**STATE UNIVERSITY**
**OF INFORMATION AND COMMUNICATION TECHNOLOGIES**
**EDUCATIONAL AND SCIENTIFIC**
**INSTITUTE OF INFORMATION TECHNOLOGIES**
**DEPARTMENT OF SOFTWARE ENGINEERING**

# QUALIFICATION WORK

on the topic: «Software development in C# for content rating and recommendations for a website about roller skating»

for a bachelor's degree
with the specialty of 121 software engineering
educational and professional program «Software engineering»

*The qualification work contains the results of own research. The use of ideas, results, and texts of other authors are linked to the appropriate source*

_____ Teymur ABBASOV

Done by: student of the PPZ-51 group
Teymur ABBASOV
Head: Igor GAMANIUK
Reviewer: _____

**Kyiv 2024**

**STATE UNIVERSITY**
**OF INFORMATION AND COMMUNICATION TECHNOLOGIES**
**EDUCATIONAL AND SCIENTIFIC**
**INSTITUTE OF INFORMATION TECHNOLOGIES**
**DEPARTMENT OF SOFTWARE ENGINEERING**

Department of software engineering
Degree of higher education - «Bachelor»
Specialty – 121 «Software engineering»
Educational and professional program «Software engineering»

**APPROVED**
Head of Department
Software engineering
_____ Iryna ZAMRII
“____” _____2024

# TASK
# FOR A QUALIFICATION WORK

Teymur ABBASOV

1. The topic: «Software development in C# for content rating and recommendations for a website about roller skating»
Head: senior lecturer of the Software engineering department Igor GAMANIUK

approved by the order of the State University of Information and Communication Technologies of February 27, 2024 No. 36.

2. Deadline for submitting the qualification work is May 28, 2024.

3. Input data to qualification work
    3.1 Official Microsoft documentation;
    3.2 Official Vue.js documentation;
    3.3 Scientific and technical literature.

4. Content of the settlement and explanatory note (list of issues to be developed)
    4.1 Analysis of the subject field;
    4.2 Review of chosen technologies and software development tools for creating a website;

4.3 Software development;
4.4 Testing;
4.5 Conclusions.

5. List of graphic material: *presentation*
5.1 Purpose, object, and subject of research;
5.2 Tasks of the graduate thesis;
5.3 Software requirements;
5.4 Development software;
5.5 Analysis of similar programs;
5.6 Use case diagram;
5.7 Data base diagram;
5.8 Activity diagram of review creation;
5.9 State diagram of web pages;
5.10 Package diagram;
5.11 Screen forms;
5.12 Approval of research results;
5.13 Conclusions.

The date of issue of the task is February 28, 2024.

**CALENDAR PLAN**

| № in order | The name of the stages of the bachelor's work | The term of performance of work stages | Note |
|---|---|---|---|
| 1 | Selection of scientific and technical literature | 28.02-06.03.2024 | Done |
| 2 | Analysis and research of existing analogues | 07.03-13.03.2024 | Done |
| 3 | Modeling, system design | 14.03.24-16.04.2024 | Done |
| 4 | Development of the functionality of the system | 17.04.24-28.04.2024 | Done |
| 5 | Conclusions and design of the work | 29.04-05.05.2024 | Done |
| 6 | Development of mandatory demonstration materials | 06.05-12.05.2024 | Done |
| 7 | Preliminary work protection | 13.05-31.05.2024 | Done |

Student _____ Teymur ABBASOV

Head: _____ Igor GAMANIUK

# ABSTRACT

The text part of the bachelor thesis: 50 pp., 3 tables, 26 pictures, 1 annex, 13 refferences.

**Object of research:** content rating and recommendations for a website about roller skating.

**Subject of research:** software development for content rating and recommendations for website about roller skating.

**The purpose of the work:** support the processes of content rating and recommendations for a website about roller skating.

Research includes analysis of how rating and recommendation systems are used by websites, development of architectural and functional design of a website about writing reviews regarding roller skates with a rating and recommendation system specially configured for roller skates.

Options for recommendation systems are considered, websites containing roller skates are analyzed, their strengths and weaknesses are identified.

For designing stage of the work, UML database diagram, use case diagram, activity diagram were made. The program was made using SPA architecture. C# programming language and ASP.NET core backend framework is used for server side RESTful API. Vue.js 3 is used as frontend framework that provides SPA functionality running in Node,js runtime. For database SQLServer was used.

**CONTENT**

# INTRODUCTION

Nowadays, the Internet contains an exorbitant amount of information, and without a system of ratings and recommendations, it is difficult to make a choice in a product. One of such products is roller skates.

**Object of research:** content rating and recommendations for a website about roller skating.

**Subject of research:** software development for content rating and recommendations for website about roller skating.

**The purpose of the work:** support the processes of content rating and recommendations for a website about roller skating.

Following tasks need to be solved for accomplishing stated goal of the work:

- Research the subject area related to ratings and recommendation systems used on websites dedicated to roller skating.

- Analyze and select most appropriate technologies and tools for website development. Define functional and nonfunctional requirements.

- Design and develop a website dedicated to roller skating, including rating and recommendation systems.

- Conduct manual testing of the application to identify shortcomings, errors or behavior not intended by the application.

Website was developed with SPA architecture, in which frontend and backend are separated. On client side Vue.js 3 is used for SPA functionality that allows for dynamic change of information on the webpage without reloading page with each interaction with links and redirection. Node.js is used for creating web server environment for users to connect to. C# language and therefore ASP.NET core is used for backend server environment, which provides RESTful Web API functionality and handles interaction with SQLServer database, while it’s HTTPS endpoints are only accessible for dedicated Node.js web server. To develop the

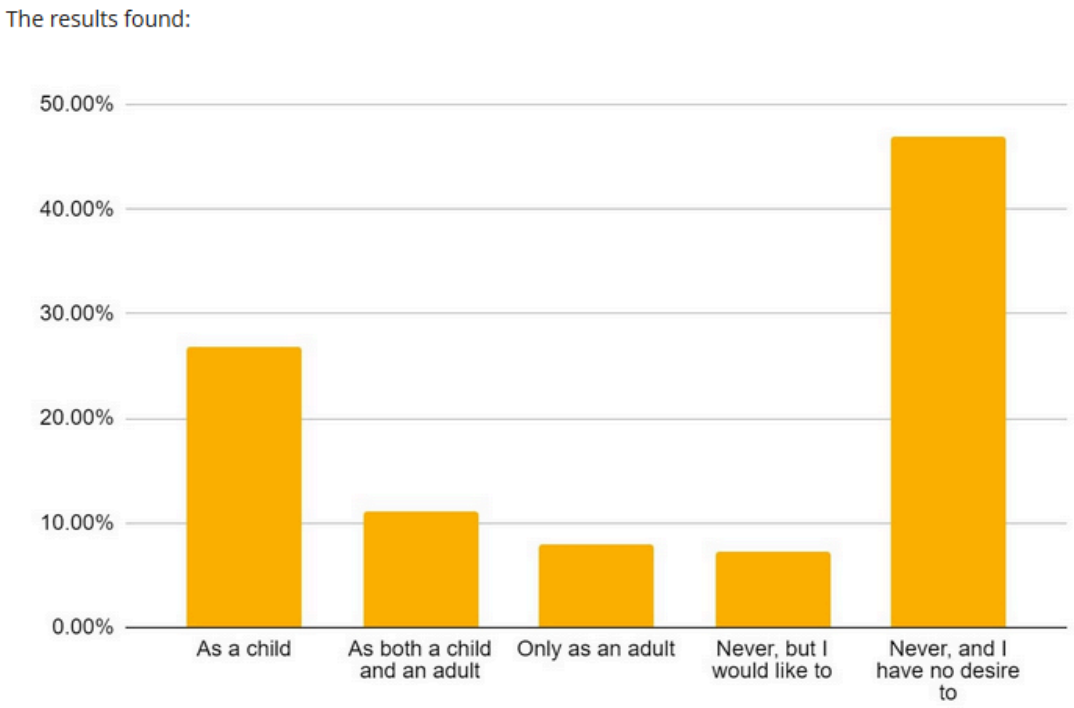
website, the following were chosen: C# language, .NET 8 development platform, ASP.NET backend framework, Vue.js frontend framework and SQL Server database.

# 1 ANALYSIS OF THE SUBJECT FIELD

## 1.1 Subject relevancy

Nowadays internet users, in any market, have access to a wide range of products, each of which attracts its own target audience and for this purpose has a unique combination of design, quality and price. Although this is a quality of a well balanced competitive market, it carries with it a problem of decision paralysis, in which the buyer is unable to make a purchase due to the fact that he/she spends too much time choosing the best product, when there are too many similar products. Situation gets worse considering the fact that in past 20 years, with the development and spread of the Internet and smart devices, the amount of information being posted, sent and collected, related to web, app, and SNS platforms, has increased significantly [1]. So it's even harder for user to find the best source of information about the product to make up his mind about purchase.

Despite the fact that many popular websites have sections for specific types of products, some do not have convenient and specifically dedicated websites for them to fully satisfy users. One such product is roller skates. They are quite popular. Research[2] shows that, for example, in UK 45.9% of adults have at least tried rollerblading. 26.8% were skating in their childhood, 11% still rollerblade into their adulthood, and 8% started rollerblading as an adults.




Picture 1.1

They have many different elements such as: wheel hardness, wheel diameter, frame type, boot type etc. On websites where a wide range of completely different products is presented, you will have to create a detailed individual set of filter parameters for each product. This can overwhelm the UX.

Therefore, in view of the above, modern users are overloaded with information quite often. One way technology can solve this is by using rating system. Today, most E-commerce websites implement product rating and comment system, to get feedback from customers, create trust in product and assure product quality. By analyzing experience of other users, customer becomes more confident in his decision to buy product that satisfies his needs. Generally, websites provide users with the opportunity to rate a product or service using a 5 star rating (which is a type of Likert scale).

Table.1.1

Likert scale

| Strongly dislike | Dislike | Neutral | Like | Strongly like |
|---|---|---|---|---|
| ★☆☆☆☆ | ★★☆☆☆ | ★★★☆☆ | ★★★★☆ | ★★★★★ |

In addition to ratings, users also read blogs, articles and watch video reviews from opinion leaders such as popular influencers or official experts. It is worth highlighting that many users often trust independent experts who are recognized through popularity, high ratings and, therefore, placed at the top of the list of review writers in their field.

Additionally platforms are increasing their collection of information that can be used to identify users' preferences. In case of social media platforms, they acquire a wide range of data, such as information about followers related to the user, information posted by the user, tweet data and more[1]. That is later being used in recommendation systems.

Recommendation systems are information filtering systems that provide a personalized item recommendation to a user in a service environment that collects and holds different types of data. A need in such systems arises due to the fact that people like to have wide range of options, but at the same time, as was previously explained, their satisfaction decreases with increase in difficulty of selection [1]. So recommendation systems helps introduce users to new products or content that they are likely to be interested in and reduce difficulty in selection by showing relatively small sample of products.

There are 3 main types of recommendation models: Content-Based Filtering uses simple user actions such as likes, ratings, view history or favoured product to determine which products are similar to the one he/she likes. It doesn't require much information about user, but needs a lot of aspects about products. If we take movies as example: date of release, genre, actors, director, length, topics in

the movie etc.

For machine learning, to convert raw text into number based vectors tf–idf algorithm is used. One of the downsides of such model is that it is bad at introducing new products that aren't similar but can be interesting for user; Collaborative Filtering uses information about user and his/her preferred products to recommend such products to other similar users. Unlike Content-Based Filtering, Collaborative Filtering doesn't need detailed information about products, but data about users themselves to create group out of them and then make recommendations. This model can be classified into: Memory-Based Collaborative Filtering which has simple approach with looking at similar users and assuming they will continue to have the similar preferences in the future. Such method allows use of simple algorithms that don't require machine learning. Such as k-nearest neighbours algorithm. The downside is that it can't see more nuanced patterns and is less accurate.

Model-Based Collaborative Filtering uses a statistical or machine learning model to identify and exploit hidden links and patterns in the data. It is more accurate and more complex. But even with machine learning model, Collaborative Filtering has problems. Such model demands big amount of highly active users. When website scales (more users and more products are being added to the system) recommendations get worse. New products can't get recommended if no one tries them out; Hybrid System combines 2 approaches together to complement each other's weaknesses. The main goal is to fight with the recommendation of new products that few people have tried so far and to improve accuracy. This can be implemented as 2 systems operating in parallel or as a combined model. Considering how much information begins to be collected about users and products as a website grows, this approach is currently the most effective.

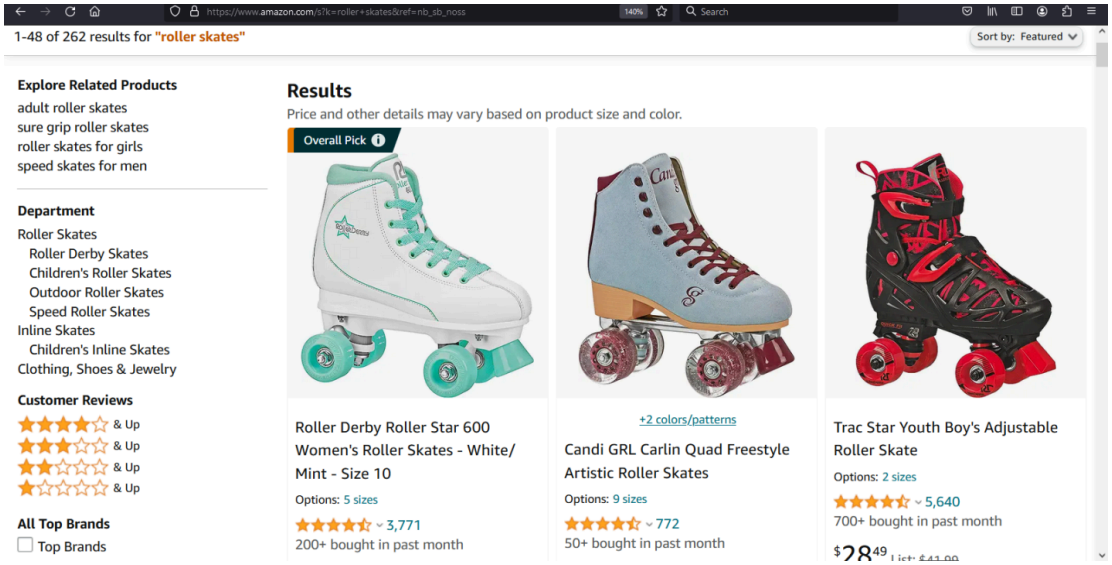
## 1.2 Analysis of similar programs

Analysis will mainly be focused on implementation of rating system and tool for filtering on e-commerce websites for roller skates. It's hard to understand which recommendation system is implemented. Although since all of them were tested without signing in. They probably mainly use Context-based filtering. Showing products that are the most similar and have high ratings.

Most of the websites dedicated for roller skates are online shops. Many allow searching, filtering, commenting and rating products. But there aren't many that provide more detailed reviews.

### 1.2.1 Amazon.com

It is one of the biggest e-commerce websites in the world. Even though it doesn't only sell roller skates, it provides broad and detailed functionality and is feature reach.




Picture 1.2

Pros:

- Large number of users. Allows you to fine-tune personalized recommendations.

- Has a wide range of filters including:

- By rating

- By brand

- By type of roller skates

- By price

- By wheel type

- By color

- By materials

- It shows the percentage distribution of ratings and the number of ratings

- There is section for top rated similar products

- There is section which compares similar products

- Technical detail section is informative

Cons:

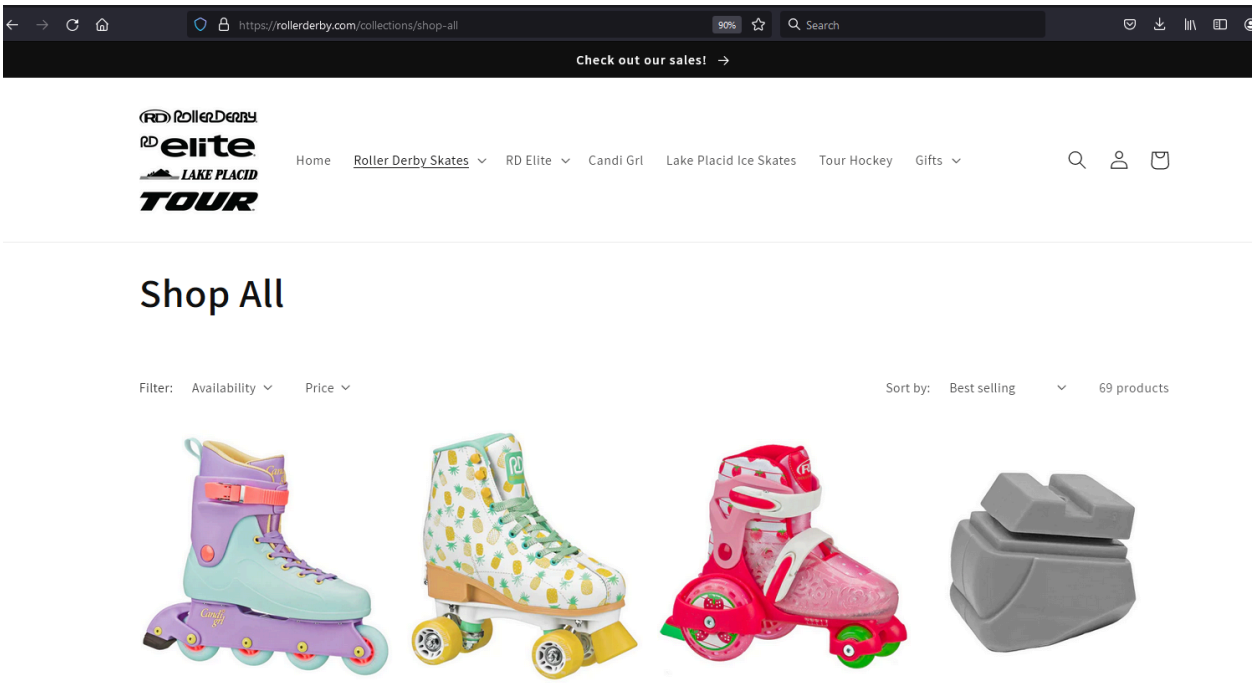
- Doesn’t allow you to create your own extensive review with sections dedicated for different aspects of roller skates.

- Isn’t fully dedicated for roller skates. You will need to know what you are looking for.

- Even though it has many filters there are no filters for wheel size and hardness.

### 1.2.2 Rollerderby.com

Rollerderby.com is a e-commerce website that is dedicated for selling roller skates related products of it's own brands. Product selection is small (only 69 products) and because of that there aren't many filters. The use of rating and review system is limited but it allows to leave comments on product page.



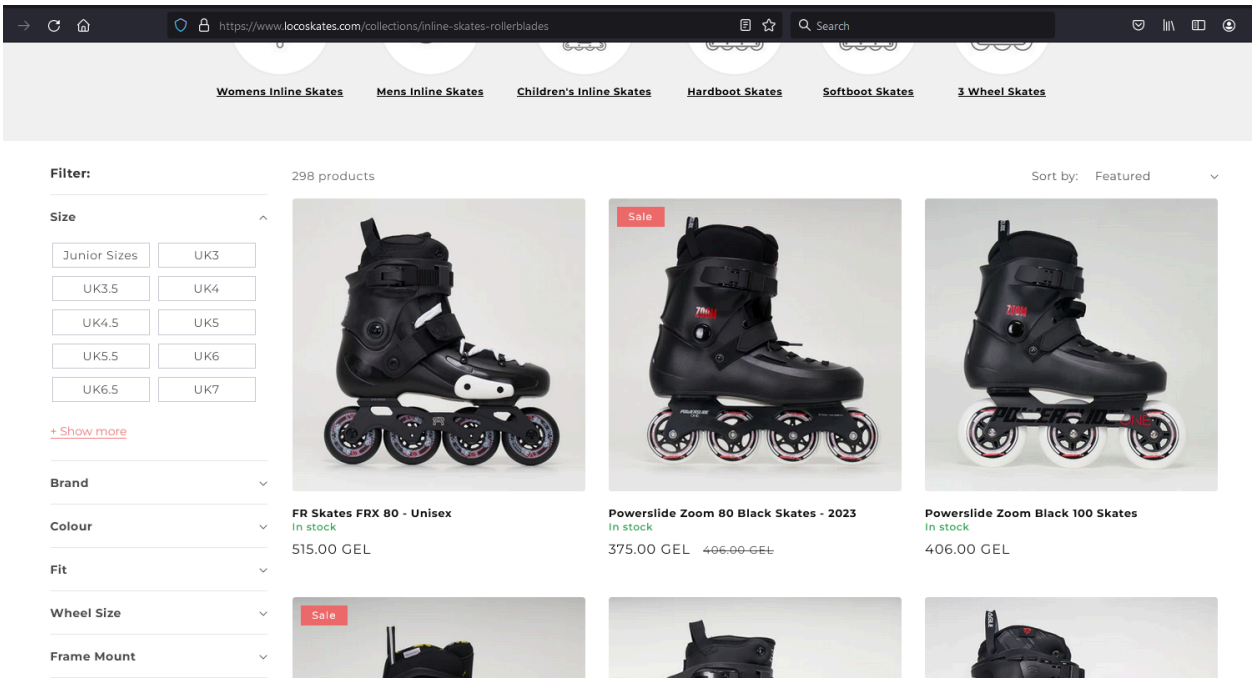

Picture 1.3

Cons:

- Doesn't allow you to create your own extensive review with sections dedicated for different aspects of roller skates.
- Filters limited only to being in stock and price range
- Product pages are lacking in technical information.
- You can't sort by rating.

### 1.2.3 Locoskates.com

Locoskates.com is a e-commerce website that is dedicated for selling roller skates related products. It has a physical shop in England. It has wide range of products related to roller skates. It has good UX: all products are well categorized and broad range of filters allows for quick and easy searching. Product descriptions are full and informative.




Picture 1.4

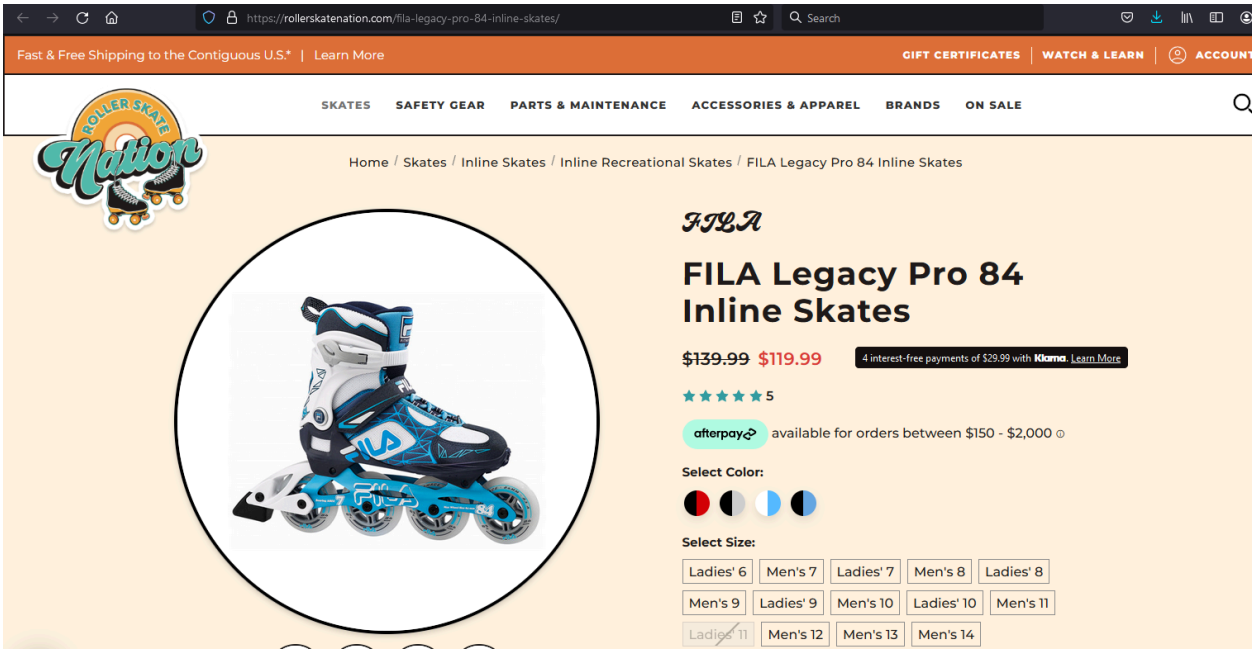
Cons:

- You can't leave comments on product page or rate the product.
- Doesn't recommend you similar products. Only related protective equipment.

