Use case name: Sign up

Participating Actors:
- User

Flow of Events:
1. User accesses the registration page or form on the system's website.
2. User provides the following information:
   - School ID
   - School Email Address
   - Password
3. The Registration System checks the provided information:
   - It verifies that the email is a Bilkent mail
4. If the information is valid, the Registration System creates a new user account
5. The system stores the user's information securely, associating it with the newly created user account.
6. The user is now registered in the system.

Entry Conditions:
- The user must be a student from Bilkent University.
- The user must provide a valid school email address.
- The user must have a connection.

Exit Conditions:
- The user's account is successfully created in the system, and they can now log in and access the system's features.

Quality Requirements:

Use case name: Login

Participating Actors:
- User

Flow of Events:
1. User navigates to the login page on the system's website.
2. User provides their login credentials:
   - School ID or School Email
   - Password
3. The Authentication System validates the provided credentials:

- It checks if the provided School ID or School Email exists in the system.
- It verifies the entered password by matching the one associated with the provided login credentials.

4. If the credentials are valid, the system logs the user into their account and provides access to the system's features and services.

Entry Conditions:
- The user must have an existing user account in the system, which is created through the sign up process.
- The user should enter the correct password.
- The user must have a connection.

Exit Conditions:
- The user is successfully logged into their account and can access the system's features and services.

Quality Requirements:

Use case name: Send Verification

Participating Actors:
- Authenticator
- User

Flow of Events:

1. The Authenticator generates a verification code and sends it to the user via their school email address.
2. User receives the verification code or link through the selected communication channel.
3. User enters the verification code, confirming their identity.
4. The Authenticator validates the provided code.
5. If the code or link is valid, the user's identity is confirmed.
6. The Authenticator is notified that the verification process was successful.

Entry Conditions:
- The user account for which verification is requested must be at the registration phase.
- The user must have a school email address.

- The user must have a connection.

Exit Conditions:
- The user's identity is verified, and the registration phase will be finished.

Quality Requirements:
- Verification mail should be sent in 2 minutes.
- Verification code should be entered in 5 minutes.

Use case name: Sign User

Participating Actors:
- Authenticator

Flow of Events:
1. Authenticator initiates the "Register User" process.
2. Authenticator creates a new and verified account in database.
3. The user is now registered in the system.

Entry Conditions:
- The user’s account shouldn’t exist in the database.
- The user must have a connection.

Exit Conditions:
- The user's account is successfully created in the system.

Quality Requirements:

Use case name: NoConnection

Participating Actors:
- System

Flow of Events:
1. If the system detects the connection loss, logs out the user of their session.
2. System sends an error message to inform the user about the connection loss
3. Login or register page opens again.

Entry Conditions:

- A connection issue must occur during the user's session.

Exit Conditions:

Quality Requirements:

- User shouldn’t be offline more than 1 hour.

Use case name: NotBilkentStudent

Participating Actors:

- System

Flow of Events:

1. If the system detects that the user is not a bilkent student, system ends the register process.
2. System sends an error message to inform the user about the website policies.
3. Register page opens again.

Entry Conditions:

- The user enters a non-bilkent mail address or a student id with the wrong format.

Exit Conditions:

Quality Requirements:

Use case name: NotUser

Participating Actors:

- System

Flow of Events:

1. If the system detects non-registered account info, the system sends an error message.
2. Login page opens again.

Entry Conditions:

- The user info shouldn’t be registered in the system

Exit Conditions:

Quality Requirements:

Use case name: WrongPassword

Participating Actors:

- System

Flow of Events:

3. If the system detects the wrong password for a registered account, the system sends an error message.
4. Login page opens again.

Entry Conditions:

- The user should already be registered in the system

Exit Conditions:

Quality Requirements:

- User can not enter the wrong password more than 5 times.

Use case name: AlreadyUser

Participating Actors:

- System

Flow of Events:

5. If the system detects registered account info while registering a new user, the system sends an error message.
6. Login page opens.

Entry Conditions:

- The user should already be registered in the system.
- The user should be trying to open a new user while having an existing account.

Exit Conditions:

Quality Requirements:

1. Name: LogOut

2. Participating Actors: User

3. Flow of Events:

1) User clicks on “Logout”.
2) The system prompts the user to confirm logout.
3) User confirms to log out of the website.

