SAVONIA

THESIS - **BACHELOR'S DEGREE PROGRAMME**
TECHNOLOGY, COMMUNICATION AND TRANSPORT

ASP.NET CORE 2.1 WEB API FOR A MOBILE APPLICATION

Author/s: Tapio Riihimäki

Field of Study Technology, Communication and Transport			
Degree Programme Degree Programme in Information Technology			
Author(s) Tapio Riihimäki			
Title of Thesis ASP.NET Core 2.1 Web API For A Mobile Application			
Date	27 May 2019	Pages/Appendices	27
Supervisor(s) Mr Jussi Koistinen, Senior Lecturer, Mr Keijo Kuosmanen, Senior Lecturer			
Client Organisation /Partners Tuusplan Oy			

Abstract

The purpose of this thesis was to develop a mobile application for restaurants. The mobile application is for restaurant customers. Customers can use the application when purchasing something in a restaurant. The customer gets points from purchases which he can use to purchase anything the restaurant wants to offer. The restaurants can decide the value of the points used in the restaurant and each restaurant will have their own bonus point system.

During the project for this thesis three different implementations were created. Frameworks were changed for various reasons on this project. The first development was done with React Native using Firebase as the backend. After React Native the framework was changed to Xamarin.Forms and Azure Active Directory B2C was used for the backend. In the third project, the backend and frontend development were separated. Another author, Kristian Tuusjärvi, continued the development of the mobile application. This thesis focuses on the development of the backend with ASP.NET Core as the application server with MySQL database.

The result was a working backend service for the mobile application. The backend service is running on Ubuntu Server 18.04. The client can authenticate and query for required data from the backend.

Keywords
React Native, Xamarin.Forms, .NET Core, backend

SAVONIA-AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Tiivistelmä

Koulutusala Tekniikan ja liikenteen ala	
Koulutusohjelma/Tutkinto-ohjelma Tietotekniikan tutkinto-ohjelma	
Työn tekijä(t) Tapio Riihimäki	
Työn nimi ASP.NET Core 2.1 Web API mobiilisovellukselle	
Päiväys 27.05.2019	Sivumäärä/Liitteet 27
Ohjaaja(t) Jussi Koistinen, lehtori, Keijo Kuosmanen, lehtori	
Toimeksiantaja/Yhteistyökumppani(t) Tuusplan Oy	

Tiivistelmä

Tämän opinnäytetyön tarkoituksena oli kehittää mobiiliapplikaatio, joka toimii bonusjärjestelmänä ravintoloille. Applikaatiota käyttäisivät ravintolan asiakkaat ja he saavat pisteitä, kun tekevät ostoksia ravintolassa. Pisteillä asiakas voi sitten ostaa ravintolan tarjoamia tuotteita asiakkaalle. Ravintolat pystyvät määrittelemään pisteiden määrän ja jokaisella ravintolalla tai ravintolaketjulla olisi oma pistejärjestelmänsä.

Projektin aikana, tätä opinnäytetyötä varten kolme eri projektia luotiin. Tekniikkoja vaihdettiin eri syiden takia tässä projektissa. Ensimmäisessä projektissa kehitys tapahtui React Nativella ja backendina toimi Firebase. Seuraavassa projektissa käytettiin Xamarin.Formsia sekä backendina toimi Azure Active Directory B2C. Viimeisessä projektissa eroteltiin backendin kehitys ja frontendin. Kristian Tuusjärvi jatkoi mobiiliapplikaation kehityksessä. Tämä opinnäytetyö keskittyy backendiin ja ASP.NET Coren kehittämiseen MySQL-tietokannan kanssa.

Opinnäytetyön tuloksena luotiin toimiva backend mobiiliapplikaatiolle. Backend on asennettu Ubuntu Server 18.04:lle. Backend pystyy autentikoimaan ja lähettämään vaadittavia tietoja mobiiliapplikaatiolle.

Avainsanat
React Native, Xamarin.Forms, .NET Core, backend

CONTENTS

ABBREVIATIONS

REST	*Representational State Transfer* REST is an architectural model for creating APIs that are based on HTTP.
SQL	*Structured Query Language* SQL is a language used for querying databases.
API	*Application programming interface* APIs allow apps to use functionality from other apps.
B2C	*Business-to-consumer* The process of a business that is selling products or services straight to the consumer.
AD	*Active Directory* *a service which holds a database consisting of users and other features such as authentication and authorization of users. Developed by Microsoft.*
SSH	*Secure Shell* SSH is a network protocol consisting of rules for computers to interpret on how to transfer data between computers.
SFTP	*SSH File Transfer Protocol* SFTP allows transfering data between a computer and a server securily.
JSON	*JavaScript Object Notation* A data format consisting of key-value pairs.

1 INTRODUCTION

There are many different bonus systems in Finland for different kinds of companies. Many restaurants have their own bonus systems for customers such as Hesburger, Subway and various S-ryhmä restaurants. The group consisting of Kristian Tuusjärvi, Tuija Simonen and Moona Partanen came up with an idea in Savonia's course on product development. The idea was to create a bonus point system for restaurants that do not already have such a system implemented. There are a lot of smaller restaurants in Finland that do not have the resources to implement a bonus system by themselves.

The business idea was to create an application that could help smaller restaurants attract more customers with a bonus point system. The group wanted to make the mobile application enjoyable for users as well, so there were many ideas about implementing various features in it. There were ideas to have games and for example a feature where users could create a team and gather more points together with other users when visiting restaurants. The group was also willing to modify the project according to customers' wishes. For example, adding or removing features that the customers might or might not want for the application. The idea was to create a mobile application that can be used by restaurant customers.

First, the group decided to use React Native as the frontend framework and backend would be implemented by some service such as Firebase. The group wanted to develop the application with a cross-platform framework because there was for the moment only Tapio Riihimäki and Kristian Tuusjärvi going to work on this project. The developers did not want to write code with two different languages because it would mean that there would be a need to have support for almost twice as much code.

The development with React Native did not go well even at the beginning but the project continued for a few months with this framework because it was thought that it would get better after learning more on how to work with it. After some time with React Native, the development was discontinued because of various problems regarding the framework. There were huge issues on upgrading the project and its libraries. Also, some functionality was not as easy to implement as was supposed to, since React Native is in beta it is expected to have some bugs.

After the development with React Native was stopped, the group wanted to find some other cross-platform framework for them to use. So, it was decided that the development would continue with the use Xamarin.Forms because it seemed like a more stable and further developed framework. Since Xamarin is Microsoft based the group decided that the backend would be Azure, because it would be easier to get support using Xamarin and Azure DB together. The author's part for the application was going to be more focused on the calls from the client and usage of the Azure DB with Xamarin.Forms. So, the author would focus on making the application functional and test its performance and responsiveness. The author would also design and develop some of the frontend.

