CS 319 - Bilcon Web Application D1-D2 Final Report

Section 1 - Team 1

Group Members:

Öykü Demir - 22101934

Begüm Kunaç - 2210383

Hakan Muluk - 22102512

Gün Taştan - 22101850

Mehmet Onur Özdemir - 22102293

Mustafa Gökalp Gökdoğan - 22102936

1. Introduction

This project is a bespoke website, purposefully designed to cater exclusively to the distinguished members of our university community, including students, esteemed faculty, and dedicated staff. "Bilcon" is poised to elevate the university experience by providing a meticulously crafted platform that caters to all your buying and selling needs. Welcome to a sophisticated digital marketplace tailored exclusively to our esteemed institution!

"Bilcon" provides a simplistic user interface, enabling seamless navigation and utilization of its diverse features. Whether you're a buyer seeking the perfect product, a seller looking to showcase your items, or a student on the lookout for the ideal roommate, "Bilcon" has you covered.

Our platform prioritizes convenience and efficiency, offering buyers the ability to search for products by tags and names, communicate with sellers, and even negotiate prices. Sellers, on the other hand, can effortlessly list items with prices and tags, creating a straightforward selling experience. "Bilcon" also facilitates borrowing transactions, allowing sellers to define borrowing durations while offering flexibility to buyers.

For those who have found or lost items, "Bilcon" provides a safe and user-friendly space to connect and exchange information, all while protecting sensitive data. Additionally, the platform empowers users to trade in knowledge by facilitating the sale of private lessons. The inclusion of a transparent seller rating system ensures that user experiences are informed and reliable.

With "Bilcon," your university's community is at the forefront. We've thoughtfully integrated a feature that allows buyers to review a seller's previous sales, giving you the confidence to make informed decisions. To further foster connections and collaborations, we've included direct messaging capabilities for all functions.

Your account page is a secure gateway, requiring school email and ID, ensuring your privacy and data protection. "Bilcon" is not just a platform; it's a community-oriented digital solution tailored to enhance your university experience. Welcome to a platform designed exclusively for you.

2. Proposed System

2.1 Non-Functional Requirements

2.1.1 Usability

Crafting a user-friendly interface was a top priority in our program's design. We have designed our interfaces to prioritize simplicity and functionality, all while maintaining consistency and ensuring a swift response time. Our early designs are centered around user needs to enhance the overall user experience. Our goal is to present the web app's elements in a clean and minimalistic manner, guaranteeing a straightforward and easily navigable appearance. Every part of the interface is carefully designed to be easily accessible, thus ensuring a pleasant and intuitive user experience with the added benefits of consistency and a quick response time for a seamless interaction.

2.1.2 Performance

We have significantly improved the overall performance of this program by prioritizing two critical aspects: response time and availability. Our dedicated efforts have resulted in a design focused on a program that not only responds swiftly but is also consistently available to users. This enhancement ensures a seamless user experience and underscores our commitment to delivering a high-performing solution.

2.1.3 Adaptability

Because of the object oriented features of our design to build our application, adding new features will be easier. We have prioritized the maintainability of our program, particularly for bug fixes and the addition of new features. Our approach ensures that making updates or resolving issues will be straightforward and won't disrupt the core codebase. This approach is not only advantageous for maintaining the program but also prevents any complexities during optimization, ultimately leading to a more efficient and robust application.

2.2 Tech Stack

The tech stack we will use in Bilcon Web Application is MERN Stack (MongoDB, Express.js, React and Node.js).
The reason for this is that MERN Stack allows us to create an efficient and scalable application. Providing flexible and scalable data storage thanks to MongoDB, minimalist and robust features of Express, and highly interactive and dynamic user interface of React were our main reasons for choosing.
Providing a runtime environment for Node.js ensures consistent use of JavaScript in the frontend and backend.

At the same time, the large number of users and developers increases the number of solutions and help that can be obtained from the internet and resources when a problem is encountered.

