|  |  |  |
| --- | --- | --- |
|  | FACULTY OF GERMAN ENGINEERING EDUCATION AND INDUSTRIAL MANAGEMENT |  |

**Development of a system for the automation of processes in a modern gym**

Prepared by:

Martin Vladimirov Gebov

201314009

A thesis submitted to the Faculty of

**GERMAN ENGINEERING AND INDUSTRIAL MANAGEMENT**

In fulfillment of the requirements for the

MASTER’S DEGREE

in Computer Science

Technical University of Sofia

Sofia 1000

Bulgaria

March 2017

# Statement by the author

I hereby declare that this submission is my own work and to the best of my knowledge, it contains no material previously published or written by another person, nor material which to a substantial extent has been accepted for the award of any other degree or diploma at any educational institution, except where due acknowledgement is made in the thesis.

|  |  |
| --- | --- |
| Martin Gebov | Date |
| Student |

|  |  |
| --- | --- |
| Doz.Dr.Ing.Adelina Aleksieva-Petrova | Date |
| Thesis Advisor |

|  |  |
| --- | --- |
| Prof. Dr. Ing. Stefan Stefanov | Date |
| Dean |

# Abstract

© Copyright 2014

by Martin Gebov

All rights reserved

# ACKNOWLEDGEMENTS

I would like to thank all of the people who helped me in the development of this thesis. Their gentle but firm direction has been most appreciated. Doz. Dr. Ing. Adelina Aleksieva-Petrova was particularly helpful in guiding me on how to conduct the needed research, how to gather the useful materials and knowledge to formulate my thesis and in the technical implementation of the project, in which she supplied me with the needed materials and source code samples to get me started on the project at hand.

# TABLE OF CONTENTS

[Statement by the author 2](#_Toc473385696)

[Abstract 3](#_Toc473385697)

[ACKNOWLEDGEMENTS 5](#_Toc473385698)

[TABLE OF CONTENTS 6](#_Toc473385699)

INTRODUCTION

## Goal

The purpose of this thesis is to construct a multi-tier web application for automating processes of a gym and provide the following functionality

1. Ability to login/logout users
   * Users will be able to stay logged for longer periods of time by using a "remember me" option
   * Future work can have a FB/Google/Microsoft login.
2. Ability to register a user.
   * Future work will require the user to confirm his email.
3. Have three different roles to distinguish users - Admin, Instructor and Student
4. Ability to manage users (Admin)
5. Ability to sell items - goods & cards (Admin, Teacher)
6. Ability to track validity of cards (Admin, Teacher)
7. Ability to view past visits and how much visits a student has left (Admin, Teacher, Student)
   * Student will be able to view only his own visits
   * Admin & Teacher will be able to check-in a user when he has completed his training.
   * Future work may have a bar code/card reader that is integrated to work with the system.
8. Ability to view the week program and reserve a spot

## practival applications

The main advantage of this project is that it is entirely web-based and requires no installation on local computers. Users will be intuitively guided through the interface and thus not required to enter the data onto plain sheets of paper. Owners (Admin) tracks this information in a central place and not have to resort to constantly require reports from their Employees (Teachers). The system will is implemented with validation and proper error handling in mind. It as well exposes limited functionality based on the current role of the user. The users of the application may be all kinds of studios, who wish to digitize their business.

## Contributions

The above mentioned web application has been successfully build by leveraging on the capabilities of Angular2 and Asp.net Core and was later successfully tested in a live scenario of a yoga studio, where the users were able to track their visits online.

## Previois work

This is the first time I have worked with the Falcon and haptic devices in general and this proved to be quite the challenge. Therefore much of the information in this thesis has been presented in such a way so that anyone who wishes to further develop applications using the falcon can benefit from the already implemented software library and examples.

Technologies

## Client side

### SPA applications

[SPA][1] stands for single page application and is a technique used for building web applications. It is a commonly used and widespread method that is beginning to outshine the traditional server side rendering frameworks like Asp.Net or JSP.

