Rapport technique

AWA-Histomap par Victorien Montavon et Benoit Pierrehumbert

Introduction

Ce document aura pour but d'expliquer le flux d'informations de l'application, les choix techniques et les difficultés rencontrées lors de la réalisation de ce projet.

Flux d'informations

L'application est structurée en 5 composants principaux et le backend :

- **App.vue** : Le composant principal qui contient la structure de l'application et les composants enfants.
- **AddCard.vue** : Le composant qui gère la timeline et les histoires.
- **Card.vue** : Le composant qui affiche les détails d'une histoire.
- **AddModal.vue** : Le composant qui gère la fenêtre modale pour ajouter une histoire.
- **TimelineView.vue** : Le composant qui gère la représentation graphique de la timeline.
- **Backend** : API Express connectée à un cluster MongoDB et qui gère les données.

App.vue

Page principale et unique contenante de l'application. Le but ici est de gérer les données de l'application et de les transmettre aux composants enfants.

AddCard.vue

Composant utilisé pour ajouter de nouvelles cartes à la timeline. Il inclut les champs nécessaires à la création d'une histoire (titre, description, date, image) et communique avec le backend pour sauvegarder ces informations.

Card.vue

1. **Structure et props :**
    - Nom du composant : `Card`.
    - Le composant reçoit trois props obligatoires : `title`, `description` et `date`.
2. **Données locales (`data`)**
    - `isFlipped` (booléen) : indique si la carte est retournée ou non.
    - `isHidden` (booléen) : contrôle l'affichage du contenu sur la face avant lors de l'animation de bascule.
    - `cardHeight` (nombre) : stocke la hauteur maximale à appliquer à la carte pour harmoniser l'affichage entre le recto et le verso.

3. **Cycle de vie (`mounted`)**

    - Appelle la méthode `updateHeight()` pour calculer la hauteur la plus grande entre la face avant et la face arrière, afin d'adapter dynamiquement la taille de la carte.

4. **Propriétés calculées (`computed`)**

    - `cardStyle` : retourne un objet style contenant la hauteur de la carte (en pixels) et une hauteur minimale fixe.

5. **Méthodes**

    - `flipCard()` :
        - Inverse la valeur de `isFlipped` pour activer la rotation en 3D (grâce à la classe `rotate-y-180`).
        - Utilise un `setTimeout()` pour gérer l'animation et inverser `isHidden` (masque le recto pendant que la carte est retournée).
    - `updateHeight()` :
        - Récupère la référence DOM des faces avant (`frontSide`) et arrière (`backSide`) et met à jour `cardHeight` selon la hauteur scrollable la plus élevée.

6. **Template**

    - Un conteneur avec une classe `perspective` pour l'effet 3D.
    - Une div interne appliquant la rotation conditionnelle (`rotate-y-180`) selon la valeur de `isFlipped`.
    - Face avant : affiche l'image, la date, le titre, et un bouton pour retourner la carte.
    - Face arrière : superposition d'un background flouté, texte descriptif, et un bouton permettant de revenir au recto.
    - Les classes CSS (dont `backface-hidden` et `perspective`) servent à gérer la transformation 3D et à cacher l'élément « derrière ».

En résumé, ce composant gère un effet de carte retournée en 3D, où la hauteur s'adapte en fonction des éléments visibles, et où la transition entre la face avant et la face arrière est animée de façon fluide.

AddModal.vue

1. **Structure et props :**

    - Nom du composant : `AddModal`.
    - Props :
        - `isOpen` (booléen) : indique si la modale est ouverte ou fermée.

2. **Template :**

    - Affiche une modale (une fenêtre superposée) uniquement si `isOpen` est vrai.
    - Contient un formulaire avec trois champs obligatoires :
        - `title` (type texte)

        - `date` (type date)
        - `description` (zone de texte)
        - Deux boutons :
            - **Cancel** : ferme la modale.
            - **Save** : soumet le formulaire.

3. **Data et validation :**

    - `formData` : objet qui stocke les valeurs saisies par l'utilisateur (`title`, `date`, `description`).
    - **Validation** (`validateForm()`) :
        - Vérifie que le titre a au moins 5 caractères.
        - Vérifie que la description a au moins 100 caractères.

4. **Méthodes principales :**

    - `closeModal()` : émet l'événement close pour fermer la modale.
    - `onSubmit()` :
        - Valide le formulaire avec `validateForm()`.
        - Envoie les données à l'API via `axios.post('http://localhost:3000/stories', this.formData)`.
        - Récupère l'identifiant inséré et l'associe à l'objet `newStory`.
        - Émet l'événement `createStory` avec la nouvelle story créée.
        - Réinitialise le formulaire et ferme la modale.

5. **Intégration et communication :**

    - Le composant communique avec son parent grâce à des événements :
        - `close` : pour signaler la fermeture de la modale.
        - `createStory` : pour transmettre la nouvelle story créée au parent.