After some time configuring Azure, the group made the decision to create their own backend with .NET Core, Ubuntu Server, Nginx and MySQL. The group thought that in the long run it might get a bit challenging to change from Azure to their own server if the group ever wanted one and it might become too expensive for them compared to developing their own server. It would also be a good learning experience for the author to develop their own backend.

2 TOOLS

2.1 Microsoft Visual Studio Professional 2017

Visual Studio is an integrated development environment (IDE) made by Microsoft. It allows development for Android, iOS, Windows, web and cloud. (Microsoft n.d.) The programming for the .NET Core project was done with Visual Studio Professional 2017.

2.2 MySQL Workbench

MySQL Workbench is a unified visual tool for SQL development and administration. MySQL Workbench also offers data modeling, user administration, backup, and other features. MySQL Workbench is supported for Windows, Linux and Mac OS X. (MySQL n.d.) MySQL Workbench was used to create a local development database and to make a connection with the database on the Virtual Machine to allow configuration on the virtual machines database.

2.3 Oracle VM VirtualBox

Virtualbox is a product for x86 and AMD64/Intel64 virtualization (VirtualBox n.d). VirtualBox was used to create a virtual server for the .NET Core application.

2.4 SqlDBM

SQL Database Modelers Is a tool that allows users to design their databases online. With it the user can design their database visually and also reverse engineer or forward engineer it for MySql or MSSQL. It was free when the author started using it for this project but since then has added limitations for the free version. (SqlDBM n.d.) In the free version it allows you to have only one project at a time and allows the user to reverse engineer or forward engineer only two tables at a time. SqlDBM was used to design the database and forward engineer it from there.

2.5 Postman

Postman is a tool that helps API development (Postman n.d.). The user can manage api calls for his application with it. Postman also offers a lot of other features such as API documentation and mock servers. Postman was used for testing REST API calls for the .NET Core project.

2.6 Git

Git is an open source version control system. Among its many features it allows its users to keep track of changes on their codebase, return to previous versions and resolve conflicts when same file is modified by multiple users. (Git n.d.) Git was used locally for version control during development of the .NET Core project.

2.7 PuTTY

PuTTY is an SSH and telnet client. It was used for remote access to the virtual machine.

2.8 PuTTY SFTP

PuTTY SFTP client was used for transferring files from the development environment to the server.

2.9 Ubuntu Server 18.04

Ubuntu Server is an open source operating system. It works with almost any hardware or virtualization platform (Wallen 2017). Ubuntu server 18.04 was the operating system for the .NET Core project.

2.10 Nginx

Nginx is an open source HTTP server, reverse proxy and an IMAP/POP3 proxy server. Nginx was first a web server and it was developed because there was a need for increased performance on web servers. Nginx offers also for example reverse proxying, caching, load balancing and more. Nginx is used by web sites for companies such as Netflix, Hulu and GitHub. (NGINX n.d.)
Nginx was used as the web server for .NET Core project.

2.11 MySQL

MySQL is an open source database management system developed by Oracle Corporation. Databases in MySQL are relational. (MySQL n.d.) MySQL was used as the database for the .NET Core project.

3 TECHNOLOGIES

React Native and Xamarin.Forms were first used for development but were not used in the final project. ASP.NET Core was used in the final project. For this thesis researching and testing of different frameworks that would best suit the development of the mobile application was done.

3.1 React Native

React Native is an open source JavaScript framework developed by Facebook. React Native is made for creating mobile applications for IOS and Android. It is based on React, a JavaScript library for creating user interfaces. For iOS React Natives "bridge" uses objective C and for Android Java to invoke the native rendering APIs meaning that the application renders with native mobile UI components instead of web components. This way of rendering is more efficient and makes it identical to Android or iOS apps that are programmed with their native programming languages (Eisenman 2017).

React Native is used on many popular mobile applications such as Facebook app, Instagram, Skype and Uber (Facebook n.d). React Native was not used in the final project. React Native was still in beta during this project.

3.1.1 Benefits

Biggest advantage in cross-platform languages such as React Native is that you need to support only one code base meaning there will be less bugs and new features will be easier to implement since you write code only once and get it for both iOS and Android platforms. This also means that there will not be many differences on the iOS and Android version. Most of the development will be done with one language which is JavaScript.

Hot Reloading is also a good feature in React Native it allows developers to see their work at the same time when the application is running. Saves time because the user does not need to restart the application every time that some changes have been implemented. In theory it means that it will boost productivity and the development time will be shorter. (Chrzanowska 2019.)

3.1.2 Cons

Performance is slightly worse compared to developing with the Android and iOs platforms native languages (Thoughtbot 2018). React Native is still in beta so it can have some issues when developing with it. There are also some features that can not be done with React Native and the user needs to program them natively if he wishes to use them.

3.2 Firebase

Firebase is a mobile and web application development platform developed by Google. Firebase allows developers to have a backend service without implementing their own backend. This can save time during the development process.

Developers can use Firebases Realtime Database to store their data in a NoSQL database for their application. The Realtime Database uses local cache to store changes made offline on the application and then sends the local data to the database when the application goes online again. Developers can also implement authentication easily with Firebases libraries. Firebase also supports many other features such as Firebase Test Labs. (Rajat 2018.) Firebase was used in the project that was created with React Native. It was used as a backend for the mobile application.

3.3 Xamarin.Forms

Xamarin.Forms is a cross-platform framework for developing mobile applications with C#. Developers can build native user interface layouts with it for three mobile platforms. Xamarin.Forms supports Android, iOS and Windows Phone. Xamarin.Forms produces a 100% native Android, iOs or Windows Phone app. (The Telerik Team 2018.)

Xamarin.Forms is good for creating prototypes quickly for multiple platforms at once. It has many user interface elements that are rendered differently to each platform which means that the application in each of the platforms such as iOS or Android might look different.

Xamarin.Forms needs most of the times some work done in native development as well because there might be bug or some missing functionality with only using Xamarin.Forms. It also might be difficult to find support for some issues when using Xamarin.Forms since the community is small compared to native development. (Stone 2017.) Xamarin.Forms seems to be best for creating smaller applications that don't have too many features in them. Xamarin.Forms was not used in the final project.

3.4 Azure Active Directory B2C

Azure AD (Active Directory) B2C is used for authentication of users on applications. Developers can set it up on Azure Portal which is a console for the Azure backend services. It allows any user to sign up with either email or a social media provider such as Facebook. Azure AD B2C was used for the Xamarin.Forms project backend. Azure AD B2C was not used in the final project.

3.5 ASP.NET Core

ASP.NET Core is an open-source framework for creating web applications, services and mobile backends on the .NET platform. It performs faster than other popular web frameworks such as Java Servlet or Node.js. ASP.NET Core is a cross-platform version of .NET Framework. (Microsoft n.d.)

ASP.NET Core uses the Model-View-Controller architectural pattern. Support for developing with ASP.NET Core is very good since there are many developers working with it and Microsoft offers many tutorials for it. ASP.NET Core was used for the application server in the final project.