### 1.2.4 Rollerskatenation.com

Rollerskatenation.com is a e-commerce website that is dedicated for selling roller skates related products. Their products are well categorized and stylized.

There is rating system on product page but it is only seen on it and can't be seen while searching or looking at catalog. It doesn't allow you to leave comments.




Picture 1.5

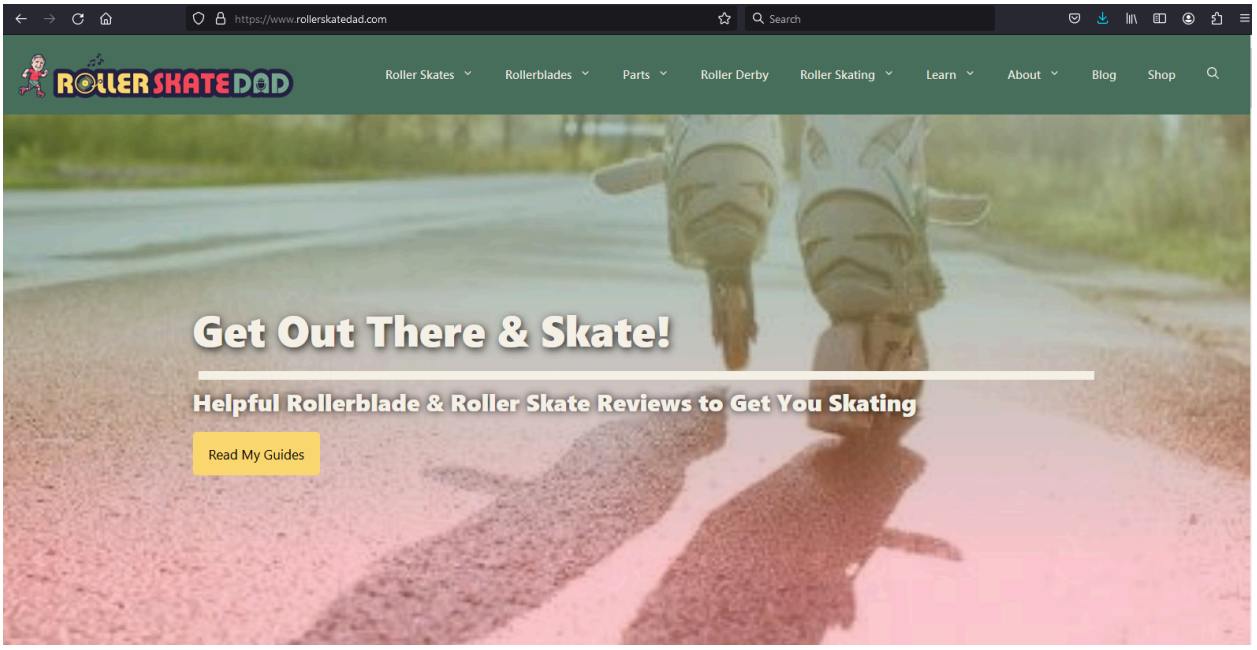
Cons:

- Doesn't allow you to create your own extensive review with sections dedicated for different aspects of roller skates.
- Rating system is underutilized.
- Users can't leave comments.

### 1.2.5 Rollerskatedad.com

Roller Skate Dad is a website, podcast and movement run by Jeff Stone. rollerskatedad.com provides blogs about roller skating, lists of top recommended products which is useful for beginners and people interested in current trends, but does not have search functionality to find a specific product through a search, has no filters and does not allow you to write your own reviews. You can leave comments under posts.

Rollerskatedad.com is an important example of websites with informative reviews. But unlike some other websites of this kind, it doesn't provide systemic rating to aspects of products. Which would allow recommending similar products or filter products buy some aspect.




Picture 1.6

Cons:

- Doesn't allow you to create your own extensive review with sections dedicated for different aspects of roller skates.

- Doesn't allow you to search for particular product. Search gives whole posts which contains input data.

- Doesn't have search filters.

**Conclusions and perspectives:**

While there are many e-commerce websites that use rating and recommendation systems, there aren't many that provide detailed reviews, let alone user created. Many lack important filters and some don't allow writing comments. All of this are points that can be improved.

Table 1.2

Comparison in functionality between websites

| Functionality | Rollerskatedad.com | Amazon.com | Locoskates.com | Rollerderby.com | Rollerskatenation.com | RollerView.com (my website) |
|---|---|---|---|---|---|---|
| Allows users to create full and detailed reviews | - | - | - | - | - | + |
| Search for certain product | - | + | + | + | + | + |
| Has list of top rated products | + | + | - | - | - | + |
| Search filters include wheel size, wheel hardness. | - | - | + | + | - | + |
| Allows users to write comments on product page | + | + | - | + | - | + |
| Provides recommendations of similar products | - | + | - | + | - | + |

# 2 REVIEW OF CHOSEN TECHNOLOGIES AND SOFTWARE DEVELOPMENT TOOLS FOR CREATING A WEBSITE

Since the initial goal was to write a recommendation and rating system for a website, this narrows the selection of necessary applications, technologies and frameworks.

Programming language should meet some criteria: must be high level object oriented language; must be one of the older languages with developed ecosystem around it. Older languages, if they are still developed and supported, have good selection of libraries and vast amount of debugging discussions on the internet. Which is really important for smoother and faster coding process and access to well documented and tested libraries and APIs; It must be used in a well known backend framework that is still relevant on the current market, free and open-source.

During practice I was introduced to C# programming language. It has all aspects I required and more[2]. It is - one of the most popular programming languages in the world. Being supported developed and support for 24 years since 2000; general-purpose; type-safe; object-oriented; while having high performance, it is easy to work with; has well functioning garbage collector which reclaims memory for objects that are no longer referenced. This transfers responsibility of explicit deallocation of the memory for an object from programmers to compiler, erasing the problem of incorrect pointers seen in languages such as C++; a managed language. Which means before conversion into machine code, it is compiled into managed code which is called Intermediate Language; has functionality for asynchronous programming, and multithreading.

Picture 2.1 C# logo

C# uses .NET as its free open source developer platform for building broad range of apps. And is essential runtime service – meaning it is converting IL code into machine code and is shipped with Base Class Library, which provides core functionality to programmers. At the moment latest version is .NET 8. .NET platform has it's designe strengths in productivity, performance, security, and reliability.


Picture 2.2 .NET logo

As a backend framework for creating websites, ASP.NET Core framework is chosen because it's the main way to write backend client-server functionality with C# language. It not only is suitable for creating web sites but also REST-based web APIs, and microservice. It can also work with popular single page application frameworks such as: React, Angular or Vue. It is cross platform. Client can run it on any platform that supports browser[2]. It is free and open-source. Has membership system for authentication and authorization called Identity API. Which has developed and tested functionality for logging in and managing users.


Picture 2.3 ASP.NET Core logo

Since C#, .NET 8 and ASP.NET Core are all officially supported by Microsoft [4]. The best IDE for developing website using C# will be Visual Studio. At the moment latest version is Visual Studio 2022.

Picture 2.4 Visual Studio 2022 logo

Visual Studio is a powerful developer tool that is suitable for completing entire development cycle in one place. It is an extensive integrated development environment (IDE) that can be used for writing, editing, debugging, and building code, and then deploying app. Visual Studio also includes compilers, code completion tools, source control, extensions, and many more features to enhance every stage of the software development process[5]. Strengths of the program are[5]: workload-based installer means that you only download parts, frameworks and tool that you will need for development, without taking excessive space on your hard drive; has effective coding tools and features; has multiple language support allows developers to use it for writing in many different languages such as C++, C#, JavaScript, TypeScript, Python, and more; cross-platform development – it can be used to build programs for Windows, Mac, Linux, iOS, and Android; has Git version control integration tools.

For simpler and faster creation and interaction with database OR/M(object-relational mapper) should be used. Entity Framework Core is the best candidate for this job. It is an open-source and cross-platform version of Entity Framework data access technology.[6]


Picture 2.5 Entity Framework Core logo

Entity Framework Core is lightweight and extensible. It enables .NET developers to work with a database using .NET objects. Avoids the need to write most of the data-access code that typically needed. EF Core can work with many database engines such as: SqlServer, Sqlite, PostgreSQL, MySql and more.

Nowadays many websites use frontend frameworks for building SPA (single page application). It is a web app implementation that loads only a single web document, and then dynamically rewrites the body content of that single document via JavaScript APIs such as Fetch when different content needs to be shown [7]. It allows to avoid downloading HTML file of each page every time user interacts with page, sends or receives data, or wants to go to different page.

As frontend framework for building user interfaces Vue.js was chosen. It builds on top of standard HTML, CSS, and JavaScript and provides a declarative, component-based programming model that allows developers to create user interfaces of any complexity [8].

Picture 2.6 Vue.js logo

Vue is one of the most popular frontend frameworks, has easier learning curve in comparison to other frameworks like React or Angular, and has excellent documentation.

Node.js is a popular JS runtime that provides server runtime for Vue. Allowing developers simple solution for setting up server environment using only JS. Node is cross-platform, free, open-source. It is build on Chrome's V8 JavaScript engine. [11]

Picture 2.7 Node.js logo

Popular choice for ASP.NET Core and Entity Framework is MS SQL (or SQL Server). It is a relational database management system (RDBMS). Instead of SQL language it uses Microsoft's and Sybase's proprietary extension called T-SQL. It is:

- easy to install,
- high performance,
- secure.


Picture 2.8 SQL Server logo

# 3 SOFTWARE DEVELOPMENT

## 3.1 Designing use case diagram

Nowadays many products have websites which are dedicated to blog reviews of products in a systemic way to rate different models and brands in same categories. But there aren't such website for roller skates and process of choosing the right one can lead to choice paralysis. My website is named RollerView.

RollerView has 3 main Actors [2]:

1. **User**: a person for whom the system is going to be useful, there are 2 types of users:

1. **Unauthorised** – are users that haven't yet logged into the system. They have access to 4 use cases:

- Search
- View main recommendation page
- View list of top rated reviews
- View review

2. **Authorised** – are users that have logged into the system. They have access to the same use cases as Unauthorised, in addition to 6 new use cases:

- Write comment
- Rate review
- View similar reviews
- Write review
- Suggest new roller skate product
- Publish review

2. **Server** – it provides users with interface through which they request data. Such as:

- Review data.
- Review rating.
- User rating.
- Search results.
- Product data.

3. **Administrator**: is a person managing the website, checking new product suggestion to add to the list of product for which reviews can be made. He has access to 4 use cases:

- Post new product
- Check of suggested new roller skate products
- Manage user accounts
- Analyse statistics

There are 14 use cases:

1. **Search**: allows Users to find products by name or reviews by name and content.
2. **View main recommendation page**: Shows list of the most popular reviews of the week. Can show more personalized list of users chooses tags of roller skate types they are mostly interested in.
3. **View list of top rated reviews**: Allows to see top rated reviews in different categories.
4. **View review**: Shows detailed review with final rating given to the product at the end.
5. **Write comment**: Allows Users to write comments under the review.
6. **Rate review**: Allows Users to rate if review was interesting/good for them.
7. **View similar**: Shows Users similar reviews that they can read.

8. **Write review**: Allows Users to write their own review on product that exists in the list of products (roller skates).

9. **Publish review**: After finishing writing a review Users can publish them on the website.

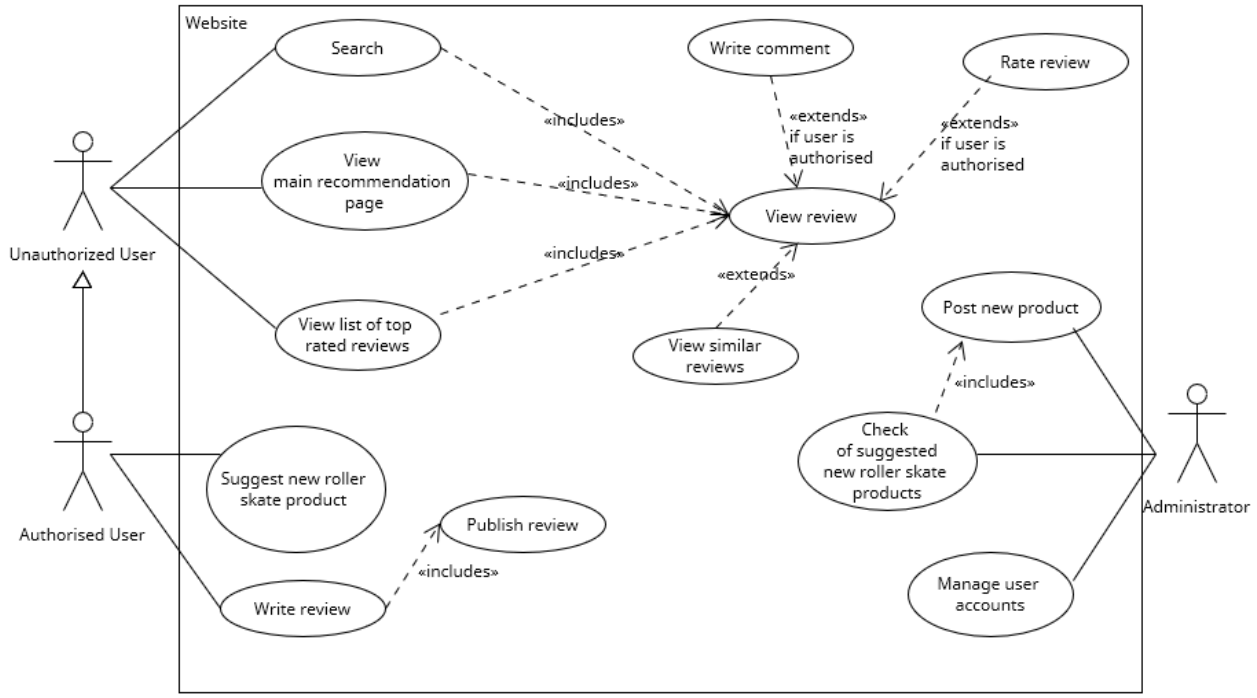
10. **Suggest new roller skate product**: Allows Users to suggest new roller skate product that doesn't exist in the products list.

11. **Check of suggested new roller skate products**: Allows Administrators to check suggestion that users make. After assessing the suggestion they can publish it.

12. **Post new product**: Allows Administrators to add new product to the list of available products for writing reviews.

13. **Manage user accounts**: Allows Administrators to manage user accounts. They are allows to ban Users that violate the rules of the website.

14. **Analyse statistics**: Allows Administrators to analyse how popular are different categories, how many people registered in different time spans and so on.




Picture 3.1 Use case diagram

### 3.2 Technical Assignment

The purpose of Technical assignment is to create a list of requirements that need to be implemented in the program. Therefore, explaining every function that program needs to be able to perform and helping plan functionality in the big picture. It is based on use case diagram which, points out main functionality in few words.

View Roller should use Vue.js for frontend, node.js to run frontend code and be a server to which users connect. It's going to communicate with ASP.NET Core 8 API to perform RESTful CRUD operations through endpoints.

Functional requirements:

- Authorization and authentication.

- Being able to search a review or roller skates using search bar.

- Being able to like and dislike a review.

- Being able to rate skates.

- Being able to write comments under reviews.

- Being able to rate comment as useful.

- Being able to write detailed review and rate skates by "build quality", "comfort of wearing", "comfort of skating".

- Being able to filter reviews by time period.

- Being able to sort reviews by date, rating.

- Show similar reviews on review page.

- Administrators should be able to ban users.

- Administrators must be able to add new products (roller skates) to catalog.

All pages need to have same **header** and **footer**:

1. **Header** – is placed first at the top of all pages. It has this parts placed in order from left to right:

1.1. **Logo**. Placed first. Aligned to the left side of header. Clicking on logo sends to “Review discovery” page.

1.2. **Navigation bar**. Placed second after Logo. Aligned to the left side of header. It consists of 5 page links:

- **Quad skates**. Sends to “Review discovery” page with filters set on best Quad skates reviews of the past month.

- **4 wheel inline skates**. Sends to “Review discovery” page with filters set on best 4 wheel inline skates reviews of the past month.

- **Triskates**. Sends to “Review discovery” page with filters set on best 3 wheel inline skates reviews of the past month.

- **Aggressive skates**. Sends to “Review discovery” page with filters set on best Aggressive skates reviews of the past month.

- **Top ratings**. Sends to the “Ratings” page with filters set to the best of all time rated by users.

1.3. **Search bar for reviews**: After inputting data, user is sent to “Review discovery” page, filtering reviews that contain input data. Aligned to the right side of header.

1.4. **Search bar for products:** After inputting data, user is sent to “Products discovery” page, filtering products that contain input data in their Model Name. Aligned to the right side of header. Is placed under the “Search bar for reviews”.

1.5. If user is not logged in:

- **Sign up** button. Aligned to the right side of header. Sends to “Sign up” page

- **Log in** button. Aligned to the right side of header. Sends to “Log in” page.

1.6. If user is logged in:

- **Profile** button that when clicked shows dropdown with links to :

- “Public profile” page

- “Profile settings” page

2. **Footer** – is placed last at the bottom of all pages. It has:

2.1. List of 2 links aligned and centered to the left.

- **Terms of use**. Sends users to “Terms of use” page.

- **About us**. Sends users to “About Us” page.

2.2. If user is not logged in:

- **Sign up** button. Aligned center to the right side of footer. Sends to “Sign up” page**.**

Website needs to consist of pages:

1. **Review discovery** (Main Page):

1.1. Fist page that user will see when connecting to website.

1.2. Clicking on logo send user to “Review discovery” page.

1.3. It contains best reviews of the past month in a list that is centered in the middle.

1.4. Review list has pagination functionality.

1.5. In the element of the list that describes review Users can see:

- Skates picture.

- Skates name.

- Name of a user that wrote review.

- Date of posting.

- Number of likes.

- Review title.

1.6. List of filters aligned to the left side. Being able to filter by:

- Brand
- Wheel size
- Wheel hardness
- Roller skates type
- Price range
- Color
- Foot sizes
- Has built in brake
- Bearing rating
- For kids
- Type of boot
- By time period
- By boot material
- Plate type

2. **Ratings**:

2.1. Default shows best roller skates of all time by arithmetic mean of reviewers' ratings in their reviews.

2.2. Can be filtered by

- Brand
- Wheel size
- Wheel hardness
- Roller skate type
- Price range
- Color
- Foot sizes
- Has built in brake
- Bearing rating
- For kids
- Type of boot

- By time period
- By boot material
- Plate type

2.3. Can be sorted by:

- Reviewer rating of skates.
- User rating of skates.
- By rated aspect “build quality”, “comfort of wearing” or “comfort of skating”.

3. **Log in**:

3.1. Has username, e-mail and password forms.

3.2. Allows users to log in to the system. When they are logged in they are able to write reviews, rate reviews, write comments and suggest new products.

4. **Sign up**:

4.1. Has username, e-mail and password forms.

4.2. Checks if password is contains an uppercase character, lowercase character, a digit, and a non-alphanumeric character. Passwords must be at least six characters long.

4.3. After filling forms correctly. New user is created in the database.

5. **Review**:

5.1. Each review has technical description of skates, 3 main sections and summery. With rating given at the end of each:

- Build quality
- Comfort of wearing
- Comfort of riding

5.2. At the end review is summed up with ratings and their arithmetic mean.

5.3. Under review is comments section

6. **Suggest product**:

6.1. Has a list of forms for technical description

6.2. Forms required to be filled:

- Model Name
- Brand
- Type
- Wheel size
- Wheel hardness
- Color
- Range of sizes
- Type of boot
- Boot Material

6.3. Optional forms:

- Price range
- Has build in brake
- Bearing rating
- Is for kids
- Plate type

7. **Products discovery**:

7.1. Shows products in grid form. 8 rows, 3 products in each row.

7.2. Has implemented pagination.

7.3. Can be filtered by

- Brand
- Wheel size
- Wheel hardness
- Roller skate type
- Price range
- Color
- Foot sizes
- Has built in brake
- Bearing rating
- For kids
- Type of boot

- By time period
- By boot material
- Plate type

7.4. After clicking on one of the products it sends User to “Product description” page.

8. **Product description**:

8.1. Has technical description of skates and their photo.

8.2. Under technical description there is a list of reviews about this product. Sorted from newest to oldest. Can be also sorted by review rating. Clicking on review sends user to “Review” page.

8.3. List of reviews has pagination functionality.

8.4. In the element of the list that describes review Users can see:

- Name of a user that wrote review.
- Date of posting.
- Number of likes.
- Review title.

9. **Profile settings**:

9.1. User can change:

- Email.
- Password.

**10. Public profile:**

10.1. Has user’s list of reviews. In the element of the list that describes user’s review can see:

- Skates picture.
- Skates name.
- Name of a user that wrote review.
- Date of posting.
- Number of likes.
- Review title.

10.2. Has user’s rating.

10.3. Other users may see what types of skates you prefer. User can decide not to show it.

11. **Review creation**:

11.1. Has 3 sections to fill:

- Build quality
- Comfort of wearing
- Comfort of riding

11.2. For each section there is 5 star rating dedicated for it. It must be rated.

11.3. A button to add up to 5 images.

11.4. Arithmetic mean of ratings is shown.

Nonfunctional requirements:

1. **Usability**: 9 out of 10 users should leave feedback that website is easy to use.

2. **Reliability**: The system should be able to handle and recover from errors without data loss or incorrect data processing. It includes cases of incorrectly inputing data into input forms while:

2.1. Writing review.

2.2. Suggesting new product.

2.3. Logging in.

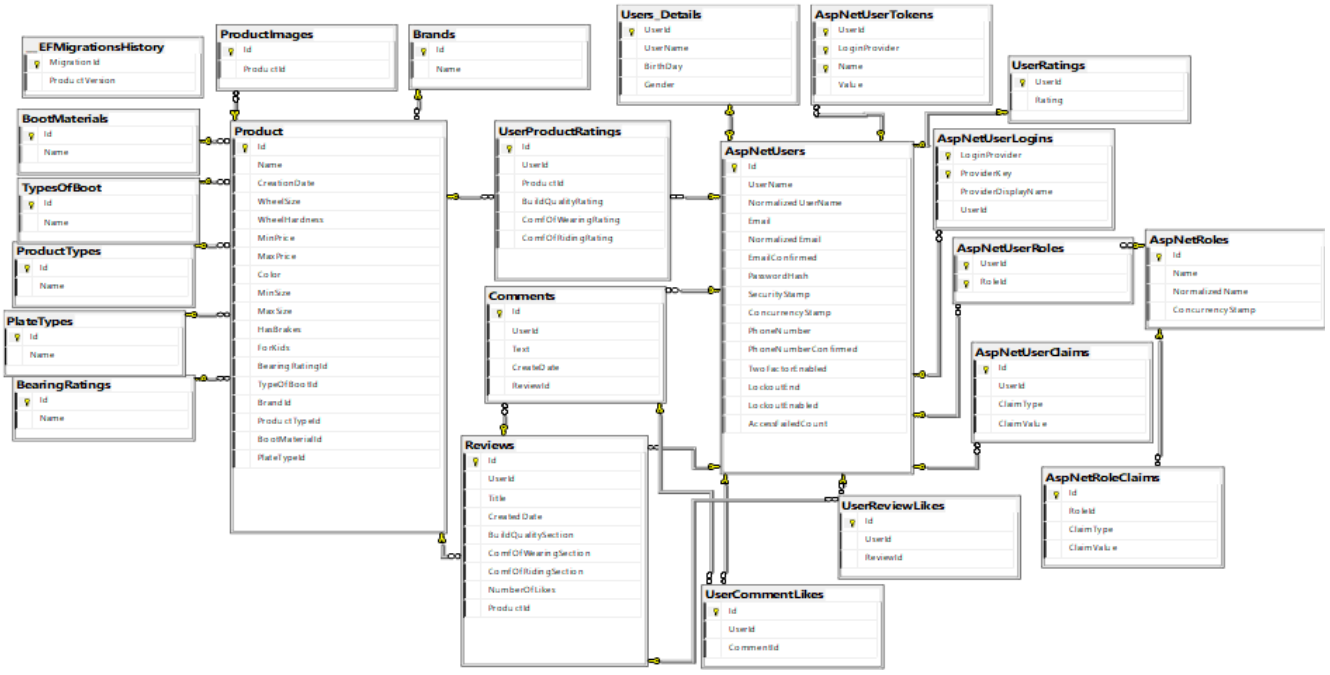
2.4. Sign in.

