FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

ASSIGNMENT OF MASTER'S THESIS

Title:	WebAssembly Approach to Client-side Web Development using Blazor Framework
Student:	Bc. Matěj Lang
Supervisor:	Ing. Marek Skotnica
Study Programme:	Informatics
Study Branch:	Web and Software Engineering
Department:	Department of Software Engineering
Validity:	Until the end of summer semester 2019/20

Instructions

The majority of applications we use every day shifted from the desktop to the web. And with this transition, there was an explosion of approaches to the client-side development. The most recent advancement is a WebAssembly technology which allows executing low-level code in a web browser. A goal of this thesis is to create a proof-of-concept application using this technology and evaluate its strengths and weaknesses.

Steps to take:

Review the WebAssembly technology and the Blazor framework.
Compare Blazor to the state-of-the-art client-side web development approaches.
Design and create a proof-of-concept application in Blazor.
Evaluate Blazor's strengths and weaknesses and its readiness to develop modern web applications.

References

Will be provided by the supervisor.

Ing. Michal Valenta, Ph.D.
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague December 10, 2018

Czech Technical University in Prague

Faculty of Information Technology

Department of Web and Software Engineering

Master's thesis

WebAssembly Approach to Client-side Web Development using Blazor Framework

Bc. Matěj Lang

Supervisor: Ing. Marek Skotnica

7th May 2019

Acknowledgements

In this place I want to thank Bc. Katerina Černíková and Mgr. Jakub Klement for language corrections. I want to thank my master thesis supervisor - Ing. Marek Skotnica for his patience and advice. CTU, Faculty of Information Technology also deserve thanks for valuable information and education I got there. Last but not least I want to thank the management of High school and College of Applied Cybernetics Ldt. for the possibility to realize the research here.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46(6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the "Work"), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work for nonprofit purposes only, in any way that does not detract from its value. This authorization is not limited in terms of time, location and quantity.

In Prague on 7th May 2019 .....

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Lang, Matěj. *WebAssembly Approach to Client-side Web Development using Blazor Framework*. Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2019.

Abstrakt

V současné době je mnoho aplikací tvořených jako webové aplikace, které se snaží poskytnout uživateli stejný nebo lepší zážitek jako desktopové aplikace. V této diplomové práci jsou shrnuty hlavní frameworky využívané pro tvorbu webových aplikací s důrazem na jejich komparaci z hlediska náročnosti učení pro programátora, náročnosti na RAM a velikosti stránek stahovaných webovým prohlížečem a též z hlediska popularity v rámci vývojářské komunity. Hlavním tématem této práce je technologie WebAssembly s použitím Blazor frameworku pro tvorbu webových stránek. V této práci jsou tedy zkoumány následující frameworky: Blazor, Angular a Vue.js.

Klíčová slova Blazor, MVC u klienta, Webové technologie, Asp.Net, .Net, .Net Core

Abstract

Nowadays there are many applications developed as web applications which aspire to provide same or better user experience as desktop applications. Aim of this diploma thesis is to summarize the most common frameworks used for web application development and compare them with emphasis on their learning difficulty, RAM requirements, browser downloading size of web pages

and popularity around community of developers point of view. The core of this thesis is focused on WebAssembly technology usage with Blazor framework for web applications development. Examined frameworks in this thesis are: Blazor, Angular and Vue.js.

Keywords Blazor, Client-side MVC, Web technology, Asp.Net, .Net, .Net Core

Contents

List of Figures

List of Tables

Introduction

Thanks to technological progress, it is possible to create most of the applications as the web.[3] Web applications provide remote access to data and better security. These applications also enable cooperation and simultaneous work of several people on the same data. For simplification of web applications development, there are emerging many web technologies which are transferring experience from desktop application development. To support these new technologies there are also emerging new standards which are being implemented by browser creators. This implementation is based on the match of the newly proposed technology with an internal policy of the implementer. If there is a new standard implemented by the significant majority of the browsers it enables the use of this new technology in the production environment of the web application. One of these technologies is WebAssembly which enables triggering of binary instructions on a virtual machine. This means faster execution of operation thanks to the compilation from high-level programming language to binary form. Single Page Application (SPA)

Last part of this thesis contains proof of concept application for learning support with Blazor framework. This application extends my bachelor's thesis[2] and use Blazor framework.

Motivation and Objectives

Nowadays most of the applications are developed as SPA with the usage of one of the Client-Side frameworks which will be evaluated in this thesis. These frameworks have to be used in JS language or in other languages which are compiled to JavaScript (JS) as TypeScript. This thesis is focused on Blazor for client-side web application development. This framework is based on WebAssembly technology which allows running application and library written in `C#` Language directly in the browser. Blazor framework uses a modern approach to web application development. For example SPA creation and also brings simplification to programmers who can use `C#` for developing web

application behavior both backend and GUI. The aim of this thesis is to describe the Blazor framework and summarize use in production environment. This thesis will also include the development of web application in Blazor framework and problem analysis.

Structure

This thesis is organized as follows:

- In chapter 1 of this thesis, the properties of the three most popular client-side frameworks are summarized. For each of the selected frameworks, the following properties are evaluated:
  - i Learning curve of technology, language and structure of selected framework.
  - ii Size needed to be downloaded to the browser to ensure proper functionality of this framework.
  - iii Existence of developer tools to support the creation of the application using these frameworks.
  - iv The possibility of implementing the framework within the existing application and customizing user experiences.
  - v What languages are used with these frameworks.
- In chapter 2 of this thesis is focused on WebAssembly and Blazor usability for developing web application.
- In chapter 3 of thesis there is comparison of web application development in SPA, Model-View-Controller (MVC) and Pages technology.
- In chapter 4 of thesis is described using Blazor technology to create Web application. This sample application will be focused on support of agenda processes in education. End user of this application will be a teacher.

Chapter 1

State Of The Art Client-Side frameworks

Based on these articles [4], [5] and Github stars [6] values shown in figure 1.1 below are the most popular web frameworks Vue.js, React and Angular. In following figure 1.2 history of values for Vue.js, React, Angular and Blazor is displayed. Stars evaluation for Blazor is disorted because this framework is really new and doesn't have a significant history of stable versions. This framework's code was at beginning of year 2019 moved to aspnet/AspNetCore repository. Blazor is described below in section 2.2. Each source code in this chapter is based on Visual Studio template. Components and pages focused on desplaying the Blazor functionality were developed by the author of this thesis.

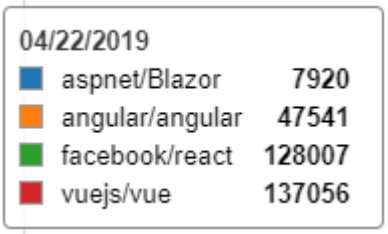


Figure 1.1: Github stars comparison of framework

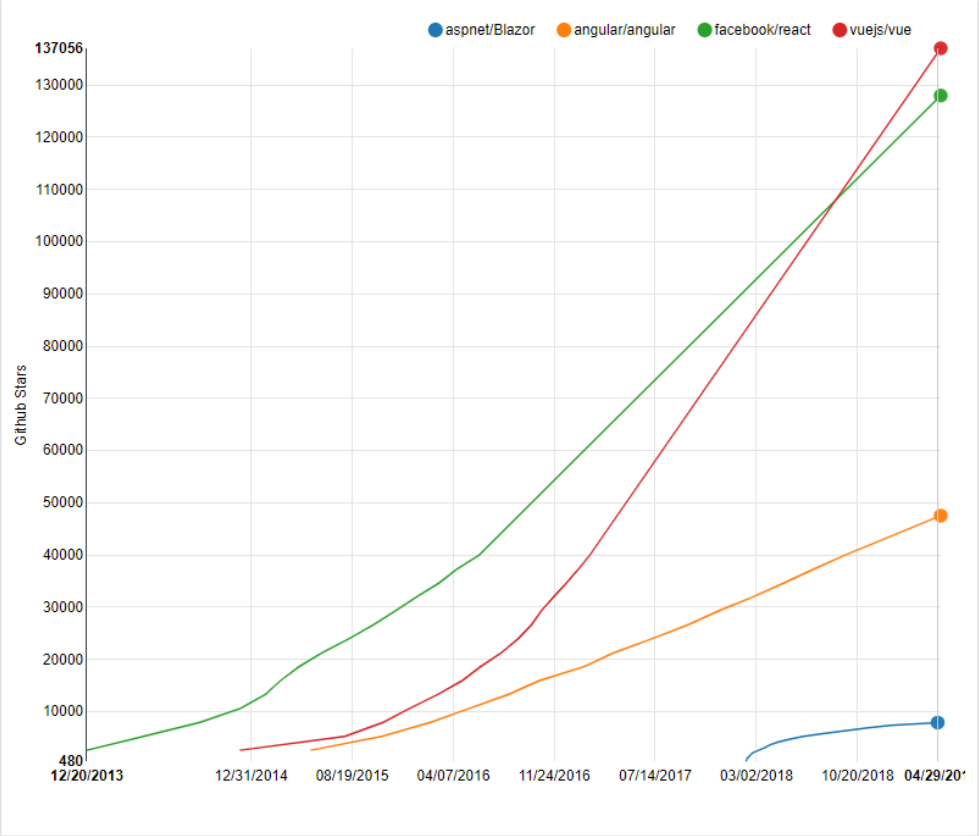


Figure 1.2: History values of github stars

1.1 Common features

Many of web application framework contains common features which is described below.

- Templates
- Components
- Routing
- Services
- Dependency injection

1.1.1 Templates

Many web frameworks prepare templating system for developers. There are two different types of template system based on style for creating a template.

The first type of template system is based on HTML which is extended with markup language defined by web framework. This type of template system is used by ASP.NET MVC, Angular, React JSX, Vue.js and more. These types of template system is similar, easy to learn and use but need more cpu time to process template input and complexity of template parser. Some frameworks compile a template when construction of web application is started. The second type of template system builds the whole HTML from a different language. These types of template system create HTML page from zero. Learning the language for these systems is difficult because they have different syntax. The languages which are used for creating a template are, for example, Slim, Haml, Pug and more. These languages are a lightweight version of HTML and allow to quickly create an HTML page. The names of HTML tags are preserved but the attributes and contents of the tag are written in a lightweight form. In the Pictures 1,2,3 there is an example of the same HTML page in this language.

1.1.2 Components

Components represent single unit which contains templates. During rendering Hypertext Markup Language (HTML) page all custom tags are processed and replaced with the associated template. The component specifies a custom HTML tag which can be used in other templates. Some frameworks allow to attach set of styles represented by Cascading Style Sheets (CSS) or language which is compiled into CSS to Component. Another important part of component is the logic associated with a component. Logic can be written in many languages. Component can be nested in another component to create a larger functional unit.

1.1.3 Routing

Routing is technology to map Uniform Resource Locator (URL) path to page. In many framework routing has capability of mapping parts of URL segment to property or variable. When URL is requested.

1.1.4 Services

Services contain function, event, properties, constants and other future which is needed by web application. It can represent database connection, Api client, state storage and many more. Services are shared through the whole web application for easy communication between any component in application. Any component in application can hook on event which is provided by service. Some frameworks provide service through dependency injection which is described below. Other frameworks use Service store which provides service to component.

1.1.5 Dependency injection

Dependency injection is a technique used in object programming. Where one object can be passed to another. Object which is passed is called service and is passed to client. Where client is a method or an object requiring service from dependency injection. Dependency injection contains three types of service: Transient, Scoped and Singleton.

1.2 Angular

Angular technology has two generations. The first generation was angularJS.[7] Second generation formally known as Angular2.[8] This framework is developed by Google INC and is the most popular client-side framework for developing SPA which is designed for the large web applications. Angular is written in TypeScript language. The architecture of this framework is separated into Component, Modules, Services, and Routing. The component in angular contains functionalities and view templates which are a combination of HTML and Angular markup. Angular markup provides data binding between HTML and functionalities which are scripted in TypeScript. Angular also contains modules which group multiple components into a single function unit. Modules can contain a component, services and another module in one unit. Angular module declare compilation context contains all components and service.

Angular is a powerful tool and framework usable to create application as Gmail or Google Docs, but learning this technology is difficult because it has many features which developer needs to know and learn before he starts using this technology.[9] Beginner developer needs to learn JavaScript for Node.js and web application. Then developer must learn Node.js Command line interface (CLI) for creating, compiling and starting application. Node.js contains package manager named Node.js package manager (NPM). Angular tool and framework downloaded trought NPM contains Angular CLI. Another necessary knowledge is Typescript language. It is also necessary to know the standard languages for creating websites such as HTML, CSS and JS.

In Angular version 7 size about 86KB compressed by gzip.[10]

Angular has many developer tools which can help in developing and debugging web application. For example Angular Augury [11] is an extension to web browser which can inspect component tree. Angury can log event and fire any event defined in component. With this extension developer can trace dependency injection.

Angular provides simplification to modifying Document object model (DOM). For example:

- Rendering content of variable in Angular template using {{ variable }} directive.

- Render collection of items using *ngFor attribute.
- Event binding with attribute (click)=”method()”
- Two-way binding with tag attribute [(ngModel)]=”variable”

Angular is a colossal tool and technology usable to create large web application but implementing angular in existing application will be complicated. Angular has complex architecture whose manual handling is difficult. Summary of Angular is showed in section 1.2.

Learning curve	High
Size	∼86KB
DevTools	Angular Augury
Languages	TypeScript

Table 1.1: Table showing summary of Angular

1.2.1 Example of Todos

There is simple hello world template created from visual studio. Only explaining code is show in source codes below.

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Angular</title>
  <base href="/">

  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
  <app-root>Loading...</app-root>
</body>
</html>
```

Source Code 1.1: This is base html page.

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html'
})
export class AppComponent {
```

```
  title = 'app';
}
```

Source Code 1.2: First component contains app tag definition.

```
<body>
  <app-nav-menu></app-nav-menu>
  <div class="container">
    <router-outlet></router-outlet>
  </div>
</body>
```

Source Code 1.3: Template for app tag.

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { HttpClientModule, HTTP_INTERCEPTORS } from '@angular/common/http';
import { RouterModule } from '@angular/router';

import { AppComponent } from './app.component';
import { NavMenuComponent } from './nav-menu/nav-menu.component';
import { HomeComponent } from './home/home.component';
import { CounterComponent } from './counter/counter.component';
import { FetchDataComponent } from './fetch-data/fetch-data.component';

@NgModule({
  declarations: [
    AppComponent,
    NavMenuComponent,
    HomeComponent,
    CounterComponent,
    FetchDataComponent
  ],
  imports: [
    BrowserModule.withServerTransition({ appId: 'ng-cli-universal' }),
    HttpClientModule,
    FormsModule,
    RouterModule.forRoot([
      { path: '', component: HomeComponent, pathMatch: 'full' },
      { path: 'counter', component: CounterComponent },
      { path: 'fetch-data', component: FetchDataComponent },
    ])
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Source Code 1.4: Module for app component.

```
import { Component,Input } from '@angular/core';

@Component({
  selector: 'app-todo-component',
  templateUrl: './todo.component.html'
})
export class TodoComponent {
  @Input() item: object;
  public onChange(event) {
    debugger;
  }
}
```

Source Code 1.5: Todo tag represented by todo component.

```
<div>
  <input style="display: inline" type="checkbox" [(ngModel)]="item.done" />
  <div style="display: inline" [ngStyle]="{ 'display': 'inline',
  ↪  'text-decoration': (item.done ? 'line-through' : '') }">
    {{item.description}}
  </div>
</div>
```

Source Code 1.6: Template for Todo component.

