# Frontend

Software Engineering - Lab

**Marco Robol - marco.robol@unitn.it**

## Contents

- HTML, `fetch()` API, DOM manipulation
- Vue.js, Tailwind, DaisyUI

> **EasyLib** https://github.com/unitn-software-engineering/EasyLib
> *Frontend* - https://github.com/unitn-software-engineering/EasyLibVue
> Demo APIs - https://easy-lib.onrender.com/api/v1
> Demo Basic Frontend - https://easy-lib.onrender.com
> Demo Vue Frontend - https://easy-lib.onrender.com/EasyLibApp or https://unitn-software-engineering.github.io/EasyLibApp/

## HTML

> The standard markup language for Web pages https://www.w3schools.com/html

```
<!DOCTYPE html>                                     <!-- HTML5 document -->
<html>                                              <!-- Root element -->
  <head></head>                                     <!-- Meta information about the page -->
  <body>                                            <!-- Visible content -->
    <h1>EasyLib</h1>                                <!-- Heading -->
    <p>Here is the list of books.</p>               <!-- Paragraph -->
    <ul id="books" class="list">                    <!-- List -->
      <li>                                          <!-- List element -->
        <a href="./api/v1/books/1">Book 1</a> <!-- Hyperlink, the `href` attribute -->
      </li>
    </ul>
  </body>
</html>
```

**TODO**: Now add a *subtitle*, make 'books' **bold**, and modify the list into a `<table>`.

## Styling with CSS

> The language we use to style an HTML document https://www.w3schools.com/css

```
<head>                                                <!-- Internal CSS -->
  <style>
    body {                                            /* Element Selector */
      background-color: lightblue;
    }
    .borded {                                         /* Class Selector */
      border: 1px solid red;
      text-align: center;
    }
    a:hover {                                         /* Pseudo-classes Selector */
      padding-top: 50px;
    }
  </style>
  <link rel="stylesheet" href="mystyle.css">          <!-- External CSS -->
</head>
<body>
  <h1 style="color:red">EasyLib</h1>                  <!-- Inline CSS: The `style` attribute -->
  <div class="borded">                                <!-- The `class` attribute -->
    Box with 1px border
  </div>                                              <!-- The `div` container element -->
```

**TODO**: Make the `div` a square and center it in the page, align text in the center.

## Responsive Design 1/2 - *The Viewport and Media Queries*

> Responsive web design makes your web page look good on all devices.
> https://www.w3schools.com/css/css_rwd_intro.asp

HTML5 introduced a method to let web designers take control over the viewport.

```html
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

Media query is a CSS technique introduced in CSS3. It uses the @media rule to include a block of CSS properties only if a certain condition is true.

```css
@media only screen and (max-width: 600px) {
  body {
    background-color: lightblue;
    height: 100vh;
  }
}
```

## Responsive Design 2/2 - *CSS Flexbox*

> Before the Flexbox Layout module, there were four layout modes: (i) **Block**, for sections in a webpage; (ii) **Inline**, for text; (iii) **Table**, for two-dimensional table data; (iv) **Positioned**, for explicit position of an element. The **Flexible Box Layout Module**, makes it easier to design flexible responsive layout structure without using float or positioning. https://www.w3schools.com/css/css3_flexbox.asp

```
<div style="display: flex; flex-wrap: wrap;">
  <div style="padding: 5rem">Item 1</div>
  <div style="padding: 5rem">Item 2</div>
</div>
```

**TODO**: Make a 1-column layout when on small screen, otherwise make it 2-columns.

## Styling Frameworks

## `tailwindcss.com`

A utility-first CSS framework packed with classes like flex, `pt-4`, `text-center` and `rotate-90` that can be composed to build any design, directly in your markup.

