

**BABEȘ-BOLYAI UNIVERSITY CLUJ-NAPOCA**  
**FACULTY OF MATHEMATICS AND COMPUTER**  
**SCIENCE**  
**SPECIALIZATION COMPUTER SCIENCE IN**  
**GERMAN**

**DIPLOMA THESIS**

**Traditional Multi-Page Applications and**  
**Modern Machine Learning Frameworks**

**Supervisor**

**Associate Professor Sanda-Maria Avram, PhD**

**Author**

**Victor-Cristian Catana**

**2022**

**BABEŞ-BOLYAI UNIVERSITÄT CLUJ-NAPOCA**  
**FAKULTÄT FÜR MATHEMATIK UND INFORMATIK**  
**INFORMATIK IN DEUTSCHER SPRACHE**

**BACHELORARBEIT**

**Traditionale Multi-Page Anwendungen und  
Moderne Machine Learning Frameworks**

**Betreuer**  
**Dozent Dr. Sanda-Maria Avram**

**Eingereicht von**  
**Victor-Cristian Catana**

**2022**

**UNIVERSITATEA BABEȘ-BOLYAI CLUJ-NAPOCA**  
**FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ**  
**SPECIALIZAREA INFORMATICĂ GERMANĂ**

**LUCRARE DE LICENȚĂ**

**Aplicații Tradiționale Multi-Page și  
Frameworkuri Moderne de Învățare  
Automată**

**Conducător științific**  
**Conf. Univ. Dr. Sanda-Maria Avram**

**Absolvent**  
**Victor-Cristian Catana**

**2022**



# Abstract

Multi-Page Applications are web applications which are fundamentally based on changing states in a web application, being a traditional approach to web development. This type of approach was chosen due to its proven stability and reliability, in order to pair it with the modern machine learning framework ML.NET which stars as the main focus of the paper, by using it to develop a model from nothing more than a dataset and the model builder interface. The experiment studied in this paper is layed out in detail throughout two main chapters. The first chapter, covering the theoretical aspects of the experiment, features comparisons between technological possibilities of existing solutions and between the technologies used in the application this paper bases its practical aspects on. Differences between Multi-Page Applications (MPAs) and Single-Page Applications (SPAs) were outlined, as well as dissimilarities between the technologies used to build an MPA, the foundation of the work. Moreover, the last section of this chapter focuses on ML.NET as a solution to Automated Machine Learning and its various training algorithms and particularities which set it apart from other ML frameworks. The practical chapter follows initial analysis, where the problem and its proposed solution are highlighted, architectural and structural analysis of the solution, as well as the implementation process and the end-result of the development. Moreover, the experiment's steps and more complex procedures are outlined in this section as well, providing an insight into the behavior of the application from simple navigation to model consumption and prediction. Finally, the chapter presents the evaluation scenarios for the application as well as the trained model, painting the image of the test environment as well as it's results. The achievements and open points of discussion are described in the final chapter, providing a deeper and more detailed perspective into not only the current state of the experiment and application but also future possible enhancements and developments.

# Contents

1. Introduction.....	1
2. Theoretical concepts .....	3
2.1 Basic concepts and state of the art.....	3
2.2 Technologies.....	5
2.2.1 ASP.NET Boilerplate .....	6
2.2.2 ASP.NET Core.....	8
2.2.3 ASP.NET Core MVC.....	9
2.2.4 Entity Framework Core.....	10
2.2.5 ML.NET .....	11
3. Core of the work .....	14
3.1 Problem and solution .....	14
3.2 Methodology and procedure .....	18
3.3 Overview of architecture .....	19
3.4 Implementation.....	22
3.5 Evaluation.....	26
3.5.1 Test environment.....	27
3.5.2 Results .....	28
4. Conclusion and future work.....	29
4.1 Conclusion .....	29
4.2 Future work.....	30
Bibliography .....	31

# 1. Introduction

As the internet and the people evolve, it just so happens that software sometimes trails behind the waves of evolution and remains, sadly, trapped in a time bubble due to lack of scalability, being poorly structured or simply having lost the interest of the public. Since changes in architecture later on in the software's lifecycle are costly and tedious, developing a solution while having a structurally sound architecture in mind and carefully designed relationships between components proves to be a first step in improving scalability and ease of use. Clean code is sometimes even more important, since the bigger the projects get, the harder it is going to be for a developer to understand the ins and outs of the software they're working on. However, none of this is as important, to the end-user, as novelty. Adding an element of novelty such as AI to a software gives it a fresh new look and, due to it gaining popularity thanks to improvements in computing power, advanced algorithms and increasing data volumes [1], it can also boost interest in the software and its popularity.

This paper aims to present the development of an application with the use of a combination of traditional and modern frameworks and Machine Learning algorithms which binds together several currently existing solutions and the creation of a unified platform intended for the Karting community, as an example. In order to do that, the architecture and structure follows Domain Driven Design approaches by using ASP.NET Boilerplate, which is a Web Application Framework based on Microsoft's ASP.NET Core web framework. The Machine Learning component was achieved by integrating Microsoft's ML.NET framework into the solution and consuming a trained model which makes recommendations by predicting possible product preferences based on previous purchases or product popularity. The existing solutions in this area are the Romanian Karting Federation's official website, which lacks any type of interactivity and behaves more like a forum, and OLX and other electronic markets which are designed for general use and allow anyone to sell products, not requiring certification, which can prove problematic since, like any motorsport activity, karting can be dangerous unless certain precautions are taken.

The paper is structured in two main chapters, the first provides an insight into the theoretical aspects the paper is based on while the second one not only showcases the development process of the application, but also highlights the parts which stood out from the rest and proved more complex.

The theoretical chapter begins by comparing Multi-Page Applications to Single-Page Applications and delves into the application's foundation, comparing structural aspects as well as practical ones, taking a deep dive into the technology stack used while developing the application. More specifically, the advantages of the chosen technologies and approaches are brought forward while also exploring other possibilities and motivating the choice.

The second chapter of the paper focuses on the practical development of the application and reveals architectural details, such as the deployment process and project and database structure, as well as implementation particularities like prediction model consumption, front-to-back communication and customized layout rendering.

Finally, the paper concludes with ending statements and the final state of the experiment, as well as possible future improvements or research approaches which could contribute to enhancing the depth of detail this work provides.



## **2. Theoretical concepts**

In the current day and age, many ordinary aspects of one's life have already been or are currently being ported into the online realm, with a person's dependency on internet access becoming more of a basic human need, rather than a whim or a simple want. Such is the case of an activity which can either be a leisure activity or perhaps a passion that can turn into a career such as karting.

Much like any other business, the Romanian Karting Federation FRK has its own website where one can find information about official events, rules and regulations and so on. However, what the currently existent solution lacks is exactly the core aspect of the "new way" of doing things, interaction and straightforwardness. There is not much more to do other than browse through the available pages and read the available information, there is no way to directly contact providers, no way to contact karting academies or access an official market that can offer exactly what one needs, it simply lacks humanity.

These are exactly the issues that the solution proposed in this paper aims to solve. More specifically, turning the website that is currently available into a web application which allows its users to communicate with sellers and officials, also being able to receive personalized product recommendations based on previous purchase history or overall product selling performance.

### **2.1 Basic concepts and state of the art**

Multi-Page Web Applications are traditional applications that depend on the pages refreshing to update the data that is displayed to the user and to communicate with the server. These types of applications are suited for large web platforms with multiple, complex and different views. It is a classic architectural approach that has proven to be dependable, easily scalable and simpler in terms of technology stacks.

As technology advances and more modern approaches to developing web applications become more and more popular, the diversity of architectural structures also grows proportionally. Nowadays, Single-Page Applications are hastily gaining traction as more popular choices for different types of applications and customer needs. However, SPAs have only been around for a little more than a decade now and, although their issues and cons have started to be solved as time goes on, this is a work in progress.

In response to the aforementioned SPAs comes the traditional and stable MPA approach. This application type is highly scalable, featuring multiple page types and more a more complex structure, can easily allow the use of the browser-based *Backwards* button and browsing history since navigating from page to page calls for changes in the URL path, thus allowing the user to take advantage of all the incorporated features of the browser. Moreover, this type of behavior allows for better and easier SEO optimization, since crawlers can more efficiently check page contents compared to SPA pages which require JavaScript to load the pages and therefore making such integrations more difficult.

In terms of security, it requires more time and effort to secure an MPA completely and efficiently with multiple pages since every single page needs to be secured, but safety is nevertheless much greater due to the heavy dependence on JavaScript that SPAs have, making them more prone to attacks and proving an easy target for cross-site scripting. SPAs also require APIs to be publicly exposed, thus creating the need for the endpoints to also be secured properly. Moreover, compared to SPAs, MPA pages are “*short lived*” and don’t create the problematic issue of memory leaks that the long lifecycles of SPA-based platforms encounter. Therefore, as far as data safety and integrity is concerned, MPAs feature increased security, however at the cost of more time and effort.

