

01/05/2024

TP 1 Architecture applicative

Création d'une application de liste de tâches



Franklin ASSOGBA

Table des matières

Présentation du projet	2
objectif.....	2
Utiliser le projet	2
L'architecture du code	2
Les fichiers	2
Explication du code	2
Le modèle	2
Le contrôleur	3
Le vue.....	4
Illustration.....	7

Présentation du projet

objectif

L'objectif de ce projet est de créer une application de liste de tâches. L'utilisateur peut saisir des tâches et leur associer une catégorie (divers, maison, travail). Il peut aussi filtrer les tâches par catégorie.

Utiliser le projet

Le projet peut être lancé en ouvrant le fichier **index.html** du dossier **view** dans un navigateur web. Les tâches sont sauvegardées dans le local storage du navigateur web sous forme JSON.

L'architecture du code

Le projet est organisé avec une architecture MVC avec les dossiers et fichiers suivants :

Les fichiers

Dossier **model**

- model.js

Dossier **view**

- index.html
- view.js
- low_task_view.js
- medium_task_view.js
- high_task_view.js

Dossier **controller**

- Controller.js

Explication du code

Le modèle

Le modèle est géré par la classe **TaskModel** en utilisant le design pattern singleton

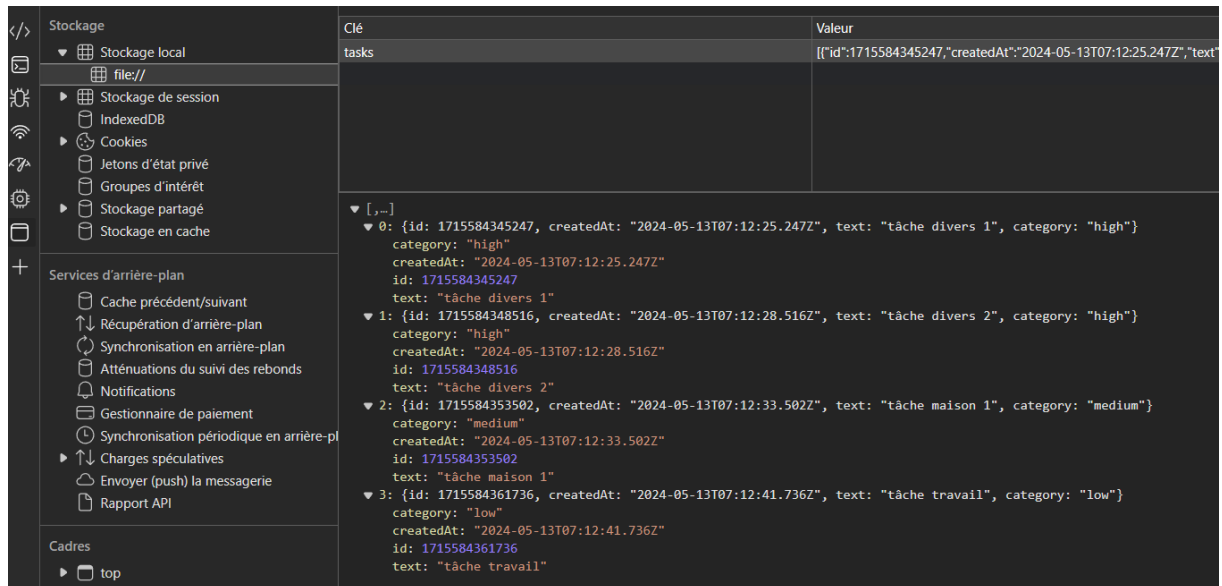
```
class TaskModel {  
  constructor() {  
    if (TaskModel.instance) {  
      return TaskModel.instance;  
    }  
    this.tasks = JSON.parse(localStorage.getItem("tasks")) || [];  
    TaskModel.instance = this;  
  }  
}
```

TaskModel possède des méthodes pour ajouter, supprimer ou actualiser des tâches. Il a aussi une méthode qui permet de récupérer la totalité des tâches qui ont été créées :

- addTask
- deleteTask
- updateTask
- getTasks

Les tâches sont enregistrées dans le local storage du navigateur web.

Clé	Valeur
tasks	[{"id":1715584345247,"createdAt":"2024-05-13T07:12:25.247Z","text":



Lorsqu'un changement d'état d'une tâche est sauvegardé le modèle diffuse un évènement.

```
addTask(taskText, taskCategory) {  
  const task = {  
    id: Date.now(),  
    createdAt: new Date().toISOString(),  
    text: taskText,  
    category: taskCategory,  
  };  
  this.tasks.push(task);  
  localStorage.setItem("tasks", JSON.stringify(this.tasks));  
  const taskAddedEvent = new CustomEvent("taskAdded", { detail: task });  
  document.dispatchEvent(taskAddedEvent);  
}
```

Le contrôleur

Cet évènement sera reçu par le contrôleur qui va actualiser la vue.

```

class TaskController {
  constructor(model, view) {
    this.model = model;
    this.view = view;

    document.addEventListener("addTask", this.handleAddTask.bind(this));
    document.addEventListener("taskAdded", this.handleTaskAdded.bind(this));
    document.addEventListener("deleteTask", this.handleDeleteTask.bind(this));
    document.addEventListener("updateTask", this.handleUpdateTask.bind(this));

    this.view.displayTasks(this.model.getTasks());
  }

  handleAddTask(event) {
    const { taskText, taskCategory } = event.detail;
    this.model.addTask(taskText, taskCategory);
  }

  handleTaskAdded(event) {
    const task = event.detail;
    this.view.displayTasks(this.model.getTasks());
  }
}

