

ÖZYEĞİN UNİVERSİTY

FACULTY OF ENGINEERING

**CS300**

**SUMMER PRACTICE REPORT**

**Barış Karaer**

**S015497**

**INTERNSHIP COMPANY & DEPARTMENT:**

**Mirsis Information Technologies / Software Development**

**13.07.2020**

#### SUMMER PRACTICE REPORT

|  |  |
| --- | --- |
| **STUDENT** | |
| **Name** | **Barış Karaer** |
| **Internship Start Date** | **13.07.2020** |
| **Internship Completion Date** | **12.08.2020** |
| **Total Working Days** | **20** |
| **COMPANY** | |
| **Name** | **Mirsis IT** |
| **Department** | **Software Development** |
| **Address** |  |
| **SUPERVISOR** | |
| **Name** |  |
| **Title** |  |
| **Department** |  |
| **Phone** |  |
| **E-Mail** |  |
| **Signature** |  |

**DAILY WORK SUMMARY**

|  |  |  |
| --- | --- | --- |
| **DAY** | **DATE** | **WORK DESCRIPTION** |
| **1** | **13.07.2020** | Meeting the team leads and an introduction to the projects we will be working on. A brief information about the Technologies we will be working with such as ReactJS and .NET. Research about ReactJS and how it Works. |
| **2** | **14.07.2020** | Research about .NET Core and making a simple CRUD application combining .NET Core and ReactJS |
| **3** | **16.07.2020** | Simple login application using ReactJS and some research about React Hooks. |
| **4** | **17.07.2020** | Specifying a new project and working towards that application. Choosing a dashboard template and integrating it to React JS |
| **5** | **20.07.2020** | Creating new pages for the front end, learning React redux framework and state containers. |
| **6** | **21.07.2020** | Downloading .Net, Microsoft SQL Server Management Studio. Created a simple database for a new project. Made a Dashboard for an Employee, Department dashboard. |
| **7** | **22.07.2020** | Learned more about .Net core, web APIs and how MVC architecture works while making a project with many frameworks and technologies. Created Models for the Employee Management Project. Connected to the database I created with changing the Web Config settings. |
| **8** | **23.07.2020** | Created controllers for both Employees and Departments. Implementing Get and Post methods in the Web API using .Net. Checked if it works using Postman. Created Get and Post calls in Postman and tested if the controller scripts work. |
| **9** | **24.07.2020** | Implementing Put and Delete methods in the Web API using .NET. Checked if it works using Postman. Created Put and Delete Calls in Postman and tested if the controller scripts work. |
| **10** | **27.07.2020** | Created the Dashboard for the Web API with ReactJS. Implemented the paging with React-Redux and downloaded the libraries that I will need for this project which are react-bootstrap and react-material-ui. Consumed the Get HTTP Request for employees and departments. |

**Student’s Name: Barış Karaer Supervisor’s Name:**

**Student’s Signature: Supervisor’s Signature-Stamp:**

**DAILY WORK SUMMARY**

|  |  |  |
| --- | --- | --- |
| **DAY** | **DATE** | **WORK DESCRIPTION** |
| **11** | **28.07.2020** | Consumed the POST HTTP Request for adding departments and employees. Created a modal so after clicking the add button a modal will pop up and the user will enter the credentials for employee or department. After clicking the submit button a POST request will happen, and the credentials will insert the data to the database. |
| **12** | **29.07.2020** | Consumed the PUT and DELETE HTTP Request. Created the Edit and Delete buttons for the table for both Departments and Employees. After the user presses the delete button it deletes that department or employee. Implemented a modal for the edit page. Each data can be changed with the edit modal page. |
| **13** | **30.07.2020** | Started a new Web API .NET project. Followed the principles of N layer architecture and did research about that. Added references and setting up the coding environment for the project. |
| **14** | **04.08.2020** | Created the Core layer and started implementing the models. Implemented the models Product and Category. Started implementing the generic interfaces. Implemented the IRepository, ICategoryRepository, IProductRepository interfaces. |
| **15** | **05.08.2020** | Continued implementing interfaces. Coded the IService, IProductService, ICategoryService interface. Then implemented the unit of work interface for the Core module. |
| **16** | **06.08.2020** | Created the Data module. Started implementing the Repositories file for the Data module. Implemented Repository, ProductRepository, CategoryRepository files. |
| **17** | **07.08.2020** | Started implementing the Seeds file for the Data module. Implemented CategorySeed and ProductSeed files. |
| **18** | **10.08.2020** | Started implementing the Configuration file for the Data module. Implemented CategoryConfiguration and ProductConfiguration files. |
| **19** | **11.08.2020** | Started implementing the UnitOfWork file for the Data module. Implemented UnitOfWork file for the Data module. |
| **20** | **12.08.2020** | Implemented the AppDbContext for the Data module. Configured the ConnectionStrings and connected the API with the MSSQL database. |

**Student’s Name: Supervisor’s Name:**

**Student’s Signature: Supervisor’s Signature:**

# Abstract

I have conducted my internship in Mirsis Information Technologies. I have earned experience in Full Stack technologies, used ReactJS, .NET Core and earned experience in multi-layer structure .NET Core projects. I have learned how to setup a web application using frameworks, building API’s and creating databases using MSSQL.