	SPA	MPA
Easy SEO		X
High security		X
Offline functionality	X	
Scalability		X
Works without JS		X
Browser Back & History		X
Smooth UX	X	
Speed	X	

Figure 1 – Single-Page Applications compared to traditional Multi-Page Applications

Performance-wise, MPAs take more time and resources to retrieve data from the server and present to the user since each action represents a request and retrieves the page from the server, therefore also feeling more rigid and not providing the user with the smooth experience an SPA is able to. This is a by-product of different page lifecycles, as each approach features different particularities that suit their individual needs. Both of these start their lifecycle with an initial page request from the client that reaches the server, which in turn returns HTML, JavaScript and CSS content to the client as a response. However, this is the point where the two begin to differ in approach. MPAs submit a POST or GET request following a user action which results in the

client having to reload the page to update the HTML content of the page. This, however, is where the modern aspect of SPAs comes into light more visibly. The modern

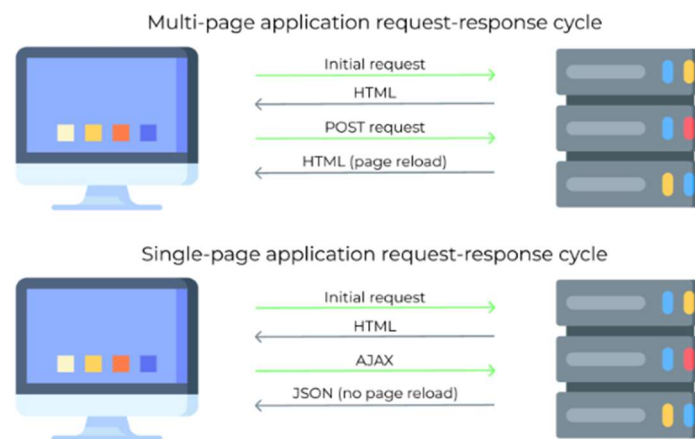


Figure 2 - Page lifecycle differences between traditional MPAs and modern SPAs

approach is to use AJAX to make requests to the server and, instead of having to get whole HTML pages from the server, the client receives JSON or other data formats which it uses to update the user's view, eliminating the need for the page to reload and only updating certain fragments of the page. This technique proves more efficient, faster and provides the user a smoother transition between changing states. One of the drawbacks of SPAs is that the initial, first load is very slow. This is caused by client-side rendering, because as efficient as SPAs are, their performance is ensured by bulky JavaScript frameworks.

## 2.2 Technologies

In order to ensure ease of scalability, use and maintenance, the application framework used for the application this paper features is ASP.NET Boilerplate [2], which is an open-source application framework encompassing multiple tools and making use of popular good-practices. The technology stack used in the application features ASP.NET Core for its foundation, uses Razor Pages for the presentation layer and Entity Framework as an Object-Relational Mapper for the Infrastructure Layer managing an SQL-Server-based database.

## 2.2.1 ASP.NET Boilerplate

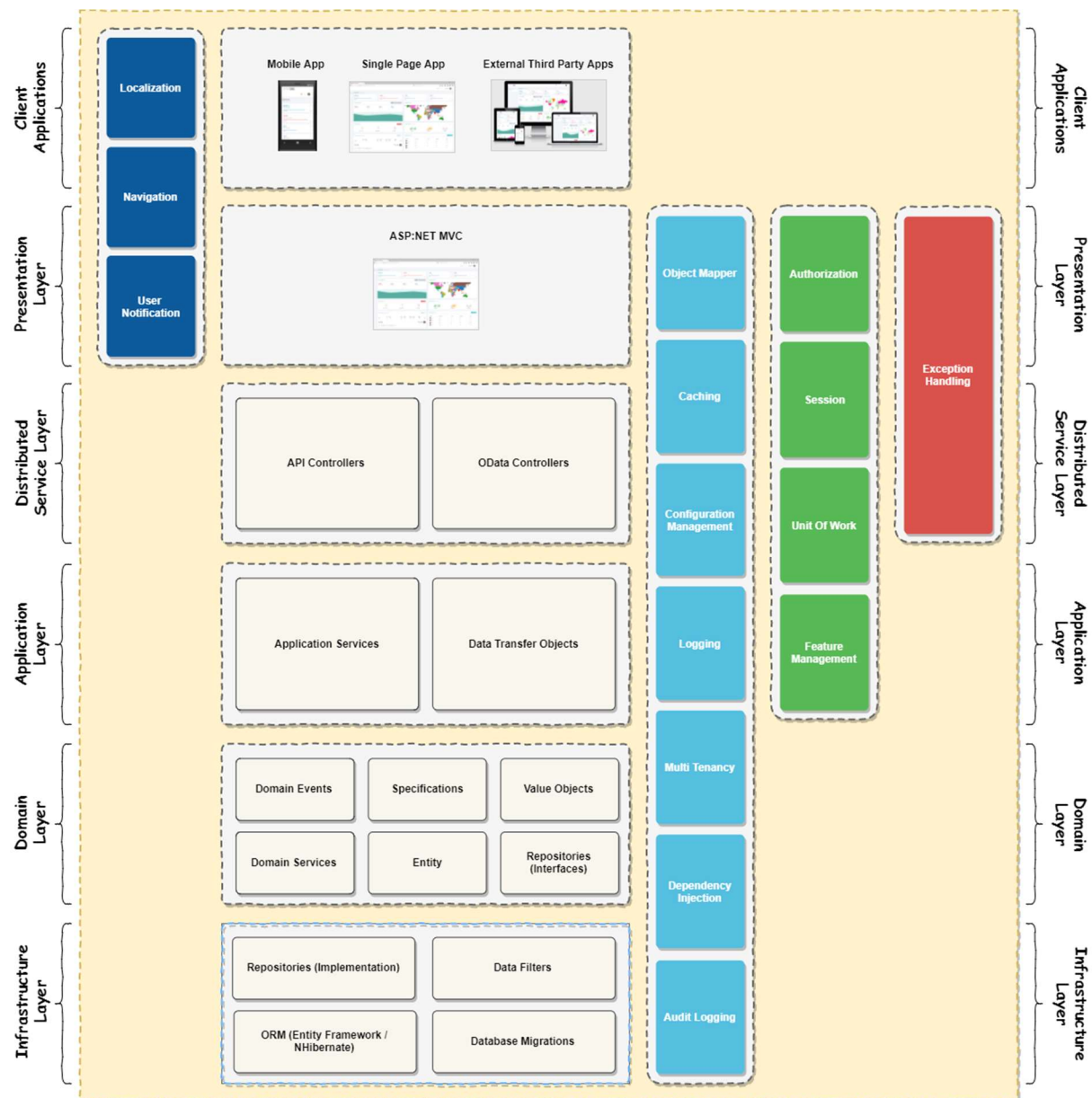
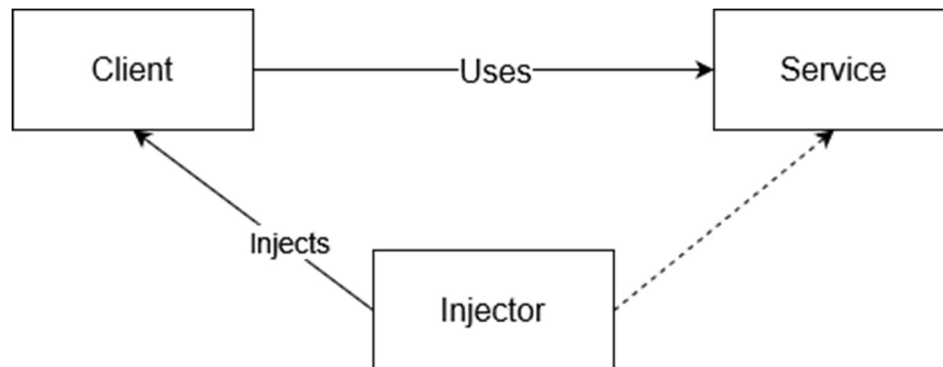


Figure 3 - ASP.NET Application Architecture [14]

One of the key aspects of ABP is that it provides several useful and important features for the developer to develop an application efficiently and more methodically. Some of these integrated features are Dependency Injection, Default Repositories, Permissions, Unit of Work, Localization, Auto Mapping and others.

Dependency Injection is a technique which allows for one or more dependencies to be injected, removing the creation from the dependent object and therefore removing a hard dependency through the use of Constructor Injection for example. Managing these dependencies would be tedious if not for the way that ABP uses dependency injection frameworks, registering dependencies conventionally using transient lifestyles. Of course,

there are also simple ways of registering dependencies directly should conventional registrations not be sufficient.



Default Repositories in ABP are used to perform CRUD operations for Entities, using a separate repository for each Entity. Through integration with Entity Framework, entering a repository method automatically opens a transaction which is either committed, if the method ends and returns, or rolled back, if the method throws any type of Exception. This type of behavior is accommodated by using a Unit of Work. All Repository, Application Service and API Controller actions are by default unit of work and are, therefore, atomic. Since they are heavily used in Dependency Injection, every repository instance has a transient lifestyle.

Since a multi-tenant application frequently needs different authorization levels, this can easily be accomplished thanks to the Permissions features integrated in the framework. Permissions can be enforced not only for API endpoints, but also for views, thanks to the versatility of Razor Views, and even in client-side JavaScript through the ABP namespace.

A Unit of Work, as described briefly above, when talking about Default Repositories, is a mechanism that allows for transactional behavior to be enforced in an application which uses a database. Methods which are managed through the unit of work system are called atomic and opening, managing and closing the connection to the database is handled automatically by the implemented system. Should one atomic method call another atomic method, they both use the same transaction, managed by the first entered method. Naturally, even though some methods are by default classified as unit of work, there is also the option of explicitly controlling the unit of work. Moreover, a non-transactional unit of work can also be explicitly used, if locking rows or tables in the database proves problematic for a developer's implementation. Saving changes can also be done automatically or explicitly, depending on the use and specific needs of the application.

Since, more often than not, applications tend to be suited not only to one culture but to a larger number of cultures, ABP features integrated Localization. Localization Sources can be accessed simply by calling the L method, after which the system, based on the user's current

culture, establishes the correct localized text to suit the user. This is done either through ASP.NET Core default providers or custom, ABP defined providers. This method allows for usage in server-side code, controllers, views, and also JavaScript code.

In order to efficiently isolate the Presentation Layer from the Domain Layer efficiently, Data Transfer Objects are used to call an Application Service that in turn uses these domain objects for specific operations and returns it back. Although DTOs might seem tedious and exhausting, they are an excellent tool to completely avoid the presentation layer from working directly with domain objects such as Entities or Repositories through abstraction. However, there needs to be a process that can map DTOs to entities and vice-versa. ABP solves this with the integration of AutoMapper, this way, mappings only have to be created once. The specific differences between the objects must be specifically stated, after which all one has to do to transform objects back and forth is to call pre-defined methods.