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-todos-component',
  templateUrl: './todos.component.html'
})
export class TodosComponent {
  public items = [{description:"Hello",done:false}];
  public addItem(description: string) {
    this.items.push({ description: description, done: false });
  }
  public f(item) {
    return !item.done;
  }
}
```

Source Code 1.7: Todos page definition.

```
<h1>Todos</h1>

<div>
  <app-todo-component *ngFor="let item of items" [item]="item">
  </app-todo-component>
  <input #newItem (keyup.enter)="addItem(newItem.value)"
   ↪  (blur)="addItem(newItem.value); newItem.value='' ">
  <button (click)="addItem(newItem.value)">Add</button>
  {{ items.filter(f).length }}/{{items.length}}
</div>
```

Source Code 1.8: Template for Todos page.

1.3 React

React is JavaScript library for building User interface (UI). React is developed by Facebook Inc. Part of React is the possibility to use the better template system called JSX. JSX is XML like syntax used in JavaScript file. JSX is described below. In many ways, React is similar to Angular as component-based, designed for large application and when using JSX needs to pre-process JavaScript file.

There are two methods of pre-procesing(compiling) JSX. The first method is similar to Angular, so that Node.js is used to compile JSX into plain Javascrip. The second method is based on runtime compilation in browser. For runtime compilation can be used Babel or other runtime compilation that has React JSX capabilities.[12]

React learning the curve is flatter than in Angular. Developer using Angular need to learn only JavaScript, HTML and CSS.

With React is almost every time used Redux state management. According to Redux website[13] "Redux is a predictable state container for JavaScript apps. It helps the developer to write applications that behave consistently, run in different environments (client, server, and native), and are easy to test. On top of that, it provides a great developer experience, such as live code editing combined with a time traveling debugger."

1.3.1 Example of Todos

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1,
     ↪  shrink-to-fit=no">
    <meta name="theme-color" content="#000000">
    <base href="%PUBLIC_URL%/" />
```

```
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json">
    <link rel="shortcut icon" href="%PUBLIC_URL%/favicon.ico">
    <title>React</title>
  </head>
  <body>
    <noscript>
      You need to enable JavaScript to run this app.
    </noscript>
    <div id="root"></div>
  </body>
</html>
```

Source Code 1.9: Main html file.

```
import React, { Component } from 'react';
import { Route } from 'react-router';
import { Layout } from './components/Layout';
import { Home } from './components/Home';
import { Todos } from './components/Todos';
import { Counter } from './components/Counter';

export default class App extends Component {
  static displayName = App.name;

  render () {
    return (
      <Layout>
        <Route exact path='/' component={Home} />
        <Route path='/counter' component={Counter} />
        <Route path='/todos' component={Todos} />
      </Layout>
    );
  }
}
```

Source Code 1.10: App component.

```
import React, { Component } from 'react';
import { Todo } from './Todo';

export class Todos extends Component {
  static displayName = Todos.name;

  constructor (props) {
    super(props);
      this.state = { todos: [{done:false,description:"Ahoj"}],
      ↪  newDescription:"" };
      this.addNewTodo = this.addNewTodo.bind(this);
      this.handleChange = this.handleChange.bind(this);
```

```
12    }
13
14      addNewTodo() {
15          const newTodos = this.state.todos.slice();
16          newTodos.push({ done: false, description: this.state.newDescription
             ↪  });
17          this.setState({
18              todos: newTodos,
19              newDescription: ""
20          });
21      }
22      handleChange(event) {
23          this.setState({ todos: this.state.todos.slice(), newDescription:
             ↪  event.target.value });
24      }
25
26      render() {
27          const items = [];
28          for (let i in this.state.todos) {
29              items.push(<Todo key={i} value={this.state.todos[i]}></Todo>);
30          }
31      return (
32        <div>
33              <div>
34                  {items}
35              </div>
36              <input type="text" value={this.state.newDescription}
                 ↪  onChange={this.handleChange} />
37              <button onClick={this.addNewTodo}>Add</button>
38              {this.state.todos.length}/{this.state.todos.length}
39        </div>
40      );
41    }
42  }
```

Source Code 1.11: Todos page definition.

```
1   import React, { Component } from 'react';
2
3   export class Todo extends Component {
4     static displayName = Todo.name;
5
6     constructor (props) {
7         super(props);
8         this.state = { done: props.value.done, description:
             ↪  props.value.description };
9         this.handleChange = this.handleChange.bind(this);
10    }
11
12      handleChange(event) {
13          debugger;
```

```
14          this.setState({ done: event.target.checked, description:
            ↪  this.state.description });
15      }
16
17      render() {
18
19      return (
20          <div>
21              <input style={{ display: 'inline' }} type="checkbox"
                ↪  value={this.state.done} onChange={this.handleChange} />
22              <div style={{ display: 'inline', 'textDecoration':
                ↪  (this.state.done ? 'line-through' : '') }}>
23                  {this.state.description}
24              </div>
25          </div>
26      );
27    }
28  }
```

Source Code 1.12: This is Todo component.

1.4 VueJS

Vue.js is an incrementally adoptable ecosystem that scales between a library and a full-featured framework.[14] This simple progressive framework is suitable for building reactive UI. This framework adopts best practice from other frameworks - easy template creation, UI separated to components and has virtual DOM. Vue.js can be used without node.js and any compilation on server-side or client-side. Development web applications using vue.js is much simplier then Angular and React. This framework can be added to existing frameworks incrementally.

Beginner developer who wants to use Vue.js must only know HTML. CSS and JS.

1.4.1 Example of Todos

```
1   @{
2       ViewData["Title"] = "Home Page";
3   }
4
5   <div id='app-root'>Loading...</div>
6
7   @section scripts {
8       <script src="~/dist/main.js" asp-append-version="true"></script>
9   }
```

Source Code 1.13: Main html file.

```
import Vue from 'vue';
import { Component } from 'vue-property-decorator';

@Component({
    components: {
        MenuComponent: require('../navmenu/navmenu.vue.html'),
        TodoComponent: require('../todos/todo.vue.html'),
    }
})
export default class AppComponent extends Vue {
}
```

Source Code 1.14: App component file.

```
import Vue from 'vue';
import { Component, Prop } from 'vue-property-decorator';

@Component
export default class TodoComponent extends Vue {
    @Prop() item: any;
}
```

Source Code 1.15: Definition of todo component.

```
<template>
    <div>
        <input style="display: inline" type="checkbox" :checked="item.done"
        ↪  @change="(t)=>{item.done=t.target.checked}" />
        <div :style="{ display: 'inline', 'textDecoration': (this.item.done ?
        ↪  'line-through' : '') }">
            {{item.description}}
        </div>
    </div>
</template>

<script src="./todo.ts">
</script>
```

Source Code 1.16: Definition template for todo component.

```
import Vue from 'vue';
import { Component } from 'vue-property-decorator';

@Component({
    components: {
        TodoComponent: require("./todo.vue.html")
```

```
    }
})
export default class TodosComponent extends Vue {
    items = [{ description: "Hello", done: false }];
    newItem = { description: "", done: false };

    public addItem(description: string) {
        this.items.push({ description: description, done: false });
        this.newItem.description = "";
        this.newItem.done = false;
    }
    public f(item: any) {
        return !item.done;
    }
}
```

Source Code 1.17: Definition of todos page.

```
<template>
    <div>
        <h1>Todos</h1>

        <div>
            <todo-component :key="item" v-for="item in this.items"
            ↪  :item="item">
            </todo-component>
            <input @keyup.enter="addItem(newItem.description)"
            ↪  v-model="newItem.description">
            <button @click="addItem(newItem.description)">Add</button>
            {{ this.items.filter(f).length }}/{{this.items.length}}
        </div>
    </div>
</template>

<script src="./todos.ts">
</script>
```

Source Code 1.18: Definition template for todos page.

1.5 Summary

Table below 1.2 shows summary of web development frameworks.
Learning curve is a metric which defines how hard is to learn something. In this thesis is the object of learning specific web application development framework. For purpose of this metric were defined three stages of learning:

- Easy - simple to start. This stage can be done in ∼hour.

- Medium - there is a need to install some prerequisites and learn architecture of the framework.

- Hard - there is a need to learn many new features and install prerequisites.

Size of the application is a value which is equal to the quantity of bytes which must be downloaded from the server to the browser while the application is starting. This value is recorded by Chrome browser with disabled cache and cleared all site data. This value is an average from 5 attempts. Transfer compression is not applied on this data. In real application is library and assets cached in browser and only changed data are downloaded.

RAM usage is a metric describing how many bytes are consumed by tab. To measure this value is Chrome browser with task manager used. This value is an average from 5 new tabs.

Browser developer tool shows name of tool to help developer to find bugs in the component or other part of the application.

Github stars is simple metric which shows how many github users like some repository or topic.[6] In the following table 1.2 are repository stars used. History of this metric is shown in the figure 1.2 below.

Another source of data for comparison is last Stack Overflow annual Developer Survey. "Stack Overflow's annual Developer Survey is the largest and most comprehensive survey of people who code around the world. Each year, we field a survey covering everything from developers' favorite technologies to their job preferences."[15]

		Angular	React	Vue.js	Blazor
Learning curve		Hard[1]	Medium	Easy	Medium
Size of application		4.2MB	2.2MB	1.1MB	5.2MB
RAM usage		51MB	39MB	32MB	156MB
Broser developer tool		Angular Augury	React developer Tools	Vue.js devtools	None [2]
Main language		TypeScript	JavaScript	JavaScript	C#
Github stars		47541	128007	137056	7920
Stack overflow survey	**Broadly used**	30.7%	31.3%	15.2%	Unknown[3]
	Loved	57.6%	74.5%	73.6%	Unknown[3]
	Wanted	12.2%	21.5%	16.1%	Unknown[3]
Included in base framework	**Dependency injection**	Yes	No	No	Yes
	Routing	Yes	No	No	Yes
	Template rendering	Yes	Yes	Yes	Yes

[1] is hard because there are many included features which developer must known and use while web application development. According to Fluin (2019) [16] this difficulty may be removed in Angular version 8.

[2] there is Visual Studio and Chrome developer tools but it is not comparable to each other.

[3] value is unknown because there is no data for evaluation

Table 1.2: Summary of compared framework

Chapter 2

Review of WebAssembly and Blazor

2.1 WebAssembly

WebAssembly[17] is designed for writing modern web application. WebAssembly is standardized by World Wide Web Consortium (W3C) and fis supported by all major web browsers. WebAssembly is a compact bytecode instruction format which can be run in web browser. WebAssembly batecode is run in Stack-based virtual machine which is included in web browser. Development of WebAssembly technology started in 2015. First preview version of WebAssembly was released on March 2017. The first release is focused on writing c/c++ source code and compiling to WebAssembly (WASM) byte code. In next WebAssembly release support of Garbage Collection was added. Support of Garbage Collection is necessary for higher programming languages such as Java, `C#` and more. Section 2.1.3 contains simple example of a code which prints "Hello world" into browser debug console. This is the simpliest program that can be written and it shows how WebAssembly works. In section 2.1.4, compilation and combination of many languages to create functional application with web assembly is described.

2.1.1 Use cases

- **Audio video processing**
- **Games**
- **Virtual reality (VR)**
- **Machine learning**
- **Other application where hight performance required**

2.1.2 Pros and cons

Acording to WebAssembly Core Specification there are some pros and cons of WebAssembly.[18]

+ **Speed** of execution code is almost as fast as native code performance.[19]
+ **Safety** of running the code is ensured by a sandboxed virtual machine.
+ **Hardware independence** provides portability to modern architectures and platforms such as desktop, mobile devices and embedded systems.
+ **Platform-independence** provides ability to run in stand-alone Virtual machine (VM), embedded in browsers, or integrated in other environments.
– **Speed of development** web application is much smaller than in JS worse because there is the worst debugging tool and therefore, compilation is needed.
– **Cannot manipulate with DOM** directly from WebAssembly. For DOM manipulation JS Interop must be used.

2.1.3 Hello world example

Simple Hello World application in C and webAssembly. This example can by tested on WebAssembly studio[20].

```
<body>
  <span id="container"></span>
  <script src="./main.js"></script>
</body>
```

Source Code 2.1: This is part of index.html

```
#define WASM_EXPORT __attribute__((visibility("default")))

/* External function which is implemented in JavaScript. */
extern void putstr_js(char* str,char count);

/* simple function which is exported to JavaScript. */
WASM_EXPORT int main(void) {
  putstr_js("hello world",11);
}
```

Source Code 2.2: This is main file for compilation

```
(module
  (type $t0 (func (param i32 i32)))
  (type $t1 (func))
  (type $t2 (func (result i32)))
  (import "env" "putstr_js" (func $putstr_js (type $t0)))
  (func $__wasm_call_ctors (type $t1))
  (func $main (export "main") (type $t2) (result i32)
    i32.const 1024
    i32.const 11
    call $putstr_js
    i32.const 0)
  (table $T0 1 1 anyfunc)
  (memory $memory (export "memory") 2)
  (global $g0 (mut i32) (i32.const 66576))
  (global $__heap_base (export "__heap_base") i32 (i32.const 66576))
  (global $__data_end (export "__data_end") i32 (i32.const 1036))
  (data (i32.const 1024) "hello world\00"))
```

Source Code 2.3: This is compiled main.c into text version WASM

```
let x = '../out/main.wasm';

let instance = null;
let memoryStates = new WeakMap();

let s = "";
fetch(x).then(response =>
  response.arrayBuffer()
).then(bytes =>
  WebAssembly.instantiate(bytes, {
    env: {
      putstr_js: function (c,count) {
        let s=String.fromCharCode.apply(null, new
        ↪  Uint8Array(instance.exports.memory.buffer).slice(c,c+count));
        console.log(s);
      }
    }
  })
).then(results => {
  instance = results.instance;
  document.getElementById("container").textContent = instance.exports.main();
}).catch(console.error);
```

Source Code 2.4: This is JavaScript main.js

2.1.4 WebAssembly compilation process

The source code of the program which is meant to be used in the browser must first be compiled into a WebAssembly. This compiled code is stored in

a binary form that is then downloaded to the browser. It is then linked to the browser using WebAssembly.instantiate shown in source code 2.4.