# Introduction

The problem I worked on was a dashboard which users can store their related information. Then later they would have access to these information’s through the dashboard. Setting up a database, the coding environment for .NET and ReactJS projects were needed for solving the problem. I needed to learn how the ReactJS framework worked and how .NET Core projects were implemented. I had experience with C# and JavaScript coding, but I have never worked with these frameworks or worked with .NET Core. I am already familiar with database concepts and relational databases so working with MSSQL was very easy and I quickly learned the workbench of MSSQL. Furthermore, a new technique I learned was organizing my code in a multi structured layer so every module would do a specific task. I have learned how interfaces worked but I haven’t used them in a project before. In my .NET Core multi-layer project, I have used interfaces extensively and used it for a better and organized code.

# Company Description

Mirsis Information Technologies responds to the distinctive needs of different sectors and businesses in the field of information technologies. There are many research and development projects that Mirsis works on such as AI Chatbots, AI Medical Assistants, Smart HR Application and ATM Security. Also, Mirsis offers many services such as Turnkey projects, process consultancy and outsourcing. Mirsis has lots of references from variety of sectors like finance, insurance, telecom, IT and Health. For example, Acıbadem, IBM, Akbank, Yapı Kredi, Garanti BBVA are some of the references among many others which can be found on their website. The founders of Mirsis are Gül Düzgider (CEO) and Ece Kutlucan (Managing Partner). They created the company in 2008. Mirsis now has over 133+ software developers, 53+ analysis / test experts, 11+ project managers, 17+ database and system experts and 26+ technicians. I worked in the IT department where these projects are implemented and developed. My role was to learn different Front End, Back End web technologies and create a Web API, dashboard where users can store their personal information.

# .NET Core Multi Structured Web API

## Problem Statement

I have worked on a Web API which is built on the principals of a multi-layer structure. In the meantime, I have created a working demo which consists of a backend (.NET Core Web API), frontend (ReactJS) and a database (MSSQL). The Web API of this project were not based on the principals of a multi-layer structure. I have created models, controllers and the communications with the database, and linked it with the front-end dashboard. After finishing the previous project, in the remaining days of my internship, I started the multi-layer structured project but haven’t finished it. The constraints for coding a multi-layer structure is creating different modules where these modules have separate tasks and linking them appropriately. This makes the problem much harder to solve because it takes time, but it is very organized and if an engineer codes their project based on these principals, other engineers will quickly understand and grasp what the code is about. This is not a new problem for the company, my supervisor said this principal is widely used in other companies too where .NET Core is used. Other national and international companies use this principal for creating their projects. Then they do unit tests for their projects and then deploy it.

## Tools and Techniques Used

I have used JavaScript for using the ReactJS framework. The purpose was to successfully link the .NET Web API with the front-end dashboard. Since the compatibility between two of them are very good, I chose this JS framework. I might have used other JS frameworks but consuming GET or POST methods are very easy using ReactJS, that’s why I used this framework. I had to use the Redux package for state containers. In the dashboard there is a menu which users can click employees, departments or the home button. With these buttons they can browse different web pages. Therefore, I needed to use the Redux package for containing these states. I also used the Bootstrap package for the front-end to look much better. With these packages the dashboard looked cleaner and much more user friendly. I believe there might be better packages for React for the front-end to look better but I am not aware of them, for my purposes the front end was not a priority it just had to be functional enough for me to test the API that I created using .NET Core. This project does not use any hardware. I have created models and controllers for the project, for the other Web API project I have employed the multi-layer structure principal. This technique is employed worldwide for better and cleaner code. The availability, usability and the readability of the code is far greater than other principals. Therefore, it is commonly used in other companies too.

## Detailed Explanation

First, I would like to talk about the API and ReactJS mixed project. I began the project by creating a ReactJS dashboard. I downloaded the framework to my computer and imported various packages like Bootstrap and Redux to my computer using the NPM command. NPM is used for commands users would like to do with the framework. If developers want to create a React application template, then they should be using NPX. If (**npx create-react-app app-name**) is typed in the command prompt, then the NPX system would create an example project template that users can start developing. After creating an example template application, I started browsing and surfing through each folder to observe the purpose of each file. Index.js is the main file which publishes the files to the web but App.js was the main file I was working with at first. After typing (**npm start**) to the command prompt, it starts the application and shows It in the web browser. The browser is started locally, so it is only working in my machine. Changing any code in the App.js file would change the application itself. The installation of bootstrap enabled me to put interactive objects to my application like buttons or tables. Then I learned about function and class components, these components allow developers to write code that is organized. After creating components, developers can import these components in any JavaScript file they want and use them. An example of a class component is shown here.

|  |
| --- |
| İmport React, { Component } from "react"; |
|  |  |
|  | export class Home extends Component { |
|  | render() { |
|  | return ( |
|  | <div className="mt-5 d-flex justify-content-left"> |
|  | <h3>Welcome to Employee Management Dashboard This is the home page.</h3> |
|  | </div> |
|  | ); |
|  | } |
|  | } |

This is my Home page component. In the App.js file, the Home component is called like this.

import { Home } from "./components/Home";

If the developers want to code the Home page and show it in the application, they should code the Home component like this:

<Home />

I have generally used class components in React because it is much easier to move within the project files. There are functional components which developers can use in their React projects, but I didn’t feel the need to use functional components. After creating Employee.js and Department.js I felt the need for the user to browse these pages, that’s why I installed the React router. In the command prompt (npm install react-router-dom) installs the react router extension. To import the React router to the application, the developer needs 3 components. They are the following:

import { BrowserRouter, Route, Switch } from "react-router-dom";