**3. Compatibility:** Website should work correctly with this list of browsers:

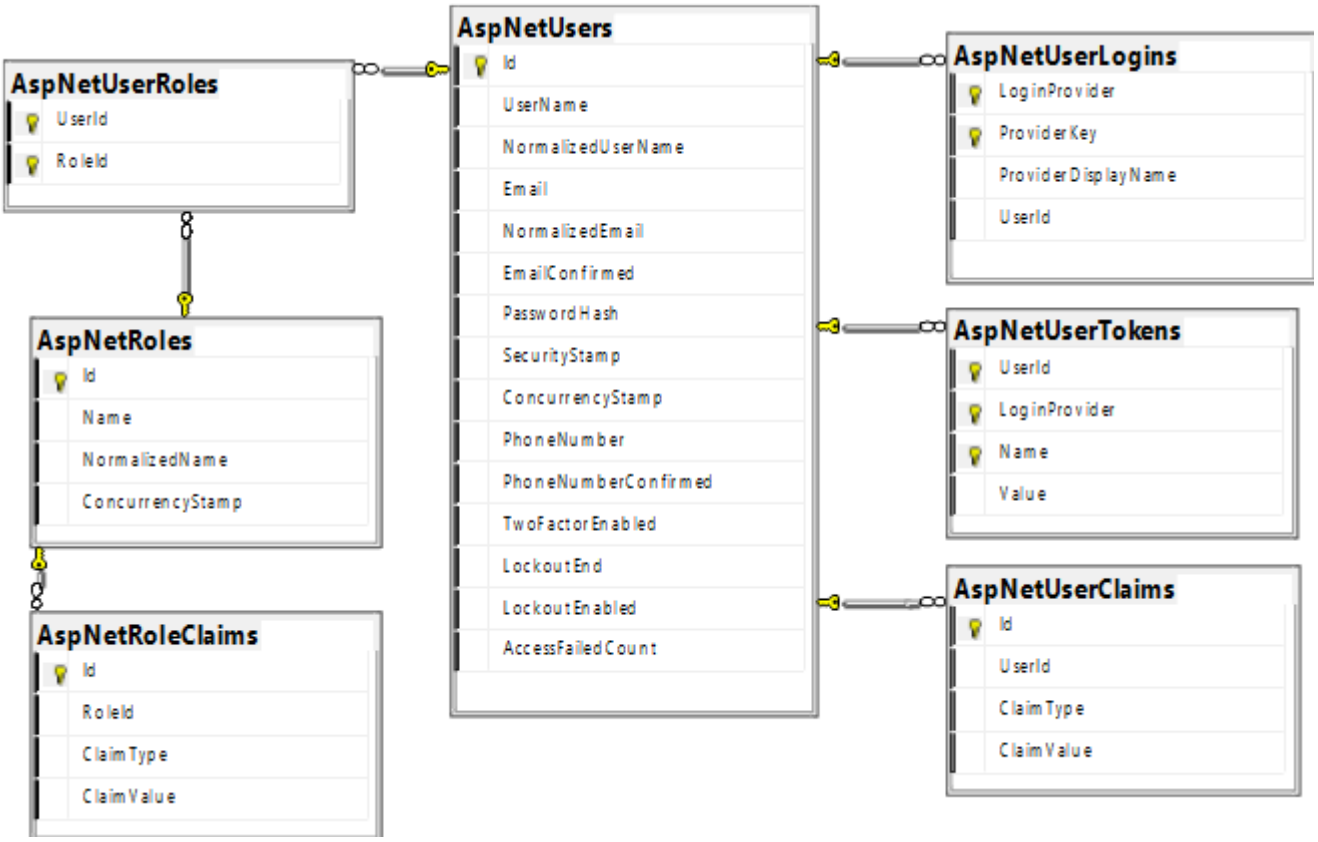
- Mozilla Firefox (v. 126.0)
- Google Chrome (v. 125)
- Opera (v. 110)
- Opera GX (v. 107)
- Safari (v. 17.4.1)

### 3.3 Designing database diagram

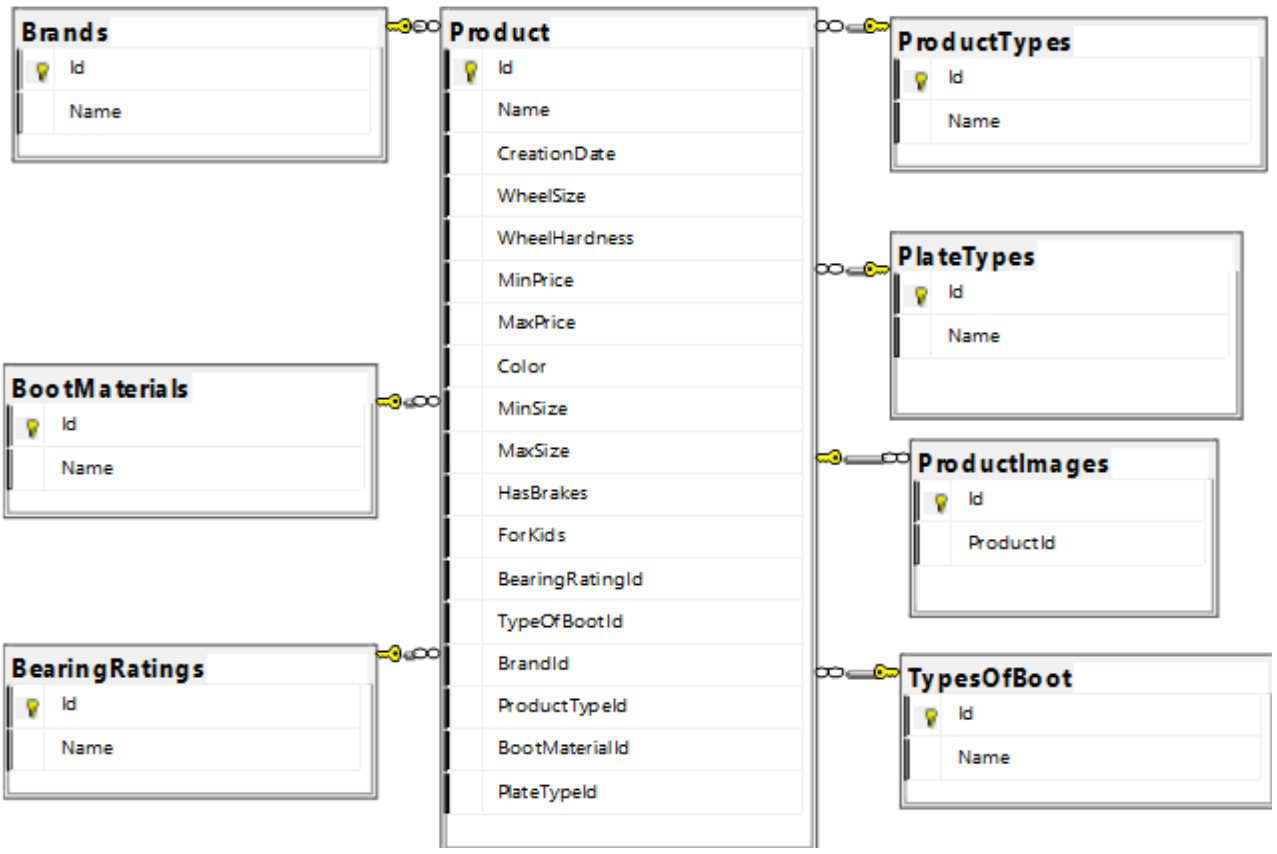
Database diagram is a scheme that visualize the structure of the database and its relationships between tables. They describe the entities and their relationships in the database, and show how the data is stored and processed in the system. Based on the technical assignment the following database diagram was made, containing all the entities required to interact with the application (Picture 3.2).



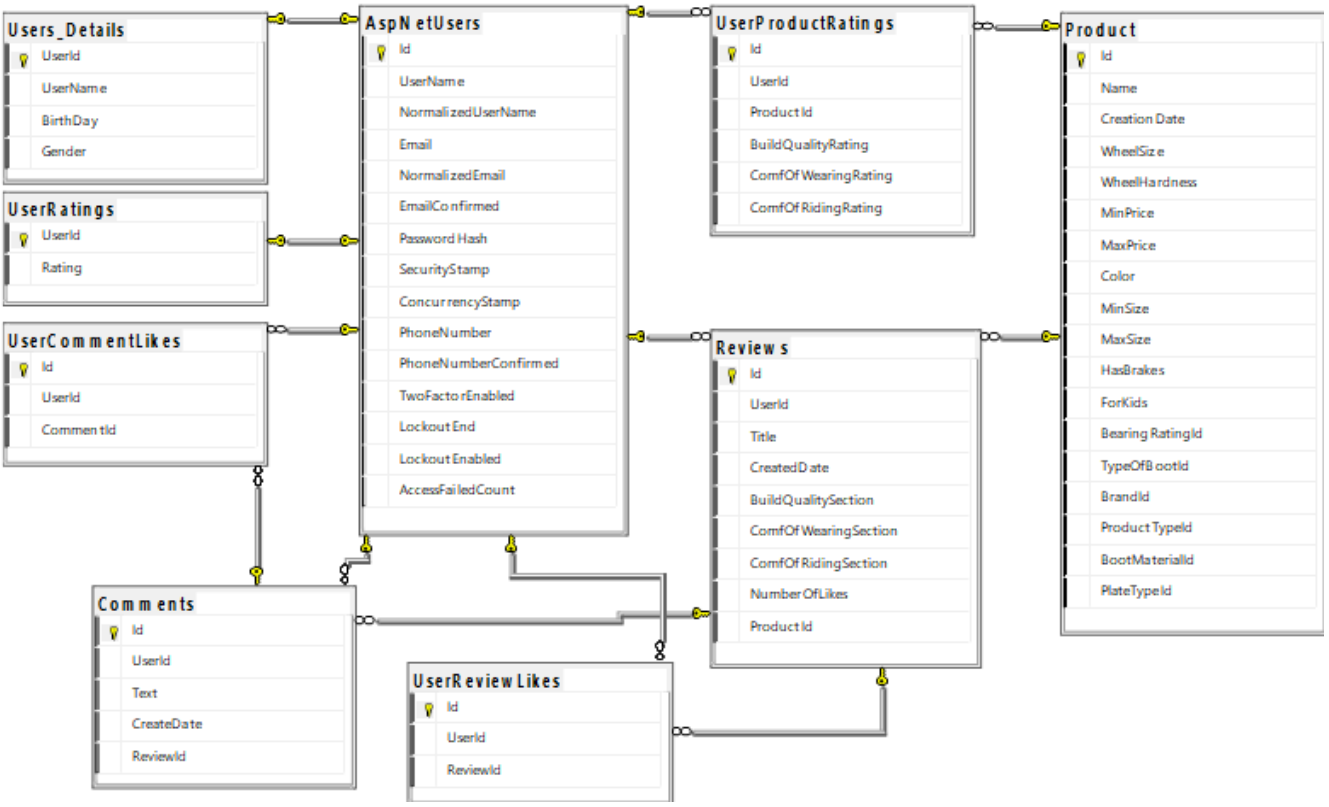

Picture 3.2 Database diagram




Picture 3.3 Database diagram of user

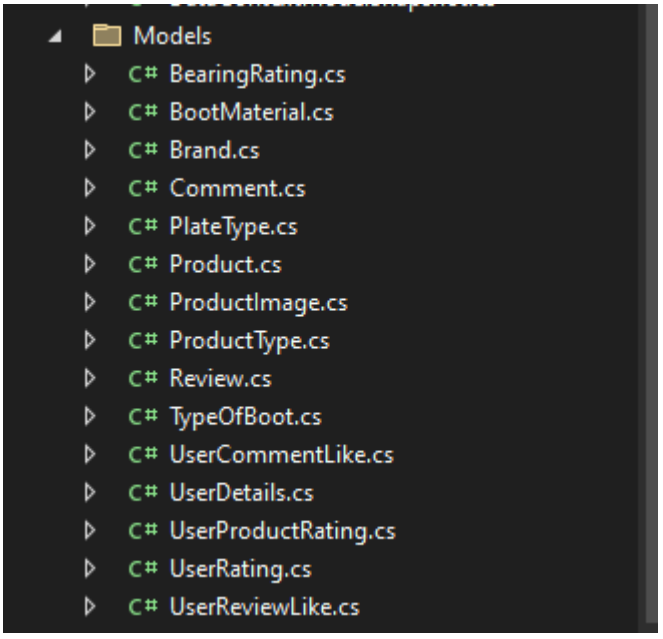



Picture 3.4 Database diagram of product




Picture 3.5 Database diagram of user and product

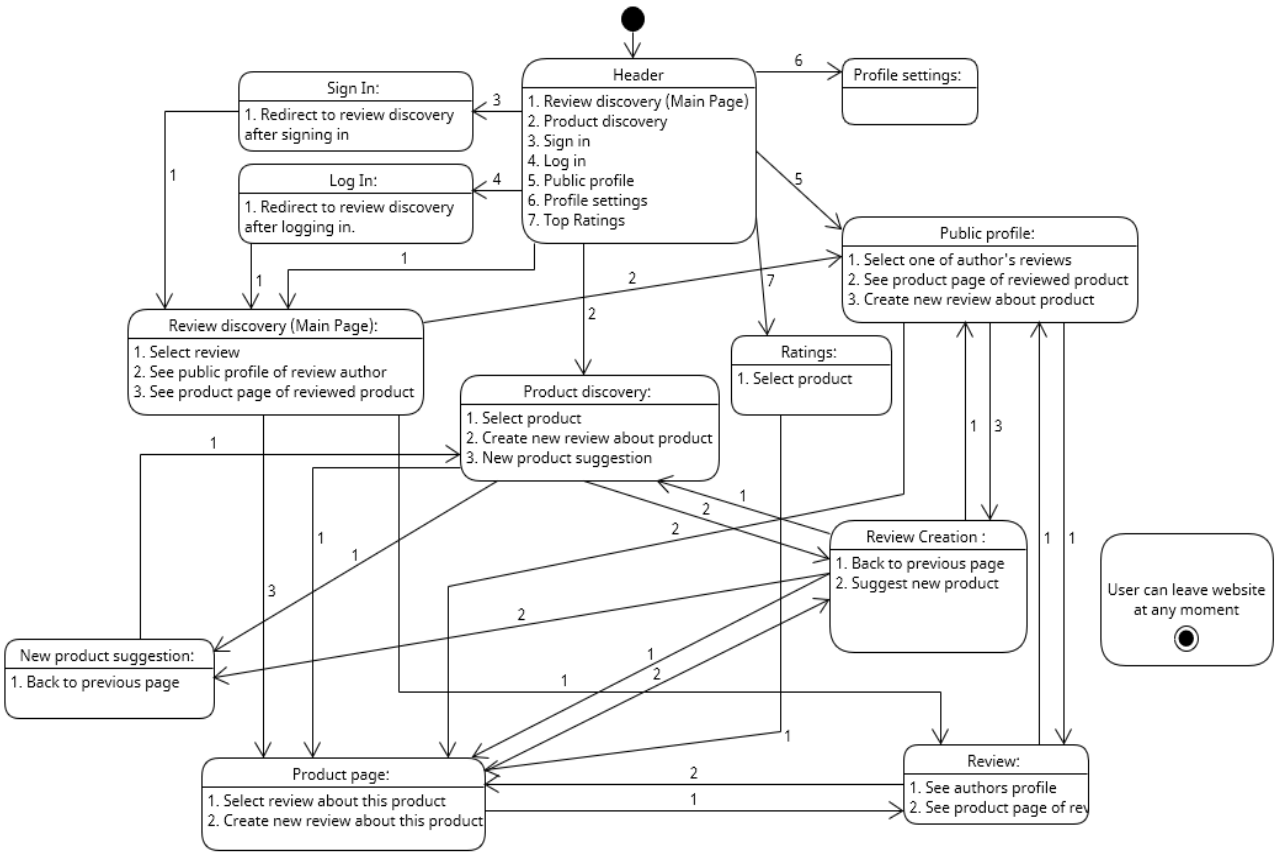
Not considering tables that were added with Identity API, this is list of all models based on which tables were created.




Picture 3.6 Data models folder

### 3.4 Designing state diagram

State diagram is a one of UML diagrams that is used model the dynamic nature of a system Such diagram shows how object during its lifetime changes it's state by responding to internal or external events. Essentially, state diagram is important to model the reactivity of the system [12]. While using website, user interacts with a page via clicking links, which leads to change of the page. How pages lead to one another is described in the next diagram:

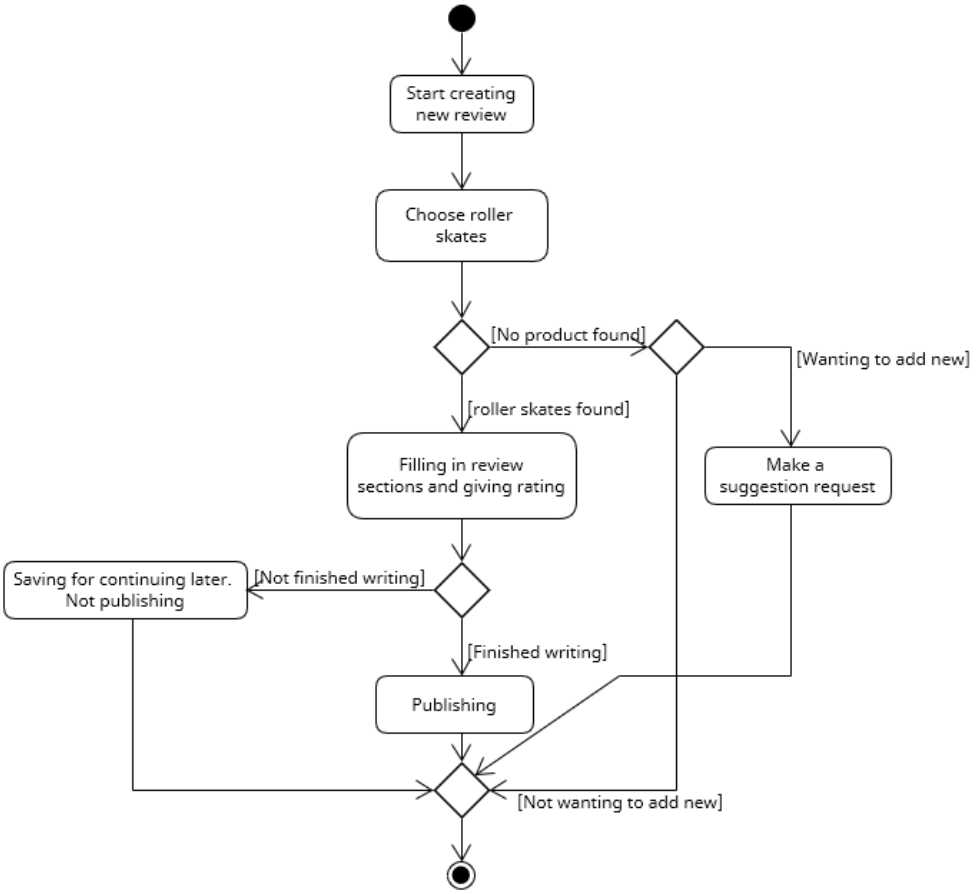



Picture 3.7 State diagram of web pages

### 3.5 Designing activity diagram

Activity diagram is a UML diagram that is used to draw the activity flow of a system, by describing the sequence from one activity to another. In the

diagram parallel, branched and concurrent flow of the system can be shown [14]. Next diagram shows activity flow in creation of new review:

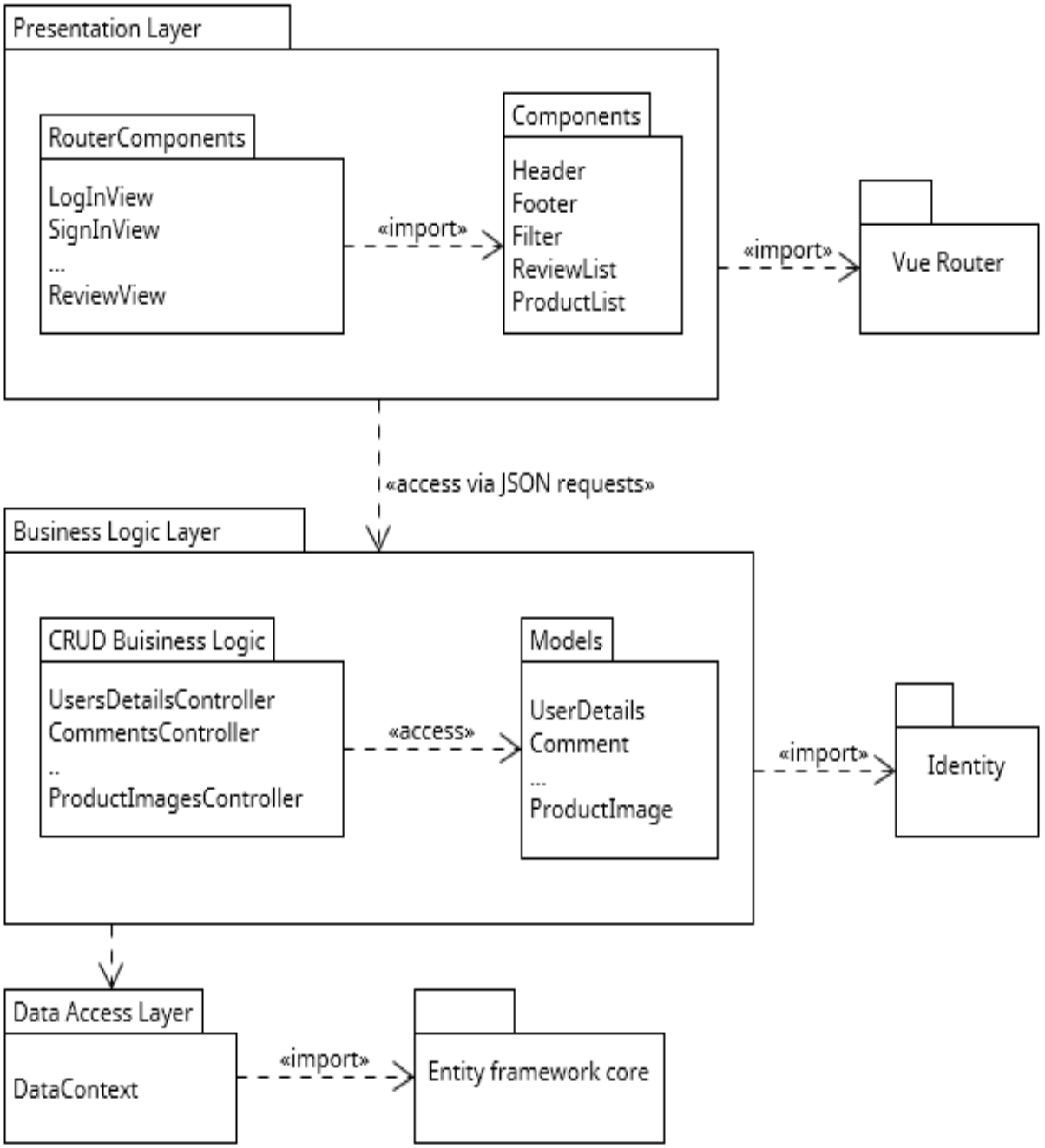



Picture 3.8 Activity diagram of review creation

### 3.6 Designing package diagram

A UML package diagram is a type of structural diagram used in software engineering to represent the organization and dependencies of various packages within a system. Packages in UML are used to group related classes, interfaces, or other packages, facilitating a modular approach to system design. A package diagram visually depicts these packages and the relationships among them, such as dependencies, where one package uses or depends on another. This helps in organizing complex systems into manageable parts, improving readability and

maintainability. By illustrating how different parts of the system are interconnected, package diagrams aid developers and architects in understanding the overall structure and planning the system's evolution.