By encompassing multiple popular and useful tools, ABP manages to create a perfect environment for an application to be developed in, enabling the developer to scale it indefinitely and easily maintain it, even though it is suited for both small and large applications alike. Not only that, but ABP also provides a multi-layered architecture and a strong infrastructure, enabling developers to work on projects that feature different technologies, such as Angular or React, or use traditional HTML, CSS and JavaScript through Razor Pages.

### 2.2.2 ASP.NET Core

ASP.NET Core is a web framework designed to run on .NET Core, developed by Microsoft and initially released in 2016, 7 years after its predecessor ASP.NET was first released. Since the release of the newer ASP.NET Core, the latter has been retired, meaning that ASP.NET Core is now under Long Time Support. The main difference between the two is their approach on cross-platform development. ASP.NET Core can be used for applications targeting not only Windows, but also Linux and MacOS [3]. Meanwhile, ASP.NET is only available for applications running on Windows machines. Another difference among them is that ASP.NET Core is meant to only use ASP.NET Core MVC solutions for the Presentation Layer, while its older brother features Web Forms, MVC and Web API. Not only that, but ASP.NET Core is optimized for use of both .NET Core, which feature modular and optimized .NET libraries and runtimes for both native Windows as well as for other OS, and .NET Framework, unlike the older version.

Concerning the Presentation layer, for which ASP.NET Core brings many changes with ASP.NET Core, performance has been proven to be a big plus for the more recent framework. Featuring an MVC design pattern [4], the View component is by far the one with the shinier

upgrades, with the Razor view engine being the one to embed .NET Code in HTML markup. Features of ASP.NET Core MVC include Routing, Model Binding, Model Validation, Razor View Engine, Tag Helpers and others [3].

Based on ASP.NET Core's routing, ASP.NET Core MVC features URLs with a searchable and simple structure. Not only can routing be done conventionally, where the routing engine parses incoming request's URLs matching them to specific templates defined by the developer in a seamless manner, but specific routing information can be added to Controllers through Attribute Routing.

Another important activity performed by the framework is working with Models. Controllers can now have request data logic removed through Model Binding, which allows for the data to be converted into objects, rather than HTTP headers and string parameters. Furthermore, Model Validation is also supported through unobtrusive annotations used to decorate the models. Not only are the attributes checked on the server, before a controller calls an action, but also on the client side prior to displaying information to the user through jQuery Validation.

Razor is a compact, expressive and fluid template markup language for defining views using embedded C# code [4]. The Razor view engine is capable of generating web content on the server, while making use of server code within its client-side content, using partial views, which increase modularity and ensure ease of use in a larger application, and working with strongly typed views, adding type checking and IntelliSense support to its multiple features.

Repetitive code has been a menace for developers for a very long time, now being seen in an even more negative light than ever with the increase in popularity of clean code followers. It is not a problem that only affects server-side code, but client-side code such as HTML as well, creating frustrating scenarios that can be avoided with the use of Tag Helpers. They create and render HTML content with the help of server-side code by defining custom tags which offer the advantage of having server-side rendering used through HTML coding style. Making great use of the open-source aspect of ASP.NET Core, Tag Helpers can be created from scratch by developers and are also available through NuGet packages and GitHub repos.

### 2.2.3 ASP.NET Core MVC

The main frameworks available with ASP.NET Core for creating web applications are Razor Pages and MVC, for server-side rendering, and Blazor for client-side rendering. Razor Pages and MVC are very similar since they both feature the same approach with server-side rendering, having a small number of differences among them. Basically, MVC is the legacy

version of the newer Razor Pages. Big differences are featured, however, in .NET's newest version of web application framework, Blazor.

The structural difference between MVC and Razor Pages is that the former features separated models, views and controller, staying true to the MVC design pattern, while the latter focuses on a more unified structure with all its infrastructure organized in Pages that feature code-behind closely tied to the HTML content.

Moreover, according to [5], many of the ASP.NET Core MVC API methods and coding patterns follow a cleaner, more expressive composition than was possible with earlier platforms. Furthermore, being part of Microsoft's new generation of web development platforms, ASP.NET Core MVC can be classified as cross-platform not only for its deployment, but also regarding development.

## 2.2.4 Entity Framework Core

In order to work with a database, a developer needs to have an object-relational mapper known as an ORM, Entity Framework Core is a version of Entity Framework that is open-source and helps .NET developers use .NET objects with their database of choice. There're multiple ways of using EF Core depending on an application's specific situation, with data access being performed through models [6]. A model can be either generated from an existing database, configured manually to match an existing database or create the model first and use Migrations to generate the database afterwards. The last is the most straightforward solution out of the three, enabling developers to save time by changing the database automatically as the model is changed, therefore, being the most efficient and least time-consuming.

EF Core allows the developer to choose which database type it should work with, allowing SQL Server or Azure SQL, SQLite, MySQL, PostgreSQL and more to be used. Being a newer and improved version of Entity Framework, it offers most features implemented in the former version, but also many more features that will only be specific to it, since it is the only one currently receiving support.

The most important aspect of Entity Framework Core, however, is that it provides ways for the software developer to create relationships programmatically and automatically between tables and entities without ever having to open a Database Management System. Moreover, cascading actions are also configurable and can be enabled by the developer to ensure proper and facilitated data workflow from source code actions to database data manipulation, allowing for a straightforward approach in programming logic.

Unlike EF6, which was designed around relational databases only, EF Core handles both relational and nonrelational databases. Even though, at the moment, the only NoSQL



database provider is Cosmos DB, which was added in EF Core 3.0, more NoSQL database providers will most probably be written now that EF Core has been altered to handle NoSQL databases. [7]

### 2.2.5 ML.NET

Since in this day and age automation and artificial intelligence are rapidly gaining popularity, .NET features its own Machine Learning framework since 2018, completely open-source and cross-platform. It provides a wide range of benefits, such as TensorFlow integration, image processing capabilities, forecasting and anomaly detection, text processing tools and traditional machine learning algorithms that are common among ML frameworks.

Compared to other popular machine learning frameworks, ML.NET proved to be faster and more accurate, being able to train and test a sentiment analysis model with 93% accuracy in only 11 minutes, which proved 6 times quicker than the more popular scikit-learn framework [8]. Researching is not limited to C#, it can also be done in Python, allowing .NET developers to completely train a model in Python and simply load the trained model within minutes.

Since ML.NET is relatively a newborn framework, it can make use of all the latest jumps in technology, including AutoML, short for *Automated machine learning* which automates the process of applying machine learning to data [9]. This technology can use training data with different machine learning algorithms and uses a variety of settings for each and every one of them to determine which algorithm is best suited to the problem by evaluating accuracy across all explored models [10]. Different problem and dataset types can be used with specific training scenarios, ranging from Data classification and Forecasting to Object detection and Image classification. Furthermore, the training environment can also be different for each scenario, as ML.NET provides the option to use an Azure and GPU based training for certain scenarios. In order to properly train a model to be trustworthy and dependable, a dataset has to be provided, which can be either a structured text file or an SQL Server database table. In order to evaluate multiple models and find the best option for the problem at hand, giving the model builder more time to thoroughly explore various models works best and results in a better final model. Ultimately, consuming the trained model is made easy through project templates, which allow a developer to use either a Console App or an ASP.NET Core web API to make predictions. Moreover, deploying the model can easily be done within minutes by deploying it to Azure.

The framework has multiple model training algorithms it makes available to the developers, depending on the dataset and the computing resources available and the task they are trying to solve. Nevertheless, the same algorithms can be used, with different

configurations, to produce different models depending on their task. The algorithms are split into several main categories, linear algorithms, decision tree algorithms, meta algorithms, matrix factorization algorithms and support vector machines. Linear algorithms use features which are linearly separable, which means that they can be split linearly into separate categories, while decision tree algorithms do not have this requirement since they are based on a series of decisions, rather than a linear combination of input data and weights [11].

Two linear algorithms which can be used for multiple tasks are L-BFGS and Stochastic Dual Coordinated Ascent, both of which can be used for regression, binary classification and multiclass classification alike. The other two algorithms, Symbolic Stochastic Gradient and Averaged Perceptron, only work for binary classification tasks. While, according to Microsoft [11], linear algorithms are fast, scalable and cheap to train and predict, decision tree algorithms shine as being very accurate in comparison. All of the available decision tree algorithms can be used for binary classification and regression, while two of them can be used for ranking and only one for multiclass classification. Matrix factorization algorithms can either be used for collaborative filtering in recommendation or in binary classification tasks. Meta algorithms can be used either for multiclass classification, clustering or anomaly detection while support vector machines can only be used for binary classification tasks.

Tasks are types of predictions which are made based on a specific problem and with specific data. These tasks can be paired with algorithms in order to produce trainers which thereafter result in a usable model. Although there are various task types being offered as part of ML.NET, this paper focuses on recommendation, regression and multiclass classification as they are the most suited for making product predictions based on past sales behavior. The first type of task that we can use for our problem is the recommendation task. The recommendation training algorithm is based on Matrix Factorization, however it can only be used for datasets that have product ratings which is not available in our situation. The next task type possibly suited for our dataset and problem is regression, which predicts a label value based on a set of related features. This task type would normally be the most suited to the problem we face, yet works best with larger datasets and feature amounts. Regression models can be trained with algorithms such as Fast Tree, Fast Forest or Lbfgs Poisson. The last type of task which is most likely to work well regarding the available data and proposed problem is multiclass classification, which is used to predict the category of an instance of data. This type of task uses a set of labeled examples as input data. It then converts them to a numeric type and produces a classifier which can then make predictions based on its training. Among the algorithms that can be used to train a multiclass classification model are Lbfgs Maximum

Entropy and One Versus All, which is able to upgrade any binary classification learner to act on multiclass datasets.

Should the trained models fail to meet satisfactory accuracy levels, improving the model can be done by providing more data samples and improving the dataset with more details, which allows the trainer to more accurately interpret data points thanks to the added context. Moreover, cross-validation or even choosing a different algorithm are also methods which can lead to better results, should the first option not successfully achieve its goals. Training machine learning models is an exploratory process which is unique due to how variable datasets, problems and even the parameter tuning.

