Product Updating system
Appendix

Erhvervsakademiet Lillebælt
OEADM16EC Computer Science
3rd Semester – Group E

Group participants:

Bogdan-Iulian Vasile	bogd0268@edu.eal.dk
David de Lima Fraga Alves	davi7816@edu.eal.dk
Elitsa Miroslavova Marinovska	elit0452@edu.eal.dk
Luca Treamamunno	luca9910@edu.eal.dk

Contents

A) Quality assurance

I) Quality plan

Item	How	Who	When
Rich picture	Criteria: Make sure it shows: various systems and the way they interact together different perspectives and the concerns that stakeholders have everything that is relevant in our system (but not detailed, details belongs to design phase)	The team	During the production process of every item
Workflow diagram	Criteria: Ensure it depicts all the actions in the process Ensure we evaluate all the various conditions Ensure to have a start event which initiates all the following activities Ensure every workflow is reaching and outcome point (end event)		
Goals list	Criteria: Make sure they are written in a way so everyone from the team has the same understanding Make sure we include them all and don't miss some major user requirements		
Scenarios	Criteria: Every scenario should have a title Ensure it describes interaction between user and a system from a general point of view Write it as a short-story (bullet points)		
Use cases	Criteria: Include: use case title description/goal actor/user preconditions (the things that must have already happened in the system) main success scenario (what will usually happen, described as a series of steps) alternate paths or extensions(variations on the above cases)		

	postconditions (what the system will have done by the end of the steps)		
Use case diagram	Criteria: Ensure it provides a graphical overview of the system goals Ensure it includes an actors representing any person using the system or an external system Ensure to have association between actor and use case to indicate that the actor will interact with the system to achieve their goal		
Object model	Criteria: Ensure it represents the relationship between instances of objects from the conceptual classes Ensure that you are able to see the multiplicity between the conceptual classes from it Ensure that the objects have example data in them that could be an actual object in the domain		
Domain model	Criteria: Ensure to list the candidate real-world conceptual classes in a box that has the name domain in the middle Ensure to add the associations necessary to record relationships Ensure to add the attributes necessary to fulfill the information requirements Ensure to uses language and terms that customers can understand, or technical terms from the domain		

System Sequence Diagram	Criteria: Ensure to determine the actors that directly interact with the software system Ensure that the diagram is easy to be followed Ensure it meets specific UML standards Ensure it explains every act between the actor and system, input and output Ensure it depicts steps from the Use Cases		
Design Class Diagram	Criteria: Ensure that it contains software classes together with the conceptual classes The connection between 2 software classes needs specify the type of relationship Ensure that public methods or variables has a '+' sign Ensure that private methods has a '-' sign Ensure properties with public get and public set has '+/+' sign Ensure properties with public get and private set has '+/-' sign		
Sequence diagram	Criteria: Ensure it shows an object sending a message to another object Ensure that you draw a line to the receiving object with a solid arrowhead (if a synchronous call operation) or with a stick arrowhead (if an asynchronous signal) Ensure that the message/method name is placed above the arrowed line Ensure that alternatives, loops, optional elements are drawn using a frame. The word "alt", "loop", "opt" is placed inside the frame's name box		

Kanban board	Criteria: Ensure it manages both the Development and Testing phases Ensure it has cards with different colours Ensure to have a general title and a description with additional details Ensure that you use Green (Initial) only when the class isn't created yet Ensure that you use Blue (Increment) only when new class functionalities are to be implemented Ensure that you use Yellow (Refactor) only when you are Updating a class Ensure that you use Orange (Test) only when you are preparing tests Ensure that you use Red (Bugfix) only when there are some problems that need fixing Ensure to update the board as soon as a task in a specific stage is done		
Test cases	Criteria: Ensure that the Label before the table is referring to a class in the source code Ensure to include the reference number of the test to the table column so it can b easily found in the code Ensure to put the method name and the action that need to be performed in the Test scenarios section Ensure to mention the state of the system before the test has been executed, in the Pre-condition section Ensure to mention the state of the system after the test has been executed, in the Postcondition section		
Experiments	Criteria: Ensure that they are done in an early design stage Ensure a record of an experiment with its main details of it is being done		

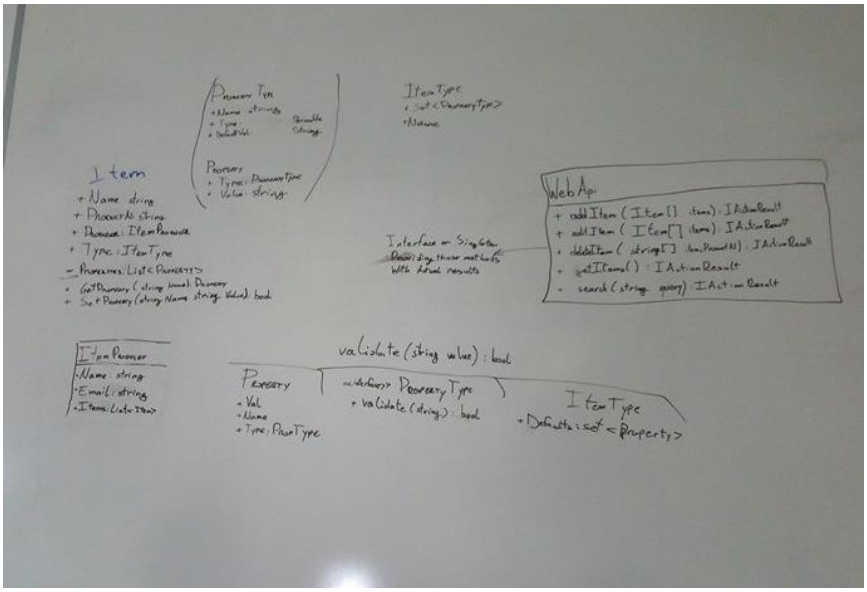
Code	Criteria: Ensure it follows SOLID and CRUD principles Ensure it translates precisely the DCD into code Ensure if any modifications are done, all related artefacts need to be changed in the way they were produced Ensure to include code comments whenever something needs some additional specifications Ensure that you leave /*TODO*/ when there is a segment from the code that has to be either created or refactored. Ensure to have summaries of a methods the team agrees as more complex than the others Ensure to make commits frequently with a meaningful message Ensure that when there are merge conflicts the contributors need to revise their work until they resolve the commit problems		

B) Traceability

I) Decision log

Base class diagram

27/11/2017



In the property class instead of having the value as string and type as IPropertyType, the property can be a generic element.