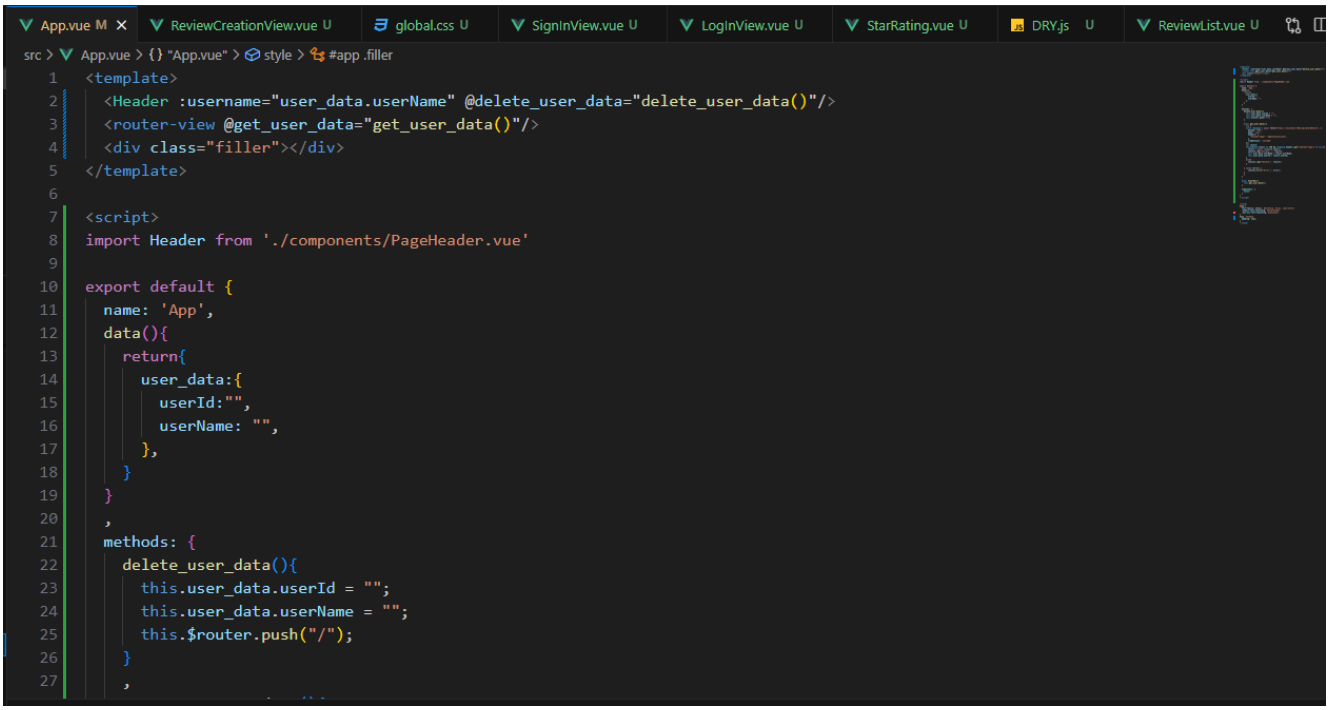



Picture 3.9 Package diagram

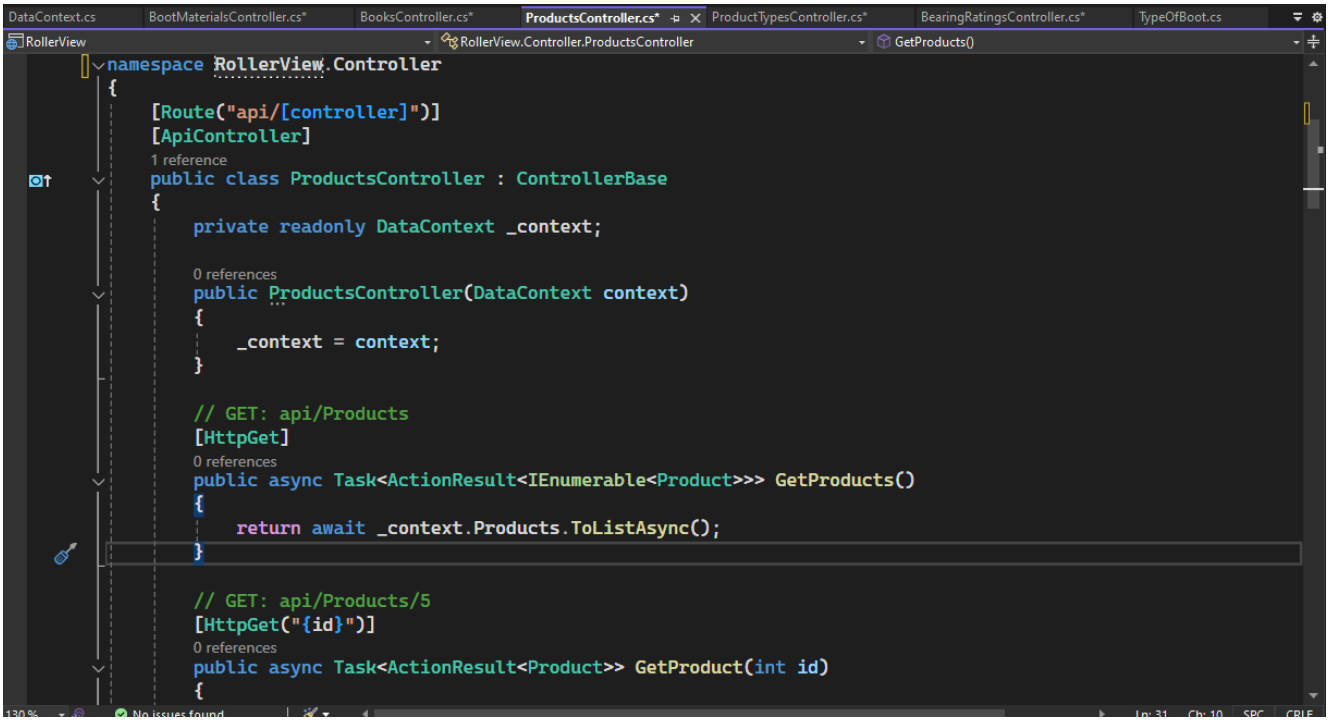
3 layer architecture was chosen for the website. Vue.js and it’s component system is used in Presentation Layer. Components appear in 2 types: router

components are used during routing, simple components are used inside router components.

Presentation Layer is sending JSON requests to Business Layer which uses controller endpoints for each of the models that represent tables in the database.

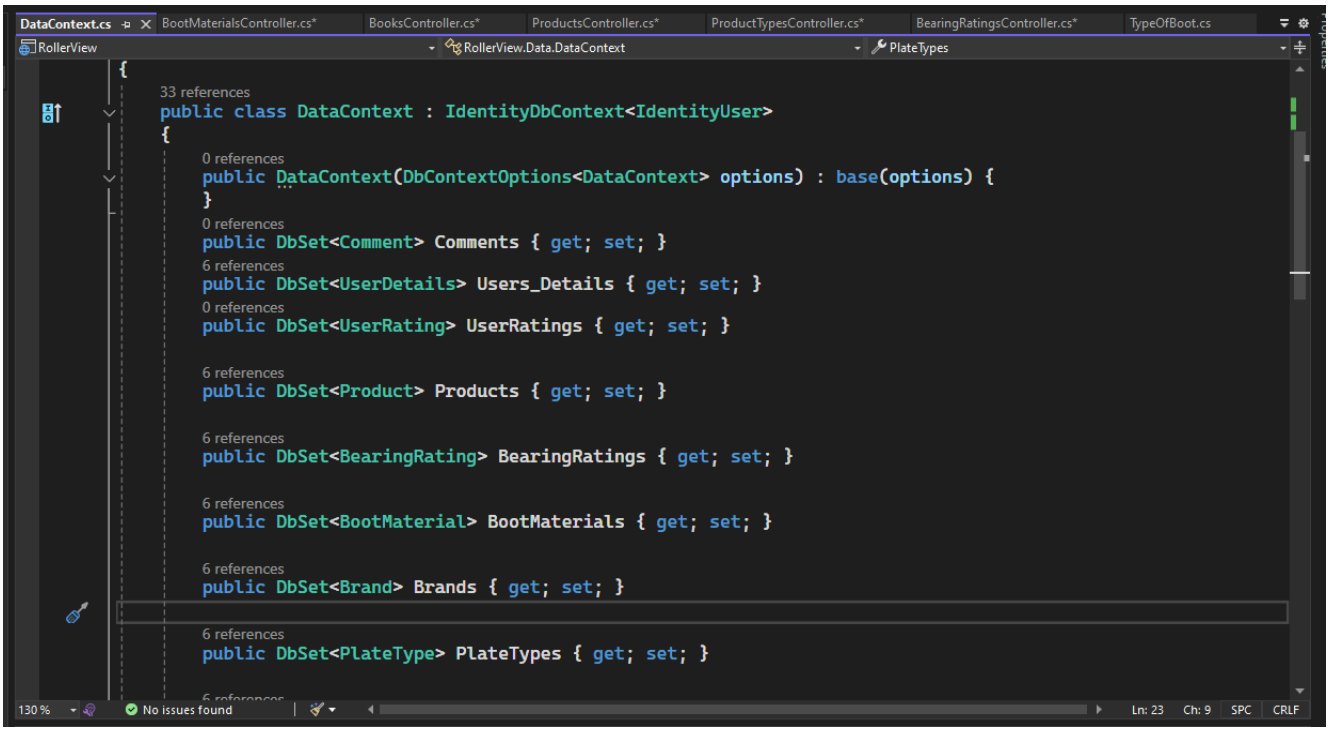



Picture 3.10 App.vue




Picture 3.11 Product controller

Data Access Layer mainly consists of DataContext file that uses Entity framework to interact with SQLServer database.




Picture 3.12 DataContext

### 3.7 Designing rating system

Each user has a rating. A user's rating depends on the number of likes given to their reviews or comments. Each review can be rated positively or negatively. Each product has 3 parameters by which it is evaluated: build quality, comfort of wearing, comfort of riding. The overall product rating is derived from the mean of these ratings. There are 2 parallel ratings: given in reviews and given by users. When a user writes a review and rates a product, his rating is also added as a user rating. For better understanding of products, users can filter and sort products which allows them to find out the best products of a given period or of a certain type.

### 3.8 Designing recommendation system

Website is designed to provide recommendations even to non-authorized users. Therefore, the system should be able to make recommendations based on similar parameters between reviews, their ratings and their relevancy.

To determine which review is recommended, points are awarded for similar aspects between products. Top 5 reviews are shown as recommendations.

Points for how new review is:

- 25 for last week's review.
- 15 for last month's review.
- 10 for review over the last six months.

Points for how similar roller skates in reviews are:

- 20 for a similar type of roller skate.
- 20 for a review from the same author.
- 15 if the product is also for children.
- 10 if the product has the same brand.
- 10 if the product has the same wheel size category.
- Small wheel category: less than 100mm
- Large wheel category: equal to or greater than 100mm
- 8 if the product has similar wheel hardness (+- 2).
- 8 if the product has the same color.
- 8 if the product has +-75 $ the same average price.
- 4 if the product also has a brakes.

If the number of points is similar between the reviewed products, the review with the higher rating is selected. If the review rating is similar, the user review with the higher rating is selected. If they are the same, then the review is chosen randomly between the two.

# 4. TESTING

Manual testing was performed using to verify that the system should be able to handle and recover from errors without data loss or incorrect data processing. For testing ASP.NET Core functionality Swagger was used.

Table 4.1

Testing results

| | Testing |
|---|---|
| Models CRUD API | + |
| Log in | + |
| Sign in | + |
| Create review | + |
| Give rating to roller skates | + |
| Suggest new product | + |
| Search product | + |
| Search review | + |

# CONCLUSIONS

1. The methods and benefits of using rating and recommendation systems have been identified. Types of recommendation systems have been researched and described. The websites rollerskatedad.com, amazon.com, locoskates.com, rollerderby.com, rollerskatenation.com were reviewed and their shortcomings were identified.

2. To develop the website, the following were chosen: C# language, .NET 8 development platform, ASP.NET backend framework, Vue.js frontend framework and SQL Server database. Functional and nonfunctional requirements have been defined.

3. Based on the database diagram, activity diagram, state diagram and package diagram, a website dedicated to roller skating with a rating and recommendation system has been developed.

4. During manual testing key shortcomings and errors of the application have been identified and corrected.

# REFERENCES


1. Khosrow-Pour M. Advanced Methodologies and Technologies in Business Operations and Management. IGI Global, 2018.
2. IBM Documentation. *IBM in Deutschland, Österreich und der Schweiz*. URL: https://www.ibm.com/docs/en/product-master/12.0.0?topic=processes-defining-use-cases
3. Ko, H., Lee, S., Park, Y., & Choi, A. (2022). A Survey of Recommendation Systems: Recommendation Models, Techniques, and Application Fields. *Electronics*, *11*(1), 141. https://doi.org/10.3390/electronics11010141
4. Albahari, J. (2023). *C# 12 in a Nutshell: The Definitive Reference*. O'Reilly Media, Incorporated.
5. *Introduction to .NET - .NET*. (n.d.). Microsoft Learn: Build skills that open doors in your career. https://learn.microsoft.com/en-us/dotnet/core/introduction
6. *.NET Official Support Policy*. (n.d.). Microsoft. https://dotnet.microsoft.com/en-us/platform/support/policy
7. *What is the Visual Studio IDE?* (n.d.). Microsoft Learn: Build skills that open doors in your career. https://learn.microsoft.com/en-us/visualstudio/get-started/visual-studio-ide?view=vs-2022
8. *Overview of Entity Framework Core - EF Core*. (n.d.). Microsoft Learn: Build skills that open doors in your career. https://learn.microsoft.com/en-us/ef/core/
9. *SPA (Single-page application) - MDN Web Docs Glossary: Definitions of Web-related terms | MDN*. (n.d.). MDN Web Docs. https://developer.mozilla.org/en-US/docs/Glossary/SPA

10. *Vue.js*. (n.d.). Vue.js - The Progressive JavaScript Framework | Vue.js. https://vuejs.org/guide/introduction.html

11. *Node.js — Run JavaScript Everywhere*. (n.d.). Node.js — Run JavaScript Everywhere. https://nodejs.org/en

12. *UML - Statechart Diagrams*. (n.d.). Online Tutorials, Courses, and eBooks Library | Tutorialspoint. https://www.tutorialspoint.com/uml/uml_statechart_diagram.htm

13. *UML - Activity Diagrams*. (n.d.). Online Tutorials, Courses, and eBooks Library | Tutorialspoint. https://www.tutorialspoint.com/uml/uml_activity_diagram.htm

# ANNEX A. DEMONSTRATION MATERIALS


ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ


## Software development in C# for content rating and recommendations for a website about roller skating

Виконав студент 5 курсу
групи ППЗ-51
Теймур Аббасов Фархад огли
Керівник роботи
Старший викладач кафедри ІПЗ Гаманюк Ігор Михайлович

Київ – 2024

## PURPOSE, OBJECT AND SUBJECT OF RESEARCH

- **The purpose of the work:** support the processes of content rating and recommendations for a website about roller skating.
- **Object of research:** content rating and recommendations for a website about roller skating.
- **Subject of research:** software development for content rating and recommendations for website about roller skating.

# TASKS OF THE GRADUATE THESIS

1. Research the subject area related to ratings and recommendation systems used on websites dedicated to roller skating.
2. Analyze and select most appropriate technologies and tools for website development. Define functional and nonfunctional requirements.
3. Design and develop a website dedicated to roller skating, including rating and recommendation systems.
4. Conduct manual testing of the application to identify shortcomings, errors or behavior not intended by the application.

3

# SOFTWARE REQUIREMENTS

**Functional requirements**:

1. Authorization and authentication.
2. Being able to search a review or roller skates using search bar.
3. Being able to **like and dislike** a review.
4. Being able to rate skates.
5. Being able to write comments under reviews.
6. Being able to rate comment as useful.
7. Being able to write detailed review and rate roller skates by “build quality”, “comfort of wearing”, “comfort of skating”.
8. Being able to filter reviews by time period.
9. Being able to sort reviews by date, rating.
10. Show similar reviews on review page.
11. Administrators should be able to ban users.
12. Administrators must be able to add new products (roller skates) to catalog.

4

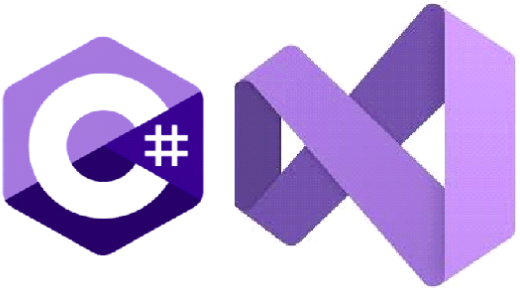
# SOFTWARE REQUIREMENTS

**Nonfunctional requirements**:

1. **Usability**: 9 out of 10 users should leave feedback that website is easy to use.
2. **Reliability**: The system should be able to handle and recover from errors without data loss or incorrect data processing. It includes cases of incorrectly inputing data into input forms while:
   2.1. Writing review.
   2.2. Suggesting new product.
   2.3. Logging in.
   2.4. Sign in.
3. **Compatibility:** Website should work correctly with this list of browsers:
   - Mozilla Firefox (v. 126.0)
   - Google Chrome (v. 125)
   - Opera (v. 110)
   - Opera GX (v. 107)
   - Safari (v. 17.4.1)

5

# DEVELOPMENT SOFTWARE



Visual Studio 2022

Vue.js





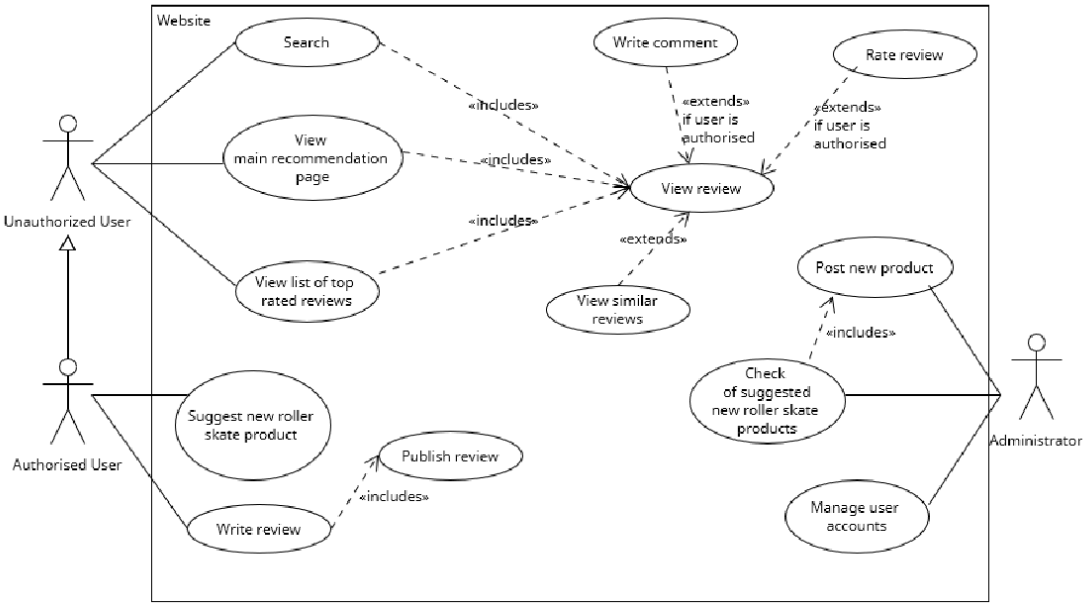
6

# ANALYSIS OF SIMILAR PROGRAMS

| Functionality | rollerskatedad .com | amazon .com | locoskates .com | rollerderby. com | rollerskaten ation.com | RollerView. com(my website) |
|---|---|---|---|---|---|---|
| Allows users to create full and detailed reviews | - | - | - | - | - | + |
| Search for certain product | - | + | + | + | + | + |
| Has list of top rated products | + | + | - | - | - | + |
| Search filters include wheel size, wheel hardness. | - | - | + | + | - | + |
| Allows users to write comments on product page | + | + | - | + | - | + |
| Provides recommendati ons of similar products | - | + | - | + | - | + |