```

Le contrôleur permet aussi de gérer le filtrage de tâche par catégorie.

```

handleFilter(event) {
  const category_filtered = event.detail;
  this.model.updateFilter(category_filtered);
  this.view.displayTasks(this.model.getTasks());
}

```

En fonction de la valeur choisi pour le filtre la liste de tâche se met à jour

TP Archi Applicatives

Filtrage des tâches par Catégorie:

Divers

▼

Tache à ajouter

Catégorie:

Divers

▼

Ajouter

ID: 1715584345247

Le 13/5/2024 à 9h12 et 25 seconds

tâche divers 1

Sauvegarder

Supprimer

divers ▼

ID: 1715584348516

Le 13/5/2024 à 9h12 et 28 seconds

tâche divers 2

Sauvegarder

Supprimer

divers ▼

Le vue

L'interface avec laquelle l'utilisateur interagit est affichée par le fichier **index.html** qui inclus l'ensemble des classes utilisées par l'application.

```
<script src="../../model/model.js"></script>
<script src="../../view/view.js"></script>
<script src="../../view/low_task_view.js"></script>
<script src="../../view/medium_task_view.js"></script>
<script src="../../view/high_task_view.js"></script>
<script src="../../controller/controller.js"></script>
```

index.html comprend le formulaire de saisie des tâches mais leur affichage est géré par la super classe **TaskView** et sa méthode **displayTasks** qui fait utiliser les sous classes **LowTaskView**, **MediumTaskView**, **HighTaskView** qui possèdent chacune leur propre méthode pour afficher les tâches d'une catégorie spécifique (travail, maison, divers)

```
class TaskView {
  constructor() {
    this.taskList = document.getElementById("taskList");
    this.taskInput = document.getElementById("taskInput");
    this.taskForm = document.getElementById("taskForm");
    this.taskCategory = document.getElementById("prioritySelect");
    this.taskForm.addEventListener(
      "submit",
      this.handleTaskFormSubmit.bind(this)
    );
  }
}
```

```
displayTasks(tasks) {
  const lowTaskView = new LowTaskView ();
  const mediumTaskView = new MediumTaskView ();
  const highTaskView = new HighTaskView ();
  highTaskView.displayTasks(tasks);
  mediumTaskView.displayTasks(tasks);
  lowTaskView.displayTasks(tasks);
}
```

TaskView empêche également une tâche d'être ajoutée si elle ne contient aucun texte sinon elle émet un évènement qui sera traité par le contrôleur.

```
handleTaskFormSubmit(event) {  
  event.preventDefault();  
  const taskText = this.taskInput.value.trim();  
  const taskCategory = this.taskCategory.value.trim();  
  if (taskText !== "" && taskCategory !== "" ) {  
    const addTaskEvent = new CustomEvent("addTask", {detail: { taskText, taskCategory } });  
    document.dispatchEvent(addTaskEvent);  
    this.taskInput.value = "";  
  }  
}
```

Chacune de ses sous classes possède des « écouteurs » pour vérifier si l'état de la tâche à été modifié et changer la couleur du bouton enregistrer pour rappeler à l'utilisateur de sauvegarder les modifications de la tâche.

```
input.addEventListener("input", () => {  
  const newText = input.value;  
  const saveButton = document.getElementById(`saveButton_${task.id}`);  
  if (newText !== task.text) {  
    saveButton.style.backgroundColor = "orange";  
  } else {  
    saveButton.style.backgroundColor = "green";  
  }  
});
```

The screenshot shows a user interface for managing tasks. It features two task entries, each with a light green background. The first entry has the ID 1715584345247, a timestamp of 'Le 13/5/2024 à 9h12 et 25 seconds', and a text input containing 'tâche divers 1' with the word 'modif' being typed. Below the input are an orange 'Sauvegarder' button, a red 'Supprimer' button, and a dropdown menu showing 'divers'. The second entry has the ID 1715584348516, a timestamp of 'Le 13/5/2024 à 9h12 et 28 seconds', and a text input containing 'tâche divers 2'. Below its input are a green 'Sauvegarder' button, a red 'Supprimer' button, and a dropdown menu showing 'divers'.

Illustration

Notre application de liste de tâches fonctionne avec une architecture MVC qui permet le filtrage, la création, suppression et mise à jour de tâche.

TP Archi Applicatives

Filtrage des taches par Catégorie:

Toutes Catégories ▾

Tache à ajouter

Saisir une tâche

Catégorie:

Divers ▾

Ajouter

ID: 1715584345247

Le 13/5/2024 à 9h12 et 25 seconds

tâche divers 1 [modifier](#)

Sauvegarder

Supprimer

divers ▾

ID: 1715584348516

Le 13/5/2024 à 9h12 et 28 seconds

tâche divers 2

Sauvegarder

Supprimer

divers ▾

ID: 1715584353502

Le 13/5/2024 à 9h12 et 33 seconds

tâche maison 1

Sauvegarder

Supprimer

maison ▾

ID: 1715584361736

Le 13/5/2024 à 9h12 et 41 seconds

tâche travail 1

Sauvegarder

Supprimer

travail ▾