```html
<!doctype html>
<html>
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <script src="https://cdn.tailwindcss.com"></script>
</head>
<body>
  <h1 class="text-3xl font-bold hover:text-red-500 underline p-2 mx-auto">
    Hello world!
  </h1>
</body>
</html>
```

**TODO**: Set `h1` background color to black, make it larger as text, and text to white.

## daisyUI A component library for Tailwind CSS

```html
<!doctype html>
<html>
  <head>
    <link href="https://cdn.jsdelivr.net/npm/daisyui@4.12.14/dist/full.min.css" rel="stylesheet" type="text/css" />
    <script src="https://cdn.tailwindcss.com"></script>
  </head>
  <body>
    <div class="navbar bg-base-100">
        <a class="btn btn-ghost text-xl">daisyUI Navbar</a>
    </div>
    <div class="collapse bg-base-200">
        <input type="checkbox" />
        <div class="collapse-title text-xl font-medium">Click me to show/hide content</div>
        <div class="collapse-content">
            <p>I'm a daisyUI Collapse element</p>
        </div>
    </div>
    <br/>
    <footer class="footer footer-center bg-base-300 text-base-content p-4">
        <aside>
            <p>Copyright © 2024 - daisyUI Footer</p>
        </aside>
    </footer>
  </body>
</html>
```

**TODO**: Add a DaisyUI dropdown menu to the navbar!

## Responsive Design with `tailwindcss`

> https://tailwindcss.com/docs/responsive-design

```html
<!doctype html>
<html>
<head>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link href="https://cdn.jsdelivr.net/npm/daisyui@4.12.14/dist/full.min.css"
      rel="stylesheet" type="text/css" />
    <script src="https://cdn.tailwindcss.com"></script>
</head>
<body>
    <div class="flex flex-row flex-wrap">
        <!-- Width of 16 by default, 32 on medium screens, and 48 on large screens -->
        <div class="w-full md:w-1/2 lg:w-1/3 bg-red-500 p-2">
            Book 1
        </div>
        <div class="w-full md:w-1/2 lg:w-1/3 bg-blue-500 p-2">
            Book 2
        </div>
        <div class="w-full md:w-1/2 lg:w-1/3 bg-green-500 p-2">
            Book 3
        </div>
        <div class="w-full md:w-1/2 lg:w-1/3 bg-yellow-500 p-2">
            Book 4
        </div>
      </div>
</body>
</html>
```

…Now that we have basic HTML structure and CSS styling, let's move on to the next step: **adding interactivity** to our web page.

## JavaScript in an HTML document

```
<!-- External script reference (in <head> or <body>) -->
<script src="myScript.js"></script>