2.3 System Models

2.3.1 Use Case Model

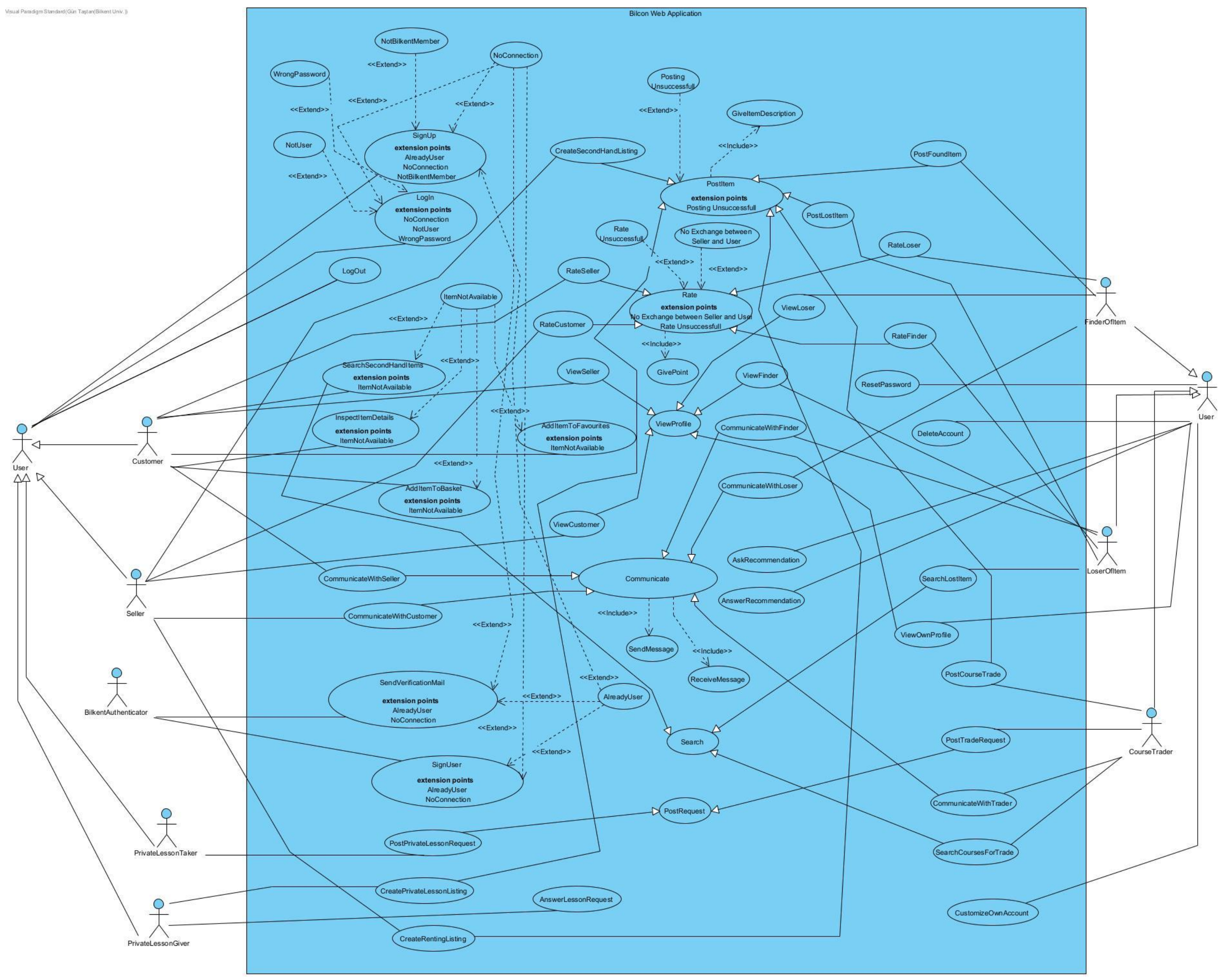


Fig. 1: Use Case Diagram for Bilcon

For easier reading of the Use Case Diagram, a file with the .vpp extension is available in the GitHub repository of the project.