After importing React router I implemented the <BrowserRouter />, <Switch />, <Route /> components so that the user could browse different pages like Home, Employee or Department. The finished App.js file is the following:

|  |
| --- |
| function App() { |
|  | return ( |
|  | <BrowserRouter> |
|  | <div className="container"> |
|  | <h3 className="m-3 d-flex justify-content-center"> |
|  | React JS with Web Api |
|  | </h3> |
|  | <h5 className="m-3 d-flex justify-content-center"> |
|  | Employee Management Portal |
|  | </h5> |
|  |  |
|  | <Navigation /> |
|  |  |
|  | <Switch> |
|  | <Route path="/" component={Home} exact /> |
|  | <Route path="/employee" component={Employee} exact /> |
|  | <Route path="/department" component={Department} exact /> |
|  | </Switch> |
|  | </div> |
|  | </BrowserRouter> |
|  | ); |
|  | } |
|  |  |
|  | export default App; |

The navigation component handles navigating, this JavaScript component is the menu of the application. NavLink components are the components that link to the class components and the “to” keyword is the links to the components in the App.js. The Navigation.js file is the following:

|  |
| --- |
| import React, { Component } from "react"; |
|  | import { NavLink } from "react-router-dom"; |
|  | import { Navbar, Nav } from "react-bootstrap"; |
|  |  |
|  | export class Navigation extends Component { |
|  | render() { |
|  | return ( |
|  | <Navbar bg="dark" expand="lg"> |
|  | <Navbar.Toggle aria-controls="basic-navbar-nav" /> |
|  | <Navbar.Collapse id="basic-navbar-nav"> |
|  | <Nav> |
|  | <NavLink className="d-inline p-2 bg-dark text-white" to="/"> |
|  | Home |
|  | </NavLink> |
|  | <NavLink |
|  | className="d-inline p-2 bg-dark text-white" |
|  | to="/department" |
|  | > |
|  | Department |
|  | </NavLink> |
|  | <NavLink className="d-inline p-2 bg-dark text-white" to="/employee"> |
|  | Employee |
|  | </NavLink> |
|  | </Nav> |
|  | </Navbar.Collapse> |
|  | </Navbar> |
|  | ); |
|  | } |
|  | } |

After finishing the Navigation menu, I started implementing the Department page. Before doing anything, I created a constructor and arranged what I needed for this page.

|  |
| --- |
| constructor(props) { |
|  | super(props); |
|  | this.state = { deps: [], addModalShow: false, editModalShow: false }; |
|  | } |

The state is the state of the page which handles the data to some properties like deps which is the departments array or addModalShow values which handles to show the modal or not.

|  |
| --- |
| componentDidMount() { |
|  | this.refreshList(); |
|  | } |
|  |  |
|  | refreshList() { |
|  | /\* |
|  | this.setState({ |
|  | deps: [ |
|  | { DepartmentID: 1, DepartmentName: "IT" }, |
|  | { DepartmentID: 2, DepartmentName: "SUP" }, |
|  | ], |
|  | }); |
|  | \*/ |
|  |  |
|  | // Fetching from the API |
|  | fetch("https://localhost:44393/api/department") |
|  | .then((response) => response.json()) |
|  | .then((data) => { |
|  | this.setState({ deps: data }); |
|  | }); |
|  | } |
|  |  |
|  | componentDidUpdate() { |
|  | this.refreshList(); |
|  | } |

The componentDidMount function runs when the web page is first opened up and the componentDidUpdate function runs when data is updated. I first created the refreshList function when I haven’t implemented an API for the project, so the code that is commented out sets the state and adds 2 objects. This worked and I tested my code this way at the beginning. After implementing the API, I fetched data from the MSSQL database from that URL endpoint and inserted it into the departments array. This fetching is called consuming the GET method from the API. Then I implemented a Table which shows the data I got from the data table. This is the related code:

|  |
| --- |
| <Table className="mt-4" striped bordered hover size="sm"> |
|  | <thead> |
|  | <tr> |
|  | <th>Department ID</th> |
|  | <th>Department Name</th> |
|  | <th>Options</th> |
|  | </tr> |
|  | </thead> |
|  |  |
|  | <tbody> |
|  | {deps.map((dep) => ( |
|  | <tr key={dep.DepartmentID}> |
|  | <td> {dep.DepartmentID} </td> |
|  | <td> {dep.DepartmentName} </td> |
|  | <td> |
|  | <ButtonToolbar> |
|  | <Button |
|  | className="mr-2" |
|  | variant="info" |
|  | onClick={() => |
|  | this.setState({ |
|  | editModalShow: true, |
|  | depid: dep.DepartmentID, |
|  | depname: dep.DepartmentName, |
|  | }) |
|  | } |
|  | > |
|  | Edit |
|  | </Button> |
|  |  |
|  | <Button |
|  | className="mr-2" |
|  | onClick={() => this.deleteDep(dep.DepartmentID)} |
|  | variant="danger" |
|  | > |
|  | Delete |
|  | </Button> |
|  |  |
|  | <EditDepModal |
|  | show={this.state.editModalShow} |
|  | onHide={editModalClose} |
|  | depid={depid} |
|  | depname={depname} |
|  | /> |
|  | </ButtonToolbar> |
|  | </td> |
|  | </tr> |
|  | ))} |
|  | </tbody> |
|  | </Table> |

I used the map function in JavaScript so I can go through the department objects in the array. I accessed certain attributes in the object as shown inside the <td> HTML code. The delete button runs the deleteDep function when the user clicks on that button. The deleteDep function is this (it runs the DELETE method and calls a specific id number with that method):

|  |
| --- |
| deleteDep(depid) { |
|  | if (window.confirm("Are you sure ?")) { |
|  | fetch("https://localhost:44393/api/department/" + depid, { |
|  | method: "DELETE", |
|  | header: { |
|  | Accept: "application/json", |
|  | "Content-Type": "application/json", |
|  | }, |
|  | }); |
|  | } |
|  | } |

