**ELBARCHAOUI Soufiane : IAWM TA**

**Rapport Technique Application to-do list**

**Présentation Générale du Projet**

***Présentation générale du projet :***

L’objectif du projet est de réaliser une application de gestion de tâches (« to-do list ») permettant à un utilisateur de :

Consulter la liste des tâches existantes

Ajouter une nouvelle tâche

Modifier une tâche (titre, statut complété)

Supprimer une tâche

L’application est structurée en deux parties :

***Backend : Fournit une API REST pour la gestion des tâches.***

**Frontend :** Offre une interface utilisateur pour interagir avec la liste de tâches.

Fonctionnalités principales du backend (Express.js)

**server.js** : Fichier principal contenant toute la logique du serveur Express, les routes, le stockage des tâches, et la configuration des middlewares.

**node\_modules/** : Dépendances Node.js.

**package.json / package-lock.json :** Fichiers de gestion des dépendances.

*Routes RESTful* ***:***

**GET /api/tasks :** Récupère toutes les tâches.

**GET /api/tasks/:id** : Récupère une tâche par son identifiant.

**POST /api/tasks** : Crée une nouvelle tâche.

**PUT /api/tasks/:id** : Modifie une tâche existante.

**DELETE /api/tasks/:id** : Supprime une tâche.

*Contrôleurs & Modèles :*

Les tâches sont stockées dans une variable locale tasks (mock data), chaque tâche ayant un id, un title et un booléen completed.

***Middlewares :***

**cors** : Autorise les requêtes cross-origin (nécessaire pour le frontend React).

**body-parser** : Gère le parsing du corps des requêtes en JSON ou URL-encoded.

*Gestion des erreurs :*

Retourne un message d’erreur (404) si une tâche n’est pas trouvée lors des opérations de lecture, modification ou suppression.

Fonctionnalités principales du frontend (React.js)

***Composants :***

**App.js** : Composant racine, gère l’état principal des tâches et orchestre les sous-composants.

**TaskForm.js** : Formulaire d’ajout de tâche. Utilise un état local pour le champ de saisie et gère l’appel à l’API pour la création.

**TaskList.js** : Affiche la liste des tâches, permet de supprimer, modifier et marquer comme complétée une tâche. Gère l’état local pour l’édition et le chargement.

Services :

**api.js** : Service centralisé pour communiquer avec l’API backend via axios. Définit **les méthodes** : getAllTasks, createTask, updateTask, deleteTask. Gère aussi l’ajout d’un éventuel token d’authentification dans les headers.

***Navigation :***

L’application fonctionne sur une seule page (SPA), sans routing avancé.

Gestion des états :

Utilisation de useState et useEffect pour gérer les états locaux (liste des tâches, chargement, erreurs, édition).

Les modifications (ajout, suppression, édition) mettent à jour l’état local pour refléter immédiatement les changements côté UI.

Architecture générale du projet

*Backend (Express.js)*

Fichier principal : server.js

Structure simple : toutes les routes et le stockage des données sont définis dans ce fichier.

Utilisation de middlewares standards (cors, body-parser).

Pas de séparation en dossiers pour modèles, contrôleurs, routes (architecture monofichier, adaptée à un projet d’apprentissage ou une démo).

*Frontend (React.js)*

**Dossier principal** : src/

**components/ :** Contient les composants React principaux (TaskForm.js, TaskList.js).

**services/** : Contient le service d’appel API (api.js).

**App.js :** Point d’entrée de l’UI, structure la page.

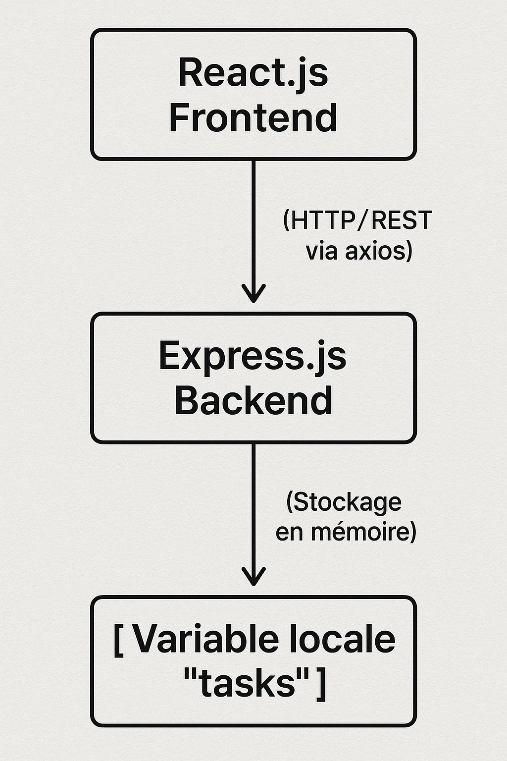
**index.js :** Point d’entrée de l’application React.

