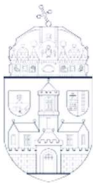

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

JOHN VON NEUMANN
FACULTY OF
INFORMATICS

THESIS

OE-NIK	Name of the student:	**Varga Dávid**
2023	Identification number of the student:	**T/006674/FI12904/N**

Óbudai University
John von Neumann Faculty of Informatics
Institute for Cyber-physical Systems

THESIS WORK SPECIFICATION

Name of the student:	**Dávid Varga**
Identification number of the student:	T/006674/FI12904/N
Neptun's code:	[redacted]

Title of the thesis:

Food supply chain innovation with modern IT solutions
Élelmiszer-ellátási lánc innovációja modern informatikai megoldásokkal

Internal supervisor	Dr. Eszter Balázsné Kail
External supervisor:	Árokszállási Ákos
Deadline:	15 December 2022
Subjects of the final exam:	Computer Architectures Cloud service technologies and IT security specialization

The task:

The length of food supply chains is more important than ever with rising awareness of the environmental load of food production and transport. The pandemic also made the world realize that locally produced goods are the least affected by changes in the global market and shipping. With constantly rising shipping and transport costs, new alternatives should be provided. The widely used smart devices and the current development of IT technologies can make it possible to connect local food producers to local consumers.

The goal of the thesis work is to get an insight into the most widely used web application development technologies during the research and to get a deeper knowledge and experience with the selected technology stack for implementing the application.

The task of the student is to specify, plan and implement an e-commerce web application that provides a platform for smallholders to sell their produced food.

The thesis must contain:

- examination of similar e-commerce web applications,
- investigating the most widely used technologies for implementing the web applications (e.g., ASP.NET Core, Angular, Blazor),
- development of the specification of the system, including the application architecture and database schema design,
- investigation of hosting a web application (e.g., self-hosted, cloud-hosted),
- implementation of a web application including a front end, back end, and database,
- providing the future work plan and future development possibilities.

Place of stamp

.....
Dr. Rita Fleiner
head of institute

This specification is valid until: **15 December 2024**
(According to OE TVSz 55.§)

I consider the thesis suitable for submission:

.....
external supervisor

.....
internal supervisor

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Faculty of Informatics

STUDENT'S DECLARATION

I the undersigned student hereby declare that this degree project / thesis is the result of my own work; I have disclosed the references and tools used in an identifiable manner. The results indicated in my degree project / thesis completed may be used by the university and the institution announcing the task for their own purposes free of charge.

Dated: Budapest, 14 December 2023

Vajda Dávid

student's signature

CONSULTATION LOG

Student's name:	Neptun code:	Specialty:
Varga Dávid	[redacted]	Full-time
Phone number:	Mailing address:	
[redacted]		

Degree project / thesis[1] title in Hungarian:

Élelmiszer-ellátási lánc innovációja modern informatikai megoldásokkal

Degree project / thesis[2] title in English:

Food supply chain innovation with modern IT solutions

Institutional supervisor:	External supervisor:
Dr. Eszter Balázsné Kail	Árokszállási Ákos

Please write the data in printed capital letters!

Occ.	Date	Content	Signature
1.	2022-04-20	General guidance for the thesis.	Kail
2.	2022-04-27	Literature research	Kail
3.	2022-05-04	Literature research	Kail
4.	2022-05-11	Final suggestions, form of the thesis.	Kail

The Consultation log is required to be countersigned by either supervisor on a total of 4 occasions of consultation.

The student has complied with the requirements of the subject titled Degree project I / Degree project II (BSc) or Thesis 1 / Thesis 2 / Thesis 3 / Thesis 4[3], and is allowed to proceed for reporting / defense[4].

Dated: Budapest, 05.12. 2022

Kail

.....
institutional supervisor

[1] Underline as appropriate
[2] Underline as appropriate
[3] Underline as appropriate
[4] Underline as appropriate

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

John von Neumann Faculty of Informatics

CONSULTATION LOG

Student's name:	Neptun code:	Specialty:
Varga Dávid	[redacted]	Full-time
Phone number:	Mailing address:	
[redacted]		

Degree project / thesis[1] title in Hungarian:

Élelmiszer-ellátási lánc innovációja modern informatikai megoldásokkal

Degree project / thesis[2] title in English:

Food supply chain innovation with modern IT solutions

Institutional supervisor:	External supervisor:
Dr. Eszter Balázsné Kail	Árokszállási Ákos

Please write the data in printed capital letters!

Occ.	Date	Content	Signature
1.	2022-09-30	General guidance for the thesis II.	Kail
2.	2022-10-14	Implementation and documentation of the application.	Kail
3.	2022-11-04	Implementation and documentation of the application.	Kail
4.	2022-11-21	Final suggestions, referencing the sources.	Kail

The Consultation log is required to be countersigned by either supervisor on a total of 4 occasions of consultation.

The student has complied with the requirements of the subject titled Degree project I / Degree project II (BSc) or Thesis 1 / Thesis 2 / Thesis 3 / Thesis 4[3], and is allowed to proceed for reporting / defense[4].

Dated: Budapest, 12.13. 2022

Kail

.....
institutional supervisor

[1] Underline as appropriate
[2] Underline as appropriate
[3] Underline as appropriate
[4] Underline as appropriate

CONTENT

Abstract

Sustainable and environmentally friendly food production and transport is starting to become more important for consumers. With the recent events, food supply chain problems, food supply shortages and rising energy prices, a possible solution could be seasonally produced food, sold and delivered in the local area, cutting the long and expensive supply chain.

The aim of the thesis work is to plan and implement an e-commerce web application, where farmers can sell their products to local customers.

1. Introduction

The modern world is facing a rising challenge of the environmental impact of food production and transportation. Recent changes also caused increased energy prices which heavily affect the prices of food which are imported or shipped from far away. The sustainability of these types of food procurement is currently highly disputed.

One possible solution to this problem could be locally produced and sold food. Food that is seasonal travels a minimal distance to the end consumer and is in harmony with the local ecosystem providing more biodiversity.

With the widespread usage of smart devices and applications that intend to make everyday life easier, food selling, and ordering e-commercial businesses have risen too. These applications try to provide a solution for buying groceries from home, option for home delivery, supply goods which might not be available in traditional grocery shops like GMO-free and bio meet, unpasteurized homemade dairy products and pesticide free vegetables produced locally.

The idea for my thesis came to me reading articles about the food production and supply chain problems in the face of the pandemic and quickly rising energy prices. I want to provide a possible solution for the problems mentioned above. The application I will develop will be able to connect local food producers to local end-consumers. I will build an e-commerce website for this purpose where food producers can create an account and sell their goods. Customers will also be able to create accounts and browse the available food supply. The distance of the sold product and the customer will be displayed but this will be available only for registered users.

2. Examination of e-commerce web applications

2.1. Facebook Marketplace

Facebook Marketplace is an e-commerce application developed by Facebook which provides a platform for individuals and small businesses to sell their goods, mostly locally. It was introduced in 2007 and its greatest benefit is that it is easy to use and practically anyone can sign up for it for free with a Facebook account. Facebook also provides the possibility for individuals and businesses to advertise their products, which they want to sell, to generate revenue from the platform. It is a fast-growing platform with more than 1 billion users. The Marketplace can be found inside the Facebook app. The app is available on multiple platforms such as desktop (via an internet browser) and mobile (Android, IOS, Windows Phone).

2.1.1. Facebook's client-side

The main Facebook web platform was launched as a simple server-rendered PHP website back in 2004. As Marketplace is part of the Facebook app, the technology stack is the same as the app's. For the front end Facebook uses React, which is a JavaScript library for building user interfaces, with CSS and HTML. HTML is responsible for the structure and CSS is for the style of the website. Relay is also an important component of the front end, it is the GraphQL client. GraphQL is a query language and server-side runtime for application programming interfaces (APIs), developed also by Facebook, that focuses on providing clients with only the information they need. [1]

Incremental code download is used by the app which results in faster performance. The loading of the main page is divided into multiple tiers. In tier 1, a skeleton is rendered for the components. Tier 2 includes all the JavaScript needed to fully render all above-the-fold content. Tier 3 comprises anything that is only required after the display and does not affect the current pixels on the screen, such as logging code and live-updating data subscriptions. By implementing this approach, developers managed to decrease the amount of code downloaded at once and improved 'time to first paint' metric.

Web pages usually have to wait until all of the JavaScript is downloaded and executed before the app can query data from the server. Relay makes it possible to what data the page needs.

This implies that as soon as the server receives a page request, it can begin preparing the appropriate data and downloading it simultaneously with the required code. [1]

2.1.2. Facebook's server-side

Facebook was built with the LAMP stack shown in Figure 2.1, which refers to Linux, Apache, MySQL, and PHP. The LAMP stack is a widely used open-source solution stack for web development. The components in the LAMP stack have the following roles:

- Linux is an open-source operating system which is used to host and run the other components.
- The Apache HTTP Server is a cross-platform web server that is free and open-source.
- MySQL is a relational database management system that is mostly used to create and administer web databases.
- PHP is a server-side scripting language that is embedded in HTML.

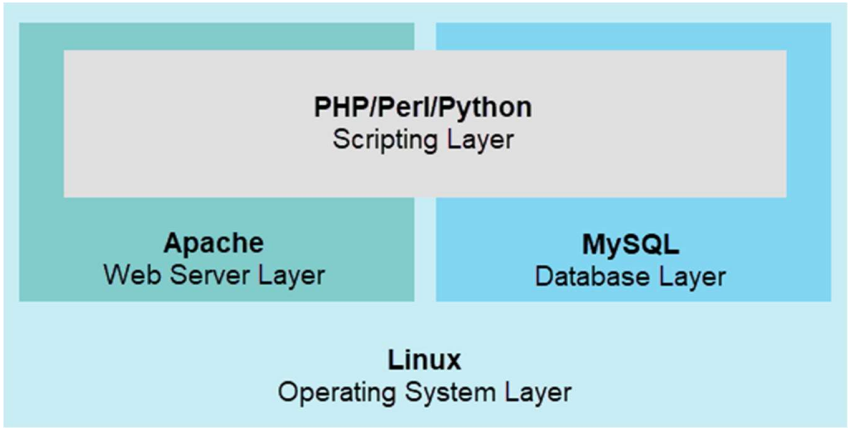

Figure 2.1: LAMP stack [2]

As technological challenges arised, they had to respond with changes in their technology stack usage. As an example, Facebook continues to use PHP, but it has developed a compiler for it so that it can be converted to native code on its web servers, resulting in improved performance. [3]

2.2. Marketwagon.com

Market Wagon is an online local food market for farmers to sell end local consumers to buy food, launched in 2016. Their mission is to create an online farmers market with home delivery to give consumers more access to local food. Farmers and artisans can advertise and sell their products online with the ease of e-commerce thanks to their technological and logistical advances. Their business experienced a fast growth during the pandemic because farmers and local food producers tried to find new ways to get their products to consumers.

Shopping as a consumer comes down to four simple steps:

1. The customer has to set the location.
2. Local products will be displayed based on the previous setting.
3. Farmers and artisans send their products to local hubs operated by Market Wagon.
4. Products are delivered to the customer's doorstep by Market Wagon.