After rendering the page, it is needed to initialize the objects like this:

|  |
| --- |
| const { deps, depid, depname } = this.state; |
|  | let addModalClose = () => this.setState({ addModalShow: false }); |
|  | let editModalClose = () => this.setState({ editModalShow: false }); |

Otherwise we would encounter some errors related to not initializing the objects. Before explaining the modal section of the dashboard, I would like to mention a problem I have encountered while working on the project. I encountered a problem called CORS. CORS policy is a policy which disables everyone from accessing the API I coded. I have overcome this solution by opening up the Powershell in my API project and typing Install Package Microsoft.AspNet.WebApi.Cors. This helped me consume the GET method in my API safely and list the objects I had in my MSSQL database. After safely getting the data, I started to implement the modals. I had access to the React Bootstrap GitHub and used the documentation to have a working modal code. After that I created modals, which would go on top of the existing page and users would be filling the form in a manner they want. As I said, after creating the modals, I implemented form fields for the user to fill in and submit the form. These are the imports I needed:

|  |
| --- |
| import React, { Component } from "react"; |
|  | import { Modal, Button, Row, Col, Form, FormGroup } from "react-bootstrap"; |
|  |  |
|  | import SnackBar from "@material-ui/core/SnackBar"; |
|  | import IconButton from "@material-ui/core/IconButton"; |

This is my constructor which initializes the component and the function which handles the POST method:

|  |
| --- |
| constructor(props) { |
|  | super(props); |
|  |  |
|  | this.state = { snackbaropen: false, snackbarmsg: "" }; |
|  | this.handleSubmit = this.handleSubmit.bind(this); |
|  | } |
|  |  |
|  | snackbarClose = (event) => { |
|  | this.setState({ snackbaropen: false }); |
|  | }; |
|  |  |
|  | handleSubmit(event) { |
|  | event.preventDefault(); |
|  | fetch("https://localhost:44393/api/department", { |
|  | method: "POST", |
|  | headers: { |
|  | Accept: "application/json", |
|  | "Content-Type": "application/json", |
|  | }, |
|  | body: JSON.stringify({ |
|  | DepartmentID: null, |
|  | DepartmentName: event.target.DepartmentName.value, |
|  | }), |
|  | }) |
|  | .then((res) => res.json()) |
|  | .then( |
|  | (result) => { |
|  | this.setState({ snackbaropen: true, snackbarmsg: result }); |
|  | //alert(result); |
|  | }, |
|  | (error) => { |
|  | this.setState({ snackbaropen: true, snackbarmsg: "Failed" }); |
|  | //alert("Failed"); |
|  | } |
|  | ); |
|  | // alert(event.target.DepartmentName.value); |
|  | } |

And these are the HTML codes I used for this modal:

|  |
| --- |
| <div className="container"> |
|  | <SnackBar |
|  | anchorOrigin={{ vertical: "center", horizontal: "center" }} |
|  | open={this.state.snackbaropen} |
|  | autoHideDuration={3000} |
|  | onClose={this.snackbarClose} |
|  | message={<span id="message-id"> {this.state.snackbarmsg} </span>} |
|  | action={[ |
|  | <IconButton |
|  | key="close" |
|  | arial-label="Close" |
|  | color="inherit" |
|  | onClick={this.snackbarClose} |
|  | > |
|  | x |
|  | </IconButton>, |
|  | ]} |
|  | /> |
|  | <Modal |
|  | {...this.props} |
|  | size="lg" |
|  | aria-labelledby="contained-modal-title-vcenter" |
|  | centered |
|  | > |
|  | <Modal.Header closeButton> |
|  | <Modal.Title id="contained-modal-title-vcenter"> |
|  | Add Department |
|  | </Modal.Title> |
|  | </Modal.Header> |
|  | <Modal.Body> |
|  | <Row> |
|  | <Col sm={6}> |
|  | <Form onSubmit={this.handleSubmit}> |
|  | <Form.Group controlId="DepartmentName"> |
|  | <Form.Label>DepartmentName</Form.Label> |
|  | <Form.Control |
|  | type="text" |
|  | name="DepartmentName" |
|  | required |
|  | placeholder="Department Name" |
|  | /> |
|  | </Form.Group> |
|  |  |
|  | <Form.Group> |
|  | <Button variant="primary" type="submit"> |
|  | Add Department |
|  | </Button> |
|  | </Form.Group> |
|  | </Form> |
|  | </Col> |
|  | </Row> |
|  | </Modal.Body> |
|  | <Modal.Footer> |
|  | <Button variant="danger" onClick={this.props.onHide}> |
|  | Close |
|  | </Button> |
|  | </Modal.Footer> |
|  | </Modal> |
|  | </div> |