Ce composant gère donc la création d'une « story » : il affiche un formulaire dans une modale, contrôle la validité des données et émet ensuite un événement vers le parent avec l'élément nouvellement créé.

TimelineView.vue

1. **Structure et style**

    - Divisé en deux sections : cartes du haut et cartes du bas.
    - Une ligne centrale connecte les cartes, représentant le fil du temps.
    - Utilise les composants enfants `CardTop` et `CardBottom` pour afficher les cartes.

2. **Dynamisme**

    - Les cartes s'ajoutent dynamiquement en haut ou en bas de la timeline via les méthodes `pushTop()` et `pushBottom()`.
    - La largeur de la ligne centrale s'adapte en fonction du nombre de cartes présentes.
    - La largeur de la ligne est calculée en fonction de la taille des cartes et de l'espace disponible à l'écran.

3. **Interaction utilisateur**

    - Une barre de progression visualise le défilement horizontal de la timeline.
    - La largeur de la barre de progression (`progressLineWidth`) est calculée dynamiquement en fonction du défilement de l'utilisateur.
    - Pour éviter des mises à jour trop fréquentes et donner un effet d'inertie, la largeur de la barre est ajustée progressivement à l'aide d'une animation qui lisse les transitions (utilisant `requestAnimationFrame` et une interpolation).

4. **Calcul et mise à jour**

    - La méthode `updateLineWidth()` calcule la largeur totale nécessaire en fonction du nombre de cartes affichées et ajuste dynamiquement la ligne centrale.
    - La méthode `updateProgessLineWidth()` ajoute un écouteur d'événements au conteneur de défilement pour recalculer la progression de manière fluide lorsque l'utilisateur fait défiler la timeline. Une cible de largeur (`targetWidth`) est utilisée pour éviter des mises à jour directes, et une animation lisse la transition.

5. **Chargement des données**

    - La méthode `loadStories()` distribue les histoires initiales entre les cartes du haut et du bas de manière alternée.

6. **Gestion des événements**

    - Le composant réagit aux redimensionnements de la fenêtre en recalculant la largeur de la ligne pour s'adapter aux nouvelles dimensions de l'écran.

Backend

Le backend est une API RESTful construite avec Express.js. Elle est connectée à une base de données MongoDB pour stocker les données des histoires. Les routes principales sont :

- `GET /stories` : récupère toutes les histoires.
- `POST /stories` : crée une nouvelle histoire.
- `DELETE /stories/:title` : supprime une histoire par son titre.

Server

- Le fichier `server.js` est le point d'entrée de l'application backend. Il configure Express, définit les routes et lance le serveur sur le port 3000.
- On retrouve aussi l'ajout des modules nécessaires pour nos besoins, comme le usage des `cors` pour éviter tous problèmes.
- Un `logTransaction` pour voir les requêtes qui sont faites sur l'API.
- Et les générations des routes dynamiques dans le `forEach(route)`.

Routes

Les routes sont définies dans le fichier `routes/stories.js`. Retourner sous forme de tableau dans server.js.

Composition d'une route :

- method: 'get'
- route: '/stories'
- component: Story.getStories

dbManager

Le fichier `dbManager.js` contient les fonctions pour interagir avec la base de données MongoDB.

- `connect()` : établit la connexion avec la base de données.
- `getStories()` : récupère toutes les histoires.
- `createStory()` : crée une nouvelle histoire.
- `deleteStory()` : supprime une histoire par son titre. Les fonctions sont donc des requêtes préparées pour la base de données.

Services

Story

Le fichier `services/story.js` contient la logique d'interaction avec la base de données et gère les comportements de ces actions.

Choix techniques

Frontend

Vue.js

Framework imposé pour sa facilité d'utilisation et sa réactivité.

Tailwind CSS

Version : 3.4.15 Utilisé pour sa simplicité d'utilisation et sa grande bibliothèque de composants prédéfinis. De plus, nous l'avons déjà utilisé dans les cours GUI 1 et 2 qui nous ont été donnés par M. Mottier.

Backend

Node.js

Imposé.

Express

Version : 4.12.2 Framework minimaliste pour Node.js, choisi pour sa simplicité et sa vaste documentation.

Axios

Version : 1.7.9 Bibliothèque minimaliste pour les requêtes HTTP, compatible avec Vue.js et Express. Axios est également connu pour son support sur les anciennes versions de navigateurs.

Dotenv

Version : 16.4.7 Permet de charger les variables d'environnement à partir d'un fichier `.env`. Simple et largement utilisé.

MongoDB

Version : 6.11.0

- Permet une grande flexibilité sur les schémas des données, adaptée à un faible volume d'écriture et l'absence de relations complexes.
- Intégration facilitée avec Node.js grâce à Mongoose.