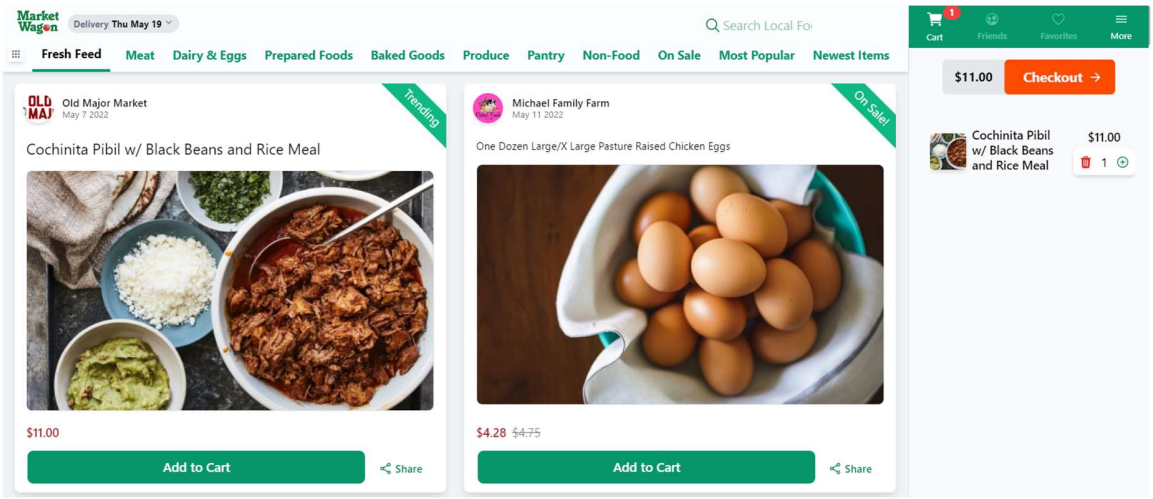

Figure 2.2: Market Wagon app [4]

Market Wagon has a browser app seen in Figure 2.2. The user interface is very clean and intuitive, with just the needed number of buttons and menu items, to maximize the user experience without overwhelming the customer with too many choices on actions. After selecting the location, products are listed in the app for buying. The items can easily be added to the cart by simply clicking on the "Add to Cart" button. The number of products can be adjusted by clicking on the particular item and selecting the quantity. In that popup window,

additional information and actions are available. The consumers can subscribe to the supplier itself indicating regular purchases. They also have the option to write reviews and a Q&A section is available too, shown in Figure 2.3.

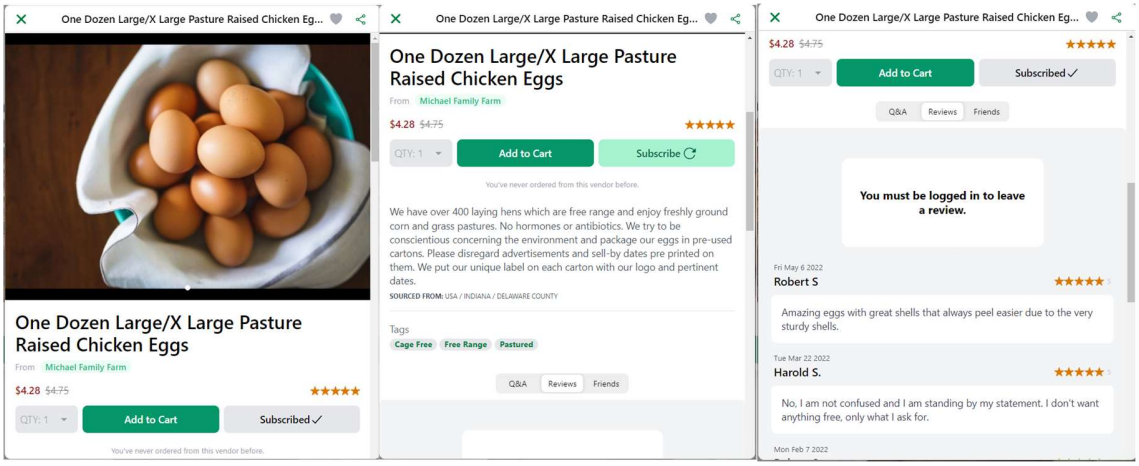

Figure 2.3. Market Wagon product details [4]

To use the application, registration is mandatory. They offer weekly delivery, therefore the desired products have to be ordered by Tuesdays. Delivery takes place on Thursday. Two delivery methods are provided, delivery to home and delivery to a market host.

My overall impression of Market Wagon is that their service is highly demanded with tremendous potential. They have the possibility to connect more and more farmers and consumers and help build local communities and promote conscious shopping.

2.3. Kifli.hu

Kifli.hu is an online grocery store which was launched in 2019. Their main service is grocery order compilation from their warehouse and home delivery. Their online shop has a wide variety of products ranging from groceries to household goods. They promise a very short deadline for home delivery, which gives them a market advantage over the competition.

For using their service, the customer must create an account. Setting the location information is mandatory. A wide variety of items can be found in their web shop, organized into simple categories. Items can be easily added to the cart by clicking the “A kosárba” button. After

clicking, the button changes into a counter where the amount can be set. Their delivery deadline is very short, currently they promise to complete the order in three hours. [5]

2.4. Zoldtanya.hu

Zöld Tanya is a Hungarian community organic garden. They produce bio vegetables and fruits on a seasonal basis. Their services include weekly food delivery which is available on a subscription basis or a simple one-time purchase. In their online store multiple subscription types are available which vary in price depending on the length of the subscription and the amount of food ordered. The delivery system is door-to-door which means that the supply chain is short, and the food producer transports the goods to the end-consumers. [6]

2.5. Conclusion

In my opinion, the previously presented enterprises are a great example of successful e-commerce shops. Facebook Marketplace is an easy-to-use and free platform for selling and buying any kind of items. Market Wagon excels at simplicity while providing an in-demand farmer and customer connecting service and product delivery system. Kifli offers short-notice food delivery and Zöld Tanya is optimized for subscription-based ordering. A successful e-commerce application must incorporate most of the above-mentioned best elements of these businesses.

3. Web application concepts and architecture

3.1. Client-side

In web development, the term ‘client-side’ or popularly called ‘front end’ refers to everything that is displayed or takes place on the client (end user device). This comprises what the user sees, such as text, graphics, and the rest of the user interface, as well as any activities performed by a program within the user's browser. As of late, developers are including client-side processes in their application architecture and moving away from doing everything on the server side. [7]

3.2. Server-side

Everything that happens on the server rather than on the client is referred to as server-side. Previously, practically all business logic was performed on the server, including dynamic webpage rendering, database interaction, identity authentication, and push notifications. [7]

3.3. Client-server model

The client-server model, shown in Figure 3.1, is a distributed application structure that divides jobs or workloads between servers and clients who requested that service. Clients and servers communicate over a computer network on separate hardware most of the time, but they can share a system. A server host is the computer that executes one or more server programs that share resources with clients. A client typically does not share any of its resources, but instead asks a server for data or services. As a result, clients start communication sessions with servers, which then wait for incoming requests. [7]

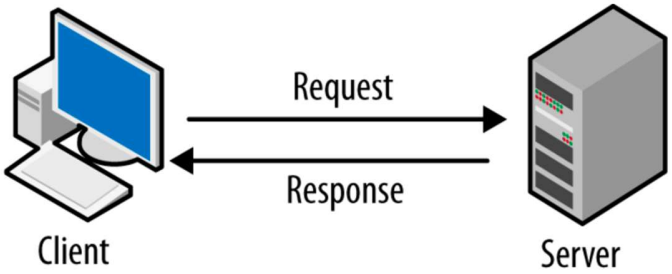

Figure 3.1: Client-server model [7]

3.4. What is an API?

API stands for application programming interface. It is a type of software interface, offering a service to other pieces of software, shown in Figure 3.2. After reading the documentation of the API, programmers can utilize the provided functionalities of the API to facilitate the development of their program. APIs greatest power is that the programmers do not have to know the exact inside implementation details of the interface to be able to understand and properly use the APIs. They are also programming language independent therefore any client should be able to call an API. [8]

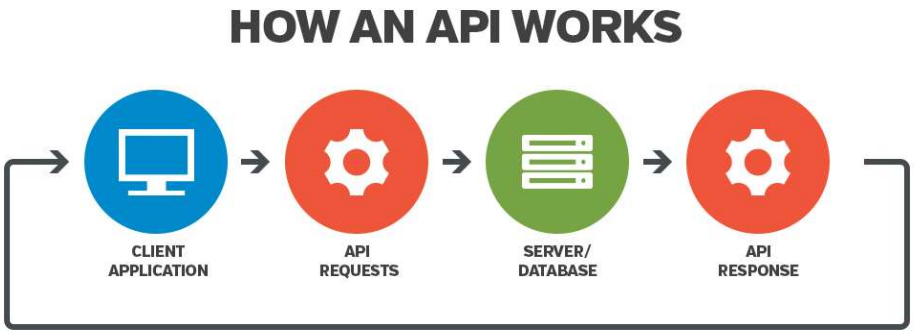

Figure 3.2: API workflow [9]

3.5. REST API

REST is also an acronym which stands for Representational State Transfer. REST was introduced in 2000 by Roy Fielding, and it is an architectural approach, not a protocol or a standard, to designing web services. The most common REST applications use the HTTP protocol, but REST is completely protocol independent. [10] [11]

In order for an API to be considered RESTful the following requirements must meet:

- Stateless client-server communication
- Client-server architecture
- Cacheable data
- Layered system
- Uniform interface

- Code on demand (optional)

3.6. Server-side rendering

The capacity of an application to turn HTML files on the server into a fully rendered HTML page for the client is known as server-side rendering (SSR). The web browser sends a request for information to the server, which answers immediately by delivering the client a completely rendered page. An advantage is of server-side rendering, that the server does the bulk of the work, the client's browser only displays the output. This leads to its disadvantage, which is if the client changes something in the page, the browser will ask for the page again and the server will re-render everything which can cause a high load. [12] [13]

3.7. Client-side rendering

With client-side rendering, a single HTML file is requested, without any content. After all the JavaScript code is fetched and compiled, the browser will request data from the server. This is a relatively new method of rendering websites, and it did not gain traction until JavaScript libraries began to include it in their development methodology. With client-side rendering, the burden on the server is much lower compared to server-side rendering. It is also faster after the first rendering and initial load. On the other hand, because much of the work is done on the client's device, it can cause problems if the client's hardware is weaker, or the browser is not up to date. [12] [13]

4. Overview of the most widely used technologies used for implementing e-commerce applications

A vast number of server and client-side technologies are available as of nowadays. Because the scope of this thesis work is limited, I intend to give a brief overview of .NET technology stack for application development purposes. It is one of the most used technology stacks with extensively documented tools and framework.

4.1. What is .NET?

.NET is a free, cross-platform, open-source developer platform. .NET has more implementations like .NET Framework for windows based and .NET Core for cross-platform based development. After .NET 5 was introduced, these implementations were merged and are included in the main implementation. [14]

4.2. ASP.NET Core

ASP (Active Server Pages) .NET Core is a popular web development framework. It is open-source and free-to-use. It has a built-in support for dependency injection and is easily integrated with many popular client-side frameworks. [15]

4.3. Entity Framework Core

Entity Framework Core, EF Core in short, is an object-relational mapping framework. It has the same benefits mentioned above such as being open-source and platform independent. It allows developers to work with data using domain-specific objects rather than the underlying database tables and columns where the data is kept. When working with data, developers can use EF Core to work at a higher level of abstraction, allowing them to construct and manage data-oriented applications with less code than traditional programs. [16]

Data access in EF Core is done through a model. Entity classes and a context object that represents a database session make up a model. Data can be queried and saved using the context object.

The Code-First and Database-First development techniques are supported by EF Core. EF Core is primarily focused on the code-first approach, with little support for the database-first

strategy. With the code-first method, EF Core API builds the database and tables using migration depending on the conventions and settings specified in the domain classes, while with the database-first approach, it uses EF Core commands to generate domain and context classes depending on the existing database-schema. [17]

4.4. LINQ

LINQ (Language Integrated Query) is a standard query syntax for retrieving data from many sources and formats in C#. EF Core is used with LINQ for querying and manipulating data. The developer does not have to know the database specific language to work with, because LINQ automatically translates to the database language as shown in Figure 4.1. [18]

Figure 4.1: LINQ data model [19]

5. Comparison of web front end frameworks

5.1. Angular

Angular is an open-source, JavaScript framework written in TypeScript. Angular includes a component-based framework for developing scalable web applications, as well as a collection of well-integrated libraries that cover a wide range of features, such as routing, form management, and client-server communication, as well as a set of developer tools for developing, building, testing, and updating code.