Discarded because:

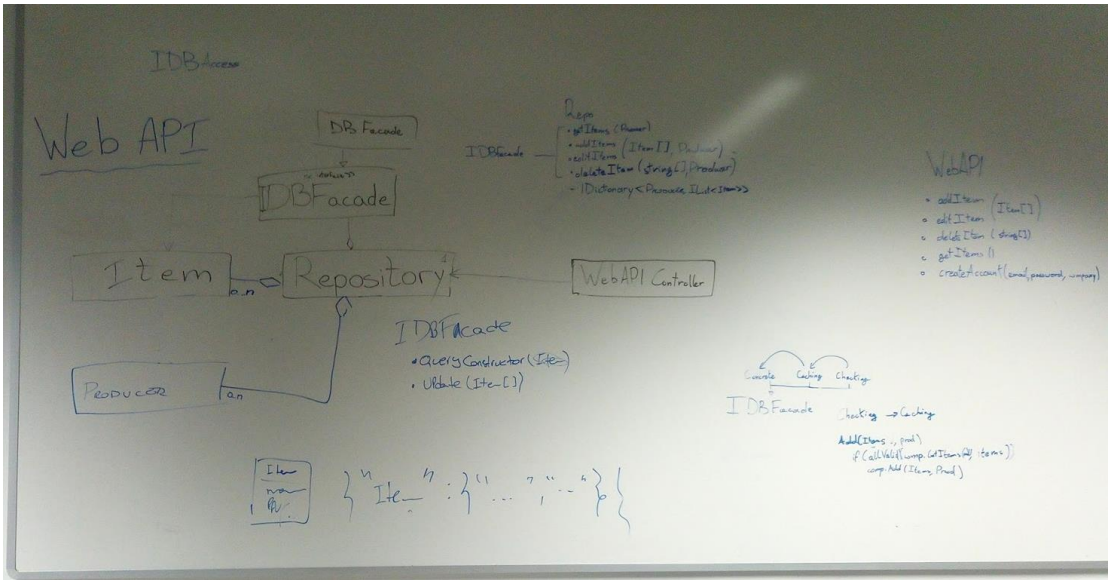
- we would need to read the property value from a string (needs validation anyway);
- harder to implement;
- having the actual object would not provide any actual advantage;

Use of classes Property and PropertyType.
PropertyType has the name, type and default value of the property. Property has a value and a PropertyType. ItemType has a set of PropertyType.
Discarded because:

- the new solution allows to specify default values in the ItemType;
- the new solution is more comprehensible;
- the new solution allows better reusability (validation of value in IPropertyType);

Design class diagram

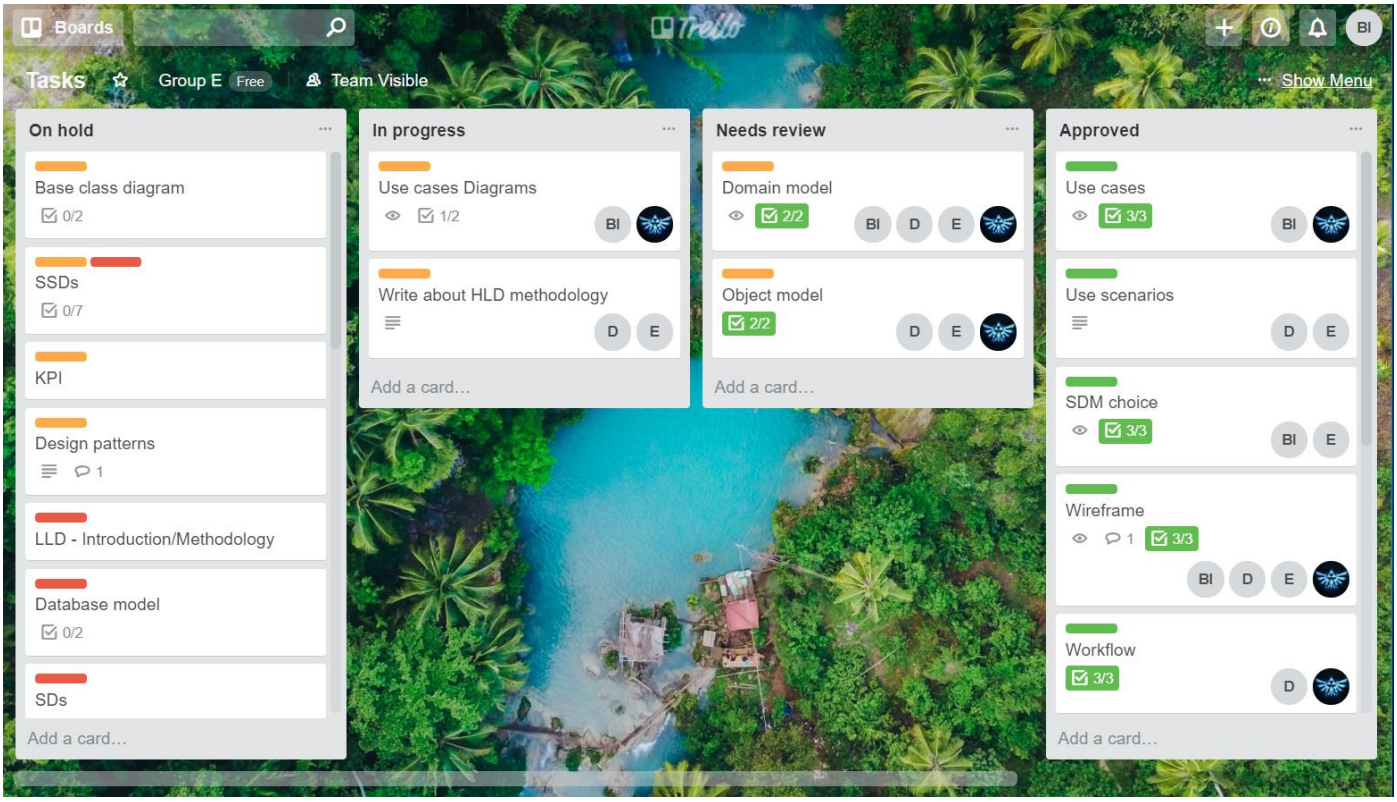
30/11/2017



- Repository is discarded in favor of using Decorator pattern
- Producer class is discarded because we are only interested in its email.
- This version doesn't specify where the checks on input (if any) are done

C) Tools and methods

I) Task board



D) Requirements gathering

I) Prioritization

M for Must have

Requirements labeled as *Must have* are critical to the product delivery in order for it to be considered a success. If even one *Must have* requirement is not included, the project delivery should be considered a failure. A requirement can be downgraded from *Must have*, by the product owner or relevant stakeholders, for instance other requirements or newly introduced ones are deemed more important.

S for Should have

Requirements labeled as *Should have* are important but not completely necessary for the final product to function. While '**Should have**' requirements can be as important as '**Must have**', they are often not as critical or there may be another way to accomplish the requirement, so it can be held back to later in the process.

C for Could have

Requirements labeled as *Could have* are desirable but not necessary, and could improve user experience or customer satisfaction for little development cost. These will typically be included if time and resources permit.

W for Would have

Requirements labeled as *Would have* should be agreed by stakeholders as the least-critical, and with the lowest-payback. *Would have* requirements are likely not put in the scheduled for delivery. "**Would have"** requirements are either dropped or reconsidered for inclusion in last project stages, as it's considered lowest priority and would only be included if there is a surplus of time.

E) Startup

I) Use case list

MVC Application

UC1.1 - Login - M

→ A user wants to insert his/her credentials into the system in order to have access to all the possible actions the system provides to him/her.

UC1.2 - Add item - M

→ A user wants to insert a new item and its details into the system.

UC1.3 - Update item - M

→ A user wants to change information about an item already present in the system.

UC1.4 - Delete item - M

→ A user wants to remove the information of an item stored in the system.

UC1.5 - Manual logout - S

→ A user wants to be logged out from the system, so he/she will have to insert credentials again (can even be different credentials).

