Quick start guide for FeedWire

Introduction

Welcome to our application! Designed to bring the best of both worlds, it combines the majority of social media functionalities with a comprehensive RSS feed management system. With features like post creation, comments, likes, and saving posts, it offers a rich social experience. Additionally, it provides powerful RSS capabilities, allowing users to explore the latest news from their favorite sources, access a curated "Today News" section, and save articles for later. To get a glimpse of the application in action, we have prepared a video demonstration. You can watch it SHOWCASE VIDEO. For access to the application, please visit the following link: DEMO.

Setting up the development environment

Requirements

Vue 3 + Vite

This application uses Vue 3 `<script setup>` SFCs, check out the script setup docs to learn more.

Recommended IDE Setup

- VS Code + Volar (and disable Vetur) + TypeScript Vue Plugin (Volar).

Install project dependencies

```
npm install
or
yarn install
```

The following dependencies will be installed:

```
{
  "name": "amicpole",
  "private": true,
  "version": "0.0.0",
  "type": "module",
  "scripts": {
    "serve": "vite",
    "build": "vite build",
    "preview": "vite preview"
  },
  "dependencies": {
    "@vuepic/vue-datepicker": "^4.4.0",
    "autoprefixer": "^10.4.14",
    "axios": "^1.3.6",
    "daisyui": "^2.51.5",
    "exceljs": "^4.3.0",
    "firebase": "^9.22.0",
    "pinia": "^2.0.33",
    "postcss": "^8.4.21",
    "sortablejs": "^1.15.0",
    "vue": "^3.2.47",
    "vue-advanced-chat": "^2.0.7",
    "vue-advanced-cropper": "^2.8.8",
    "vue-router": "^4.1.6"
  },
  "devDependencies": {
    "@vitejs/plugin-vue": "^4.1.0",
    "@voerro/vue-tagsinput": "^2.7.1",
    "sass": "^1.60.0",
    "tailwindcss": "^3.2.7",
    "vite": "^4.2.0"
  }
}
```

Application Launch

Don't forget to change your macros to launch your scripts as you wish, by default I left it as vue-cli, that is to say that to launch the project you must do:

```
npm run serve
```

Dependencies

- **@vuepic/vue-datepicker**: A date picker component for Vue.js.
  > **Note:** The `@vuepic/vue-datepicker` library was used to create the `InputDate.vue` component, which is notably used for the "Read Later" functionality and for setting the promotion duration of a content.
- **autoprefixer**: A plugin to parse CSS and add vendor prefixes automatically.
- **axios**: A promise-based HTTP client for making API requests.
  > **Note:** The `axios` library is used to fetch data from RSS feeds in the `fetchRssFeed` action of the `rssStore.js` store.
- **daisyui**: A utility-first CSS framework for rapidly building custom designs.
- **exceljs**: A library for reading, manipulating, and writing Excel files.
  > **Note:** The `exceljs` library is used to export the saved content in your application. Specifically, it is utilized in the `exportBookmarks` function in the `MyBookmarks.vue` component.
- **firebase**: A platform for building web and mobile applications backed by Google Cloud.
  > **Note:** Firebase is used for real-time data storage in the chat application. It is responsible for storing and retrieving chat messages in real time. The setup and configuration details can be found in the `.env` file.
- **pinia**: A state management system for Vue.js.
- **postcss**: A tool for transforming CSS with JavaScript plugins.
- **sortablejs**: A library for creating sortable lists and grids.
  > **Note:** SortableJS is utilized in the creation of the ToDoList, enabling the ability to drag and reorder tasks. This functionality allows users to easily rearrange tasks according to their preference.
- **vue**: A progressive JavaScript framework for building user interfaces.
- **vue-advanced-chat**: A feature-rich chat component for Vue.js.
- **vue-advanced-cropper**: A powerful image cropper component for Vue.js.
- **vue-router**: The official router for Vue.js.
- **@vitejs/plugin-vue**: A Vite plugin for Vue.js development.
- **@voerro/vue-tagsinput**: A tags input component for Vue.js.
  > **Note:** The `@voerro/vue-tagsinput` library is used in the creation of the `InputTags.vue` component. This component enables users to input and manage tags for organizing and categorizing various elements.
- **sass**: A CSS extension language.
- **tailwindcss**: A utility-first CSS framework.