<!-- The <script> tag (in <head> or <body>) -->
<script>
    console.log( "Hello World" );
</script>
```

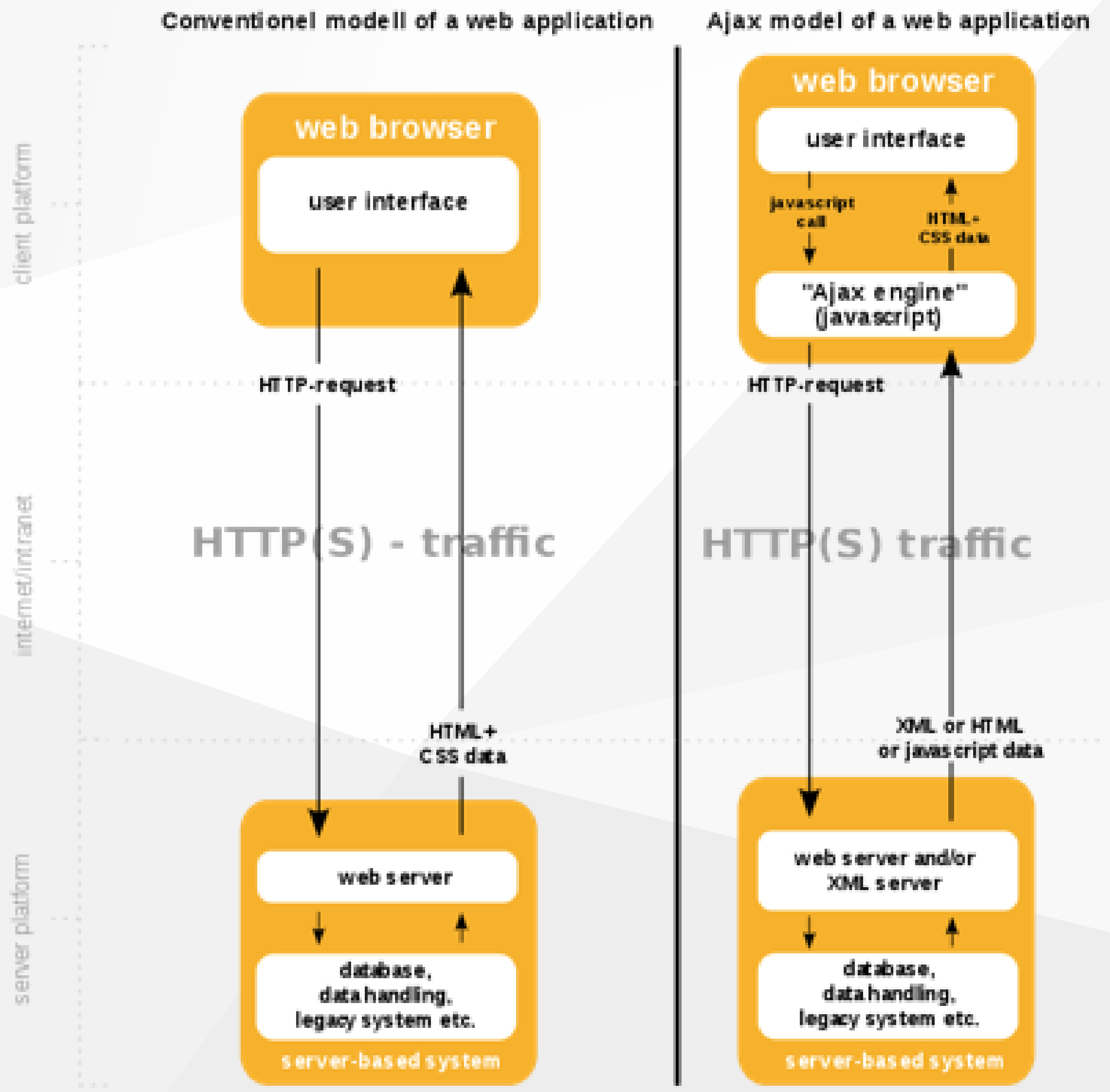
**TODO**: Try this, then modify the script to save "hello" to `document.hello`, finally check its value from within the browser console!

## AJAX and `XMLHttpRequest`

> Back in 1999, AJAX made possible to: (i) **Read data from a web server** - after the page has loaded; (ii) **Update a web page** without reloading the page; (iii) **Send data to a web server** - in the background.
> https://www.w3schools.com/js/js_ajax_intro.asp
> Today, with `fetch` API, there is no need for `XMLHttpRequest` anymore.

## AJAX

© Wikipedia



## JavaScript HTML DOM

> The HTML DOM is a standard object model and programming interface for HTML. It defines: The HTML elements as objects; The properties of all HTML elements; The methods to access all HTML elements; The events for all HTML elements.
> In other words: The HTML DOM is a standard for how to get, change, add, or delete HTML elements. https://www.w3schools.com/js/js_htmldom.asp

```
document.getElementById("demo").innerHTML   // The easiest way to get the content of an element
document.getElementById("image").src        // The attribute of an HTML element
document.createElement("div")               // Create an HTML element
document.removeChild(element)               // Remove an HTML element
document.appendChild(element)               // Add an HTML element
```

**TODO**: Read the content of an HTML element, then modify its innerHTML!

## JavaScript `fetch()` API

> https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API

```
// GET
fetch('../api/v1/books')
.then( resp => resp.json() )
.then( console.log )
.catch( error => console.error(error.message) );
// POST
const response = await fetch("../api/v1/books", {
    method: "POST",
    body: JSON.stringify({ title: "Learning fetch() API" }),
    headers: { "Content-Type": "application/json" }
});
if ( ! response.ok )
    console.log( response.status )
else
    console.log( await response.json() )
```

**TODO**: Try retrieving all the books from easy-lib.onrender.com, then insert a new one!

**TODO**: Now, try access the APIs from your project backend!

Because the html page loaded in the browser (e.g. file://index.html) resolves to a different url with respect to our APIs backend (e.g. http://localhost:3000), we won't be able to make asynchronous requests from within the browser, because of the CORS policy!

## Cross-Origin Resource Sharing policy

> Cross-Origin Resource Sharing (CORS) is an HTTP-header based mechanism that allows a server to indicate any origins (domain, scheme, or port) other than its own from which a browser should permit loading resources.

`npm install cors` https://expressjs.com/en/resources/middleware/cors.html

```
const cors = require('cors')
app.use(cors())
```

**TODO**: Add support to CORS in your backend!

## More datails on CORS in Express.js

> CORS also relies on a mechanism by which browsers make a "preflight" request, in order to check that the server hosting the cross-origin resource will permit the actual request. https://developer.mozilla.org/en-US/docs/Glossary/Preflight_request