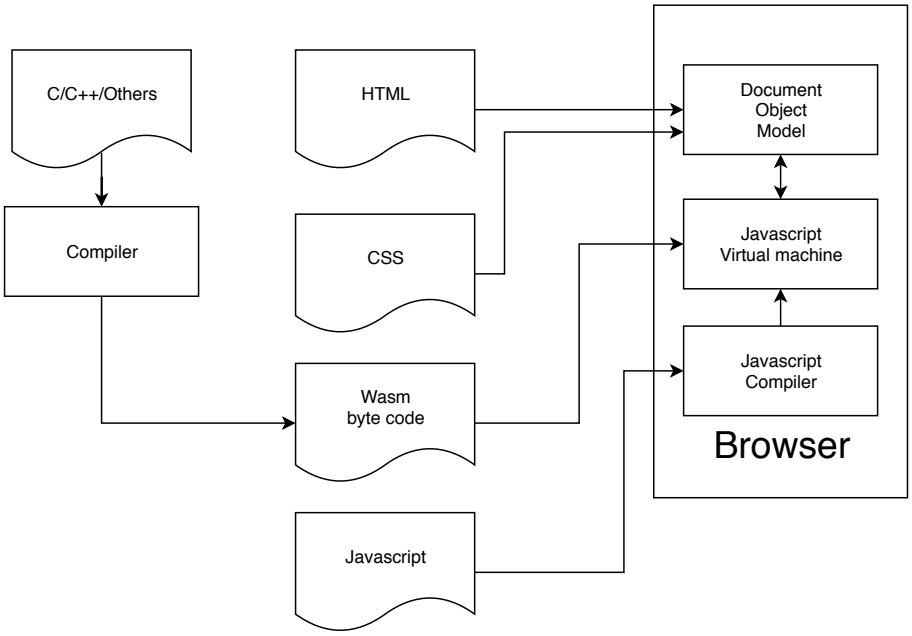


Figure 2.1: Describe combination of many file into one applicatiton page

2.2 Blazor

Blazor is an experimental .NET web framework using `C#`, HTML and Razor that runs on WebAssembly in the browser.[21] This framework is designed and created by Microsoft. Blazor is SPA framework for developing interactive user friendly application. Blazor's name is a combination of Browser and Razor. Razor is Markup Language extension used for web templating. Razor allows to embed `C#` or VB.Net into web pages.

Blazor provides Common Intermediate Language (CIL) runtime to browser. In this runtime, any .Net Standard library can be run such as Newtonsoft.Json (JavaScript Object Notation (JSON) serialize/deserialize library), EPPlus (create or modify Excel spreadsheets) and all other library. [22] Library sent to browser is .Net Standard Dll file it is not recompiled or pre-processed to a different language.

Any dll compiled as .Net Standard library is runtime loadable. This feature provides customization of downloaded size depending on needed components and functionalities. But this feature could be potentially security risk because it is vulnerable to malicious code to client browser injections. This is the same as Cross-site scripting (XSS) but on high level programming language.

Blazor contains all features used by most web application.

- Parameters
- Event handling
- Data binding
- Routing
- Dependency injection
- Layouts
- Templating

Even though this thesis is focused on client-side Blazor technology there is another alternative to run Blazor on server. It is Server-side Blazor and it is described below in section 2.2.4.

Blazor is component-based technology as well as Angular, React and Vue.js. Component is defined in Shared folder and written in cshtml or with Code-Behind. Component file name is used as name of HTML tag.

2.2.1 Example of Todos

In 2.8 was created simple page which contains collection of todo. Todo item class that is provided in todo component which is shown in 2.9 source code. Todos pages has url "/todos" which is defined by **@page** directive.

In file which is shown below in source code 2.5 is defined app component and linked Blazor runtime. This component is the main element of the Blazor application.

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <meta charset="utf-8" />
5      <meta name="viewport" content="width=device-width">
6      <title>BlazorTodos</title>
7      <base href="/" />
8      <link href="css/bootstrap/bootstrap.min.css" rel="stylesheet" />
9      <link href="css/site.css" rel="stylesheet" />
10 </head>
11 <body>
12     <app>Loading...</app>
13
14     <script src="_framework/components.webassembly.js"></script>
15 </body>
16 </html>
```

Source Code 2.5: This is main html file.

App component is simple file containing only router definition. This component contains HTML and may also contain other components which will be placed in all pages in the application.

```
<Router AppAssembly="typeof(Program).Assembly" />
```

Source Code 2.6: App component define only router.

Listing 2.7 contains layout for all pages in Blazor application. This layout is combined with page and index.html into HTML document.

```
@inherits LayoutComponentBase

<div class="sidebar">
    <NavMenu />
</div>

<div class="main">
    <div class="content px-4">
        @Body
    </div>
</div>
```

Source Code 2.7: Default page layout in MainLayout.cshtml

```
@page "/todos"
@using BlazorTodos.Shared
    <div>
        @foreach (var t in todos)
        {
            <todo Value="@t" OnChangeEvent="@(StateHasChanged)"></todo>
        }
        <input type="text" bind="@newDescription" />
        <button onclick="@(()=> AddNewTodo())">Add</button>
        @(todos.Count(t=>!t.Done))/@(todos.Count)
    </div>

@functions{
    List<TodoData> todos = new List<TodoData>();
    string newDescription = "";
    int i = 0;
    protected override void OnInit()
    {
        todos.Add(new TodoData("Test"));
        base.OnInit();
    }
    private void AddNewTodo()
    {
        todos.Add(new TodoData(newDescription));
        newDescription = "";
```

```
26      }
27  }
```

Source Code 2.8: Page contains colection of todo.

```
1   @using BlazorTodos.Shared
2   <div>
3       <input style="display:inline" type="checkbox" bind="@_done"/>
4       <div
        ↪   style="display:inline;text-decoration:@(Value.Done?"line-through":"")">@Value.Description</div
5   </div>
6
7   @functions{
8       private bool _done {
9           get { return Value.Done; }
10          set { Value.Done = value; OnChangeEvent?.Invoke(); }
11      }
12      [Parameter]
13      private TodoData Value { get; set; }
14      [Parameter]
15      private Action OnChangeEvent { get; set; }
16
17
18  }
```

Source Code 2.9: Todo component

At the beginning of year 2019 Blazor framework was moved from separately developed source code to main line of ASP.Net Core. This step suggests that Microsoft believe the potential of Blazor technology.This step prompts that this framework could be the next step in Web development.During April 2019 Microsoft released Blazor as preview.

One programming language used for all purposes from microcontroller (NETMF continued by TinyCLR) to browser client-side application (Blazor).

2.2.2 Pros and cons

[23]

+ **Speed** of execution code is almost as fast as native code performance.
+ **Safety** of the running code is ensured by a sandboxed virtual machine.
+ **Hardware independence** provides portability to modern architectures and platforms such as desktop, mobile devices and embedded systems.
+ **.NET Ecosystem** existing ecosystem of .NET libraries.
+ **Speed of development** web application with existing code is rapid.

– **Size** of downloaded content to the browser is huge. Size is around 6MB.

– **Official state** of this technology is experimental and unsupported.

2.2.3 Blazor client-side

In client-side Blazor whole web application is running in browser. The following figure 2.2 shows the sequence of browser communication with Blazor and server. Architecture of client-side Blazor is described in figure 2.3 below. This architecture is based on running .Net Application on Mono Common Language Runtime (CLR). Mono is open-source project providing CLR to Linux and other platforms including Microsoft Windows.[24] Now is Mono developed by Xamarin(a Microsoft subsidiary) and .Net foundation. Original author of Mono project is Ximian.[25] Mono CLR is compiled to WASM.

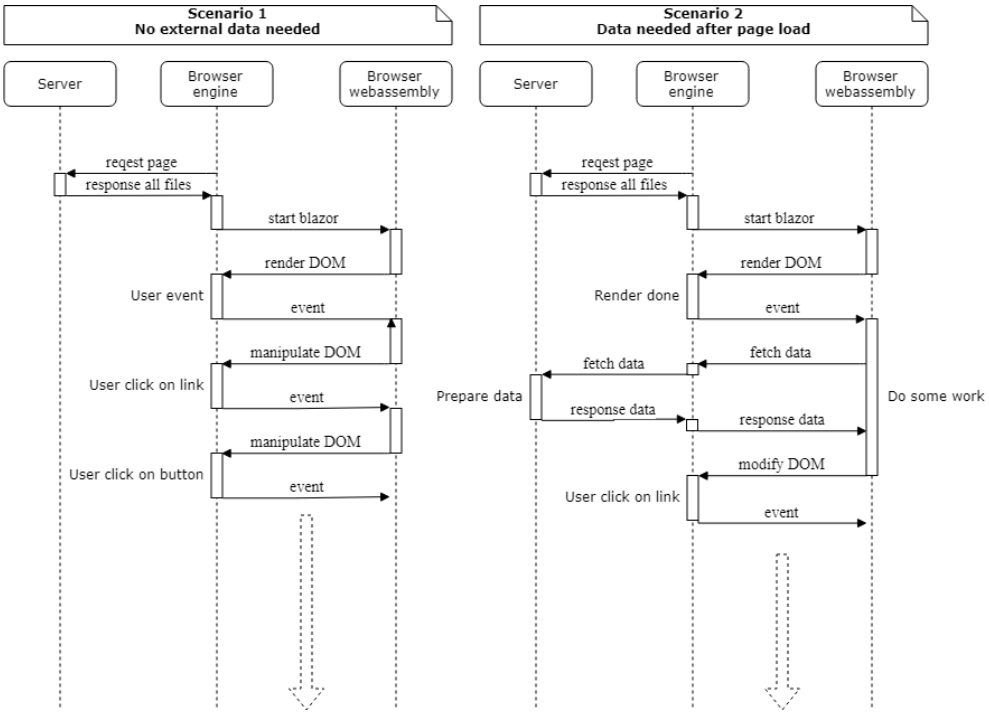


Figure 2.2: Timing of client-side Blazor.

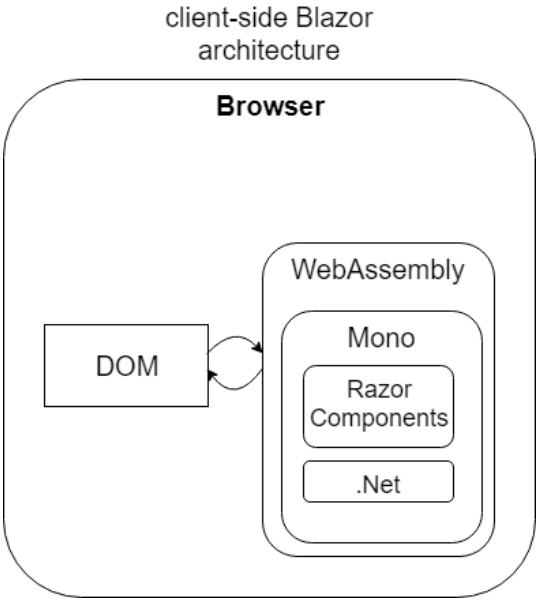


Figure 2.3: Architecture of client-side Blazor.

First scenario shows application which is downloaded to browser and run offline. There is only one contact with server when the server downloads the application. After that it can be stored in Random access memory (RAM) and run without network connection such as Timer application, Clock and many more.

Second scenario describes regular client-side application with some requests to server. In this scenario asynchronous computing is shown during waiting for data from server. Network communication is provided by browser engine.

Client-side Blazor can modify DOM or invoke JS function at any time. Client-side Blazor provides many benefits:

- Processing on client side reduces the server load.
- No server-side .Net is required.
- Can be run offline without internet connection.
- Graphical user interface (GUI) response is faster in comparison to reloading whole page.

2.2.4 Blazor server-side

During its development, Blazor was known as Razor components. It is similar to client-side version with only a few differences. First difference is that WASM is not required. All DOM manipulation is invoked from server. Same

Blazor application can be run either client-side or server-side. The browser notifies Blazor with every event which is fired in browser trough SignalR connection.[26] SignalR is a technology useful for creating stable connection between server and client. SignalR tries to use many protocols to establish the connection.[27] Appropriate timing diagram and architecture is shown below in figure 2.4 and 2.5

Server-side Blazor is not working without stable connection. Server can control browser over SignalR during established connection. When the connection losts client-script of SignalR it tries to reconnect. The event **onclose** is called after four attempts and connection is then lost. The developer must predict this situation and application must be prepared for this situation.

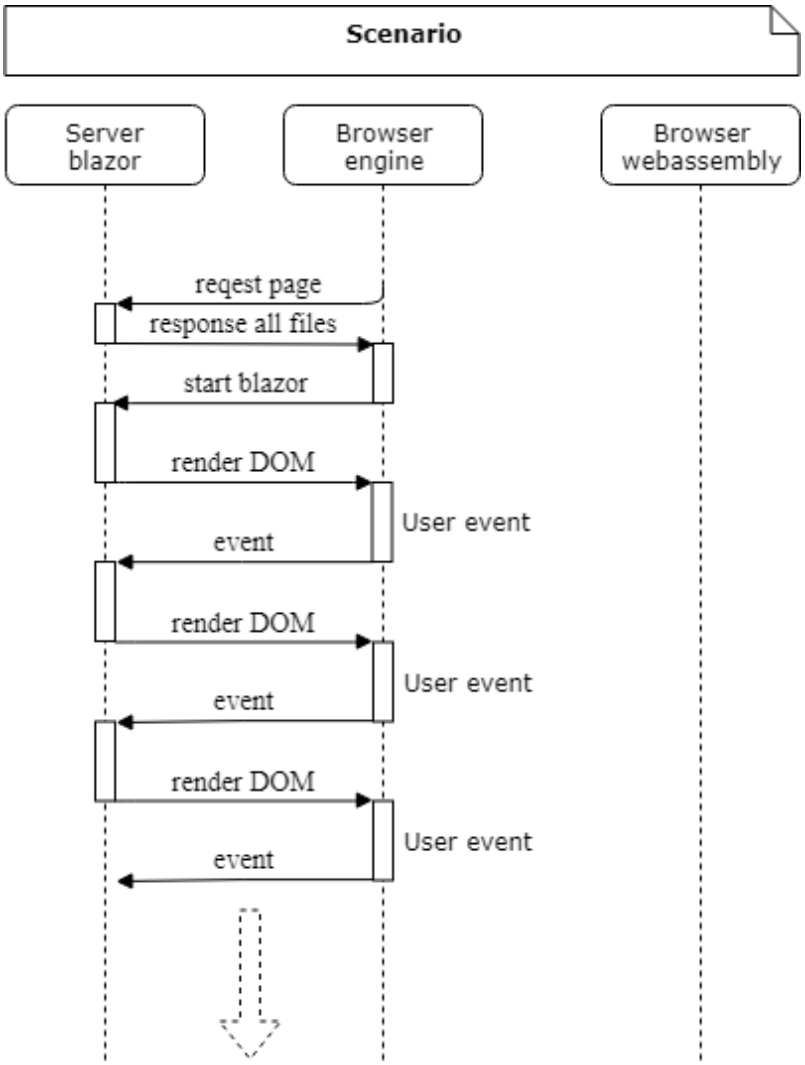


Figure 2.4: Timing of server-side Blazor.

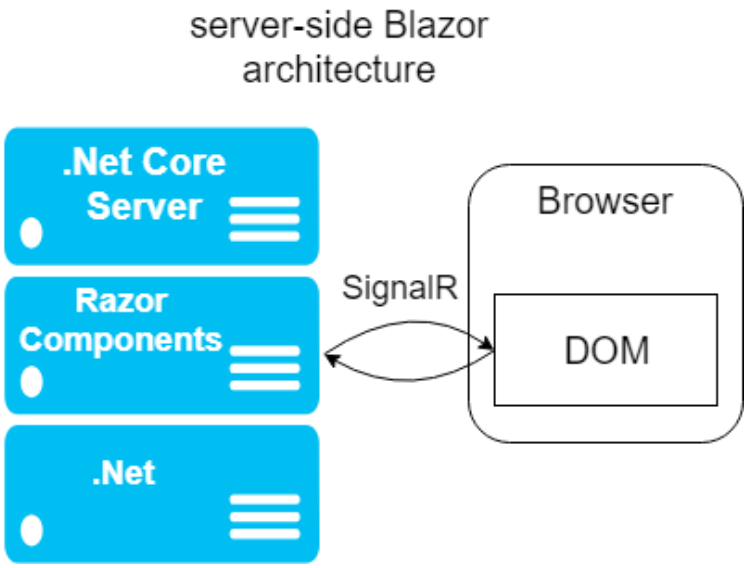


Figure 2.5: Architecture of server-side Blazor.

Chapter 3

SPA, MVC and Pages comparison

According to Microsoft documentation website ([28]) developers should use SPA architecture when:

- Your application must expose a rich user interface with many features.
- Your team is familiar with JavaScript and/or TypeScript development.
- Your application must already expose an API for other (internal or public) clients.

Or developers should use traditional web applications when:

- Your application's client-side requirements are simple or even read-only.
- Your application needs to function in browsers without JavaScript support.
- Your team is unfamiliar with JavaScript or TypeScript development techniques.

3.1 SPA

Single Page Application is modern architecture providing ability to create user friendly environment. In SPA architecture the whole application is loaded during the start. According to technology used in the application all styles,template and function can be downloaded and cached. After this SPA can run in offline mode. When application needs more data it connects to Application Programming Interface (API) on server and get it.

3.1.1 Pros and cons

+ **Fast** reaction on user input
+ **Fat client** reduce server load
+ **Native** feeling from application
– **Requires broad knowledge** before starting the development of application

3.2 MVC

Model-View-Controller architecture provide technology to develop large web applications. This architecture separates application into three parts. Model part contains business logic which means entities and their logic. Second part is View. View contains template and code providing render functionality. And last part is the controller which connects it all together. Every request is routed through routing process to action in controller. The controller is selected according to url in the request. Action is a method that contains data retrieving, processing and invoke rendering (View or data serialization).

3.2.1 Pros and cons

+ **Resource oriented** architecture - every controller represents one resource
+ **Complex** routing for this architecture
+ **Better testability** of each part
– **Cpu load** of server is high because every request must render page

3.3 Pages

Pages are simple web architecture. Page represents one screen of website. In most cases page has only one route defined as relative path to root of web folder. Custom routing in pages architecture is not necessary. Page usually contains a template and logic in one file. There are many programming languages focused on pages architecture such as PHP, Perl, `C#` (Web Forms and now Razor pages) and many more. This architecture is the best approach for small application which is read only or has simple input form. This architecture can be used as base for SPA. This architecture is not very suitable for web API application.

3.3.1 Pros and cons

+ **Simple** routing for this architecture
+ **Easy** understanding
+ **Fast** for creating simple GUI
− **Cpu load** of server is same as in MVC application

3.4 Summary

	SPA	**MVC**	**Pages**
Server CPU load	Low	High	High
Client CPU load	High	Low	Low
Knowledge requirements[1]	High	Medium	Low
Representative	Angular React+Redux Blazor Vue.js	Asp.Net MVC PHP Nette Ruby On Rails	Pure PHP Asp.Net WebForms Asp.Net Razor Pages

[1] Quantity of additional knowledge required

Table 3.1: Summary of architecture

Chapter 4

Proof of Concept - Blazor WebAssembly Classbook

4.1 Assignment

Aim of this chapter is to create replacement of application which is the content of my Bachelor thesis [2]. Application in the Bachelor thesis is written in Java for Android. This application cannot be run on Windows, Linux or other mobile or desktop Operating system (OS). This proof of concept shows the way how to create multiplatform application in simple programming language when using modern web trends.

The application will be a replacement for Classbook developed in my bachelor thesis[2]. The Classbook is the application which supports daily agenda in a specific high school in Hradec Králové. According to complexity of processes in the school there are many requirements on specific functionalities. This application must contain functionalities for timetable, classification and student administration. This application will be designed for teacher. The teacher needs to have simple, fast and easy to use tool to fill the attendance form with ability to add delay to those students who will arrive late to the lesson. Then the teacher needs to be able to add description of the lesson which is another requirement on the application. The last function of this application should be student classification so the teacher should be able to administrate marks in appropriate class in appropriate subject.

4.2 Implementation

The whole application is developed and implemented on Microsoft technologies. Application is divided into four parts that together create a functional unit.

The first part is GUI implemented with client-side Blazor. GUI commu-

nicates with API trough Hypertext Transfer Protocol (HTTP)(S) communication. After application is downloaded to browser all communication is realized over JSON serialization.

API part is developed as ASP.NET MVC Core application that provides Create, read, update and delete (CRUD) operation from database to GUI.

Database part is implemented in Microsoft SQL Server.

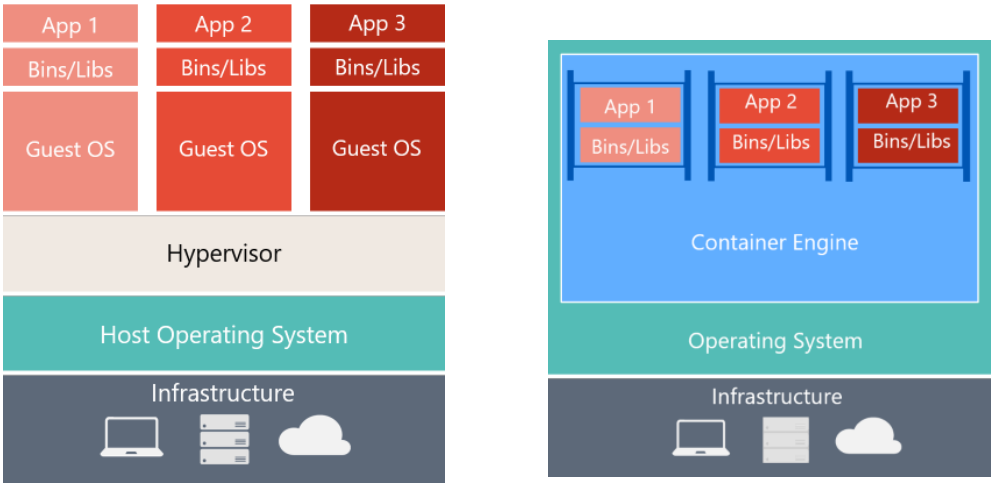
The last part is communication with Identity provider (IDP).This communication is mandatory because this application is primarily designed for High school and college of applied cybernetics Ltd.[29] which has and uses the IDP. Each part of application is described in detail below. Even though school's main language is Czech, native English speakers are also teaching there and they use the application as well, so it must contain multi-language support.

4.2.1 Docker

Docker is a tool to automate the deployment of applications as portable and standalone containers that can be run in the cloud or locally. This tool provide lightweight virtualization. The comparison of Docker and standard VM architecture is visible from following figures 4.1a and 4.1b. Docker container may represent one application or service which is started standalone. Every Docker container is created from Docker image with additional configuration such as mounted volume, network communication and resource limit(Central processing unit (CPU), Memory). Docker image contains library and binary file to run the application. Docker images are created by Dockerfile that defines how to combine library and binary file. It is step by step cookbook for creating the image. This thesis includes Dockerfile for creating Docker image for all part without IDP.

There are some additional docker images for production use of this thesis which are not included in this thesis such as firewall, revers proxy and Domain name service (DNS). This services are also running in Docker.

All parts of this application that need to run on the server run in the Docker as container.



(a) Standard architecture of Virtual Machines

(b) Docker architecture

Figure 4.1: Comparison of Docker architecture and VM architecture. (source: Microsoft Docs website[1])

4.2.2 GUI

GUI is developed in project InformacniSystemCore.Blazor.Client that is part of InformacniSystemCore.Blazor solution. In this solution is InformacniSystemCore.Blazor.Server that provide simple http server for providing Blazor.Client app to browser. The whole application GUI is optimalized for mobiles because this application may replace previous mobile application. Due to the size of the application, there is a need of initialization of CLR so there was add a simple loading screen into index.html(source code 4.1). This loading element will be replaced after application is loaded.

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <meta charset="utf-8" />
5      <meta name="viewport" content="width=device-width">
6      <title>InformacniSystemCore.Blazor</title>
7      <base href="/" />
8      <!-- <link href="css/bootstrap/bootstrap.min.css" rel="stylesheet" /> -->
9      <link href="css/site.css" rel="stylesheet" />
10 </head>
11 <body>
12     <app>
13         <div class="splash-screen">
14             <img src="images/logo.png" />
15                 <div class="loading-text">
16                     Načítání
17                 </div>
```

```
18                  <div class="dots">
19                      <span style="animation-delay: 0s;">*</span>
20                      <span style="animation-delay: 0.25s;">*</span>
21                      <span style="animation-delay: 0.5s;">*</span>
22                      <span style="animation-delay: 0.75s;">*</span>
23                      <span style="animation-delay: 1s;">*</span>
24                      <span style="animation-delay: 1.25s;">*</span>
25                      <span style="animation-delay: 1.5s;">*</span>
26                      <span style="animation-delay: 1.75s;">*</span>
27                  </div>
28          </div>
29      </app>
30
31      <script src="_framework/components.webassembly.js"></script>
32      <script>
33          window.getOrientation = () => {
34              return screen.orientation.type;
35          }
36          window.addEventListener("orientationchange", (e, d) => {
37              DotNet.invokeMethod("InformacniSystemCore.Blazor.Client",
                ↪  "OrientationChanged", screen.orientation.type);
38          });
39      </script>
40  </body>
41  </html>
```

