VueJS

VueJS-techniques for implementing

SPA-applications

Data-management

Using data from the server (REST and AJAX)

State management

Pinia (Vuex)

Designing rest

- Implementing RESTful services is rather easy
  - However some aspects require some thought
- Design consistent url-patterns
  - GET /cars;from=100;to=200&make=volvo;order=model - Return list
  - GET /cars/:id - Return single item
  - POST /cars - Create item
  - PUT /cars/:id - Update item
  - DELETE /cars/:id - Delete item
- What amount of data is returned
  - Basic information first, detailed information with separate request
  - How is binary data accessed
  - Do the listings require paging
  - How should the contained entities be handled
- Should put and post return data

Using the RESTful interface

- Connecting to the RESTful services is not complicated either
- We can use any method we are familiar with
  - Traditional XMLHttpRequest
  - Fetch
  - JQuery
  - Axios
- Basically it just comes down to
  - Using the correct http-method and headers
  - And parsing the data received
- One thing to consider Architecture-wise
  - Are we going to hold large amount of data in memory
  - Or make separate request for each use case

Exercise

- Create file BookServiceHttp with a new version of bookService. It should have similar interface to the original bookService.
  - In the implementation use Http-object from http.js
  - Any problems having to make (some of) the methods asynchronous

- Http.get('/api/books').then(books => process array)
- Http.get('/api/books/2').then(book => got book with id 2)
- Http.post('/api/books',book).then(book => book was created)
- Http.put('/api/books/2',book).then(book => book was saved)
- Http.delete('/api/books/2').then(() => book was deleted)

- Data manipulation becomes easier if you pass all the books that come from the server through the "verify"-method

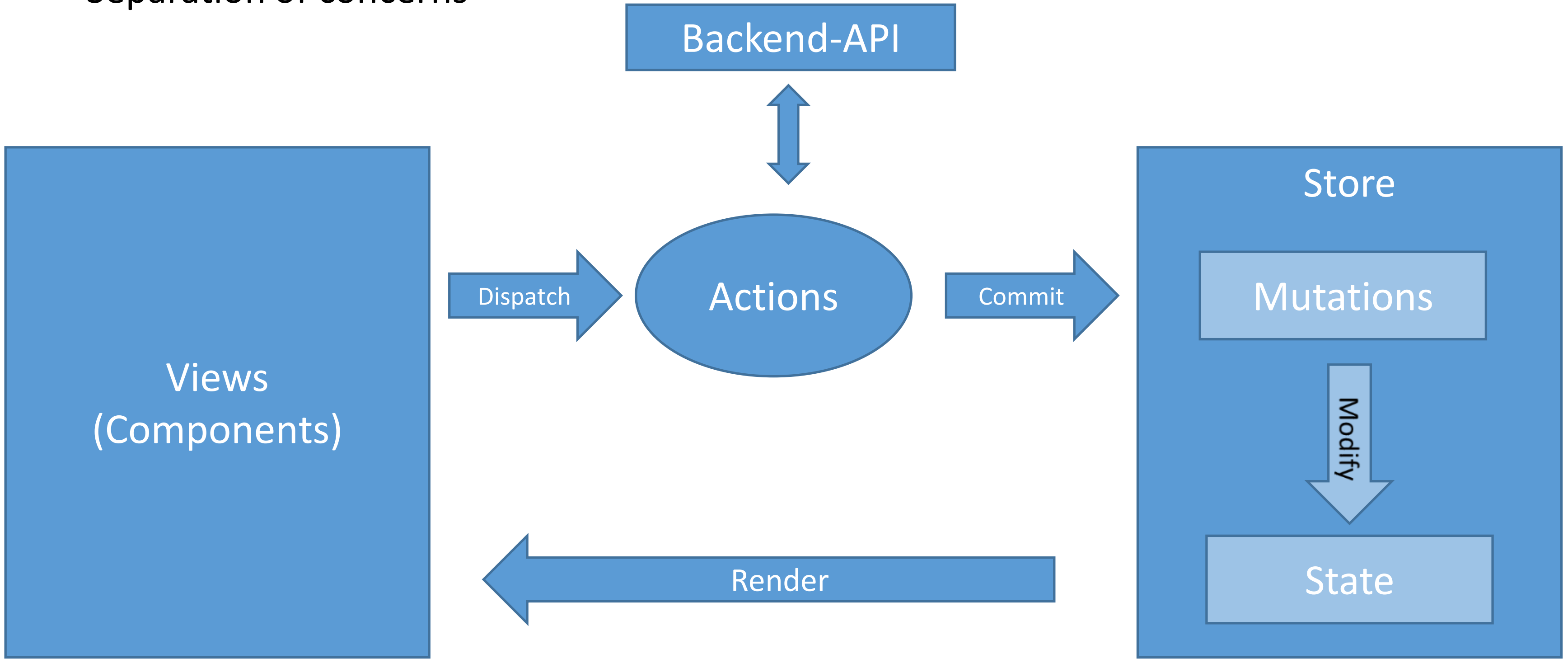
```
export const bookService=reactive({
    books:[],

    verify(book){
        book.published=new Date(book.published);
        let existing=this.books.find(b => b.id==book.id);
        if (!existing) this.books.push(book);
        else Object.assign(existing,book);
    },

    getAll(){
        HTTP.get("/simple/books").then(books => {
            books.forEach(book => this.verify(book));
        })
    },
. . .
}
```