```
app.use(function (req, res, next) { // Add headers before the routes are defined
    // Website you wish to allow to connect
    res.setHeader('Access-Control-Allow-Origin', 'http://localhost:3000');
    // Request methods you wish to allow
    res.setHeader('Access-Control-Allow-Methods', 'GET, POST, OPTIONS, PUT, PATCH, DELETE');
    // Request headers you wish to allow
    res.setHeader('Access-Control-Allow-Headers', 'X-Requested-With,content-type');
    // Set to true if you need the website to include cookies in the requests sent
    // to the API (e.g. in case you use sessions)
    res.setHeader('Access-Control-Allow-Credentials', true);
    // Pass to next layer of middleware
    next();
});
```

## EasyLib basic frontend

EasyLib basic WebApp is composed by an html file `\static\index.html` and some javascript code in `\static\script.js`, where data are fetched from the WebService and used to updated the page accordingly. **This is the minumum requirement for the project**! Source code at https://github.com/unitn-software-engineering/EasyLib.

**TODO**: Let's run the server or visit https://easy-lib.onrender.com and check out the network connections happening in the backgroung (using the browser console) while playing with the frontend!

## EasyLib basic frontend - HTML 1/1

`EasyLib\static\index.html`

```html
<h1>EasyLib</h1>

<form action="api/v1/students" method="post" name="loginform" id="loginform">
  <span>Logged User:</span> <span id="loggedUser"></span>
  <input name="email" value="mario.rossi@unitn.com" id="loginEmail"/>
  <input name="email" value="123" id="loginPassword"/>
  <button type="button" onclick="login()">LogIn</button>
</form>

<h2>Books:</h2> <ul id="books"></ul>

<h2>Insert new book:</h2>
<form action="api/v1/books" method="post" name="bookform" id="bookform">
  <input name="title" value="title" id="bookTitle"/>
  <button type="button" onclick="insertBook()">Insert new book</button>
</form>

<script src="script.js"></script>
```

## EasyLib basic frontend - Javascript 1/2

`EasyLib\static\script.js`

```javascript
var loggedUser = {} // This variable stores the logged in user

//This function is called when login button is pressed.
function login() { ... }
// This function refresh the list of books
function loadBooks() { ... }
loadBooks();
// This function is called by the Take button beside each book.
function takeBook(bookUrl) { ... }
// This function refresh the list of bookLendings.
function loadLendings() { ... }
// This function is called by clicking on the "insert book" button.
function insertBook() { ...}
```

## EasyLib basic frontend - Javascript 2/2

`EasyLib\static\script.js` `function login()`

```javascript
function login() {
  var email = document.getElementById("loginEmail").value;
  var password = document.getElementById("loginPassword").value;

  fetch('../api/v1/authentications', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify( { email: email, password: password } ),
  }).then((resp) => resp.json()) // Transform the data into json

  .then(function(data) { // Here you get the data to modify as you please
    loggedUser.token = data.token;
    loggedUser.email = data.email;
    loggedUser.id = data.id;
    loggedUser.self = data.self;
    document.getElementById("loggedUser").innerHTML = loggedUser.email;
    loadLendings();
    return;
  }).catch( error => console.error(error) );
};
```

## Declarative UI Components

### Front-end Frameworks

## Vue.js

> Vue (pronounced /vjuː/, like view) is a JavaScript framework for building user interfaces. It builds on top of standard **HTML**, **CSS** and **JavaScript**, and provides a **declarative** and **component-based** programming model that helps you efficiently develop user interfaces, be it simple or complex.
> https://vuejs.org/

## Using Vue `ES Module Build` from CDN (Without Build Tools)

> https://vuejs.org/guide/quick-start.html (other quick-start alternatives)

```html
<div id="app">
    <h1>{{ message }}</h1>
    <button @click="count++"> Count is: {{ count }} </button>
</div>

<script type="module">
    import { createApp, ref } from 'https://unpkg.com/vue@3/dist/vue.esm-browser.js'
    createApp({                                      // will later replace setup()
        setup() {                                    // with the <script setup> syntax
            const message = ref('Hello Vue!');
            const count = ref(0);
            return { message, count };
        }
    }).mount('#app')
</script>
```

**TODO**: Try this in your `html` file!

**Declarative Rendering**: Vue extends standard HTML with a template syntax that allows us to declaratively describe HTML output based on JavaScript state.