### 3. Core of the work

In aiming to create a stable and dependable solution, combining modern and mature technologies was necessary. In order for the mix between the two to be successful and not lead to unexpected issues, the structural and architectural layout of the application had to be very well defined, be it the separation of projects within the solution through an N-Layered architecture or the structure of the database.

In order to preserve principles and best practices necessary for a SOLID programming environment, the application was structured as for allowing ease of use and effortless scalability through popular and common separation and abstraction techniques. Moreover, tests were implemented in order to ensure a normal and predictable behavior of the features within the application. Endpoints and base methods for more used features were thoroughly tested.

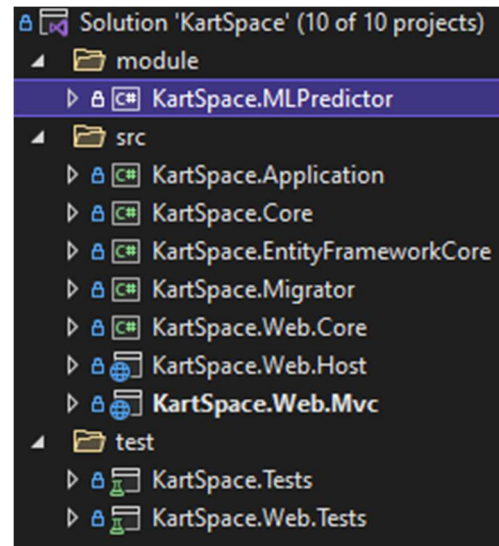


Figure 4 - Projects' layout within the solution

#### 3.1 Problem and solution

The aforementioned problem of the current situation was that information could not be accessed by users or merchants and the process of gaining access to that information was tedious and frustrating. Through the developed solution, any user can access all the information they might need directly from one source, albeit only by creating an account. Not only that, but merchants can also deliver their products to clients by simply having a Tenant Account created. Once that is done, company users can manage the items for sale, client orders and check statistics.

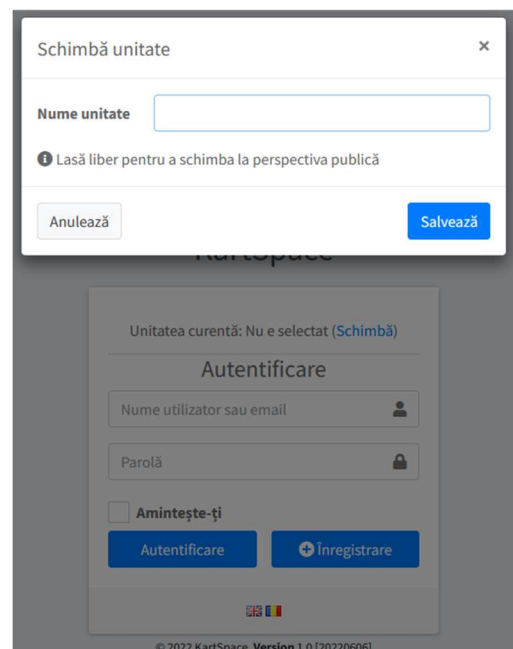


Figure 5 - Tenant change menu

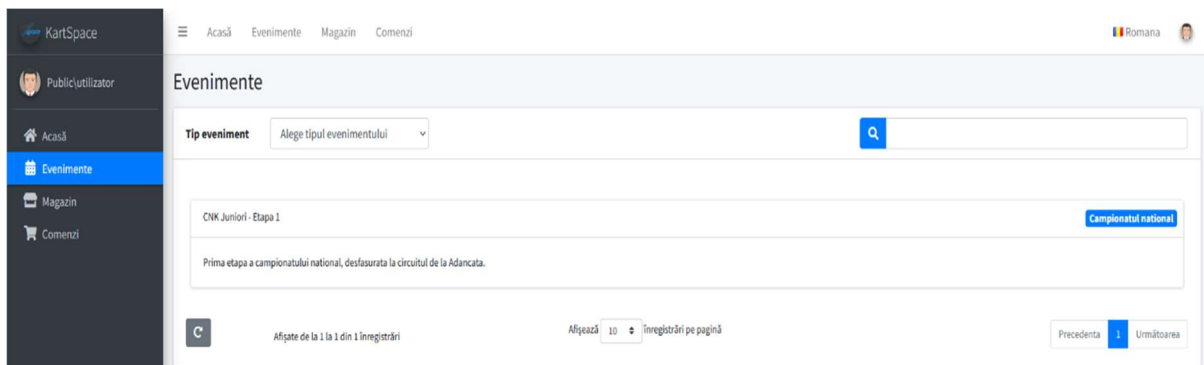


Figure 6 - Events section for a regular user

After logging in, one of the features available for every user is the Events section. In this section, administrator can modify or delete existing events or create new ones. On the other hand, users can see past, ongoing or future events in a table featuring pagination and filtering while also being able to select specific event types they would like to see from a drop-down

Figure 7 - Edit role modal

menu. This behavior is implemented through jQuery DataTables and permissions for the administration of this sections is managed through permission checkers in the page's code. Such permissions can be managed by administrators in the Roles section of the application, allowing for different roles to have specific permissions assigned to them, without the need for roles to be made for each task. Each user can have a role, and each role can have different permissions, enabling administrators to easily manage activities through configurable settings. On tenant creation, an admin account is automatically created and given all permissions. However, there can also be host-specific permissions awarded only by the host administrator,

such as tenant management. In code, this is achieved through an AuthorizationProvider class

```

1 using Abp.Authorization;
2 using Abp.Localization;
3 using Abp.MultiTenancy;
4
5 namespace KartSpace.Authorization
6 {
7     3 references | Victor Catana, 1 hour ago | 2 authors, 4 changes
8     public class KartSpaceAuthorizationProvider : AuthorizationProvider
9     {
10         0 references | Victor Catana, 1 hour ago | 2 authors, 4 changes
11         public override void SetPermissions(IPermissionDefinitionContext context)
12         {
13             context.CreatePermission(name: PermissionNames.Pages_Users, displayName: L(name: "Users"));
14             context.CreatePermission(name: PermissionNames.Pages_Users_Activation, displayName: L(name: "UsersActivation"));
15             context.CreatePermission(name: PermissionNames.Pages_Roles, displayName: L(name: "Roles"));
16             context.CreatePermission(name: PermissionNames.Pages_Tenants, displayName: L(name: "Tenants"), multiTenancySides: MultiTenancySides.Host);
17             context.CreatePermission(name: PermissionNames.Pages_Events_Management, displayName: L(name: "EventsManagement"));
18             context.CreatePermission(name: PermissionNames.Pages_Merch_Management, displayName: L(name: "MerchManagement"));
19             context.CreatePermission(name: PermissionNames.Pages_Purchases_Management, displayName: L(name: "PurchasesManagement"));
20         }
21
22         7 references | Victor Catana, 94 days ago | 1 author, 1 change
23         private static ILocalizableString L(string name)
24         {
25             return new LocalizableString(name, KartSpaceConsts.LocalizationSourceName);
26         }
27     }
28 }

```

Figure 8 - Authorization Provider for permission creation (Code snippet)

that can be configured for the needs of the application. By specifying the tenancy side that can obtain a permission, the permission only becomes available to host accounts or tenant accounts, whichever is specified. This type of behavior allows for further configurability and separates host admins from tenant admins without the need for complex permission checks.

Events can be accessed both by admins as well as regular users, yet can only be managed by admins which have been granted the Events Management permission. This is only done through checking permissions in the front-end, meaning that in the back-end the endpoints are not permission-proofed.

The second and third views are by far the most important in the whole application since they are the main focus of this work. By using these two pages, Merchandise and Orders, the platform is able to make predictions based on what the user has previously purchased or, if he did not purchase anything, it recommends products which are most popular among other users. The Merchandise page features products from all sellers, unseparated to ensure ease of use and straightforwardness ensuring a smooth user experience. Here, regular users can filter though

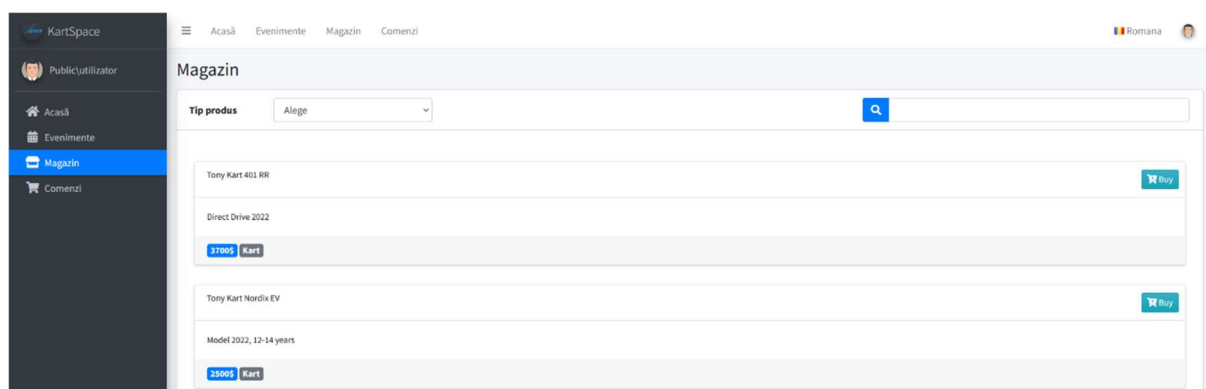


Figure 9 - Merchandise section for a regular user

all available products by either searching for a specific character sequence or choose a product category from the dropdown menu. Also, it is possible for them to purchase a product by pressing the **Buy** button in the top-right corner of each product card, enter their phone number in the pop-up modal and hit save. Tenant users, on the other hand, can not buy products on their tenant accounts. They can all see the products their company has for sale, but only the ones with the *Merchandise Management* permission can create, update or delete merchandise. Such behavior is enforced through permission checks in the JavaScript code of the page, creating and rendering specific buttons and card layouts according to current user data.