Use case name: SignUp
Participating Actors: Initiated by User
Flow of Events:

1. The User clicks the RegisterButton.

    2. Bilcon responds by presenting the registration form to the User.

3. The User provides the required information: Bilkent ID, Bilkent mail address, password.

4. Once the form is completed, the User submits the registration information by clicking SignUpButton.

5. Otherwise, if the User changes their mind, they click on CancelButton to exit registration.

    6. Bilcon verifies the information and creates a new user account.

    7. Otherwise, the system does not create any new account if the User clicks CancelButton and exits registration.

Entry Conditions:

- The User has 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, OR
- The User changes their mind about registering and clicks on CancelButton, and the system closes the form.

Use case name: AlreadyUser
Participating Actors: Communicates with User
Flow of Events:

1. The User fills the registration form and clicks SignUpButton.

    2. If the user already exists, the system sends the warning message: “This user already exists.”

Entry Conditions:

- This use case **extends** the SignUp use case. It is initiated by the system whenever a user already exists with the same information.

Use case name: NoConnection
Participating Actors: Communicates with User
Flow of Events:

1. The User sends a request to the system.

    2. The system attempts to reach the data.

    3. If the system detects connection loss, the system sends the warning message: “No connection.” and logs the User out of their session.

    4. The system navigates back to the Login screen.

Entry Conditions:

- This use case **extends** all use cases. It is initiated by the system whenever a connection issue occurs during the user's session.

Exit Conditions:

- The session of the User ends and the system navigates back to the Login screen.

Quality Requirements:
- The User shouldn’t be offline for more than 1 hour.

Use case name: NotBilkentMember
Participating Actors: Communicates with User
Flow of Events:
1. The User fills the registration form and clicks SignUpButton.
	2. The system checks the provided information.
	3. If Bilkent ID or Bilkent mail is invalid, the system sends the warning message: “Invalid Bilkent ID or Bilkent mail.”

Entry Conditions:
- This use case **extends** the SignUp use case. It is initiated by the system whenever the provided information doesn’t belong to a Bilkent member.

Use case name: Login
Participating Actors: Initiated by User
Flow of Events:
1. The User navigates to the Login page.
	2. Bilcon responds by presenting the login form to the User.
3. The User provides their login credentials: Bilkent ID or Bilkent mail, and password.
4. Once the form is completed, the User submits the credentials by clicking LoginButton.
	5. The system validates the provided information.
	6. 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 has a connection.
- The User navigates to the Login page.

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

Use case name: NotUser
Participating Actors: Communicates with User
Flow of Events:
1. The User fills the registration form and clicks LoginButton.
	2. The system checks if the provided information matches with an existing user.
	3. If the information does not belong to a user in the system, the system sends the warning message: “User not found.”

Entry Conditions:
- This use case **extends** the Login use case. It is initiated by the system whenever the provided information doesn’t belong to a user that exists in the system.

Exit Conditions:
- The system sends the warning message: “User not found.”

Use case name: WrongPassword
Participating Actors: Communicates with User
Flow of Events:
1. The User fills the registration form and clicks LoginButton.
 2. The system checks if the provided information matches with an existing user.
 3. If the password does not match with the one in the system, the system sends the warning message: “Invalid credentials.”
Entry Conditions:
- This use case **extends** the Login use case. It is initiated by the system whenever the provided password does not match with the one in the system.

Exit Conditions:
- The system sends the warning message: “Invalid credentials.”

Use case name: LogOut
Participating Actors: Initiated by User
Flow of Events:
1. The User clicks on LogOutButton.
 2. The system prompts the user to confirm logout.
3. The User confirms to log out of the website.
4. Otherwise, if the User changes their mind, they click on CancelButton to cancel log out.
 5. Bilcon logs the User out of the system and ends the session.
 6. Otherwise, the system does not log the User out if the User clicks CancelButton.
Entry Conditions:
- The User has a connection.
- The User is logged into the system.
- The User clicks on the Settings page.

Exit Conditions:
- The User logs out and the session ends, OR
- The User changes their mind about logging out and clicks on CancelButton.

Quality Requirements:
- The system should log the User out in less than 15 seconds.

Use case name: ResetPassword
Participating Actors: Initiated by User
Flow of Events:
1. The User clicks on the ResetPasswordButton.
 2. The system provides a form for the User to fill with old and new password information.
3. The User provides the old password once and new password twice.
 4. The system checks if the old password is true and the new password inputs

are matched.