```
<div id="app">
    <button @click="count++"> Count is: {{ count }} </button>
</div>
```

**Reactivity**: Vue automatically tracks JavaScript state changes and efficiently updates the DOM when changes happen.

```
import { ref } from 'vue'
const count = ref(0);
```

## Vue 3 API Styles

> Vue components can be authored in two different API styles: Options API and Composition API - https://vuejs.org/guide/introduction.html#api-styles

With **Options API**, we define a component's logic using an object of options.

```
<script>
export default {
  data() { return { count: 0 } },
  methods: {} ...
```

With **Composition API**, we define a component's logic using imported API functions.

```
<script setup>
import { ref, onMounted } from 'vue'
const count = reactive( { value: 0 } )  // deep reactive state; It only works for object types.
const count = ref(0)                    // reactive "refs"
function increment() { count.value++ } ...
```

## The core feature of Vue is declarative rendering

Using a template syntax that extends HTML, we can describe how the HTML should look based on JavaScript state. When the state changes, the HTML updates automatically!

```
<!-- /src/App.vue -->
<script setup>
    import { ref } from "vue";
    const message = ref('Hello Vue!');
    const count = ref(0);
</script>
<template>
    <header class="pr-10">
        <h1 class="text-red-500 text-3xl font-bold underline">{{ message }}</h1>
    </header>
    <main>
        <button class="btn btn-outline btn-accent" @click="count++"> Count is: {{ count }} </button>
    </main>
</template>
<style scoped> </style>
```

## Attribute Bindings

In Vue, mustaches are only used for text interpolation. To bind an attribute to a dynamic value, we use the `v-bind:` directive (or short-hand `:`).

```
const titleClass = ref('text-green-500')
```

```
<h1 :class="titleClass">Make me red</h1>
```

## Two-way bindings: form input elements

Using `v-bind` and `v-on` together, we can create two-way bindings.

```
const text = ref('')
function onInput(e) { // a v-on handler receives the native DOM event as argument
    text.value = e.target.value
}
```

```
<input :value="text" @input="onInput">
<p>{{ text }}</p>
```

**TODO**: Try this, what is happening? Then try the easier syntax for two-way bindings:

```
const text = ref('')
```

```
<input v-model="text" placeholder="Type here">
<p>{{ text }}</p>
```

## Conditional Rendering (if)

We can use the `v-if` directive to conditionally render an element:

```
<h1 v-if="awesome">Vue is awesome!</h1>
<h1 v-else>Oh no 😢</h1>
```

**TODO**: Try this, then add a button to toggle `awesome`:

```
function toggle() {
  awesome.value = ! awesome.value
}
```

```
<button @click="toggle">Toggle</button>
```

## List Rendering (for loop)

We can use the `v-for` directive to render a list of items:

```
<ul>
    <li v-for="todo in todos" :key="todo.id">
        {{ todo.text }}
    </li>
</ul>
```

```
let id = 0;
const todos = ref([
    { id: id++, text: 'Learn HTML' },
    { id: id++, text: 'Learn JavaScript' },
    { id: id++, text: 'Learn Vue' }
])
```

**TODO**: Try this, then add a button to delete each element.

> https://vuejs.org/tutorial/#step-7

## Vue Components

So far, we've only been working with a single component. Real Vue applications are typically created with nested components. https://vuejs.org/tutorial/#step-11

```
<!-- App.vue -->
<script setup>
    import ChildComp from './ChildComp.vue'
</script>
<template>
    <ChildComp />
</template>
```

```
<!-- ChildComp.vue -->
<template>
  <h2>A Child Component!</h2>
</template>
```

**TODO** What are these .vue files?

## Using Vue Single File Components (SFCs) (With Build Tools)

> A build setup allows us to use Vue Single-File Components (SFCs). The official Vue build setup is based on Vite, a frontend build tool.
> https://vuejs.org/guide/quick-start.html#with-build-tools

- `npm init vue@latest` This command will install and execute create-vue, the official Vue project scaffolding tool.
- `npm install`
- `npm run dev`

```
<!-- src/components/App.vue -->
<script setup> ... </script>
<template> ... </template>
<style scoped> ... </style>
```

- Install VSCode Vue - Official extension

**TODO**: Run this to create a Vue project! Discuss about the structure and the build process.

## Install tailwindcss and DaisyUI with `npm`

> **tailwindcss** https://tailwindcss.com/docs/installation/using-postcss
>
> **DaisyUI** https://daisyui.com/docs/install/

- `npm install -D tailwindcss postcss autoprefixer daisyui`
- `npx tailwindcss init -p`
- Add the paths to all of your template files in your `/tailwind.config.js`.