After a product has been bought by a user, they can view their order in the Orders page, along with recommendations based on their last purchase. They can filter their purchases by *Order status* or by simply searching for a specific character sequence which filters through all visible columns except order status, since there is already a dropdown for that. Should the user want to check their recommended product out, all they need to do is press the header, which has an embedded hyperlink, and they will be redirected to the Merchandise page. Tenant users, on this page, can not see recommendations since there is no way for them to make purchases on this type of account. Moreover, unless they have the *Purchases Management* permission where they can also edit orders' status, they are not allowed to see order details. Instead, they

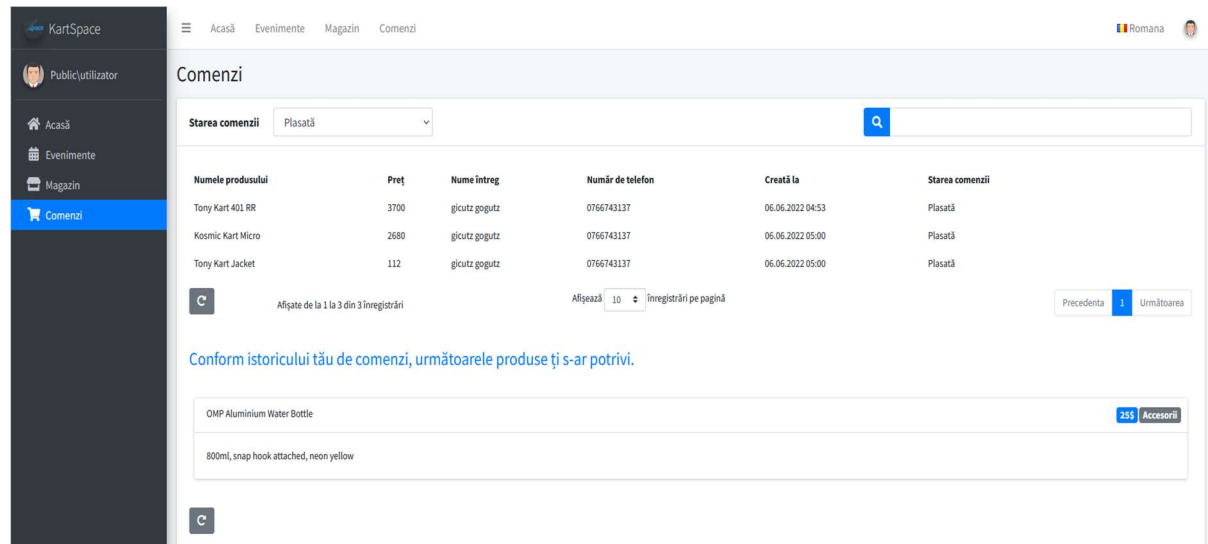


Figure 10 - Orders section for a regular user

will be met with a message telling them that they are not authorized to view orders in the place where the purchases normally render.

## 3.2 Methodology and procedure

In order to provide the user with meaningful and relevant recommendations, the Purchases

```
//Load sample data
var sampleData = new ProductRecommender.ModelInput()
{
    MerchId = lastProduct.Id,
    Category = (int)lastProduct.Category,
    BoughtWithCategory = (int)TipMerch.Accesorii,
};

//Load model and predict output
var result :ModelOutput = ProductRecommender.Predict(sampleData);
```

Figure 11 - Consuming the model

application service creates an input data model for the model to consume and calls the *Predict* method of

*ProductRecommender*. This is done only if the user has purchases, in order to minimize unnecessary database queries and improve code quality. In order to correctly predict recommendations, a maximum entropy classification model trained with the L-BFGS method is used, providing the best accuracy out of 88 explored models after a training period of 600 seconds. In order to train this model, a dataset of 77 entries was used. The dataset features a product's Id and its category along with a co-purchased product's Id and the product category it belongs to. Maximum entropy model is a generalization of linear logistic regression, and the optimization technique implemented in this case is based on the limited memory Broyden-Fletcher-Goldfarb-Shanno method, shortly called L-BFGS. This method is a quasi-Newtonian method which replaces the expensive computation of the Hessian matrix with an approximation but still enjoys a fast convergence rate like Newton's method [12].

Displaying data to the end-user is done through AJAX calls from the JavaScript code of each page to the Application Services. In order to properly display the data paginated, the methods each have a request DTO as one of the input parameters, which contains the keyword a user can search for, the skip count and a maximum result count for pagination. After the method processes the input data

```
public string GetDisplayName(TipMerch enumValue)
{
    string displayName;

    displayName = enumValue.GetType() // Type
        .GetMember(name: enumValue.ToString()) // Me
        .FirstOrDefault() // MemberInfo?
        .GetCustomAttribute<DisplayAttribute>()?
        .GetName(); // string?

    if (string.IsNullOrEmpty(displayName))
    {
        displayName = enumValue.ToString();
    }

    return displayName;
}
```

Figure 12 - Retrieving the Display attribute from an Enum value



and retrieves the necessary information from the database, the results are returned in a paged format through a specific Data Transfer Object. In case of a category filter being used, a standard method is used to retrieve the chosen Enum value's name. This is done by accessing the *Display* annotation of the Enum.

Accessing Application Service methods is possible from JavaScript code thanks to Dependency Injection, being registered to it and allowing it to be used by other classes. ASP.NET Boilerplate registers Application Services automatically by convention and provides ease of use in JavaScript code. Moreover, the Controllers in the Presentation Layer also benefit from conventional routing, making them easy to access and straightforward to use.

### 3.3 Overview of architecture

In order to properly reduce complexity, the application follows the principles of Domain Driven Design and features an N-Layered Architecture with the main layers being the Application Layer, providing Application Services and DTOs, the Domain Layer, called Core in the Solution, featuring Entities and Specifications, the Infrastructure Layer, making use of Entity Framework Core and enabling use of Repositories through Dependency Injection and

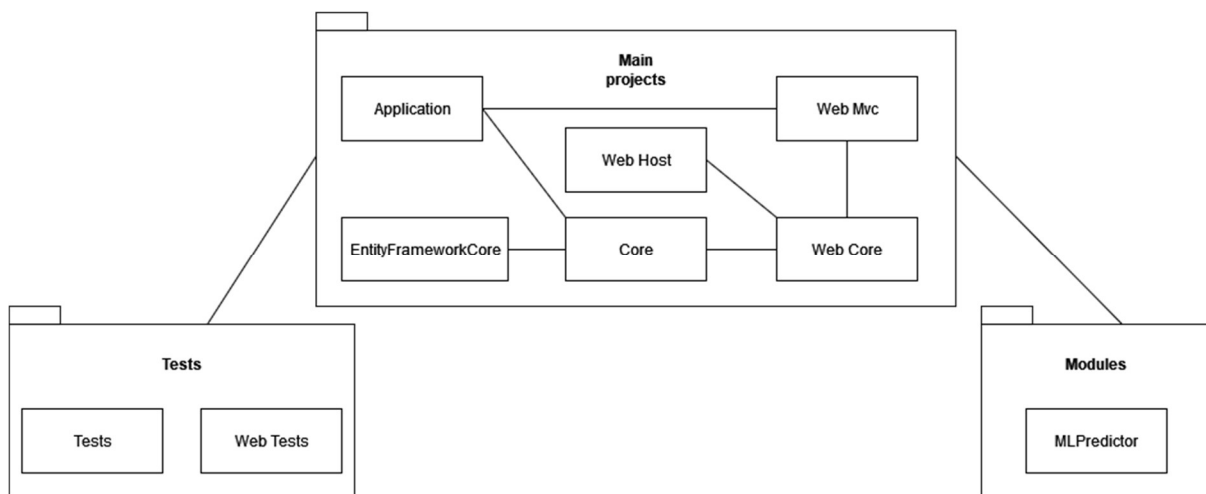


Figure 27 - Structure diagram

Object-Relational Mapping, and the Presentation Layer, represented by an ASP.NET MVC application. Also, a Swagger integration is also provided and features easy access to the API Endpoints. Moreover, in order to separate external components, not bound to the application's behavior, the Machine Learning project, which contains the model that is consumed in the Purchases Application Service, is contained, alike the Test projects, in another Solution Directory, other than the source one where the primary, main projects and related functionalities reside. Moreover, the ML project also contains the training information, dataset and other files which have no connection to the main projects, serving a logical separation as well.