4. Entry Conditions:
- The user must be already registered to the system.
- The user must have a connection.

5. Exit Conditions:
- The user is logged out and the session has ended.

6. Special/quality requirements: The system should log out in less than 15 seconds.

1. Name:: ResetPassword

2. Participating Actors: User

3. Flow of Events:

1) User clicks on “Reset Password” and navigates to the related screen.
2) User provides his/her old password once and new password twice.
3) The system checks if:
    a. The old password is true.
    b. New password inputs are matched.
4) If the inputs are matched, the system replaces the old password of the user with the new one.

4. Entry Conditions:
- The user must be logged in.
- The user must have a connection.

5. Exit Conditions:
- The user’s password is changed.

6. Special/quality requirements: The system should check the new password and replace it in under 10 seconds when the request is made.

1. Name: DeleteAccount

2. Participating Actors: User

3. Flow of Events:

1) User clicks on “Delete Account” and navigates to the related screen.
2) User provides the password to the system.
3) The system validates the password.
4) If the password is true, the system prompts the user to confirm to delete the account.
5) If the user confirms, the account is deleted.

4. Entry Conditions:
- The user must be registered to the system.
- The user must have a connection.

5. Exit Conditions:

- The user’s account is permanently deleted.
- Session ends and the user is logged out.

6. Special/quality requirements: The account should be deleted in less than 10 seconds.

1. Name: AddItemtoBasket

2. Participating Actors: Customer

3. Flow of Events:

1) Customer browses the desired item.
2) Customer clicks on “Add to Basket.”
3) If the item is available, the system adds the item to the basket of the customer.

4. Entry Conditions:
- The user must be logged in.
- The item must be available in the stock.

5. Exit Conditions:
- The selected item is added to the customer’s basket.

6. Special/quality requirements: The system should add the item instantaneously (<1 second)

1. Name: InspectItem

2. Participating Actors: Customer

3. Flow of Events:

1) Customer clicks on the desired item.
2) The system checks the availability of the item in stock.
3) If the item is available, the system displays information about the item, including:

    - The price of the item
    - The image of the item
    - Descriptions

4. Entry Conditions:

- The user must be logged in.

5. Exit Conditions:
- The selected item's information is successfully displayed.

6. Special/quality requirements: The item should be showed instantaneously (<2 seconds.)

1. Name: SearchSecondHandItems

2. Participating Actors: Customer

3. Flow of Events:

1) Customer navigates to the stock page of the system.
2) Customer searches the item, providing filters including:

    - Price range
    - Course code in case of books
    - Tags
    - Keywords

3) If the item is available, the system displays the item in stock.

4. Entry Conditions:
- The user must be logged in.

5. Exit Conditions:
- The system displays the items that match the search criteria.

6. Special/quality requirements: The algorithm should terminate in less than 10 seconds.

1. Name: RateSeller
2. Participating Actor: Customer
3. Entry Condition:
- Customer has bought a good/service from the specific seller

4. Exit Condition:

- Customer has entered a rating
- Customer cancels the action

5. Flow of Events:

1) Customer activates the generic **Rate** functionality on the terminal
2) Customer rates the quality of the good/service bought from the seller using **Rate Seller** use case which uses the generic **Rate** functionality
3) A public rating appears on the profile of the seller, for other customers to see

6. Special/quality requirements: The given rating should be visible to other users instantaneously (<5 seconds).

1. Name: RateCustomer

2. Participating Actor: Seller

3. Entry Condition:

- Seller has sold a good/service to the customer

4. Exit Condition:

- Seller has entered a rating
- Seller cancels the action

5. Flow of Events:

1) Seller has sold a good/service to the customer
2) Seller rates the quality of the process based on the specific customer using the **Rate Customer** use case which uses the generic **Rate** functionality
3) A public rating appears on the profile of the customer, for other sellers to see

6. Special/quality requirements: The given rating should be visible to other users instantaneously (<5 seconds).

1. Name: PostFoundItem

2. Participating Actor: FinderOfItem

3. Entry Condition:

- FinderOfItem decides to notify the LoserOfItem

4. Exit Condition:

- FinderOfItem post request is successful
- FinderOfItem cancels the post

5. Flow of Events:

1) FinderOfItem activates the **PostFoundItem** functionality on the terminal
2) FinderOfItem enters the specifics of the item that is found
3) Using the generic **PostItem** functionality, **PostFoundItem** requests to post the item and it's specifications (picture, attributes) to the app where LoserOfItem actors can view the found item posts
4) Post request returns successful

6. Special/quality requirements: Post should be visible to other users instantaneously (<5 seconds)

1. Name: PostLostItem
2. Participating Actor: LoserOfItem
3. Entry Condition:
- LoserOfItem decides to make a post request for the lost item