```
import daisyui from 'daisyui';
export default {    content: ["./src/**/*.{html,js,vue}"],
                    theme: { extend: {} },
                    plugins: [ daisyui ]                    }
```

- Include tailwind in your `/src/assets/main.css` file

```
@tailwind base; @tailwind components; @tailwind utilities;
```

- `npm run dev`

## Props

A child component can accept input from the parent via props.

```
<!-- App.vue -->
<template>
    <ChildComp :msg="greeting" />
</template>
```

```
<!-- ChildComp.vue -->
<script setup>
    const props = defineProps({
        msg: String
    })
</script>
<template>
    <h2>{{ msg || 'No props passed yet' }}</h2>
</template>
```

## Emits

In addition to receiving props, a child component can also emit events to the parent.

```
<!-- App.vue -->
<!-- const childMsg = ref('No child msg yet') -->
<template>
    <ChildComp @response="(msg) => childMsg = msg" />
    <p>{{ childMsg }}</p>
</template>
```

```
<!-- ChildComp.vue -->
<script setup>
    const emit = defineEmits(['response'])
    emit('response', 'hello from child')
</script>
<template>
    <h2>Child component</h2>
</template>
```

## vuejs.org/guide/quick-start#next-steps

- Continue the Guide - The guide walks you through every aspect of the framework.
- Try the Tutorial - For those who prefer learning things hands-on.
- Check out the Examples - Explore examples of core features and common UI tasks.

## EasyLib Vue-based Frontend

Repository - https://github.com/unitn-software-engineering/EasyLibVue

Deploy - https://unitn-software-engineering.github.io/EasyLibApp/

```
<!-- `EasyLibVue\src\App.vue` -->
import Login from '@/components/Login.vue'
...
<nav>
  ...
  <RouterLink to="/books">Books</RouterLink>
  <RouterLink to="/booklendings">Booklendings</RouterLink>
</nav>
<Login />
...
```

## EasyLibVue\src\components\Login.vue

```
<script setup>
import { ref } from 'vue'
import { loggedUser, setLoggedUser, clearLoggedUser } from '../states/loggedUser.js'
const HOST = `http://localhost:8080`;
const email = ref('mario.rossi@unitn.com');
const password = ref('123')
function login() {
    fetch(HOST+'/api/v1'+'/authentications', {
        method: 'POST', headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify( { email: email.value, password: password.value } ),
    })
    .then((resp) => resp.json()) // Transform the data into json
    .then(function(data) { setLoggedUser(data) })
};
function logout() { clearLoggedUser() }
</script>
<template>
  <form>
    <span v-if="loggedUser.token">
      Welcome <a :href="HOST+'/'+loggedUser.self">{{loggedUser.email}}</a>
      <button type="button" @click="logout">LogOut</button>
    </span>
    <span v-if="!loggedUser.token">
      <input name="email" v-model="email" />
      <input name="password" v-model="password" />
      <button type="button" @click="login">LogIn</button>
    </span>
  </form>
</template>
```

## `EasyLibVue\src\states\loggedUser.js`

> https://vuejs.org/guide/scaling-up/state-management.html#simple-state-management-with-reactivity-api

```
import { reactive } from 'vue'
const loggedUser = reactive({
    token: undefined, email: undefined,
    id: undefined, self: undefined
})

function setLoggedUser (data) {
    loggedUser.token = data.token; loggedUser.email = data.email;
    loggedUser.id = data.id; loggedUser.self = data.self;
}
function clearLoggedUser () {
    loggedUser.token = undefined; loggedUser.email = undefined;
    loggedUser.id = undefined; loggedUser.self = undefined;
}

export { loggedUser, setLoggedUser, clearLoggedUser }
```

## EasyLibVue\src\router\index.js