UC1.6 - Automatic logout - C

→ A user has been inactive for too long and gets logged out automatically by the system.

UC1.7 - Search - C

→ A user wants to search for an item already present in the system.

WEB API

UC2.1 - Login - M

→ A user wants to start using the Web API actions.

UC2.2 - Add items - M

→ A user wants to add one or more new items.

UC2.3 - Update items - M
→ A user wants to update information about one or more existing items.

UC2.4 - Delete items - M
→ A user wants to remove one or more existing items from the system.

UC2.5 - Get items - M (it is used by the MVC)
→ A user wants to get the items stored in the system.

UC2.6 - Create account - S (you can always insert an account manually in the Database)
→ An admin wants to register a new user.

II) Use case descriptions

Remarks:
Initially UC1.4, "Delete item", was a Should, but after a meeting with the client it was decided that it had be a Must.
Prioritizing the Web API was done after considering the priority of MVC, as MVC is built on top of Web API, the functions that allow some MVC use cases should have the same (or higher) level of priority.

MVC Application

UC1.1 - Login
Short description
A user wants to log into the system.
Primary actors
• ***Alice***
Secondary actors
• *None*
Overview
Alice wants to insert her credentials into the system in order to have access to all the possible actions the system provides to her.
Precondition

- Alice is not logged in.

Primary Scenario

Actor	System
1.Alice inserts her username and password 2.Alice clicks on the login button	3.The system checks the credentials 4.The credentials are valid 5.The system logs in Alice 6.The system redirects Alice to the main page

Alternative scenario

Actor	System
6a.Alice receives the error message	4a.The credentials are not valid 5a.The system sends an error message to Alice

Post-conditions

- Alice is logged in

UC1.2 - Add item

Short description

A user wants to add a new item.

Primary actors

- ***Alice***

Secondary actors

- *None*

Overview

Alice wants to insert a new item and its details into the system.

Precondition

- Alice is logged in.

Primary Scenario

Actor	System
1.Alice clicks on the "Create new" button (or equivalent)	
	2.The system redirects Alice to the appropriate page
3.Alice uses the page to insert the item information	
4.Alice clicks on the "Confirm" button	
	5.The system checks the information inserted by Alice
	6.The information is valid
	7.The system asks Alice for confirmation
8.Alice accept the confirmation	

Alternative scenario 1

Actor	System
	6a.The information is not valid
	7a.The system sends an error message to Alice
8a.Alice receives the error message.	
9a.Alice can change the inserted values	
10a.Alice confirms again (back to point 5 of primary scenario)	

Alternative scenario 2

Actor	System

8b.Alice denies the confirmation 9b.Alice can change the inserted values 10b.Alice confirms again (back to point 5 of primary scenario)	

Post-conditions

- The system creates the new item and stores it

UC1.3 - Update item

Short description

A user wants to update information about an existing item.

Primary actors

- ***Alice***

Secondary actors

- *None*

Overview

Alice wants to change information about an item already present in the system.

Precondition

- Alice is logged in.
- At least an item belonging to Alice exists in the system.

Primary Scenario

Actor	System

1.Alice clicks on the "Edit" button (or equivalent) related to a specific item	
	2.The system redirects Alice to the appropriate page, containing the current information about the item
3.Alice uses the page to modify the item information	
4.Alice clicks on the "Confirm" button	
	5.The system checks the information inserted by Alice
	6.The information is valid
	7.The system asks Alice for confirmation
8.Alice accepts the confirmation	
Alternative scenario 1	
Actor	**System**
	6a.The information is not valid
	7a.The system sends an error message to Alice
8a.Alice receives the error message	
9a.Alice can change the inserted values	
10a.Alice confirms again (back to point 5 of primary scenario)	
Alternative scenario 2	
Actor	**System**
8b.Alice denies the confirmation	
9b.Alice can change the inserted values	
10b.Alice confirms again (back to point 5 of primary scenario)	
Post-conditions	
• The system updates the item information	

UC1.4 - Delete item
Short description

A user wants to remove an existing item from the system.

Primary actors

- ***Alice***

Secondary actors

- *None*

Overview

Alice wants to remove the information of an item stored in the system.

Precondition

- Alice is logged in.
- At least an item belonging to Alice exists in the system.

Primary Scenario

Actor	System
1.Alice clicks on the "Delete" button (or equivalent) related to a specific item	
	2.The system asks Alice for confirmation
3.Alice accepts the confirmation	

Alternative Scenario

Actor	System
3a.Alice denies the confirmation	

Post-conditions

- The system removes the item

UC1.5 - Manual logout

Short description

A user wants to logout from the system.

Primary actors

- ***Alice***

Secondary actors

- *None*

Overview

Alice wants to be logged out from the system, so she will have to insert her credentials again.

Precondition

- Alice is logged in.

Primary Scenario

Actor	System
1.Alice clicks on the "Logout" (or equivalent) button	2.The system logs Alice out 3.The system redirects Alice to the login page

Post-conditions

- Alice is logged out

UC1.6 - Automatic logout

Short description

A user gets logged out automatically.

Primary actors

- ***System***

Secondary actors

- ***Alice***

Overview

A user has been inactive for too long and gets logged out automatically by the system.

Precondition

- Alice is logged in.

Primary Scenario

System

1.The system detects that Alice has been inactive for a predefined amount of time
2.The system logs Alice out (next time she wants to perform an action she will have to login again)

Post-conditions

- Alice is logged out

UC1.7 - Search

Short description

A user wants to search for an item that is stored in the system.

Primary actors

- ***Alice***

Secondary actors

- *None*

Overview

Alice wants to search for an item by its name or its serial number.

Precondition

- Alice is logged in.

Primary Scenario

Actor	System
1.Alice inserts a value on the search bar 2.Alice requests the search	3.The system redirects Alice to the result page

Post-conditions

- *None*

Web API

UC2.1 - Login

Short description

A user wants to start using the Web API actions.

Primary actors

- ***Bob***

Secondary actors

- *None*

Overview

Bob wants to use the Web API, but he needs an authorization token.

Precondition

- *None*

Primary Scenario

Actor	System
1.Bob prepares a request with his username and password 2.Bob sends the request 7.Bob receives the token	 3.The system checks the credentials 4.The credentials are valid 5.The system creates a new token 6.The system sends the token to Bob

Alternative scenario

Actor	System
	4a.The credentials are not valid 5a.The system sends an error message to Bob

Post-conditions

- The new token is stored in the system

UC2.2 - Add items

Short description

A user wants to add one or more new items.

Primary actors

- ***Bob***

Secondary actors

- *None*

Overview

Bob wants to insert one or more new items and their details into the system.

Precondition

- Bob owns a token.

Primary Scenario

Actor	System
1.Bob prepares the request with the token and the items' details 2.Bob sends the request	3.The system checks the items provided by Bob 4.The information about the items is all valid 5.The system performs the action 6.The system sends a "200 OK" message to Bob

Alternative scenario 1

Actor	System
	4a.One or more items are invalid 5a.The system prepares a "400 Bad Request" message, containing the error details 6a.The system sends the message

Post-conditions

- The system creates the new items and stores them

UC2.3 - Update items

Short description

A user wants to update information about one or more existing items.