A component-based architecture is followed by Angular. In a component-based architecture, a complex application is decoupled into functional and logical components. Because these components are reusable, they can be used throughout the application. This architecture makes Angular code highly testable because these components are autonomous and can be tested independently. [20] [21]

5.2. Blazor

Blazor is a free and open-source web framework that allows programmers to create web applications using C# and HTML. Microsoft oversees its development. Blazor allows us to create interactive web user interfaces without having to utilize JavaScript. C# code can be run on the server as well as in the client's browser. Browsers understand and execute only JavaScript, but with the help of WebAssembly, they can run C# code in the same security sandbox as JavaScript frameworks like Angular. WebAssembly is based on open web standards. As a result, it comes equipped with all modern browsers, including mobile browsers. This means that no further plugins are required for the Blazor application to function. [22]

5.3. Angular vs Blazor

The main difference is that Angular uses a Typescript, a strongly typed version of JavaScript, whereas Blazor uses C#. Based on programming languages, JavaScript is a well-known programming language among front end web developers, so it makes sense for them to use Angular. On the other hand, if a back end developer has to work in the domain of front end, it is not needed to learn a completely different programming language, like JavaScript, to

build the front end, instead Blazor can be used with HTML and C# which is originally a back end oriented programming language.

Angular supports client-side rendering by default. It is possible to add Angular Universal for server-side rendering support. Blazor also supports both client-side rendering with the help of WebAssembly and client-side rendering using ASP.NET Core.

Comparing their maturity, Angular was released in 2016 while Blazor in 2018 but only gained more reputation recently with the release of .NET 6. Angular is more widely known and used than Blazor, one of the reasons is that it has been around for a longer time. Blazor, on the other hand, is very promising but is still evolving. [23] [24]

6. Web application hosting and security

In this section I intend to examine, give a brief comparison and overview of possible web application hosting ways.

6.1. Cloud hosting

6.1.1. Cloud computing deployment models

Private cloud infrastructure exists on a private network and is managed by the organization in its internal enterprise data center or by the cloud provider and may reside on-premises or off-premises. It's safer since only the company that owns it has access to and control over the service delivery settings. It seeks to address data security problems and provides greater control, but it does not deliver benefits such as lower capital and operational expenses.

Public cloud infrastructure is designed for wide-ranging groups or public, owned and managed by a cloud provider. On a pay-per-use basis, the resources are dynamically delivered on demand. It offers a variety of advantages to its customers, including scalability, location independence, flexibility, and no upfront infrastructure investment. Another typical subscription type is fixed priced where the user pre-pays a custom amount of money for a limited number of services or hardware usage.

A hybrid cloud infrastructure is a collection of clouds connected by standardized technology to share data and applications regardless of ownership or location. It provides more flexibility and control over the application by combining the advantages of each while also addressing the limits.

Community cloud infrastructure is intended for usage by several organizations within a single community with shared interests. The visual representation of the mentioned deployment models can be seen in Figure 6.1. [25]

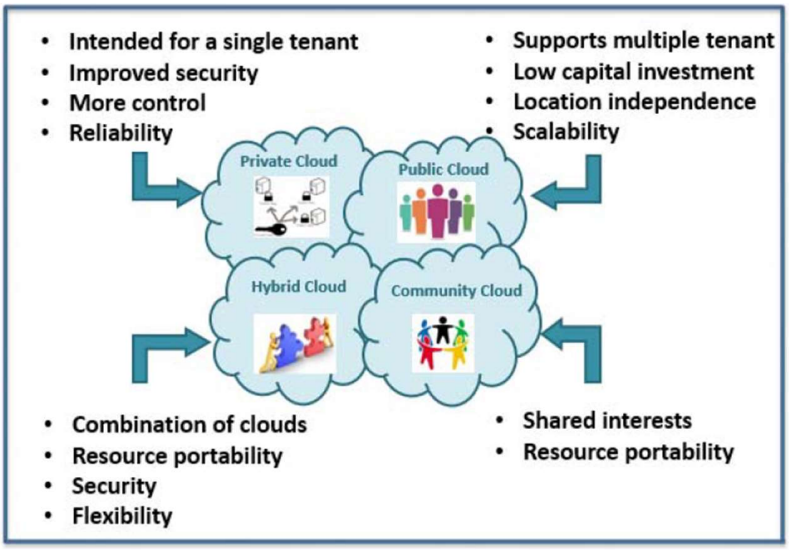

Figure 6.1: Cloud computing deployment models [25]

6.1.2. Cloud computing service models

Software as a Service (SaaS) is the top layer, where service providers' programs are remotely deployed on customers' devices and accessed through interfaces without having to manage the underlying infrastructure. Applications provided by SaaS providers must be secured utilizing various ways such as WS (Web Service) security, XML (Extensible Markup Language) encryption, Secure Sockets Layer (SSL), and other data protection methods.

Platform as a Service (PaaS) is the middle layer, where the consumer can design and deploy his own apps despite having no control over the underlying infrastructure or computing resources. PaaS providers can offer a predetermined combination of application servers and operating systems, such as LAMP (Linux, Apache, MySQL, and PHP).

The bottom layer is infrastructure as a service (IaaS), where cloud providers supply on-demand services such as storage, networks, processing power, and other computational resources. Consumers can deploy and run any software and operating systems they want, paying just for what they use and not for the infrastructure.

The physical resources of the cloud, which are often implemented in data centers, are managed by the hardware layer underneath the stack. Hardware setup, fault tolerance, traffic

management, power and cooling resource management are all possible challenges. Figure 6.2 summarizes the most important properties of each service model. [25]

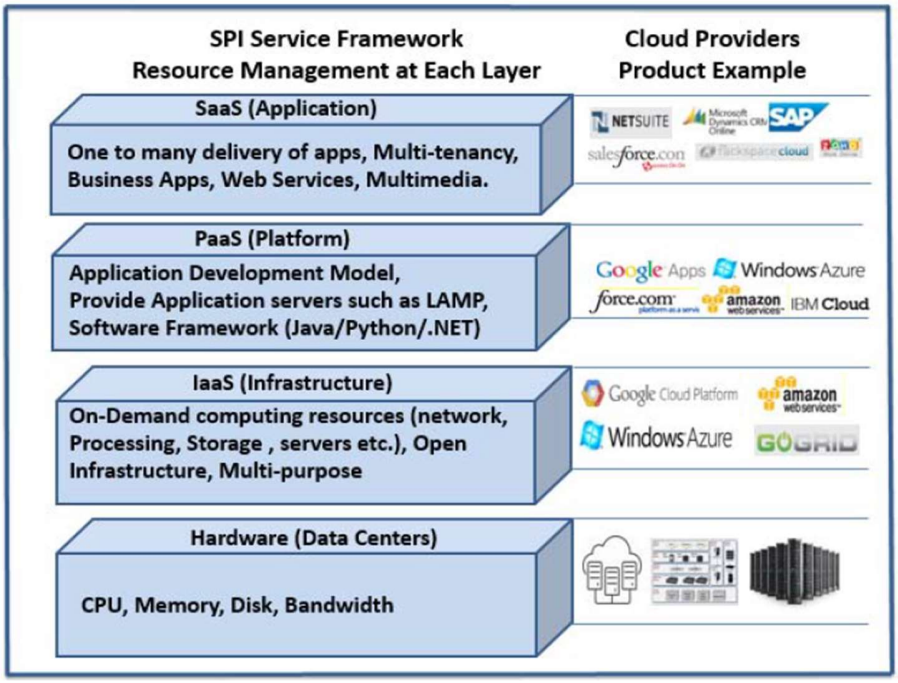

Figure 6.2: Cloud computing service models [25]

6.2. Self-hosting overview

When an organization installs and accesses software from their own server, it is known as self-hosting. On-premises or local hosting are other terms for self-hosting. In an on-premises setup, the hardware and software components are owned and managed by the organization. Self-hosting requires larger upfront capital, because server infrastructure and software product has to be bought and installed. This process can take months. Dedicated employees also must be hired to operate the system. This type of hosting does not scale well, it has to be improved by the organization, which can also take up a lot of resources. Depending on the services the organization offers, the server usage could vary in a wide range, which could mean under or maximum utilization. These changes in the demand for the service and the lack of quick scalability mean it is less cost-efficient than a system that can scale up or down on demand. [26] [27]

6.3. Cloud-hosting overview

Hosting an application in the cloud means that hardware and software components of the hosting system are accessed over the internet, managed, and owned by a third-party organization.

Cloud-hosting has some benefits over self-hosting. There is no need for upfront capital, no hardware or software that has to be bought, everything is provided by the cloud service provider. The resources are scalable on demand. Depending on the service model used, the hosting environment can be configured comprehensively. It could include the hosting operating system, hardware resources, network settings.

On the other hand, shared cloud infrastructure raises some concerns regarding security. Security threats can come from the inside as well as the outside. They can also come from the physical infrastructure, software infrastructure and human infrastructure too. [26] [27]

6.4. Conclusion

Low upfront costs, resource elasticity, and cost savings resulting from economies of scale are all advantages of cloud hosting. When compared to leasing shared infrastructure from the cloud, self-hosting allows more direct control over infrastructure. Obtaining the benefits of cloud infrastructure by passing infrastructure control to a third party, on the other hand, does not have to result in a net loss of security, it may also benefit from scale economies. Cloud providers may afford security measures that would be unaffordable in self-hosting environments, amortizing these costs over a large number of tenants. [25]

7. Specification of the web application

7.1. Chosen technology stack

I chose the .NET technology stack for implementing the web application. It is well-documented and has all the needed components. I also intend to further my knowledge in the .NET domain as it is my current interest and focus. For the front end, Blazor WebAssembly will be used. Its main advantage over the other front end technologies is that it does not require Javascript knowledge, the codebase can be built purely with the combination of C# and HTML. The back end will be implemented with ASP.NET Core Web API. It provides easy-to-use tools to build an API which can communicate with the front end of the application and has been around for many years, so the established official and community knowledgebase is substantial. For data storage, I will use Microsoft SQL Server. It is a free-to-use database management software, compatible with other Microsoft technologies such as Visual Studio and Entity Framework. The chosen object relational mapper is Entity Framework Core. It is a well-established technology and is being used extensively in the software development industry.

7.2. Functional specification

7.2.1. Overview

The application will have a simple intuitive user interface. This will consist of a navigational side bar with menu items and a main area where the content of the selected page will be displayed. The products and product details page will be browsable without a user account but advanced features such as placing and order or viewing the distance will not be available.

If the user creates a customer account, products can be added to the shopping cart by clicking on the product, selecting the amount and clicking the “Add product” button. The items in the cart can be viewed on a separate page after clicking the “Shopping Cart” menu item. When the customer is finished with the shopping, the “Check out” button will have to be clicked. This will conclude the order. Order history will be accessible by clicking the “Order History” menu item.

If the user creates a supplier account, it will be possible to add new products to the website for sale. Suppliers cannot place orders in the application. They will be able to see their sold products with additional information by clicking “My Products” menu item.

7.2.2. Functionalities without a user account

Using the basic functionalities of the application does not require a user account. The customer can browse and search between the products. These functionalities are visually represented in Figure 7.1.

Usable functionalities without a user account:

- Create a new account.
- View the product page.
- View the details of a product.
- Filter the products on the product page based on name.

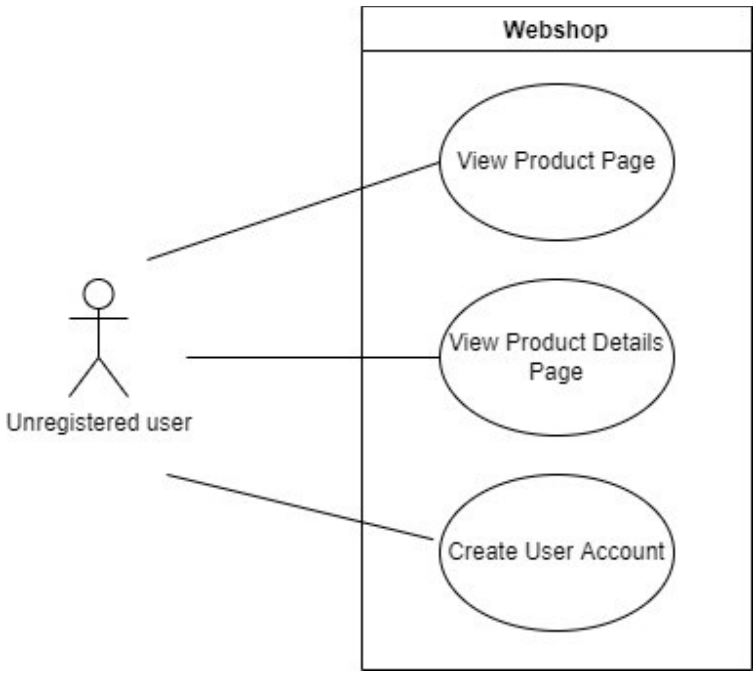

Figure 7.1: Unregistered user use-case diagram

7.2.3. User registration

To use the advanced features of the application a user account is needed. A registration page is available to create an account. On the registration page, the user must fill the following mandatory fields:

- Email address – must be unique
- Password – must contain letters, at least one capital letter, one number and one special character
- First name
- Last name
- Postal code
- City
- Street
- Street number
- Supplier or not

7.2.4. Functionalities with a user account

After registration, the user can log into the application by specifying the registered e-mail address and password through the login page. The available features divide into two categories based on if the user is a customer or supplier. The use-case diagram of these functionalities can be seen in Figure 7.2 and Figure 7.3.

