# Documentation of the “online store” project

The project is a user-friendly website for an online store. The website allows users to view products, add them to the cart, place orders, and manage their account. It will be created by Artur Šuškevitš, Denis Goryunov, and Oleksandr Bohatyrov.t

Sisukord

[Documentation of the “online store” project 1](#_Toc182827819)

[Idea and requirements analysis 3](#_Toc182827820)

[Technical analysis and evaluation. 4](#_Toc182827821)

[Planning and design 5](#_Toc182827822)

[Planning and prioritization 6](#_Toc182827823)

[Development 7](#_Toc182827824)

[Testing 8](#_Toc182827825)

[Code-review 9](#_Toc182827826)

[Documentation 10](#_Toc182827827)

[Testing on the production environment (Pre-release) 14](#_Toc182827828)

[Release 15](#_Toc182827829)

## Idea and requirements analysis

Make an online store, with different functionality

Make a convenient store with its own payment system.

People who want to buy through a user-friendly interface a variety of products.

Possibility of high site load.

## Technical analysis and evaluation.

Back-end:

ASP.NET Core Web API C#

Entity Framework (Embedded database in c#)

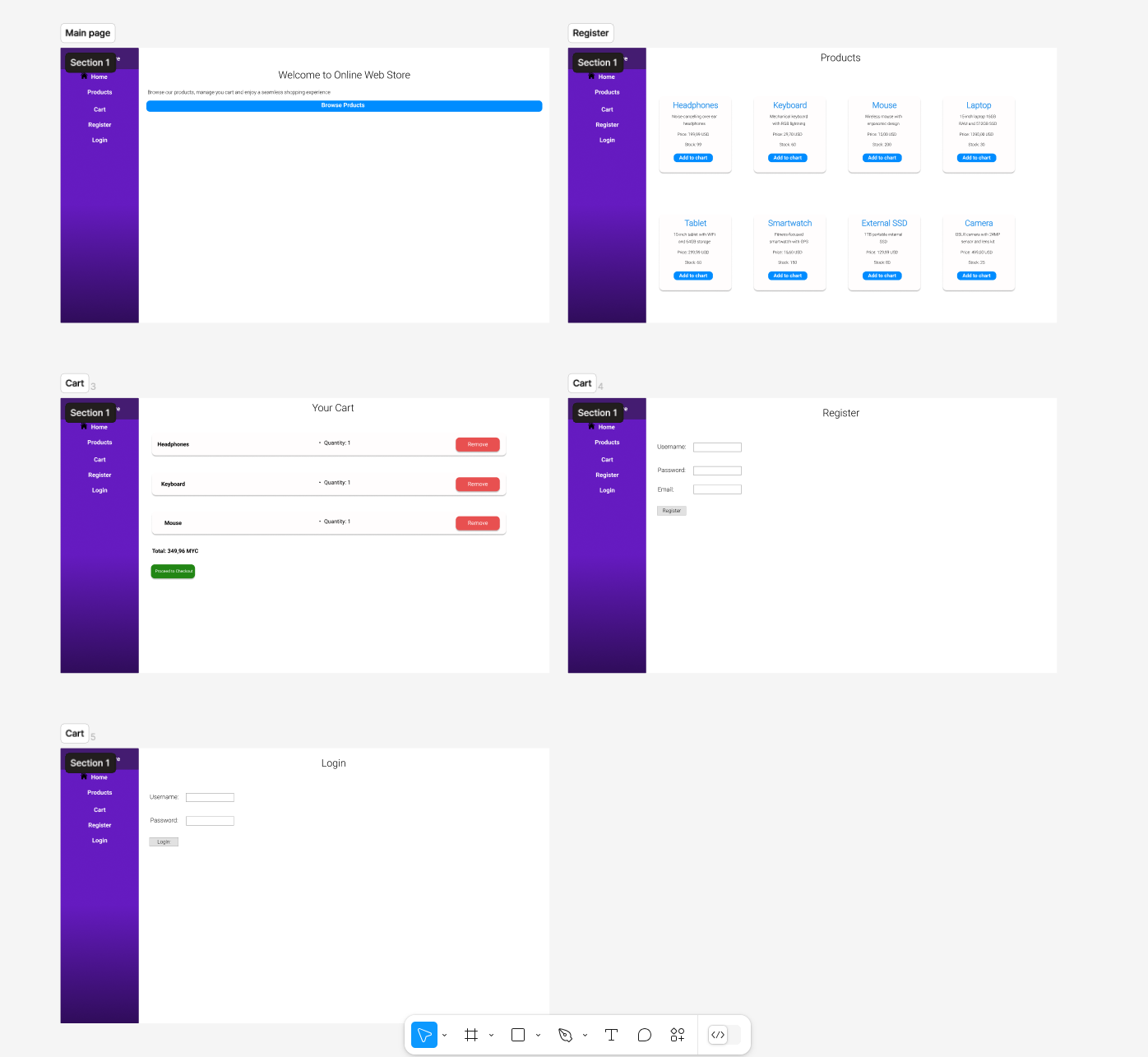
Frontend:

Blazor Web App

Version control system on GitHub

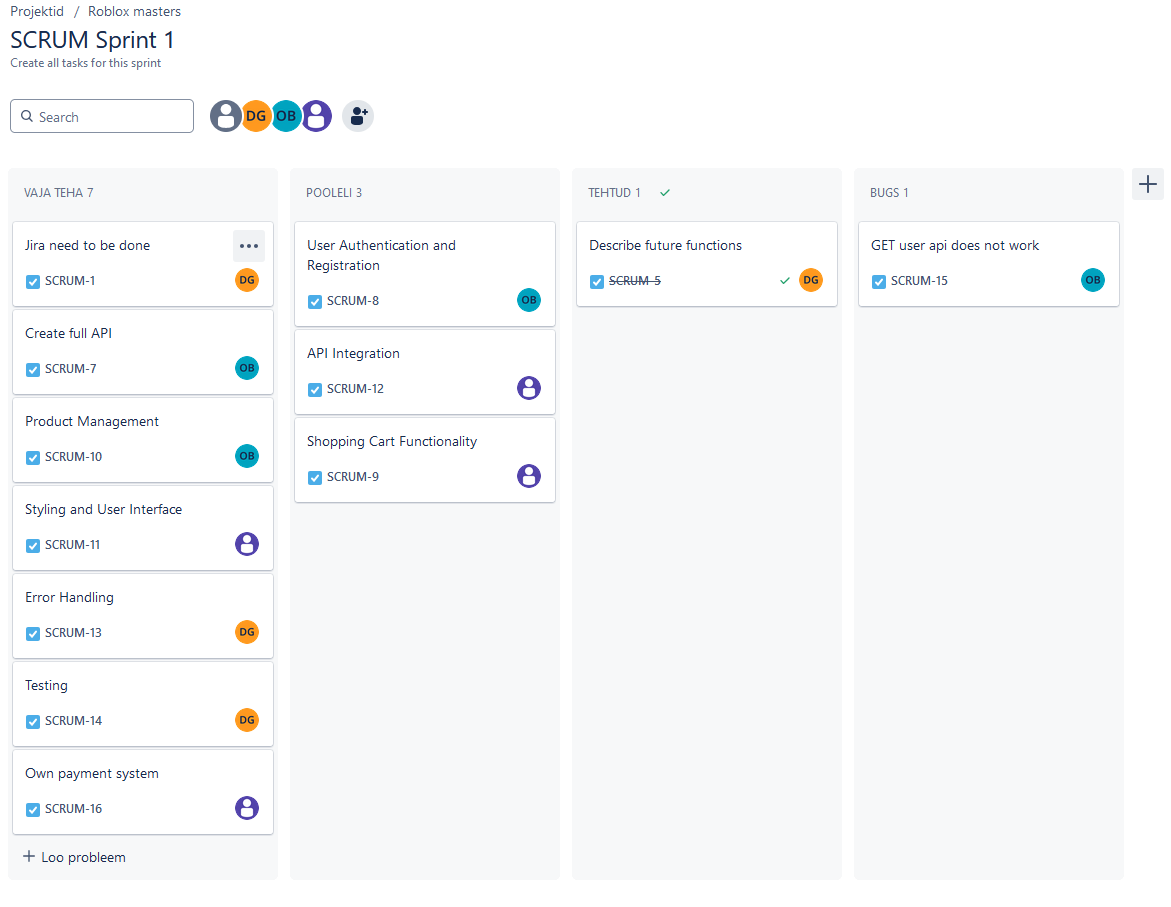
Restful API

## Planning and design

I think we will have the blazer style as the main one, which is internal. And we will do the styles ourselves in figma.

## Planning and prioritization

In Jiro we will prescribe all tasks, roles and scrum meetings.



