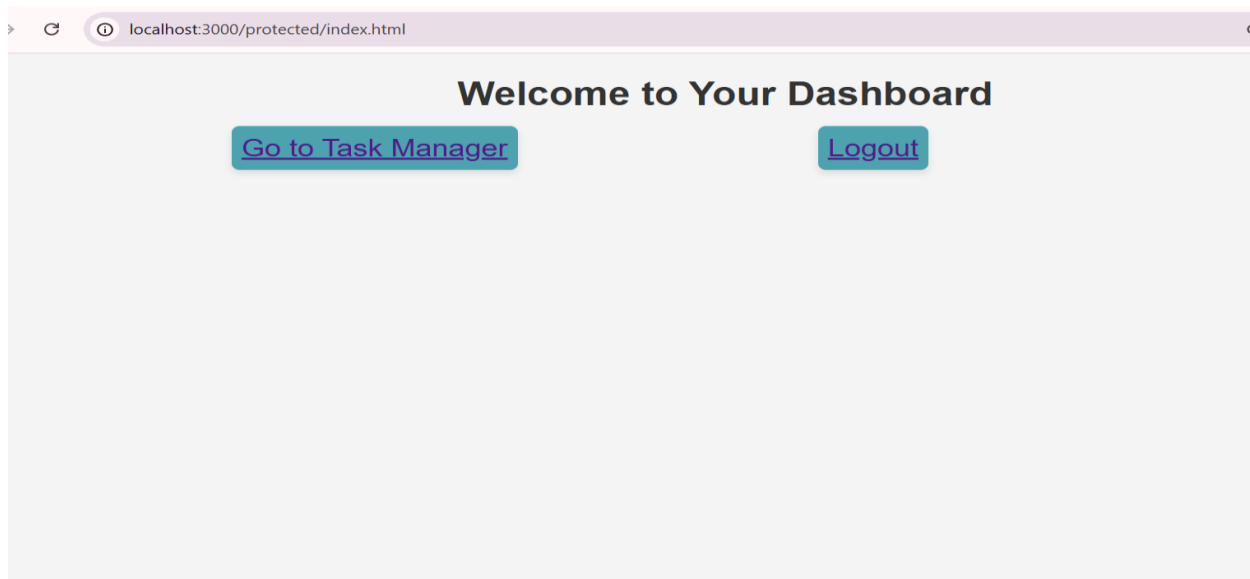
La page **index.html** développé

Après que l'utilisateur s'authentifie on le dirige vers la page **index.html** qui Contient deux liens une permet à l'utilisateur de se diriger vers la page dashboard.html (qui englobe les fonctionnalités d'ajouter, affichage et suppression des Taches) l'autre lien permet à l'utilisateur de se déconnecté dans ce cas l'utilisateur se dirige vers la page de login(**login.html**)

```

# styles.css JS dashboard.js ReadME.md JS register.js M {} database.json M index.html X JS login.js M
private > protected > index.html > html > body
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4    <meta charset="UTF-8">
5    <meta name="viewport" content="width=device-width, initial-scale=1.0">
6    <title>Dashboard</title>
7    <link rel="stylesheet" href="styles.css">
8  </head>
9  <body>
10   <h1>Welcome to Your Dashboard</h1>
11   <a href="/protected/dashboard.html">Go to Task Manager</a>
12   <a href="/logout">Logout</a>
13 </body>
14 </html>
15

```



La page **dashboard.html** englobe les fonctionnalités d'ajout, affichage, suppression et stockage des Taches dans la base de données en utilisant Vanilla.js

```
# styles.css JS dashboard.js ReadME.md JS register.js M database.json M dashboard.html X JS login.js M
private > protected > dashboard.html > html > body > div.dashboard-container > table#taskTable > thead > tr > th
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Task Manager</title>
7   <link rel="stylesheet" href="styles.css">
8 </head>
9 <body>
10   <div class="dashboard-container">
11     <h2>Tableaux de Gestion des Taches Task Manager</h2>
12     <form id="addTaskForm">
13       <label for="task">Ajouter Nouveaux Tache </label>
14       <input type="text" id="task" name="task" required>
15       <button type="submit">Add Task</button>
16     </form>
17     <h3>Your Tasks</h3>
18     <table id="taskTable">
19       <thead>
20         <tr>
21           <th>Task</th>
22           <th>Actions</th>
23         </tr>
24       </thead>
25       <tbody></tbody>
26     </table>
27   </div>
28   <script src="dashboard.js"></script>
29 </body>
30 </html>
```