Available advanced features as a supplier:

- Add a new product
- View the added products

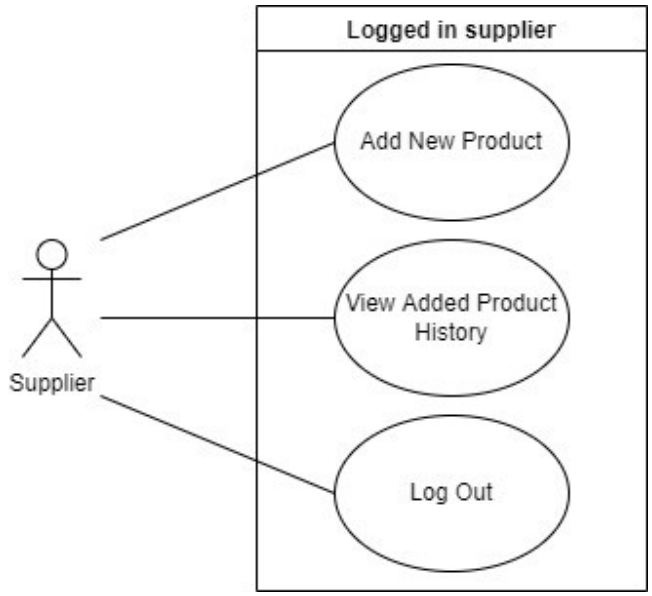

Figure 7.2: Logged in supplier use-case diagram

Available advanced features as a customer:

- Add a product to the “Cart” of the selected product and quantity
- View the “Cart” of the selected items
- Place an order of the “Cart” elements
- View order history

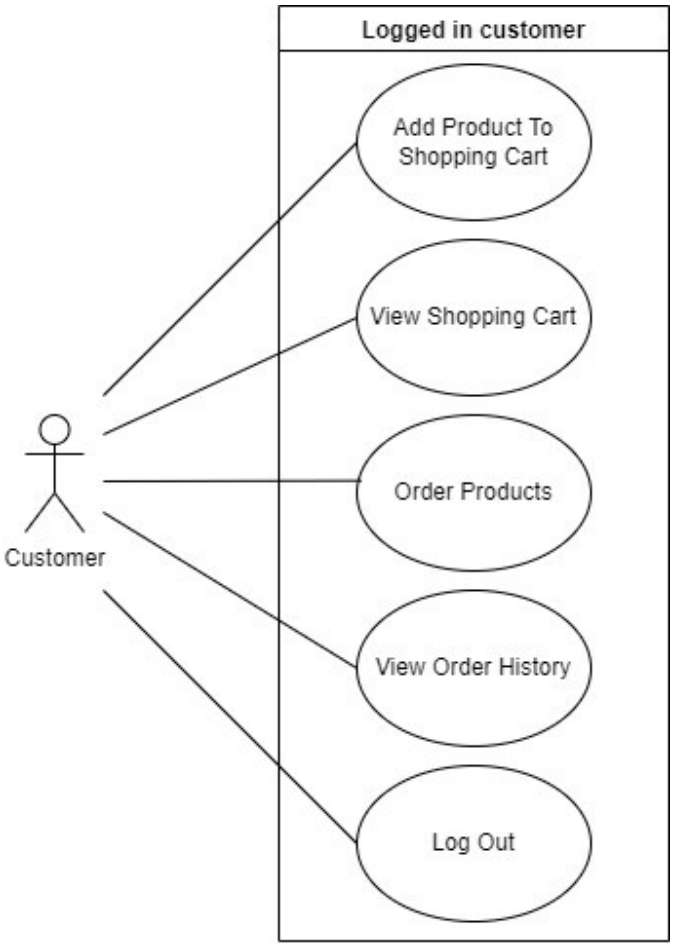

Figure 7.3 Logged in customer use-case

8. Architecture

A common approach in modern application design is to use a multi-layer or n-layer architecture, shown in Figure 8.1, as this ensures that the application is modular and therefore easily changeable, replaceable, and extensible, avoiding the thick client architecture that used to be the result, with a code base that was difficult to interpret and manage.

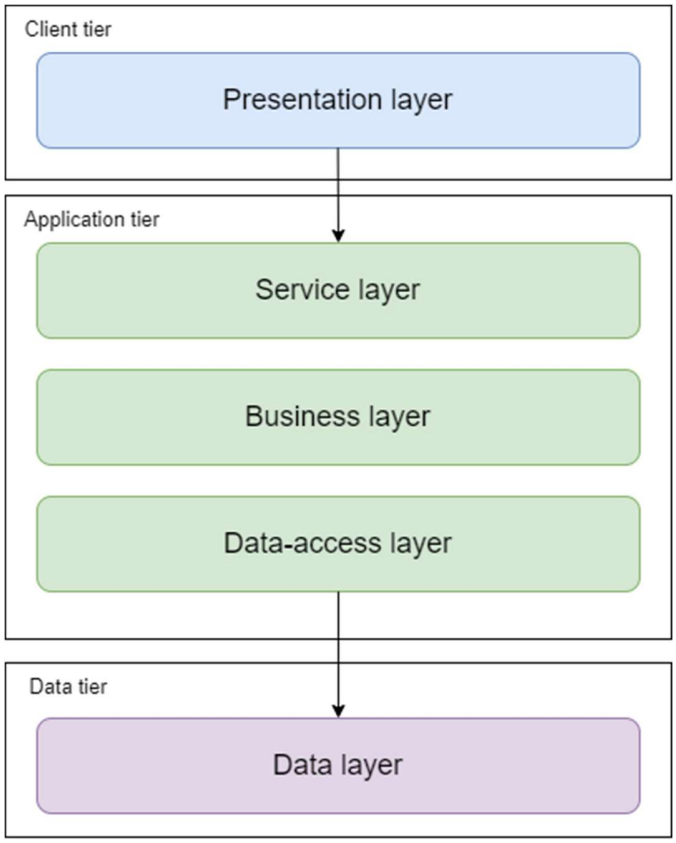

Figure 8.1 Layered architecture

The architecture is divided into three main sections, called tiers. The first tier is the Client tier which is responsible for creating the view that the user sees.

The middle section is the Application tier, which consists of three layers, Service, Business, and Data-access layer. The Service layer represents the services provided by the application which can be accessed by API calls. The next layer is the Business layer, where all the business logic operations happen, including calculation of more complicated actions. The last layer of the Application tier is the Data-access layer. This layer is responsible for data

access and communication between the database and the application. It is usually done with the help of object-relational mapper frameworks.

The last one is the Data tier, which only consists of the Data layer. Its responsibility is to store the data the application needs. This data can take the form of files, file systems, but is most often stored in a database.

9. Implementation

In the section I intend to detail the implementation process of the e-commerce application. I will go through all of the layers and present the challenges I encountered during implementation.

9.1. Data model and data access

9.1.1. Data model

For creating the data model, I used Entity Framework Core Code-First approach. With the Code-First approach, C# classes are created first. These classes represent the to-be-generated database model. After creating the classes with the help of Entity Framework Core the database itself and the tables can be created. It is done via migrations. Migrations allow us to create and modify database tables following the changes in the codebase. To serve the needs of the application, I created the following database table diagram shown in Figure 9.1:

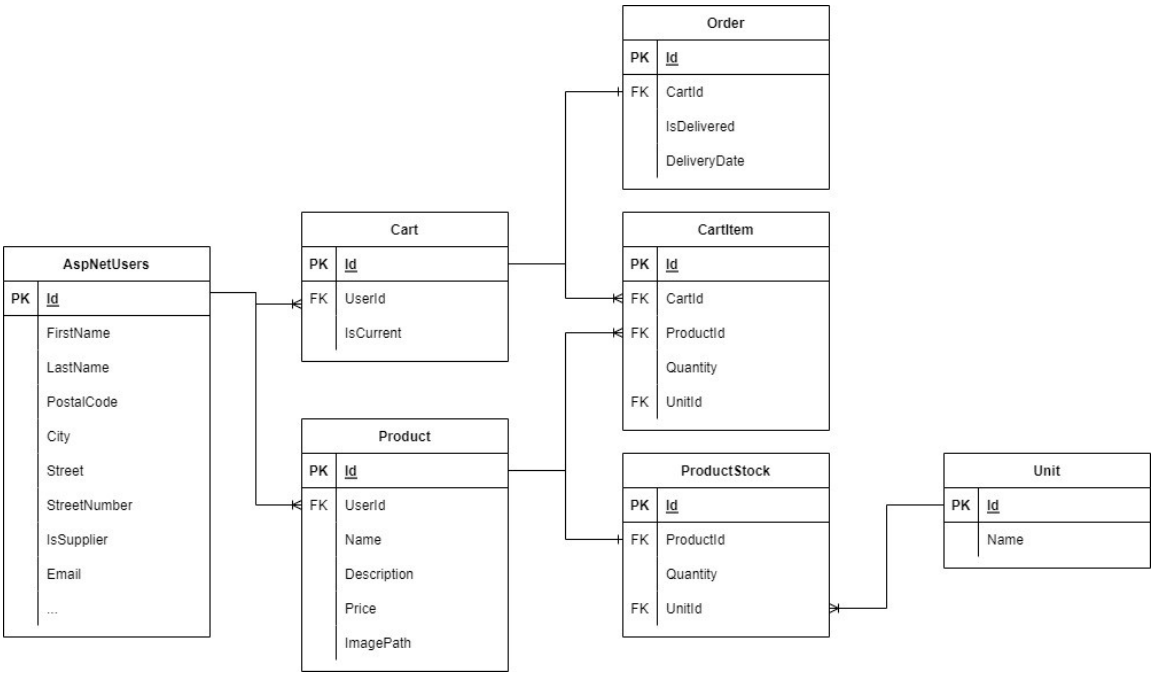

Figure 9.1: Table diagram

The “AspNetUsers” table contains the core information about a user, like the address details which are needed to calculate the distance between a supplier and customer. This table also contains the user’s credentials, which will be explained in detail in the “Security and user management” section. The “AspNetUsers” table connects to the “Products” table with a one-to-many relation because every Product must be associated with a user. The core information of a Product is stored here like “Price” and “Name”. The available quantity and unit which the product is sold in is stored in the “ProductStock” table. With these two tables the system can provide the information about the available products and their prices and quantities.

The other important feature of the application is the shopping cart system. A user can have one active cart at a time, which is represented with the “AspNetUsers” table’s one-to-many relation to the “Cart” table. If a user does not have a cart, or already ordered a new cart record is inserted. Every product a user puts into the cart is inserted into the “CartItem” table. With the one-to-many relation of the “Cart” and “CartItem” tables, a shopping cart can be represented. After a shopping cart is checked out, a new record is inserted into the “Order” table, which represents the orders made by the customers.