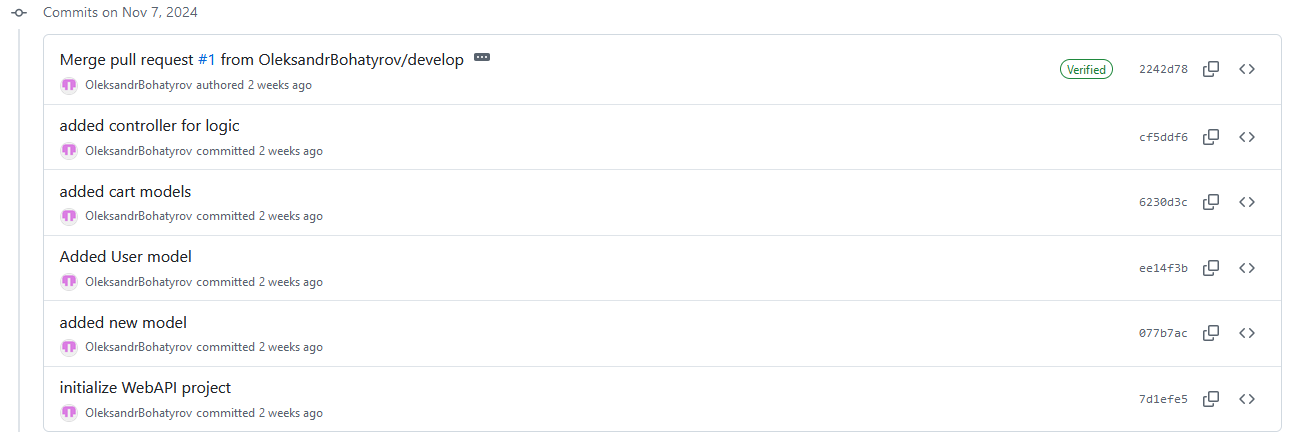
## Development

We'll write the code on GitHub.

Commits will be made on a regular basis.

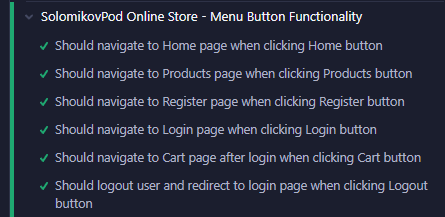
We're going to make the right commits.

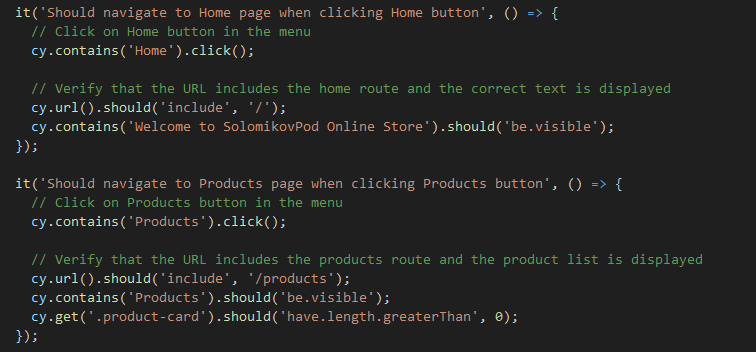
We write the code clean and clear, and write clear comments to the code





## Testing





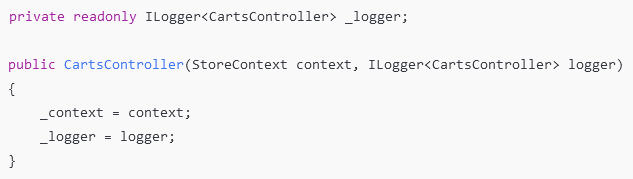
## Code-review

 **Repeated Logic:**

* Logic for working with the cart is duplicated in multiple places (CartsController, PaymentController). For example, loading the cart with its items (Include + ThenInclude). This logic should be extracted into a common service or private method.

 **Logging:**

* There is no logging for error handling. Use ILogger to track potential issues:



 **Security Issues:**

* **Registration and Login:** Passwords are passed without encryption. Use libraries such as Identity or BCrypt for storing passwords instead of implementing it manually.
* **User Permissions:** There is no validation to check if the current user has access to the cart or other resources.

 **Error Handling:**

* There is no error handling for exceptions (e.g., DbUpdateConcurrencyException is only handled in ProductsController). This could cause the application to crash if an unexpected error occurs.

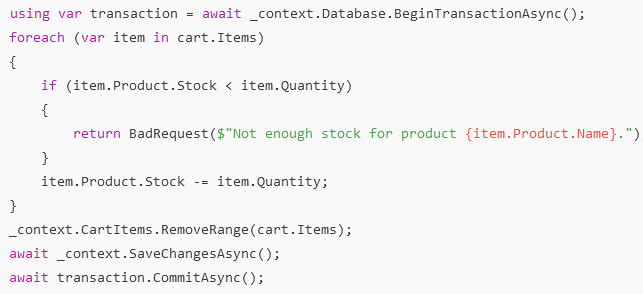
## **CartsController**

### **Positive Aspects:**

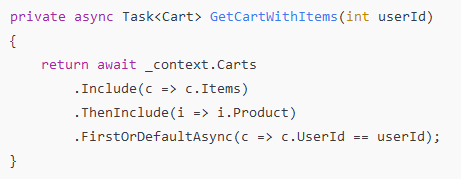
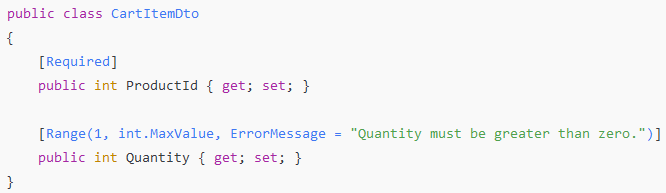
1. **Handling Nested Entities:**
   * Proper use of Include and ThenInclude to load related data (Items and Products).
2. **Cart State Handling:**
   * Good checks for empty cart and stock availability before checking out.