4. Exit Condition:
- LoserOfItem's post is posted publicly
- LoserOfItem cancels the post

5. Flow of Events:
1) LoserOfItem activates the PostLostItem functionality on the screen
2) LoserOfItem enters the specifics of the lost item
3) **PostLostItem** requests to post the lost item with its specifications (picture, attributes) using the generic **PostItem** functionality
4) Post request returns successful

6. Special/quality requirements: Post should be made instantaneously (<5 seconds)

1. Name: RateLoser
2. Participating Actor: FinderOfItem
3. Entry Condition:
- There exists a previous lost item that the **FinderOfItem** has found and successfully returned to the **LoserOfItem**.

4. Exit Condition:
- FinderOfItem enters the rating
- FinderOfItem cancels the action

5. Flow of Events:
1) FinderOfItem activates the **RateLoser** functionality on the terminal
2) FinderOfItem enters the rating using the **RateLoser** use case which uses the generic **Rate** functionality
3) The entered rating becomes visible in the **LoserOfItem**'s profile

6. Special/quality requirements: The entered rating should become visible to other users instantaneously (<5 seconds)

1. Name: RateFinder
2. Participating Actor: LoserOfItem
3. Entry Condition:
- There exists a previous lost item that was retrieved by the **FinderOfItem** and returned to the **LoserOfItem**

4. Exit Condition:

- LoserOfItem enters the rating
- LoserOfItem cancels the action

5. Flow of Events:

1) LoserOfItem activates the **RateFinder** functionality on the terminal
2) LoserOfItem enters the rating supported by the generic **Rate** functionality
3) The entered rating becomes visible in the **FinderOfItem**'s profile

6. Special/quality requirements: The entered rating should be visible instantaneously (<5 seconds)

1. Name: ViewSeller

2. Participating Actor: Customer

3. Entry Condition:

- Customer decides to view the profile of a Seller

4. Exit Condition:

- Customer has viewed the profile of a Seller

5. Flow of Events:

1) Customer selects the Seller, from which he/she is interested in buying a good/service from (or not interested).
2) Customer then enters the profile of the selected seller (which is an object) with the support of the generic **ViewProfile** functionality
3) Customer views the different ratings the seller has gotten from different users, and other specific information seller has specified about him/herself

6. Special/quality requirements: The profile of the selected Seller should be accessible instantaneously (< 3 seconds)

1. Name: ViewCustomer

2. Participating Actor: Seller

3. Entry Condition:

- Seller decides to view the profile of the Customer

4. Exit Condition:

- Seller has viewed the profile of the Customer

5. Flow of Events:

1) Seller selects the Customer, from which he/she might be interested in doing business with for a good/service (or not interested)
2) Seller then enters the profile of the selected Customer with the support of the generic **ViewProfile** functionality
3) Seller views the different ratings the customer has gotten from different users, and other specific information customer has specified about him/herself

6. Special/quality requirements: The profile of the selected Customer should be accessible and viewable instantaneously (< 3 seconds)

1. Name: ViewFinder

2. Participating Actor: LoserOfItem
3. Entry Condition:
- LoserOfItem selects a FinderOfItem

4. Exit Condition:
- LoserOfItem has viewed the profile of the selected FinderOfItem

5. Flow of Events:
1) LoserOfItem selects the FinderOfItem
2) LoserOfItem then enters the profile of the selected FinderOfItem with the support of the generic **ViewProfile** functionality
3) LoserOfItem views the different ratings the FinderOfItem has gotten from different users, and other specific information FinderOfItem has specified about him/herself

6. Special/quality requirements: The FinderOfItem profile should accessible and viewable instantaneously (< 3 seconds)

1. Name: ViewLoser
2. Participating Actor: FinderOfItem
3. Entry Condition:
- FinderOfItem selects a LoserOfItem

4. Exit Condition:
- FinderOfItem has viewed the profile of the LoserOfItem

5. Flow of Events:
1) FinderOfItem selects the LoserOfItem
2) FinderOfItem then enters the profile of the selected LoserOfItem with the support of the generic **ViewProfile** functionality
3) **FinderOfItem** views the different ratings the **LoserOfItem** has gotten from different users, and other specific information **LoserOfItem** has specified about him/herself