4 DEVELOPMENT PROCESS

The author and Kristian Tuusjärvi first started developing with React Native to create the mobile application. The React Native mobile application would use Firebase as the backend. During development the framework was changed from React Native to Xamarin.Forms due to problems faced with React Native. The backend was also changed from Firebase to Azure. Lastly, the developers stopped development with Xamarin.Forms and decided that they should implement their own backend. The developers thought the costs might be too high with ready-made backend services and, some issues might be faced with these services that could not be solved because the framework could not be modified accordingly. The final application for this thesis was done with the ASP.NET Core framework.

4.1 React Native

The group first decided with Kristian Tuusjärvi, Moona Partanen and Tuija Simonen to use React Native as the framework for development. The group wanted to go with a cross-platform framework because it would mean that there would only be a need to write code once for both iOS and Android. The group also read good things about it from various sites online.

The plan for development with Kristian Tuusjärvi and the author was that the author would focus on programming the functionality and logic for the mobile application. The author was also making sure the app would work with the backend. Implementing the backend with Firebase was part of the work as well. Using Firebase was easy and creating a database at least for testing purposes there was easy.

When developing with React Native Node.js was used to run the application. It was used from the command-line interface in Windows 10. The emulator used was from Android Studio. Developing with hot reloading seemed to work well in the beginning.

In the beginning when starting the development with React Native, the author had some minor issues with the framework that he was able to fix. Using Yarn as the package manager seemed to work better compared to using npm, it fixed some issues.

After some time developing, the author ran into major problems with updating different libraries and having a stable connection to our Firebase backend. In a meeting with Kristian Tuusjärvi after a few months of development with React Native, a decision was made to stop the development with it and try to find something more stable for the group to use.

4.2 Xamarin.Forms

It was decided with Kristian Tuusjärvi to change the framework for development from React Native to Xamarin.Forms. It seemed like a more stable and mature framework after researching on it for a while. This time the group also made the decision that if the project would have some major issues, they would change from it to something else faster and not waste too much time trying to fix issues regarding the framework.

The development process was going to be similar to what it was with React Native. The author would focus more on the functionality and backend for our application and Kristian Tuusjärvi more on the design and frontend.

It was found out rather quickly that it would be more difficult to implement a Xamarin.Forms application with Firebase as the backend of the project. There was a lot more documentation on Xamarin.Forms working with Azure Active Directory backend compared to Firebase. It would be possible to work with Firebase as well, but the author made the decision to switch to Azure Active Directory to be safe and have more support in case there would be problems with something regarding the backend working with the application.

Implementing a database and everything regarding the backend was rather simple with Azure Active Directory. The author used Azure Active Directory B2C for the applications authentication on the backend. With it, users could sign in using various social media accounts or create their own account using it.

The author created an Xamarin.Forms application which can be seen in Figure 1. The purpose was to test how the backend works with Azure AD B2C. The user could sign in using Facebook to automatically fill values for the account or registering with an email account. The email account was verified by Azure AD B2C during registration. It was also used to test how to return data that should be available to everyone or only authenticated users. For example, a GetSecuredData method was created to receive secured data from the backend as shown in Figure 2.

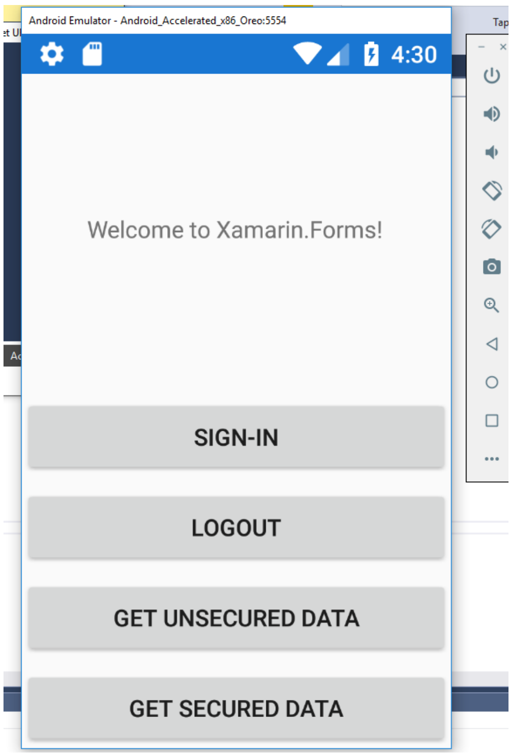
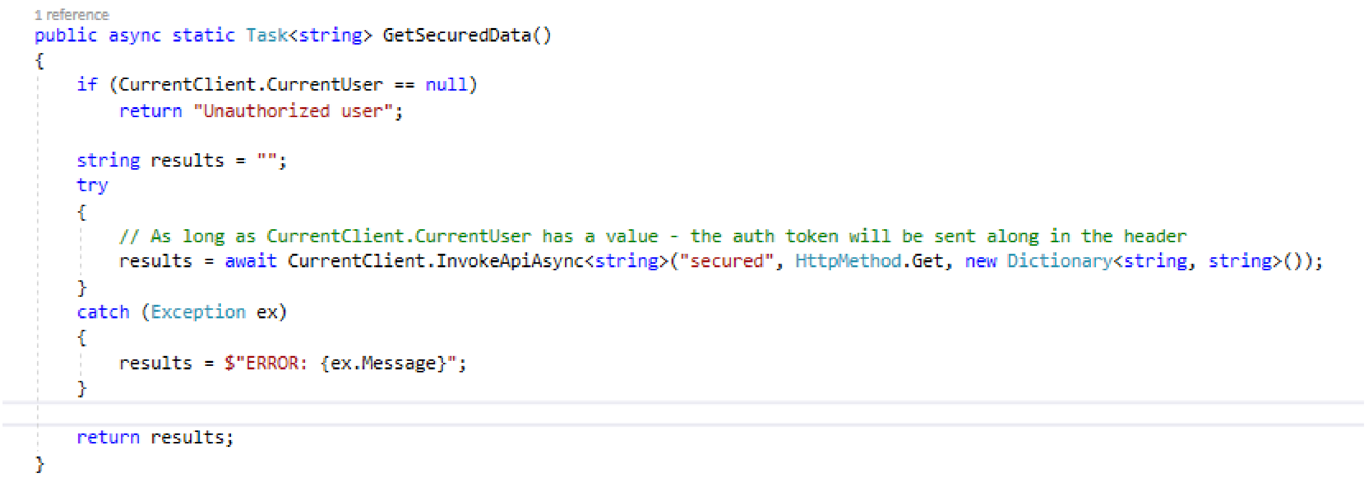

Android Emulator - Android_Accelerated_x86_Oreo:5554
4:30
Welcome to Xamarin.Forms!
SIGN-IN
LOGOUT
GET UNSECURED DATA
GET SECURED DATA

FIGURE 1. Xamarin.Forms mobile application



```
1 reference
public async static Task<string> GetSecuredData()
{
    if (CurrentClient.CurrentUser == null)
        return "Unauthorized user";

    string results = "";
    try
    {
        // As long as CurrentClient.CurrentUser has a value - the auth token will be sent along in the header
        results = await CurrentClient.InvokeApiAsync<string>("secured", HttpMethod.Get, new Dictionary<string, string>());
    }
    catch (Exception ex)
    {
        results = $"ERROR: {ex.Message}";
    }

    return results;
}
```

FIGURE 2. GetSecuredData method that returns secured data to the client.