Source Code 4.1: Index html with splash screen

Content of App component is replaced with MainLayout shown in source code 4.2. This layout contains component NavMenu which includes navbar(top menu) and drawer(side menu). Next what the MainLayout include is **@Body**. **@Body** is RenderFragment defined in LayoutComponentBase and represents content of displayed page. RenderFragment is a delegate that is invoked by each page refresh. Last part included in MainLayout is component Dialogs. Dialogs component shown in (source 4.3) represent any displayed dialog window. NavMenu contains dynamic part that can be replaced by page. For example timetable page replaces dynamic part with date selector or search input.

```
@inherits LayoutComponentBase
<div class="sidebar">
    <NavMenu />
</div>

<div class="main">
    <div class="content">
        @Body
    </div>
</div>
<Dialogs></Dialogs>
```

Source Code 4.2: MainLayout of this application

```
@inject DialogService dialog
<div class="dialogs-container" style="@dialogStyles" onclick="@(()=> {
↪  dialog.Hide(); })">
    <div class="backdrop"></div>
    @dialog.CurrentDialogRender
</div>

@functions{
    public string dialogStyles { get; set; } = "display:none";
    protected override void OnInit()
    {
        base.OnInit();
        dialog.ShownChanged += (s, e) =>
        {
            Console.WriteLine(e);
            dialogStyles = (!e ? "display:none" : "");
            StateHasChanged();
        };
    }
}
```

Source Code 4.3: Dialogs components

During the first rendering, the user account validity is checked. When the user account is not valid, user will be redirected to login through IDP. When the user account is valid user will be redirected to \timetable. Timetable page contains lessons for day or lessons for week depending on mobile orientation. Orientation of device cannot be obtained from Blazor so there is a need to use JavaScript Interop.

Another problem that has occurred is animation events. The first attempt to solve the problem was using "ontransitionend" event but this was still recognized as error by intelliSense. According to website: "IntelliSense is a code-completion aid that includes a number of features: List Members, Parameter Info, Quick Info, and Complete Word. These features help developers to learn more about the code they are using, keep track of the parameters

they are typing, and add calls to properties and methods with only a few keystrokes." [30] Fortunately, this error does not prevent compilation. Using this event, an animation is created to open the side menu.

Drawer component shown in source code 4.4 is an interesting component representative because it contains both C# logic and HTML template with razor directives. In this component dependency injection through **@inject** directive is used.

```
1  @inject Microsoft.AspNetCore.Components.Services.IUriHelper helper
2
3  <div ontransitionend="@transitionEnd">
4      <div class="backdrop @State.ToString().ToLower()" onclick="@Toggle"></div>
5      <div class="drawer @State.ToString().ToLower()">
6          <div class="logo">
7              <img src="/images/kybernalogo.png" />
8          </div>
9          <div class="line"> </div>
10         <AccountInfo></AccountInfo>
11         <div class="menu">
12             <Menu Item="@menu"></Menu>
13         </div>
14     </div>
15 </div>
16 @functions{
17     [Parameter]
18     private DrawerState State { get; set; } = DrawerState.Closed;
19
20     protected override void OnInit()
21     {
22         helper.OnLocationChanged += locationChanged;
23     }
24     public void locationChanged(object sender,string newlocation)
25     {
26         Console.WriteLine("Ahoj svete");
27         if(State!=DrawerState.Closed)
28             State = DrawerState.Closing;
29         StateHasChanged();
30     }
31
32     private Menu.MenuItem menu = new Menu.MenuItem("", "","")
33     {
34         Items = new List<Menu.MenuItem>()
35     {
36             new Menu.MenuItem("timetable.svg","Timetable","/timetable"),
37             new Menu.MenuItem("test.svg","Marks","/fetchdata"),
38             new Menu.MenuItem("study.svg","Students","/students"),
39         }
40     };
41
42     private void transitionEnd()
43     {
44         if (State == DrawerState.Opening) State = DrawerState.Opened;
```

```
45          if (State == DrawerState.Closing) State = DrawerState.Closed;
46      }
47
48      public void Toggle()
49      {
50          if (State == DrawerState.Closed || State == DrawerState.Closing)
51              State = DrawerState.Opening;
52          else
53              State = DrawerState.Closing;
54          StateHasChanged();
55      }
56
57      enum DrawerState
58      {
59          Opening,
60          Opened,
61          Closing,
62          Closed
63      }
64  }
```

Source Code 4.4: Drawer components

AccountInfo is component that is used to show state of Account such as Full name and Role. In this component, my resource manager for multi-language support is also used .

MyResourceManager is based on standard ResourceManager but there is an unknown problem with multiple language. When is created multiple Resource file but only one is compiled. I resolve this problem with little trick. I customize file naming of resource. For my resource manage is pattern name.resx for default and name_ISO Language Code.resx for "ISO Language Code Table". In this application there is Base.resx with Czech language resource and Base_en-US.resx with English language resource.

