Project – Report

Vue-Exercise-Sandbox

Made By:

Љубомир Илиевски

Contents

[About the Project 3](#_Toc206329556)

[The main App 3](#_Toc206329557)

[Exercise Strings 4](#_Toc206329558)

[Read Write Methods 4](#_Toc206329559)

[Pinia Stores 4](#_Toc206329560)

[ExercisePickerToggleStore 5](#_Toc206329561)

[PickedExerciseStore 5](#_Toc206329562)

[Components 8](#_Toc206329563)

[Nav Bar 8](#_Toc206329564)

[Exercise Picker 9](#_Toc206329565)

[App component 9](#_Toc206329566)

[Code Editing 10](#_Toc206329567)

[ExercisesWithServer 11](#_Toc206329568)

[Node.js Express server 14](#_Toc206329569)

[The finished Product 15](#_Toc206329570)

# About the Project

The project is made in order to make access to the exercises specified in the book **"Maya Shavin, Raymond Camden - Frontend Development Projects with Vue.js, 2nd edition"** quick and easy.

Simply select a lecture on the left and a specific exercise that it holds, that exercise will pop in the middle, and its code on the right. Above the code the different files that correspond to that exercise will be seen, click each of them to switch between files for that current exercise. You can make changes in the code and click on the "Compile" button to commit those changes, if you did not make a mistake, you would see your changes in the middle, if you did, you would see nothing :). In that case simply click the button to reset the code and undo the mistake you made.

**PS.** In the project I sometimes use tailwind for styling.

If you want to read the big picture of the project first skip to [The Finished project](#_The_finished_Product) section.

# The main App

## Exercise Strings

A screenshot of a computer code

AI-generated content may be incorrect.The name of this section is a bit confusing, its name comes from the initial way of making this project, but that’s a story for another time.

Figure 1 - Exercise Object

In the project there is a folder named Exercise Strings. This folder holds subfolders of each chapter in the book, each of these subfolders holds a set of these “Exercise” objects, these objects hold all the meta data needed for later communication with the backend and distinction and representation of the exеcrises. Figure 1, shows an example of this exercise object.

So, from now on I will be calling these objects **Exercise objects**.

Each exercise object has a list of **“file objects”** as I will be calling them, they are under the “components” key value in the object.

## Read Write Methods

A screenshot of a computer code

AI-generated content may be incorrect.

Figure 2 - [RWMethods](https://github.com/Ljubomir-Ilievski/Vue-Exercise-Sandbox/blob/master/ExercisesFromBook/src/MethodsRW/RWMethods.js)

These two methods will be responsible for fetching and changing the code at the backend.

They will either read from the node express server backend (**fetchFileCode method**) or write to its file system (**writeFileCode method**). They will take a **filePath** for the backend file system, to know where to read or write, and the string code from the code editor if it to be written at the respected path.

## Pinia Stores

Note. Most of the logic used in the project comes from the pinia stores, jump to components and read as the methods and data are used from these stores, just a hint, do what you want.

### **A screen shot of a computer AI-generated content may be incorrect.**ExercisePickerToggleStore

Figure 3

It’s a simple store where its only purpose is to tell the [Exercise Picker](#_Exercise_Picker) component to be displayed or not. Maybe a store is redundant here, but it’s just what I used now, if you see another solution to this problem go nuts.

### PickedExerciseStore

Store [source code](https://github.com/Ljubomir-Ilievski/Vue-Exercise-Sandbox/blob/master/ExercisesFromBook/src/stores/PickedExerciseStore.js).

The most important store in the project, most of the components will use it’s methods and information to display the correct exercise at the center of the screen.

#### Store initial stateA screenshot of a computer code AI-generated content may be incorrect.

Figure 4

currentExercise: it’s just an exercise Object as we said in the Exercise Strings section,

appcode: will be a string containing the current selected exercise’s file code. This variable is important as it will be used as the value in the code editor.

currentActiveFile: is the name of a file of a picked exercise

ipExerciseServer: is the host of the backend node express.

#### changeFileCodeRWAPI()

A screenshot of a computer code

AI-generated content may be incorrect.

Figure 5

This method is used to change between the files of an exercise; a file object is sent to it.

We set “activeFilePath” to local storage and use it later to read from it upon page reloads, so the current exercise stays put upon reloads.

And of course, the method [fetchFileCode](#_Read_Write_Methods) is used to read from the backend based on the path of the new clicked file.

#### compileComponentRWAPI()

A computer screen shot of text

AI-generated content may be incorrect.

Figure 6

Based on the current exercise file selected, writes to the file system of the respectful file’s path to the backend, treat the post messages as a black box for now, when we get to the **exercisesWithServer** app part of the project they will be explained, for now just interpret them as commands we will send to the application with all the exercises.

