# Back-end .NET developer challenge

## Introduction

Your task would be to extend the existing ASP.NET MVC5 application with a new feature. You have to study the code, add new parts, and possibly modify some existing parts.

You need Visual Studio (2019) and .NET 4.7 to build the solution.

Be aware that some parts of the code are built according to the best practices and some are not – moreover, there are some deliberate bugs. You will get extra points by pinpointing the issues.

## Pre-built part

There is a web application with several users which could log in.

Each user has an amount of gold coins, which is shown somewhere in the top bar.

E.g. user1 has 120 gold, user2 has 30 gold etc.

There is a stock of items to offer, with a price in gold coins and an available amount.

E.g. 10 swords costing 8 each and 2 axes costing 12 each and so on.

For the simplicity of the setup, the "persistent" storage is in-memory (i.e. some static collections, not a SQL database).

The front-end part of this new feature has already been supplied and can be found under the "Buy" navigation link in the header.

The dialog shows the following elements:

[image] [name] [-] [amount to buy input] [+] [price]

......

total: [xxx]

[buy] [cancel]

A user can see available items and their price, can type in an amount to buy (integer value), or increment/decrement it with buttons + and -.

User can click buy to submit the configured order

The contract between the front-end and back-end is defined via a mock implementation and can be seen described in the OrderController.

## Part to be built

### Functional requirements

Make it possible for the user to buy items from the stock via the "Order" dialog (see the "Buy" navigation link).

Make sure constraints with regard to number of items available and user balance are not violated, even if several concurrent users are acting, and prevent other potential abuse from the users. Make sure this feedback is available to the user.

When an order is successfully processed, the price of the order is deducted from the user's balance and the stock reflects the reduced number of available items.

Add several test cases to "Back-end challenge tests" for the new functionality (i.e. the processing of an Order)

## Implementation requirements

Existing features should continue to function.

All decisions not described here are up to you.

Good luck!

## Front-end note

You should not need to make any changes to the frontend, however if you do encounter a problem and you are capable or fixing it yourself, you can do the following to build the front-end:

1. Navigate to "\Back-end challenge\Content\scripts"
2. Run "npm install"
3. Run "npm run build"

This will compile the typescript files into build.js

**Version Notes**

* I added Key to id fields to make it a unique filed.
* I changed id values for the sample data in the ItemRepository.cs to 1,2,3,4. Because we need unique fields. Can not be 1,1,1,1.
* I limited the random id generation for the items.
* For a persistent database, I installed entity framework
* Class diagram added for a clear view