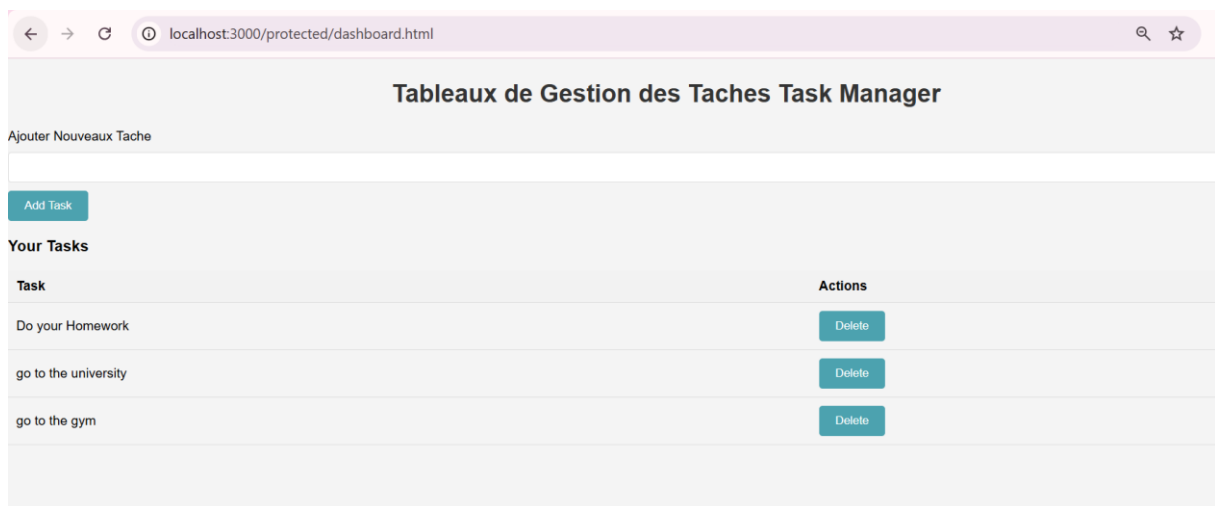
La page **dashboard.js** englobe Utilisation de l'API **Fetch** pour l'intégration backend et la Manipulation du **DOM** pour les mises à jour dynamiques

```
# styles.css JS dashboard.js ReadME.md JS register.js M {} database.json M <> dashboard.html JS login.js M
private > protected > JS dashboard.js > addEventListener('submit') callback
1 async function fetchTasks() { // pour Récupérer et afficher les tâches depuis le serveur
2   const response = await fetch('/tasks');// Requête pour récupérer les tâches
3   if (response.ok) {
4     const tasks = await response.json();
5     const tbody = document.querySelector('#taskTable tbody');// pour Sélectionner le corps du tableau
6     tbody.innerHTML = '';
7     tasks.forEach((task) => {
8       const row = document.createElement('tr');// pour Créer une nouvelle ligne pour le tableau
9       row.innerHTML = `
10         <td>${task.task}</td>
11         <td>
12           <button onclick="deleteTask(${task.id})">Delete</button>
13         </td>
14       `;
15       tbody.appendChild(row); // pour Ajouter la ligne au tableau
16     });
17   }
18 }
19 document.getElementById('addTaskForm').addEventListener('submit', async (e) => {
20   e.preventDefault();
21   const task = document.getElementById('task').value;//pour Récupérer la valeur de la tâche saisie
22   const response = await fetch('/add', {
23     method: 'POST', // Envoie une requête POST
24     headers: { 'Content-Type': 'application/json' },
25     body: JSON.stringify({ task }),// l'Envoie de la tâche au format JSON
26   });
27
28   if (response.ok) {
29     fetchTasks();
30     document.getElementById('task').value = '';
```

```
26   });
27
28   if (response.ok) {
29     fetchTasks();
30     document.getElementById('task').value = '';
31   } else {
32     alert('Failed to add task.');
```

```
33   }
34 }
35 // Supprime une tâche par son ID
36 async function deleteTask(taskId) {
37   const response = await fetch(`/tasks/${taskId}`, { method: 'DELETE' });// Envoie une requête DELETE
38   if (response.ok) {
39     fetchTasks();
40   } else {
41     alert('Failed to delete task.');
```

```
42   }
43 }
44 // Récupération et affichage des tâches lors de chargement de la page
45 fetchTasks();
46
```