At the beginning of the method, we use the parse() method from the vue/compiler-sfc'

library. The functionality in this project is irrelevant, we only use its exception throwing power, to throw an exception if a string of a Vue type component can’t be parsed (there are syntax errors in the component).

#### resetComponentCode()

A screenshot of a computer code

AI-generated content may be incorrect.

Figure 7

It is almost the same as the [*compileComponentRWAPI()*](#_compileComponentRWAPI()) *method, but we don’t check for syntax errors, (because we know the default code has no syntax errors).*

*The reset code strings are a part of the file objects in the corresponding exercise objects. As specified in the* [*Exercise Strings*](#_Exercise_Strings) *section.*

*Again, the post messages are a black box for now.*

*They use* **Cross-document messaging** and you will read about it in the [ExercisesWithServer](#_ExercisesWithServer) section.

#### changeExerciseRWAPI()

A screen shot of a computer code

AI-generated content may be incorrect.The purpose of this method is to change the context of the displayed exercise from one to another, much like context switching on a CPU. Sets the **activeFilePath** and **Exercise** to local storage to keep the state of the selected exercises upon reloads.

Figure 8

A router path is extracted from the file object (if it exists), this is used later in the application with the exercises, to tell it where its router settings are. This is used only on exercises who employ the Vue router, and it is necessary because different exercises will have different router settings or have none.

We fetch the new changed exercise’s code of its **App file** and change the **currentExercise** to the new clicked exercise from the send exercise object and **currentActiveFile** to App\_Name which is a global variable set to”App.vue” in this case. We basically tell it to represent the new clicked exercise’s App.vue’s code in the code editor.

The **changeExercise** action is called with **postMessage** to the backend Application.

## Components

Note. In the caption of each component’s picture will be a hyperlink to the source code of that component if you are interested, explaining every line of the source code will be redundant, if something picks your eye feel free to read the source code.

### Nav Bar



Figure 9 - [Navbar](https://github.com/Ljubomir-Ilievski/Vue-Exercise-Sandbox/blob/master/ExercisesFromBook/src/components/Nav_Bar.vue)

The navbar has a couple of treats. The hamburger menu button toggles on/off the listed exercises at the left of the screen ( [**ExercisePicker**](#_ExercisePicker) ), it uses the Pinia store, [**ExercisePickerToggleStore**](#_ExercisePickerToggleStore)**,**

The other two functionalities the navbar has are the **compile and Reset code button**, the compile button uses the [compileComponentRWAPI()](#_compileComponentRWAPI()) method from the [PickedExerciseStore](#_PickedExerciseStore), respectively the Reset code button uses the [resetComponentCode()](#_resetComponentCode())

### A screenshot of a cell phone AI-generated content may be incorrect.Exercise Picker

Figure 10 - [ExercisePicker](https://github.com/Ljubomir-Ilievski/Vue-Exercise-Sandbox/blob/master/ExercisesFromBook/src/components/ExercisePicker.vue)

This component uses all the [ExerciseObjects](#_Exercise_Strings), compacted into [lectures](https://github.com/Ljubomir-Ilievski/Vue-Exercise-Sandbox/blob/master/ExercisesFromBook/src/components/ExerciseStrings/Lectures/Lectures.js)

Each of these lectures is a list of exercise objects that correspond to that lecture’s exercises.

This list of exercise objects aggregates all the exercise objects. Example [List](https://github.com/Ljubomir-Ilievski/Vue-Exercise-Sandbox/blob/master/ExercisesFromBook/src/components/ExerciseStrings/Part01/Part01_lecture_object.js).

The Exercise Picker component uses the [Lecures\_component](https://github.com/Ljubomir-Ilievski/Vue-Exercise-Sandbox/blob/master/ExercisesFromBook/src/components/Lectures_component.vue), it creates a lecture component for every lecture object, and then every Lecture component creates an [Exercise component](https://github.com/Ljubomir-Ilievski/Vue-Exercise-Sandbox/blob/master/ExercisesFromBook/src/components/ExerciseComponent.vue) for every exercise object inside that lecture list.

Upon clicking a lecture, a dropdown list appears of all the exercises connected to it. This is achieved by a simple **toggleOpen** method upon clicking the lecture much like with the [ExercisePickerToggleStore](#_ExercisePickerToggleStore)

Upon clicking an exercise, the [changeExerciseRWAPI()](#_changeExerciseRWAPI()) method is called, and the whole exercise object is sent as an argument.

### App component

Source code: [here](https://github.com/Ljubomir-Ilievski/Vue-Exercise-Sandbox/blob/master/ExercisesFromBook/src/App.vue).