Timetable page shown in source code 4.5 is used TimetableDay component that receive parameter Day. TimetableDay component represent one day of week and contains all lesson for this day and logged teacher. Min and Max parameter define first and last hour shown in day. This parameter binding is one way of binding which means only data from Timetable component is sent to TimetableDay component. Blazor.Server project is Asp.Net Core application compiled and runned in Docker container.

```
1   @page "/timetable"
2   @inject DeviceState state
3   @inject ApplicationState AppState
4   <div class="timetable">
5       <div class="date">
6       </div>
7       <div class="days">
8           @if (Orientation == Orientation.Landscape)
```

```
9           {
10              @for (int i = 0; i < 5; i++)
11              {
12                  <TimetableDay Day="@StartOfWeek.AddDays(i+1)" Min="@Min"
                     ↪  Max="@Max"></TimetableDay>
13              }
14          }
15          else
16          {
17              <TimetableDay Day="@SelectedDate" Min="@Min"
                 ↪  Max="@Max"></TimetableDay>
18          }
19      </div>
20  </div>
21  @functions{
22
23
24      private Orientation Orientation { get; set; }
25      private DateTime SelectedDate { get; set; } = DateTime.Now.Date;
26      private DateTime StartOfWeek =>
         ↪  SelectedDate.AddDays(-(int)SelectedDate.DayOfWeek);
27      private int Min { get; set; } = 7;
28      private int Max { get; set; } = 19;
29      protected override void OnInit()
30      {
31          Orientation = state.DeviceOrientation;
32          base.OnInit();
33          state.OnOrientationChanged += OrientationChanged;
34          AppState.DynamicActionBar = (t) =>
35          {
36              var seq = 0;
37              t.OpenElement(seq++, "div");
38              t.AddAttribute(seq++, "class", "date-info");
39              t.AddAttribute(seq++, "onclick", () =>
40              {
41                  Console.WriteLine("OpenDialog");
42              });
43              t.AddMarkupContent(seq++,"  <div>" +
44                  "    <div>" +
45                  "      "+SelectedDate.ToString("dddd")+"" +
46                  "    </div>" +
47                  "    <div>" +
48                  "      "+SelectedDate.ToString("d.M.yyyy") +
49                  "    </div>" +
50                  "  </div>" +
51                  "  <div>" +
52                  "    */" +
53                  "  </div>");
54              t.CloseElement();
55
56              t.OpenElement(seq++,"a");
57              t.AddAttribute(seq++, "class", "prev");
58              t.AddAttribute(seq++, "onclick",
                 ↪  BindMethods.GetEventHandlerValue<UIMouseEventArgs>(
```

```
59                  ()=> {
60                  Console.WriteLine("tmp");
61                  prev();
62              }));
63              t.AddContent(seq++, "<");
64              t.CloseElement();
65
66              t.OpenElement(seq++,"a");
67              t.AddAttribute(seq++, "class", "next");
68              t.AddAttribute(seq++, "onclick",
69                  BindMethods.GetEventHandlerValue<UIMouseEventArgs>(next));
70              t.AddContent(seq++, ">");
71              t.CloseElement();
72
73          };
74      }
75      private void OrientationChanged(object sender, Orientation newOrientation)
76      {
77          this.Orientation = newOrientation;
78          Console.WriteLine(Orientation);
79          StateHasChanged();
80      }
81      private void prev()
82      {
83          SelectedDate = SelectedDate.AddDays(-1);
84          StateHasChanged();
85      }
86      private void next()
87      {
88          SelectedDate = SelectedDate.AddDays(1);
89          StateHasChanged();
90
91      }
92  }
```

Source Code 4.5: Timetable components

4.2.3 API

Because I want to provide access to API for student. I create API completely separated from Blazor application. API is created with ASP.NET MVC Core technology and there are public documentation for this api generated by Swegger library.

For authentication and authorization to API is used JSON Web Token (JWT) and IDP. IDP is used only for action generating token. JWT is used for other action. Client must refresh token after token expires. Token expiration is set to 5minutes. Refresh token can be invoked when expiration time is in 45 minutes from now.

I choose simple CRUD operation on this API documented by Swegger. There are many generators that can create client for API documented by

Swegger. NSwagStudio(figure 4.2) can generate client for TypeScript and `C#`. I choose this generator because it can create client based on HttpClient for Swagger API. HttpClient is injected to generated client trough constructor. Api is running in Docker container based on Microsoft/dotnet image.

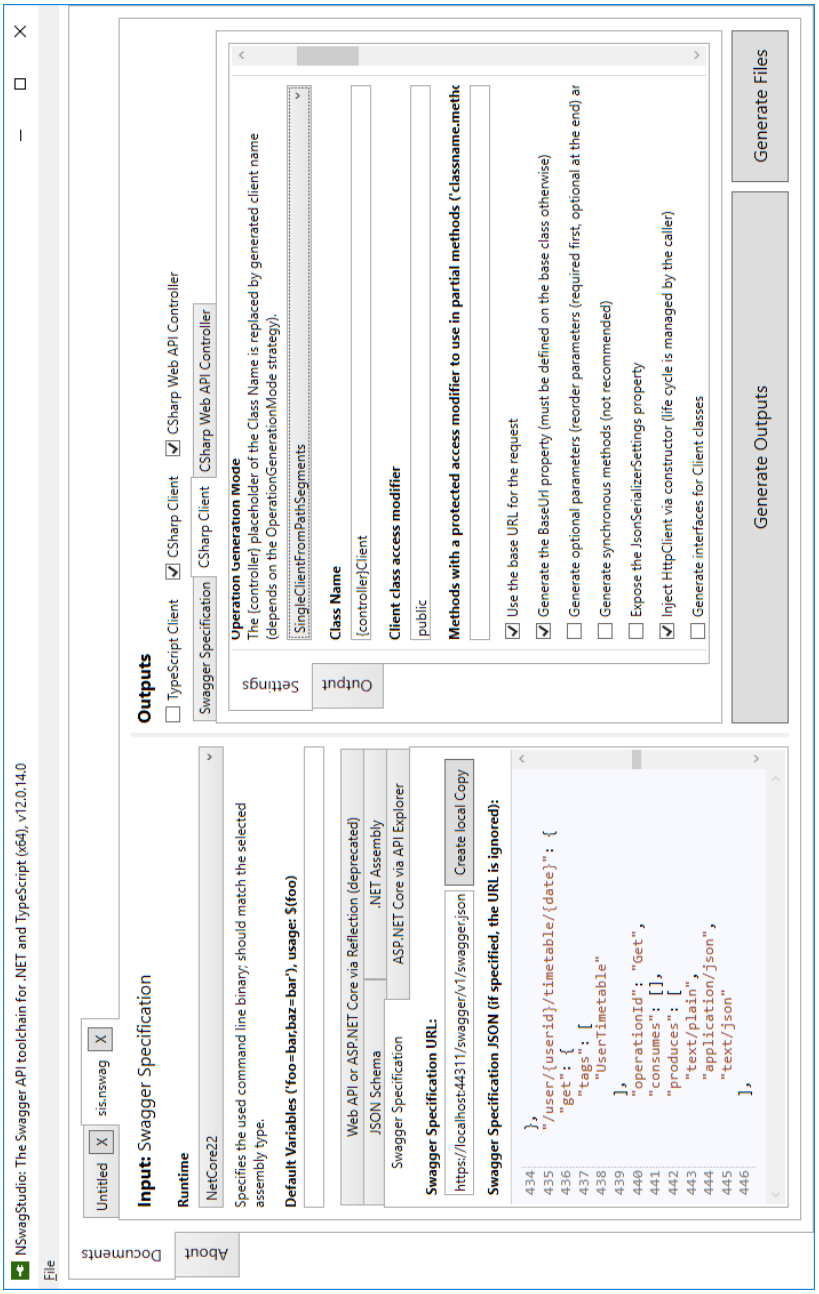


Figure 4.2: Screnshot of NSwagStudio

4.2.4 Database

Database for this application is combination of ASP.Net identity database and My custom design. This custom design is created using database first methods. As database engine Microsoft SQL Server is used. In my architecture I used Microsoft SQL Server 14.0.3035.2 running on Ubuntu GNU/Linux. This server is running in Docker container that provides portability and better maintenance.

Connection to database is done through Entity framework (EF). EF is Object relational mapping (ORM) that provide access to database via .Net objects.[31] Base object is called Context and provide Structured query language (SQL) connection. In context class is referenced entity class where one entity class represent one database table.

4.2.5 IDP

IDP is a service that provides user authentication. This service provides Single sign on (SSO) for user. For communication with IDP is used Security assertion markup language (SAML). SAML is standadized protocol for SSO. In this application Shibboleth IDP is used as server and as client it is library Sustainsys.Saml2.AspNetCore2.

4.2.6 Features of Blazor

In this proof of concept, some features are added even though, they do not solve the assignment but they are at least interesting. In page /ref is example of using reflection in Blazor. In solution InformacniSystemCore.Blazor is created project TestReflection that contains logic compiled in library. This TestReflection library is downloaded to Blazor application and loaded. With class Activator is created instance of Ref class. This feature is shown in source code 4.6.

```
1  @page "/ref"
2  @inject HttpClient http
3
4  <h1>Reflection</h1>
5  @functions{
6
7      protected override async Task OnInitAsync()
8      {
9          var ns = await
           ↪  http.GetByteArrayAsync("http://localhost:63458/_framework/netstandard.dll");
10         AppDomain.CurrentDomain.Load(ns);
11         var data = await
           ↪  http.GetByteArrayAsync("http://localhost:63458/_framework/_bin/TestReflection.dll");
12         //var assembly=System.Reflection.Assembly.Load(data);
13
```

```
14          var assembly=AppDomain.CurrentDomain.Load(data);
15          Console.WriteLine(assembly.FullName);
16          Console.WriteLine("loaded");
17          var type = assembly.GetTypes().First(t => t.Name.Contains("Ref"));
18          Console.WriteLine("type founded");
19
20          Console.WriteLine(type.Name);
21          var r= Activator.CreateInstance(type, true);
22          Console.WriteLine("Instance created");
23
24          Console.WriteLine(type.GetProperty("Name").GetValue(r));
25      }
26  }
```

Source Code 4.6: Reflection page

4.3 Screenshots

Design proposal for this application is based on output of the project which was part of course User Interface Design at Czech Technical University in Prague.

(a) Portrait timetable screenshot

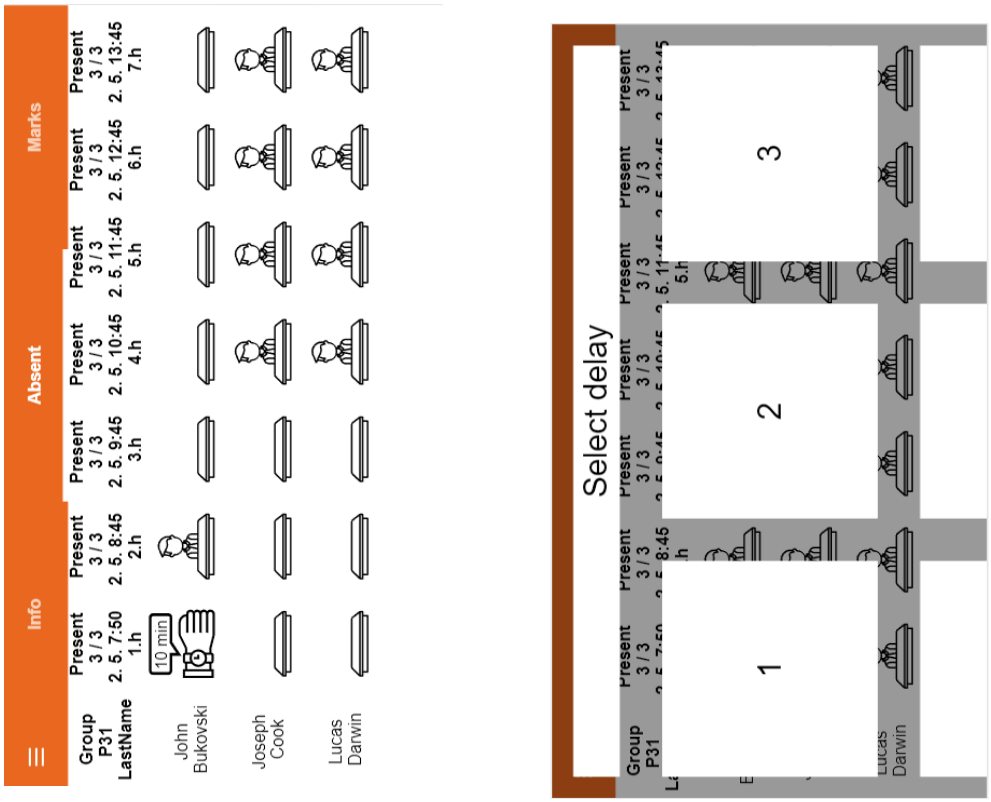
(b) Landscape timetable screenshot

Figure 4.3: Screenshot of timetable

(a) Portrait drawer

(b) Landscape drawer

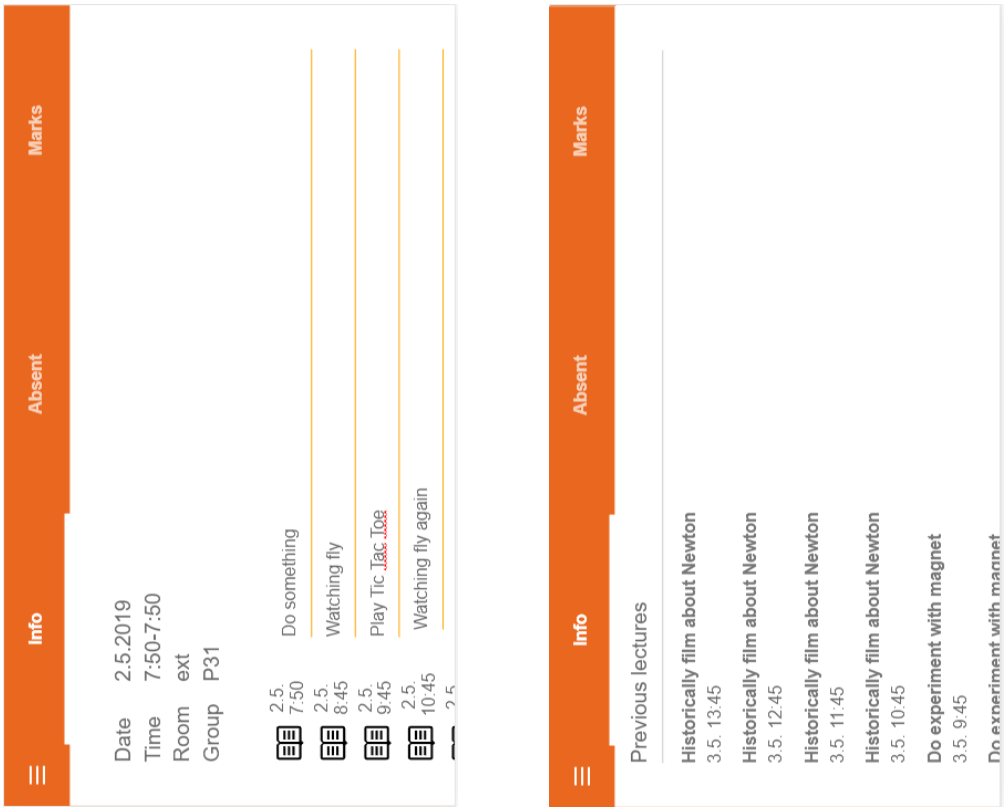
Figure 4.4: Screenshot of drawer



(a) Landscape student attendance screen

(b) Landscape student delay dialog

Figure 4.5: Screenshot of attendance



(a) Portrait lesson info screen (b) Portrait lesson info screen

Figure 4.6: Screenshot of lesson info

4.4 Diagrams

4.4.1 Database

In figure 4.8, there is visualisation of database entities. This database is based on database from bachelor thesis shown in figure 4.7 but there are many improvements. In this application, small subset of all entities is used . This subset is shown in figure 4.9.