6. Special/quality requirements: The LoserOfItem's profile should accessible and viewable instantaneously (< 3 seconds)

1. Name: ViewOwnProfile
2. Participating Actor: User
3. Entry Condition:
- User selects own profile

4. Exit Condition:
- User has viewed the profile

5. Flow of Events:
1) **User** selects his/her own profile
2) **User** views the different ratings that were given to the profile by different **Users**, and the ratings him/herself has given to different **Users**

6. Special/quality requirements: The user profile should accessible and viewable instantaneously (< 3 seconds)

1. Name: SearchLostItem
2. Participating Actor: LoserOfItem
3. Entry Condition:

- LoserOfItem makes a lost item post to the system. The system tries to create a match between: the specified lost item post that contains the specifications of the lost item, and the found item post by the FinderOfItem that also contains the specifications of the found item. If the system is unable to create a match within 3 days, then all the found item posts become visible to the LoserOfItem who then is able to search him/herself.

4. Exit Condition:

- LoserOfItem finds the lost item
- LoserOfItem stops searching for the item

5. Flow of Events:

1) LoserOfItem activates the search algorithm by entering keywords to the designated search bar
2) The algorithm tries to find the potential matches between the entered keywords and the posted found items specifications using word vectorization techniques etc.
3) The matched posts are filtered and is showed to the LoserOfItem (if there are no matches, nothing is shown)

6. Special/quality requirements: The algorithm should terminate seamlessly (< 5 seconds)

1. Use Case Name: CommunicateWithSeller
2. Participating Actors: Customer
3. Entry Condition:

-Customer should become interested to the post and has to send a message to the seller via **Chat** use case or a Seller should be interested in one of Customer's requests such as private lesson request or course trade request.

4. Exit Condition: Customer's message should be sent to the seller.
5. Flow of events:

1) Customer sees a post or receives a message from the seller about one of his/her requests.
2) If the customer wants to contact the seller in order to talk about the details of the trade, he/she can contact the seller by using the **Chat** use case.
3) By using the **Chat** use case, the customer can directly send a message to the seller and talk about the details.

6. Quality Requirements:
Customer's message should be delivered to the seller instantaneously; in other words, **Chat** use case which allows momentary direct messages which are private to

sender and receiver, and stores the past messages, should be dynamic enough to provide the customer and seller a communication environment without delays.

1. Use Case Name: CommunicateWithCustomer
2. Participating Actors: Seller
3. Entry Condition:

-Seller should have received a message from the customer or seller should be interested in one of a user's request(private lesson request or course trade request.

4. Exit Condition: Seller's message should be sent to the seller.
5. Flow of events:

1) Seller has received a message via **Chat** use case, from a Customer who is interested in one of the seller's posts or the Seller has seen a request from a customer and became interested in it.
2) Seller sends a message back to the Customer via the **Chat** use case and talks about the details for the trade or if the case is about requests, the seller sends a message to customer and starts messaging.

6. Quality Requirements:
Seller's message should be delivered to the customer instantaneously; in other words, **Chat** use case which allows momentary direct messages which are private to sender and receiver, and stores the past messages, should be dynamic enough to provide the customer and seller a communication environment without delays.

1. Name: CommunicateWithFinder
2. Participating Actor: LoserOfItem
3. Entry Condition:

-The posts for the lost items must be matched for the loser to communicate with the finder.

-If the posts are not matched, then the lost item post of the finder can be seen by everybody, after 3 days, and the loser of the item can communicate with the finder so 3 days should be passed after the post date of the FinderOfItem.

4. Exit Condition: LoserOfItem's message should be sent to the seller.
5. Flow of Events:

1) LoserOfItem sees the post of the FinderOfItem, either by post matching(the post that loser's and finder's posts for the item matches) or after 3 days of no match.
2) If the item that is posted by the FinderOfItem belongs to LoserOfItem, LoserOfItem sends a message to FinderOfItem by using the **Chat** use case and talk about the details such as the features of the item(to be sure that item belongs to him/her) or the details of the meeting(to get the item back from FinderOfItem).

6. Special/quality requirements:
LoserOfItem's message should be delivered to the FinderOfItem instantaneously; in other words, **Chat** use case which allows momentary direct messages which are

private to sender and receiver, and stores the past messages, should be dynamic enough to provide the LoserOfItem and FinderOfItem a communication environment without delays.

1. Name: CommunicateWithLoser
2. Participating Actor: FinderOfItem
3. Entry Condition:
-FinderOfItem has received a message from the LoserOfItem about one of his/her lost item posts.
4. Exit Condition: FinderOfItem's message should be sent to LoserOfItem.
5. Flow of Events:
1) FinderOfItem receives a message from the LoserOfItem via the **Chat** use case.
2) FinderOfItem sends a message back to LoserOfItem with the **Chat** use case in order to talk about the details such as the features of the item(to be sure that item belongs to him/her) or the details of the meeting(to give the item back to LoserOfItem).