9.1.2. External dependencies

To be able to implement the entities in the table diagram and then create and migrate the SQL database and tables, NuGet dependencies were added to the project. NuGet packages are compiled source code usually created by other developers or organizations. The advantage of using these packages is that the developer can use already written and compiled source code and the provided functionalities without having to write a custom solution for the particular problem. The data layer has the following NuGet dependencies:

- Microsoft.EntityFrameworkCore.Design
- Microsoft.EntityFrameworkCore.SqlServer
- Microsoft.EntityFrameworkCore.Tools

These components together provide the possibility to write C# models classes first, which represent the data model, and with a series of commands the database and the tables can be created.

9.1.3. Code-first approach

A *DbContext* implementation has to be created for the Entity Framework in the codebase. The class that implements the *DbContext* contains the entities of the application and can contain other optional configuration settings such as data seed inserted upon database creation or additional entity settings like overriding default table or column names or specifying constraints on columns. My implementation of the *DbContext* class is shown in Figure 9.2.

```
public class FreshFarmsDbContext : IdentityDbContext<User>
{
    6 references
    public DbSet<Cart> Carts { get; set; }
    5 references
    public DbSet<CartItem> CartItems { get; set; }
    1 reference
    public DbSet<Order> Orders { get; set; }
    4 references
    public DbSet<Product> Products { get; set; }
    2 references
    public DbSet<ProductStock> ProductStocks { get; set; }
    1 reference
    public DbSet<Unit> Units { get; set; }
    0 references
    public FreshFarmsDbContext(DbContextOptions<FreshFarmsDbContext> options) : base(options) { }
    0 references
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        base.OnModelCreating(modelBuilder);
        modelBuilder.ApplyConfiguration(new RoleConfiguration());
    }
}
```

Figure 9.2: DbContext

The *DbSet<TEntity>* represents an entity in application and based on the *TEntity* type a database table will be created by Entity Framework Core. As a base class the *IdentityDbContext* was used instead of the default *DbContext* because the application makes use of the Microsoft ASP.NET Core Identity framework which is responsible for the authentication.

Entities can be defined in separate classes following Entity Framework Core conventions. A property named Id will be interpreted as a primary key for the table. The types of the SQL table columns are determined by the types of the properties. Addition restrictions can be placed on properties. Here the *[Required]* attribute is placed on the Name property which defines that is a not nullable column in the database. To represent the connection between the entities, navigation properties are used. In the application one-to-many and one-to-one relations are used. For example, the Unit entity, shown in figure 9.3, has a one-to-many

relation with the ProductStock entity. A navigational property must be placed in both classes to point at each other.

```
public class Unit
{
    1 reference
    public int Id { get; set; }
    [Required]
    4 references
    public string Name { get; set; }

    0 references
    public ICollection<ProductStock> ProductStocks { get; set; }
}
```

Figure 9.3: An entity of the system

To create the database and the tables, an initial migration needs to be generated. This is done by executing specific commands in Visual Studio Package Manager console.

```
Add-Migration -Name <migration-name> -Project <target-project> -StartupProject <startup-project>
```

Running this command, a migration file is created which describes the to-be-created entities and their relationships. After a migration is created, it can be applied to the database with the following command:

```
Update-Database -Project FreshFarms.Data -StartupProject FreshFarms.Api
```

This command updates the database schema. Upon changing the entity models, the previous steps can be performed, and the schema of the will always match with the class representation of the entities. Using this tool, the development process can be much faster compared to the traditional way, which includes creating the database schema first, and then manually creating the matching entities in the codebase.

9.1.4. Data access

DbContext is a crucial component of Entity Framework. A session with the database can be used to query and save instances of the entities to a database using an instance of *DbContext*.

A dedicated *Repository* project is responsible for the CRUD (Create, Read, Update, Delete) operations against the database. Every *Repository* class has an interface which defines the functionalities it has to perform. The Repository classes have the following structure, presented in Figure 9.4.

```
public class OrderRepository : IOrderRepository
{
    private readonly FreshFarmsDbContext dbContext;

    0 references
    public OrderRepository(FreshFarmsDbContext dbContext)
    {
        this.dbContext = dbContext ?? throw new ArgumentNullException(nameof(dbContext));
    }

    2 references
    public async Task<Order> AddOrder(Order newOrder)
    {
        var result = await dbContext.Orders.AddAsync(newOrder);
        await dbContext.SaveChangesAsync();
        return result.Entity;
    }
}
```

Figure 9.4: Structure of a repository class

A *DbContext* instance is injected into the *Repository* instance upon creation. The exposed public methods defined in the implemented interface work with the injected *DbContext* and perform CRUD operations.

9.1.5. Data Transfer Object – DTO

Data Transfer Objects (DTOs) carry data between processes. A business service might need data from multiple database entities or sensitive data that must not be returned to the user, is when DTOs are mostly used. One or more entities can be mapped into DTOs.

9.1.6. Mapping

To create Data Transfer Objects, mapping is required. Mapping can be implemented manually or with the help of auto-mappers. Using auto-mappers can speed up the implementation process, because less code has to be typed manually, but the mapping itself can be slower and mapping complex DTOs is challenging. On the other hand, manual mapping is faster, but the implementation of the mapping functionality is slow and monotonous. I chose manual mapping, because I have complex DTOs and I have more experience with this approach. Figure 9.5 demonstrates a simple case of mapping.

```
public static class ProductStockMapper
{
    1 reference
    public static ProductStock ToEntity(this NewProductDto source, int productId)
    {
        return new ProductStock()
        {
            ProductId = productId,
            Quantity = source.Quantity ?? 0,
            UnitId = source.UnitId ?? 0
        };
    }
}
```

Figure 9.5: Mapping to DTO example

The mapper methods are “extension methods” which are a C# specific feature. This makes it possible to call them on the to-be-mapped object of the same type it extends.

9.2. Security and user management

The application provides registration and login functionality. Sensitive user information such as email addresses and passwords must be handled carefully. Leaking user data can cause potential harm to the users and be detrimental for the business. To have a good level of security I used the ASP.NET Core Identity which supports user interface login functionality, manages users, passwords, roles, claims and tokens. With Identity users have two options for creating accounts: they can use an external login provider or utilize the login information held in Identity. Facebook, Google, Microsoft Account, and Twitter are examples of external login services that are supported by Identity. In my implementation external authentication is not possible. [29]

9.2.1. ASP.NET Core Identity configuration

I configured Identity with SQL Server. To use the functionalities provided by Identity the following dependency has to be added to the project:

- Microsoft.AspNetCore.Identity.EntityFrameworkCore

The *IdentityUser* class represents a user, but it can be extended by a custom implementation which I did in the application. The custom *User* class adds the following extra properties:

first name, last name, postal code, city, street, street number and supplier status. This information is required for the external distance matrix API call to determine the distance between suppliers and customers. The basic *DbContext* must be replaced with the *IdentityDbContext* types, this was presented in the "Code-first approach" section previously. The last configuration step is adding it to the service container in the Program.cs file.

```
builder.Services.AddIdentity<User, IdentityRole>()
    .AddEntityFrameworkStores<FreshFarmsDbContext>();
```

With the previously introduced *add-migration* and *update-database* commands, the necessary tables will be created, which can be seen in Figure 9.6.

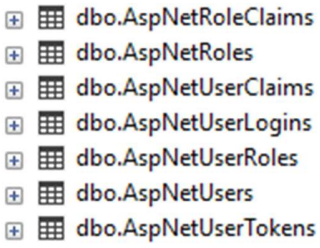

Figure 9.6: Identity tables

9.2.2. Adding a user

User registration can be done on the web application /registration page. After providing the mandatory user information presented in the user registration section previously, the client browser sends a HTTP POST request to the API's *RegisterUser* endpoint. After validating and mapping the data a new user can be inserted into the database with the help of the *UserManager<TUser>* class. This class ensures that all of the provided user data is saved to the appropriate database tables and the sensitive information is hashed (passwords).

9.2.3. Logging in

Logging into the application can be done on the /login page and requires email and password credentials. The client sends the login credentials wrapped into the *UserAuthenticationDto* class to the server. The server checks if the model is valid and then uses the *UserManager* service to find the user by the specified email address and then checks if the password is correct. The implementation details can be seen in Figure 9.7.

```
[HttpPost("Login")]
0 references
public async Task<ActionResult<AuthenticationResponseDto>> Login([FromBody] UserAuthenticationDto userForAuthentication)
{
    try
    {
        var user = await userManager.FindByEmailAsync(userForAuthentication.Email);

        if (user == null || !await userManager.CheckPasswordAsync(user, userForAuthentication.Password))
        {
            return Unauthorized(new AuthenticationResponseDto { ErrorMessage = "Invalid Authentication" });
        }

        var signingCredentials = GetSigningCredentials();
        var claims = await GetClaimsAsync(user);
        var tokenOptions = GenerateTokenOptions(signingCredentials, claims);
        var token = new JwtSecurityTokenHandler().WriteToken(tokenOptions);

        return Ok(new AuthenticationResponseDto { IsAuthSuccessful = true, Token = token });
    }
    catch (Exception)
    {
        return StatusCode(StatusCodes.Status500InternalServerError, "Error while logging in");
    }
}
```

Figure 9.7: Login implementation

9.2.4. JSON Web Token

A proposed Internet standard called JSON Web Token (JWT) allows for the creation of data with optional encryption and signatures, whose payload contains JSON and makes a variety of claims. Either a public/private key pair or a private secret is used to sign the tokens.

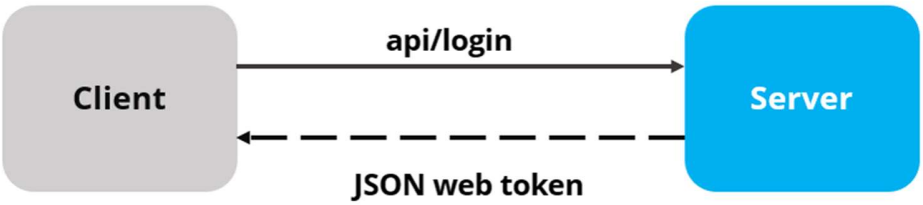

Figure 9.8: JSON Web Token [30]

The server will deliver an encoded JWT to the client after verifying the user's credentials and determining that they are valid, shown in Figure 9.8. In essence, a JSON web token is a JavaScript object that has the potential to include the logged-in user's properties. It might include the user's username, user roles, or some other related data. On the client-side, the JWT is stored in the browser's storage to remember the user's session.

JSON web tokens consist of three basic parts: the header, payload, and signature. The *Header*, which is a JSON object encoded in base64, is the first component of JWT. It is a

fundamental element of JWT and comprises details such as the kind of token and the name of the algorithm. The *Payload*, which is also a JavaScript object encoded in base64 format, follows the *Header*. Some information about the user who is logged in is contained in the payload. The user id, user subject, and details on whether a user is an admin user or not, for instance, can all be found there. Sensitive information should never be placed in the Payload since JSON web tokens are not encrypted and can be deciphered with any base64 decoder. The *Signature* is the last component. Typically, the server checks the signature portion to make sure the token contains the information it is providing as acceptable data. By merging the header with the payload, a digital signature is created. Additionally, it is based on a secret key that is only known by the server. [30]

9.2.5. JWT configuration

To configure the JWT authentication the following NuGet package was needed:

- Microsoft.AspNetCore.Authentication.JwtBearer

In the *Program.cs* file, JWT had to be configured. Figure 9.9 demonstrates the required configuration commands.

```
var jwtSettings = builder.Configuration.GetSection("JWTSettings");
builder.Services.AddAuthentication(opt =>
{
    opt.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
    opt.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
}).AddJwtBearer(options =>
{
    options.TokenValidationParameters = new TokenValidationParameters
    {
        ValidateIssuer = true,
        ValidateAudience = true,
        ValidateLifetime = true,
        ValidateIssuerSigningKey = true,
        ValidIssuer = jwtSettings["validIssuer"],
        ValidAudience = jwtSettings["validAudience"],
        IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(jwtSettings["securityKey"]))
    };
});
```

Figure 9.9: JWT configuration

First the JWT authentication middleware is registered by calling the *AddAuthentication* method. In the next lines, the default authentication scheme is defined by the *JwtBearerDefaults.AuthenticationScheme* and *DefaultChallengeScheme* settings. In the *AddJwtBearer* method JWT options can be configured. The issuer is the server that creates

the token, the audience is the client who receives the created token, the *Validatelifetime* defines that the expiry of the token must be checked. Some of the configuration values come from a file called *appsettings.json*. [30]

9.2.6. API endpoint security

By default, the endpoints of the application can be accessed by anyone. For some functionalities of the application, it is not a problem, for example viewing the available products can be done by anyone. On the other hand, endpoints which add a new product or list customer order history should not be accessed by anyone other than the actual user. In order to protect and endpoint from unauthorized access, an *[Authorize]* attribute, shown in Figure 9.10, must be placed either on the Controller containing the endpoint, or on the endpoint itself. [31]

```
[HttpPost]
[Authorize]
0 references
public async Task<ActionResult> AddCartItem([FromBody] CartItemToAddDto cartItemToAdd)
```

Figure 9.10: [Authorize] attribute

With this simple way of securing the endpoint, the controller checks if the user is authorized before it allows access to the endpoint. If the user is not authorized, then it returns a HTTP 401 (Unauthorized) message.