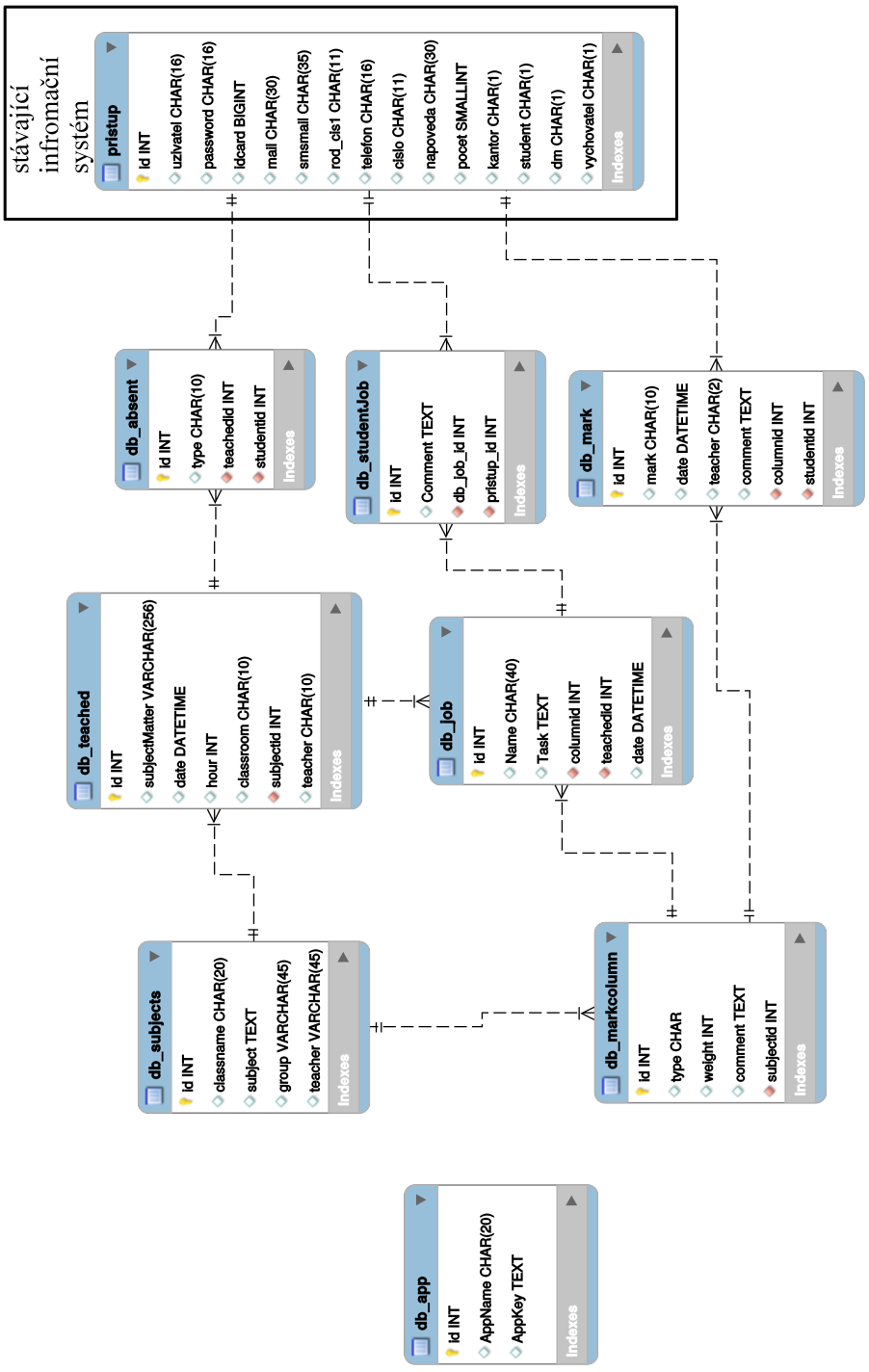


Figure 4.7: Old database from bachelor thesis[2]

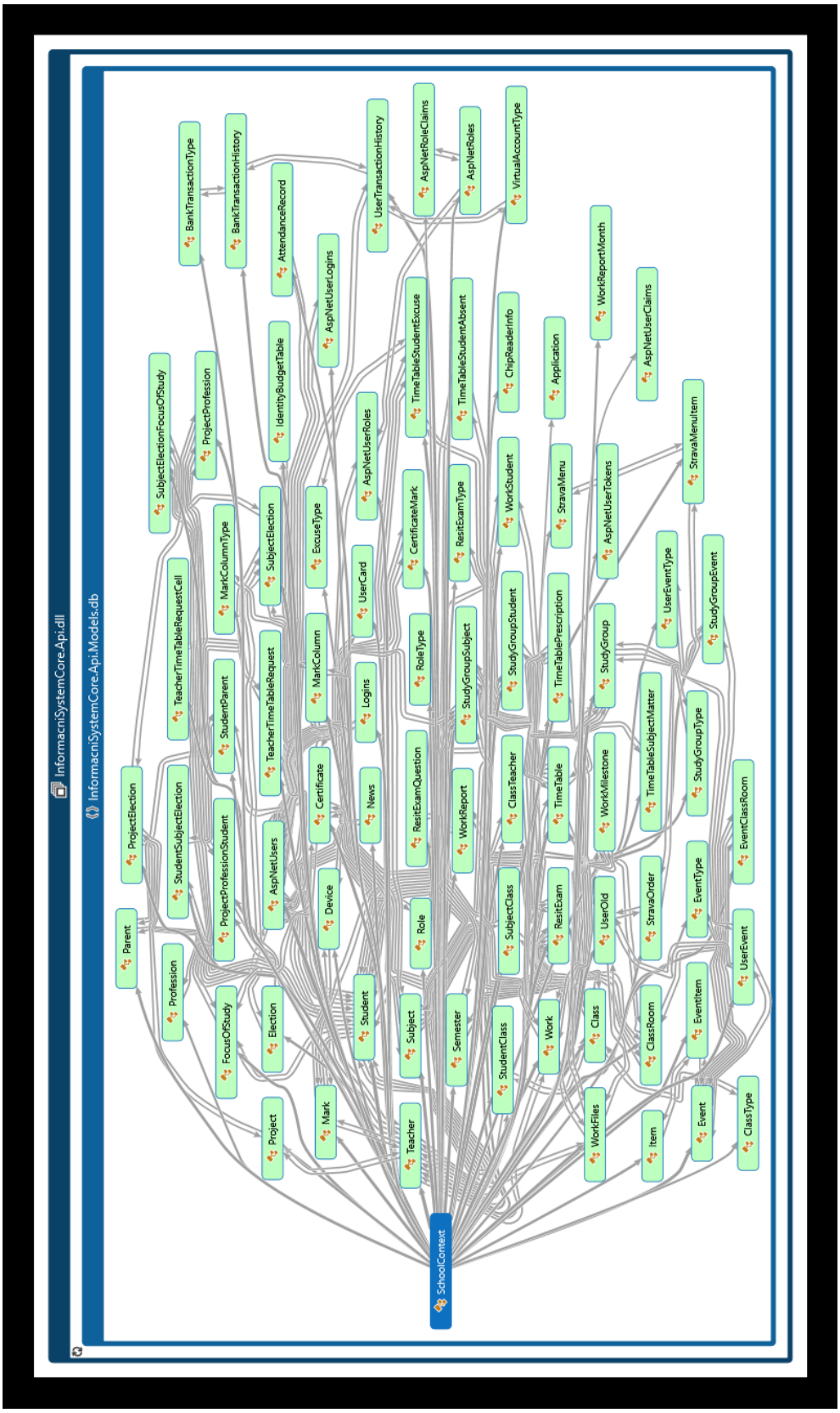


Figure 4.8: Entities in EF

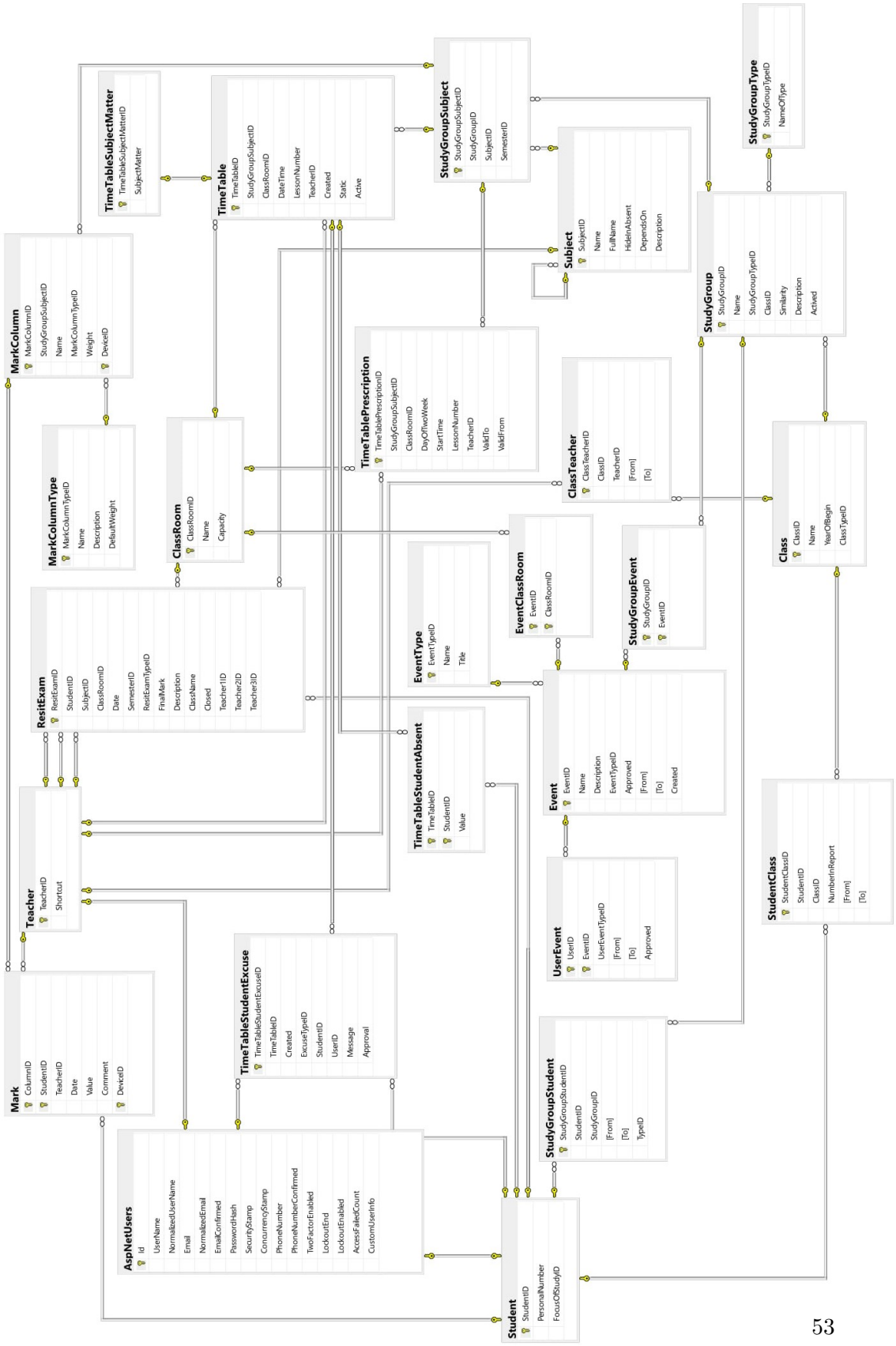


Figure 4.9: Database diagram

4.4.2 User flow diagram

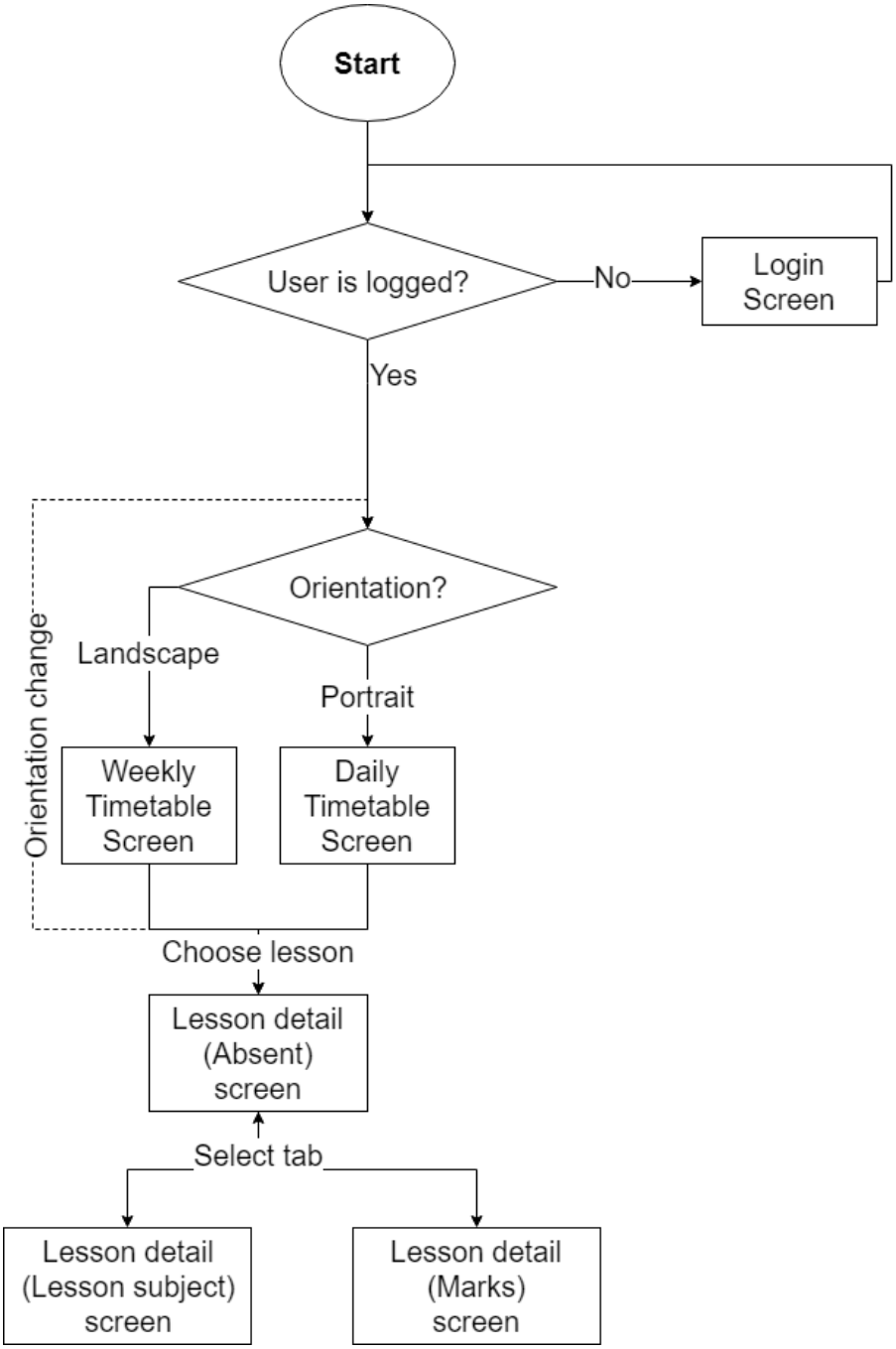


Figure 4.10: User flow diagram

4.5 Testing

For testing, I chose two groups of people to come in contact with Blazor. The first group is teachers who use the original application. Because Blazor WebAssembly Classbook does not yet have the potential to fully replace the original mobile application, user testing is primarily focused on the user experience in this application compared to the mobile application. The second addressed group are programmers who already have some experience with this framework and have experience with some of the other frameworks mentioned in this work. For this group, the questionnaire is focused on creating an application in Blazor. Most of these people are closely connected to High school and College of Applied Cybernetics Ldt and are fully informed about Blazor technology and its progress. These two groups then answered the final series of questions about the comparison of Blazor with other web pages.