The component that connects everything.

A computer screen shot of text

AI-generated content may be incorrect.

Figure 11

On the beforeMount() lifecycle hook, we use the data saved in the local storage from either methods ([changeFileCodeRWAPI](#_changeFileCodeRWAPI()) or [changeExerciseRWAPI](#_changeExerciseRWAPI())) to hold on to the state that is the current picked exercise (the Exercise named local Storage) and current picked file of that exercise (the activeFilePath named local storage).

If these values are not set in local Storage, we simply default to the values set as initial values in the pickedExerciseStore state.

Here we also place the Monaco editor, and the iframe to the [other application](#_ExercisesWithServer) that holds all the exercises with the server to edit them.

### Code Editing

In this section we will talk about the code editing part of the project.

For a code editor I used [monaco-editor-vue3](https://www.npmjs.com/package/monaco-editor-vue3).

A screen shot of a computer

AI-generated content may be incorrect.

Figure 12

Above the code editor you can see a list of the [FileComponent](https://github.com/Ljubomir-Ilievski/Vue-Exercise-Sandbox/blob/master/ExercisesFromBook/src/components/FileComponent.vue). This component is fed a file objects of the current picked exercise, upon clicking the [changeFileCodeRWAPI()](#_changeFileCodeRWAPI()) method is called with the corresponding file object, to context switch the code from one file to another in the code editor.

The code editor always represents the **appCode** variable from the **pickedExerciseStore** state, so when we change the file or exercise, we always place the fetched code of the new file into this variable as seen in the pickedExerciseStore methods.

# ExercisesWithServer

This second application is another Vue application that has a node express server reading and writing to it. It’s the application that we communicate with postMessages and displaying its exercises with an iframe in our main application as said before.

Basically, all the logic from this application happens at the [App.vue](https://github.com/Ljubomir-Ilievski/Vue-Exercise-Sandbox/blob/master/Exercises_with_server/src/App.vue) component, so we will explain this component as much as possible.

A close-up of a word

AI-generated content may be incorrect.Firstly, as template, we will simply use the “component” component from Vue, that enables dynamic component changes.

Figure 13

The idea is to dynamically import the App.vue component from the corresponding exercise based on the path the application is instructed with.

This instructions come as **Cross-document messaging** communication as shown in the [CompileComponent](#_compileComponentRWAPI()) and [ChangeExercise](#_changeExerciseRWAPI()) methods in [PickedExerciseStore](#_PickedExerciseStore) from the main applictaion.

A computer screen shot of a program code

AI-generated content may be incorrect.

Figure 14

This application responds to these instructions while listening on “message” event listener.

There are two possible scenarios to handle: if the [compileComponent()](#_compileComponentRWAPI()) method is called, the site is simply reloaded with the updated code, whereas if we switch from one exercise to another, the App.vue component of the new exercise must be dynamically imported using the provided file path and used as the **compiledComponent** in the dynamic template. In the latter case, special care is required when the new exercise uses a Vue Router: the path to the new router should be stored in local Storage, and the entire application must then be reloaded. During initialization, as shown in the main.js file in Figure 15, the application checks local Storage for a router path; if one is present, the corresponding router is initialized, otherwise a default router is used.A screenshot of a computer program

AI-generated content may be incorrect.

Figure 15

# Node.js Express server

Source code [here](https://github.com/Ljubomir-Ilievski/Vue-Exercise-Sandbox/blob/master/Exercises_with_server/src/server/server.js).

Finally, we need to explain how the changes in the code happen.

Now come the [RWMethods](#_Read_Write_Methods) we specified from the beginning.

These methods contact this node.js express server that resides inside the iframed second frontend application. It has two endpoints, as shown in Figure 16 and Figure 17 . The endpoint in Figure 16 responds to the **fetchFileCode** function, it receives a path to a file and uses the “fs” JavaScript library to read its contents and send them back.

The endpoint in Figure 17 is used by the **writefileCode** function. It receives a path to a file and body contents so to what to rewrite the file with. Upon rewriting this file, the application now hot reloads with its new contents thus giving the updated look in the main application.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 16

A screenshot of a computer

AI-generated content may be incorrect.

Figure 17

# The finished Product

A screenshot of a computer

AI-generated content may be incorrect.

Let’s make a simple summary to how it all works.

We have one Vue application that communicates with another Vue application through an iframe.

The first application is responsible for giving instructions to the second as to what exercise should be displayed in it and to retrieve the code from its files and to rewrite the code.

The second application listens to these instructions and displays the appropriate exercise.

The node.js server gets instructions from the first Vue application for reading and writing to the second. When it writes to it the hot-reloader detects the changes and hot reloads the second application giving the changing results in it.