L'ajout automatique de ces taches dans la base données :

```

} database.json > [ ] tasks
24   "tasks": [
44   ],
45   {
46     "id": 1737939431395,
47     "task": "visit your Manager",
48     "user": "Maryam"
49   },
50   {
51     "id": 1737996328505,
52     "task": "doo",
53     "user": "BOUTAYEB"
54   },
55   {
56     "id": 1738002561444,
57     "task": "Do your Homework",
58     "user": "Wissal"
59   },
60   {
61     "id": 1738002599332,
62     "task": "go to the university",
63     "user": "Wissal"
64   },
65   {
66     "id": 1738002619503,
67     "task": "go to the gym",
68     "user": "Wissal"
69   }
70 ]
71 }

```

Suppression des Taches :

←

→

↺

localhost:3000/protected/dashboard.html

🔍 ☆

Tableaux de Gestion des Taches Task Manager

Ajouter Nouveaux Tache

Add Task

Your Tasks

Task	Actions
Do your Homework	Delete

```
# styles.css | JS dashboard.js | ReadME.md | JS register.js M | {} database.json M X | JS login.js M
{} database.json > [ ] tasks > {} 4
24   "tasks": [
35     {
37       "task": "Complete project",
38       "user": "user2"
39     },
40     {
41       "id": 1674220830000,
42       "task": "Plan vacation",
43       "user": "user2"
44     },
45     {
46       "id": 1737939431395,
47       "task": "visit your Manager",
48       "user": "Maryam"
49     },
50     {
51       "id": 1737996328505,
52       "task": "doo",
53       "user": "BOUTAYEB"
54     },
55     {
56       "id": 1738002561444,
57       "task": "Do your Homework",
58       "user": "Wissal"
59     }
60   ]
61 }
```

Le fichier **styles.css** développé pour **index.html** et **dashboard.html**

```
private > protected > # styles.css > {} @media (max-width: 768px)
1  /* Global Styles */
2  body {
3    font-family: Arial, sans-serif;
4    margin: 0;
5    padding: 0;
6    background-color: #f4f4f4;
7  }
8  body h1{
9    text-align: center;
10   font-size: 40px;
11 }
12 body a{
13   text-align: center;
14   font-size: 30px;
15   margin-left: 300px;
16   padding-left: 300px;
17   border-radius: 8px;
18   background-color: #4ca2af;
19   box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
20   padding: 10px;
21 }
22 .container {
23   width: 80%;
24   margin: 0 auto;
25   padding: 20px;
26 }
27 h1, h2 {
28   color: #333;
29 }
30 /* Dashboard Styles */
```

private > protected > # styles.css > {} @media (max-width: 768px)

```
30 /* Dashboard Styles */
31 .dashboard {
32     background-color: #ffffff;
33     padding: 20px;
34     border-radius: 8px;
35     box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
36 }
37 .dashboard-container h2{
38
39     text-align: center;
40     font-size: 30px;
41 }
42 .dashboard-header {
43     display: flex;
44     justify-content: space-between;
45     align-items: center;
46     margin-bottom: 20px;
47 }
48 .dashboard-header h1 {
49     margin: 0;
50 }
51 .add-task-section, .task-display-section {
52     margin-top: 20px;
53 }
54 button {
55     background-color: #4ca2af;
56     color: white;
57     border: none;
58     padding: 10px 20px;
59     cursor: pointer;
```