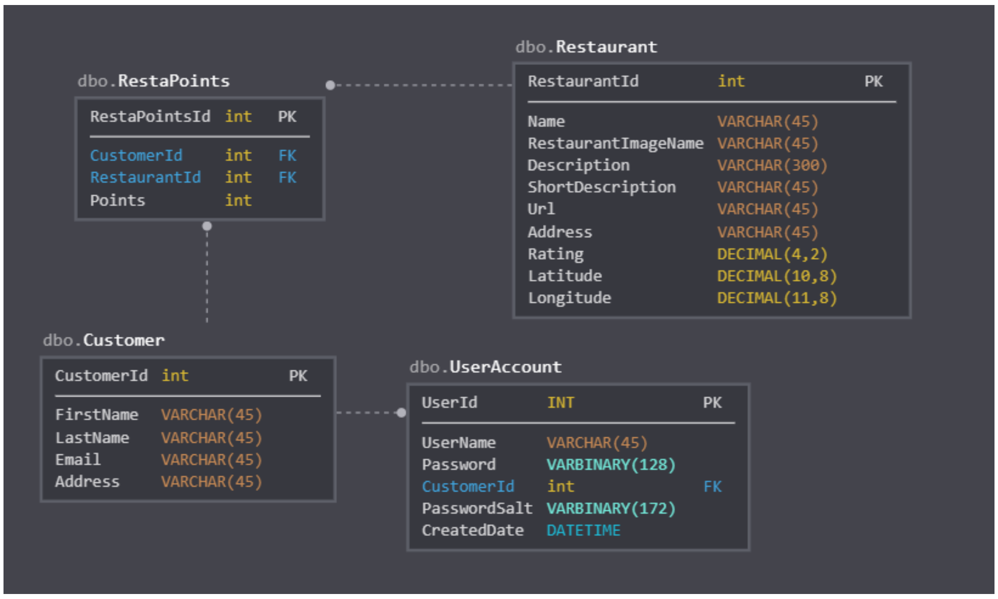
The development went well with Xamarin.Forms and Azure as backend there were no major problems during this time. Kristian Tuusjärvi had some issues regarding the differences on the rendering of the application for iOS and Android. The author had a meeting with Kristian Tuusjärvi about their progress. The developers decided that the author should develop a backend for the project. This decision was made because the costs might become too expensive for the project, using these kinds of ready-made backend services. Azure offered some free trial to use its services. The group also decided that the author would not be developing the mobile application anymore since the author would have enough work with designing and developing their own backend services for the project.

Kristian Tuusjärvi, the developer for the frontend, was going to stop using Xamarin.Forms and start using Flutter which was a new cross-platform framework developed by Google. Xamarin.Forms was not as good for the frontend development. He was also going to do some other work with flutter so thought he might learn it better when doing this project with it also.

4.3 Database

The author wanted to use MySQL as our framework for the database because it was free and popular which implied that it would have good support. The author started planning the first version of the database with SqlDBM. Designing the database was easy with SqlDBM. The database was also forward engineered with SqlDBM. Meaning that SqlDBM creates the SQL commands from the model for creating the database. The database was created with MySQL workbench using the SQL.

The first database consisted of tables for users, customers, restaurants and points. The author would first implement it there and think about use cases for the project's database such as does the user and customer tables need to be kept separate from each other or not. First version of the database was created with SqlDBM and then forward engineered to MySQL database. The first version of the database can be seen in Figure 3.



dbo.RestaPoints

RestaPointsId	int	PK
CustomerId	int	FK
RestaurantId	int	FK
Points	int	

dbo.Restaurant

RestaurantId	int	PK
Name	VARCHAR(45)	
RestaurantImageName	VARCHAR(45)	
Description	VARCHAR(300)	
ShortDescription	VARCHAR(45)	
Url	VARCHAR(45)	
Address	VARCHAR(45)	
Rating	DECIMAL(4,2)	
Latitude	DECIMAL(10,8)	
Longitude	DECIMAL(11,8)	

dbo.Customer

CustomerId	int	PK
FirstName	VARCHAR(45)	
LastName	VARCHAR(45)	
Email	VARCHAR(45)	
Address	VARCHAR(45)	

dbo.UserAccount

UserId	INT	PK
UserName	VARCHAR(45)	
Password	VARBINARY(128)	
CustomerId	int	FK
PasswordSalt	VARBINARY(172)	
CreatedDate	DATETIME	

FIGURE 3. First version of the database in SqlDBM.

After creating the database to a local environment, the author also moved it to a virtual machine with Ubuntu Server 18.04 installed. A connection was established with MySQL Workbench to the database in the server and the database was created there using SQL. The first version of the database had four tables on it. All images for the application are going to be stored on the server but not in the database. The database will have the file paths for the images in case the group is going to add them in the future.

After some time developing the application server, the author added some additional tables and columns to the database as can be seen in Figure 4. Most of the tables and columns were part of the Identity framework for .NET Core. With Identity framework the author was able to use email authentication and improve the security of the application easily. It also adds support for recovering passwords, authentication with applications such as Google and Facebook and other platforms. Identity framework also provides many other useful features such as Account lock-out after failed login attempts, account confirmation with email and authorization.