6. Special/quality requirements:
FinderOfItem's message should be delivered to the LoserOfItem instantaneously; in other words, **Chat** use case which allows momentary direct messages which are private to sender and receiver, and stores the past messages, should be dynamic enough to provide the LoserOfItem and FinderOfItem a communication environment without delays.

1. Name: AskRecommendation
2. Participating Actor: User
3. Entry Condition:
- User wants to gather information for a good or service by asking other users' opinions in a forum-like discussion environment.
4. Exit Condition: User has created a discussion topic.
5. Flow of Events:
1) User creates a new discussion topic and enters his/her question about a good or service, in order to get information from the users who have prior experience or knowledge for that good or service.
2) The new discussion topic is added to the system and can be seen by other users.

6. Special/quality requirements:
The new discussion topic should be added to the system and be visible to other users instantaneously.

1. Name: AnswerRecommendation
2. Participating Actor: User
3. Entry Condition:

-User wants to participate in the discussion by publishing his/her opinions/answer about the topic, in the text format.

4. Exit Condition: User has published his/her answer in the forum-like discussion environment.

5. Flow of Events:

1) User becomes interested in the discussion topic and wants to participate in it.
2) User writes his/her opinion about the topic in text format.
3) User publishes the text he/she wrote in the discussion environment.
4) User's text is added at the bottom of the discussion page.

6. Special/quality requirements:

User's text should be added to the system and be visible to other Users instantaneously.

1. Name: CreateSecondHandListing

2. Participating Actor: Seller

3. Entry Condition:

-Seller wants to sell a second hand item.

4. Exit Condition: Seller posts every second hand item that he/she wants to sell.

5. Flow of Events:

1) Seller decides to sell one or more second hand items.
2) Seller uses the **postItem** use case, which allows him/her to post the items that he/she wants to sell one by one. To post an item, the Seller uploads the photo(s) of the item, an explanation of the item, the price of the item and, the item's name and associated tags(For example, "CS 223" can be a tag for a Digital Design Book).
3) Seller's post(s) are uploaded to the system and available for Customers.
4) Seller can view his/her own second hand item posts and edit them.

6. Special/quality requirements:

The posts should be visible to Customers instantaneously.

1. Name: CreatePrivateLessonListing

2. Participating Actor: Seller

3. Entry Condition:

-Seller wants to give private lessons to customers.

4. Exit Condition: Seller posts every different type of private lesson that he/she wants to give.

5. Flow of Events:

1) Seller decides to give one or multiple private lessons.
2) Seller uses the **postItem** use case, which allows him/her to post the private lessons that he/she wants to give one by one. To post a private lesson, the Seller uses the name, explanation, price and tag features of the **postItem**(For example, "Math 101" can be a tag for Calculus private lesson.
3) Seller's post(s) are uploaded to the system and available for Customers.
4) Seller can view his/her own private lesson posts and edit them.

6. Special/quality requirements:
The posts should be visible to Customers instantaneously.

1. Name: CreateRentingListing
2. Participating Actor: Seller
3. Entry Condition:
-Seller wants to rent his/her item(s).
4. Exit Condition: Seller has posted all of the items that he/she wants to rent to customers.
5. Flow of Events:

1) Seller decides to rent one or multiple items.
2) Seller uses the **postItem** use case, which allows him/her to post items for renting one by one. In order to post an item for for renting, Seller uploads the photo(s) of the item, an explanation of the item, the price of the item and, the item's name and associated tags(For example, "CS 223" can be a tag for a Digital Design Book).
3) Seller's posts are uploaded to system and available for Customers.
4) Seller can view his/her own posts and edit them.

6. Special/quality requirements:
The posts should be visible to customers instantaneously.

1. Name: PostPrivateLessonRequest
2. Participating Actor: Customer
3. Entry Condition:
-Customer wants to have a private lesson in any field.
4. Exit Condition: Customer posts a private lesson request.
5. Flow of Events:

1) Customer decides to have a private lesson in any field/discipline.
2) Customer creates a post by using the **PostRequest** use case which allows him/her to notify the sellers that he/she has an interest in a good or service. By posting a request, the customer specifies his/her requests by writing the name(the field of the private lesson, such as "Calculus", and the price that he/she is ready to pay for a private lesson.
3) Customer's request is uploaded to the system and available for Sellers.
4) Customer can view his/her request and edit it.