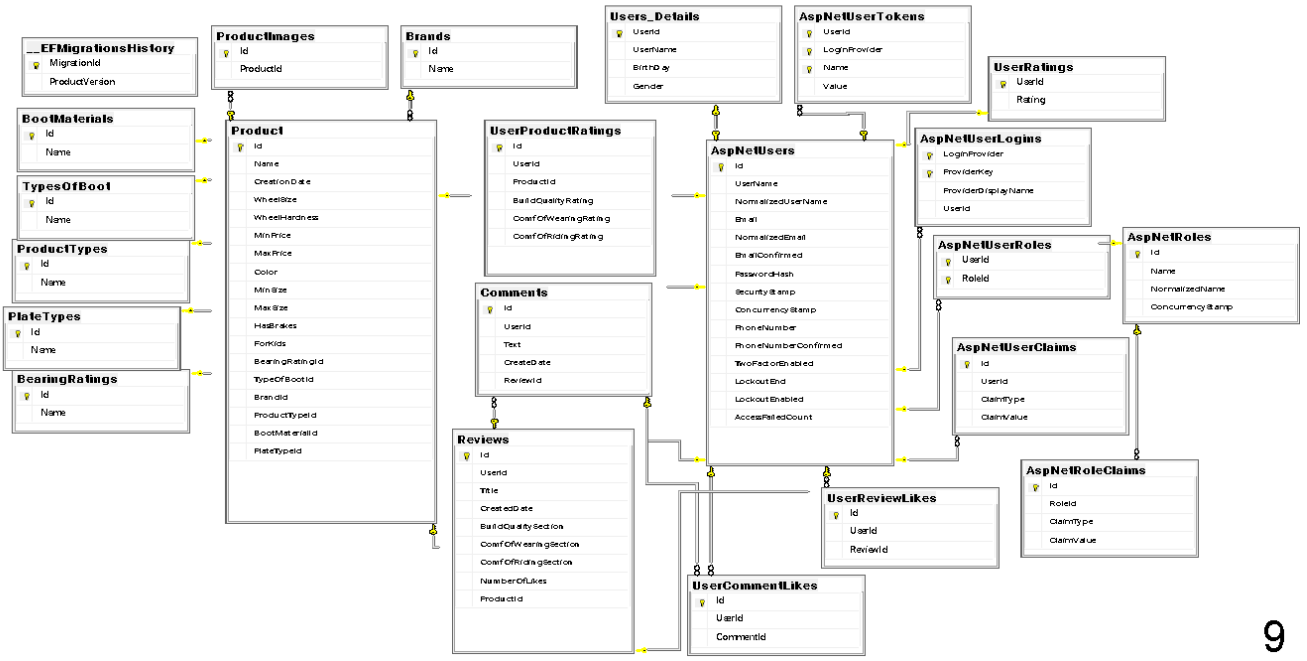
7

# USE CASE DIAGRAM



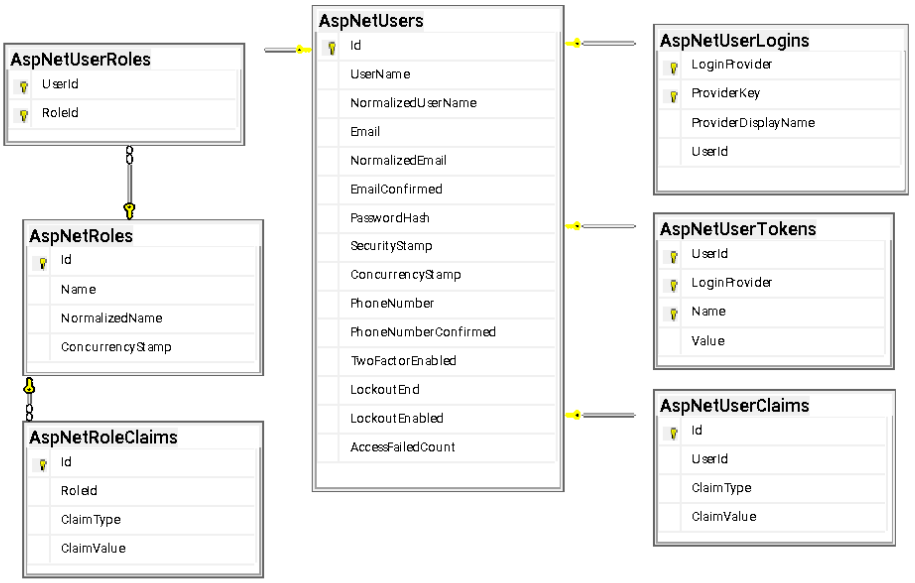

8

# DATABASE DIAGRAM



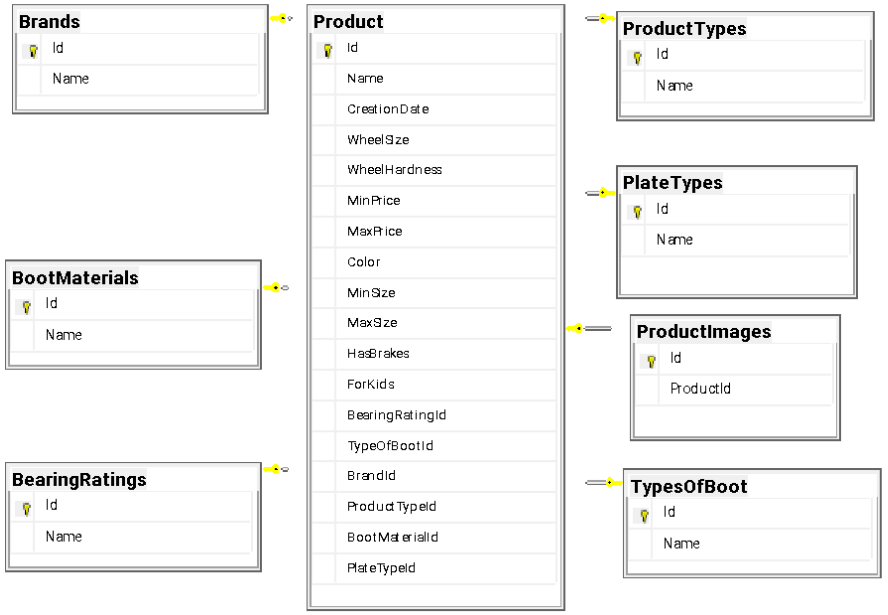

9

# DATABASE DIAGRAM OF USER



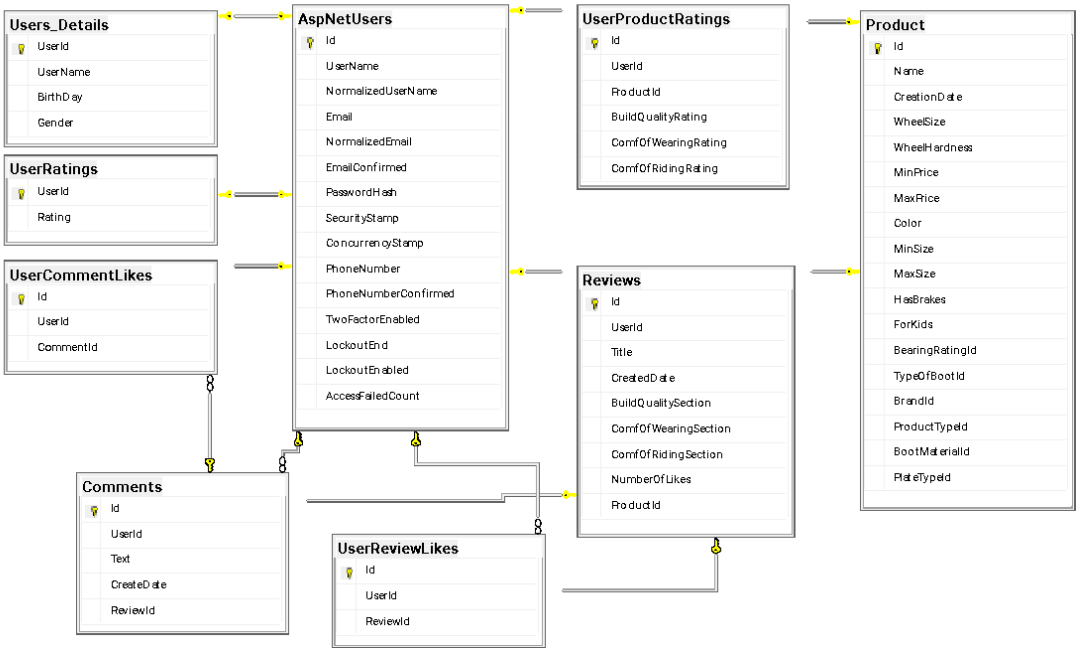

10

# DATABASE DIAGRAM OF PRODUCT



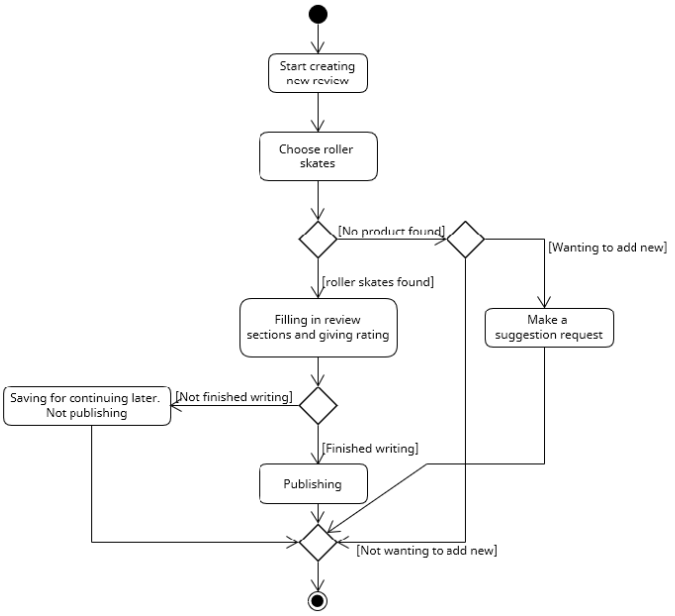

11

# DATABASE DIAGRAM OF USER AND PRODUCT



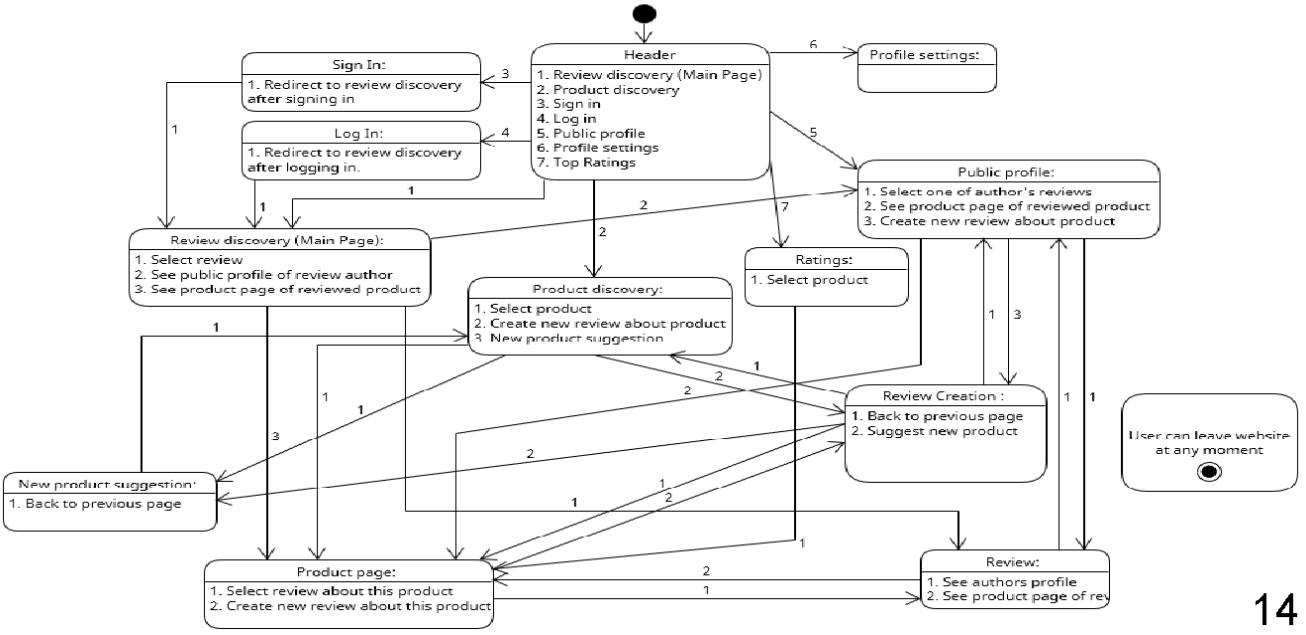

12

# ACTIVITY DIAGRAM OF REVIEW CREATION



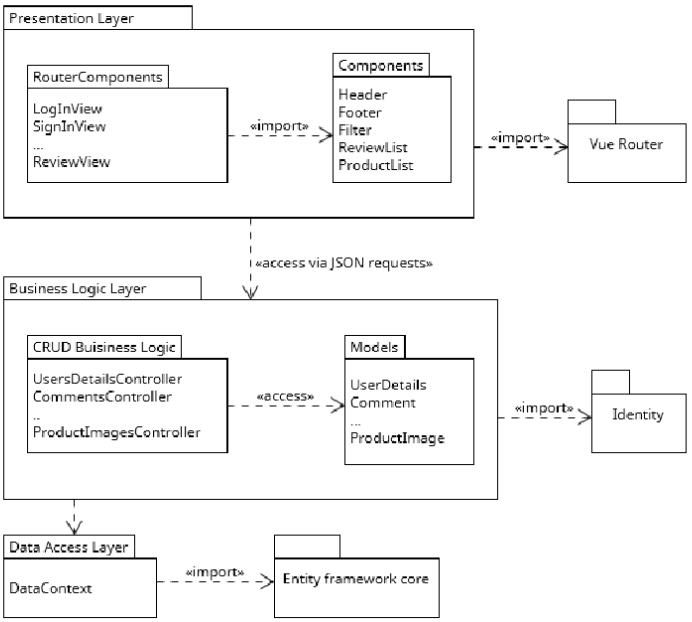

13

# STATE DIAGRAM OF WEB PAGES



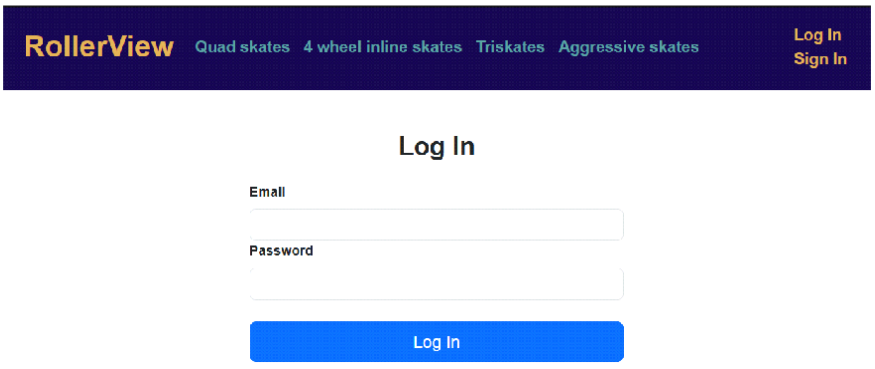

14

# PACKAGE DIAGRAM




15

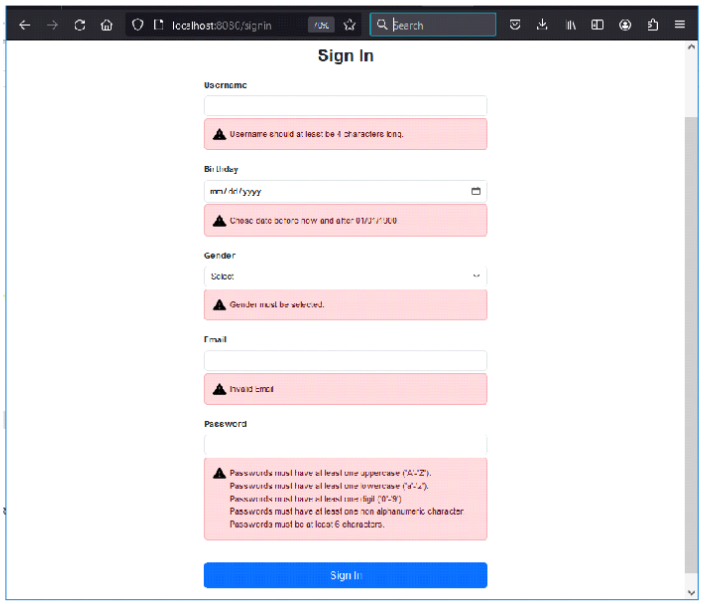
# SCREEN FORMS




Log in page

16

# SCREEN FORMS




Sign in page

17

# APPROVAL OF RESEARCH RESULTS

1) Аббасов Т.Ф., Гаманюк І.М. Software development of recommendation and rating systems for reviewing roller skates using uml use case diagram. Застосування програмного забезпечення в інформаційно-комунікаційних технологіях: Всеукраїнська науково-технічна конференція. Збірник тез. 24.04.2024, ДУІКТ, м. Київ К.: ДУІКТ, 2024. С. 209.

2) Аббасов Т.Ф., Гаманюк І.М. Analysis of applications that use recommendation and rating systems for reviewing roller skates. Застосування програмного забезпечення в інформаційно-комунікаційних технологіях: Всеукраїнська науково-технічна конференція. Збірник тез. 24.04.2024, ДУІКТ, м. Київ, К.: ДУІКТ, 2024. С. 212.

18

# CONCLUSIONS

1. The methods and benefits of using rating and recommendation systems have been identified. Types of recommendation systems have been researched and described. The websites rollerskatedad.com, amazon.com, locoskates.com, rollerderby.com, rollerskatenation.com were reviewed and their shortcomings were identified.
2. To develop the website, the following were chosen: C# language, .NET 8 development platform, ASP.NET backend framework, Vue.js frontend framework and SQL Server database. Functional and nonfunctional requirements have been defined.
3. Based on the database diagram, activity diagram, state diagram and package diagram, a website dedicated to roller skating with a rating and recommendation system has been developed.
4. During manual testing key shortcomings and errors of the application have been identified and corrected.

# ANNEX B. LISTING OF SOFTWARE MODULES

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using RollerView.Data;
using RollerView.Models;

namespace RollerView.Controller
{
   [Route("api/[controller]")]
   [ApiController]
   public class BearingRatingsController :
ControllerBase
   {
      private readonly DataContext _context;

      public
BearingRatingsController(DataContext context)
      {
         _context = context;
      }

      // GET: api/BearingRatings
      [HttpGet]
      public async
Task<ActionResult<IEnumerable<BearingRating
>>> GetBearingRatings()
      {
         List < BearingRating >  l = await
_context.BearingRatings.ToListAsync();
         l = l.OrderBy(item => item.Name).ToList();
         return l;
      }

      // GET: api/BearingRatings/5
      [HttpGet("{id}")]
      public async
Task<ActionResult<BearingRating>>
GetBearingRating(int id)
      {
         var bearingRating = await
_context.BearingRatings.FindAsync(id);

         if (bearingRating == null)
         {
            return NotFound();
         }

         return bearingRating;
      }

      [HttpGet("get_example")]
      public ActionResult<BearingRating>
GetExample()
      {
         BearingRating b = new BearingRating
         {
            Name = ""
         };

         return b;
      }

      // PUT: api/BearingRatings/5
      // To protect from overposting attacks, see
https://go.microsoft.com/fwlink/?linkid=2123754
      [HttpPut("{id}")]
      public async Task<IActionResult>
PutBearingRating(int id, BearingRating
bearingRating)
      {
         if (id != bearingRating.Id)
         {
            return BadRequest();
         }

         _context.Entry(bearingRating).State =
EntityState.Modified;

         try
         {
            await _context.SaveChangesAsync();
         }
         catch (DbUpdateConcurrencyException)
         {
            if (!BearingRatingExists(id))
            {
               return NotFound();
            }
            else
            {
               throw;
            }
         }

         return NoContent();
      }

      // POST: api/BearingRatings
      // To protect from overposting attacks, see
https://go.microsoft.com/fwlink/?linkid=2123754
      [HttpPost]
      public async
Task<ActionResult<BearingRating>>
PostBearingRating(BearingRating bearingRating)
      {

_context.BearingRatings.Add(bearingRating);
         await _context.SaveChangesAsync();

         return
CreatedAtAction("GetBearingRating", new { id =
bearingRating.Id }, bearingRating);
      }

      // DELETE: api/BearingRatings/5
      [HttpDelete("{id}")]
```

```
        public async Task<IActionResult>
DeleteBearingRating(int id)
        {
            var bearingRating = await
_context.BearingRatings.FindAsync(id);
            if (bearingRating == null)
            {
                return NotFound();
            }


_context.BearingRatings.Remove(bearingRating);
            await _context.SaveChangesAsync();

            return NoContent();
        }

        private bool BearingRatingExists(int id)
        {
            return _context.BearingRatings.Any(e =>
e.Id == id);
        }
    }
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using RollerView.Data;
using RollerView.Models;

namespace RollerView.Controller
{
    [Route("api/[controller]")]
    [ApiController]
    public class BootMaterialsController :
ControllerBase
    {
        private readonly DataContext _context;

        public BootMaterialsController(DataContext
context)
        {
            _context = context;
        }

        // GET: api/BootMaterials
        [HttpGet]
        public async
Task<ActionResult<IEnumerable<BootMaterial>
>> GetBootMaterials()
        {
            List<BootMaterial> l = await
_context.BootMaterials.ToListAsync();
            l = l.OrderBy(item => item.Name).ToList();
            return l;
        }

        // GET: api/BootMaterials/5
        [HttpGet("{id}")]
        public async
Task<ActionResult<BootMaterial>>
GetBootMaterial(int id)
        {
            var bootMaterial = await
_context.BootMaterials.FindAsync(id);

            if (bootMaterial == null)
            {
                return NotFound();
            }

            return bootMaterial;
        }

        [HttpGet("get_example")]
        public ActionResult<BootMaterial>
GetExample()
        {
            BootMaterial b = new BootMaterial
            {
                Name = ""
            };

            return b;
        }

        // PUT: api/BootMaterials/5
        [HttpPut("{id}")]
        public async Task<IActionResult>
PutBootMaterial(int id, BootMaterial
bootMaterial)
        {
            if (id != bootMaterial.Id)
            {
                return BadRequest();
            }

            _context.Entry(bootMaterial).State =
EntityState.Modified;

            try
            {
                await _context.SaveChangesAsync();
            }
            catch (DbUpdateConcurrencyException)
            {
                if (!BootMaterialExists(id))
                {
                    return NotFound();
                }
                else
                {
                    throw;
                }
            }

            return NoContent();
        }

        // POST: api/BootMaterials
```

```
        // To protect from overposting attacks, see
https://go.microsoft.com/fwlink/?linkid=2123754
        [HttpPost]
        public async
Task<ActionResult<BootMaterial>>
PostBootMaterial(BootMaterial bootMaterial)
        {

_context.BootMaterials.Add(bootMaterial);
            await _context.SaveChangesAsync();

            return
CreatedAtAction("GetBootMaterial", new { id =
bootMaterial.Id }, bootMaterial);
        }

        // DELETE: api/BootMaterials/5
        [HttpDelete("{id}")]
        public async Task<IActionResult>
DeleteBootMaterial(int id)
        {
            var bootMaterial = await
_context.BootMaterials.FindAsync(id);
            if (bootMaterial == null)
            {
                return NotFound();
            }


_context.BootMaterials.Remove(bootMaterial);
            await _context.SaveChangesAsync();

            return NoContent();
        }

        private bool BootMaterialExists(int id)
        {
            return _context.BootMaterials.Any(e =>
e.Id == id);
        }
    }
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using RollerView.Data;
using RollerView.Models;

namespace RollerView.Controller
{
    [Route("api/[controller]")]
    [ApiController]
    public class BrandsController : ControllerBase
    {
        private readonly DataContext _context;

        public BrandsController(DataContext
context)
        {
            _context = context;
        }

        // GET: api/Brands
        [HttpGet]
        public async
Task<ActionResult<IEnumerable<Brand>>>
GetBrands()
        {
            List<Brand> l = await
_context.Brands.ToListAsync();
            l = l.OrderBy(item => item.Name).ToList();
            return l;
        }

        // GET: api/Brands/5
        [HttpGet("{id}")]
        public async Task<ActionResult<Brand>>
GetBrand(int id)
        {
            var brand = await
_context.Brands.FindAsync(id);

            if (brand == null)
            {
                return NotFound();
            }

            return brand;
        }
        [HttpGet("get_example")]
        public  ActionResult<Brand> GetExample()
        {
            Brand b = new Brand
            {
                Name = ""
            };

            return b;
        }

        // PUT: api/Brands/5
        // To protect from overposting attacks, see
https://go.microsoft.com/fwlink/?linkid=2123754
        [HttpPut("{id}")]
        public async Task<IActionResult>
PutBrand(int id, Brand brand)
        {
            if (id != brand.Id)
            {
                return BadRequest();
            }

            _context.Entry(brand).State =
EntityState.Modified;

            try
            {
                await _context.SaveChangesAsync();
            }
            catch (DbUpdateConcurrencyException)
```

```
            {
                if (!BrandExists(id))
                {
                    return NotFound();
                }
                else
                {
                    throw;
                }
            }

            return NoContent();
        }

        // POST: api/Brands
        // To protect from overposting attacks, see
https://go.microsoft.com/fwlink/?linkid=2123754
        [HttpPost]
        public async Task<ActionResult<Brand>>
PostBrand(Brand brand)
        {
            Brand b = new Brand
            {
                Name = brand.Name
            };
            _context.Brands.Add(b);
            await _context.SaveChangesAsync();

            return CreatedAtAction("GetBrand", new
{ id = brand.Id }, brand);
        }

        // DELETE: api/Brands/5
        [HttpDelete("{id}")]
        public async Task<IActionResult>
DeleteBrand(int id)
        {
            var brand = await
_context.Brands.FindAsync(id);
            if (brand == null)
            {
                return NotFound();
            }

            _context.Brands.Remove(brand);
            await _context.SaveChangesAsync();

            return NoContent();
        }

        private bool BrandExists(int id)
        {
            return _context.Brands.Any(e => e.Id ==
id);
        }
    }
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using RollerView.Data;
using RollerView.Models;

namespace RollerView.Controller
{
    [Route("api/[controller]")]
    [ApiController]
    public class PlateTypesController :
ControllerBase
    {
        private readonly DataContext _context;

        public PlateTypesController(DataContext
context)
        {
            _context = context;
        }

        // GET: api/PlateTypes
        [HttpGet]
        public async
Task<ActionResult<IEnumerable<PlateType>>>
GetPlateTypes()
        {
            List<PlateType> l = await
_context.PlateTypes.ToListAsync();
            l = l.OrderBy(item => item.Name).ToList();
            return l;
        }