5. If the inputs are correct, the system replaces the old password with the new one and prompts “Successful.”

Entry Conditions:

- The User has a connection.
- The User is logged into the system.
- The User clicks on the Settings page.

Exit Conditions:

- The User’s password is changed.

Quality Requirements:

- The system should check the new password and replace it in under 10 seconds when the request is made.

Use case name: DeleteAccount

Participating Actors: Initiated by User

Flow of Events:

1. User clicks on the DeleteAccountButton.

2. The system prompts the user to provide their password.

3. The User provides their password to the system.

4. Otherwise, if the User changes their mind, they click on CancelButton to cancel account deletion.

4. The system validates the password and if the inputs are correct, deletes the account.

5. Otherwise, the system does not delete the account if the User clicks CancelButton.

Entry Conditions:

- The User has a connection.
- The User is logged into the system.
- The User clicks on the Settings page.

Exit Conditions:

- The User’s account is permanently deleted. The session ends and the user is logged out, OR
- The User changes their mind about deleting the account and clicks on CancelButton.

Use case name: Search

Participating Actors: Initiated by Customer, LoserOfItem or CourseTrader

Flow of Events:

1. If the user is a Customer, they can search items on the market.

2. If the user is a LoserOfItem, they can search for their lost item.

3. If the user is a CourseTrader, they can search for courses to trade.

4. The system displays the items that match the search criteria.

Entry Conditions:

- The user has a connection.
- The user is logged into the system.
- If the user is a Customer, they navigate to the Market page of the system.

- If the user is a LoserOfItem, they navigate to the LostandFound page of the system.
- If the user is a CourseTrader, they navigate to the CourseTrade page of the system.

Exit Conditions:

- The system displays the items that match the desired product.

Use case name: SearchSecondHandItems

Participating Actors: Inherited from Search use case

Flow of Events:

1. Customer searches the item, providing filters if necessary, such as: price range, course code in case of books, keywords, tags and sees the results.

Entry Conditions:

- **Inherited** from Search use case.

Exit Conditions:

- **Inherited** from Search use case.

Quality Requirements:

- Search can be done by using tags and filters.

Use case name: ItemNotAvailable

Participating Actors: Communicates with Customer

Flow of Events:

1. The Customer searches for the item, OR attempts to inspect the item, OR attempts to add it to favorites or basket.
2. The system checks if there is an item that matches the criteria.
3. If there is no such item, the system sends the warning message: “Item not available.”

Entry Conditions:

- This use case **extends** SearchSecondHandItems, InspectItemDetails, AddItemToBasket, and AddItemToFavourites use cases. It is initiated by the system whenever the desired item does not exist in the system.

Exit Conditions:

- The system sends the warning message: “Item not available.”

Use case name: SearchLostItem

Participating Actors: Inherited from Search use case

Flow of Events:

1. LoserOfItem searches the item that they lost, providing the specifications of the item, and sees the results.

Entry Conditions:

- **Inherited** from Search use case.

Exit Conditions:

- **Inherited** from Search use case.

Use case name: SearchCoursesForTrade

Participating Actors: Inherited from Search use case

Flow of Events:

1. CourseTrader searches for a course to trade providing the information about the course to drop and the course to add, and sees the results.

Entry Conditions:

- **Inherited** from Search use case.

Exit Conditions:

- **Inherited** from Search use case.

Use case name: InspectItemDetails

Participating Actors: Initiated by Customer

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, its image and descriptions.

Entry Conditions:

- The user has a connection.
- The user is logged into the system.
- The user is on the Market screen of Bilcon.

Exit Conditions:

- The selected item's information is successfully displayed.

Use case name: AddItemToBasket

Participating Actors: Initiated by Customer

Flow of Events:

1. Customer clicks on AddToBasketButton on the desired item.

    2. The system checks the availability of the item in stock.

    3. If the item is available, the system adds the item to the basket of the Customer.

Entry Conditions:

- The user has a connection.
- The user is logged into the system.
- The user is on the Market screen of Bilcon.

Exit Conditions:

- The selected item is added to the Customer's basket.