Vuex

- As the application grows bigger managing the application state becomes complex
  - Several components need to access the same data
- Vuex is an extension that offers a “store” into which we can place the items belonging to the application state
  - Components become simpler
  - Separation of concerns



Defining the store

- Store should be designed
  - What data belongs to the application state
  - What data should remain in the component
- Ignore the authorFilter for now

```
// app.js
import Vuex from 'vuex';
import {bookstore} from './bookstore';


Vue.use(Vuex);
const store=new Vuex.Store(bookstore);

// add store to data
```

```
import {BookServiceHttp} from './bookservice';

export const bookstore={
    state:{
        books:BookServiceHttp.getAll(),
        sortOrder:'title',
        titleFilter:'',
        /// authorFilter:'',
        currentBook:{}
    },
    mutations:{
        books: (state,ba) => state.books=ba,
        book: (state,b) => state.currentBook=b,
        sort: (state,sort) => state.sortOrder=sort,
        titleFilter:(state,filter) => state.titleFilter=filter,
        //authorFilter:(state,filter) => state.authorFilter=filter,
    },
    actions:{
        getBooks: ctx => BookServiceHttp.items(books=>ctx.commit('books',books)),
        setSort: (ctx,sort) => ctx.commit('sort',sort),
        setTitleFilter: (ctx,filter) => ctx.commit('titleFilter',filter),
        // setAuthorFilter: (ctx,filter) => ctx.commit('titleFilter',filter),
        selectBook: (ctx,id) => BookServiceHttp.get(id,book => ctx.commit('book',book))
    }
};
```

In real life these would most likely belong to the component state

Pinia

- Pinia is a state management library is well suited for Vue-development
  - Especially when using Composition API
- Currently Pinia is preferred over Vuex

- npm i pinia
- And create use pinia for the application
  - Main.js

```
const app=createApp(App);
const pinia=createPinia();
app.use(pinia);
```

At main.js

Create Pinia store

- The return value from defineStore is a function that refers to the store object
  - The store object is returned by the second parameter of defineStore

```
import { defineStore } from "pinia"
import { ref } from "vue";
import { HTTP } from "./ajaxutils";

export const useBookStore = defineStore('books', () => {
    const books=ref([]);

    function verify(book){
        book.published=new Date(book.published);
        let existing=books.value.find(b => b.id==book.id);
        if (!existing) books.value.push(book);
        else Object.assign(existing,book);
    }

    function getAll(){
        HTTP.get("/simple/books").then(books => {
            books.forEach(book => verify(book))
        })
    }

    function get(id){
        HTTP.get("/simple/"+id).then(book => verify(book));
    }

     // ETC, save, create and delete

    return { books,getAll,get }
})
```

Using the store

- Books can be loaded at App.vue

```
import {useBookStore} from '../utils/BookStorePinia';
const bookStore=useBookStore();
bookStore.getAll();
```

- BookList
  - Let’s call the variable through which we refer to the store “bookService” so that we don’t have to change code elsewhere

```
import {useBookStore} from '../utils/BookStorePinia';
const bookService=useBookStore();
const books=bookService.books;
```

- BookDetail
  - Similar approach could be used as in BookList

Walkthrough

- Server js implements a websocket server
  - It pushes random values at one second intervals to connected clients
- Follow the instructors lead to implement

- SocketStore with Pinia
- WebSocket connection to App.vue
  - Push values to store
- SocketComponent that displays data

Exercise

- Instead of bookStore let’s create calculatorStore
  - Holds an array of calculations (objects with fig1 and fig2)
  - Reference to current calculation
- Create StoreCalculator
  - Similar to event calculator, has a button that does the calculation
  - Also when the button is pressed store the calculation to the calculations in the store and set the reference to the current calculation also
  - Display current calculation in the footer of the application
- Create CalculationsComponent that just displays the list of calculations
  - When item is clicked it is set to current
  - Add navigation to the CalculationsComponent to the nav-section of App-component

Security of SPA-application

Security

- You want to protect your data
  - Especially secure the RESTful interface
  - But also
    - Credentials in application logic
    - Secure the server (technical credentials)
- You want to protect the data communication
  - HTTPS
- You might even want to protect the application
  - Logic
  - Templates
  - Resources
- Security measures you choose must come from the requirements of your solution
  - Traceabilty
  - Undeniability
  - Usability

Protecting the RESTful services

- You have to select the authentication method depending on
  - The possibilities your sever environment gives you
  - Your application and its requirements
- In your code
  - You may pass the Authorization-header with your request
  - You may pass authorization information in a cookie generated in user login
  - You may pass authorization information in the body of each request
- NEVER store technical credentials for authentication into the JavaScript-code loaded to the browser

Protecting the application itself

- If you don’t want the unauthenticated users to access the
  - Templates
  - JavaScript
  - On other resources
- You have to
  - Design the distribution directory structure so that unauthenticated users can only access the “Login application”
    - Login application then gives access to the rest of the application
  - Figure out how to pass authorization token from login application to the actual application
- This requires some server specific configurations

cφde tutors

Thank you!

Any remaining questions?