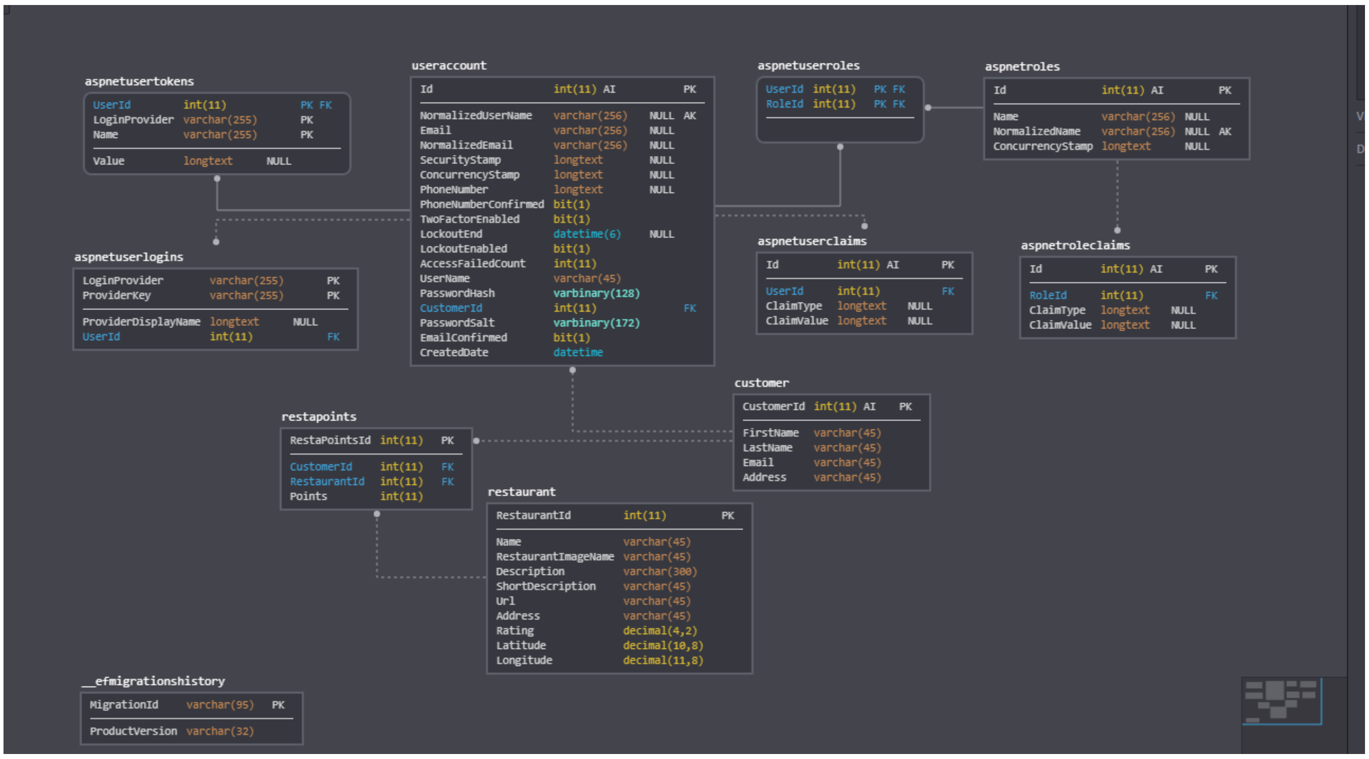


FIGURE 4. The final version of the database in SqlDBM.

4.4 ASP .NET Core

The final application server project for this thesis was made with ASP.NET Core 2.1.1.
The author made the decision to use .NET Core as the project's framework for the application server because he had already worked with it before with success and it was easy and fast to develop with.

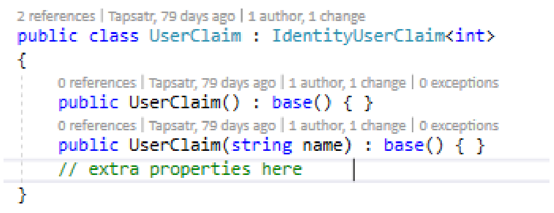
4.4.1 Pomelo

The author used Pomelo.EntityFrameworkCore.MySql library for the connection with the database and application server. It allows the scaffolding of the models and the context class for the application server.

4.4.2 Model-view-controller

The architectural pattern used was the Model-View-Controller (MVC) pattern. The idea with it is that the model holds the context for the database and view is the UI for the user. Controller is the part that handles the calls to the application from the client.

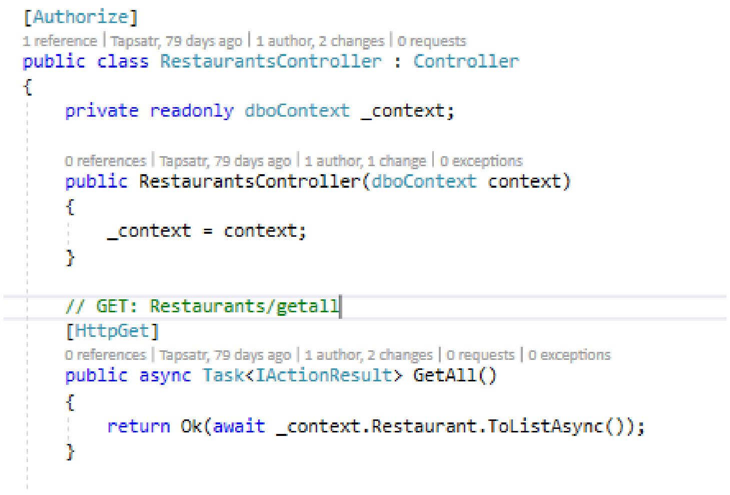
Model holds the data as objects for points, restaurants, customers and users, user claims, and user roles. The models for user claims, user roles and users inherit from the identity framework. UserClaim class can be seen in Figure 5.



```
2 references | Tapsatr, 79 days ago | 1 author, 1 change
public class UserClaim : IdentityUserClaim<int>
{
    0 references | Tapsatr, 79 days ago | 1 author, 1 change | 0 exceptions
    public UserClaim() : base() { }
    0 references | Tapsatr, 79 days ago | 1 author, 1 change | 0 exceptions
    public UserClaim(string name) : base() { }
    // extra properties here
}
```

FIGURE 5. UserClaim model that inherits IdentityUserClaim class.

The controllers handle all the calls the client makes to the server. For example, if the user wants to get a list of all the restaurants the mobile application calls the RestaurantsController and in there the GetAll method which returns the restaurants. RestaurantsController class is shown in Figure 6. Every controller has the Authorize attribute which means that only authorized users can get the data. The method for Authentication has the attribute AllowAnonymous to allow all users to try and log in.