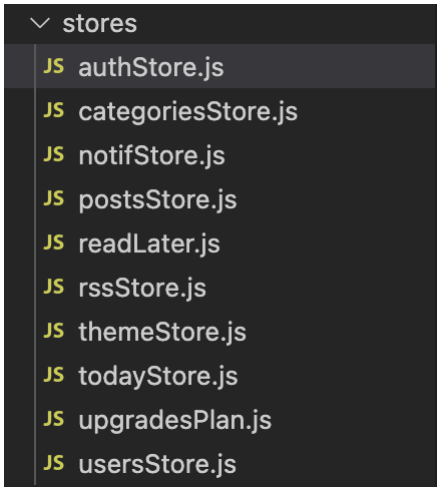
- **vite**: A fast development server and build tool for modern web applications.
- **hammer.js**: A library used to detect swipe gestures.

  > **Note:** hammer.js is only used when accessing the application on mobile devices to enable swipe gesture detection for opening the navigation menu.

Please refer to the respective documentation for detailed information on how to use each library.

Pinia Stores

In this application, Pinia stores are utilized to centralize the storage and manipulation of application data. By isolating the data management in stores, modifications and actions can be easily controlled and coordinated.



Store files: Each store represents a specific domain or module of the application

Store State: The state defines the data structure and initial values for the specific store

```
export const useAuthStore = defineStore({
  id: 'auth',
  state: () => ({
    user: {
      userId: 1,
      username: "Amicpole",
      email: "amicpole@gmail.com",
      certified: true,
      pdp: 'https://i.pravatar.cc/100?img=11',
      description: `Hi, I'm Amicole! I'm an experienced project manage
      followers: 24,
      following: 103,
      wallpaper: 'https://images.unsplash.com/photo-1508247967583-7d98
    },
    myPlan: 3, // The id of plan subscribe of user
    postsSavedIds: new Set([3, 4, 7]),
    whoIFollowing: new Set([2, 3, 5, 6]),
    whoFollowingMe : new Set([2 , 3, 7, 8]),
    myTags: [
      {
        id: 1,
        name: "CRYPTO",
        archive: false,
      },
      {
        id: 2,
        name: "CINEMA",
        archive: false,
      },
      {
        id: 3,
        name: "ARCHIVE TAG",
        archive: true,
      },
    ],
    associateTags: { 3: [1, 3], 4: [2] },
    currentPostSelected: null // Used for actions (Report & Share)
  }),
```

Store Actions: Actions define the operations or functions that can be performed on the store's data, can be used on each vue

```
actions: {
  logout() {
    this.user = null
    router.push('/')
  },
  signIn(username, password) {
    const usersStore = useUsersStore()
    const user = usersStore.users.find(user => user.username === username)
    if (user) {
      this.user = {
        ...user
      }
      router.push('/')
      return true
    }
    return false
  },
  signUp(data) {
    this.user = {
      ...data
    }
    const usersStore = useUsersStore()
    usersStore.users.push(this.user)
    router.push('/')
  },
  updateProfilePicture(picture) {
    this.user.pdp = picture
  },
  updateWallpaper(picture) {
    this.user.wallpaper = picture
  },
  follow(userId) {
    this.whoIFollowing.has(userId) ? this.whoIFollowing.delete(userId) : this.whoIFollowing.add(us
  }
},
```

Store Getters: Getters retrieve and compute derived values based on the store's state

```
getters: {
  getUsername(state) {
    return state.user ? state.user.username : 'Guest'
  },
  getProfileLink(state) {
    return state.user ? `/profile/${state.user.userId}` : '/login'
  },
  getPdpLink(state) {
    return state.user && state.user.pdp ? state.user.pdp : '/guest.png'
  }
}
```

Using Pinia stores provides the following benefits:

- Centralizes and isolates data storage and manipulation in a single location.
- Facilitates easy modification and coordination of actions, such as integrating database calls.
- Offers detailed insight into the required application data structure.
- Enables straightforward renaming of stores if needed.
- Encourages following the appropriate data types to minimize future code changes.