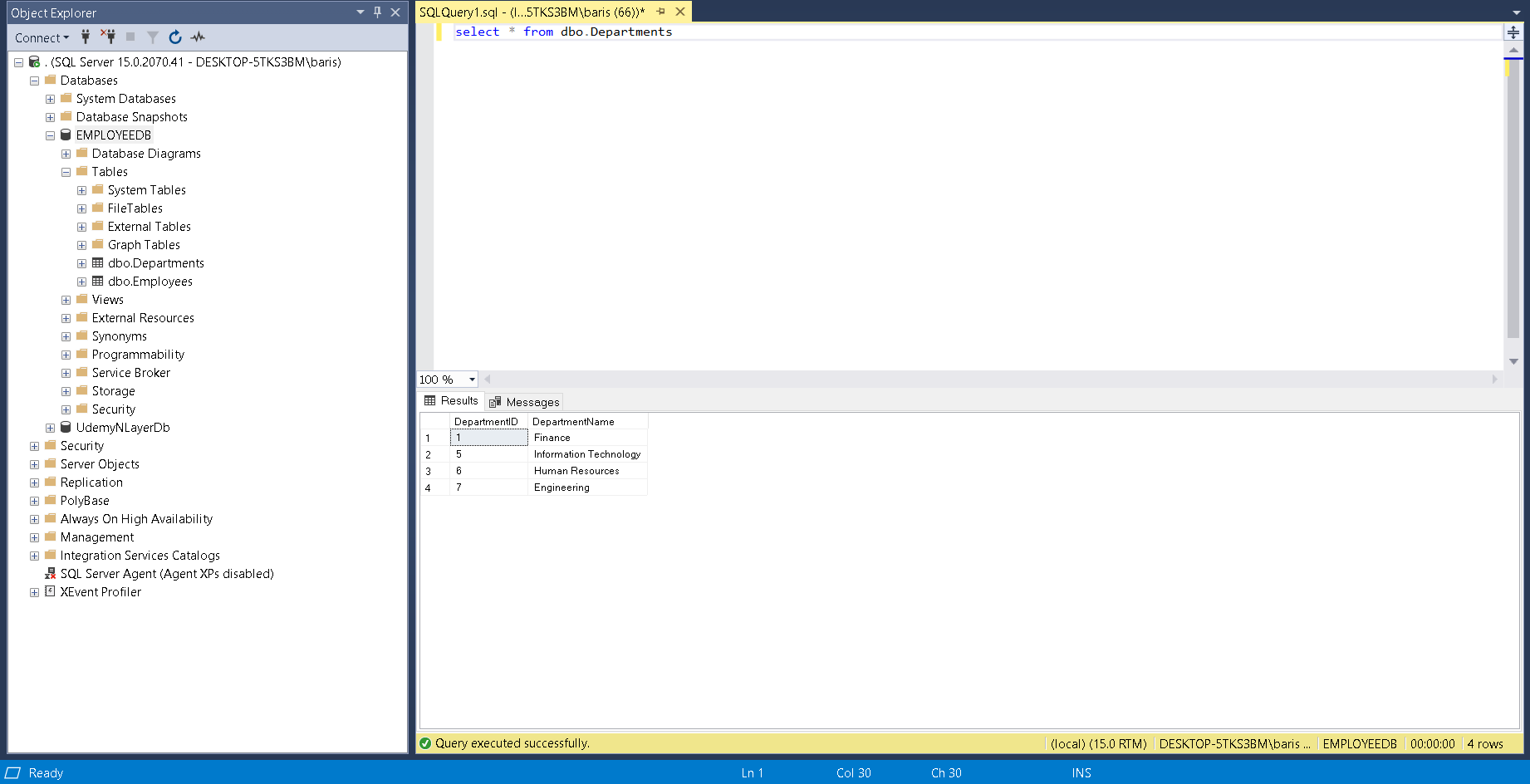
This related code Is the adding modal page that runs after clicking the add button. Now I will be talking about the modal that runs for editing a department. The handleSubmit function is same for the editing modal script but I used a PUT method instead of a POST method and I had the selected department data from the “props” variable. This is the way I implemented the edit form placeholders:

defaultValue={this.props.depname}

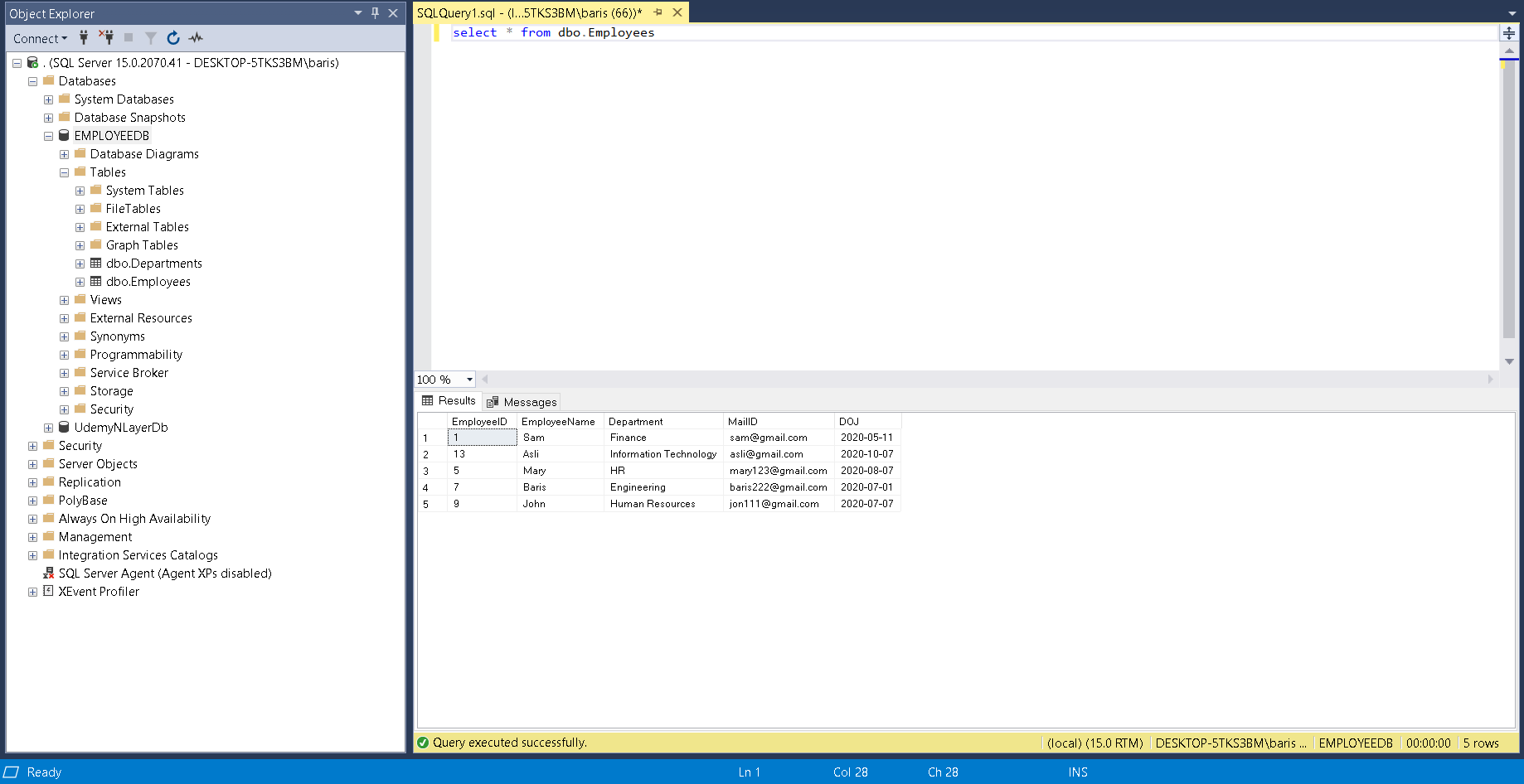
I read the React Bootstrap documentation and implemented date and time form properties from that site. I have finished this React project by creating the dashboard and consuming DELETE, GET, PUT and POST methods. Now I will briefly explain the MSSQL database and Web API I created with this project.

Before starting the project, I created a database called EMPLOYEEDB and created an Employee and Department data table. The attributes of the Employee table are EmployeeID, EmployeeName, Department, MailID and DOJ (Date of Join). The attributes for the Department table are DepartmentID and DepartmentName. I will be providing the pictures of the data tables.

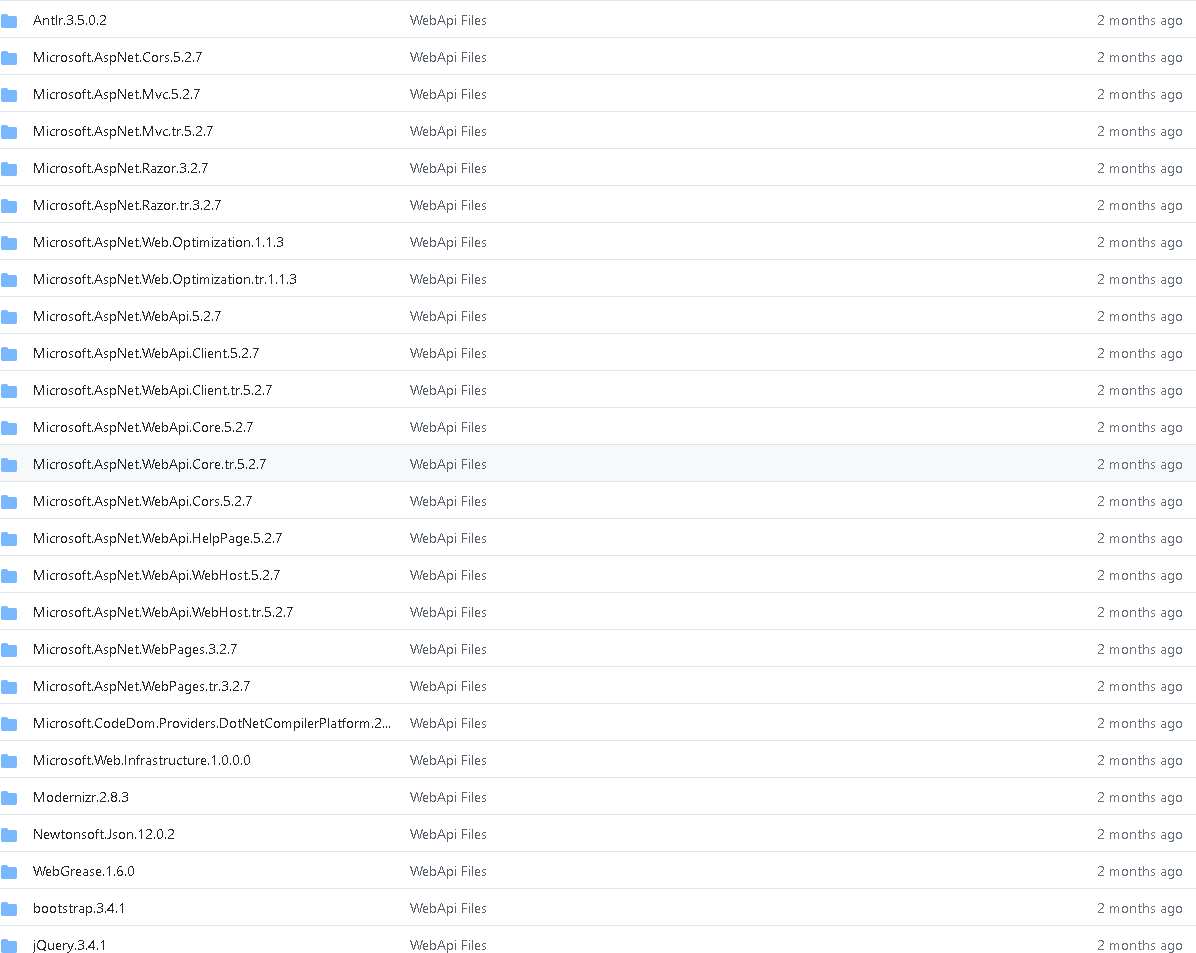
This is the Departments table:



This is the Employees table:



After creating the MSSQL database I started the Web API and installed the necessary packages that I will need later. Here are the packages I downloaded to this project:



After downloading all the packages I needed to force the Web API to send JSON responses. With this line in the WebAPIConfig.cs I handled this problem:

config.Formatters.JsonFormatter.SupportedMediaTypes.Add(

new MediaTypeHeaderValue("text/html"));

I tested this code by implementing this method to the ValuesController.cs script:

public HttpResponseMessage Get()

{

DataTable dt = new DataTable();

dt.Columns.Add("DepID");

dt.Columns.Add("DepName");

dt.Rows.Add(1,"IT");

dt.Rows.Add(2, "Support");

return Request.CreateResponse(HttpStatusCode.OK,dt);

}

By running this project, I checked if this controller worked and started to implement the models for Employees and Departments. Here are the Department and Employee models:

The Department model:

namespace WebApi.Models

{

public class Department

{

public long DepartmentID { get; set; }

public string DepartmentName { get; set; }

}

}

The Employee model:

namespace WebApi.Models

{

public class Employee

{

public long EmployeeID { get; set; }

public string EmployeeName { get; set; }

public string Department { get; set; }

public string MailID { get; set; }

public DateTime? DOJ { get; set; }

}

}

After implementing the models for the project, I created controllers for each of them and implemented the GET, POST, PUT, DELETE methods for each of them.

The Employee Controller:

public class EmployeeController : ApiController

{

public HttpResponseMessage Get()

{

DataTable dt = new DataTable();

string query = @"select EmployeeID, EmployeeName,

Department, MailID, convert(varchar(10),DOJ, 120) as DOJ

from dbo.Employees";

using (var con = new SqlConnection(ConfigurationManager.ConnectionStrings["EmployeeAppDB"].ConnectionString))

using (var cmd = new SqlCommand(query, con))

using (var da = new SqlDataAdapter(cmd))

{

cmd.CommandType = CommandType.Text;

da.Fill(dt);

}

return Request.CreateResponse(HttpStatusCode.OK, dt);

}

public string Post(Employee emp)

{

try

{

DataTable dt = new DataTable();

string DOJ = emp.DOJ.ToString().Split(' ')[0];

string query = @"insert into dbo.Employees

(EmployeeName,Department, MailID, DOJ) values

('" +emp.EmployeeName + @"'

,'" + emp.Department + @"'

,'" + emp.MailID + @"'

,'" + DOJ + @"'

)";

using (var con = new SqlConnection(ConfigurationManager.ConnectionStrings["EmployeeAppDB"].ConnectionString))

using (var cmd = new SqlCommand(query, con))

using (var da = new SqlDataAdapter(cmd))

{

cmd.CommandType = CommandType.Text;

da.Fill(dt);

}

return "Added succesfully";

}

catch (Exception)

{

return "Failed to Add";

}

}

public string Put(Employee emp)

{

try

{

DataTable dt = new DataTable();

string query = @"update dbo.Employees set

EmployeeName = '" + emp.EmployeeName + @"'

, Department = '" + emp.Department + @"'

, MailID = '" + emp.MailID + @"'

, DOJ = '" + emp.DOJ + @"'

where EmployeeID = "+ emp.EmployeeID + @"

";

using (var con = new SqlConnection(ConfigurationManager.ConnectionStrings["EmployeeAppDB"].ConnectionString))

using (var cmd = new SqlCommand(query, con))

using (var da = new SqlDataAdapter(cmd))

{

cmd.CommandType = CommandType.Text;

da.Fill(dt);

}

return "Changed succesfully";

}

catch (Exception)

{

return "Failed to Change";

}

}

public string Delete(int id)

{

try

{

DataTable dt = new DataTable();

string query = @"delete from dbo.Employees where EmployeeID = " + id;

using (var con = new SqlConnection(ConfigurationManager.ConnectionStrings["EmployeeAppDB"].ConnectionString))

using (var cmd = new SqlCommand(query, con))

using (var da = new SqlDataAdapter(cmd))

{

cmd.CommandType = CommandType.Text;

da.Fill(dt);

}

return "Deleted succesfully";

}

catch (Exception)

{

return "Failed to delete";

}

}

}

The Department Controller:

public class DepartmentController : ApiController

{

public HttpResponseMessage Get()

{

DataTable dt = new DataTable();

string query = @"select DepartmentID, DepartmentName from dbo.Departments";

using (var con = new SqlConnection(ConfigurationManager.ConnectionStrings["EmployeeAppDB"].ConnectionString))

using (var cmd = new SqlCommand(query,con))

using (var da = new SqlDataAdapter(cmd))

{

cmd.CommandType = CommandType.Text;

da.Fill(dt);

}

return Request.CreateResponse(HttpStatusCode.OK,dt);

}

public string Post(Department dep)