Use case name: AddItemToFavourites

Participating Actors: Initiated by Customer
Flow of Events:
1. Customer clicks on AddToFavouritesButton on the desired item.
	2. The system checks the availability of the item in stock.
	3. If the item is available, the system adds the item to the Favorites of the Customer.

Entry Conditions:
- The user has a connection.
- The user is logged into the system.
- The user is on the Market screen of Bilcon.

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

Use case name: PostItem

Participating Actors: Initiated by Seller, FinderOfItem, LoserOfItem, CourseTrader or PrivateLessonGiver
Flow of Events:
1. The Seller/FinderOfItem/LoserOfItem/CourseTrader/PrivateLessonGiver clicks to the related Post button.
	2. The system displays the form to post an item.
3. This use case **includes** the GiveItemDescription use case. The GiveItemDescription use case is initiated when the user provides item descriptions for the item they want to post.
4. If the user is a Seller, they can create a second-hand or renting listing for the item they want to sell or donate.
5. If the user is a FinderOfItem, they can post information about the item that they found.
6. If the user is a LoserOfItem, they can post information about the item that they lost.
7. If the user is a PrivateLessonGiver, they can create a private session listing for the lesson they want to give.
8. If the user is a CourseTrader, they can post information about the course that they want to trade.
	9. The system checks the data and posts the item if the required information is provided.

Entry Conditions:
- The user has a connection.
- The user is logged into the system.
- If the user is a Seller, they navigate to the Market or Renting page of the system.
- If the user is a FinderOfItem or LoserOfItem, they navigate to the LostandFound page of the system.
- If the user is a CourseTrader, they navigate to the CourseTrade page of the system.
- If the user is a PrivateLessonGiver, they navigate to the PrivateLessons page of the system.

Exit Conditions:
- The user posts the desired item.

Quality Requirements:

●	The posts should be visible to Customers instantaneously.

Use case name: PostingUnsuccessful
Participating Actors: Communicates with Seller, FinderOfItem, LoserOfItem, CourseTrader or PrivateLessonGiver
Flow of Events:

1. The Seller/FinderOfItem/LoserOfItem/CourseTrader/PrivateLessonGiver attempts to post an item.
	2. The system checks the data.
	3. If the required information is missing or the posting can not be completed, the system sends the warning message: “Posting Unsuccessful.”

Entry Conditions:

●	This use case **extends** the PostItem use case. It is initiated by the system whenever the required information is missing or the posting can not be completed.

Exit Conditions:

●	The system sends the warning message: “Posting Unsuccessful.”

Use case name: CreateSecondHandListing
Participating Actors: Inherited from PostItem use case
Flow of Events:

1. Seller clicks on CreateSecondHandListingButton.
2. 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, and posts the item.

Entry Conditions:

●	**Inherited** from PostItem use case.

Exit Conditions:

●	**Inherited** from PostItem use case.

Quality Requirements:

●	The price of the item can not be a negative number.

Use case name: CreateRentingListing
Participating Actors: Inherited from PostItem use case
Flow of Events:

1. Seller clicks on CreateRentingListingButton.
2. Seller uploads the photo(s) of the item, an explanation of the item, the price of the item, the time interval for renting the item, and the item’s name and associated tags, and posts the item.

Entry Conditions:

●	**Inherited** from PostItem use case.

Exit Conditions:

●	**Inherited** from PostItem use case.

Quality Requirements:

- The price of the item can not be a negative number.

Use case name: CreatePrivateLessonListing
Participating Actors: Inherited from PostItem use case
Flow of Events:
1. PrivateLessonGiver clicks on CreatePrivateLessonListingButton.
2. PrivateLessonGiver decides to give one or multiple private lessons; they also provide the name, price, tags, course codes and duration.

Entry Conditions:
- **Inherited** from PostItem use case.

Exit Conditions:
- **Inherited** from PostItem use case.

Quality Requirements:
- The price of the item can not be a negative number.

Use case name: PostLostItem
Participating Actors: Inherited from PostItem use case
Flow of Events:
1. LoserOfItem clicks on PostLostItemButton.
2. LoserOfItem enters the specifics of the lost item.

Entry Conditions:
- **Inherited** from PostItem use case.