private > protected > # styles.css > {} @media (max-width: 768px)

```
54 button {
57     border: none;
58     padding: 10px 20px;
59     cursor: pointer;
60     border-radius: 4px;
61     font-size: 14px;
62 }
63 button:hover {
64     background-color: #4585a0;
65 }
66 input, textarea {
67     width: 100%;
68     padding: 10px;
69     margin: 10px 0;
70     border: 1px solid #ddd;
71     border-radius: 4px;
72 }
73 form {
74     margin-bottom: 20px;
75 }
76 /* Task Table Styles */
77 table {
78     width: 100%;
79     border-collapse: collapse;
80 }
81 table th, table td {
82     padding: 10px;
83     text-align: left;
84     border-bottom: 1px solid #ddd;
85 }
```

```
# styles.css • JS dashboard.js • ReadME.md JS register.js M {} database.json M <> dashboard.html
private > protected > # styles.css > {} @media (max-width: 768px)
86   table th {
87     background-color: #f2f2f2;
88   }
89   table tr:hover {
90     background-color: #f1f1f1;
91   }
92   .delete-btn {
93     background-color: red;
94     color: white;
95     padding: 5px 10px;
96     border: none;
97     cursor: pointer;
98     border-radius: 4px;
99   }
100  .delete-btn:hover {
101    background-color: #e60000;
102  }
103  /* Responsive Design */
104  @media (max-width: 768px) {
105    .container {
106      width: 95%;
107    }
108    .dashboard-header {
109      flex-direction: column;
110    }
111    .task-display-section table, .add-task-section {
112      width: 100%;
113    }
114  }
115
```

Package et bibliothèque installer :

```
PS C:\Users\smart lenovo\Documents\web-app-vulnerability-analysis--project>
PS C:\Users\smart lenovo\Documents\web-app-vulnerability-analysis--project> npm init --y
Wrote to C:\Users\smart lenovo\Documents\web-app-vulnerability-analysis--project\package.json:

{
  "name": "web-app-vulnerability-analysis--project",
  "version": "1.0.0",
  "scripts": {
    "name": "web-app-vulnerability-analysis--project",
    "version": "1.0.0",
    "version": "1.0.0",
  }
}
PS C:\Users\smart lenovo\Documents\web-app-vulnerability-analysis--project> npm install express

added 69 packages, and audited 70 packages in 9s

  run `npm fund` for details

found 0 vulnerabilities
PS C:\Users\smart lenovo\Documents\web-app-vulnerability-analysis--project> npm install express-session

added 6 packages, and audited 76 packages in 3s

14 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
PS C:\Users\smart lenovo\Documents\web-app-vulnerability-analysis--project> npm install bcrypt
npm warn deprecated inflight@1.0.6: This module is not supported, and leaks memory. Do not use it. Check out lru-cache if you want a good tested way to coalesce async requests by a key value, which is much more comprehensive and powerful.
npm warn deprecated rimraf@3.0.2: Rimraf versions prior to v4 are no longer supported
npm warn deprecated glob@7.2.3: Glob versions prior to v9 are no longer supported
npm warn deprecated are-we-there-yet@2.0.0: This package is no longer supported.
npm warn deprecated gauge@3.0.2: This package is no longer supported.

added 59 packages, and audited 135 packages in 13s

  run `npm fund` for details

found 0 vulnerabilities
PS C:\Users\smart lenovo\Documents\web-app-vulnerability-analysis--project> npm i -D nodemon

added 24 packages, and audited 159 packages in 7s

21 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
PS C:\Users\smart lenovo\Documents\web-app-vulnerability-analysis--project> npm i --save-dev nodemon

up to date, audited 159 packages in 1s

21 packages are looking for funding
```

Conclusion

Ce projet de développement d'une application web simple a permis d'explorer plusieurs aspects essentiels du développement web et de la sécurité des applications. À travers la création de fichiers statiques et dynamiques, nous avons conçu des interfaces conviviales pour la gestion des utilisateurs et des tâches, tout en adoptant des techniques modernes telles que CSS avancé, JavaScript natif, l'API Fetch et la manipulation du DOM.

L'analyse approfondie du code a révélé des vulnérabilités potentielles et des scénarios d'erreurs non gérés, mettant en évidence l'importance de la sécurité et de la robustesse dans toute application web. Les solutions proposées visent à renforcer la sécurité et à améliorer la fiabilité, créant ainsi une base solide pour des applications plus complexes.

Ce projet n'est pas seulement un exercice technique, mais également une opportunité d'acquérir des compétences pratiques en analyse de sécurité, en conception de systèmes robustes et en expérience utilisateur. En résumé, il reflète un cycle complet de développement, d'analyse et d'amélioration continue, essentiel pour tout développeur ou analyste en cybersécurité.