Primary actors

- ***Bob***

Secondary actors

- *None*

Overview

Bob wants to change information about one or more items already present in the system.

Precondition

- Bob owns a token.

Primary Scenario

Actor	System
1.Bob prepares a request with items' information 2.Bob sends the request	3.The system checks the items provided by Bob 4.The items are all valid 5.The system performs the action 6.The system sends a "200 OK" message to Bob

Alternative scenario 1

Actor	System

	4a.At least one item is not valid 5a.The system prepares a "400 Bad Request" message, containing the error details 6a.The system sends the message

Post-conditions

- The system updates the item(s)' information

UC2.4 - Delete items

Short description

A user wants to remove one or more existing items from the system.

Primary actors

- ***Bob***

Secondary actors

- *None*

Overview

Bob wants to remove the information about one or more items stored in the system.

Precondition

- Bob owns a token.

Primary Scenario

Actor	System
1.Bob prepares a request with the identifier of the items he wants to remove 2.Bob sends the requests	3.The system checks if the request is valid 4.The request is valid

	5.The system performs the action 6.The system sends a "200 OK" message to Bob
Alternative Scenario 1	
Actor	**System**
	4a.The request is not valid 5a.The system prepares a "400 Bad" Request message, containing the error details 6a.The system sends the message
Post-conditions	
• The system removes the item(s)	

UC2.5 - Get items
Short description
A user wants to get the items stored in the system.
Primary actors
• ***Bob***
Secondary actors
• *None*
Overview
Bob wants to get all the information about the items he owns, stored in the system.
Precondition
• Bob owns a token.

Primary Scenario

Actor	System
1.Bob prepares the request 2.Bob sends the request 4.Bob receives the list of items	3.The system prepares a "200 OK" message, containing the information about the items

Post-conditions

- *None*

UC2.6 - Create account

Short description

An admin wants to register a new user.

Primary actors

- ***Admin***

Secondary actors

- *None*

Overview

Admin needs to register a new user that will be able to use the system.

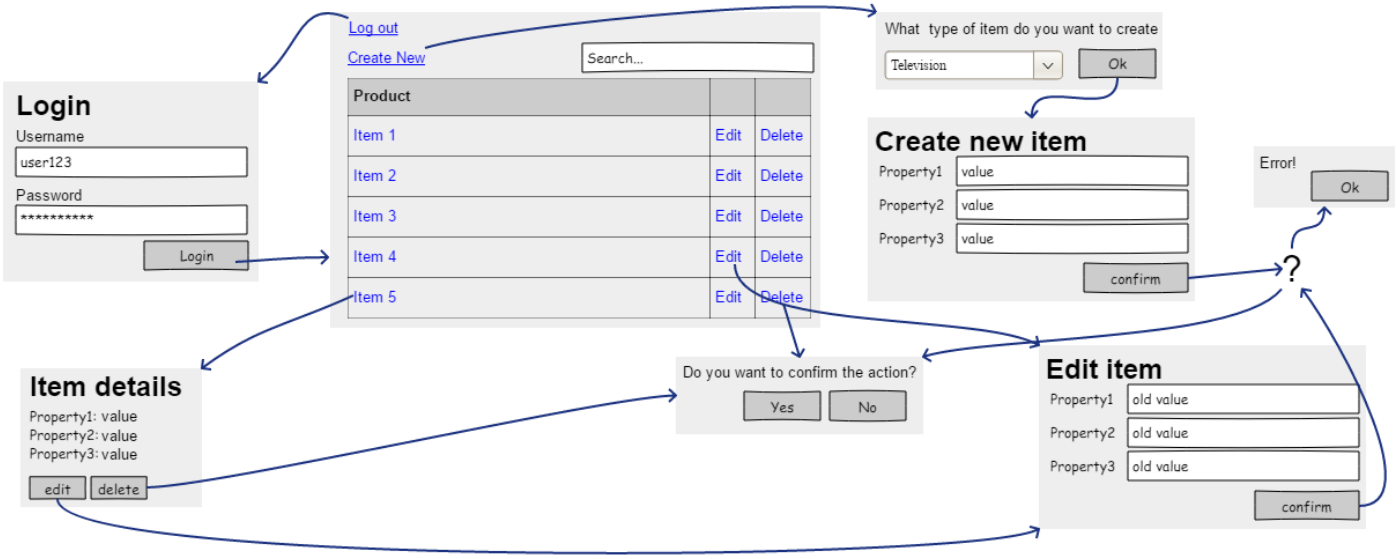
Precondition

- Admin owns a (admin) token.

Primary Scenario

Actor	System
1.Admin prepares a request with the details of the new account 2.Admin sends the request	3.The system checks if the request is valid 4.The request is valid 5.The system performs the action 6.The system sends a "200 OK" message to the admin
Alternative Scenario 1	
Actor	**System**
	4a.The request is not valid 5a.The system prepares a "400 Bad Request" message, containing the error details 6a.The system sends the message
Post-conditions	
• The new user is stored into the system	

III) Wireframe



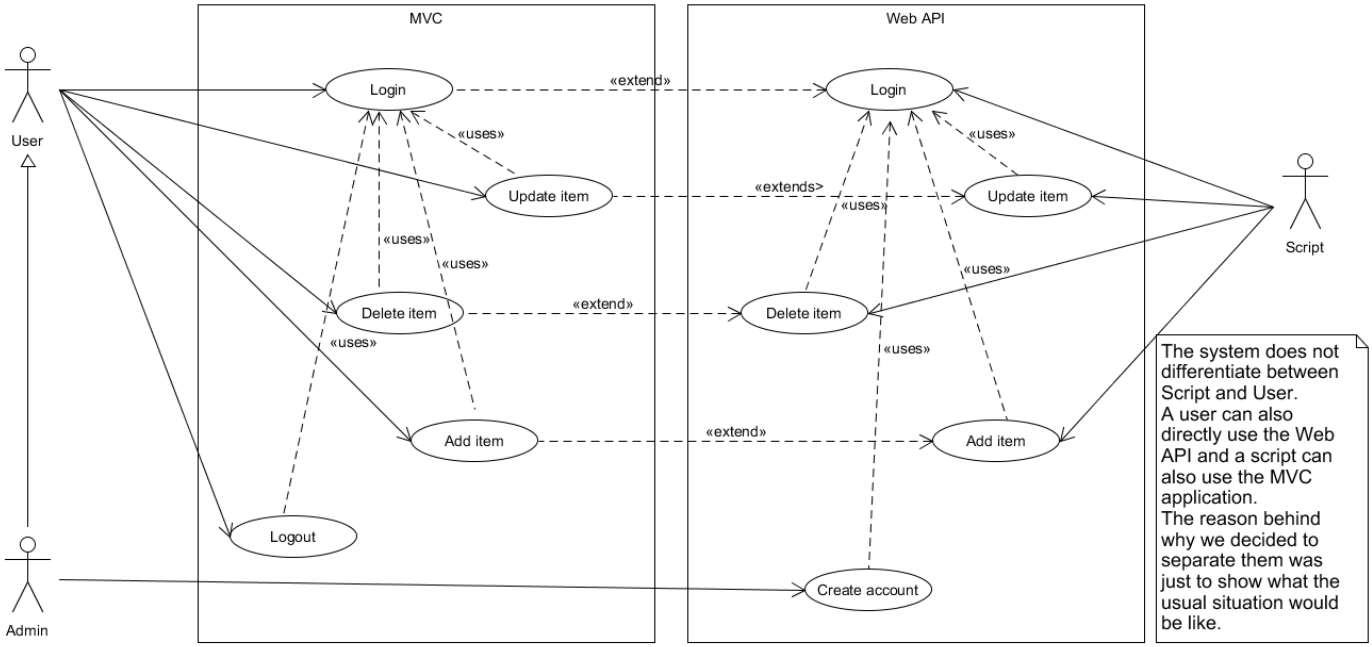
IV) Rich picture explanation

Our system is created in order to help the reseller, which in the current situation has to manually update a database of items from different suppliers. The idea is to develop a system that allows each supplier to update the (reseller's) database, whenever they have new products, or make modifications on existing ones.