Utilisation de hooks React pour la gestion des états et des effets.

Communication avec le backend via le service api.js.

Communication

Le frontend interagit avec le backend via des requêtes HTTP sur l’API REST définie par Express.js.

Les URLs et ports sont configurés pour fonctionner en local (localhost:5000 pour l’API)

*Communication client-serveur*

**Mécanisme**

Le frontend utilise axios (dans src/services/api.js) pour envoyer des requêtes HTTP à l’API Express.

L’URL de base est http://localhost:5000/api.

Exemples de requêtes API

GET /api/tasks : Récupère la liste de toutes les tâches.

POST /api/tasks : Ajoute une nouvelle tâche (corps : { title: "..." }).

PUT /api/tasks/:id : Modifie une tâche (corps : { title: "...", completed: true/false }).

DELETE /api/tasks/:id : Supprime une tâche

*// src/services/api.js*

const response = await apiClient.get('/tasks'); // GET

const response = await apiClient.post('/tasks', taskData); // POST

const response = await apiClient.put(`/tasks/${id}`, taskData); // PUT

const response = await apiClient.delete(`/tasks/${id}`); // DELETE

***Fonctionnalités principales de l’application***

Ajouter une nouvelle tâche

Saisie du titre dans TaskForm.

Appel à taskService.createTask() (POST).

Mise à jour de la liste côté frontend.

Afficher la liste des tâches

TaskList utilise taskService.getAllTasks() (GET).

Affichage dynamique de chaque tâche.

Modifier une tâche

TaskList permet l’édition inline.

Appel à taskService.updateTask() (PUT).

Mise à jour de l’état local.

Supprimer une tâche

Bouton de suppression dans TaskList.

Appel à taskService.deleteTask() (DELETE).

Suppression dans l’état local.

Marquer une tâche comme complétée

Checkbox ou bouton dans TaskList.

Appel à taskService.updateTask() (PUT) avec completed: true/false.

Mise à jour de l’état local.



*Frontend (React) – Détails techniques*

Composants principaux

App.js : Point d’entrée, gère l’état global des tâches.

TaskForm.js : Formulaire d’ajout.

TaskList.js : Affichage, édition, suppression, marquage des tâches.

Gestion des interactions utilisateur

Utilisation de formulaires contrôlés (useState pour les champs).

Validation simple (ex : champ non vide).

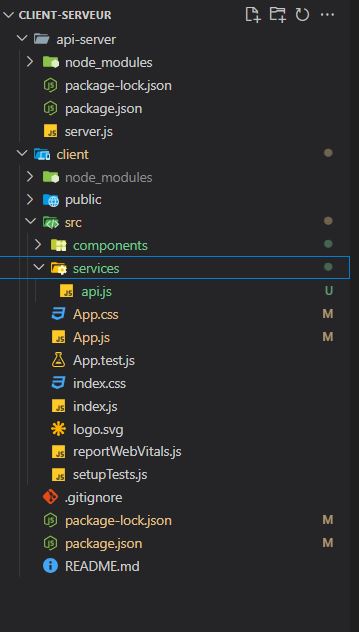
Retour d’erreur affiché si nécessaire.

Gestion de l’état

Utilisation de useState pour gérer : liste des tâches, champs de formulaire, état de chargement, erreurs, édition.

Utilisation de useEffect pour charger les tâches au montage du composant.

Pas de gestionnaire d’état global externe (Redux, Context API non utilisés).

*Résumé visuel*

# *Déploiement du projet sur GitHub*

Dans le cadre du développement de mon projet, j’ai réalisé les étapes suivantes pour le mettre en ligne sur GitHub :

1. **Création du dépôt** : J’ai créé un nouveau dépôt nommé Client-Serveur sur GitHub.
2. **Initialisation locale** : J’ai initialisé le projet en local avec git init.
3. **Liaison au dépôt distant** : J’ai lié le dépôt local à GitHub avec la commande git remote add origin.
4. **Commit initial** : J’ai ajouté tous les fichiers (git add .) et effectué un premier commit avec un message d’initialisation.
5. **Poussée vers GitHub** : J’ai défini la branche principale (git branch -M main) et poussé le projet sur GitHub avec git push.

🔗 Lien du projet : <https://github.com/soufianeelbar/Client-Serveur>

*Conclusion*

Ce projet a permis de créer une application de gestion de tâches (to-do list) avec un backend en **Express.js** et un frontend en **React.js**. Les fonctionnalités principales incluent l'ajout, la modification, la suppression, et le marquage des tâches comme complétées. Le frontend interagit avec le backend via des requêtes API RESTful. Le système utilise **axios** pour communiquer entre les deux parties. Bien que l’architecture soit simple et adaptée à un projet d’apprentissage, des améliorations pourraient être apportées pour une mise en production, notamment l’ajout d’une base de données et une meilleure gestion des erreurs. Le projet met en évidence l’intégration fluide de ces deux technologies.