Application and Blazor framework test result were collected using Google Forms.

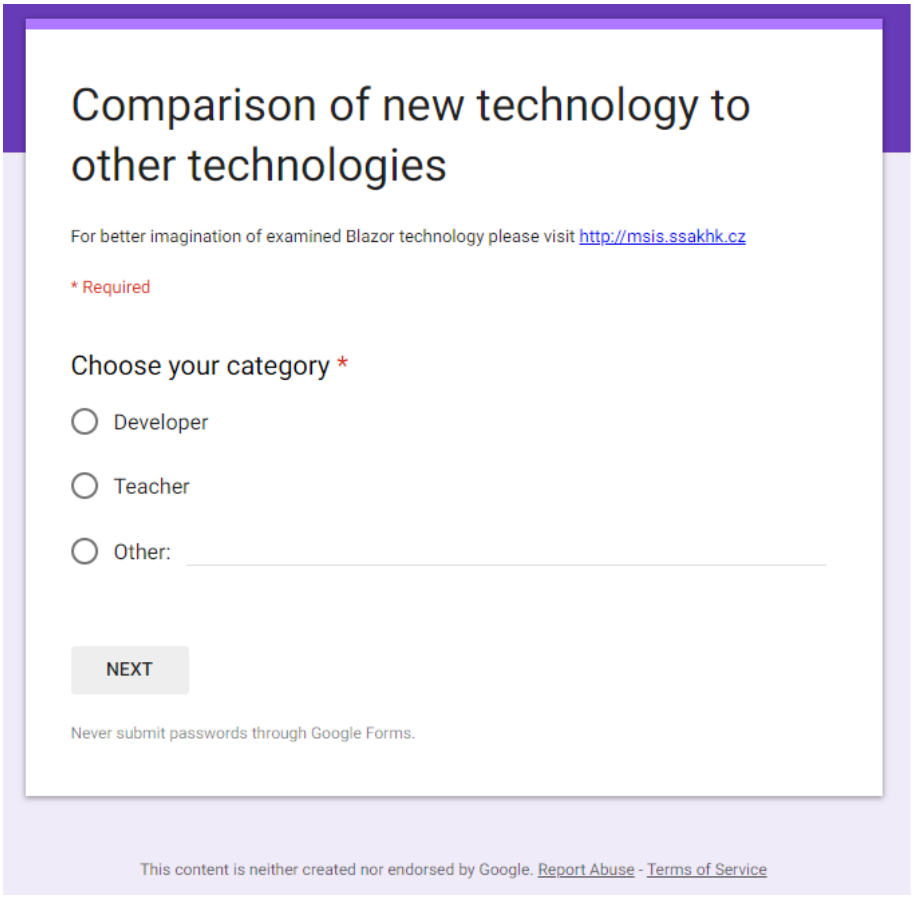


Figure 4.11: Google forms questionnaire - first part

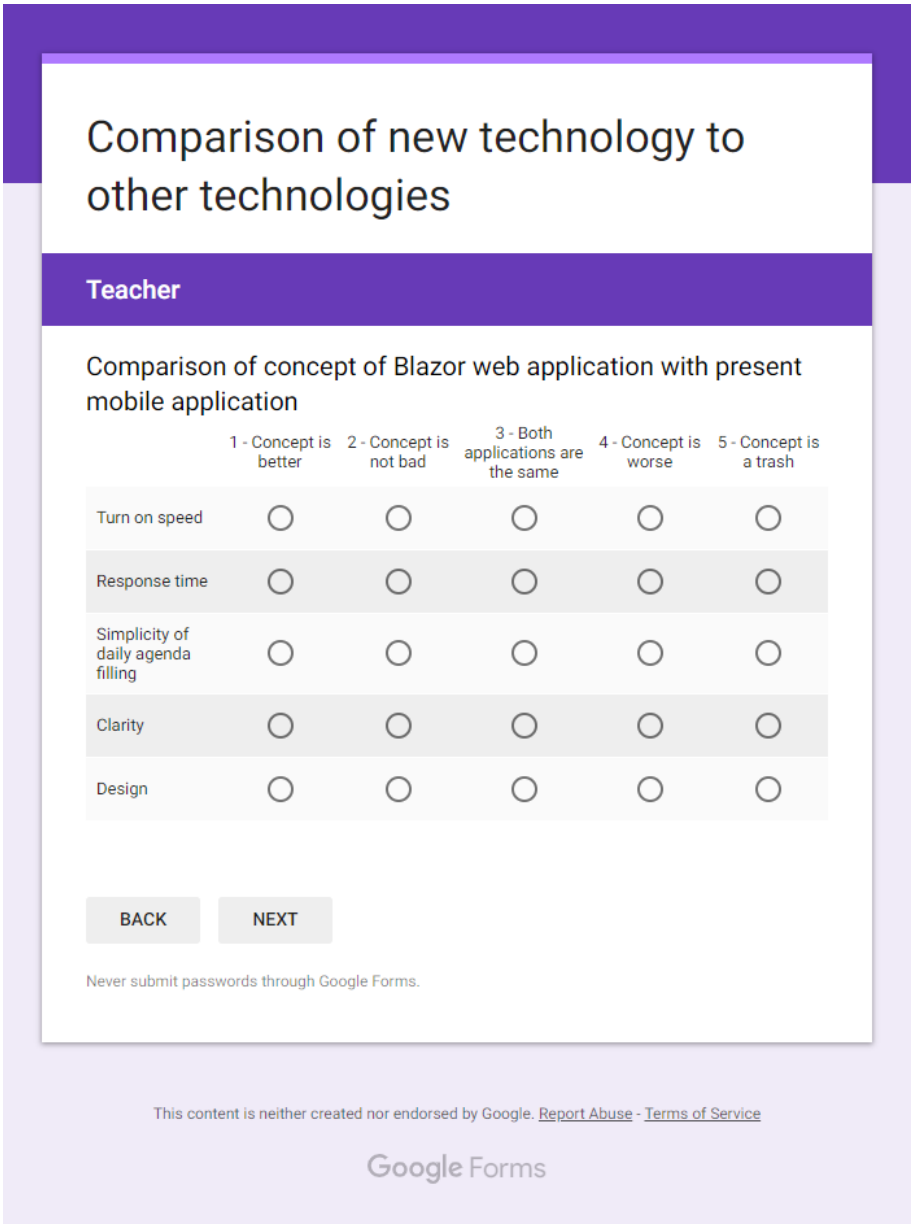
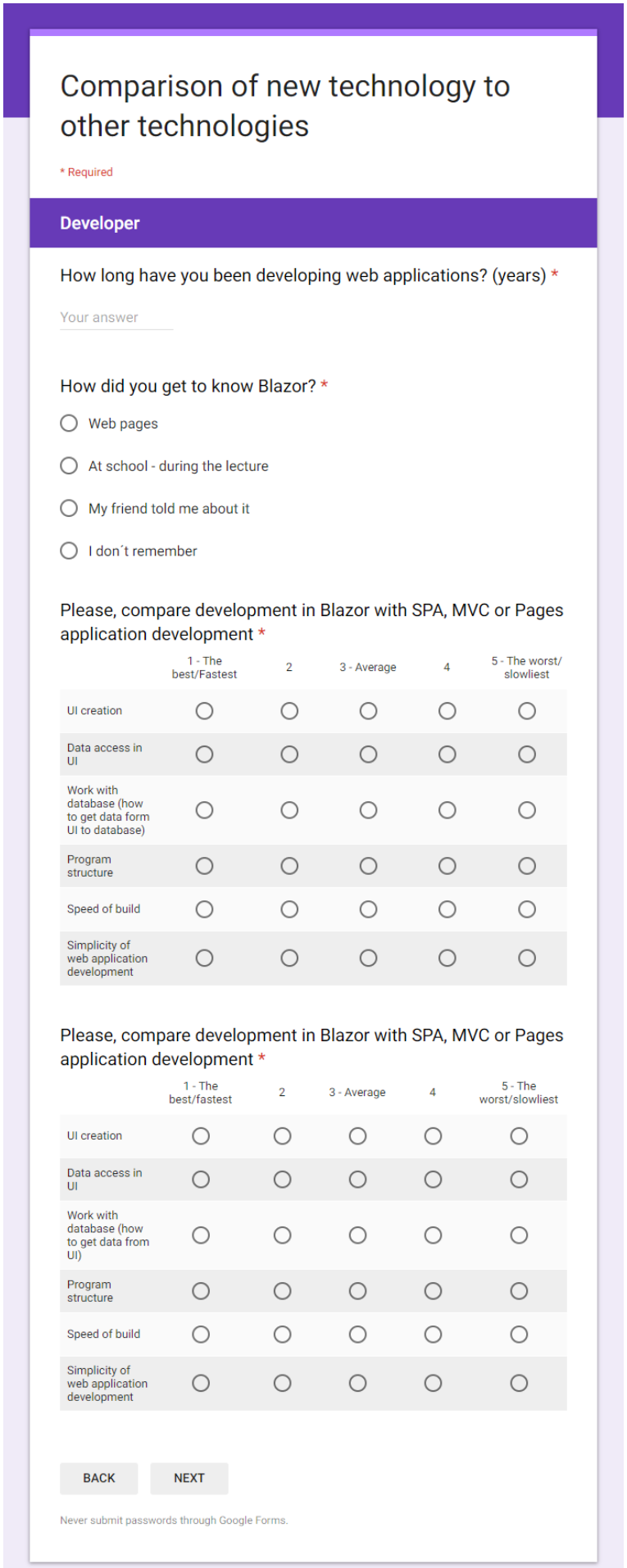


Figure 4.12: Google forms questionnaire - second part for teachers



Comparison of new technology to other technologies

* Required

Developer

How long have you been developing web applications? (years) *

Your answer

How did you get to know Blazor? *

- ○ Web pages
- ○ At school - during the lecture
- ○ My friend told me about it
- ○ I don´t remember

Please, compare development in Blazor with SPA, MVC or Pages application development *

	1 - The best/Fastest	2	3 - Average	4	5 - The worst/ slowliest
UI creation	○	○	○	○	○
Data access in UI	○	○	○	○	○
Work with database (how to get data form UI to database)	○	○	○	○	○
Program structure	○	○	○	○	○
Speed of build	○	○	○	○	○
Simplicity of web application development	○	○	○	○	○

Please, compare development in Blazor with SPA, MVC or Pages application development *

	1 - The best/fastest	2	3 - Average	4	5 - The worst/slowliest
UI creation	○	○	○	○	○
Data access in UI	○	○	○	○	○
Work with database (how to get data from UI)	○	○	○	○	○
Program structure	○	○	○	○	○
Speed of build	○	○	○	○	○
Simplicity of web application development	○	○	○	○	○

BACK NEXT

Never submit passwords through Google Forms.

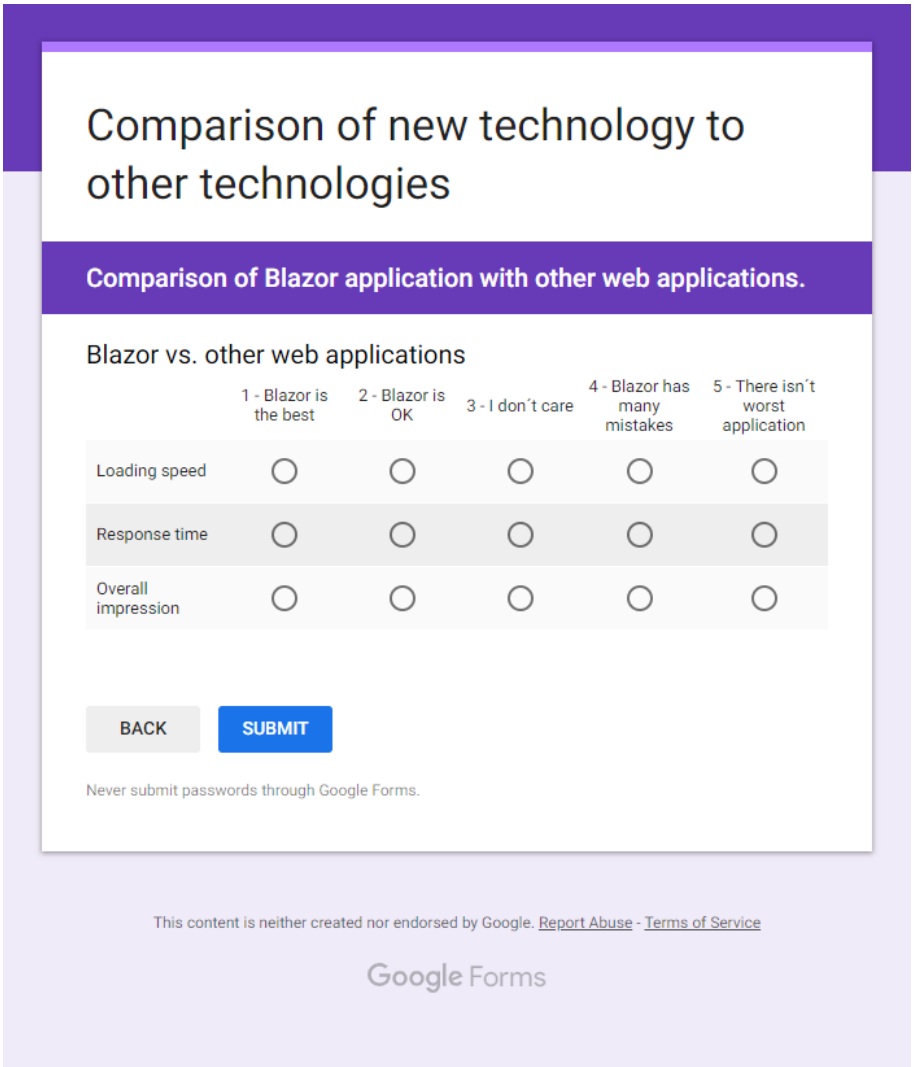


Figure 4.14: Google forms questionnaire - last part.