Our system will consist of a server, communicating with the reseller's database, and will run a Web API which will provide the functionality of the system. Additionally, on top of it a UI (through web pages) will be created in order to allow users to work with the system in an easy way. The supplier may then use either the UI, i.e. having an employee manually inserting data in the system, or directly use the Web API, for example, through a script.

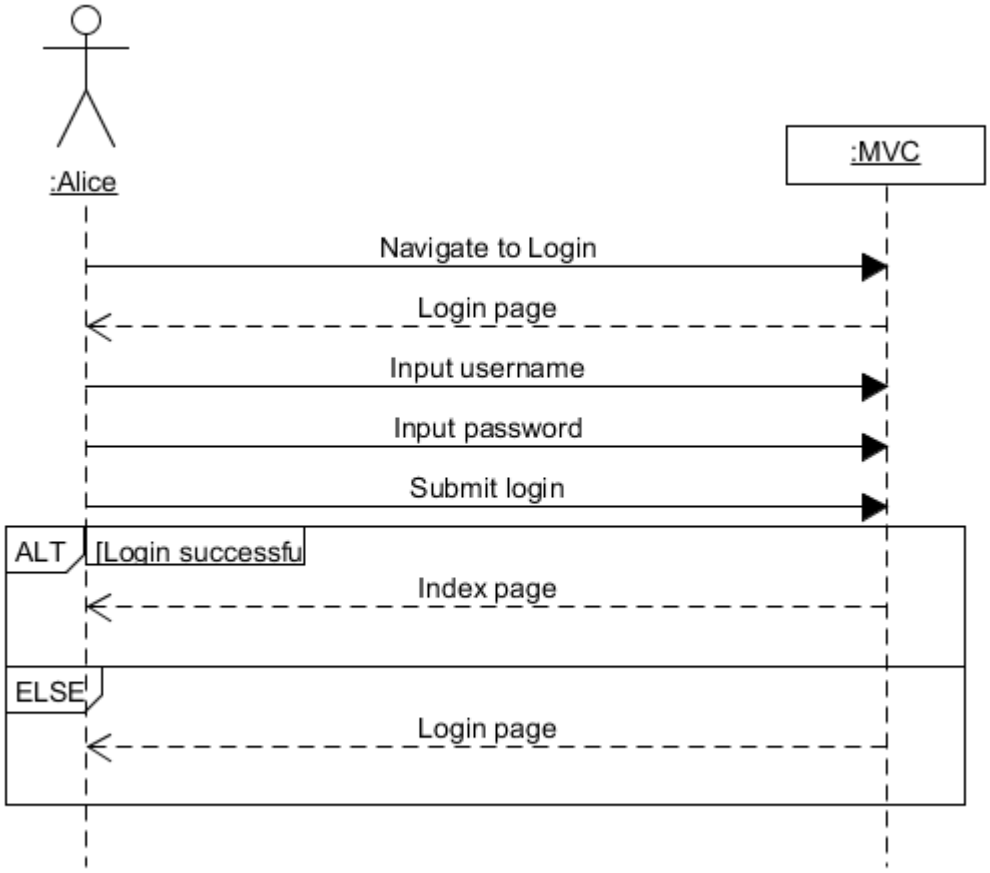
F) High-Level design

I) Use case diagram

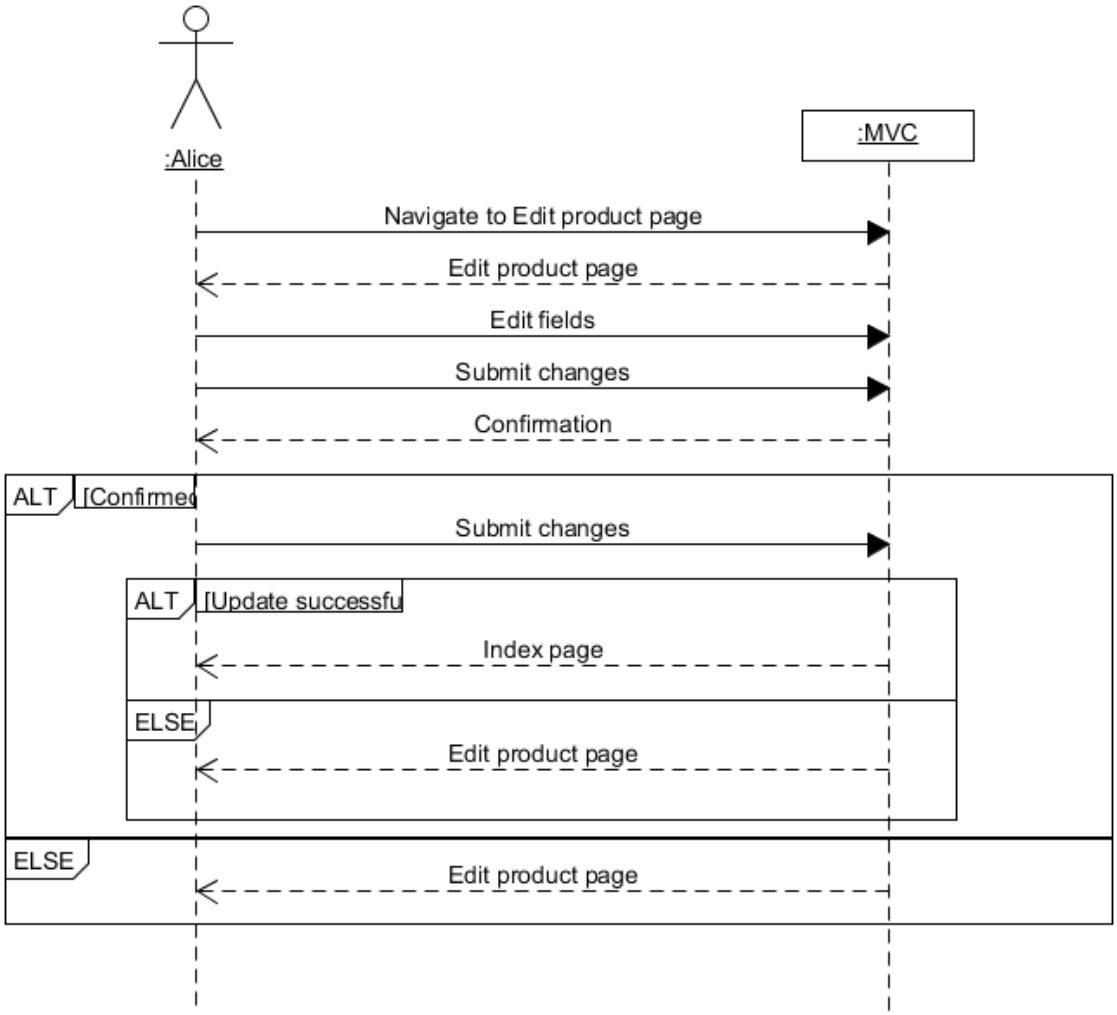