        // GET: api/PlateTypes/5
        [HttpGet("{id}")]
        public async
Task<ActionResult<PlateType>>
GetPlateType(int id)
        {
            var plateType = await
_context.PlateTypes.FindAsync(id);

            if (plateType == null)
            {
                return NotFound();
            }

            return plateType;
        }

        [HttpGet("get_example")]
        public ActionResult<PlateType>
GetExample()
        {
            PlateType b = new PlateType
            {
                Name = ""
            };

            return b;
        }

        // PUT: api/PlateTypes/5
```

```
        // To protect from overposting attacks, see
https://go.microsoft.com/fwlink/?linkid=2123754
        [HttpPut("{id}")]
        public async Task<IActionResult>
PutPlateType(int id, PlateType plateType)
        {
            if (id != plateType.Id)
            {
                return BadRequest();
            }

            _context.Entry(plateType).State =
EntityState.Modified;

            try
            {
                await _context.SaveChangesAsync();
            }
            catch (DbUpdateConcurrencyException)
            {
                if (!PlateTypeExists(id))
                {
                    return NotFound();
                }
                else
                {
                    throw;
                }
            }

            return NoContent();
        }

        // POST: api/PlateTypes
        // To protect from overposting attacks, see
https://go.microsoft.com/fwlink/?linkid=2123754
        [HttpPost]
        public async
Task<ActionResult<PlateType>>
PostPlateType(PlateType plateType)
        {
            _context.PlateTypes.Add(plateType);
            await _context.SaveChangesAsync();

            return CreatedAtAction("GetPlateType",
new { id = plateType.Id }, plateType);
        }

        // DELETE: api/PlateTypes/5
        [HttpDelete("{id}")]
        public async Task<IActionResult>
DeletePlateType(int id)
        {
            var plateType = await
_context.PlateTypes.FindAsync(id);
            if (plateType == null)
            {
                return NotFound();
            }

            _context.PlateTypes.Remove(plateType);
            await _context.SaveChangesAsync();

            return NoContent();
        }

        private bool PlateTypeExists(int id)
        {
            return _context.PlateTypes.Any(e => e.Id
== id);
        }
    }
}
using System;
using System.Collections;
using System.Collections.Generic;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Linq;
using System.Linq.Expressions;
using System.Threading.Tasks;
using FuzzyString;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using RollerView.Data;
using RollerView.Models;
using RollerView.RecivedModels;
using RollerView.RecivedModels.SubClasses;
using RollerView.SentModels;

namespace RollerView.Controller
{
    [Route("api/[controller]")]
    [ApiController]
    public class ProductsController :
ControllerBase
    {
        private readonly DataContext _context;

        public ProductsController(DataContext
context)
        {
            _context = context;
        }

        // GET: api/Products
        [HttpGet]
        public async
Task<ActionResult<IEnumerable<Product>>>
GetProducts()
        {
            return await _context.Products.Include(p
=> p.BearingRating)
                .Include(p => p.TypeOfBoot)
                .Include(p => p.Brand)
                .Include(p => p.ProductType)
                .Include(p => p.BootMaterial)
                .Include(p => p.PlateType)
                .ToListAsync();
        }

        // GET: api/Products/5
```

```
    [HttpGet("{id}")]
    public async Task<ActionResult<Product>>
GetProduct(int id)
    {
      var product = await
_context.Products.FindAsync(id);

      if (product == null)
      {
        return NotFound();
      }

      return product;
    }
    [HttpGet("get_example")]
    public ActionResult<Product> GetExample()
    {
      Product p = new Product
      {
        Name = "",
        CreationDate = new DateTime(),
        WheelSize = 0,
        WheelHardness = 0,
        MinPrice = 0,
        MaxPrice = 0,
        Color = "",
        MinSize = 0,
        MaxSize = 0,
        HasBrakes = false,
        ForKids = false
      };

      return p;
    }

    [HttpGet("get_colors")]
    public async
Task<ActionResult<IEnumerable<string>>>
GetColors()
    {
      List<string> colors = await
_context.Products.Select(p =>
p.Color).Distinct().ToListAsync();
      colors.Sort();
      return colors;
    }
    [HttpGet("get_wheel_sizes")]
    public async
Task<ActionResult<IEnumerable<int>>>
GetWheelSizes()
    {
      List<int> wheel_sizes = await
_context.Products.Select(p =>
p.WheelSize).Distinct().ToListAsync();
      wheel_sizes.Sort();
      return wheel_sizes;
    }
    [HttpGet("get_wheel_hardness")]
    public async
Task<ActionResult<IEnumerable<int>>>
GetWheelHardness()
    {
      List<int> wheel_hardness = await
_context.Products.Select(p =>
p.WheelHardness).Distinct().ToListAsync();
      wheel_hardness.Sort();
      return wheel_hardness;
    }

    [HttpGet("get_size_range")]
    public async Task<ActionResult<List<int>>>
GetSizeRange()
    {

      List<int> p_size = new List<int>
      {
        await _context.Products.MinAsync(p =>
p.MinSize),
        await _context.Products.MaxAsync(p =>
p.MaxSize)
      };
      return p_size;
    }

    [HttpGet("get_price_range")]
    public async Task<ActionResult<List<int>>>
GetPriceRange()
    {
      List<int> p_price = new List<int>
      {
        await _context.Products.MinAsync(p =>
p.MinSize),
        await _context.Products.MaxAsync(p =>
p.MaxSize)
      };
      return p_price;
    }


    [HttpGet("search_product")]
    public async
Task<ActionResult<IEnumerable<S_ProductNam
e>>> SearchProducts(string filter)
    {
      if (string.IsNullOrWhiteSpace(filter))
      {
        return BadRequest("Filter cannot be
empty");
      }

      IQueryable<Product> unsorted_request =
_context.Products.Include(p =>
p.Brand).Where(p => p.Name.Contains(filter) ||
p.Brand.Name.Contains(filter));
      List<Product> result = new
List<Product>();
      if (unsorted_request.Count() != 0) {
        result = await
unsorted_request.ToListAsync();
      }
      result = result.OrderByDescending(p =>
p.CreationDate).ToList();
      result = result.OrderByDescending(p =>
p.Name.JaroWinklerDistance(filter) >
```

```
p.Brand.Name.JaroWinklerDistance(filter) ?
p.Name.JaroWinklerDistance(filter) :
p.Brand.Name.JaroWinklerDistance(filter)).ToLis
t();

        List<S_ProductName> list = new
List<S_ProductName>();
        foreach (Product prod in result) {
            list.Add(new S_ProductName{ id=
prod.Id , name = prod.Name});
        }
        return list;
    }

    // PUT: api/Products/5
    // To protect from overposting attacks, see
https://go.microsoft.com/fwlink/?linkid=2123754
    [HttpPut("{id}")]
    public async Task<IActionResult>
PutProduct(int id, Product product)
    {
        if (id != product.Id)
        {
            return BadRequest();
        }

        _context.Entry(product).State =
EntityState.Modified;

        try
        {
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateConcurrencyException)
        {
            if (!ProductExists(id))
            {
                return NotFound();
            }
            else
            {
                throw;
            }
        }

        return NoContent();
    }

    [HttpPost("filtered")]
    public async
Task<ActionResult<IEnumerable<Product>>>
GetFilteredProducts(R_ProductRequestOptions
options)
    {
        int position = (options.page-1) * 10;
        List<Product> products = await
_context.Products.Include(p => p.BearingRating)
            .Include(p => p.TypeOfBoot)
            .Include(p => p.Brand)
            .Include(p => p.ProductType)
            .Include(p => p.BootMaterial)
            .Include(p => p.PlateType)
            .ToListAsync();

        products = products.Where(p =>
CheckDate(p, options.time_scope)).Where(p =>
CheckProduct(p, options.filter)).ToList();


        products = Order(products,
options.sort_command);
        products =
products.Skip(position).ToList();

        return products;
    }

    // POST: api/Products
    // To protect from overposting attacks, see
https://go.microsoft.com/fwlink/?linkid=2123754
    [HttpPost, Authorize]
    public async Task<ActionResult<Product>>
PostProduct(R_Product product)
    {
        ProductType? typ;
        TypeOfBoot? typ_b;
        BearingRating? ber_rate;
        var ba = await
_context.Brands.FirstOrDefaultAsync(b =>
b.Name == product.brand);
        var pl_t = await
_context.PlateTypes.FirstOrDefaultAsync(p =>
p.Name == product.plate_type);
        var b_mat = await
_context.BootMaterials.FirstOrDefaultAsync(bm
=> bm.Name == product.boot_material);

        if (product.bearing_rating != "") {
            ber_rate = await
_context.BearingRatings.FirstOrDefaultAsync(b
=> b.Name == product.bearing_rating);
            if (ber_rate == null)
            {
                return BadRequest();
            }
        }
        else
        {
            return BadRequest();
        }


        if (product.type.Name != "") {
            typ = await
_context.ProductTypes.FindAsync(product.type.I
d);
            if (typ == null) {
                return BadRequest();
            }
        }
        else {
            return BadRequest();
        }
```

```
        if (product.type_of_boot.Name != "")
        {
            typ_b = await
_context.TypesOfBoot.FindAsync(product.type_of
_boot.Id);
            if (typ_b == null)
            {
                return BadRequest();
            }
        }
        else
        {
            return BadRequest();
        }


        if (ba == null) {
            ba = new Brand
            {
                Name = product.brand
            };
            _context.Brands.Add(ba);
        }

        if (pl_t == null) {
            pl_t = new PlateType
            {
                Name = product.plate_type
            };
            _context.PlateTypes.Add(pl_t);
        }

        if (b_mat == null)
        {
            b_mat = new BootMaterial
            {
                Name = product.boot_material
            };
            _context.BootMaterials.Add(b_mat);
        }



        Product p = new Product
        {
            Name = product.model_name,
            CreationDate = DateTime.UtcNow,
            WheelSize = product.wheel_size,
            WheelHardness =
product.wheel_hardness,
            MinPrice = product.min_size,
            MaxPrice = product.max_size,
            Color = product.color,
            MinSize = product.min_size,
            MaxSize = product.max_size,
            HasBrakes = false,
            ForKids = false,
            Brand = ba,
            PlateType = pl_t,
            BootMaterial = b_mat,
            BearingRating = ber_rate,
            TypeOfBoot = typ_b,
            ProductType = typ,
        };

        _context.Products.Add(p);
        await _context.SaveChangesAsync();

        return Ok(p);
    }

    // DELETE: api/Products/5
    [HttpDelete("{id}"), Authorize]
    public async Task<IActionResult>
DeleteProduct(int id)
    {
        var product = await
_context.Products.FindAsync(id);
        if (product == null)
        {
            return NotFound();
        }

        _context.Products.Remove(product);
        await _context.SaveChangesAsync();

        return NoContent();
    }

    private List<Product> Order (List<Product>
list, string order) {

        switch (order)
        {
            case "Date, new to old":
                return list.OrderByDescending(p =>
p.CreationDate).ToList();

            case "Date, old to new":
                return list.OrderBy(p =>
p.CreationDate).ToList();

            case "Rating, best to worst":

                return list.OrderByDescending(p =>
CalculateRating(p)).ToList();

            case "Rating, worst to best":
                return list.OrderBy(p =>
CalculateRating(p)).ToList();
        }
        //default
        return list.OrderByDescending(p =>
p.CreationDate).ToList();

    }

    private double CalculateRating(Product p) {
        double result = 0;
        List<UserProductRating> ratings =
_context.UserProductRatings.Where(r =>
r.ProductId == p.Id).ToList();
```

```
            foreach (UserProductRating rating in
ratings) {
                double tmp = (
                    rating.BuildQualityRating +
                    rating.ComfOfWearingRating +
                    rating.ComfOfRidingRating) /3;
                result = tmp;
            }
            return result;
        }

        private bool CheckProduct(Product p,
FilterOptions opt) {

            if (
                opt.brands.Count != 0 ?
opt.brands.Contains(p.Brand.Name) : true &&
                opt.product_types.Count != 0 ?
opt.product_types.Contains(p.ProductType.Name
) : true &&
                opt.boot_materials.Count != 0 ?
opt.boot_materials.Contains(p.BootMaterial.Nam
e) : true &&
                opt.types_of_boot.Count != 0 ?
opt.types_of_boot.Contains(p.TypeOfBoot.Name)
: true &&
                opt.plate_types.Count != 0 ?
opt.plate_types.Contains(p.PlateType.Name) :
true &&
                opt.bearing_ratings.Count != 0 ?
opt.bearing_ratings.Contains(p.BearingRating.Na
me) : true &&
                opt.wheel_sizes.Count != 0 ?
opt.wheel_sizes.Contains(p.WheelSize) : true &&
                opt.wheel_hardness.Count != 0 ?
opt.wheel_hardness.Contains(p.WheelHardness) :
true &&
                (p.MinSize <= opt.size_range[1]) &&
(p.MaxSize >= opt.size_range[0]) &&
                (p.MinPrice <= opt.price_range[1]) &&
(p.MaxPrice >= opt.price_range[0]) &&
                opt.has_brakes == p.HasBrakes &&
                opt.for_kids == p.ForKids
                ) {
                return true;
            }
            return false;
        }

        private bool CheckDate(Product p, string
time_scope) {
            DateTime past_border;

            switch (time_scope) {
                case "1 week":
                    past_border =
DateTime.UtcNow.AddDays(-7);
                    break;
                case "1 month":
                    past_border =
DateTime.UtcNow.AddMonths(-1);
                    break;
                case "6 months":
                    past_border =
DateTime.UtcNow.AddMonths(-6);
                    break;
                case "All time":
                    past_border = new DateTime();
                    break;
                default:
                    past_border = new DateTime();
                    break;
            }

            if (p.CreationDate >= past_border) {
                return true;
            }
            return false;
        }

        private bool ProductExists(int id)
        {
            return _context.Products.Any(e => e.Id ==
id);
        }
    }
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using RollerView.Data;
using RollerView.Models;

namespace RollerView.Controller
{
    [Route("api/[controller]")]
    [ApiController]
    public class ProductTypesController :
ControllerBase
    {
        private readonly DataContext _context;

        public ProductTypesController(DataContext
context)
        {
            _context = context;
        }

        // GET: api/ProductTypes
        [HttpGet]
        public async
Task<ActionResult<IEnumerable<ProductType>
>> GetProductTypes()
        {
            List<ProductType> l = await
_context.ProductTypes.ToListAsync();
            l = l.OrderBy(item => item.Name).ToList();
            return l;
        }
```

```
        // GET: api/ProductTypes/5
        [HttpGet("{id}")]
        public async
Task<ActionResult<ProductType>>
GetProductType(int id)
        {
            var productType = await
_context.ProductTypes.FindAsync(id);

            if (productType == null)
            {
                return NotFound();
            }

            return productType;
        }

        [HttpGet("get_example")]
        public ActionResult<ProductType>
GetExample()
        {
            ProductType b = new ProductType
            {
                Name = ""
            };

            return b;
        }

        // PUT: api/ProductTypes/5
        // To protect from overposting attacks, see
https://go.microsoft.com/fwlink/?linkid=2123754
        [HttpPut("{id}")]
        public async Task<IActionResult>
PutProductType(int id, ProductType
productType)
        {
            if (id != productType.Id)
            {
                return BadRequest();
            }

            _context.Entry(productType).State =
EntityState.Modified;

            try
            {
                await _context.SaveChangesAsync();
            }
            catch (DbUpdateConcurrencyException)
            {
                if (!ProductTypeExists(id))
                {
                    return NotFound();
                }
                else
                {
                    throw;
                }
            }

            return NoContent();
        }

        // POST: api/ProductTypes
        // To protect from overposting attacks, see
https://go.microsoft.com/fwlink/?linkid=2123754
        [HttpPost]
        public async
Task<ActionResult<ProductType>>
PostProductType(ProductType productType)
        {
            _context.ProductTypes.Add(productType);
            await _context.SaveChangesAsync();

            return
CreatedAtAction("GetProductType", new { id =
productType.Id }, productType);
        }

        // DELETE: api/ProductTypes/5
        [HttpDelete("{id}")]
        public async Task<IActionResult>
DeleteProductType(int id)
        {
            var productType = await
_context.ProductTypes.FindAsync(id);
            if (productType == null)
            {
                return NotFound();
            }


_context.ProductTypes.Remove(productType);
            await _context.SaveChangesAsync();

            return NoContent();
        }

        private bool ProductTypeExists(int id)
        {
            return _context.ProductTypes.Any(e =>
e.Id == id);
        }
    }
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using RollerView.Data;
using RollerView.Models;
using RollerView.RecivedModels;

namespace RollerView.Controller
{
    [Route("api/[controller]")]
    [ApiController]
    public class ReviewsController : ControllerBase
    {
```

```
        private readonly DataContext _context;
        private readonly
UserManager<IdentityUser> _userManager;
        public ReviewsController(DataContext
context, UserManager<IdentityUser>
userManager)
        {
            _context = context;
            _userManager = userManager;
        }

        // GET: api/Reviews
        [HttpGet]
        public async
Task<ActionResult<IEnumerable<Review>>>
GetReviews()
        {
            return await
_context.Reviews.ToListAsync();
        }

        // GET: api/Reviews/5
        [HttpGet("{id}")]
        public async Task<ActionResult<Review>>
GetReview(int id)
        {
            var review = await
_context.Reviews.FindAsync(id);

            if (review == null)
            {
                return NotFound();
            }

            return review;
        }

        [HttpGet("get_example")]
        public ActionResult<Review> GetExample()
        {
            Review r = new Review
            {
                UserId = "",
                Title = "",
                GeneralInformation = "",
                BuildQualitySection = "",
                ComfOfWearingSection = "",
                ComfOfRidingSection = "",
                CreatedDate = new DateTime()
            };

            return r;
        }


        // PUT: api/Reviews/5
        // To protect from overposting attacks, see
https://go.microsoft.com/fwlink/?linkid=2123754
        [HttpPut("{id}")]
        public async Task<IActionResult>
PutReview(int id, Review review)
        {
            if (id != review.Id)
            {
                return BadRequest();
            }

            _context.Entry(review).State =
EntityState.Modified;

            try
            {
                await _context.SaveChangesAsync();
            }
            catch (DbUpdateConcurrencyException)
            {
                if (!ReviewExists(id))
                {
                    return NotFound();
                }
                else
                {
                    throw;
                }
            }

            return NoContent();
        }

        // POST: api/Reviews
        // To protect from overposting attacks, see
https://go.microsoft.com/fwlink/?linkid=2123754
        [HttpPost]
        public async Task<ActionResult<Review>>
PostReview(R_Review review)
        {
            List<string> errors =
R_Review.CheckReview(review);
            if(errors.Count > 0){
                return BadRequest(errors);
            }

            var user = await
_userManager.GetUserAsync(User);
            if (user == null)
            {
                return NotFound();
            }

            var product = await
_context.Products.FirstOrDefaultAsync(p => p.Id
== review.product_id);
            if (product == null)
            {
                return NotFound();
            }


            Review r = new Review()
            {
                UserId = user.Id,
                Title = review.title,
                GeneralInformation =
review.general_information,
```

```
            BuildQualitySection =
review.build_quality,
            ComfOfWearingSection =
review.comfort_of_wearing,
            ComfOfRidingSection =
review.comfort_of_riding,
            CreatedDate = DateTime.UtcNow
        };

        UserProductRating rating = new
UserProductRating
        {
            UserId = user.Id,
            ProductId = product.Id,
            BuildQualityRating = review.bq_rating,
            ComfOfWearingRating =
review.cw_rating,
            ComfOfRidingRating =
review.cr_rating,
        };

        _context.UserProductRatings.Add(rating);
        _context.Reviews.Add(r);

        await _context.SaveChangesAsync();

        return Ok();
    }

    // DELETE: api/Reviews/5
    [HttpDelete("{id}")]
    public async Task<IActionResult>
DeleteReview(int id)
    {
        var review = await
_context.Reviews.FindAsync(id);
        if (review == null)
        {
            return NotFound();
        }

        _context.Reviews.Remove(review);
        await _context.SaveChangesAsync();

        return NoContent();
    }

    private bool ReviewExists(int id)
    {
        return _context.Reviews.Any(e => e.Id ==
id);
    }
  }
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using RollerView.Data;
using RollerView.Models;

namespace RollerView.Controller
{
  [Route("api/[controller]")]
  [ApiController]
  public class TypesOfBootController :
ControllerBase
  {
    private readonly DataContext _context;

    public TypesOfBootController(DataContext
context)
    {
      _context = context;
    }

    // GET: api/TypeOfBoots
    [HttpGet]
    public async
Task<ActionResult<IEnumerable<TypeOfBoot>>
> GetTypesOfBoot()
    {
      List<TypeOfBoot> l = await
_context.TypesOfBoot.ToListAsync();
      l = l.OrderBy(item => item.Name).ToList();
      return l;
    }

    // GET: api/TypeOfBoots/5
    [HttpGet("{id}")]
    public async
Task<ActionResult<TypeOfBoot>>
GetTypeOfBoot(int id)
    {
      var typeOfBoot = await
_context.TypesOfBoot.FindAsync(id);

      if (typeOfBoot == null)
      {
        return NotFound();
      }

      return typeOfBoot;
    }

    [HttpGet("get_example")]
    public ActionResult<TypeOfBoot>
GetExample()
    {
      TypeOfBoot b = new TypeOfBoot
      {
        Name = ""
      };

      return b;
    }

    // PUT: api/TypeOfBoots/5
    // To protect from overposting attacks, see
https://go.microsoft.com/fwlink/?linkid=2123754
    [HttpPut("{id}")]
```

```
        public async Task<IActionResult>
PutTypeOfBoot(int id, TypeOfBoot typeOfBoot)
        {
            if (id != typeOfBoot.Id)
            {
                return BadRequest();
            }

            _context.Entry(typeOfBoot).State =
EntityState.Modified;

            try
            {
                await _context.SaveChangesAsync();
            }
            catch (DbUpdateConcurrencyException)
            {
                if (!TypeOfBootExists(id))
                {
                    return NotFound();
                }
                else
                {
                    throw;
                }
            }

            return NoContent();
        }

        // POST: api/TypeOfBoots
        // To protect from overposting attacks, see
https://go.microsoft.com/fwlink/?linkid=2123754
        [HttpPost]
        public async
Task<ActionResult<TypeOfBoot>>
PostTypeOfBoot(TypeOfBoot typeOfBoot)
        {
            _context.TypesOfBoot.Add(typeOfBoot);
            await _context.SaveChangesAsync();

            return
CreatedAtAction("GetTypeOfBoot", new { id =
typeOfBoot.Id }, typeOfBoot);
        }

        // DELETE: api/TypeOfBoots/5
        [HttpDelete("{id}")]
        public async Task<IActionResult>
DeleteTypeOfBoot(int id)
        {
            var typeOfBoot = await
_context.TypesOfBoot.FindAsync(id);
            if (typeOfBoot == null)
            {
                return NotFound();
            }


_context.TypesOfBoot.Remove(typeOfBoot);
            await _context.SaveChangesAsync();

            return NoContent();
        }

        private bool TypeOfBootExists(int id)
        {
            return _context.TypesOfBoot.Any(e =>
e.Id == id);
        }
    }
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using RollerView.Data;
using RollerView.Models;
using RollerView.RecivedModels;
using RollerView.SentModels;

namespace RollerView.Controller
{
    [Route("api/[controller]")]
    [ApiController]
    public class UserDetailsController :
ControllerBase
    {
        private readonly DataContext _context;
        private readonly
UserManager<IdentityUser> _userManager;
        public UserDetailsController(DataContext
context, UserManager<IdentityUser>
userManager)
        {
            _context = context;
            _userManager = userManager;
        }

        // GET: api/Users_Details
        [HttpGet, Authorize]
        public async
Task<ActionResult<S_UserDetails>>
GetCurUserDetails()
        {
            var user = await
_userManager.GetUserAsync(User);
            if (user == null) {
                return NotFound();
            }
            var userDetails = await
_context.Users_Details.FindAsync(user.Id);
            if (userDetails == null)
            {
                return NotFound();
            }
            S_UserDetails s_UserDetails = new
S_UserDetails
            {
```

```
            userId = userDetails.UserId,
            userName = userDetails.UserName,
            birthDay = userDetails.BirthDay,
            gender = userDetails.Gender,
            email = user.Email
        };
        return s_UserDetails;
    }