For the code complexity to be reduced and to allow the Application Services not to work with Entities directly, DTOs are used as a situational middle-man between them. In order to allow mapping between them and entities where needed, a Map Profile needs to be provided, inheriting from Profile, in which the Mappings should be created. As stated before, communication between the application and the database is done through Dependency Injection and, by extension,

Figure 13 - Standard layout for each Application Service

Repositories. These come with basic CRUD operations, managed in units of work, that provide a simple development flow. The database the application uses Microsoft SQL Server through Entity Framework Core. It is and should always be managed programmatically, through code,

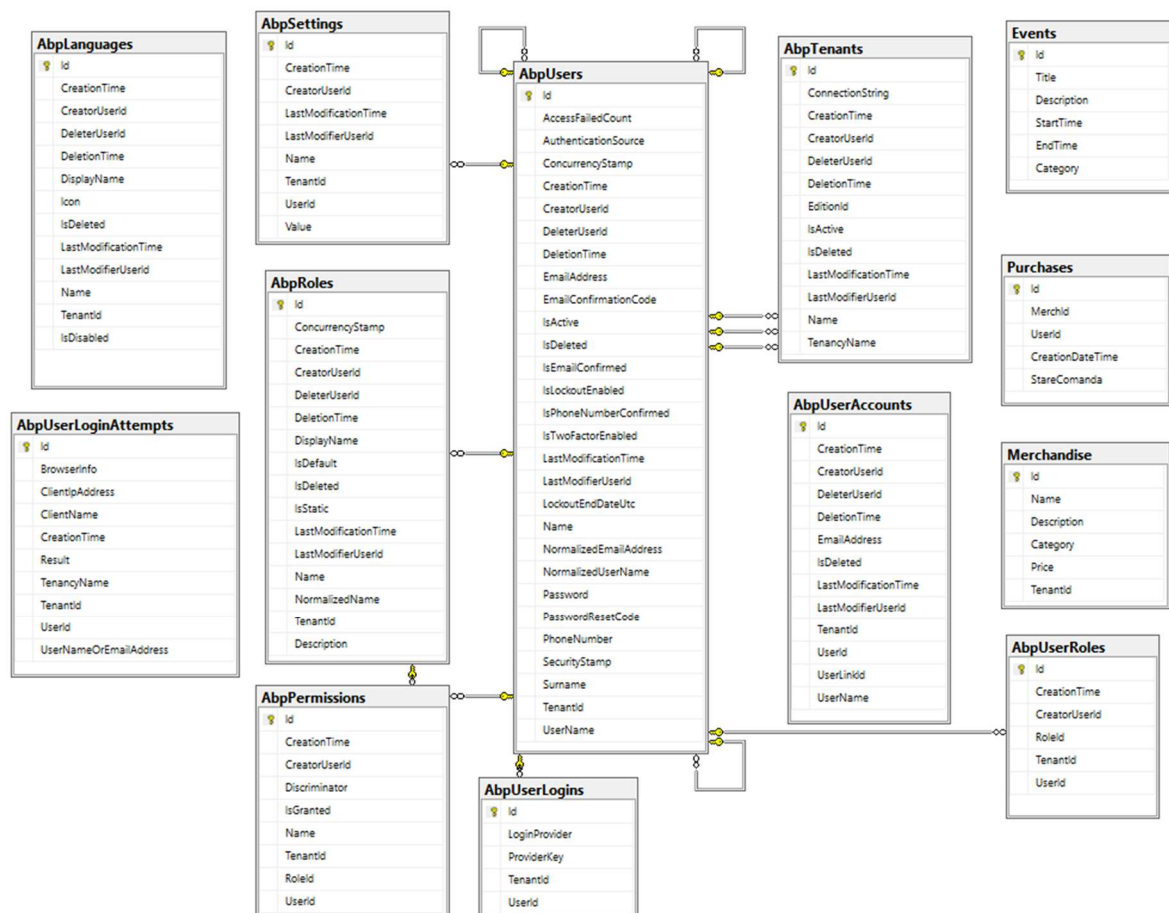


Figure 14 - Database architecture

since changes in the structure could harm the workflow of the framework and prevent it from efficiently performing its tasks. The Database architecture features the central piece of the

puzzle in the application, the User, along with different adjacent entities, such as Roles, Permissions, Tenants, Accounts or Settings, and individual components such as Languages, used for Localization, Login Attempts and Purchases.

Structurally, the solution attempts to follow Domain Driver Design practices and ensure scalability and ease of use. Readability and understandability were also key aspects which were kept in mind when developing this application, as most of the relationships between components are designed to be as direct and simple as possible, allowing for a stranger-friendly approach. Moreover, a clean and well structured project architecture results in more time being dedicated to improving features, instead of fixing dependencies or undergoing exhausting refactoring processes.

Deployment of the application was performed through publishing the MVC project to IIS Manager and assigning an address and a port. This was accomplished by utilizing a built-in feature that ASP.NET Boilerplate provides, a PowerShell script which builds the project into

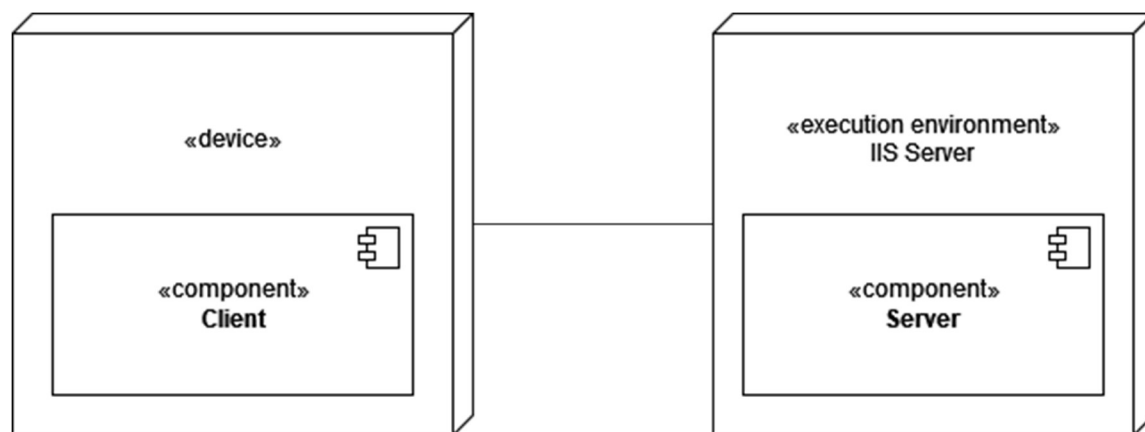


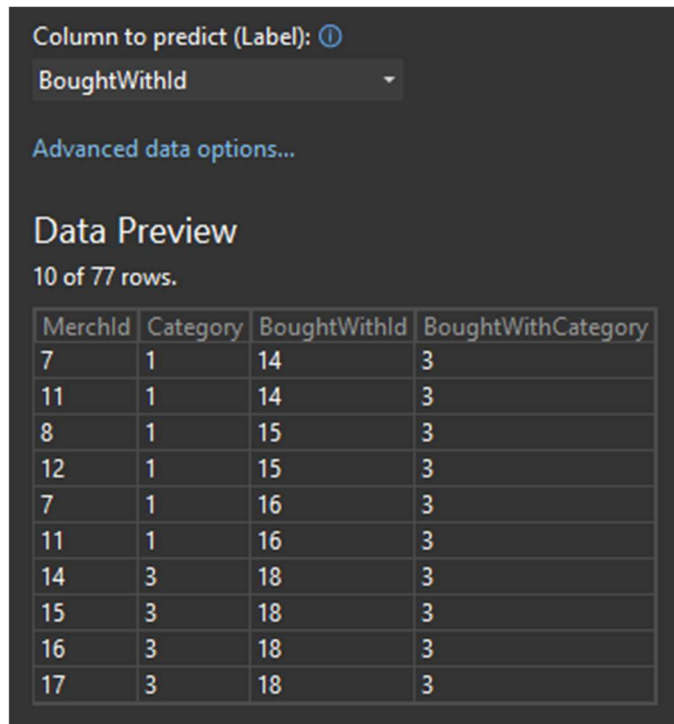
Figure 26 – Deployment diagram


a *release* format, along with any module projects. In our case, deploying the Machine Learning project was necessary as well, since consuming and researching the model is the main aim of this paper. This way, accessing the features is possible without depending on the IDE, simply by entering the address assigned to the application. Just like any other ASP.NET Core application, hosting is done by utilizing the .NET Core Hosting Bundle and configuring an IIS Server after enabling its services.

### 3.4 Implementation

In implementing the features this application uses as a basis for researching modern machine learning framework ML.NET, making use of model builders, complex queries and data rendering strategies was necessary. The ML.NET model builder itself provides a smooth experience, featuring a friendly design. After selecting the dataset for the experiment, the column for the prediction must be chosen, as well as choosing whether or not to ignore certain columns of the dataset. In our case, no column was ignored and the column for the prediction was

BoughtWithId. Moreover, a complete overview throughout the training process is visible



Column to predict (Label): 

BoughtWithId

[Advanced data options...](#)

### Data Preview

10 of 77 rows.

MerchId	Category	BoughtWithId	BoughtWithCategory
7	1	14	3
11	1	14	3
8	1	15	3
12	1	15	3
7	1	16	3
11	1	16	3
14	3	18	3
15	3	18	3
16	3	18	3
17	3	18	3

Figure 15 - Dataset preview

	Trainer	MicroAccuracy	MacroAccuracy	Duration	#Iteration
0	SdcaMaximumEntropyMulti	0.1201	0.1261	1.0	0
1	LbfgsMaximumEntropyMulti	0.1240	0.1136	0.3	1
2	FastTreeOva	0.1794	0.1794	5.3	2
3	FastForestOva	0.1980	0.1697	5.8	3
4	SdcaLogisticRegressionOva	0.1240	0.1136	6.9	4
5	SdcaMaximumEntropyMulti	0.0865	0.0969	0.4	5
6	LbfgsLogisticRegressionOva	0.1240	0.1136	0.7	6
7	LbfgsMaximumEntropyMulti	0.1451	0.1372	0.2	7
8	LightGbmMulti	0.1908	0.1919	0.5	8
9	FastForestOva	0.1948	0.1725	7.4	9
10	FastTreeOva	0.0354	0.0403	7.0	10

Figure 16 - Model training output

through the Output window. Each iteration provides an insight into the trainer used, the duration of the iteration and the accuracies the trainer had with certain settings. However, it is not possible to observe the settings AutoML uses to adjust the trainers, which might feel simplistic in some concerns. However, observing the model trainers throughout the experiment provides an insight into how AutoML works. That is, since different settings for certain algorithms provide slightly different accuracy results, providing an insight into the way that Microsoft have created a solution to the never-ending problem that was lengthy and exhausting model training.

