Assignment

Setup

Virtual machine

In the given `.zip`-File is an OVF template to create a virtual machine. You can import the given template either to VirtualBox or VMWare Player. No special setup is required.

The virtual machine is configured to log you in automatically. You'll find a link to Git repository on the desktop when the virtual machine is up and running.

Side note: It's recommended to pull the repository before starting the assignment as it might be the case that there are outstanding changes!

Docker

If you don't want to use the virtual machine, or if you just prefer to use Docker, you can use the image knsit/yarn to avoid the Installation of the whole toolchain (Node.js, Yarn). Just navigate to the `assignment` directory in the Git repository and run the container like this (Linux/Mac):

```
1 docker run --name yarn --rm -v "$PWD":/home/yarn knsit/yarn:1.3.2 run
    dev
```

or Windows:

```
1 docker run --name yarn -v "$((pwd).Path):/home/yarn" --rm -p 8080:8080
    knsit/yarn:1.3.2 run dev
```

As editor/IDE we recommend to use Visual Studio Code with the following plugins:

- EditorConfig for VS Code
- TSLint Vue
- Vue TypeScript Snippets
- yarn

The plugins are not required but they would make it easier.

Dev-Server

To run the Dev-Server execute the following command:

```
yarn run dev
# or if you prefer npm
npm run dev
```

No matter if you're using *npm* or *yarn* the task runner should open a new browser tab/window. The tiny web application you're confronted with contains:

- the lecture notes rendered as HTML
- this document which will lead you through this assignment (also rendered as HTML)
- the API spec (OpenAPI) of a custom **I**nternet **C**huck **N**orris **D**ata**b**ase we'll use for one part of the assignment
- Links to the components you have to complete as part of this assignment
- Links to the solutions for all parts of this assignment

Part 1

The first part of the assignment covers the basic Vue.js concepts for databinding like `v-model`, template strings (`{{myVariable}}`), `v-for`, `v-if` and `v-else`.

Have a look at the folder `assignment/src/components/assignments/part1`.

It contains (more or less) a typical Vue.js TypeScript component:

- `index.ts` - controls the exports of the module
- `part1.html` - contains the template rendered with this component
- `part1.scss` - contains styles required **only** for this component (common styles are located in the file `main.scss`)
- `part1.ts` - contains the business logic of the component and imports the *.html* and the *.scss* files

The file `part1.html` contains a tab view with two tabs. One to register your favorite movie actors and one for displaying them as card view. Once you have "submitted" the actor registration form (actually you don't submit anything as we are in a single page application) the actor should show up in the second tab.

The required HTML code is already there but all data bindings are missing. Start by binding the form to the properties of the `currentInputActor` in the component. If you are not sure if the binding is working correctly have a look at the browser console. The given model logs the new value of a property every time a setter is called (of course just for debugging purposes! Don't do that in production!).

If you have no idea how to start have a look at the docs. Or just ask us!

Part 2

The second part deals with TypeScript in general and in combination with Vue. js in particular.

At first have a look at the API spec of the ICNDB integrated in the application. The most interesting endpoint is `/jokes` because we want to fetch from 1 to 500 Chuck Norris jokes and display them once more as card view.

Because writing TypeScript but handling the API responses as objects of type `any` is nonsense start by implementing the required interfaces. We recommend to implement one interface (e.g. `JokesArrayResponse`) as a wrapper for the outer response and one interface (e.g. `Joke`) as wrapper for the concrete inner object.

If you have completed these two interfaces you're ready to implement the HTTP calls to the API. The recommended way to do HTTP communication in Vue.js is to use axios (especially with TypeScript because axios provides typings). Of course you can use other libraries but axios is already installed so it's the fastest way to get started.

When you have logged your first successful responses you're good to go to display the retrieved data. The given template already contains the required markup but some bindings and directives are missing. The *TODO* entries should guide you through this relatively quickly.

Now that you're able to display one joke (the default page size) it's time to adopt the drop down by adding a click-handler. The handler should pass the new page size to the component and trigger a reload of the items.

Hint: the doc page mentioned in the first part also explains how to register click handlers.

The last thing we want to achieve is that the input fields at the top of the page are used to change the first and last name displayed in the jokes (live of course).

Hint: you might need an extra `Joke` implementation to accomplish this behaviour.

A few words about asynchronous calls in TypeScript

Additionally to the classic callback style to handle asynchronous calls (callback hell - yeah!) TypeScript provides a nice feature to await the response and handle it in a (pseudo) synchronous way. The following snippet shows how to use *axios* with TypeScripts `async` and `await`:

```
1 private async doCall() {
2     let url = 'https://icndb.kns-it.de/api/v1/jokes/random';
3     let response: AxiosResponse<JokesArrayResponse> = await this.axios.
          get(url);
4     // ...
```

```
5 }
```

Part 3

You're still here?

Well, it's time to make your hands **really** dirty!

The last part of the assignment covers the underlying concepts of data binding in Vue.js. (Don't you want to go for a mulled wine after all?)

You have been warned!

As already explained in the lecture Vue.js uses `Object.defineProperty()` (Mozilla Docs) to create a proxy for every property of every object defined in a component.

The file `observer.ts` already contains all necessary classes. What's missing is the logic to create these proxy properties and the whole event handling.

The given class structure is oriented at the real implementation of Vue.js but shortened and simplified to focus on the concepts.

The best starting point is probably the original source code. When you have an idea how to start, implement the classes in this order:

1. `Dependency`
2. `Observer`
3. `ModelBinder`
4. `DOMBinder`

The given code contains *TODOs* and hints to guide you through the implementation. The component `AssignmentPart3Component` and the model `Person` are predefined and can be used directly.

As soon as you have completed the first 3 parts you should be able to update the GUID of the person by clicking the button in the view.

To test the last part (a way-two binding if you like) the view contains two input fields (one of them is disabled). When you're changing the *first name* in the enabled input element the *first name* in the disabled input should change too.

A few words about the `keyof` feature in TypeScript

Since version 2.1 TypeScript has a nice feature called `keyof` (Announcement).

This is particular helpful when you have to deal with names of properties (as in the `Observer` to bind a property to a view element).

If you define the `Observer` like this:

```
class Observer<TModel> {}
```

and the method `addModelBinding(...)` like this:

```
addModelBinding<K extends keyof TModel>(propertyName: K, domId: string
    = propertyName) {
    // ...
}
```

TypeScript will throw compiler errors when you pass a `propertyName` which is not present in the current class. Of course this only takes affect when you stay within the TypeScript universe!