    // GET: api/Users_Details/5
    [HttpGet("{id}")]
    public async
Task<ActionResult<UserDetails>>
GetUserDetails(string id)
    {
        var userDetails = await
_context.Users_Details.FindAsync(id);

        if (userDetails == null)
        {
            return NotFound();
        }

        return userDetails;
    }

    // PUT: api/Users_Details/5
    // To protect from overposting attacks, see
https://go.microsoft.com/fwlink/?linkid=2123754
    [HttpPut("{id}"), Authorize]
    public async Task<IActionResult>
PutUserDetails(string id, UserDetails userDetails)
    {
        if (id != userDetails.UserId)
        {
            return BadRequest();
        }

        _context.Entry(userDetails).State =
EntityState.Modified;

        try
        {
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateConcurrencyException)
        {
            if (!UserDetailsExists(id))
            {
                return NotFound();
            }
            else
            {
                throw;
            }
        }

        return NoContent();
    }

    // POST: api/Users_Details
    // To protect from overposting attacks, see
https://go.microsoft.com/fwlink/?linkid=2123754
    [HttpPost, Authorize]
    public async
Task<ActionResult<UserDetails>>
PostUserDetails(R_UserDetails r_UserDetails)
    {
        var user = await
_userManager.GetUserAsync(User);
        if (user == null)
        {
            return NotFound();
        }
        UserDetails userDetails = new UserDetails
        {
            UserId = user.Id,
            UserName = r_UserDetails.userName,
            BirthDay = r_UserDetails.birthDay,
            Gender = r_UserDetails.gender,
        };
        _context.Users_Details.Add(userDetails);
        try
        {
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateException)
        {
            if
(UserDetailsExists(userDetails.UserId))
            {
                return Conflict();
            }
            else
            {
                throw;
            }
        }

        return CreatedAtAction("GetUserDetails",
new { id = userDetails.UserId }, userDetails);
    }

    // DELETE: api/Users_Details/5
    [HttpDelete("{id}"), Authorize]
    public async Task<IActionResult>
DeleteUserDetails(string id)
    {
        var userDetails = await
_context.Users_Details.FindAsync(id);
        if (userDetails == null)
        {
            return NotFound();
        }


_context.Users_Details.Remove(userDetails);
        await _context.SaveChangesAsync();

        return NoContent();
    }

    private bool UserDetailsExists(string id)
```

```
        {
            return _context.Users_Details.Any(e =>
e.UserId == id);
        }
    }
}
using Microsoft.AspNetCore.Identity;
using
Microsoft.AspNetCore.Identity.EntityFramework
Core;
using Microsoft.EntityFrameworkCore;
using RollerView.Models;

namespace RollerView.Data
{
    public class DataContext :
IdentityDbContext<IdentityUser>
    {
        public
DataContext(DbContextOptions<DataContext>
options) : base(options) {
        }
        public DbSet<Comment> Comments { get;
set; }
        public DbSet<UserDetails> Users_Details {
get; set; }
        public DbSet<UserRating> UserRatings {
get; set; }

        public DbSet<Product> Products { get; set; }

        public DbSet<BearingRating>
BearingRatings { get; set; }

        public DbSet<BootMaterial> BootMaterials {
get; set; }

        public DbSet<Brand> Brands { get; set; }

        public DbSet<PlateType> PlateTypes { get;
set; }

        public DbSet<ProductType> ProductTypes {
get; set; }
        public DbSet<Review> Reviews { get; set; }
        public DbSet<TypeOfBoot> TypesOfBoot {
get; set; }
        public DbSet<UserProductRating>
UserProductRatings { get; set; }
        public DbSet<UserReviewLike>
UserReviewLikes { get; set; }
        public DbSet<UserCommentLike>
UserCommentLikes { get; set; }
        public DbSet<ProductImage> ProductImages
{ get; set; }

        protected override void
OnModelCreating(ModelBuilder modelBuilder)
        {
            base.OnModelCreating(modelBuilder);
        }
    }
}
namespace RollerView.Models
{
    public class BearingRating
    {
        public int Id { get; set; }
        public string Name { get; set; }
    }
}
namespace RollerView.Models
{
    public class BootMaterial
    {
        public int Id { get; set; }
        public string Name { get; set; }
    }
}
namespace RollerView.Models
{
    public class Brand
    {
        public int Id { get; set; }
        public string Name { get; set; }
    }
}
using Microsoft.AspNetCore.Identity;
using
System.ComponentModel.DataAnnotations.Sche
ma;

namespace RollerView.Models
{
    public class Comment
    {
        public int Id { get; set; }

        public string? UserId { get; set; }
        public string Text { get; set; }

        [ForeignKey("UserId")]
        public IdentityUser? identityUser { get; set; }

        public DateTime CreateDate { get; set; }

    }
}
namespace RollerView.Models
{
    public class PlateType
    {
        public int Id { get; set; }
        public string Name { get; set; }
    }
}
using System.ComponentModel;

namespace RollerView.Models
{
    public class Product
    {
```

```
        public int Id { get; set; }
        public string Name { get; set; }

        public DateTime CreationDate { get; set; }

        public int WheelSize { get; set; }

        public int WheelHardness { get; set; }
        public int MinPrice { get; set; }
        public int MaxPrice { get; set; }

        public string Color { get; set; }

        public int MinSize { get; set; }
        public int MaxSize { get; set; }

        public bool HasBrakes { get; set; }

        public bool ForKids { get; set; }

        public BearingRating BearingRating { get;
set; }

        public TypeOfBoot TypeOfBoot { get; set; }

        public Brand Brand { get; set; }
        public ProductType ProductType { get; set; }
        public BootMaterial BootMaterial { get; set; }

        public PlateType PlateType { get; set; }

        public ICollection<Review>? Reviews { get;}
        public ICollection<ProductImage>?
ProductImages { get;}

    }
}
using System.ComponentModel.DataAnnotations;
using
System.ComponentModel.DataAnnotations.Sche
ma;

namespace RollerView.Models
{
    public class ProductImage
    {
        public int Id { get; set; }
        [NotMapped]
        public byte[] Image { get; set; }
    }
}
namespace RollerView.Models
{
    public class ProductType
    {
        public int Id { get; set; }
        public string Name { get; set; }
    }
}
using Microsoft.AspNetCore.Identity;
using
System.ComponentModel.DataAnnotations.Sche
ma;

namespace RollerView.Models
{
    public class Review
    {
        public int Id { get; set; }

        public string UserId { get; set; }

        [ForeignKey("UserId")]
        public IdentityUser identityUser { get; set; }

        public string Title { get; set; }

        public DateTime CreatedDate { get; set; }
        public string GeneralInformation { get; set; }

        public string BuildQualitySection { get; set; }

        public string ComfOfWearingSection { get;
set; }
        public string ComfOfRidingSection { get; set;
}

        public int NumberOfLikes {  get; set; }

        public ICollection<Comment> Comments {
get;}

    }
}
namespace RollerView.Models
{
    public class TypeOfBoot
    {
        public int Id { get; set; }
        public string Name { get; set; }
    }
}
using Microsoft.AspNetCore.Identity;
using
System.ComponentModel.DataAnnotations.Sche
ma;

namespace RollerView.Models
{
    public class UserCommentLike
    {
        public int Id { get; set; }

        public string? UserId { get; set; }

        [ForeignKey("UserId")]
        public IdentityUser? identityUser { get; set; }

        public int CommentId { get; set; }

        [ForeignKey("CommentId")]
        public Comment comment { get; set; }
```

```
    }
}
using Microsoft.AspNetCore.Identity;
using
System.ComponentModel.DataAnnotations.Sche
ma;
using System.ComponentModel.DataAnnotations;

namespace RollerView.Models
{
    public class UserDetails
    {
        [Key]
        public string UserId { get; set; }

        [ForeignKey("UserId")]
        public IdentityUser identityUser { get; set; }
        [Required]
        public string UserName { get; set; }
        [Required]
        public DateTime BirthDay { get; set; }
        [Required]
        public string Gender { get; set; }

    }
}
using Microsoft.AspNetCore.Identity;
using
System.ComponentModel.DataAnnotations.Sche
ma;

namespace RollerView.Models
{
    public class UserProductRating
    {
        public int Id { get; set; }

        public string UserId{ get; set; }

        [ForeignKey("UserId")]
        public IdentityUser identityUser { get; set; }

        public int ProductId { get; set; }
        [ForeignKey("ProductId")]
        public Product product { get; set; }

        public int BuildQualityRating { get; set; }
        public int ComfOfWearingRating { get; set; }
        public int ComfOfRidingRating { get; set; }
    }
}
using Microsoft.AspNetCore.Identity;
using System.ComponentModel.DataAnnotations;
using
System.ComponentModel.DataAnnotations.Sche
ma;

namespace RollerView.Models
{
    public class UserRating
    {
        [Key]
        public string UserId { get; set; }
        [ForeignKey("UserId")]
        public IdentityUser identityUser { get; set; }

        public int Rating { get; set; }

    }
}
using Microsoft.AspNetCore.Identity;
using
System.ComponentModel.DataAnnotations.Sche
ma;

namespace RollerView.Models
{
    public class UserReviewLike
    {
        public int Id { get; set; }

        public string? UserId { get; set; }

        [ForeignKey("UserId")]
        public IdentityUser? identityUser { get; set; }

        public int ReviewId { get; set; }

        [ForeignKey("ReviewId")]
        public Review review { get; set; }
    }
}
namespace RollerView.RecivedModels.SubClasses
{
    public class FilterOptions
    {
        public List<string> brands { get; set; } = new
List<string>();
        public List<string> plate_types { get; set; }  =
new List<string>();
        public List<string> boot_materials { get; set;
}  = new List<string>();
        public List<string> bearing_ratings { get; set;
} = new List<string>();
        public List<string> types_of_boot { get; set; }
= new List<string>();
        public List<string> product_types { get; set; }
= new List<string>();
        public List<int> wheel_sizes { get; set; } =
new List<int>();
        public List<int> wheel_hardness { get; set; }
= new List<int>();
        public int[] size_range { get; set; } = new
int[2];
        public int[] price_range { get; set; } = new
int[2];
        public bool has_brakes { get; set; } = false;
        public bool for_kids { get; set; } = false;


    }
}
using RollerView.Models;
```

```
namespace RollerView.RecivedModels
{
   public class R_Product
   {
      public string model_name { get; set; }
      public string brand { get; set; }
      public string plate_type { get; set; }
      public string boot_material { get; set; }

      public int wheel_size { get; set; }

      public int wheel_hardness { get; set; }
      public int min_price { get; set; }
      public int max_price { get; set; }

      public string color { get; set; }

      public int min_size { get; set; }
      public int max_size { get; set; }

      public bool has_brakes { get; set; }

      public bool for_kids { get; set; }
      public string bearing_rating { get; set; }
      public TypeOfBoot type_of_boot { get; set; }

      public ProductType type { get; set; }

   }
}
using RollerView.RecivedModels.SubClasses;

namespace RollerView.RecivedModels
{
   public class R_ProductRequestOptions
   {
      public FilterOptions filter { get; set; } = new
FilterOptions();
      public int page { get; set; } = 1;
      public string time_scope { get; set; } = "1
month";
      public string sort_command { get; set; } =
"Date, new to ol";
   }
}
using Microsoft.AspNetCore.Identity;
using RollerView.Models;
using
System.ComponentModel.DataAnnotations.Sche
ma;

namespace RollerView.RecivedModels
{
   public class R_Review
   {
      public int product_id { get; set; }

      public string title { get; set; }
      public string general_information { get; set; }

      public string build_quality { get; set; }
      public string comfort_of_wearing { get; set; }
      public string comfort_of_riding { get; set; }

      public int bq_rating { get; set; }

      public int cw_rating { get; set; }
      public int cr_rating { get; set; }

      public static List<string>
CheckReview(R_Review review) {
         List<string> errors = new List<string>();

         if (review.product_id < 0) {
            errors.Add("Incorrect product data.");
         }
         if (review.title == "")
         {
            errors.Add("Title can't be empty.");
         }
         if (review.title.Length > 100)
         {
            errors.Add("Title can't more than 100
characters.");
         }


         if (review.general_information.Length <
150)
         {
            errors.Add("General information can't
less than 150 characters.");
         }
         if (review.general_information.Length >
2000)
         {
            errors.Add("General information can't
more than 2000 characters.");
         }

         if (review.build_quality.Length < 150)
         {
            errors.Add("Build quality can't less than
150 characters.");
         }
         if (review.build_quality.Length > 2000)
         {
            errors.Add("Build quality can't more
than 2000 characters.");
         }

         if (review.comfort_of_wearing.Length <
150)
         {
            errors.Add("Comfort of wearing can't
less than 150 characters.");
         }
         if (review.comfort_of_wearing.Length >
2000)
         {
            errors.Add("Comfort of wearing can't
more than 2000 characters.");
         }
```

```
            if (review.bq_rating == 0) {
                errors.Add("BQ rating can't be 0.");
            }
            if (review.cw_rating == 0)
            {
                errors.Add("CW rating can't be 0.");
            }
            if (review.cr_rating == 0)
            {
                errors.Add("CR rating can't be 0.");
            }

            return errors;
        }

    }
}
namespace RollerView.RecivedModels
{
    public class R_UserDetails
    {
        public String userName { get; set; }
        public DateTime birthDay { get; set; }
        public String gender { get; set; }
    }
}
namespace RollerView.SentModels
{
    public class S_ProductName
    {
        public int id { get; set; }

        public string name { get; set; }
    }
}
namespace RollerView.SentModels
{
    public class S_Range
    {
        public int min {  get; set; }
        public int max { get; set; }
    }
}
namespace RollerView.SentModels
{
    public class S_UserDetails
    {
        public string userId { get; set; }
        public string userName { get; set; }
        public DateTime birthDay { get; set; }
        public string gender { get; set; }
        public string email { get; set; }
    }
}
```

```
<template>
<div class="container d-flex
justify-content-center">
  <div class="col-8">
    Accessible tables
    <button @click="ChangeTable(table)"
class="btn btn-primary me-2" v-for="table in
tables" :key="table">{{table}}</button>
    <DBTable :TableName="table_name"
v-if="table_name!="" />
  </div>

</div>
</template>

<script>
import DBTable from
'../components/DBTableCRUD.vue'

export default {
  name: 'ChangeDB',
  components: {
    DBTable
  },
  data(){
    return{
      tables:["Brands", "BootMaterials",
"PlateTypes", "ProductTypes", "TypesOfBoot",
"BearingRatings", "Products", "Reviews"],
      table_name:""
    }

  }
  ,
  methods:{
    ChangeTable(name){
      this.table_name = name;
    }
  }

}
</script>
```

```
<template>
    <div class="d-flex justify-content-center
container inputform">
    <svg xmlns="http://www.w3.org/2000/svg"
class="d-none">
        <symbol id="exclamation-triangle-fill"
viewBox="0 0 16 16">
          <path d="M8.982 1.566a1.13 1.13 0 0 0 0-1.96
0L.165 13.233c-.457.778.091 1.767.98
1.767h13.713c.889 0 1.438-.99.98-1.767L8.982
1.566zM8 5c.535 0 .954.462.9.995l-.35
3.507a.552.552 0 0 1-1.1 0L7.1 5.995A.905.905 0 0
1 8 5zm.002 6a1 1 0 1 1 0 2 1 1 0 0 1 0-2z"/>
        </symbol>
    </svg>

    <div class="row justify-content-center col-6
sug_prod" >
```

```
    <h2 class="col-6 text-center mt-5 mb-4
bold">Suggest product</h2>
    <form action="" class="">

      <div class="mb-4">
        <label class="form-label bold">Model
name</label>
        <input type="text"
class="form-control" required
@keyup="check_forms()"
v-model="product.model_name">
      </div>

      <div class="mb-4">
        <label class="form-label bold">Brand
name</label>
        <input type="text"
class="form-control" required
@keyup="check_forms()"
v-model="product.brand">
      </div>

      <div class="mb-4">
        <label class="form-label me-3
bold">Type of skates</label>
        <select @change="check_forms()"
class="form-select" aria-label="Default select
example" v-model="product.type.name">
          <option disabled value=""
selected>Select</option>
          <option v-for="type in
lists_of_choises.types"
@click="select_product_type(type)"
:key="type">{{type.name}}</option>
        </select>
      </div>


      <div class="mb-4">
        <label class="form-label me-3
bold">Boot type</label>
        <select @change="check_forms()"
class="form-select" aria-label="Default select
example"
v-model="product.type_of_boot.name">
          <option disabled value=""
selected>Select</option>
          <option v-for="type_of_boot in
lists_of_choises.types_of_boot"
@click="select_product_type_of_boot(type_of_bo
ot)"
:key="type_of_boot">{{type_of_boot.name}}</op
tion>
        </select>
      </div>

      <div class="mb-4">
        <label class="form-label me-3
bold">Bearing rating</label>
        <select @change="check_forms()"
class="form-select" aria-label="Default select
example" v-model="product.bearing_rating">
          <option disabled value=""
selected>Select</option>
          <option v-for="bearing_rating in
lists_of_choises.bearing_ratings"
:key="bearing_rating">{{bearing_rating}}</optio
n>
        </select>
      </div>


      <div class="mb-4">
        <label class="form-label me-3
bold">Color</label>
        <input type="text"
class="form-control" required
@keyup="check_forms()"
v-model="product.color">
      </div>
      <div class="mb-4">
        <label class="form-label me-3
bold">Type of plate</label>
        <input type="text"
class="form-control" required
@keyup="check_forms()"
v-model="product.plate_type">
      </div>
      <div class="mb-4">
        <label class="form-label me-3
bold">Boot Material</label>
        <input type="text"
class="form-control" required
@keyup="check_forms()"
v-model="product.boot_material">
      </div>

      <div class="mb-4">
        <label class="form-label me-3
bold">Size range (Euro Size)</label> <br>
        <span>Min =  </span>
        <input type="number"
class="form-control d-inline-block w-25"
required @change="check_forms()"
v-model="product.min_size" min="0"
max="100">

        <span class = "ms-5">Max =  </span>
        <input type="number"
class="form-control d-inline-block w-25 "
required @change="check_forms()"
v-model="product.max_size" min="0"
max="100">
      </div>

      <div class="mb-4">
        <label class="form-label me-3
bold">Price range ($)</label> <br>
        <span>Min =  </span>
        <input type="number"
class="form-control d-inline-block w-25"
required @change="check_forms()"
v-model="product.min_price" min="0">
```

```
            <span class = "ms-5">Max =  </span>
            <input type="number"
class="form-control d-inline-block w-25 "
required @change="check_forms()"
v-model="product.max_price" min="0">
        </div>

        <div class="mb-4">
            <label class="form-label bold">Wheel
size in millimeters</label>
            <input type="number"
class="form-control w-25" required
@change="check_forms()"
v-model="product.wheel_size" min="0">
        </div>


        <div class="mb-4">
            <label class="form-label bold">Wheel
hardness</label>
            <input type="number"
class="form-control w-25" required
@change="check_forms()"
v-model="product.wheel_hardness" min="0">
        </div>

        <div class="mb-4 d-flex
align-items-center">
            <input type="checkbox"
class="form-check-input mt-0 me-3" required
@change="check_forms()"
v-model="product.for_kids">
            <label class="form-check-label bold">Is
for kids</label>
        </div>
        <div class="mb-4 d-flex
align-items-center">
            <input type="checkbox"
class="form-check-input mt-0 me-3" required
@change="check_forms()"
v-model="product.has_brakes">
            <label class="form-check-label
bold">Has build in brakes</label>
        </div>


        <div class="alert alert-danger d-flex mt-1"
role="alert" v-if="errors.length != 0">
            <svg class="bi flex-shrink-0 me-2 "
width="24" height="24" role="img"
aria-label="Danger:"><use
xlink:href="#exclamation-triangle-fill"/></svg>
            <div>
                <div v-for="error in errors"
:key="error" class="d-block">
                    {{error}}
                </div>
            </div>
        </div>

        <div class="container d-flex
justify-content-center pt-2 pb-5 px-0">
            <button type="button"
@click="create_product()" class="btn
btn-primary btn-lg w-100">Create</button>
        </div>

    </form>
  </div>
  </div>
</template>

<script>
export default{
  name: 'SignIn',
  data(){
    return{
      product:{
        model_name:'',
        color: '',
        brand: '',
        plate_type: '',
        boot_material:'',
        bearing_rating: "",
        wheel_size: 0,
        wheel_hardness: 0,
        min_size:0,
        max_size:0,
        min_price: 0,
        max_price: 0,
        has_brakes: false,
        for_kids:false,

        type: {
          id:-1,
          name:""
        },
        type_of_boot:{
          id:-1,
          name:""
        }


      },
      lists_of_choises:{
        types: [],
        types_of_boot:[],
        bearing_ratings:[
          "ABEC 1",
          "ABEC 3",
          "ABEC 5",
          "ABEC 7",
          "ABEC 9"
        ]
      },
      errors:[],
      errors_active: false

    }
  },
  mounted(){
    this.get_all_tables()
  },
  methods:{
```

```
    async get_all_tables(){
        this.lists_of_choises.types = await
this.get_table("ProductTypes");
        this.lists_of_choises.types_of_boot = await
this.get_table("TypesOfBoot");
    },
    select_product_type(type){
        this.product.type = type;
    }
    ,
    select_product_type_of_boot(type_of_boot){
        this.product.type_of_boot = type_of_boot;
    }
    ,
    check_forms(){
        if(this.errors_active){
            this.errors = [];
            if(this.product.model_name.length ==
0){
                this.errors.push("Model name can't
be empty");
            }
            if(this.product.model_name.length >
100){
                this.errors.push("Model name can't
be longer than 100 symbols");
            }
            if(this.product.wheel_size < 1){
                this.errors.push("Wheel size can't be
less than 1");
            }
            if(this.product.wheel_hardness < 1){
                this.errors.push("Wheel hardness
can't be less than 1");
            }
            if(this.product.color.replace(/\s+/g,
'').length == 0){
                this.errors.push("Color can't be
empty");
            }
            if(this.product.min_size < 1){
                this.errors.push("Minimal size can't
be less than 1");
            }
            if(this.product.max_size > 100 ){
                this.errors.push("Maximum size can't
be more than 100");
            }
            if(this.product.max_size <
this.product.min_size ){
                this.errors.push("Maximum size can't
less than minimal size");
            }
            if(this.product.max_price <
this.product.min_price ){
                this.errors.push("Maximum price
can't less than minimal price");
            }
            if(this.product.brand.length == 0){
                this.errors.push("Brand name can't
be empty");
            }
            if(this.product.brand.length > 100){
                this.errors.push("Brand name can't
be longer than 100 symbols");
            }
            if(this.product.type.id == -1){
                this.errors.push("Roller skates type
has to be chosen");
            }
            if(this.product.type_of_boot.id == -1){
                this.errors.push("Type of boot has to
be chosen");
            }
            return this.errors.length == 0;
        }
    }
    ,
    async create_product(){

        this.errors_active = true;
        if(this.check_forms()){
            try {
            const response = await
fetch("https://localhost:7041/api/Products", {
                method: "POST",
                mode: 'cors',
                headers: {
                    "Content-Type": "application/json"
                },
                credentials: 'include',
                body: JSON.stringify(this.product)
            });
            const result = await response.status;
            if (result == 200){
                console.log("Success: ", result);
            }
            else{
                console.log("Failer: ", result);
                let server_errors = await
response.json();
                for (let key in server_errors.errors){

this.errors.push(server_errors.errors[key][0]);
                }
            }

            } catch (error) {
                console.error("Error:", error);
            }
        }
    },
    async get_table(table_name){
      try {
          const response = await
fetch("https://localhost:7041/api/" + table_name, {
          method: "GET",
          mode: 'cors',
          headers: {
            "Content-Type": "application/json",
          },
          credentials: 'include'
        });
        var result;
```

```
        if(response.status == 200 &&
response.headers.get("Content-Type") != null){
          result = await response.json();
          console.log("Success: ", result[0]);
          return result;
        }
        else{
          console.log("Failure:", result);
          return [];
        }
       }
       catch(error) {
        console.error("Error:", error);
        return [];
       }
      }
  }
}
</script>

<style>
#app .sug_prod .form-check-input{
  width: 20px;
  height: 20px;
}
</style>