Feel free to modify the naming conventions, structure, and details of the store code based on your application's specific requirements.

Usage examples for using a store in vue

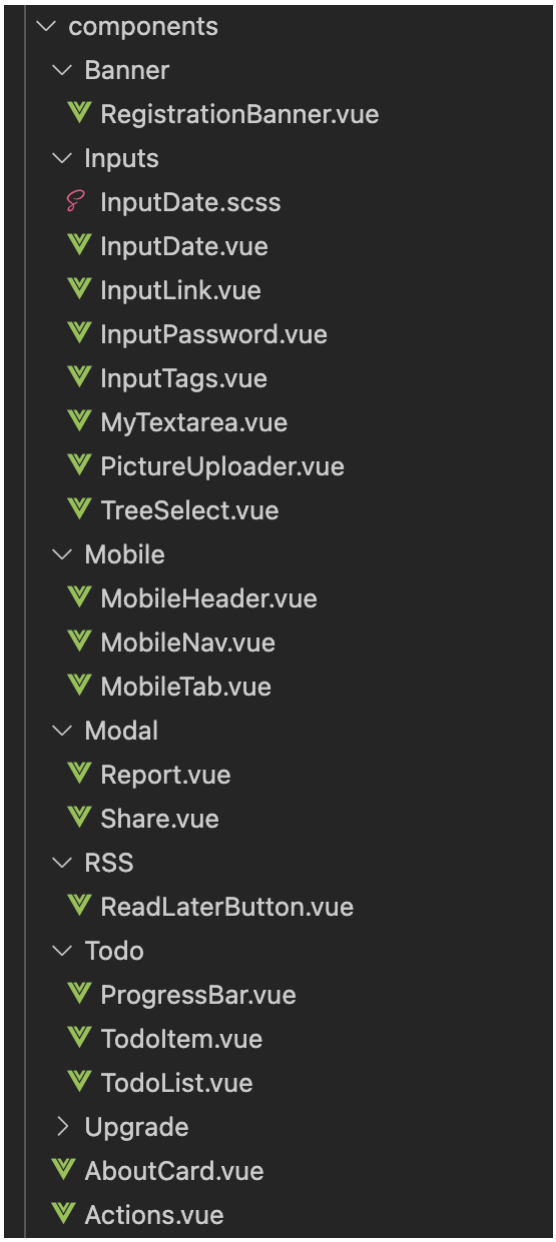
You only need to import your store is the used one in order to have access to all its content:

```
 import { useUsersStore } from "@/stores/usersStore"
const usersStore = useUsersStore()

// Call getters
usersStore.getUsername
usersStore.getProfileLink
usersStore.getPdpLink

// Call actions
usersStore.logout()
usersStore.follow(192 /* userId to follow */)

// Change/Migrate value directly with pinia
usersStore.user.username = "NewPseudo"
usersStore.user.email = "myemail@example.com"
```

Components

In this application, a highly atomic design approach has been followed to ensure modularity and reusability. The components have been organized in a directory structure that promotes easy extraction of specific features or components that are of interest.

Component directory structure: The components are organized based on their atomic nature, allowing easy access and extraction of individual components or functionalities.



The atomic design methodology provides the following advantages:

- **Modularity**: Components are structured in a way that each encapsulates a specific functionality or feature. This promotes code reusability and maintainability.
- **Reusability**: Components can be easily extracted and reused in other parts of the application or in future projects, thanks to their atomic design.
- **Isolation**: Each component is designed to be self-contained and independent. This facilitates easier testing and debugging.

By adhering to the atomic design principles, your application benefits from a flexible and scalable architecture. Developers can selectively retrieve and integrate only the desired components or functionalities, ensuring efficient development and customization.

Feel free to explore the `components` directory and leverage the modular nature of the components to enhance your application as needed.