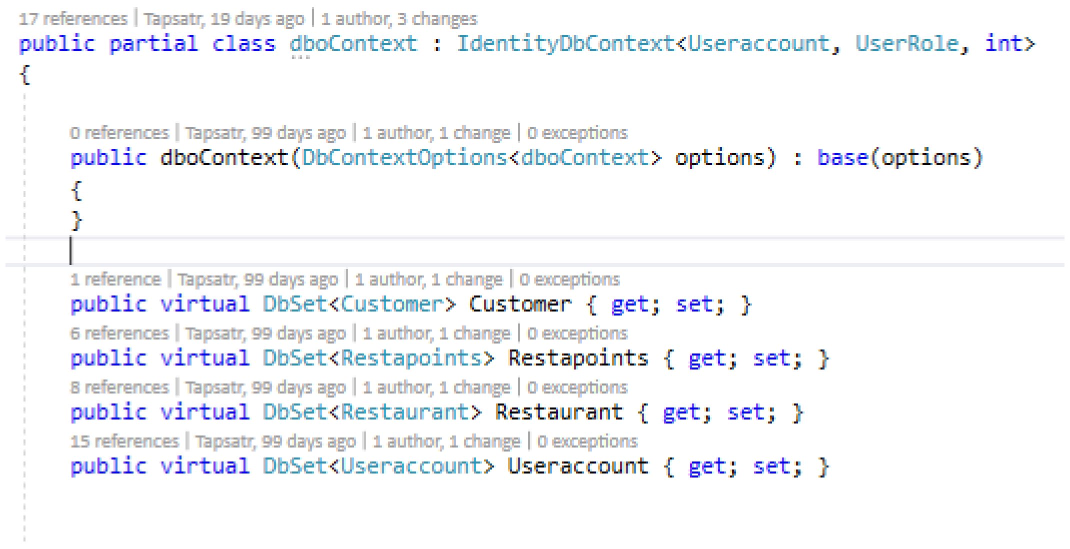
```
[Authorize]
1 reference | Tapsatr, 79 days ago | 1 author, 2 changes | 0 requests
public class RestaurantsController : Controller
{
    private readonly dboContext _context;

    0 references | Tapsatr, 79 days ago | 1 author, 1 change | 0 exceptions
    public RestaurantsController(dboContext context)
    {
        _context = context;
    }

    // GET: Restaurants/getall
    [HttpGet]
    0 references | Tapsatr, 79 days ago | 1 author, 2 changes | 0 requests | 0 exceptions
    public async Task<IActionResult> GetAll()
    {
        return Ok(await _context.Restaurant.ToListAsync());
    }
```

FIGURE 6. RestaurantsController class.

A dboContext class was implemented for the application to use create, update, read and delete operations against the database. The DbSet objects map to the corresponding database tables. the dboContext class inherits from IdentityDbContext. It manages the connection with the database and provides functionality with methods to use with the database.


```
17 references | Tapsatr, 19 days ago | 1 author, 3 changes
public partial class dboContext : IdentityDbContext<Useraccount, UserRole, int>
{

    0 references | Tapsatr, 99 days ago | 1 author, 1 change | 0 exceptions
    public dboContext(DbContextOptions<dboContext> options) : base(options)
    {
    }

    1 reference | Tapsatr, 99 days ago | 1 author, 1 change | 0 exceptions
    public virtual DbSet<Customer> Customer { get; set; }
    6 references | Tapsatr, 99 days ago | 1 author, 1 change | 0 exceptions
    public virtual DbSet<Restapoints> Restapoints { get; set; }
    8 references | Tapsatr, 99 days ago | 1 author, 1 change | 0 exceptions
    public virtual DbSet<Restaurant> Restaurant { get; set; }
    15 references | Tapsatr, 99 days ago | 1 author, 1 change | 0 exceptions
    public virtual DbSet<Useraccount> Useraccount { get; set; }
```

FIGURE 7. Part of the dboContext class

One Data Transfer Object (DTO) was created from the customer and user account models. The purpose of the DTO is to allow the client to get the data for both customer and user account tables easily.

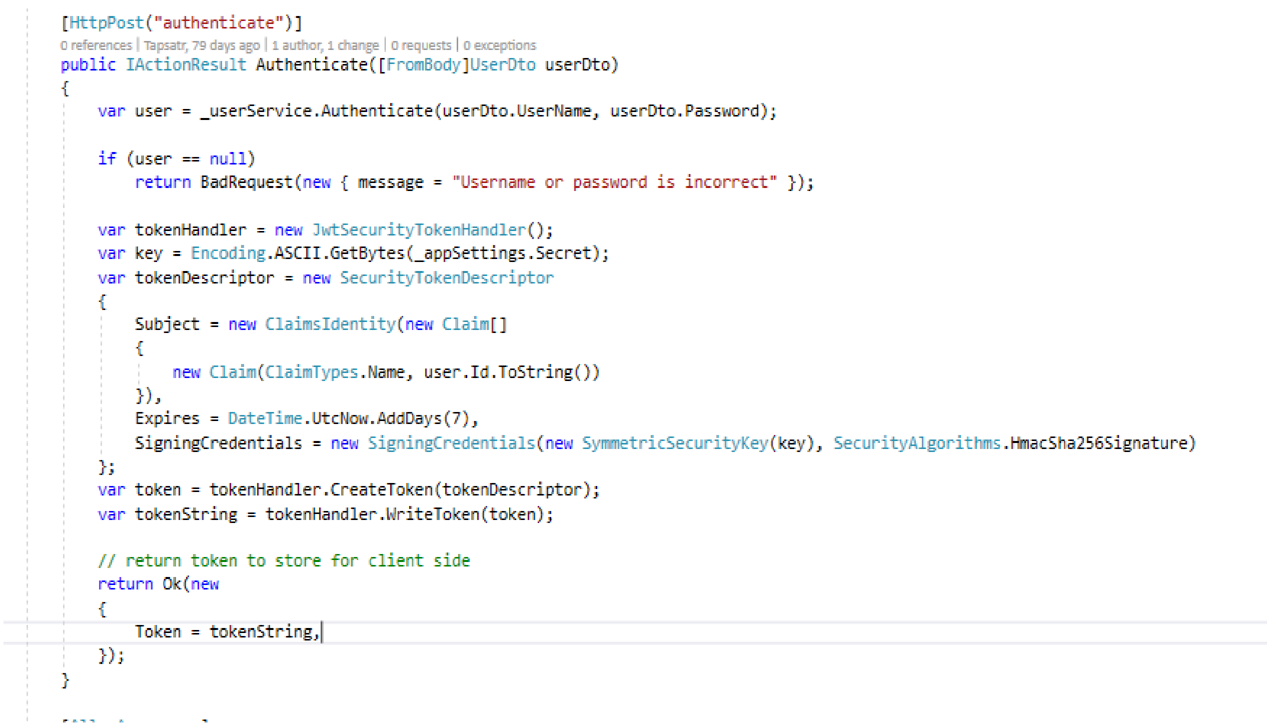
The View was not used yet for the project, but it might be used later to create a control panel for the restaurants. The control panel would allow them for example to change values of their points and add and edit information about their restaurant.

4.4.3 Authentication

For authentication the Identity framework was used and a JavaScript Object Notation Web Token (JWT) was used in the project to store the information of an already authenticated user. There are no plain text passwords used for security.

The passwords are hashed which means that they are converted into a fixed size string with some algorithm. With hashing if the same password is used the hash will be the same for both so it might be of use to a hacker. For this reason, the passwords are also salted which means that there is some random data added to them to further improve the security of the passwords. Salting the password means that even though there might be same passwords used the salted password will be unique.