<template>
  <svg xmlns="http://www.w3.org/2000/svg"
class="d-none">
    <symbol id="exclamation-triangle-fill"
viewBox="0 0 16 16">
      <path d="M8.982 1.566a1.13 1.13 0 0 0-1.96
0L.165 13.233c-.457.778.091 1.767.98
1.767h13.713c.889 0 1.438-.99.98-1.767L8.982
1.566zM8 5c.535 0 .954.462.9.995l-.35
3.507a.552.552 0 0 1-1.1 0L7.1 5.995A.905.905 0 0
1 8 5zm.002 6a1 1 0 1 1 0 2 1 1 0 0 1 0-2z"/>
    </symbol>
  </svg>
  <div class="d-flex justify-content-center
container inputform">
  <div class="row justify-content-center col-6">
    <h2 class="col-6 text-center mt-5 mb-4
bold">Log In</h2>
    <form action="" class="">
      <div calss="mb-4">
        <label class="form-label
bold">Email</label>
        <input type="email"
class="form-control" required
v-model="login_data.email">
      </div>
      <div calss="mb-4">
        <label class="form-label
bold">Password</label>
        <input type="password"
class="form-control" required
v-model="login_data.password">
      </div>
      <div class="alert alert-danger d-flex mt-1"
role="alert" v-if="error != ''">
        <svg class="bi flex-shrink-0 me-2 "
width="24" height="24" role="img"
aria-label="Danger:"><use
xlink:href="#exclamation-triangle-fill"/></svg>
        <div>
          {{error}}
        </div>
      </div>

      <div class="container d-flex
justify-content-center pt-3 px-0">
        <button type="button" @click="login"
class="btn btn-primary btn-lg w-100">Log
In</button>
      </div>


    </form>
  </div>
  </div>

</template>

<script>
export default{
  name: 'LogIn',
  data(){
    return{
      login_data: {
        email:'',
        password:''
      },
      error:""
    }
  },
  methods:{
    async login(){
      try {
        const response = await
fetch("https://localhost:7041/login?useCookies=tr
ue", {
          method: "POST",
          mode: 'cors',
          headers: {
            "Content-Type": "application/json",
          },
          credentials: 'include',
          body: JSON.stringify(this.login_data)
        });
          const result = await response.status;
          if (result == 200){
            console.log("Success: ", result);
            await this.$emit("get_user_data");
            this.$router.push("/");
          }
          else{
            console.log("Failer: ", result);
            this.error = "Email or Password is
incorrect.";
```

```
            }

          } catch (error) {
             console.error("Error:", error);
          }

      }
    }
}
</script>

<style>
button{
margin-top:10px;
}

form{
margin-top:10px;
}
</style>



<template>
<div class="container d-flex
justify-content-center">
  <div class="col-8">
    <Filter :p_filter="for_request.filter"
    :loaded_lists="loaded_lists"
    :size_scope="this.size_scope"
    :price_scope="this.price_scope"/>
    <ReviewList :list_of_reviews="list_of_reviews"
/>
  </div>

</div>

</template>

<script>
import Filter from '../components/Filter.vue'
import ReviewList from
'../components/ReviewList.vue'

export default {
  name: 'ProductDiscovery',
  props:{
    p_page: Number,
    p_time_scope: String,
    p_sort_command: String,
    p_filter: Object,
  }
  ,
  components: {
    Filter,
    ReviewList
  },
  data(){
    return{
      for_request:{
        filter:{
          brands: [],
          colors: [],
          plate_types: [],
          boot_materials: [],
          bearing_ratings: [],
          types_of_boot: [],
          product_types: [],
          wheel_sizes: [],
          wheel_hardness: [],
          size_range: [1, 100],
          price_range:[1, 1000],
          has_brakes: false,
          for_kids: false,
        }
        ,
        page:1,

        time_scope: '1 month',

        sort_command: 'Date, new to old',
      }
      ,
      loaded_lists:{
        brands: [],
        colors: [],
        plate_types: [],
        boot_materials: [],
        bearing_ratings: [],
        types_of_boot: [],
        product_types: [],
        wheel_sizes: [],
        wheel_hardness: [],
      },

      size_scope: [1, 100],
      price_scope:[1, 1000],

      list_of_reviews: []

    }

  },
  mounted(){
    this.for_request.page = this.$props.p_page;
    this.for_request.time_scope =
this.$props.p_time_scope;
    this.for_request.sort_command =
this.$props.p_sort_command;
    this.for_request.filter = this.$props.p_filter;
    this.get_all_tables();
  },
  watch: {
    $route: function() {
      this.for_request.page = this.$props.p_page;
      this.for_request.time_scope =
this.$props.p_time_scope;
      this.for_request.sort_command =
this.$props.p_sort_command;
      this.for_request.filter = this.$props.p_filter;
    }
  }
  ,
```

```
  methods:{
    async get_all_tables(){
      this.loaded_lists.brands = await
this.get_request("Brands");
      this.loaded_lists.plate_types = await
this.get_request("PlateTypes");
      this.loaded_lists.boot_materials = await
this.get_request("BootMaterials");
      this.loaded_lists.bearing_ratings = await
this.get_request("BearingRatings");
      this.loaded_lists.types_of_boot = await
this.get_request("TypesOfBoot");
      this.loaded_lists.product_types = await
this.get_request("ProductTypes");

      this.loaded_lists.colors = await
this.get_request("Products/get_colors");
      this.loaded_lists.wheel_sizes = await
this.get_request("Products/get_wheel_sizes");
      this.loaded_lists.wheel_hardness = await
this.get_request("Products/get_wheel_hardness");

      this.size_scope = await
this.get_request("Products/get_size_range");
      this.for_request.filter.size_range =
this.size_scope;
      this.price_scope = await
this.get_request("Products/get_price_range");
      this.for_request.filter.price_range =
this.price_scope;

      this.list_of_reviews = await
this.post_request("Products/filtered",
this.for_request);
    },

    async get_request(request_text){
    try {
        const response = await
fetch("https://localhost:7041/api/" + request_text,
{
        method: "GET",
        mode: 'cors',
        headers: {
          "Content-Type": "application/json",
        },
        credentials: 'include'
      });
      var result;
      if(response.status == 200 &&
response.headers.get("Content-Type") != null){
        result = await response.json();
        console.log("Success: ", result);
        return result;

      }
      else{
        console.log("Failure:", result);
        return null;
      }
    }
    catch(error) {
      console.error("Error:", error);
      return null;
    }

  },
    async post_request(request_text, body){
    try {
        const response = await
fetch("https://localhost:7041/api/" + request_text,
{
        method: "POST",
        mode: 'cors',
        headers: {
          "Content-Type": "application/json",
        },
        credentials: 'include',
        body: JSON.stringify(body)
      });
      var result;
      if(response.status == 200 &&
response.headers.get("Content-Type") != null){
        result = await response.json();
        console.log("Success: ", result);
        return result;

      }
      else{
        console.log("Failure:", result);
        return null;
      }
    }catch (error) {
            console.error("Error:", error);
    }
    }
  }
}
</script>



<template>
    <div class="d-flex justify-content-center
container inputform">
        <svg xmlns="http://www.w3.org/2000/svg"
class="d-none">
            <symbol id="exclamation-triangle-fill"
viewBox="0 0 16 16">
            <path d="M8.982 1.566a1.13 1.13 0 0
0-1.96 0L.165 13.233c-.457.778.091 1.767.98
1.767h13.713c.889 0 1.438-.99.98-1.767L8.982
1.566zM8 5c.535 0 .954.462.9.995l-.35
3.507a.552.552 0 0 1-1.1 0L7.1 5.995A.905.905 0 0
1 8 5zm.002 6a1 1 0 1 1 0 2 1 1 0 0 1 0-2z"/>
            </symbol>
        </svg>
    <div class="row justify-content-center">
        <h2 class="col-6 text-center mt-5 mb-4
bold">Review Creation</h2>
        <form action="" class="col-12">
```

```
        <div class="mb-2">
          <span class="bold ">Choose product to
review</span>
        </div>
        <ProductSearch class="mb-3"
@item_selected="select_item" />

        <div calss="mb-4">
          <label class="form-label
bold">Title</label>
          <input v-model="review.title"
class="form-control" maxlength="100"
required>
        </div>

        <div calss="mb-4">
          <label class="form-label bold">General
information</label>
          <div class="position-relative">
            <textarea
v-model="review.general_information"
class="form-control" rows="4"
:maxlength="text_area.max_length"
required></textarea>
            <div class="position-absolute counter
bold"
            :class="{wrong:
review.general_information.length <
text_area.min_length ||
review.general_information.length >
text_area.max_length}">
              {{text_area.min_length}} &lt;
{{review.general_information.length}} &lt;
{{text_area.max_length}}
            </div>
          </div>
        </div>
        <div calss="">
          <div class="d-flex d-flex
justify-content-between align-items-center mt-4">
            <label class="form-label bold">Build
quality</label>
            <StarRating class=" ms-2"
:value="0" @set_bq_rating="set_bq_rating"
:func_to_emit="'set_bq_rating'"
:required="true" />
          </div>
          <div class="position-relative">
            <textarea
v-model="review.build_quality"
class="form-control" rows="4"
:maxlength="text_area.max_length"
required></textarea>
            <div class="position-absolute counter
bold"
            :class="{wrong:
review.build_quality.length <
text_area.min_length ||
review.build_quality.length >
text_area.max_length}" >
              {{text_area.min_length}} &lt;
{{review.build_quality.length}} &lt;
{{text_area.max_length}}
            </div>
          </div>

        </div>

        <div calss="mb-4 mt-4">
          <div class="d-flex d-flex
justify-content-between align-items-center mt-4">
            <label class="form-label
bold">Wearing comfort</label>
            <StarRating class="ms-2" :value="0"
@set_cw_rating="set_cw_rating"
:func_to_emit="'set_cw_rating'"
:required="true"/>
          </div>

          <div class="position-relative">
            <textarea
v-model="review.comfort_of_wearing"
class="form-control" rows="4"
:maxlength="text_area.max_length"
required></textarea>
            <div class="position-absolute counter
bold"
            :class="{wrong:
review.build_quality.length <
text_area.min_length ||
review.comfort_of_wearing.length >
text_area.max_length}" >
              {{text_area.min_length}} &lt;
{{review.comfort_of_wearing.length}} &lt;
{{text_area.max_length}}
            </div>
          </div>
        </div>

        <div calss="mb-4 mt-4">
          <div class="position-relative">
          <div class="d-flex d-flex
justify-content-between align-items-center mt-4">
            <label class="form-label
bold">Riding comfort</label>
            <StarRating class="d-inline-block
ms-2 " :value="0"
@set_cr_rating="set_cr_rating"
:func_to_emit="'set_cr_rating'"
:required="true" />
          </div>
            <textarea
v-model="review.comfort_of_riding"
class="form-control" rows="4"
:maxlength="text_area.max_length"
required></textarea>
            <div class="position-absolute counter
bold"
            :class="{wrong:
review.build_quality.length <
text_area.min_length ||
```

```
review.comfort_of_riding.length >
text_area.max_length}" >
                {{text_area.min_length}} &lt;
{{review.comfort_of_riding.length}} &lt;
{{text_area.max_length}}
            </div>
          </div>
        </div>

        <div class="alert alert-danger d-flex mt-1"
role="alert" v-if="errors.length != 0">
          <svg class="bi flex-shrink-0 me-2 "
width="24" height="24" role="img"
aria-label="Danger:"><use
xlink:href="#exclamation-triangle-fill"/></svg>
          <div>
            <div v-for="error in errors"
:key="error" class="d-block">
              {{error}}
            </div>
          </div>
        </div>

        <div class="container d-flex
justify-content-center pt-3 px-0">
          <button type="button"
@click="create_review()" class="btn
btn-primary btn-lg w-100">Create</button>
        </div>

      </form>
    </div>
    </div>
</template>

<script>
import StarRating from
'../components/StarRating.vue'
import ProductSearch from
'../components/ProductSearch.vue'

export default{
    name: 'LogIn',
    data(){
      return{
        review: {
          product_id: -1,
          title:'',
          general_information:'',
          build_quality:'',
          comfort_of_wearing:'',
          comfort_of_riding:'',
          bq_rating: 0,
          cw_rating: 0,
          cr_rating: 0
        },
        text_area:{
          min_length: 150,
          max_length: 2000
        },
        errors: []
      }
    },
    components: {
      StarRating,
      ProductSearch
    },
    methods:{
      select_item(id){
        this.review.product_id = id;

      }
      ,
      async create_review(){
        try {
          const response = await
fetch("https://localhost:7041/api/Reviews", {
            method: "POST",
            mode: 'cors',
            headers: {
              "Content-Type": "application/json",
            },
            credentials: 'include',
            body: JSON.stringify(this.review)
          });
            const result = await response.status;
            if (result == 200){
              this.errors = [];
              console.log("Success: ", result);
            }
            else{
              this.errors = await response.json();
              console.log("Failer: ", result,
this.errors);
            }

        } catch (error) {
          console.error("Error:", error);
        }

      },
      set_bq_rating(rating){
        this.review.bq_rating = rating;
      },
      set_cw_rating(rating){
        this.review.cw_rating = rating;
      },
      set_cr_rating(rating){
        this.review.cr_rating = rating;
      },

    }
}
</script>

<style>
button{
margin-top:10px;
}

form{
margin-top:10px;
}
```

```
#app .counter{
   bottom:3px;
   right:10px;
   font-size: 13px;
   color: #30b43b;
}
#app .counter.wrong{
   color: #eb0505;
}
</style>



<template>
  <div>
    <h3>Dynamic Checkbox List</h3>
    <div v-for="(value, key) in options"
:key="key">
      <input type="checkbox" :id="key"
:value="key" v-model="selectedOptions">
      <label :for="key">{{ key }} ({{ value
}})</label>
    </div>
    <p>Selected Options: {{ selectedOptions }}</p>
    <button @click="addOption">Add
Option</button>
    <button
@click="preCheckOptions">Pre-check
Options</button>
  </div>
</template>

<script>
export default {
  data() {
    return {
      options: {
        option1: 'Description 1',
        option2: 'Description 2',
        option3: 'Description 3'
      },
      selectedOptions: ['option1', 'option2'] //
Pre-checked options
    };
  },
  methods: {
    addOption() {
      const newKey =
`option${Object.keys(this.options).length + 1}`;
      this.$set(this.options, newKey, `Description
${Object.keys(this.options).length + 1}`);
    },
    preCheckOptions() {
      this.selectedOptions.push('option3'); //
Example of pre-checking another option
    }
  },

};
</script>

<style scoped>
/* Add your styles here */
</style>


<template>
   <div class="d-flex justify-content-center
container inputform">
   <svg xmlns="http://www.w3.org/2000/svg"
class="d-none">
      <symbol id="exclamation-triangle-fill"
viewBox="0 0 16 16">
        <path d="M8.982 1.566a1.13 1.13 0 0 0-1.96
0L.165 13.233c-.457.778.091 1.767.98
1.767h13.713c.889 0 1.438-.99.98-1.767L8.982
1.566zM8 5c.535 0 .954.462.9.995l-.35
3.507a.552.552 0 0 1-1.1 0L7.1 5.995A.905.905 0 0
1 8 5zm.002 6a1 1 0 1 1 0 2 1 1 0 0 1 0-2z"/>
      </symbol>
   </svg>

   <div class="row justify-content-center col-6">
      <h2 class="col-6 text-center mt-5 mb-4
bold">Sign In</h2>
      <form action="" class="">

         <div class="mb-4">
            <label class="form-label
bold">Username</label>
            <input type="text"
class="form-control" required
@keyup="check_username"
v-model="user_details.username">
            <div class="alert alert-danger d-flex
mt-1" role="alert" v-if="errors.username.length
!= 0">
               <svg class="bi flex-shrink-0 me-2 "
width="24" height="24" role="img"
aria-label="Danger:"><use
xlink:href="#exclamation-triangle-fill"/></svg>
               <div>
                  <div v-for="error in
errors.username" :key="error.id"
class="d-block">
                     {{error.data}}
                  </div>
               </div>
            </div>
         </div>
         <div class="mb-4">
            <label class="form-label me-3
bold">Birthday</label>
            <input @change="check_birthday()"
type="date" class="w-100 p-2 border rounded"
min="1900-01-01" max="{{new Date()}}"
v-model="user_details.birth_day"/>
            <div class="alert alert-danger d-flex
mt-1" role="alert" v-if="errors.birthday.length
!= 0">
               <svg class="bi flex-shrink-0 me-2 "
width="24" height="24" role="img"
aria-label="Danger:"><use
xlink:href="#exclamation-triangle-fill"/></svg>
```

```
                <div>
                  <div v-for="error in
    errors.birthday" :key="error.id"
    class="d-block">
                    {{error.data}}
                  </div>
                </div>

              </div>
            </div>
            <div class="mb-4">
              <label class="form-label me-3
    bold">Gender</label>
              <select @change="check_gender()"
    class="form-select" aria-label="Default select
    example" v-model="user_details.gender">
                <option disabled value=""
    selected>Select</option>
                <option
    value="Male">Male</option>
                <option
    value="Female">Female</option>
              </select>
              <div class="alert alert-danger d-flex
    mt-1" role="alert" v-if="errors.gender.length !=
    0">
                <svg class="bi flex-shrink-0 me-2 "
    width="24" height="24" role="img"
    aria-label="Danger:"><use
    xlink:href="#exclamation-triangle-fill"/></svg>
                <div>
                  <div v-for="error in errors.gender"
    :key="error.id" class="d-block">
                    {{error.data}}
                  </div>
                </div>

              </div>
            </div>

            <div class="mb-4">
              <label class="form-label
    bold">Email</label>
            <input type="email" class="form-control"
required @keyup="check_email"
v-model="sign_in_data.email">
              <div class="alert alert-danger d-flex
    mt-1" role="alert" v-if="errors.email.length !=
    0">
                <svg class="bi flex-shrink-0 me-2"
    width="24" height="24" role="img"
    aria-label="Danger:"><use
    xlink:href="#exclamation-triangle-fill"/></svg>
                <div v-for="error in errors.email"
    :key="error.id">
                  {{error.data}}
                </div>
              </div>
            </div>

            <div class="mb-4">
              <label class="form-label
bold">Password</label>
              <input type="password"
class="form-control" required
@keyup="check_password"
v-model="sign_in_data.password">
              <div class="alert alert-danger d-flex
mt-1" role="alert" v-if="errors.password.length
!= 0">
                <svg class="bi flex-shrink-0 me-2 "
width="24" height="24" role="img"
aria-label="Danger:"><use
xlink:href="#exclamation-triangle-fill"/></svg>
                <div>
                  <div v-for="error in
errors.password" :key="error.id"
class="d-block">
                    {{error.data}}
                  </div>
                </div>

              </div>
            </div>

            <div class="container d-flex
justify-content-center pt-2 pb-5 px-0">
              <button type="button"
@click="sign_in" class="btn btn-primary btn-lg
w-100">Sign In</button>
            </div>

          </form>
        </div>
        </div>
</template>

<script>
export default{
  name: 'SignIn',
  data(){
    return{
      sign_in_data: {
        email:'',
        password:''
      },
      user_details:{
        username:'',
        birth_day: '',
        gender: ''
      },
      errors:{
        username: [],
        email: [],
        password:[],
        birthday:[],
        gender:[],
        active: false
      }

    }
  },
  methods:{
```

```
        check_username(){
            if(this.errors.active){
                this.errors.username = [];
                if(this.user_details.username.length < 4){
                    this.errors.username = [{id:0,
data:"Username should at least be 4 characters
long."}];
                    return false;
                }
                return true;
            }
            return true;
        }
        ,
        check_birthday(){
            if(this.errors.active){
                this.errors.birthday = [];
                let br_date = new
Date(this.user_details.birth_day);
                let check = (br_date.getTime() > new
Date("1900-01-01").getTime())
                && (br_date.getTime() < new
Date().getTime());
                if(!check){
                    this.errors.birthday = [{id:0,
data:"Chose date before now and after
01/01/1900."}];
                    return false;
                }
                return true;
            }
            return true;
        }
        ,
        check_gender(){
            if(this.errors.active){
                this.errors.gender = [];
                if(this.user_details.gender == ""){
                    this.errors.gender = [{id:0,
data:"Gender must be selected."}];
                    return false;
                }
                return true;
            }
            return true;
        },
        check_email(){
            if(this.errors.active){
                this.errors.email = [];
                if(!String(this.sign_in_data.email)
                .toLowerCase()
                .match(

/^(([^<>()[\]\\.,;:\s@"]+(\.[^<>()[\]\\.,;:\s@"]+)*)|.(
".+"))@((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1
,3}\])|(([a-zA-Z\-0-9]+\.)+[a-zA-Z]{2,}))$/
                )){
                    this.errors.email = [{id:0,
data:"Invalid Email"}];
                    return false;
                }
                return true;
            }
            return true;
        }
        ,
        check_password(){
            if(this.errors.active){
                let pw = this.sign_in_data.password;
                this.errors.password = [];
                if(!(/[A-Z]/.test(pw))){
                    this.errors.password.push({id:1,
data:"Passwords must have at least one uppercase
('A'-'Z')."});
                }
                if(!(/[a-z]/.test(pw))){
                    this.errors.password.push({id:2,
data:"Passwords must have at least one lowercase
('a'-'z')."});
                }
                if(!(/[0-9]/.test(pw))){
                    this.errors.password.push({id:3,
data:"Passwords must have at least one digit
('0'-'9')."});
                }
                if(!(/[^A-Za-z0-9]/.test(pw))){
                    this.errors.password.push({id:4,
data:"Passwords must have at least one non
alphanumeric character."});
                }
                if(!(pw.length >= 6)){
                    this.errors.password.push({id:5,
data:"Passwords must be at least 6 characters."});
                }
                if(this.errors.password.length > 0){
                    return false;
                }
                else{
                    return true;
                }
            }
            return true;
        }
        ,
        async sign_in(){
            this.errors.active = true;
            let c_g = this.check_gender();
            let c_bd = this.check_birthday();
            let c_un = this.check_username();
            let c_e = this.check_email();
            let c_pw = this.check_password();
            if(c_e && c_pw && c_un && c_bd &&
c_g){
                try {
                const response = await
fetch("https://localhost:7041/register", {
                    method: "POST",
                    mode: 'cors',
                    headers: {
                        "Content-Type": "application/json"
                    },
                    credentials: 'include',
                    body:
JSON.stringify(this.sign_in_data)
```

```
            });
            const result = await response.status;
            if (result == 200){
               console.log("Success: ", result);
               if(await this.login()){
                  await this.send_details();
                  await this.$emit("get_user_data");
                  this.$router.push("/");
               }
            }
            else{
               console.log("Failer: ", result);
               let server_errors = await
response.json();
               let i = 6;
               for (let key in server_errors.errors){
                  this.errors.email.push({id:i,
data:server_errors.errors[key][0]});
                  i += 1;
               }
            }
            } catch (error) {
               console.error("Error:", error);
            }
         }
      },
      async login(){
         try {
            const response = await
fetch("https://localhost:7041/login?useCookies=tr
ue", {
            method: "POST",
            mode: 'cors',
            headers: {
               "Content-Type": "application/json",
            },
            credentials: 'include',
            body: JSON.stringify(this.sign_in_data)
         });
            const result = await response.status;
            if (result == 200){
               console.log("Success: ", result);
               return true;
            }
            else{
               console.log("Failer: ", result);
               return false;
            }
         } catch (error) {
            console.error("Error:", error);
            return false;
         }
      },
      async send_details(){
         try {
            const response = await
fetch("https://localhost:7041/api/UserDetails", {
            method: "POST",
            mode: 'cors',
            headers: {
               "Content-Type": "application/json",
            },
            credentials: 'include',
            body: JSON.stringify(this.user_details)
         });
            const result = await response.status;
            if (result == 200){
               console.log("Success: ", result);
               return true;
            }
            else{
               console.log("Failer: ", result);
               return false;
            }
         } catch (error) {
            console.error("Error:", error);
            return false;
         }
      }
   }
}
</script>
<style>
</style>
```