Exit Conditions:
- **Inherited** from PostItem use case.

Use case name: PostFoundItem
Participating Actors: Inherited from PostItem use case
Flow of Events:
1. FinderOfItem clicks on PostFoundItemButton.
2. FinderOfItem enters the specifics of the lost item as well as found location and date.

Entry Conditions:
- **Inherited** from PostItem use case.

Exit Conditions:
- **Inherited** from PostItem use case.

Use case name: PostCourseTrade
Participating Actors: Inherited from PostItem use case
Flow of Events:
1. CourseTrader clicks on PostCourseTradeButton.

2. CourseTrader enters the course section to add and the course section to drop.
Entry Conditions:
- **Inherited** from PostItem use case.

Exit Conditions:
- **Inherited** from PostItem use case.

Use case name: Communicate

Participating Actors: Initiated by Seller, Customer, FinderOfItem, LoserOfItem, CourseTrader, PrivateLessonGiver, or PrivateLessonTaker
Flow of Events:
1. The Seller/Customer/FinderOfItem/LoserOfItem/CourseTrader/PrivateLessonGiver/PrivateLessonTaker clicks to the related Chat button.

    2. The system displays the Chat screen.

3. If the user is a Seller, they can communicate with the Customer, and the Customer can communicate with the Seller.

4. If the user is a FinderOfItem, they can communicate with the LoserOfItem, and the LoserOfItem can communicate with the FinderOfItem.

5. If the user is a CourseTrader, they can communicate with the other CourseTraders.

6. If the user is a PrivateLessonGiver, they can communicate with the PrivateLessonTaker, and the PrivateLessonTaker can communicate with the PrivateLessonGiver.

7. Otherwise, if the user changes their mind, they click the ExitButton and exit the Chat.

    8. The system receives the messages and sends them to the other user.

    9. Otherwise, if the user clicks the ExitButton, the system goes back to the last screen on the stack.

Entry Conditions:
- The user has a connection.
- The user is logged into the system.
- The user is in the related InspectPost screen.

Exit Conditions:
- The user communicates with another user, OR
- The user clicks the ExitButton and exits the Chat.

Quality Requirements:
- This use case **includes** the SendMessage and ReceiveMessage use cases. The SendMessage use case is initiated when the user clicks to send a message to another user. ReceiveMessage use case is initiated whenever a message is sent to another user.
- Messages should be delivered and sent in 5 seconds; in other words, Communicate 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 sender and receiver a communication environment without delays.

Use case name: CommunicateWithSeller
Participating Actors: Inherited from Communicate use case
Flow of Events:

1. Customer clicks on SendMessageToSellerButton on the related product's InspectPost screen.

    2. The system displays the screen where the Seller and Customer can chat.

3. Customer sends messages and asks questions to the Seller about that certain product, its price etc.

Entry Conditions:

- **Inherited** from Communicate use case.

Exit Conditions:

- **Inherited** from Communicate use case.

Use case name: CommunicateWithCustomer
Participating Actors: Inherited from Communicate use case
Flow of Events:

1. Seller clicks on ViewCustomersButton on the related product's InspectPost screen.
2. Seller chooses the Customer to communicate with.

    3. The system displays the screen where the Seller and Customer can chat.

4. Seller sends messages and answers questions about that certain product, its price etc.

Entry Conditions:

- **Inherited** from Communicate use case.

Exit Conditions:

- **Inherited** from Communicate use case.

Quality Requirements:

- In order for the Seller to communicate with Customer, it is required that the Customer has previously sent a message to that Seller about the product.

Use case name: CommunicateWithFinder
Participating Actors: Inherited from Communicate use case
Flow of Events:

1. LoserOfItem clicks on SendMessageToFinderButton on the related product's InspectPost screen.

    2. The system displays the screen where the LoserOfItem and FinderOfItem can chat.

3. LoserOfItem sends messages and asks questions to the FinderOfItem about that lost item.

Entry Conditions:

- **Inherited** from Communicate use case.

Exit Conditions:

- **Inherited** from Communicate use case.