At the end of the experiment, a summary is generated in the model builder interface, as

=====Experiment Results=====					
Summary					
ML Task: Classification					
Dataset: D:\Licenta\aspnet-core\module\KartSpace.MLPredictor\TrainingData.csv					
Label : BoughtWithId					
Total experiment time : 582.24 Secs					
Total number of models explored: 88					
Top 5 models explored					
	Trainer	MicroAccuracy	MacroAccuracy	Duration	#Iteration
35	LbfgsMaximumEntropyMulti	0.2727	0.2675	0.4	35
23	LbfgsMaximumEntropyMulti	0.2727	0.2675	0.4	23
43	LightGbmMulti	0.2606	0.2475	0.6	43
85	FastForestOva	0.2359	0.2217	52.1	85
69	FastForestOva	0.2359	0.2217	27.6	69

Figure 17 - Model training end summary and best models

well as in the output window, the latter also featuring the top 5 models explored throughout the training period. If certain model trainers appear multiple times as it happened in this case, it means that slightly different settings were used for the same trainer which resulted in them having the same accuracy. This only happens in cases where smaller datasets are used since there are limited classifications to be made. Testing the trained model can be done manually in the model builder as well.



Model consumption is featured in the *PurchaseAppService*, along with a mechanism which provides the user with two recommendations based on overall user preference.

```
public async Task<PagedResultDto<PurchaseRecommendationDto>> GetRecommendationsAsync()
{
    var lastPurchaseOrEmpty = _purchaseRepository // IRepository<Purchase,int>
        .GetAll() // IQueryable<Purchase>
        .OrderByDescending(x:Purchase => x.CreationDateTime) // IOrderedQueryable<Purchase>
        .FirstOrDefault(x:Purchase => x.UserId == AbpSession.UserId.Value);

    if (lastPurchaseOrEmpty == null)
    {
        var mostPurchasedQuery:IQueryable<int> = _purchaseRepository.GetAll() // IQueryable<Purchase>
            .GroupBy(m:Purchase => m.MerchId) // IQueryable<IGrouping<_,Purchase>>
            .OrderByDescending(mg:IGrouping<int,Purchase> => mg.Count()) // IOrderedQueryable<IGrouping<_,Purchase>>
            .Take(2) // IQueryable<IGrouping<_,Purchase>>
            .Select(m:IGrouping<int,Purchase> => m.Key);

        var queryRez :List<int> = await AsyncQueryableExecuter.ToListAsync(mostPurchasedQuery);

        var recList = queryRez.Select(x:int =>
        {
            var merch = _merchRepository.Get(x);

            var lista = new PurchaseRecommendationDto
            {
                Name = merch.Name,
                Price = merch.Price,
                CategoryName = _merchAppService.GetDisplayName(merch.Category),
                Description = merch.Description
            };

            return lista;
        }).ToList();

        var recDisplay = new PagedResultDto<PurchaseRecommendationDto>(recList.Count, recList);
        return recDisplay;
    }
}
```

Figure 19 - Product recommendation for users with purchase history, using the trained model

The method searches for any purchases the user might have and recommends a product based on the last one. Should he be a new user that has not yet purchased anything, it provides two products that are preferred by other users instead. Afterwards, the model input is created with the last purchased product and is given a product category chosen by the developer. The model is then used to predict a product Id based on the information provided and, after retrieving the product information from the database, the method returns the recommended item in a paginated list format.

```
ajax: function (data, callback, settings) {
    var input = $('#MerchandiseSearchForm').serializeFormToObject(true);
    var tipMerch = $('#TipMerch').val();
    input.maxResultCount = data.length;
    input.skipCount = data.start;

    _merchService.getMerchList(input, tipMerch).done(function (result) {
        callback({
            data: result.items,
            recordsTotal: result.totalCount,
            recordsFiltered: result.totalCount
        });
    }).always(function () {
        abp.ui.clearBusy(_$merchTable);
    });
}
```

Figure 20 - AJAX call for DataTable data retrieval

In order to receive data from the application services under DataTables, AJAX calls were used, according to server-side data sourcing, that retrieved information which allowed the table to also have pagination usable. Details like input keyword, skip count and maximum page

```

55 if (abp.auth.isGranted('Pages.Merch.Management')) {
56     return [
57         <div class="card">
58             <div class="card-header">
59                 <h3 class="card-title">${row.name}</h3>,
60                 <div class="card-tools">
61                     <button type="button" class="btn btn-sm bg-secondary edit-merch" data-merch-id="${row.id}" data-toggle="modal" data-target="#MerchEditModal">,
62                     <i class="fas fa-pencil-alt"></i> ${l('Edit')}',
63                 </button>,
64                     <button type="button" class="btn btn-sm bg-danger delete-merch" data-merch-id="${row.id}" data-merch-name="${row.name}">,
65                     <i class="fas fa-trash"></i> ${l('Delete')}',
66                 </button>,
67             </div>
68         </div>,
69         <div class="card-body">
70             ${row.description},
71         </div>
72         <div class="card-footer">
73             <span class="badge badge-primary">${row.price}</span> <span class="badge badge-secondary">${row.categoryName}</span>,
74         </div>
75     </div>
76 ].join('');
77 }
78 else {
79     if (abp.session.multiTenancySide == 2) {
80         return [
81             <div class="card">
82                 <div class="card-header">
83                     <h3 class="card-title">${row.name}</h3>,
84                 <div class="card-tools">
85                     <button type="button" class="btn btn-sm bg-gradient-info buy-merch" data-merch-id="${row.id}" data-toggle="modal" data-target="#MerchBuyModal">,
86                     <i class="fas fa-cart-plus"></i> ${l('Buy')}',
87                 </button>,
88             </div>
89         </div>,
90         <div class="card-body">
91             ${row.description},
92         </div>
93         <div class="card-footer">
94             <span class="badge badge-primary">${row.price}</span> <span class="badge badge-secondary">${row.categoryName}</span>,
95         </div>
96     </div>
97 ].join('');
98     }
99     else {
100         return [
101             <div class="card">
102                 <div class="card-header">
103                     <h3 class="card-title">${row.name}</h3>,
104                 <div class="card-tools">
105                     <span class="badge badge-primary">${row.price}</span> <span class="badge badge-secondary">${row.categoryName}</span>,
106                 </div>
107             </div>,
108             <div class="card-body">
109                 ${row.description},
110             </div>
111         </div>
112     ].join('');
113     }
114 }

```

Figure 21 - DataTable rendering of merchandise depending on user type and permissions

result count, as well as the category selected by the user (if any), were used in devising the method call. The data would then be received under Callback form and would be used to configure the table. Furthermore, as the data had to be rendered differently for each type of user, the table was user more as a container that offered pagination rather than a table, with only one column being used and card elements being created through JavaScript code, rather than Cshtml. By accessing authorization data, it renders management buttons such as Edit and Delete. Similarly, the Create button is contingent on having the same permission, although the check is done in the Cshtml file. Should the user not have authorization to see the management buttons yet is still a tenant user, a different card layout is generated in order to remove the empty toolbar where the buttons should be. For the regular users, the card looks the same as for tenant administrators, only with a Buy button instead of the management ones. Should one

of the buttons be clicked, an event is triggered in the JavaScript code and, by calling an Action in the Controller, opens up a modal depending on the button that was pressed.

### 3.5 Evaluation

Evaluation for the platform is done in two ways, each providing advantages for their respective fields of application. In order to evaluate the trained model, testing is done manually and is done through the Model Builder interface. However, when evaluating the Application Services and Web API Controllers, automated testing is the main tool and is performed through different types of tests such as integration, white-box and black-box testing.

By entering the input parameters, the model predicts the 5 most suitable results and displays them in order of confidence. Currently there is no method that automates this process

**Best model:**  
**Accuracy:** 0.2727  
**Model:** LbfgsMaximumEntropyMulti

**Try your model**

**Sample data**  
The following fields are pre-filled by a row of your data.

**MerchId**  
11

**Category**  
1

**BoughtWithCategory**  
3

**Predict**

**Results**  
Click Predict to see the predicted result

Figure 22 - ML Model evaluation, in Model Builder interface

since it relies upon human insight and intuition, needing to evaluate preferences. Should the results not be satisfactory, the model can be retrained in order to obtain further improvements in Accuracy and even change the model, should the framework find a more suitable one for the new dataset.

Testing the services and controllers, however, is automated through xUnit and can be set to run before every build or arbitrarily started by developers. The sample data is provided and used to ensure the natural and working flow of the application through various testing scenarios. Should any of the tests fail, the developer is notified of the issue and detailed



diagnosis is provided as to the reason of failure. There are several test scenarios in the Test

```
[Fact]
0 references | Victor Catana, 47 days ago | 1 author, 1 change
public async Task Events_BigBang_Test()
{
    /**
     * Testare big-bang traseu A-B-C
     * Contine testare unitara pentru modulele A, B, C individuale si testare integrare A-B-C
     */

    var eveniment = new EventDto
    {
        Title = "Campionat Etapa 1",
        Description = "Stock 600, Juniori, Stock 1000",
        StartTime = DateTime.Now,
        EndTime = DateTime.Now
    };

    //Testare modul A
    var output :EventDto = await _eventAppService.CreateAsync(eventiment);
    output.Title.ShouldBe("Campionat Etapa 1");

    //Testare modul B
    output.Title = "Campionat";
    output = await _eventAppService.UpdateAsync(output);
    output.Title.ShouldBe("Campionat");

    //Testare modul C
    await _eventAppService.DeleteAsync(output);
    var items:PagedResultDto<EventDto> = await _eventAppService // IEventAppService
        .GetAllAsync(input:new PagedEventResultRequestDto { Keyword = null, MaxResultCount = 20, SkipCount = 0 }); // Task<PagedResultDto<>>
    items.Items.ShouldNotContain(output);

    //Testare integrare cu module combine
    var data:EventDto = await _eventAppService.CreateAsync(eventiment);
    data.Title = "Campionat";
    data = await _eventAppService.UpdateAsync(data);
    await _eventAppService.DeleteAsync(data);
    items = await _eventAppService // IEventAppService
        .GetAllAsync(input:new PagedEventResultRequestDto { Keyword = null, MaxResultCount = 20, SkipCount = 0 }); // Task<PagedResultDto<>>
    items.Items.ShouldNotContain(data);
}
```

Figure 23 - Test scenario for Events CRUD

project for the Application layer, two of which are integration tests that use Big-Bang and Top-Down testing approaches.