6. Special/quality requirements:
The post should be visible to customers instantaneously.

1. Name: PostTradeRequest
2. Participating Actor: CourseTrader
3. Entry Condition:
-CourseTrader wants to have a course.
4. Exit Condition: CourseTrader posted a request.
5. Flow of Events:

1) CourseTrader decides to have a course.
2) CourseTrader creates a post by using **PostRequest** use case which allows him/her to notify other CourseTraders that he/she wants to take that course. By posting a request, the CourseTrader specifies the exact course and section in a specific format (for ex: MATH 101- Sec:3), without any other details.
3) CourseTrader's request is uploaded to the system and available for other CourseTraders.
4) CourseTrader can view his/her request and edit it.

6. Special/quality requirements:
The post should be visible to other CourseTraders instantaneously.

1. Name: PostTradeRequest
2. Participating Actor: CourseTrader
3. Entry Condition:
-CourseTrader wants to give a course.
4. Exit Condition: CourseTrader has posted the course that he/she wants to give.
5. Flow of Events:
1) CourseTrader decides to drop a course or ready to drop a course in exchange with another course.
2) By using **postItem** use case, he/she posts the course that he/she wants to drop, by specifying the course code and section(ex: MATH 101 - Sec: 3), without any other details.
3) CourseTrader's post is uploaded to the system and available for other CourseTraders.
4) CourseTrader can view and edit his/her post.

6. Special/quality requirements:
The post should be visible to other CourseTraders instantaneously.

1. Name: CommunicateWithTrader
2. Participating Actor: CourseTrader
3. Entry Condition:
- CourseTrader is interested in a course that some other CourseTrader has posted for trade(the other one wants to give) or is interested with some other CourseTrader's course request(that the other CourseTrader wants to take). Also, another CourseTrader might communicate with him/her according to his/her course requests or course trade posts.

4. Exit Condition: CourseTrader has sent a message to other CourseTrader.
5. Flow of Events:
1) CourseTrader might receive a message from another CourseTrader or might be interested in another CourseTrader's course request or trade post(if the original CourseTrader wants to have a course, he/she will be interested in other's trade post and if the original CourseTrader wants to drop a course, he/she will be interested in other's course request post.

2) CourseTrader sends a message to other by using **Chat** use case, to talk about the details for course trade. For example, CourseTrader sees a course request of another CourseTrader. If the original CourseTrader wants to give that course and take another one, he/she can send a message to other CourseTrader specifying that he/she can give him/her the course but in exchange with some other course. So, CourseTraders can communicate with each other and by using the **Chat** use case, they can talk about the details of the course trade and make themselves better off if the courses they want to take and give match.

6. Special/quality requirements:

CourseTrader's message should be delivered to other CourseTrader instantaneously; in other words, **Chat** use case which allows momentary direct messages which are private to sender and receiver, and stores the past messages, should be dynamic enough to provide the LoserOfItem and FinderOfItem a communication environment without delays.

1. Name: SearchCoursesForTrade

2. Participating Actor: CourseTrader

3. Entry Condition:

-CourseTrader wants to see other CourseTraders' course requests and trade posts, for some specific courses and sections.

4. Exit Condition: CourseTrader sees other posts and requests.

5. Flow of Events:

1) If the CourseTrader wants to take a course, he/she searches for other CourseTrades' course trade posts, with the specific course code and section. Vice versa, if the CourseTrader wants to give a course, he/she searches for other CourseTraders' course requests.
2) The trade and request posts about the specific course are available to the CourseTrader.

6. Special/quality requirements:

The post should be visible to CourseTrader instantaneously, when the search is applied.

1. Name: AddItemToFavourites

2. Participating Actor: Customer

3. Entry Condition:

-Customer likes an item(or any post, such as private lesson) and wants to save it for the future.

4. Exit Condition: Customer adds the post to his/her favorites list.

5. Flow of Events:

1) Customer sees a post and wants to save it for the future in order to not to lose the post or track the price of the item-service.
2) Customer adds the post to his/her favorites list.

3) Customer can view his/her favorite posts. However, if the post is deleted by the Seller(the item is selled to somebody else etc.), the post also is deleted from the Customer's favorite list.

6. Special/quality requirements:

Customers should be able to add the post to his/her favorite list instantaneously. Also, when the post is deleted, the post should be removed from the Customer's list momentarily.