Use case name: CommunicateWithLoser
Participating Actors: Inherited from Communicate use case
Flow of Events:

1. FinderOfItem clicks on ViewLosersButton on the related product's InspectPost screen.
2. FinderOfItem chooses the LoserOfItem to communicate with.
    3. The system displays the screen where the FinderOfItem and LoserOfItem can chat.
4. FinderOfItem sends messages and answers questions about that lost item.

Entry Conditions:

- **Inherited** from Communicate use case.

Exit Conditions:

- **Inherited** from Communicate use case.

Quality Requirements:

- In order for the FinderOfItem to communicate with LoserOfItem, it is required that the LoserOfItem has previously sent a message to that FinderOfItem about the product.

Use case name: CommunicateWithTrader
Participating Actors: Inherited from Communicate use case
Flow of Events:

1. CourseTrader clicks on ViewTradersButton on the related section's screen.
2. CourseTrader chooses the CourseTrader to communicate with.
    3. The system displays the screen where the CourseTraders can chat.
4. CourseTraders send messages about the course to trade.

Entry Conditions:

- **Inherited** from Communicate use case.

Exit Conditions:

- **Inherited** from Communicate use case.

Use case name: PostRequest
Participating Actors: Initiated by CourseTrader or PrivateLessonTaker
Flow of Events:

1. The CourseTrader or PrivateLessonTaker enters the related SendRequest screen.
    2. The system asks for confirmation.
3. If the CourseTrader/PrivateLessonTaker confirms, they send a request to the desired Course/PrivateLesson.
4. Otherwise, if the CourseTrader/PrivateLessonTaker changes their mind, they click CancelButton and cancel the request.
    5. The system notifies the CourseTrader(s)/PrivateLessonGiver about the request.
    6. Otherwise, if the CourseTrader/PrivateLessonTaker clicks CancelButton, the system navigates back to the previous screen.

Entry Conditions:

- The user has a connection.
- The user is logged into the system.
- If the user is a CourseTrader, they are in the CourseTrade page of the system.

- If the user is a PrivateLessonTaker, they are in the PrivateLessons page of the system.

Exit Conditions:

- The system takes the request and notifies the CourseTrader or PrivateLessonGiver about the request, OR
- The CourseTrader/PrivateLessonTaker clicks CancelButton, and the request gets cancelled.

Quality Requirements:

- The notification about the request should be sent to the related users instantaneously.

Use case name: PostPrivateLessonRequest
Participating Actors: Inherited from PostRequest use case
Flow of Events:

1. PrivateLessonTaker clicks on the desired PrivateLesson’s SendRequestButton.
2. The PrivateLessonTaker specifies their request by writing the expected duration, and the price that he/she is ready to pay for a private lesson.

Entry Conditions:

- **Inherited** from PostRequest use case.

Exit Conditions:

- **Inherited** from PostRequest use case.

Use case name: PostTradeRequest
Participating Actors: Inherited from PostRequest use case
Flow of Events:

1. CourseTrader specifies the section of the course that they want to trade and posts a request to other CourseTraders.

Entry Conditions:

- **Inherited** from PostRequest use case.

Exit Conditions:

- **Inherited** from PostRequest use case

.

Use case name: Rate
Participating Actors: Initiated by Customer, Seller, FinderOfItem or LoserOfItem
Flow of Events:

1. After the purchase or return of the lost item event occurs, the Customer/Seller/FinderOfItem/LoserOfItem clicks the RateButton.
2. The system provides a form to rate.
3. This use case **includes** the GivePoint use case. The users rate with giving points to the other user.
4. If the user is a Customer, they rate the Seller.
5. If the user is a Seller, they rate the Customer.
6. If the user is a LoserOfItem, they rate the FinderOfItem.

7. If the user is a FinderOfItem, they rate the LoserOfItem.

8. The system takes the data and uses it in order to calculate the rating of the specific user.

Entry Conditions:

- The user has a connection.
- The user is logged into the system.
- The user is in the related PurchasedItem/FoundItem screen.

Exit Conditions:

- The target user is rated and the given data is used in the rating of the target user.

Use case name: RateSeller
Participating Actors: Inherited from Rate use case
Flow of Events:

1. The Customer clicks the RateButton and rates the Seller according to the purchase process.

Entry Conditions:

- **Inherited** from Rate use case.