```javascript
import HomeView from '../views/HomeView.vue'
routes: [
  {
    path: '/',
    name: 'home',
    component: HomeView
  },
  {
    path: '/books',
    name: 'books',
    // route level code-splitting
    // this generates a separate chunk (About.[hash].js) for this route
    // which is lazy-loaded when the route is visited.
    component: () => import('../views/BooksView.vue')
  },
  {
    path: '/booklendings',
    name: 'booklendings',
    component: () => import('../views/BooklendingsView.vue')
  }
]
```

## `EasyLibVue\src\views\BooklendingsView.vue`

```
<script setup>
import BooklendingsTable from '@/components/BooklendingsTable.vue'
</script>

<template>
  <div>
    <h1>Booklendings:</h1>
      <BooklendingsTable />
  </div>
</template>

<style>
</style>
```

## `EasyLibVue\src\components\BooklendingsTable.vue`

```
<script setup>
import { loggedUser } from '../states/loggedUser.js'
import { books, fetchBooks, createBook, deleteBook } from '../states/books.js'

const HOST = `http://localhost:8080`; const API_URL = HOST+`/api/v1`;
const booklendings = ref([])
onMounted( () => { fetchBooks(); fetchData(); })
watch(loggedUser, (_loggedUser, _prevLoggedUser) => { fetchBooks(); fetchData(); })

async function fetchData() {
  if (!loggedUser.token) { booklendings.value = []; return; }
  const url = API_URL+'/booklendings?studentId=' + loggedUser.id + '&token=' + loggedUser.token
  booklendings.value = await (await fetch(url)).json()
}
async function deleteLending(lending) {...};
</script>
<template>
  <span v-if="loggedUser.token"> Here are you booklendings, {{loggedUser.email}}: </span>
  <span v-if="!loggedUser.token" style="color: red"> 'Please login to visualize booklendings!' </span>
  <ul>
    <li v-for="lending in booklendings" :key="lending.self">
      <a :href="HOST+lending.book">{{ ( books.value.find(b=>b.self==lending.book) || {title: 'unknown'} ).title}}</a> -
      <button @click="deleteLending(lending)">RETURN {{lending.self}}</button>
    </li>
  </ul>
</template>
```

## Questions?

marco.robol@unitn.it

## Back up slides

## Vue.js additional pointers

Tutorialon how to consume REST WebService from a Vue.js application: https://bezkoder.com/vue-js-crud-app/

Tutorial on the stack Vue.js + Node.js + Express + MongoDB: https://bezkoder.com/vue-node-express-mongodb-mevn-crud/

JWT authentication: https://bezkoder.com/jwt-vue-vuex-authentication/

## Build and serve Vue app from our backend

When ready to ship app to production, run the following: `npm run build`. This generates minified html+javascript frontend in `.\dist` folder. We can then serve the frontend on a dedicated server or on our API server.

```
// Serving frontend files from process.env.FRONTEND
app.use('/', express.static(process.env.FRONTEND || 'static'));
// If request not handled, try in ./static
app.use('/', express.static('static'));
// If request not handled, try with next middlewares ...
```

`EasyLib\app\app.js`

```
# Path to external frontend - If not provided, basic frontend in static/index.html is used
FRONTEND='../EasyLibVue/dist'
```

`EasyLib\.env`

- Serving over HTTP using ES modules syntax

```
<script type="importmap">
  {
    "imports": {
      "vue": "https://unpkg.com/vue@3/dist/vue.esm-browser.js"
    }
  }
</script>

<div id="app">{{ message }}</div>

<script type="module">
  import { createApp } from 'vue'

  createApp({
    data() {
      return {
        message: 'Hello Vue!'
      }
    }
  }).mount('#app')
</script>
```

## Vuetify - Material Design Framework

> Vue UI Library with beautifully handcrafted Material Components. No design skills required - everything you need to create amazing applications is at your fingertips.
> https://vuetifyjs.com/en/

## Google Authentication

**vs.**

## Stateless Authentication for RESTful APIs

Using **Passport** to *Sign In With Google* and **JWT** to sign and verify token and allows for stateless

## Cookies vs. localStorage and sessionStorage

> Rispetto ai cookies, gli oggetti web storage non vengono inviati al server con ogni richiesta - https://it.javascript.info/localstorage

```
localStorage.setItem('test', 1);
alert( localStorage.getItem('test') ); // 1
```