9.2.7. Client-side security

It is not enough to only secure the server-side of the application, but necessary steps must be taken to secure the client-side too, however it is limited. The client executes Blazor WebAssembly applications. Only the UI options to be displayed are selected using authorization. A Blazor WebAssembly application cannot enforce authorization access requirements because client-side checks can be altered or bypassed by a user. Three services have to be added to the Program.cs file, shown in Figure 9.11, which will provide the necessary functionalities to secure the client-side.

```
builder.Services.AddAuthorizationCore();
builder.Services.AddBlazoredLocalStorage();
builder.Services.AddScoped<AuthenticationStateProvider, FreshFarmsAuthenticationStateProvider>();
```

Figure 9.11: Client-side authorization configuration

The *AddAuthorizationCore()* provides the basic authorization services for the client-side. The *AddBlazoredLocalStorage()* function adds a library that gives Blazor programs access to the local storage APIs of the browser. The handling of serializing and deserializing values while saving or retrieving them is another advantage of utilizing this library. Finally, the *AuthenticationStateProvider* implementation has to be registered so it can be resolved and injected if needed.

For the client-side to be able to use the previously implemented server-side authentication and authorization a class implementing the *AuthenticationStateProvider* component must be created and JWT handling is required too. The *FreshFarmsAuthenticationStateProvider* class implements the *AuthenticationStateProvider* component which provides information about the authentication state of the user.

```
public override async Task<AuthenticationState> GetAuthenticationStateAsync()
{
    var token = await localStorage.GetItemAsync<string>("authToken");
    if (string.IsNullOrWhiteSpace(token))
    {
        return anonymous;
    }

    httpClient.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("bearer", token);
    return new AuthenticationState(new ClaimsPrincipal(new ClaimsIdentity(JwtParser.ParseClaimsFromJwt(token), "jwtAuthType")));
}
```

Figure 9.12: Handling authentication states

The *GetAuthenticationStateAsync* method, shown in Figure 9.12, returns an “anonymous” authentications state, when no authentication token is found, meaning the user is not logged in. If the user is logged in the application, then it returns an authentication state with the user’s information from the token. [32]

9.2.8. Role-based authorization

The application has three roles, *Supplier*, *Customer* and *Administrator*. The *Administrator* role is currently not used but it was created for possible future expanse of the program. *Supplier* and *Customer* roles are distinguished, based on these roles the user can access only

a limited number of pages and perform limited action as stated in the functional specification section previously.

```
public class RoleConfiguration : IEntityTypeConfiguration<IdentityRole>
{
    0 references
    public void Configure(EntityTypeBuilder<IdentityRole> builder)
    {
        builder.HasData(
        new IdentityRole
        {
            Name = "Customer",
            NormalizedName = "CUSTOMER"
        },
        new IdentityRole
        {
            Name = "Supplier",
            NormalizedName = "SUPPLIER"
        },
        new IdentityRole
        {
            Name = "Administrator",
            NormalizedName = "ADMINISTRATOR"
        });
    }
}
```

Figure 9.13: Available roles

To implement the distinguished functionality, the roles have to be added to the database, shown in Figure 9.13. This is done upon the first migration detailed previously in the "Code-first approach" section. A user has to be associated with one or more roles. That is done when the user registers an account and depends on if the supplier checkbox was checked on the registration form or not. After registration, when the user logs in, the associated roles are added to the user claims. Claims are name-value pairs, which represent what a user is and can do. These claims are encoded in the JWT with the help of the *GetClaimsAsync* method, shown in figure 9.14. [33]

```
private async Task<List<Claim>> GetClaimsAsync(User user)
{
    var claims = new List<Claim>
    {
        new Claim(ClaimTypes.NameIdentifier, user.Id),
        new Claim(ClaimTypes.Email, user.Email)
    };

    var roles = await userManager.GetRolesAsync(user);
    foreach (var role in roles)
    {
        claims.Add(new Claim(ClaimTypes.Role, role));
    }

    return claims;
}
```

Figure 9.14: Claim creation

The client-side extracts the claims from the JWT and enables or forbids access to components. Access to certain components of the web application can be restricted with the *<AuthorizeView></AuthorizeView>* and *<Authorized></Authorized>* elements. Components inside these elements can be hidden or shown based on if the user is authorized or not. Setting the *Roles=""* parameter in the *<AuthorizeView></AuthorizeView>* even allows to distinguish between roles. For example, in the navigational sidebar the menu items "Add Product" and "My Products" are hidden from the user with *Customer* role. [34]

9.3. Server-client communication

9.3.1. Server API

Blazor WebAssembly runs in the client's web browser and must communicate with the server to function properly. This communication is accomplished by the client calling the server-side endpoints. The endpoints are placed into Controllers which were created based on different functionalities the system provides like user or product management.

```
[Route("api/[controller]")]
[ApiController]
1 reference
public class ProductController : ControllerBase
```

Figure 9.15: Controller attributes and base class

Controllers must extend the *ControllerBase* class and have the *[ApiController]* attribute. The *[Route]* attribute specifies the route a controller and its methods can be accessed by. These requirements are presented in Figure 9.15. The controllers consist of multiple HTTP methods. The application uses HTTP GET methods which is used to retrieve certain data or resources from the server.

```
[HttpGet]
[Route(nameof(GetUnits))]
1 reference
public async Task<ActionResult<IEnumerable<UnitDto>>> GetUnits()
{
    try
    {
        var units = await this.unitRepository.GetUnits();

        if (units == null)
        {
            return NotFound();
        }
        else
        {
            var unitDtos = units.ToDtos();
            return Ok(unitDtos);
        }
    }
    catch (Exception)
    {
        return StatusCode(StatusCodes.Status500InternalServerError, "Error retrieving units from the database");
    }
}
```

Figure 9.16: HTTP GET implementation

The *GetUnits* method, shown in Figure 9.16, is designated with the *[HttpGet]* attribute which determines that this is a HTTP GET method. It does not have any parameters, but it is possible to provide for HTTP GET methods.

The HTTP POST method is another type which the application uses. POST is usually used to create or update a resource.

```
[HttpPost(nameof(AddProduct))]
[Authorize(Roles = "Supplier")]
1 reference
public async Task<ActionResult> AddProduct([FromBody] NewProductDto newProductToAdd)
```

Figure 9.17: HTTP POST implementation

The *[FromBody]* attribute, displayed in Figure 9.17, defines that the data has to be extracted from the body of the message.

9.3.2. Calling the server APIs from the client

In the client-side, the communication with the server is accomplished with the help of *HttpClient*. It is provided by the framework. To set up this service, only one configuration step is needed. In the Program.cs file, it has to be added to the services collection with the URI of the server, shown in Figure 9.18.

```
builder.Services.AddScoped(sp => new HttpClient { BaseAddress = new Uri("https://localhost:7122") });
```

Figure 9.18: HttpClient configuration

The client-side code contains services which are responsible for implementing the communication between the client and the server. The services contain a *HttpClient* instance, which is injected into the constructor of the class.

```
public async Task<ProductDto?> GetItem(int id)
{
    try
    {
        var response = await this.httpClient.GetAsync($"api/Product/{id}");

        if (response.IsSuccessStatusCode)
        {
            if (response.StatusCode == System.Net.HttpStatusCode.NoContent)
            {
                return default(ProductDto);
            }

            return await response.Content.ReadFromJsonAsync<ProductDto>();
        }
        else
        {
            var message = await response.Content.ReadAsStringAsync();
            throw new Exception(message);
        }
    }
    catch (Exception)
    {
        throw;
    }
}
```

Figure 9.19: Service using the HttpClient

The *HttpClient* is then used to initiate calls to the server, shown in Figure 9.19. Depending on the types of the calls, the route of and endpoint can be set and parameters to send. The response of the server is deserialized from the JSON into C# objects. This is the basic workflow that was implemented for the application.

9.4. Distance matrix API

The application provides a service which determines the distance between the customer and supplier. This information can be important for a customer. A closer location might be preferred because delivering the product can be cheaper, more environmentally friendly with less emissions and the customer can support a local farmer.

This service is implemented by using an external distance matrix API. The documentation and terms of usage can be found on the https://distancematrix.ai/ website. It comes with different plans, from free tier to prepaid and pay as you go. I chose the free tier version, that enables one origin and one destination. API calls are limited to 5000/month and one call per second.

```
private HttpClient httpClient = new HttpClient();
private const string APIkey = "Z9GXbB1qL3q57tdB259pRO460iLT8";

2 references
public async Task<string> GetDistanceAsync(Address origin, Address destination)
{
    try
    {
        string originParam = $"?origins={origin.Street} street {origin.StreetNumber}, {origin.City}, {origin.PostalCode}";
        string destinationParam = $"&destinations={destination.Street} street {destination.StreetNumber}, {destination.City}, {destination.PostalCode}";
        var request = new HttpRequestMessage
        {
            Method = HttpMethod.Get,
            RequestUri = new Uri($"https://api.distancematrix.ai/maps/api/distancematrix/json" + originParam + destinationParam + "&key=" + APIkey),
        };

        using (var response = await httpClient.SendAsync(request))
        {
            var responseBody = await response.Content.ReadFromJsonAsync<Root>();

            if (response != null && response.IsSuccessStatusCode)
            {
                return responseBody?.rows?.FirstOrDefault()?.elements?.FirstOrDefault()?.distance?.text ?? string.Empty;
            }
        }

        return string.Empty;
    }
```

Figure 9.20: Distance API implementation

The *GetDistanceAsync* method, shown in Figure 9.20, takes two parameters which are both of type *Address*. The address class contains the ZIP code, city, street and street number values of one user. The API key is provided by the external service, it has to be attached at the end of the parameters. The parameters of the API call have to follow a specific form:

?origins=<origin_location_1|origin_location_2|...|origin_location_n>
&destinations=<destination_location_1|destination_location_2|...|destination_location_n>
&key=<your_access_token>

In my implementation, only one origin and destination are supplied, because with the free tier plan that is the limitation. [35]

9.5. Services provided by the application

9.5.1. User registration

The registration page, presented in Figure 9.22, can be accessed by clicking on the “Register” menu item. On the page a registration form is displayed. All the fields are mandatory to fill in. If the user does not fill in a field a warning is displayed, shown in Figure 9.21, and cannot continue the registration.

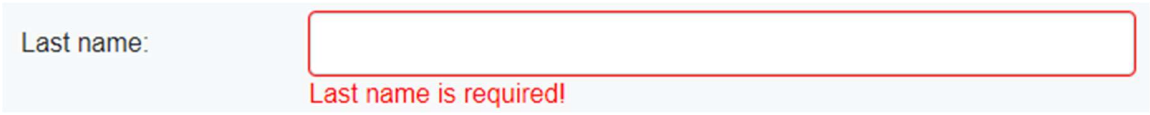

Figure 9.21: Required field

Checking the “Supplier” checkbox determines if the registered user will be a *Supplier* or *Customer*. The address related fields are mandatory because the application provides a distance calculation service.

