Software Requirements Specification

Program for creating web applications based on concrete values

Prepared by:
Jaroslav Švarc

1. May 2023

Contents

Chapter 1

Introduction

1.1 Description and focus of the project

The idea behind this project is to design a programming system that lets developers easily construct web UI elements by starting from concrete values. More specifically, developers can start defining behaviour of the given element only when all necessary data for the given element is provided. That way, the programming environment can help the user define behaviour of the element based on those concrete inputs.

The described project aims at creating web application to which the developer can provide data. Then the application will provide a selection of UI elements for which the developers can then create behaviour. The project will recognize different data formats for 2 of the 7 basic UI elements as defined by 7GUIs and a third element defined as a basic to-do list.

1.2 Technologies used

- F#
- Fable
- Elmish

1.3 References

- 7GUIs - `https://eugenkiss.github.io/7guis/tasks/`
- F# - `https://fsharp.org/`
- Fable - `https://fable.io/`
- Elmish - `https://elmish.github.io/elmish/`
- Project based on description from - `https://d3s.mff.cuni.cz/students/topics/programming-systems/`

1.4 Conventions of this document

Example of a link to a website:

- `https://fsharp.org/`

Chapter 2

Overall Description

2.1 The reason for the creation of the project

The reason for the creation of this project is to provide a new way of creating web UI elements based on concrete values. The same principle as defined in chapter 1 can be seen in a Jupyter Notebook where it lets you load concrete data before writing more code.

2.2 Main functions

The two main functions of this project will be:

- Recognition of data which fits a certain UI element
- Modification of behaviour of an UI element based on the concrete values

2.3 Motivational example of usage

- The user creates data in a JSON format
- The user provides this data to the application
- The system provides the user with a selection of UI elements which can use the provided data
- User defines the behaviour of the element
- User can load a page with the defined element and try it out

2.4 Application environment

The application will use the SAFE stack, which uses Dotnet core and NodeJS. This means that the application should be available on most major operating systems. The graphical user interface will be accessible via a browser.

2.5 Application restrictions

The project will recognize different data formats for 2 of the 7 basic UI elements as defined by 7GUIs and a third element defined as a basic to-do list. Recognition for other types of elements will not be implemented.

Chapter 3

External interface

3.1 User interface

The application will provide a GUI. The main window will serve as an overview of the UI component created. Developer will be able to modify its behaviour on this screen as well. Other window will be used to show the component which the developer created.

3.2 Inputs and outputs

The developer must provide data to the application before the system is able to create a component. Once the developer defines the behaviour of the component, the output will be a file containing the source code for the component.

Chapter 4

Description of functionality

4.1 Creating component based on data recognition

The developer will provide data to the application and the application will try to recognize the format of the data. If the data fits a certain format, the program will provide a selection of components for which this data can be used.

The basic format which all data should adhere to is shown below. The JSON data should contain a single property called 'data' which then contains properties specific to the UI element.

```
{
  "data": {
    "custom": {
      "custom1": "value1",
      "custom2": "value2",
      ...
    }
  }
}
```

Example of data which can be used to create a simple to-do list. It contains the required data property which then contains a list of tasks. Each task has an ID, task description and a state of completion property.

```
{
  "data": {
    "tasks": [
      {
        "id": 1,
        "task": "Complete project proposal",
        "completed": false
      },
      {
        "id": 2,
        "task": "Prepare presentation slides",
        "completed": true
      },
      {
        "id": 3,
        "task": "Send meeting agenda to team",
        "completed": false
      }
    ]
```

```
    }
  }
```

When the developer supplies this data to the application, the application analyzes the data. The application can see that the data contains a list of items and each item has the properties matching a description of a task. The data matches a specific format and the application recognizes it as data for a to-do list and no other specified component can use this data.

The application then gives the developer an option to create the to-do list component. The developer can either create the component or cancel. Then the developer can either supply more data and create other components, modify the behaviour of the to-do list, or view the already created components in a separate window.

4.2 Modification of component behaviour

Generally, most components can accept input from an end-user using a specified interface such as a button, input field or a checkbox. Once a component is created, the developer can modify the behaviour of these potential inputs. More specifically, these inputs serve as triggers which cause some action. This action can then be modified by the developer using the provided application constructs.

Let's look at the previous example of a simple to-do list. A to-do list contains a list of items. Each item provides a checkbox which the end-user can click on. What happens after that is up to the developer who specified the behaviour. For example, the developer can specify that after clicking the checkbox, the task will be deleted. Or the item can be changed in such a way indicating that it is completed. Or this item can be duplicated 10 times. These are just examples of potential behaviour in response to an end-user clicking a checkbox.

The application will give the developer a set of actions and the developer can combine these actions to create more complex behaviour. These actions are for example: deleting data, changing the data, loops, etc.