The authentication is implemented in the following way: User calls the Authenticate method in the LoginController class. Figure 8 shows the Authenticate method. The client sends a JavaScript Object Notation (JSON) that has values for password and username. First thing the method does is checks if the username and password are correct and if incorrect returns an error message. The method also creates a token for the user that expires in a week, so the user does not have to log in every-time he uses the application. If the authentication is successful it returns only the token to the client which can be used for authentication in the future.



```
[HttpPost("authenticate")]
0 references | Tapsatr, 79 days ago | 1 author, 1 change | 0 requests | 0 exceptions
public IActionResult Authenticate([FromBody]UserDto userDto)
{
    var user = _userService.Authenticate(userDto.UserName, userDto.Password);

    if (user == null)
        return BadRequest(new { message = "Username or password is incorrect" });

    var tokenHandler = new JwtSecurityTokenHandler();
    var key = Encoding.ASCII.GetBytes(_appSettings.Secret);
    var tokenDescriptor = new SecurityTokenDescriptor
    {
        Subject = new ClaimsIdentity(new Claim[]
        {
            new Claim(ClaimTypes.Name, user.Id.ToString())
        }),
        Expires = DateTime.UtcNow.AddDays(7),
        SigningCredentials = new SigningCredentials(new SymmetricSecurityKey(key), SecurityAlgorithms.HmacSha256Signature)
    };
    var token = tokenHandler.CreateToken(tokenDescriptor);
    var tokenString = tokenHandler.WriteToken(token);

    // return token to store for client side
    return Ok(new
    {
        Token = tokenString,
    });
}
```

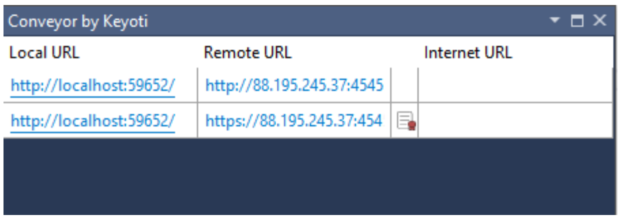
FIGURE 8. Authenticate method in LoginController class

4.5 Testing

Testing different calls and what they returned from the application server was done with Postman and an Android device.

4.5.1 Conveyor by Keyoti

Conveyor by Keyoti allows the developer to access the application server during development from other machines when the application is running on the development web server. It creates a Remote URL for the application so it can be accessed with other than local devices. Figure 9 shows the Conveyor by Keyoti interface in Visual Studio.



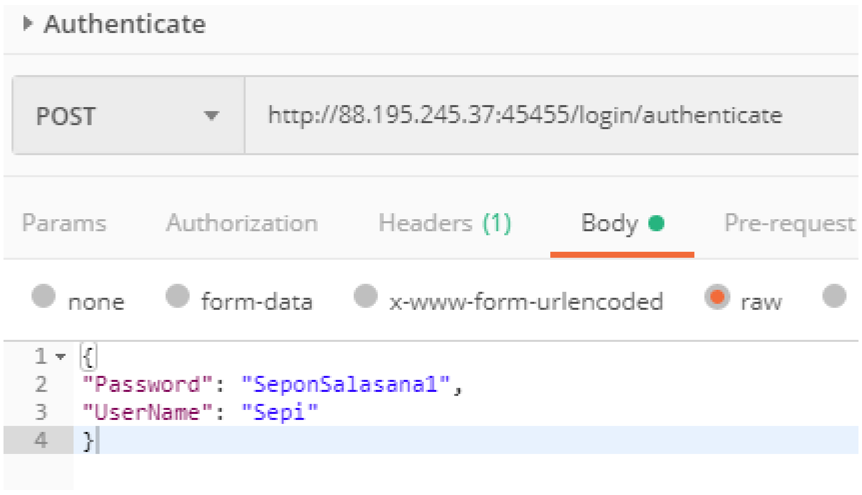
Conveyor by Keyoti

Local URL	Remote URL	Internet URL
http://localhost:59652/	http://88.195.245.37:4545	
http://localhost:59652/	https://88.195.245.37:454	

FIGURE 9. Conveyor by Keyoti

Conveyor by Keyoti was used on testing the application with an Android device. This way the author could replicate how the application would work with a mobile device.

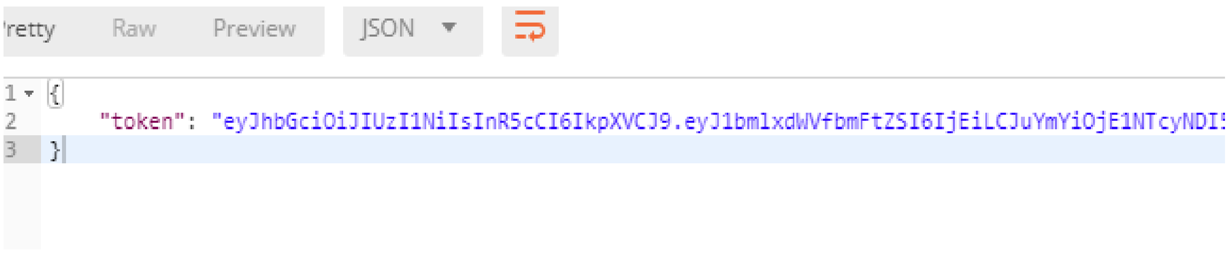
Testing the calls was done mostly with Postman. Figure 10 shows an example of testing the authentication call for the application. The client sends a JSON with Password and UserName values and gets a JWT Token in return and a status code 200 which means ok. Part of the JWT Token is shown in Figure 11.



http://88.195.245.37:45455/login/authenticate

```
{
"Password": "SeponSalasana1",
"UserName": "Sepi"
}
```

FIGURE 10. Authentication call with Postman



```
{
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1bmlxdWVfbmFtZSI6IjEiLCJuYmYiOjE1NTcyNDI
}
```

FIGURE 11. Part of the token returned from the server

4.5.2 Unit testing

xUnit.net was the framework used for creating unit tests and testing the application server. The xUnit project was created in the same solution as the application server project. There were some problems when creating the unit tests with xUnit regarding the versions of the .NET Core libraries used in the project. After rebuilding and installing the same versions the tests were able to run. The author also had a problem with the path variable in Windows 10. Visual Studio would not allow the tests to run in Visual Studio. There was an error about there being illegal characters in path. The tests were able to run with Command Prompt using the command “Dotnet test” in the test project directory though. Later the author found out that the error was caused by quotation marks in the environment variable path in Windows 10. After removing them the author was able to use Visual Studios Test Explorer to run the tests.

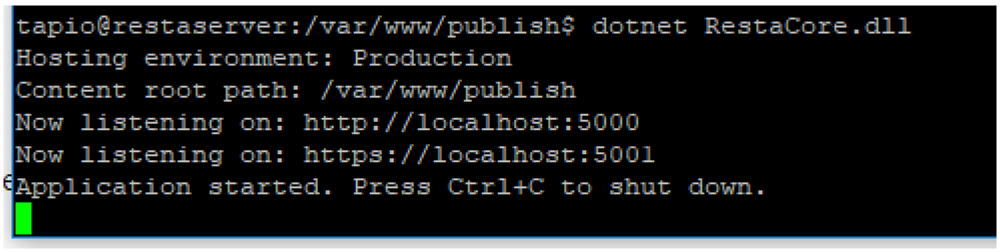
Creating unit tests with xUnit was easy at least for the smaller parts of the project. Mocking some parts was difficult. Mocking in programming means creating objects that replicate the real objects used in the software. Creating tests for parts of the controller that for example authenticate is hard because the developer would need to mock so much data. Testing such parts is integration testing which the author was not going to create for the project during the work on this thesis.