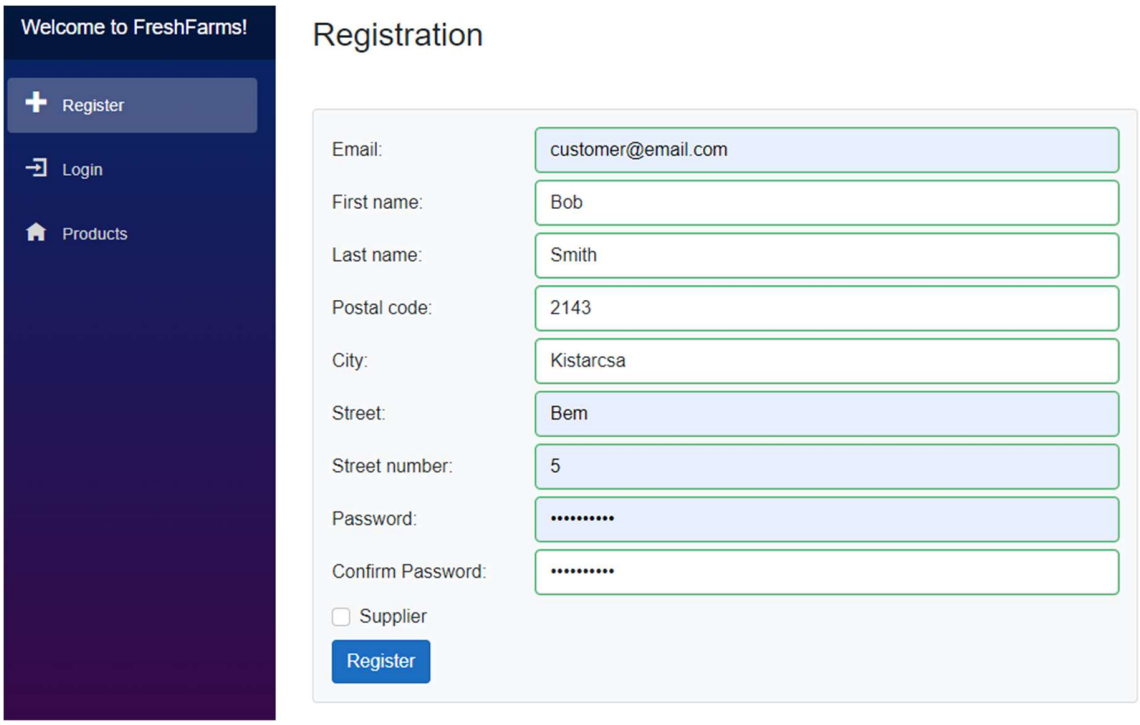

Figure 9.22: User registration form

9.5.2. Logging in

After creating an account, logging into the application becomes possible. Clicking the "Login" menu item in the navigational side directs the user to the /login page, shown in Figure 9.23. Specifying the email and password credentials is mandatory. The system checks if the specified credentials are correct and returns an *AuthentitacionState* based on that. If the login was successful, the user is redirected to the index page of the application which displays products.

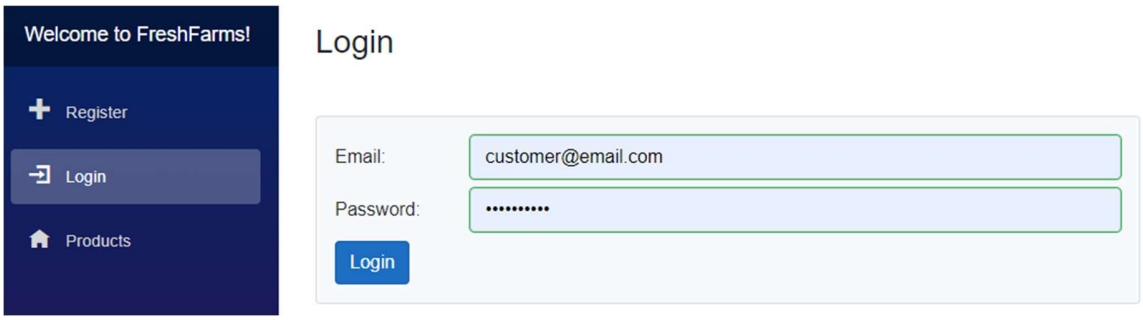

Figure 9.23: Login form

The session of a user is set to expire after 30 minutes. This is a simple server-side setting which can be easily changed based on business requirements.

9.5.3. Adding a product

If the user is a *Supplier*, then adding a product to sell is possible. Clicking the "Add Product" menu item navigates the user to the /addproduct page, shown in Figure 9.24. On that page, all of the fields are mandatory to fill. Images are stored in a folder on the server, with a unique name. The path of the image is then stored in the database.

Figure 9.24: New product from

If adding the new product was successful, the user is navigated to the Products page where it is immediately displayed.

9.5.4. Browsing products

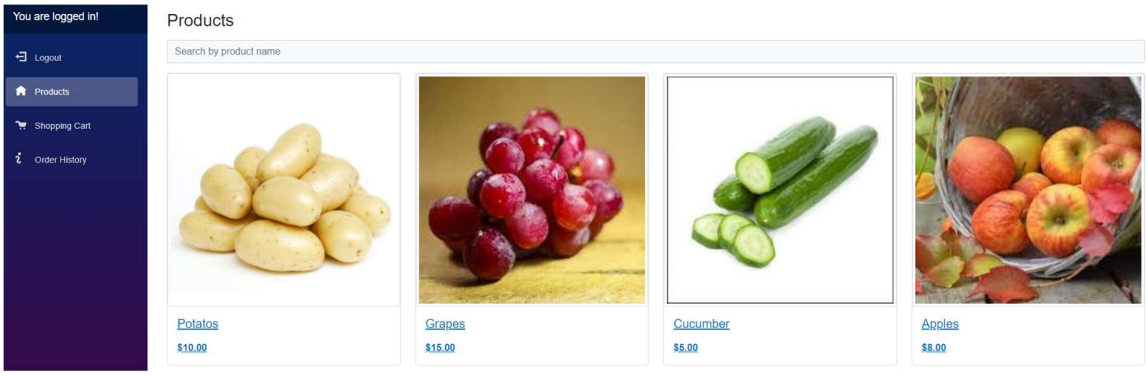

Figure 9.25: Product page

The users can browse the added products by clicking on the “Products” menu item, shown in Figure 9.25. This page is accessible with or without registration. Only a limited amount of information is displayed, which includes a picture of the product, name and price. For details, the individual product can be clicked on.

9.5.5. Product details

Clicking on a product the user will be navigated to the /productdetails page, presented in Figure 9.26. On this page, additional information is displayed to the user like description, stock, distance of the supplier. The distance is calculated between the address of the supplier

and customer. If the user is a *Customer*, it is possible to place an order after specifying the quantity and clicking the "Add product" button. This functionality is not available for suppliers.

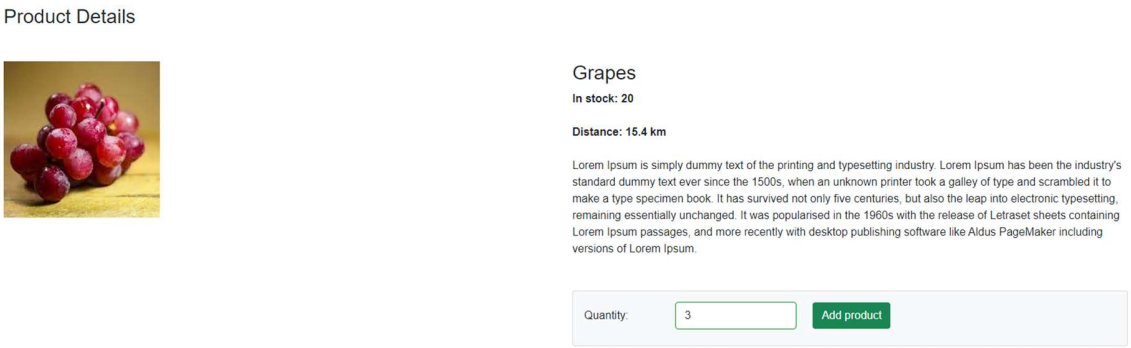

Figure 9.26: Product detail page

9.5.6. Shopping Cart

The shopping cart page can be reached by clicking the "Shopping Cart" menu item or clicking the "Add product" button on a product detail page which will navigate the user to the /shoppingcart page, shown in Figure 9.27. On this page, all the ordered products are displayed. The prices are shown per product calculated with the quantity selected. Items can be deleted by clicking the red "bin" button. On the top right part of the page, a summary is shown displaying the total number of items ordered and the total amount.

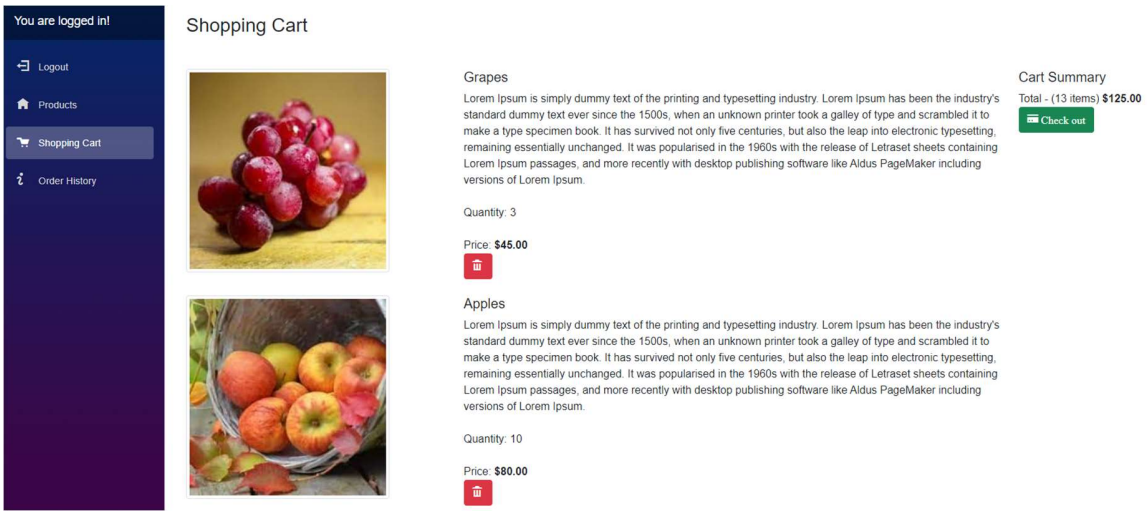

Figure 9.27 Shopping cart page

Clicking the “Check out” button will conclude the order. The number of available products is decremented, and the shopping cart will become empty.

9.5.7. Order history

The customer can view his or her order history on the order history page. It can be accessed by clicking on the “Order History” menu item. This page is only accessible with an authenticated customer account.

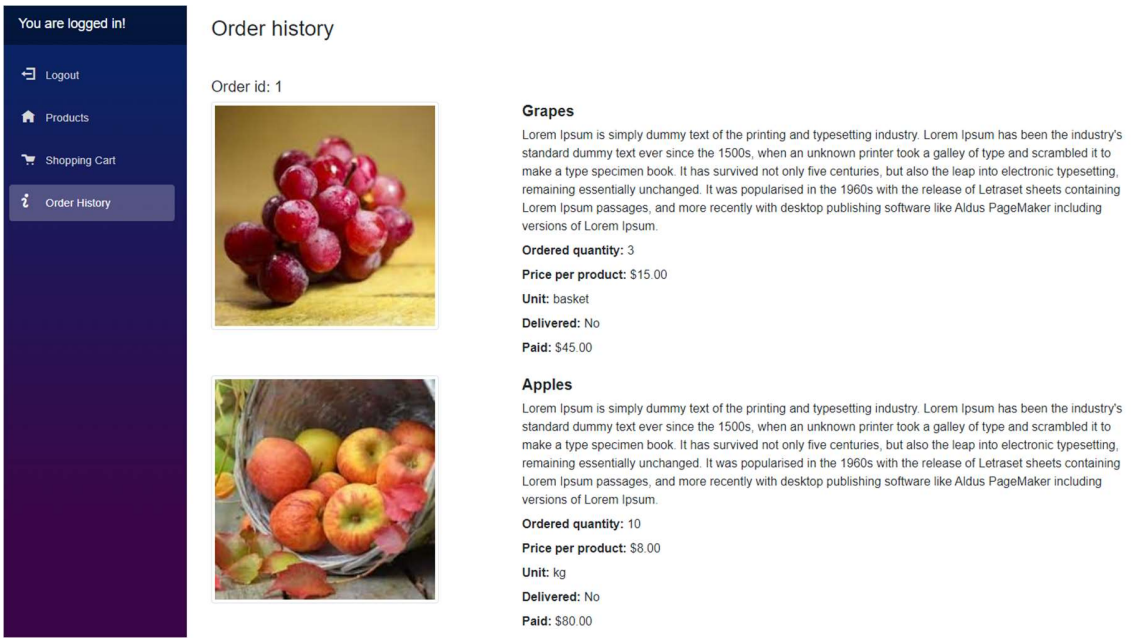

Figure 9.28 Order history page

Every item from a placed purchase is displayed here. Orders can be distinguished by the order id. The customer can find detailed information about the product like the quantity, unit, price and paid amount. Figure 9.28 presents the order history page.