II) System sequence diagrams

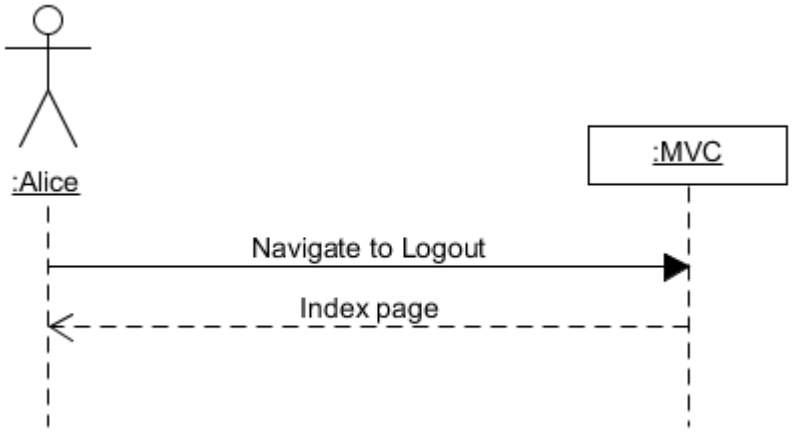
- MVC Login



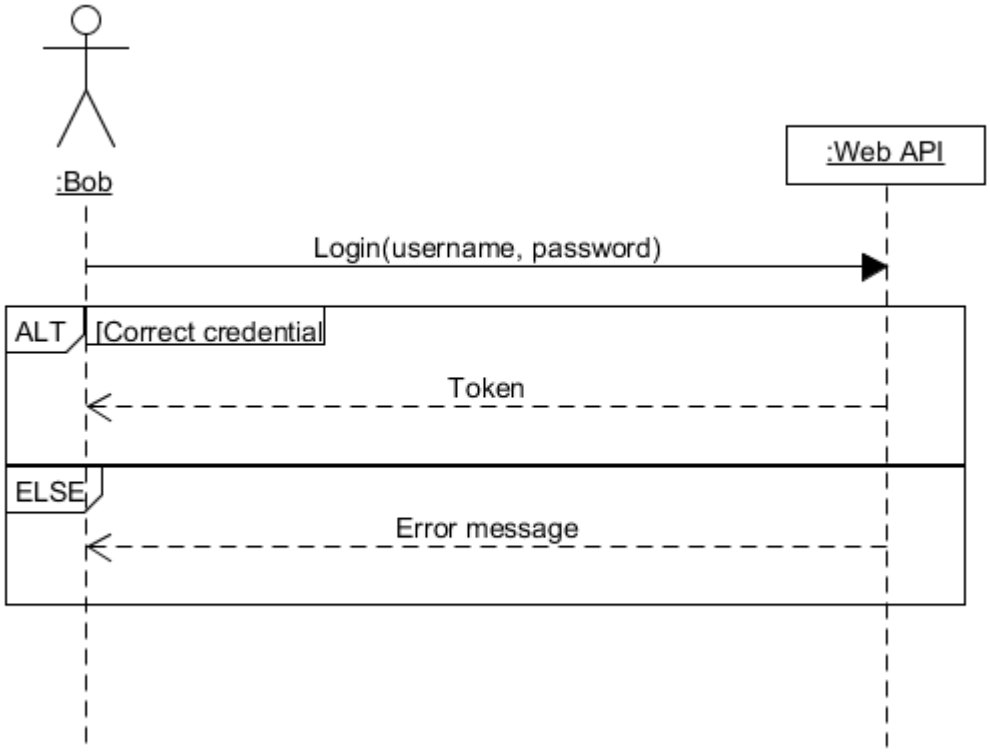
- MVC Update item



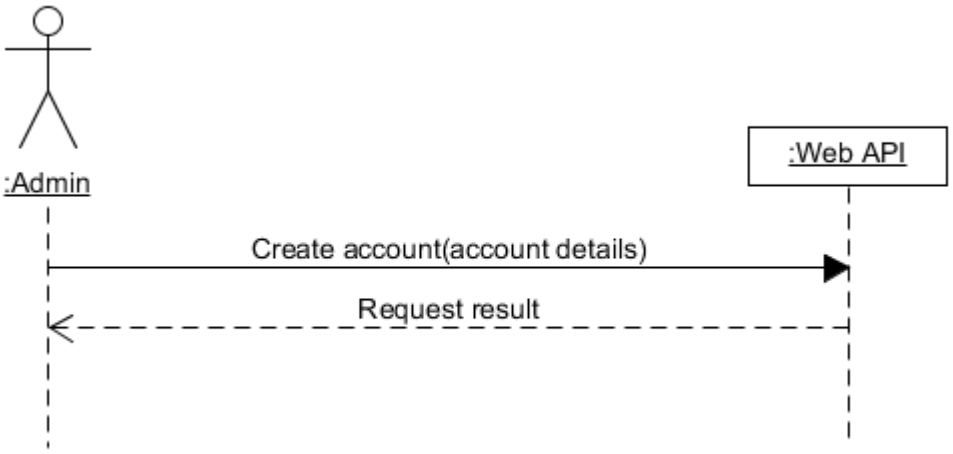
- MVC Logout



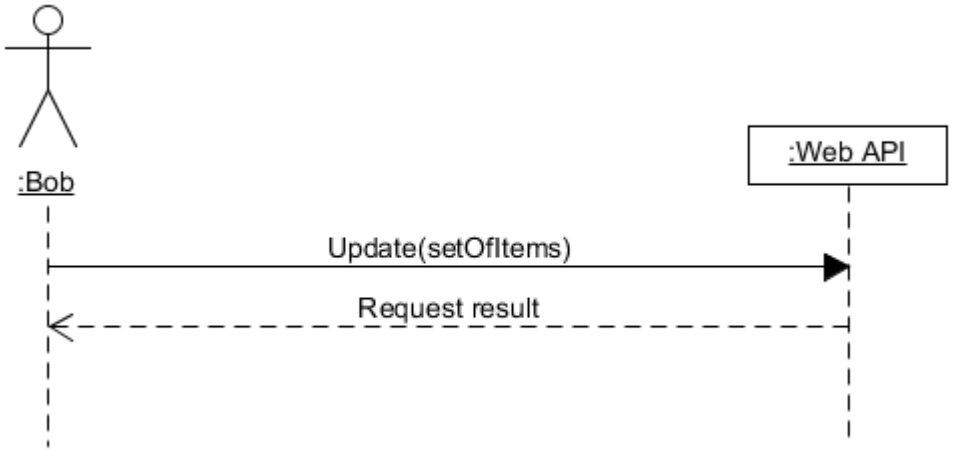
- Web API Login



- Web API Create



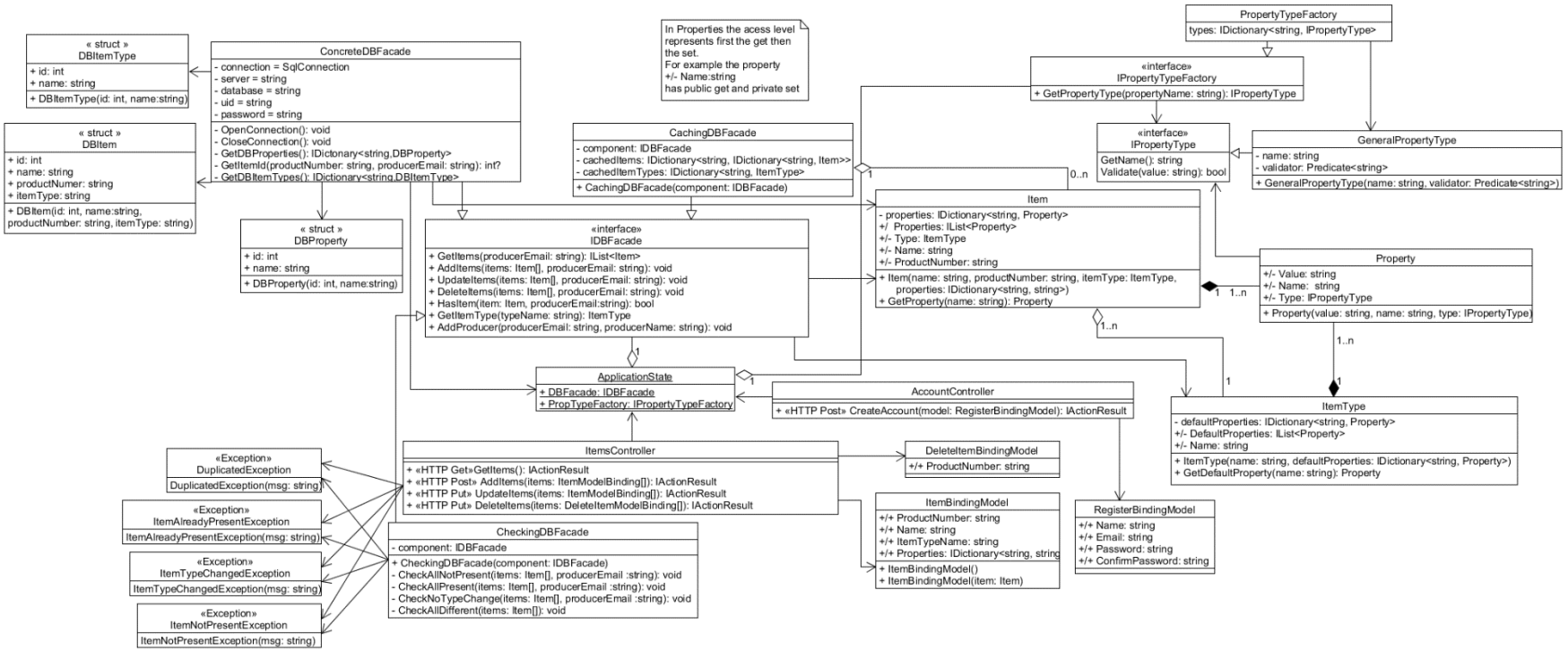
- Web API Update



G) Low-Level design

I) Design class diagram

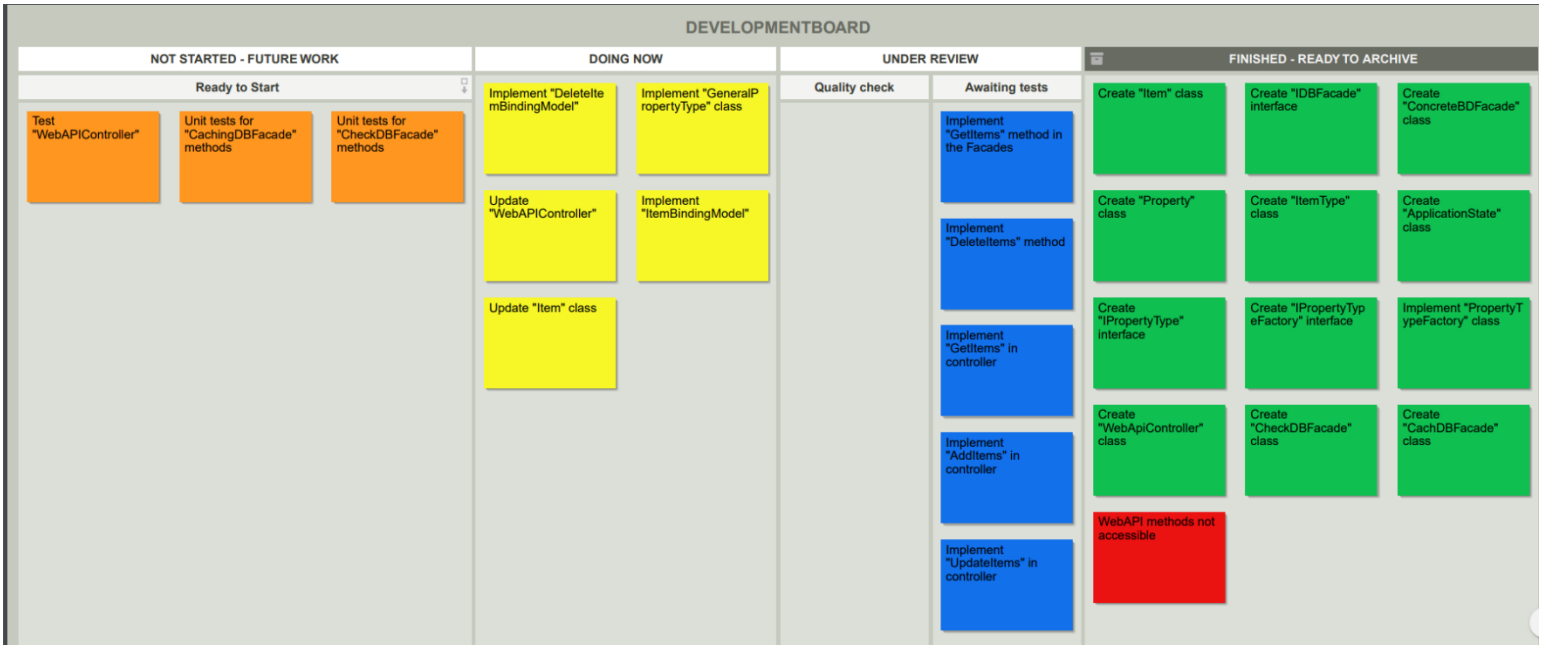
DESIGN CLASS DIAGRAM



H) Development and testing

I) Kanban board

Initial (Green) - initial phase of implementation for a class; could be just creating a skeleton of the class, or complete implementation if it is a very simple class (only data, no behaviour).
Increment (Blue) - for a task about incrementing one or more classes functionality, for example implementing methods that were not implemented yet.
Refactor (Yellow) - for a task about refactoring one or more classes, to improve the code quality.
Test (Orange) - for a task about testing implemented classes.
Bugfix (Red) - for a task about fixing a problem that was found in one or more classes.