Entity Framework Core's InMemory library was used to test controller methods without establishing a connection to the real database. It helps making the unit tests run faster during development.

4.6 Ubuntu Server 18.04

On the Ubuntu server the author installed and configured Nginx and MySQL database.
A remote connection was established with MySQL Workbench to the database in Ubuntu Server. It made configuring the database faster and easier. To achieve this, the author first enabled SSH connections on the Ubuntu Server and then established a connection with MySQL Workbench.

The project for the application server was published as a self-contained deployment which means that that there is no need to install the .NET Core runtime and .NET Core libraries separately. In Visual Studio you can create a publish version quickly. There was some issue with publishing with .Net Core version 2.1.x because the TargetFramework attribute in the projects .csproj file was wrong because of updating the .Net Core version during development. After moving the self-contained deployment to the Ubuntu Server, it was able to run using dotnet command with the dll file as shown in Figure 12.



```
tapio@restaserver:/var/www/publish$ dotnet RestaCore.dll
Hosting environment: Production
Content root path: /var/www/publish
Now listening on: http://localhost:5000
Now listening on: https://localhost:5001
Application started. Press Ctrl+C to shut down.
```

FIGURE 12. Running the published project on Ubuntu Server

During the project the database, the application server and the web server were on the same virtual machine but in the future, the author was planning to first separate the database from the web server and application server. Security and performance are reasons to separate them. Testing the project in Ubuntu Server was done with Postman as well.

4.6.1 Nginx

Nginx was setup on the ubuntu server as the web server. Installing and setting up Nginx was easy and fast. Only some configuration was required to get it up and running.

4.6.2 MySQL

MySQL database was set up on the ubuntu server as the database. On the Ubuntu Server the author allowed remote connections to the MySQL server so he could use MySQL Workbench to configure the database.

5 CONCLUSION

First, the group was going to create a mobile application with React Native. A backend for the React Native mobile application was going to be implemented with Firebase. After developing awhile, it was decided to stop the development with it due to various problems regarding the framework. React Native was not a working framework for this project at least, but the team behind React Native is still working on it so it might become more stable in the future.

After React Native the group started development with Xamarin.Forms. The author would implement a backend with Azure. A decision was made to change from Firebase to Azure because it would be better suited with Xamarin.Forms. There was also more documentation regarding Xamarin.Forms working with Azure instead of Firebase.

After developing with Xamarin.Forms and Azure the group decided that they should implement their own backend for the application because it would become cheaper and it would be a good learning experience as well. The author would be responsible for designing and developing the backend. The author chose to develop with .NET Core as the application server and MySQL as the database. This .NET Core project was the final project that was developed for this thesis.

The project was a great learning experience for me because I got the chance to get to know some new technologies and how they work. I also discovered some new useful tools for me to use in the future.

The purpose of this application was the thesis and, in the future, to try and start a business with it. The group has spent a lot of time developing this project without customers and have only gone forward with some ideas on how to further develop this project. The group has made the decision to not spend so much time developing the application anymore, instead to focus more on attracting customers for them and only if they acquire a customer, they would focus on development again.

It is a common problem for developers when working on their own projects that they focus too much on the product and not on the other important aspects of creating a profitable product.
In the future when working on such projects I have made the decision to only first create smaller prototypes for showcasing to the potential customers.

6 REFERENCES AND SELF-PRODUCED MATERIALS

CHRZANOWSKA, Natalia. March 7, 2019 . React Native Pros and Cons - Facebook's Framework in 2019 (update) [Referenced 2019-03-26.]

Available: https://www.netguru.com/blog/react-native-pros-and-cons

EISENMAN, Bonnie. October 23, 2017 Bonnie Eisenman. Learning React Native [Referenced 2019-03-24.]

Available: https://www.oreilly.com/library/view/learning-react-native/9781491929049/ch01.html

FACEBOOK. n.d. Who's using React Native? [Referenced 2019-03-26.]

Available: https://facebook.github.io/react-native/showcase

GIT. n.d. git –local-branching-on-the-cheap [Referenced 2019-03-22.]

Available: https://git-scm.com/

MICROSOFT. n.d. Visual Studio best-in-class tools for any developer [Referenced 2019-03-22.]

Available: https://visualstudio.microsoft.com/

MICROSOFT. n.d. What is ASP.NET Core? [Referenced 2019-03-22.]

Available: https://dotnet.microsoft.com/learn/web/what-is-aspnet-core

MYSQL. n.d. 1.3.1 What is MySQL? [Referenced 2019-03-24.]

Available: https://dev.mysql.com/doc/refman/8.0/en/what-is-mysql.html

MYSQL. n.d. MySQL Workbench Enhanced Data Migration. [Referenced 2019-03-22.]

Available: https://www.mysql.com/products/workbench/

NGINX. n.d. Welcome to NGINX Wiki! [Referenced 2019-03-24.]

Available: https://www.nginx.com/resources/wiki/

POSTMAN. n.d. Postman Simplifies API Development [Referenced 2019-03-22.]

Available: https://www.getpostman.com/

RAJAT, S. Dec 28, 2017 . Introduction to Firebase. [Referenced 2019-03-27.]

Available: https://hackernoon.com/introduction-to-firebase-218a23186cd7

STONE, Sam. Dec 22, 2017 . Pros and cons of Xamarin Native and Xamarin Forms development—from a native viewpoint. [Referenced 2019-04-17.]

Available: https://medium.com/@samstone/pros-and-cons-of-xamarin-native-and-xamarin-forms-development-from-a-native-viewpoint-65a06148582

SQLDBM. n.d. SQL Database Modeler. [Referenced 2019-03-22.]

Available: https://sqldbm.com/Home/

TELERIK. Feb 21, 2018 . What is Xamarin.Forms?. [Referenced 2019-04-17.]

Available: https://www.telerik.com/blogs/what-is-xamarin-forms

THOUGHTBOT. May 31, 2018 . Examining performance differences between Native, Flutter, and React Native mobile development. [Referenced 2019-03-26.]

Available: https://thoughtbot.com/blog/examining-performance-differences-between-native-flutter-and-react-native-mobile-development

WALLEN, Jack. March 22, 2017. Ubuntu Server: A cheat sheet [Referenced 2019-03-22.]

Available: https://www.techrepublic.com/article/ubuntu-server-the-smart-persons-guide/