9.5.8. Supplier statistics

The page supplier product info can be accessed by authorized suppliers by clicking on the “My Products” menu item, shown in Figure 9.29. On this page, the supplier can find all his or her sold products. Additional information is also displayed like the remaining quantity and unit. If a product is purchased and its supplier checks the product info page, the generated revenue is displayed.

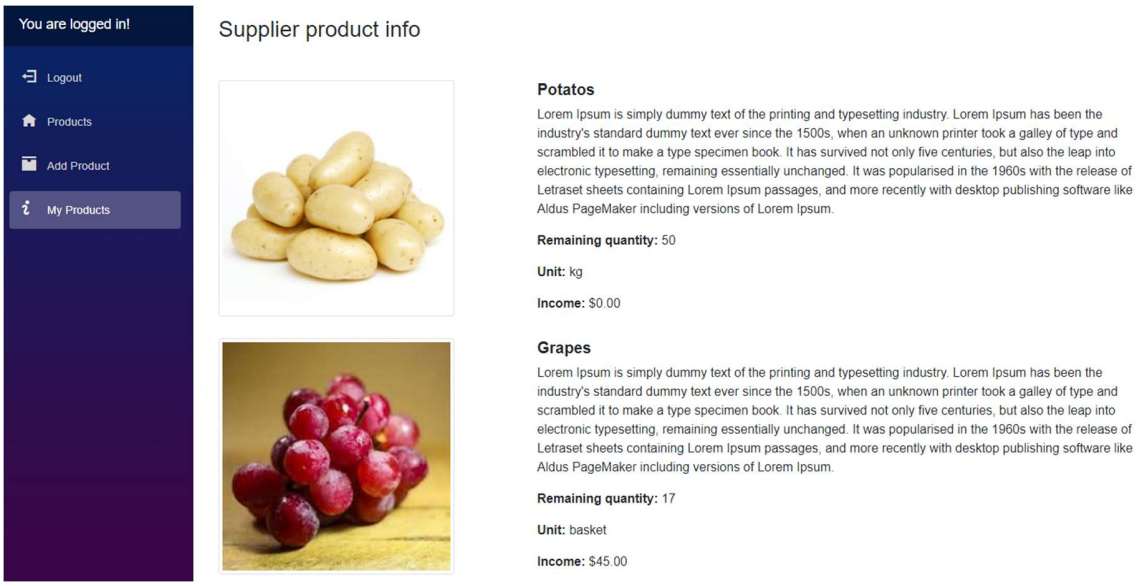

Figure 9.29 Supplier product info page

10.Future development possibilities

The application has a lot of future development possibilities. Registration and logging in can be extended with external authentication providers like OAuth. This could make registration easier for users with existing Google or Facebook accounts. With this solution sensitive user data do not have to be stored in the application's database.

Another development possibility is that the main product page could be improved with better search functionality. Now it only provides filter based on product name, but it could be extended with price ranges, quantity or distance filtering. The used distance matrix API can handle one origin and multiple destinations, therefore displaying or ordering by distance could be implemented.

To improve the user experience, product categories and a better user interface design could be created. With multiple product categories, it is easier to browse and find products. A category selection possibility should be also added to the product adding page. A stylish and user-friendly user interface is a must for modern applications. In the scope of this thesis work just a minimal looking user interface was created this could also be improved in the future.

11.Summary

My goal with the chosen thesis topic was to create a simple and easily extensible platform for selling locally produced food to local customers and to get a deeper knowledge about the technology such a platform can be created with.

In the first part of the thesis, I presented the currently available similar platform platforms and gave an overview of the technology that is used for building these applications. I also presented the different types of hosting possibilities with their pros and cons.

In the second part of the thesis, the implementation of the application is explained. The program provides the basic core functionalities an ecommerce food shop should have. It has a user management system which handles registration and login, products can be added to the website and purchased by customers, the supplier can track the sold products and the revenue they generate, and the customer can view the history of previous orders. These services cover the needs of an ecommerce food shop. During development I got to know the basics of web development with HTML and Blazor WebAssembly and managed to deepen my preexisting knowledge of server-side programming and working with databases. The other important part of the implementation was learning the basics of application security and then implementing it, which was the biggest challenge. I think this project was a very useful experience and I can build on this acquired knowledge in the future.

12.References

[1] M. Gunning, “Rebuilding our tech stack for the new Facebook.com,” Engineering at Meta, May 08, 2020. https://engineering.fb.com/2020/05/08/web/facebook-redesign/ (accessed Dec. 11, 2022).

[2] S. Simic, “LAMP Stack - What Is It, Advantages & Alternatives | phoenixNAP KB,” Knowledge Base by phoenixNAP, Jan. 06, 2022. https://phoenixnap.com/kb/what-is-a-lamp-stack (accessed Dec. 11, 2022).

[3] “Exploring the Software Behind Facebook, the World’s Largest Social Media Site - Pingdom,” pingdom.com. https://www.pingdom.com/blog/the-software-behind-facebook/ (accessed Dec. 11, 2022).

[4] “Market Wagon Feed,” Market Wagon. https://mw-next-poc.netlify.app/shop (accessed Dec. 11, 2022).

[5] Kifli.hu, “Kifli.hu | Online szupermarket - Minőségi online bevásárlás,” Kifli.hu. https://www.kifli.hu/koszontunk-a-kiflinel (accessed Dec. 11, 2022).

[6] “Bio zöldség házhozszállítás | Közvetlenül a termelőtől | Zöld Tanya Biokert | Magyarország,” Zöld Tanya Biokert. https://www.zoldtanya.hu (accessed Dec. 11, 2022).

[7] “Client-Server Architecture | EN.601.421: Object-Oriented Software Engineering (OOSE).” https://darvishdarab.github.io/cs421_f20/docs/readings/client_server (accessed Dec. 11, 2022).

[8] “What is an API?” https://www.redhat.com/en/topics/api/what-are-application-programming-interfaces (accessed Dec. 11, 2022).

[9] Bridget, “Quick guide to Calaméo’s API,” Calaméo Blog, Mar. 25, 2021. https://blog.calameo.com/2744/api-quick-guide/ (accessed Dec. 11, 2022).

[10] EdPrice-MSFT, “Web API design best practices - Azure Architecture Center.” https://learn.microsoft.com/en-us/azure/architecture/best-practices/api-design (accessed Dec. 11, 2022).

[11] “What is a REST API?” https://www.redhat.com/en/topics/api/what-is-a-rest-api (accessed Dec. 11, 2022).

[12] “What is Server-Side Rendering? Definition and FAQs | HEAVY.AI.” https://www.heavy.ai/technical-glossary/server-side-rendering (accessed Dec. 11, 2022).

[13] T. F. Iskandar, M. Lubis, T. F. Kusumasari, and A. R. Lubis, “Comparison between client-side and server-side rendering in the web development,” IOP Conf. Ser.: Mater. Sci. Eng., vol. 801, no. 1, p. 012136, 2020, doi: 10.1088/1757-899X/801/1/012136.

[14] gewarren, .“NET (and .NET Core) - introduction and overview.” https://learn.microsoft.com/en-us/dotnet/core/introduction (accessed Dec. 11, 2022).

[15] Rick-Anderson, “ASP.NET overview.” https://learn.microsoft.com/en-us/aspnet/overview (accessed Dec. 11, 2022).

[16] ajcvickers, “Overview of Entity Framework Core - EF Core.” https://learn.microsoft.com/en-us/ef/core/ (accessed Dec. 11, 2022).

[17] “Entity Framework Core Tutorials.” https://www.entityframeworktutorial.net/efcore/entity-framework-core.aspx (accessed Dec. 11, 2022).

[18] BillWagner, “Language-Integrated Query (LINQ) (C#).” https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/ (accessed Dec. 11, 2022).

[19] “What is LINQ? | How does it works | Need & Uses | Scope and Advantages,” EDUCBA, Oct. 24, 2019. https://www.educba.com/what-is-linq/ (accessed Dec. 11, 2022).

[20] “Angular - Introduction to Angular concepts.” https://angular.io/guide/architecture (accessed Dec. 11, 2022).

[21] “How Angular Works | How an Angular Application Works,” EDUCBA, Apr. 22, 2020. https://www.educba.com/how-angular-works/ (accessed Dec. 11, 2022).

[22] “What is Blazor | Blazor tutorial for beginners.” https://www.pragimtech.com/blog/blazor/what-is-blazor/ (accessed Dec. 11, 2022).

[23] J. Sampaio, “Blazor vs. Angular comparison guide,” LogRocket Blog, Feb. 24, 2022. https://blog.logrocket.com/blazor-vs-angular-comparison/ (accessed Dec. 11, 2022).

[24] “Angular vs Blazor? A decision aid for web developers in 2022,” DEV Community. https://dev.to/katholder/angular-vs-blazor-a-decision-aid-for-web-developers-in-2022-5e50 (accessed Dec. 11, 2022).

[25] A. Naseer, H. Zhiqui, and A. Ali, “Cloud Computing Security Threats and Attacks with Their Mitigation Techniques,” Oct. 2017, pp. 244–251. doi: 10.1109/CyberC.2017.37.

[26] D. Molnar and S. Schechter, “Self Hosting vs. Cloud Hosting: Accounting for the security impact of hosting in the cloud,” Jun. 2010.

[27] UNCTAD, Ed., The cloud economy and developing countries. New York: United Nations, 2013.

[28] “Self-Hosted vs Cloud-Based Project Management Tool · ActiveCollab Blog,” ActiveCollab. https://activecollab.com/blog/product/self-hosted-vs-cloud (accessed Dec. 12, 2022).

[29] jongalloway, “Introduction to ASP.NET Identity - ASP.NET 4.x.” https://learn.microsoft.com/en-us/aspnet/identity/overview/getting-started/introduction-to-aspnet-identity (accessed Dec. 12, 2022).

[30] M. Spasojevic, “ASP.NET Core Authentication with JWT and Angular - Part 1,” Code Maze, Jul. 09, 2018. https://code-maze.com/authentication-aspnetcore-jwt-1/ (accessed Dec. 12, 2022).

[31] Rick-Anderson, “Authentication and Authorization in ASP.NET Web API.” https://learn.microsoft.com/en-us/aspnet/web-api/overview/security/authentication-and-authorization-in-aspnet-web-api (accessed Dec. 12, 2022).

[32] M. Spasojevic, “Blazor WebAssembly Authentication with ASP.NET Core Identity,” Code Maze, Aug. 10, 2020. https://code-maze.com/blazor-webassembly-authentication-aspnetcore-identity/ (accessed Dec. 12, 2022).

[33] Rick-Anderson, “Claims-based authorization in ASP.NET Core.” https://learn.microsoft.com/en-us/aspnet/core/security/authorization/claims (accessed Dec. 12, 2022).

[34] guardrex, “ASP.NET Core Blazor authentication and authorization.” https://learn.microsoft.com/en-us/aspnet/core/blazor/security/ (accessed Dec. 12, 2022).

[35] “Distance Matrix API | Travel Time & Distance.” https://distancematrix.ai/distance-matrix-api (accessed Dec. 12, 2022).