II) Test cases

CachingDBFacade

Ref №	Test scenarios	Test Case	Pre-condition	Expected result

#1	Check GetItems Method	Check if the method returns an empty list if the provided user is not in the system	The specified user is not in the system	The resulting list should be empty
#2	Check GetItems Method	Check if the user is in the system and the method returns a list containing all and only the items for that specific user	The specified user is in the system	The resulting list contains all the items for the aforementioned user
#3	Check AddItems Method	Check if the method is adding the items in the cache	The items that are added are not duplicated nor present in the system for the specified user	The list returned by GetItems method contains the newly added items as well as the old ones
#4	Check AddItems Method	Check if the method is adding the items in the cache even if the specified user had no entries	The added items are not duplicated	The list returned by GetItems returns the newly added items
#5	Check DeleteItems Method	Check if the method is removing the items for the specified user from the cache	The aforementioned items for the specified user are present in the system and not duplicated	The list returned by GetItems method does not contain the newly removed items
#6	Check AddItems and DeleteItems Methods	Check if subsequent calls to the methods are working properly in any given situation	At any moment the methods are called, the items passed as arguments are not duplicated. When AddItems method is being called, all the items should already be	After every call we have consistent results

			present for the specified user and when we call the DeleteItems method we pass as arguments items that are already present	
#7	Check UpdateItems Method	Check if the method is updating the values for all and only the specified items	The items passed as arguments are equivalent to already present items in the system for the specified user	The next GetItems method call will return the same amount of items as before and the modified items will have the values updated while the others remain the same
#8	Check HasItem Method	Check if the method is returning "true" if the specified user has equivalent item already present in the system	The user already has an item in the system equivalent to the item passed as a parameter	The method returns "true"
#9	Check HasItem Method	Check if the method is returning "false" if the specified user has no equivalent item already present in the system	The user has no item in the system equivalent to the item passed as a parameter	The method returns "false"
#10	Check GetItemType Method	Check if the method returns the right ItemType for a given ItemType name, if it is	An ItemType with the provided name exists in the system	The returned type is correct

		present in the system		
#11	Check GetItemType Method	Check if the method returns “null” for the given ItemType name if there is no ItemType with such name present in the system	The ItemType passed as a parameter does not exists in the system	The returned value is null

CheckingDBFacade

Ref №	**Test scenarios**	**Test Case**	**Pre-condition**	**Expected result**
#1	Check AddItems Method	Check if the method is throwing an exception if the items passed as a parameter are duplicated and not already present in the system for that user	The items passed as a parameter are duplicated and not already present in the system for that user	The method throws “DuplicatedItemException” exception

#2	Check AddItems Method	Check if the method is throwing an exception if the items passed as a parameter are not duplicated but at least one of them is already present in the system for the specified user	The items passed as a parameter are not duplicated but at least one of them is already present in the system for the specified user	The method throws "ItemAlreadyPresentException" exception
#3	Check UpdateItems Method	Check if the method is throwing an exception if the items passed as a parameter are duplicated and already present in the system for that user	The items passed as a parameter are duplicated and already present in the system for that user	The method throws "DuplicatedItemException" exception
#4	Check UpdateItems Method	Check if the method is throwing an exception if the items passed as a parameter are not duplicated but at least one of them is not present in the system	The items passed as a parameter are not duplicated but at least one of them is not present in the system for the specified user	The method throws "ItemNotPresentException" exception

		for the specified user		
#5	Check UpdateItems Method	Check if the method is throwing an exception if the items passed as a parameter are not duplicated and they are all present in the system for the specified user, but at least one of them has a different type from the previous	The items passed as a parameter are not duplicated and are all present in the system for the specified user, but at least one of them has a different type from the previous	The method throws "ItemTypeChangedException" exception
#6	Check DeleteItems Method	Check if the method is throwing an exception if the items passed as a parameter are duplicated and already present in the system for that user	The items passed as a parameter are duplicated but at least one of them is not present in the system for the specified user	The method throws "DuplicatedItemException" exception
#7	Check DeleteItems Method	Check if the method is throwing an exception if the items passed as a	The items passed as a parameter are not duplicated but at least	The method throws "ItemNotPresentException" exception

		parameter are not duplicated but at least one of them is not present in the system for the specified user	one of them is not present in the system for the specified user	

Item

Ref №	**Test scenarios**	**Test Case**	**Pre-condition**	**Expected result**
#1	Check Item Constructor	Check if the constructor throws an exception if at least one of the properties has an invalid value	The property passed with invalid value exists in the ItemType	The constructor throws "FormatException" exception
#2	Check Item Constructor	Check if the constructor throws an exception if at least one of the properties does not exists in the ItemType	*None*	The constructor throws "ArgumentException" exception
#3	Check GetProperty Method	Check if the method returns the right value if the given property was set during the construction of the item	The given property was set during the construction of the item	The method returns a property with the value set during the construction
#4	Check GetProperty method	Check if the method returns "null" when we ask for a property that does not exist for the given ItemType	The property does not exist for the given ItemType	The method returns null

#5	Check GetProperty and SetProperty Methods	Check if the SetProperty method returns “false” when we try to set a property that does not exist and subsequent calls of GetProperty on the same property still returns “null”	The property does not exist for the given ItemType	SetProperty returns “false” and GetProperty returns null
#6	Check “Properties” Property	Check if the “Properties” get property returns all the properties of an Item, including the default properties if they were not set before	*None*	The returned value contains all the properties of an Item
#7	Check SetProperty Method	Check if the SetProperty method updates the value of a property that was not set before	The new value is valid for the property that is getting modified	GetProperty now returns the updated value for that property
#8	Check SetProperty Method	Check if the SetProperty method updates the value of a property that was set before	The new value is valid for the property that is getting modified	GetProperty now returns the updated value for that property

III) Fiddler tests

```
POST http://localhost:45337/Token HTTP/1.1
Accept: */*
Content-Type: application/json; charset=utf-8
Host: localhost:45337
Content-Length: 64

grant_type=password&username=admin@example.com&password=password

--

GET http://localhost:45337/api/Items/Get HTTP/1.1
Accept: */*
Content-Type: application/json; charset=utf-8
```

```
Host: localhost:45337
Authorization: Bearer 3uMn...

--

POST http://localhost:45337/api/Account/Register HTTP/1.1
Accept: */*
Content-Type: application/json; charset=utf-8
Host: localhost:45337
Authorization: Bearer 3uMn...
{"Email": "sozy@email.com", "Name":"SozyEnterprise", "Password":"password",
"ConfirmPassword":"password"}

--

POST http://localhost:45337/api/Items/Add HTTP/1.1
Accept: */*
Content-Type: application/json; charset=utf-8
Host: localhost:45337
Authorization: Bearer BdKj...

[{"Name":"Sumsang Universe S7","ProductNumber":"sms_S7","ItemTypeName":"Phone",
"Properties":{"4g":"false"}}]

--

PUT http://localhost:45337/api/Items/Delete HTTP/1.1
Accept: */*
Content-Type: application/json; charset=utf-8
Host: localhost:45337
Authorization: Bearer BdKj...

[{"ProductNumber":"sms_S7"}]
```