In a SPA app all the resources, required to load the page are either retrieved on the first page load or are lazy loaded. Lazy-loading means that resources are loaded on demand when needed. This is the preferable method as this way the resources that are required for the initial page load are smaller. This helps for faster loading of resources and improves the overall performance of the web app not only on start-up, but during the whole life-cycle of the application. Resources are retrieved in small optimized chunks from the server and can vary in size and content. For instance, a resource can contain JavaScript logic for customizing the UI elements` behaviour or it can contain .html file templates for data-binding to data. Such resources may even contain a mix of the latter two resource types combined into a "bundle". Using this approach, a module loader framework such as [webpack][2] is required to help retrieve the files from the bundle. One of the differences between SPA apps and other frameworks such as MVC is that at no time does the client go to the server to do a full page render. Resources are retrieved, modified and consumed by the client framework. This helps achieve a seamless transition between the different sections of the site.

Another significant difference is that all of the UI logic happens on the client. The server is unaware of the client and knows only how to provide the data to it (provided that the current user has permission to access it). The benefit of this approach is that this solution is scalable and loosely coupled. It's scalable, because whenever the load increases the servers can be increased to handle more client requests. It's loosely coupled because the UI can connect to any of the servers that provide the needed APIs.

All of the data is retrieved via web services using the http protocol. The data is in the form of JSON as it is lightweight data structure and can be consumed by a wide variety of clients including JavaScript clients like browsers.

The responsibilities of the server consist of serving the data via web APIs and the HTTP REST protocol.

Building a SPA app requires a framework that can manage all of the data-binding, navigation, security restrictions and user interaction. The two most popular frameworks that are mostly adopted by the community and have support for building large enterprise applications are [Angular2][5] and [React][6]. These frameworks are built from start with performance and memory management in consideration. For instance, Angular2 supports Ahead of time compilation (AOT) of html templates. For this project Angular2 will be used as it has a larger community build around it and is a rewrite from AngularJS (the v1 version of the framework). Using Angular is preferable, because it was built from scratch to address all of the issues the v1 (AngularJS) version had.

### Angular 2

[Angular2][5] is a modern framework for building scalable, testable, memory and CPU performing client web applications. Its official release data was on 15.09.2016 and is relatively stable for use. It provides the necessary functionalities which can be used to quickly build a web application, some of which are described below.

[Angular image here]

### Data binding

Since the data and html are separated, there needs to be a mechanism for providing the data to the templates, so they can be populated with data. Angular provides declarative data binding with the html template syntax, that lets developers use directives and curly braces or even write JavaScript code in the templates.

### Routing

The browser is a familiar model of application navigation. We enter a URL in the address bar and the browser navigates to a corresponding page. We click links on the page and the browser navigates to a new page. We click the browser's back and forward buttons and the browser navigates backward and forward through the history of pages we've seen.

The Angular Router ("the router") borrows from this model. It can interpret a browser URL as an instruction to navigate to a client-side page and pass optional parameters along to the supporting page to help it decide what specific content to present. We can bind the router to links on a page and it will navigate to the appropriate application view when the user clicks a link. We can even navigate imperatively when the user clicks a button, selects from a drop box, or in response to some other stimulus from any source. And the router logs activity in the browser's history journal so the back and forward buttons work as well.

Routing enables the user to navigate to different parts of the app, without doing a full page reload. Behind each route lies an entry [component][7] that renders the html output on the browser. This component may be composed of child components, which may have child components of their own. This way a component tree structure is build that in the end outputs html to the browser.

### Modularity

Angular Modules help organize an application into cohesive blocks of functionality. They aim to group components, directives, pipes and services with similar or dependent functionality. Each is focused on a feature area, application business domain, workflow, or common collection of utilities. Modules can also add services to the application. Such services might be internally-developed such as the application logger. They can come from outside sources such as the Angular router and Http client. Modules can be loaded eagerly when the application starts. They can also be lazy loaded asynchronously by the router.

### Software