### 3.5.1 Test environment

As stated before, evaluation of the trained model can be done by accessing the Model Builder and feeding input parameters to the *Evaluate* interface menu. The tests for the rest of the application are to be found in the *test* solution directory. Inside it are 2 different projects, one for testing features related to the Application Layer and one for tests related to the Presentation Layer, having the appendix *Web* in the name. The Web Test project should also be the one to contain UI evaluation, such as tests featuring the Selenium framework tools. Currently, tests are performed using xUnit and Shouldly, the former being a free open-source unit testing tool developed by the original author of NUnit. The latter is an

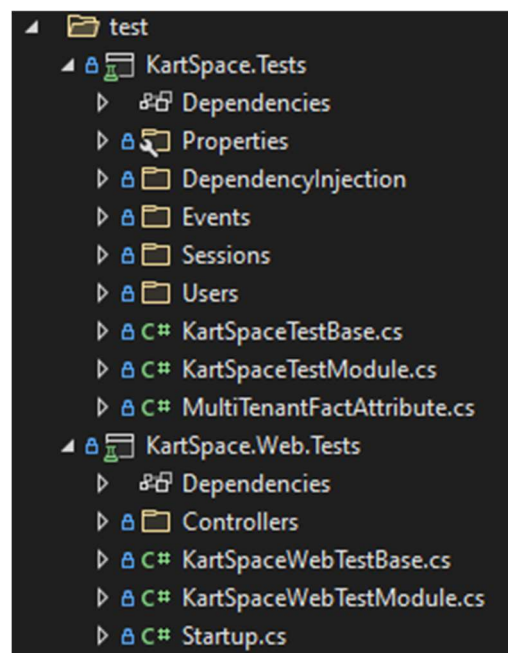
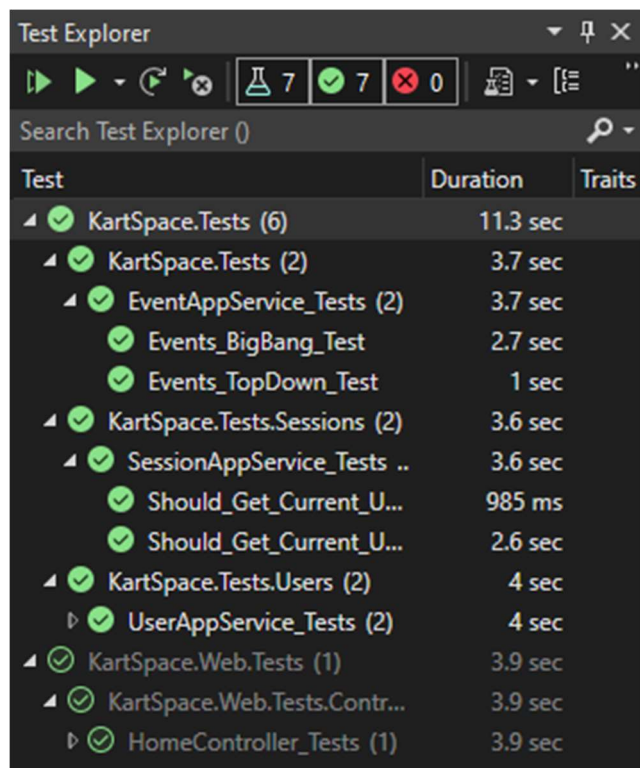


Figure 24 - Test projects' layout within the solution

assertion framework with over 22 million NuGet package downloads which focuses on giving error messages when assertion fails while being simple and terse [13].

### 3.5.2 Results

Evaluation results for the trained model can be observed in the Model Builder interface under the Results section. For the automated tests, Visual Studio provides a Test Explorer for testing to be done easily in. There, the tests can be run, debugged and their results can be seen for diagnosis.



Test	Duration	Traits
▲ ✓ KartSpace.Tests (6)	11.3 sec	
▲ ✓ KartSpace.Tests (2)	3.7 sec	
▲ ✓ EventAppService_Tests (2)	3.7 sec	
✓ Events_BigBang_Test	2.7 sec	
✓ Events_TopDown_Test	1 sec	
▲ ✓ KartSpace.Tests.Sessions (2)	3.6 sec	
▲ ✓ SessionAppService_Tests ..	3.6 sec	
✓ Should_Get_Current_U...	985 ms	
✓ Should_Get_Current_U...	2.6 sec	
▲ ✓ KartSpace.Tests.Users (2)	4 sec	
▶ ✓ UserAppService_Tests (2)	4 sec	
▲ ✓ KartSpace.Web.Tests (1)	3.9 sec	
▲ ✓ KartSpace.Web.Tests.Contr...	3.9 sec	
▶ ✓ HomeController_Tests (1)	3.9 sec	

Figure 25 - Tests completed, in Test Explorer of Visual Studio

## **4. Conclusion and future work**

### **4.1 Conclusion**

The aim of this thesis was to create a medium which integrates ML.NET, a relatively new Machine Learning framework released in 2018, into a web application built with traditional and reliable frameworks and techniques which would allow us to research the behavior and process of training and consuming an AI recommendation model without having to improvise or rely on an unstable foundation which might compromise the integrity of the experiment. The scientific contributions this work brings are related to the practical and theoretical aspect of creating a mixture between traditional and modern technologies and highlighting the bridge points between the two.

On the theoretical side, comparisons were made between the technologies used for the application, motivating why certain technologies have various advantages over others or why they were chosen despite lacking features that their counterparts provided. Choosing to follow neither a completely traditional nor a completely modern route but a mixture between the two for the experiment was necessary due to the nature of technology in this current day and age. Rushing the natural evolution of software could lead to potential issues due to more modern solutions not being thoroughly enough researched and tested, which is why making a compromise and choosing to integrate new software elements with older ones seemed to be the ideal choice, providing a stable and trustworthy foundation for an experimental feature.

As to the practical aspects of the thesis, the complete lifecycle of the application was outlined, featuring the initial analysis, the architecture and procedures that the implementation followed, as well as final evaluation through both manual and automated testing. Moreover, implementation specifics and communication methodology between the client application and the prediction model were highlighted in order to allow for deeper understanding of the process and reveal the complex details of the thesis' focus. Furthermore, the implementation of necessary features which allowed for the integration to behave as expected was also described in detail to provide no shortage of information about the experiment's particularities.

As a whole, the personal contributions this thesis brings are related to the compatibility between modern and traditional technologies proven through theoretical analysis as well as practical implementation of these technologies in a platform which serves the end-user. Moreover, bringing together existent yet separate solutions to a problem and unifying them

into a single solution for a dedicated purpose along with an intelligent component to enhance the user experience is ultimately what makes this work unique.

## 4.2 Future work

Given the modern and experimental aspect of the machine learning component implemented and documented in this thesis, much of the possible future applications of the component is based on further developments in the Artificial Intelligence field of research. The experiment focuses on only one type of machine learning task, which is prediction through data classification, whereas other machine learning tasks unresearched by this paper could provide further information about the behavior of machine learning under the situation proposed by this thesis.

Currently, the model used to predict user preference and recommend products was trained with a manually generated dataset with under one hundred data points, whereas future improvements to the application could envision a mechanism for the dataset to be dynamically created, by identifying which products were bought after the software has recommended them and enlarge the dataset to eventually retrain the model and achieve even higher predictive accuracy.

All in all, this thesis achieves its main and primary aim of researching the behavior of a modern machine learning framework alongside a traditional multi-page application and highlighting the findings in order to contribute to the current research of artificial intelligence.

## Bibliography

- [1] SAS Institute, "Artificial Intelligence (AI) - What it is and why it matters | SAS," [Online]. Available: [https://www.sas.com/en\\_us/insights/analytics/what-is-artificial-intelligence.html](https://www.sas.com/en_us/insights/analytics/what-is-artificial-intelligence.html).
- [2] Volosoft, "AspNet Boilerplate - Web Application Framework," Volosoft, 2013. [Online]. Available: <https://aspnetboilerplate.com/>.
- [3] A. Lock, ASP.NET Core in Action, Second Edition, 2021.
- [4] S. Smith, Overview of ASP.NET Core MVC, Microsoft, 2018.
- [5] A. Freeman, Pro ASP.NET Core MVC, Apress, 2016.
- [6] Microsoft Corporation, "Overview of Entity Framework Core - EF Core | Microsoft Docs," [Online]. Available: <https://docs.microsoft.com/en-us/ef/core/>.
- [7] J. P. Smith, Entity Framework Core in Action, Second Edition, Manning, 2021.
- [8] Z. Ahmed, S. Amizadeh and others, *Machine learning at Microsoft with ML.NET*, 2019.
- [9] e. a. L. Quintanilla, "How to use the ML.NET automated ML API - ML.NET | Microsoft Docs," 12 10 2021. [Online]. Available: <https://docs.microsoft.com/en-us/dotnet/machine-learning/how-to-guides/how-to-use-the-automl-api>.
- [10] P. e. a. Gijbbers, An open source AutoML benchmark., arXiv preprint arXiv:1907.00909, 2019.
- [11] Microsoft Corporation, "How to choose an ML.NET algorithm - ML.NET | Microsoft Docs," [Online]. Available: <https://docs.microsoft.com/en-us/dotnet/machine-learning/how-to-choose-an-ml-net-algorithm>.
- [12] Microsoft Corporation, "LbfgsMaximumEntropyMulticlassTrainer Class (Microsoft.ML.Trainers) | Microsoft Docs," [Online]. Available: <https://docs.microsoft.com/en-us/dotnet/api/microsoft.ml.trainers.lbfgsmaximumentropy-multiclass-trainer?cv=1&view=ml-dotnet>.
- [13] X. B. A. E. P. v. W. J. G. D. Newman, "shouldly/shouldly: Should testing for .NET - the way Asserting \*Should\* be!," [Online]. Available: <https://github.com/shouldly/shouldly>.
- [14] Volosoft, "Articles Tutorials | AspNet Boilerplate," [Online]. Available: <https://aspnetboilerplate.com/Pages/Documents/NLayer-Architecture>.