4.3 Trying out the created component

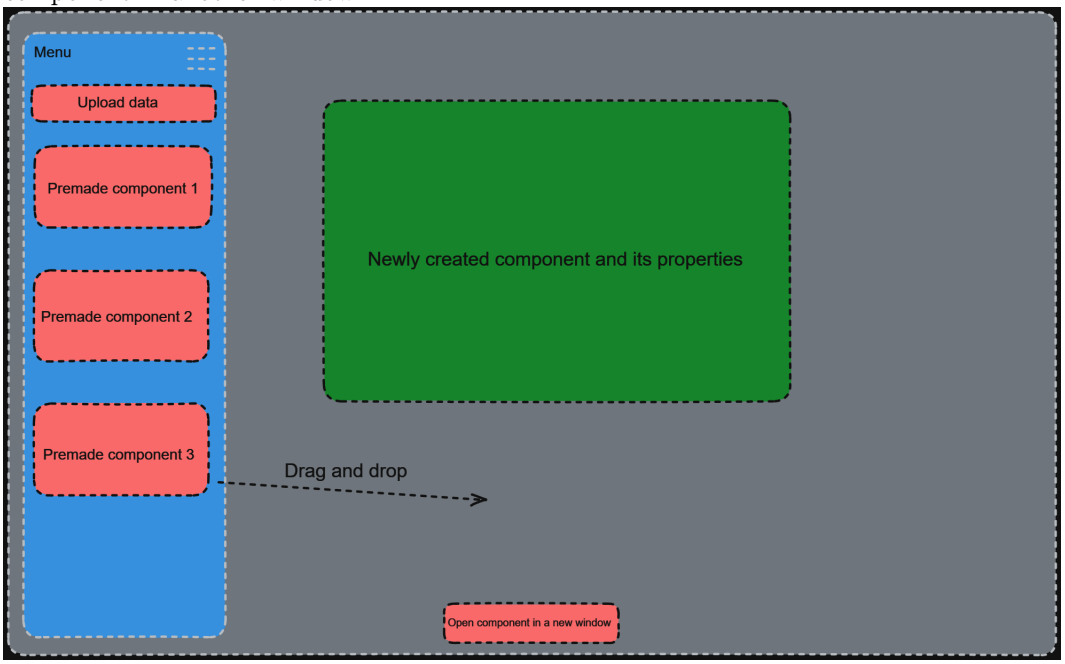
The user will be able to open a window containing the newly created component and try out its functionality.

Chapter 5

Views

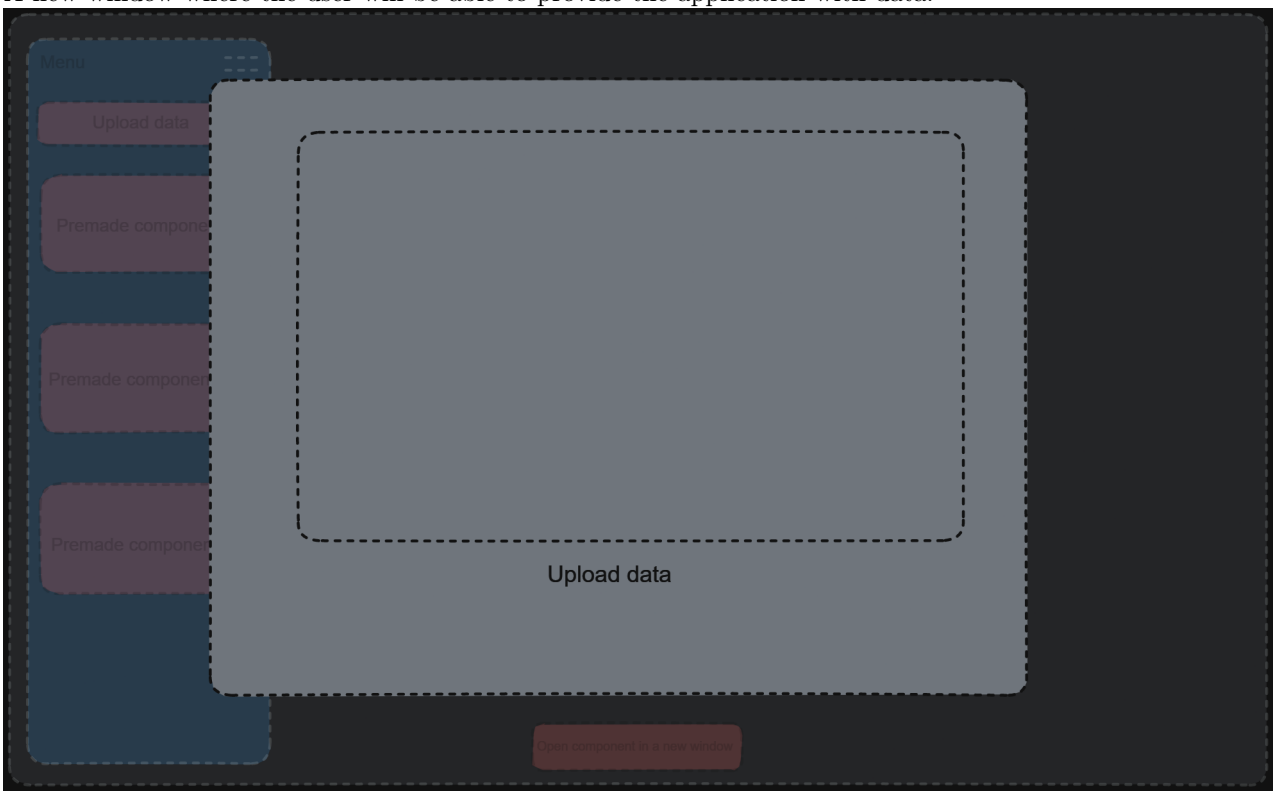
5.1 Main window

The main window where the user will be able to modify the behaviour of the component, provide new data to the application or create a new premade component. The user will also be able to open a preview of the component in another window.