{

try

{

DataTable dt = new DataTable();

string query = @"insert into dbo.Departments values('" + dep.DepartmentName +@"')";

using (var con = new SqlConnection(ConfigurationManager.ConnectionStrings["EmployeeAppDB"].ConnectionString))

using (var cmd = new SqlCommand(query, con))

using (var da = new SqlDataAdapter(cmd))

{

cmd.CommandType = CommandType.Text;

da.Fill(dt);

}

return "Added succesfully";

}

catch (Exception)

{

return "Failed to Add";

}

}

public string Put(Department dep)

{

try

{

DataTable dt = new DataTable();

string query = @"update dbo.Departments set DepartmentName = '" + dep.DepartmentName + @"'

where DepartmentID = "+ dep.DepartmentID +@" ";

using (var con = new SqlConnection(ConfigurationManager.ConnectionStrings["EmployeeAppDB"].ConnectionString))

using (var cmd = new SqlCommand(query, con))

using (var da = new SqlDataAdapter(cmd))

{

cmd.CommandType = CommandType.Text;

da.Fill(dt);

}

return "Changed succesfully";

}

catch (Exception)

{

return "Failed to Change";

}

}

public string Delete(int id)

{

try

{

DataTable dt = new DataTable();

string query = @"delete from dbo.Departments where DepartmentID = " + id;

using (var con = new SqlConnection(ConfigurationManager.ConnectionStrings["EmployeeAppDB"].ConnectionString))

using (var cmd = new SqlCommand(query, con))

using (var da = new SqlDataAdapter(cmd))

{

cmd.CommandType = CommandType.Text;

da.Fill(dt);

}

return "Deleted succesfully";

}

catch (Exception)

{

return "Failed to delete";

}

}

}

I implemented this Web API before I started the dashboard with React, I tested this API by installing Postman and using it. This project [1] is finished and related repository can be found in the appendix.

Secondly, I would like to talk about the Web API multi-layer project. So far, the Core module and the Data module has been created. I have finished the Core module and finished all the interfaces I had to code for the project. Interfaces are rough sketches of the functions I am supposed to code on other modules. Lastly, I have updated the Connection Strings and I have connected to the MSSQL database I have created early on before I started the project. I first created the models and started to think about what properties I need for Category and Product models. Based on those I have created the getters and setters for these models. Then I created the repositories folder. The files in this folder consists of Actions the users might need in the project. These files are interfaces for the project so it will be implemented later, on another module. I have used the Entity Framework in this project because it is much easier to code for asynchronous functions this way. Also, these methods will return an Entity, this way it will be easier to get the data later other modules. I created the Services folder after finishing the Repository folder. The services folder looks same with the Repository folder, but the Service folder consists of methods that the project use to communicate with the database. I created different files for Category and Product because each does have similar methods, but they have different methods they must consist of too. Unit of Work folder consists of just the interface for the unit of work files on other modules. This unit of work file does commit both asynchronously and synchronously. Also, it has the methods for other modules to get product and category repository interface files. And the last day consisted of me trying to connect to the MSSQL database I have created. The people who inspect my code can check the appsettings.json file and there they can find out the Connection Strings that is based on the localhost of my computer. I have started this project on the last days of my 20-day internship period. Since it has only been in development for 4-5 days, I will be explaining this section in the CS400 document. The code blocks would make much more sense in that document.

This project [2] is not finished but related repository can be found in the appendix and I will continue working on this project.

## Results

The mixed ReactJS and .NET Core API project was successful. Since my solution is in my computer and can only be shown in my own computer, the code repository [1] can be found in the appendix.

The Web API project that is based on the multi-layer structure is not finished but I will be finishing it in the next 20 days of my internship. I will update the repository [2] for the project in the next 20 days and finish the other modules.

# Conclusions

The variety of computer science courses I took from Ozyegin University helped a lot in this internship. CS 102 Object Oriented Programming course helped me a lot for the .NET Project I made in this internship. If it wasn’t for that course I wouldn’t have heard of interfaces and how a project would be created based on the Object-Oriented programming principals. Also, the CS 201 Database course I took really helped me learn relational databases. We learned My SQL in that course, but it is similar to MSSQL. That’s why I was familiar with the concepts. The in-class education assignments and the project assignment in my internship was very similar because I used the same principals, I learned in the courses I took. For example, I used Git for version control systems, and I tried to code very efficiently In a short timeframe. The CS 320 course project was very similar to this project. They were different in a way that there wasn’t any specific hardlines. As an example, in some course projects some of our professors provide a demo and want us to follow that demo to finish up a project, but in this internship I didn’t start from any demo and I planned my coding journey, then started programming. So far, I don’t think any information I learned can be used on any of my class works because I don’t think Ozyegin has any web development courses, but this internship helped me learn how companies start projects and how they develop their API’s. In the future, these skills might help me achieve the web development positions I would want to apply. This internship changed my career goals because in the past I was not aware of the API technologies but now I would like to be involved in researching and developing in this area of Computer Science. My internship was online due to Covid-19 and I couldn’t get to work in the offices of Mirsis but my supervisor helped me and guided me these past 20 days. It was fun working and learning new things. I would have wanted to work with other engineers and get feedback from them, but it wasn’t possible because of the pandemic.

# Appendix

Include relevant material such as catalogues, product specifications, papers

1. <https://github.com/bariskaraer/ReactJS-.NET-Project>
2. https://github.com/bariskaraer/UdemyNLayerProject.API

# References

Each information, figure, table, etc. that does not belong to you (has been found online, taken from some other document, etc.) **must** be referenced, or you risk being penalized due to plagiarism.