### **Remarks and Recommendations:**

#### **1. Improving** Checkout**:**

* **Issue:**
  + If multiple users check out simultaneously, there could be race conditions related to stock updates.
* **Solution:**
  + Use a transaction:

#### **Improving Code Structure:**

* Repeated code (e.g., retrieving a cart) should be extracted into a private method:**DTO Validation:**
* Add validation annotations for CartItemDto:**PaymentController**

### **Positive Aspects:**

* Stock availability is checked before processing checkout.

### **Remarks and Recommendations:**

#### **1. Unacceptable Messages:**

* Remove lines such as "denis loh suka" and "daniil chmo". These are inappropriate even for test code.

#### **2. Overlapping Logic:**

* The Checkout method duplicates similar logic from CartsController. Create a separate service to centralize cart-related logic.

#### **3. Improving Responses:**

* Instead of string messages, return a structured object:**ProductsController**

### **Positive Aspects:**

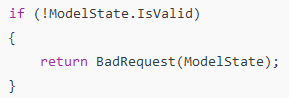
1. **CRUD Operations:**
   * Implements all standard operations (Create, Read, Update, Delete).
2. **Product Existence Validation:**
   * The ProductExists method efficiently checks for the existence of a record.

### **Remarks and Recommendations:**

#### **1. Exception Handling:**

* The PutProduct method includes handling for DbUpdateConcurrencyException, but other methods (PostProduct, DeleteProduct) lack database error handling. Add global exception handling middleware to cover this.

#### **2. Model Validation:**

* Add ModelState validation before saving data:

## **UsersController**

### **Positive Aspects:**

1. **Registration and Login Logic:**
   * Basic checks for existing users and password verification are implemented.
2. **Password Hashing:**
   * Passwords are hashed before storage.

### **Remarks and Recommendations:**

#### **1. Outdated Password Hashing Implementation:**

* Using SHA256 without a salt is insecure. Replace it with a more modern solution, such as:



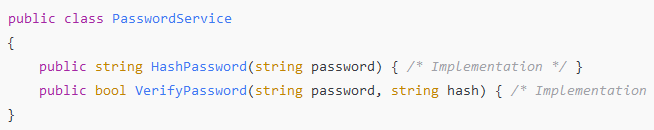
#### **Data Security:**

* Instead of returning the full User object (which might contain sensitive information), return only a token or minimal information:



#### **Improving Code Structure:**

* Move password hashing logic to a separate service:



## **Conclusion**

### **Summary of Recommendations:**

1. Extract repeated logic (e.g., cart operations, password hashing) into separate services.
2. Add logging using ILogger to track errors and events.
3. Use transactions for critical operations like checkout.
4. Remove inappropriate messages from PaymentController.
5. Improve user data security by using modern password hashing libraries and avoiding the return of sensitive data.
6. Add global exception handling middleware for better error management.
7. Standardize response formats across all controllers.

## Documentation

Core Functions:

 Adding cards

 Send products

 Update product

 Delete products

 Registration

 Login

 Password hash

Technology stack

Frontend:

Blazor Web App

JSON (For API)

Backend:

RESTful API for client interaction

ASP.NET Core Web API C#

Entity Framework (Embedded database in c#())

Tools:

Git (for version control)

Project Architecture

General structure

/Github

│

├─── / SolomikovPod # Frontend sources (Blazor Web App)

│ │ ├─── /pages # Pages

│ │ ├ ├─── /services # Services for working with the API

│

├─── /OnlineStoreApi # Backend sources (ASP.NET Core Web API C#)

│ ├──── /controllers # Request processing logic

│ ├──── /models # Database schemas

Data models:

Cart

 Id int

 UserId int

 List<CarItem> Items

CartItem

 Id int

 ProductId int

 Quantity int

 CartId int

Product

 Id int

 Name string

 Description string

 Price decimal

 Stock int

 Currency string

User

 Id int

 Username string

 PasswordHash string

 Email decimal

## Testing on the production environment (Pre-release)

Pre-release testing will be performed in a production-like environment to ensure quality and performance.

## Release

Release date 25.11.2024

At this time we will be handing in our work.