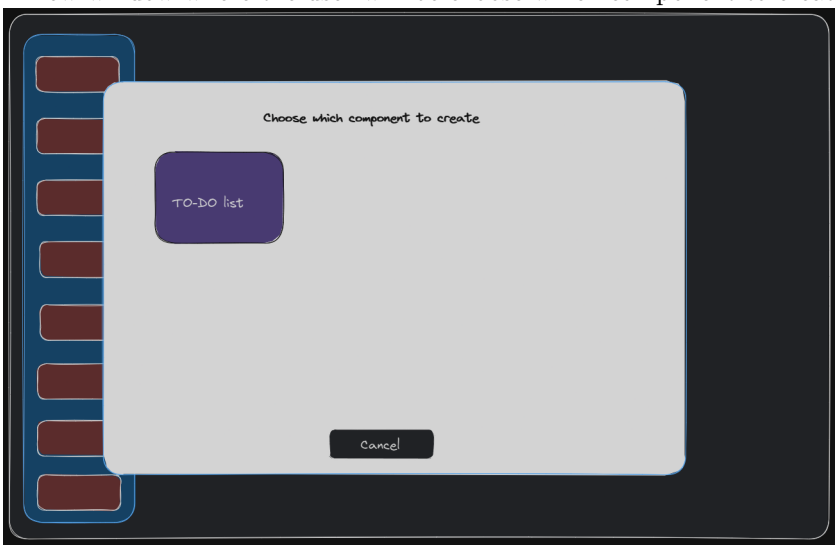
5.2 Loading data

A new window where the user will be able to provide the application with data.



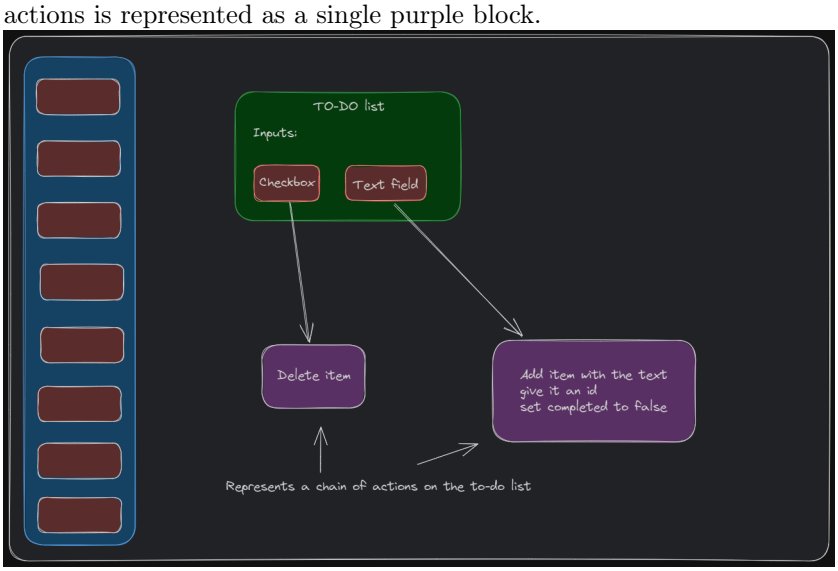
5.3 Component creation

A new window where the user will be choose which component to create.



5.4 Behaviour modification

Developer can use different actions chained together to modify the behaviour. In this image, the chain of actions is represented as a single purple block.



Chapter 6

Milestones

6.1 Implementation milestones

- Creating the UI of the application. These parts should be implemented:
  - the main screen
  - blocks
  - side menu
  - upload screen
  - modification blocks
  - preview window
  - Estimated date of completion: 23.7.2023
- Implementation of data recognition for the specified elements. This includes specifying the data formats, implementing a parser to recognize the different formats and supply the the frontend with a selection of suitable UI elements. This recognition does not have to be implemented for all elements at once. We can start with a single UI element and then add support for more. Estimated date of completion: 10.8.2023
- The creation of the selected UI element. This includes generating code for the element along with the supplied data so that it can be viewed in the preview window. Estimated date of completion: 17.8.2023
- Implementation of the different modification blocks starting with blocks which are easier to implement. These block should have the ability to be chained together and form a pipeline. Estimated date of completion: 3.9.2023
- Implementation of the final code generation. The program should generate code containing the UI elements created by the developer, including the specified behaviour for the inputs. Estimated date of completion: 18.9.2023