Number	Question
Q1	How long have you been developing web applications? (years)
Q2	How did you get to know Blazor?
Please, compare development in Blazor with SPA, MVC or Pages application development	
Q3	UI creation
Q4	Data access in UI
Q5	Work with database (how to get data form UI to database)
Q6	Program structure
Q7	Speed of build
Q8	Simplicity of web application development
Please, compare development in Blazor with desktop application development	
Q9	UI creation
Q10	Data access in UI
Q11	Work with database (how to get data from UI)
Q12	Program structure
Q13	Speed of build
Q14	Simplicity of web application development

Table 4.1

ID	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Q13	Q14
1	5	Web pages	2	1	3	1	3	2	3	2	4	2	3	2
2	3	My friend told me about it	2	1	2	1	1	1	2	1	2	1	1	1
3	2	Web pages	3	2	2	2	1	1	2	2	2	2	1	1
4	5	Web pages	1	3	3	2	1	1	2	2	2	3	1	1
5	3	At school - during the lecture	1	1	1	1	1	1	1	1	1	1	1	1
6	9	Web pages	2	2	2	1	1	1	3	3	3	1	1	1
7	6	My friend told me about it	2	2	2	1	1	1	2	2	2	1	1	1
8	3	At school - during the lecture	1	2	3	2	1	1	1	2	3	1	1	1
9	5	My friend told me about it	1	1	1	1	1	1	1	1	1	1	1	1
10	8	I don't remember	2	2	2	2	2	2	2	2	2	2	2	2
11	8	I don't remember	3	3	3	2	1	1	3	3	3	2	1	1
12	1	At school - during the lecture	1	1	1	1	1	1	1	1	1	1	1	1
13	6	Web pages	1	1	1	1	1	1	1	1	1	1	1	1
14	4	I don't remember	2	2	2	2	2	2	2	2	2	2	2	2
15	15	I don't remember	2	2	2	2	2	2	2	2	2	2	2	2
16	20	I don't remember	1	1	1	1	1	1	1	1	1	1	1	1
17	1	At school - during the lecture	2	1	2	2	1	2	1	1	1	1	1	1
18	2	At school - during the lecture	3	2	5	3	4	4	2	2	2	2	2	2
19	8	My friend told me about it	1	1	2	2	4	5	1	1	2	2	5	3
20	6	Web pages	1	1	3	2	2	1	1	2	4	3	3	2
21	3	At school - during the lecture	3	3	2	2	3	2	3	1	2	3	2	1
22	2	At school - during the lecture	3	3	3	3	3	3	2	2	3	2	1	3

Table 4.2: Raw survey data for develpper

ID	Turn on speed	Response time	Simplicity of daily agenda filling	Clarity	Design
23	1	1	1	1	2
24	2	2	3	2	3
25	1	1	1	1	3
26	1	1	1	1	1
27	1	1	2	2	2
28	2	2	2	2	2
29	3	3	3	3	3
30	1	1	1	1	1
31	2	1	1	1	2
32	1	1	1	1	1
33	1	1	1	1	1
34	1	1	2	2	2
35	1	1	1	1	2
36	1	1	1	1	1
37	3	3	3	3	3
38	2	2	2	2	2
39	1	1	1	3	2
40	1	1	1	1	1
41	2	2	2	2	2
42	2	2	2	2	2
43	2	1	2	1	2
44	1	1	1	1	1
45	3	3	2	2	2
46	4	2	3	1	4

Table 4.3: Raw survey data for teacher group

4.5.1 Summary of testing

From aquired data there are some indication that blazor can be good election for internal Information system.

From acquired data it is possible to say that developers who filled this form are satisfied with Blazor technology. Surprisingly there are both - junior and even really experienced developers- excited about this new technology. Most of these developers appreciate the simplicity of web application development, speed of build and program structure.

		Developers	
ID	Loading speed	Response time	Overall impression
1	1	1	1
2	2	1	2
3	2	2	2
4	1	1	2
5	1	1	1
6	2	2	2
7	1	1	1
8	1	1	1
9	1	1	1
10	2	2	2
11	2	2	2
12	1	1	1
13	1	1	1
14	2	2	2
15	2	2	2
16	1	1	1
17	1	1	2
18	4	2	2
19	2	2	1
20	2	3	2
21	3	2	2
22	1	1	2
		Teachers	
23	2	2	2
24	1	1	1
25	2	2	3
26	3	3	3
27	2	2	2
28	2	2	3
29	3	3	3
30	1	1	1
31	2	2	2
32	1	1	1
33	1	1	1
34	1	1	2
35	1	1	2
36	1	1	1
37	3	3	3
38	2	2	2
39	2	2	2
40	1	1	1
41	2	2	2
42	2	2	2
43	1	1	3
44	1	1	1
45	2	2	3
46	3	3	3

Table 4.4: Blazor vs. other web applications

4.6 Summary of this concept

This proof of concept is not fully completed application. There are many problems that must be resolved before this application can be used.

- GUI is optimalized only for mobile but sometime teacher wants to access from Personal computer (PC). Sometimes GUI is lagged and ugly.
- In proof of concept, reading and changing marks for student is not realized. And in real application I will replace API on technology Asp.Net MVC with OData because OData simplify communication and developing server side API.
- Next problem is size of this application which can be resolved by setting browser cache using Expires and Cache-Control or by Last-Modified and ETag headers.
- Additional feature which can be implemented is Service worker (SW). SW provide offline abbility for application. SW can control traffice sent to server because it sits between web application and server. SW can cache CRUD operation when application is offline. After reconnect to network all operation can be synchronized to server. SW must be written in JS.

4.7 Summary of the benefits and potential of the Blazor

Web developers respond quickly to the development of Blazor. Companies such as Telerik are implementing some of there user components for Blazor.

Big advantage of Blazor is the possibility to use the code and libraries that are written in the .Net Standard.

Another advantage is the ability to share code (library) containing methods, classes and complete logic with other web architecture.

Next big advantage for beginner web developers is that he doesn't have to learn another programming language to be able to develop web application. Another advantage is the similarity with previous technologies released by Microsoft like Asp.Net MVC and Razor pages. There is also a lot of community-based extensions such as access to browser LocalStorage and many more. Some Blazor extension can be found on github AdrienTorris/awesome-blazor repository. [32]

The potential of Blazor is mainly in the possibility of replacing technologies for GUI development.So besides using Blazor separetly as client-side or server-side there is a potential to run it as native application. In this case there are some first attemps to use it this way for example in Ionic technology. [33]

This attemp of using Ionic technology with Blazor framework was named Bionic.[34]

I see another potential in a simple development with the help of the Visual Studio development environment, which is one of the best Integrated Development Environmentr (IDE) for development and debugging.

Conclusion

The aim of my thesis was to explore WebAssembly and Blazor framework. This review is in the chapter 2. From parameters explored in this thesis it is possible to say that WebAssembly and Blazor has the potential to be leading technologies in the future of the web application development.

Comparison of Blazor and Big three of client-side web development tools and framework is in chapter 1 and chapter 2.2. In short, Blazor takes many good features of the other web frameworks but also gets a few drawbacks such as its memory requirements.

A part of this thesis is application that may show the way how to create web application in Blazor. This application is described in chapter 4 and source code is included on the attached CD-ROM. It is also possible to follow the development of this application at gitlab of target school [35] and [36].

Blazor's strengths and weaknesses and its readiness to develop modern web applications is described in chapter section 2.2.

A quick summary at the conclusion. In this thesis, multiple frameworks and tools for creating web application were analyzed and were compared to Blazor technology. The analysis shows that each of the picked frameworks is suitable for different uses.

Vue.js is currently the most growing framework and is suitable for the simplest web application. Vue.js can be simply added to existing application to extend user interface. Vue.js has the smallest size compared to others but doesn't have a router in it. I would prefer Vue.js to improve the usability experience with regular web apps such as Asp.Net MVC or Asp.Net Pages.

Angular framework is complex and provides many features to help developer create user-friendly application but when application is done without Angular, it is too expensive to extend this application with Angular. This framework is good for big projects like information systems.

React framework is something between Vue.js and Angular. This framework can be used on existing projects and has many features which help developer to create big user-friendly application. JSX templation system is

really easy to use and can help to fix some errors which will appear during development.

This master thesis is about WebAssembly and Blazor technology that has a lot changes during the writing of this thesis. Now is server-side Blazor released in stable version with .net core 3.0. Server-side Blazor contains component prerendering which helps search engines to crawl site. It can be used on production website without any problems. Client-side Blazor is in preview state so that it can be used in internal information systems because there is missing prerendering and search engines cannot crawl web site content. This problem can be suppressed by site manifest and sitemaps. Web sites created by client-side Blazor are not so much bigger than the sites written in Angular or other big framework.

Blazor framework has the potential to become main framework for web development but there are still some mistakes which must be resolved.

Bibliography

[1] Cesardelatorre. What is Docker? Aug 2018. Available from: https://docs.microsoft.com/en-us/dotnet/standard/microservices-architecture/container-docker-introduction/docker-defined

[2] Lang, M. *Elektronická třídní kniha*. Bachelor's thesis, Czech Technical University - Faculty of Informatics, June 2014. Available from: https://dspace.cvut.cz/handle/10467/23280

[3] Inc., R. The Future of JavaScript in the Front End World. Aug 2018. Available from: https://medium.com/@rangleio/the-future-of-javascript-in-the-front-end-world-2544c1814e2

[4] Goel, A. 10 Best JavaScript Frameworks to Use in 2019. Mar 2019. Available from: https://hackr.io/blog/10-best-javascript-frameworks-2019

[5] Sviatoslav, A. The Best JS Frameworks for Front End. Available from: https://rubygarage.org/blog/best-javascript-frameworks-for-front-end

[6] About stars. Available from: https://help.github.com/en/articles/about-stars

[7] Freeman, A. *Pro AngularJS*. Berkeley, CA New York, NY: Apress,Distributed to the Book trade worldwide by Springer, 2014, ISBN 978-1430264484.

[8] Sanctis, V. *ASP.NET Core 2 and Angular 5 : full-stack web development with .NET Core and Angular*. Birmingham, UK: Packt Publishing, 2017, ISBN 978-1788293600.

[9] Freeman, A. *Pro Angular 6*. London: Apress, 2018, ISBN 978-1484236482.

[10] RFC2616: Hypertext Transfer Protocol. Available from: https://www.rfc-editor.org/rfc/rfc2616.txt

[11] Angular Augury. Available from: https://augury.rangle.io/

[12] Mardan, A. *React quickly : painless web apps with React, JSX, Redux, and GraphQL*. Shelter Island, NY: Manning Publications Co, 2017, ISBN 978-1617293344.

[13] Getting Started with Redux · Redux. Available from: https://redux.js.org/introduction/getting-started

[14] Hanchett, E. *Vue.js in action*. Shelter Island, NY: Manning Publications Co, 2018, ISBN 978-1617294624.

[15] Stack Overflow Developer Survey 2019. Available from: https://insights.stackoverflow.com/survey/2019

[16] Fluin, S. A plan for version 8.0 and Ivy. Feb 2019. Available from: https://blog.angular.io/a-plan-for-version-8-0-and-ivy-b3318dfc19f7

[17] Rourke, M. *Learn WebAssembly: Build web applications with native performance using Wasm and C/C++*. Packt Publishing, sep 2018, ISBN 1788997379. Available from: https://www.xarg.org/ref/a/1788997379/

[18] WebAssembly Core Specification. Apr 2019. Available from: https://webassembly.github.io/spec/core/bikeshed/index.html

[19] Haas, A.; Rossberg, A.; Schuff, D. L.; et al. Bringing the Web Up to Speed with WebAssembly. *SIGPLAN Not.*, volume 52, no. 6, June 2017: pp. 185–200, ISSN 0362-1340, doi:10.1145/3140587.3062363. Available from: http://doi.acm.org/10.1145/3140587.3062363

[20] Lang, M. Project on WebAssembly Studio site. Available from: https://webassembly.studio/?f=ik2138pme4e

[21] Sharma, A. *Blazor Quick Start Guide: Build web applications using Blazor, EF Core, and SQL Server*. Packt Publishing, 2018, ISBN 9781789341300. Available from: https://books.google.cz/books?id=VOh1DwAAQBAJ

[22] Himschoot, P. *Blazor Revealed: Building Web Applications in .NET*. Apress, 2019, ISBN 1484243420. Available from: https://www.amazon.com/Blazor-Revealed-Building-Applications-NET/dp/1484243420

[23] Guardrex. Introduction to Blazor in ASP.NET Core. Available from: https://docs.microsoft.com/en-us/aspnet/core/client-side/spa/blazor/?view=aspnetcore-3.0

[24] Mono. Available from: https://www.mono-project.com/

[25] Easton, M. J. *Cross-platform .NET development : using Mono, Portable.NET, and Microsoft .NET*. Berkeley, Calif: Apress, 2004, ISBN 978-1590593301.

[26] Vemula, R. *Real-time web application development : with ASP.NET Core, SignalR, Docker, and Azure*. Berkeley, CA New York, NY: Apress,Distributed to the Book trade worldwide by Springer, 2017, ISBN 978-1484232699.

[27] Aguilar, J. *SignalR programming in Microsoft ASP.NET*. Redmond, Wash: Microsoft Press, 2014, ISBN 978-0735683884.

[28] Ardalis. Choose between traditional web apps and single page apps. Available from: https://docs.microsoft.com/en-us/dotnet/standard/modern-web-apps-azure-architecture/choose-between-traditional-web-and-single-page-apps

[29] Sš a Voš aplikované kybernetiky s.r.o. Available from: https://www.kyberna.cz/

[30] Gewarren. C# IntelliSense - Visual Studio. Available from: https://docs.microsoft.com/cs-cz/visualstudio/ide/visual-csharp-intellisense?view=vs-2019

[31] Smith, J. *Entity Framework core in action*. Shelter Island, NY: Manning Publications Co, 2018, ISBN 978-1617294563.

[32] AdrienTorris. AdrienTorris/awesome-blazor. May 2019. Available from: https://github.com/AdrienTorris/awesome-blazor

[33] Cheng, F. *Build mobile apps with Ionic 4 and Firebase : hybrid mobile app development*. New York, New York: Apress, 2018, ISBN 978-1484237748.

[34] Bmsantos. bmsantos/bionic. Oct 2018. Available from: https://github.com/bmsantos/bionic

[35] Lang.Matej/mSIS repository. Available from: https://gitlab.kyberna.cz/Lang.Matej/msis

[36] Lang.Matej/schoolApi repository. Available from: https://gitlab.kyberna.cz/Lang.Matej/schoolapi

Acronyms

API Application Programming Interface. 31, 32, 36, 43, 44, 63

CIL Common Intermediate Language. 22

CLI Command line interface. 6

CLR Common Language Runtime. 26, 37

CPU Central processing unit. 36

CRUD Create, read, update and delete. 36, 43, 63

CSS Cascading Style Sheets. 5, 6, 10, 13

DNS Domain name service. 36

DOM Document object model. 6, 13, 20, 27

EF Entity framework. xiii, 45, 52

GUI Graphical user interface. 27, 33, 35–37, 63

HTML Hypertext Markup Language. 5, 6, 10, 13, 22–24, 40

HTTP Hypertext Transfer Protocol. 36

IDE Integrated Development Environmentr. 64

IDP Identity provider. 36, 39, 43, 45

JS JavaScript. 1, 6, 13, 20, 27, 39, 63

JSON JavaScript Object Notation. 22

JWT JSON Web Token. 43

MVC Model-View-Controller. 2, 32

NPM Node.js package manager. 6

ORM Object relational mapping. 45

OS Operating system. 35

PC Personal computer. 63

RAM Random access memory. 27

SAML Security assertion markup language. 45

SPA Single Page Application. 1, 2, 6, 22, 31, 32

SQL Structured query language. 45

SSO Single sign on. 45

SW Service worker. 63

UI User interface. 10, 13

URL Uniform Resource Locator. 5

VM Virtual machine. xiii, 20, 36, 37

VR Virtual reality. 19

W3C World Wide Web Consortium. 19

WASM WebAssembly. 19, 21, 26, 27

XSS Cross-site scripting. 22

Appendix A

Contents of enclosed CD

- `readme.txt` ..... the file with CD contents description
- `src` ..... the directory of source codes
  - `ComparsionWebFrameworksSamples` ..... implementation sources for comparsion
  - `InformacniSystemCore.Blazor` ... implementation sources for Blazor
  - `InformacniSystemCore` ..... implementation sources for Api
  - `Docker` ..... docker files
  - `thesis` ..... the directory of LaTeX source codes of the thesis
- `text` ..... the thesis text directory
  - `thesis.pdf` ..... the thesis text in PDF format
  - `thesis.ps` ..... the thesis text in PS format