Exit Conditions:

- **Inherited** from Rate use case.

Use case name: RateCustomer
Participating Actors: Inherited from Rate use case
Flow of Events:

1. The Seller clicks the RateButton and rates the Customer according to the purchase process.

Entry Conditions:

- **Inherited** from Rate use case.

Exit Conditions:

- **Inherited** from Rate use case.

Use case name: RateLoser
Participating Actors: Inherited from Rate use case
Flow of Events:

1. The FinderOfItem clicks the RateButton and rates the LoserOfItem according to the lost and found process.

Entry Conditions:

- **Inherited** from Rate use case.

Exit Conditions:

- **Inherited** from Rate use case.

Use case name: RateFinder

Participating Actors: Inherited from Rate use case
Flow of Events:
1. The LoserOfItem clicks the RateButton and rates the FinderOfItem according to the lost and found process.
Entry Conditions:
- **Inherited** from Rate use case.

Exit Conditions:
- **Inherited** from Rate use case.

Use case name: ViewProfile
Participating Actors: Initiated by User, FinderOfItem, Seller, Customer or LoserOfItem
Flow of Events:
1. In the InspectPost screen of the related screen or the Chat screen or Account screen, the user clicks the desired profile.
    2. The system provides the profile of the user.
3. The user views the different ratings the inspected user has gotten from different users, and other specific information the user has specified about themselves.
4. If the user clicks Account, the User can view their own profile.
5. If the user is a Customer, they can view the Seller.
6. If the user is a Seller, they can view the Customer.
7. If the user is a LoserOfItem, they can view the FinderOfItem.
8. If the user is a FinderOfItem, they can view the LoserOfItem.

Entry Conditions:
- The user has a connection.
- The user is logged into the system.
- The user is in the related InspectPost screen, or the user is in the Chat screen or the Account screen.

Exit Conditions:
- The target user is viewed.

Use case name: ViewSeller
Participating Actors: Inherited from ViewProfile use case
Flow of Events:
1. Customer clicks the profile of the Seller.
2. Customer then enters the profile of the selected Seller and views the different ratings the Seller has gotten from different users, and other specific information Seller has specified.

Entry Conditions:
- **Inherited** from ViewProfile use case.

Exit Conditions:
- **Inherited** from ViewProfile use case.

Use case name: ViewCustomer
Participating Actors: Inherited from ViewProfile use case
Flow of Events:

1. Seller clicks the profile of the Customer.

2. Seller then enters the profile of the selected Customer and views the different ratings the Customer has gotten from different users, and other specific information Customer has specified.

Entry Conditions:

- **Inherited** from ViewProfile use case.

Exit Conditions:

- **Inherited** from ViewProfile use case.

Use case name: ViewFinder
Participating Actors: Inherited from ViewProfile use case
Flow of Events:

1. LoserOfItem clicks the profile of the FinderOfItem.

2. LoserOfItem then enters the profile of the selected FinderOfItem and views the different ratings the FinderOfItem has gotten from different users, and other specific information FinderOfItem has specified.

Entry Conditions:

- **Inherited** from ViewProfile use case.

Exit Conditions:

- **Inherited** from ViewProfile use case.

Use case name: ViewLoser
Participating Actors: Inherited from ViewProfile use case
Flow of Events:

1. FinderOfItem clicks the profile of the LoserOfItem.

2. FinderOfItem then enters the profile of the selected LoserOfItem and views the different ratings the LoserOfItem has gotten from different users, and other specific information LoserOfItem has specified.

Entry Conditions:

- **Inherited** from ViewProfile use case.

Exit Conditions:

- **Inherited** from ViewProfile use case.

Use case name: ViewOwnProfile
Participating Actors: Inherited from ViewProfile use case
Flow of Events:

1. User clicks on Account.

2. User enters their profile and views information about themselves.

Entry Conditions:

- **Inherited** from ViewProfile use case.

Exit Conditions:

- **Inherited** from ViewProfile use case.

The flow of events of the BilkentAuthenticator are omitted in order to enhance readability.

2.3.2 Class Diagram

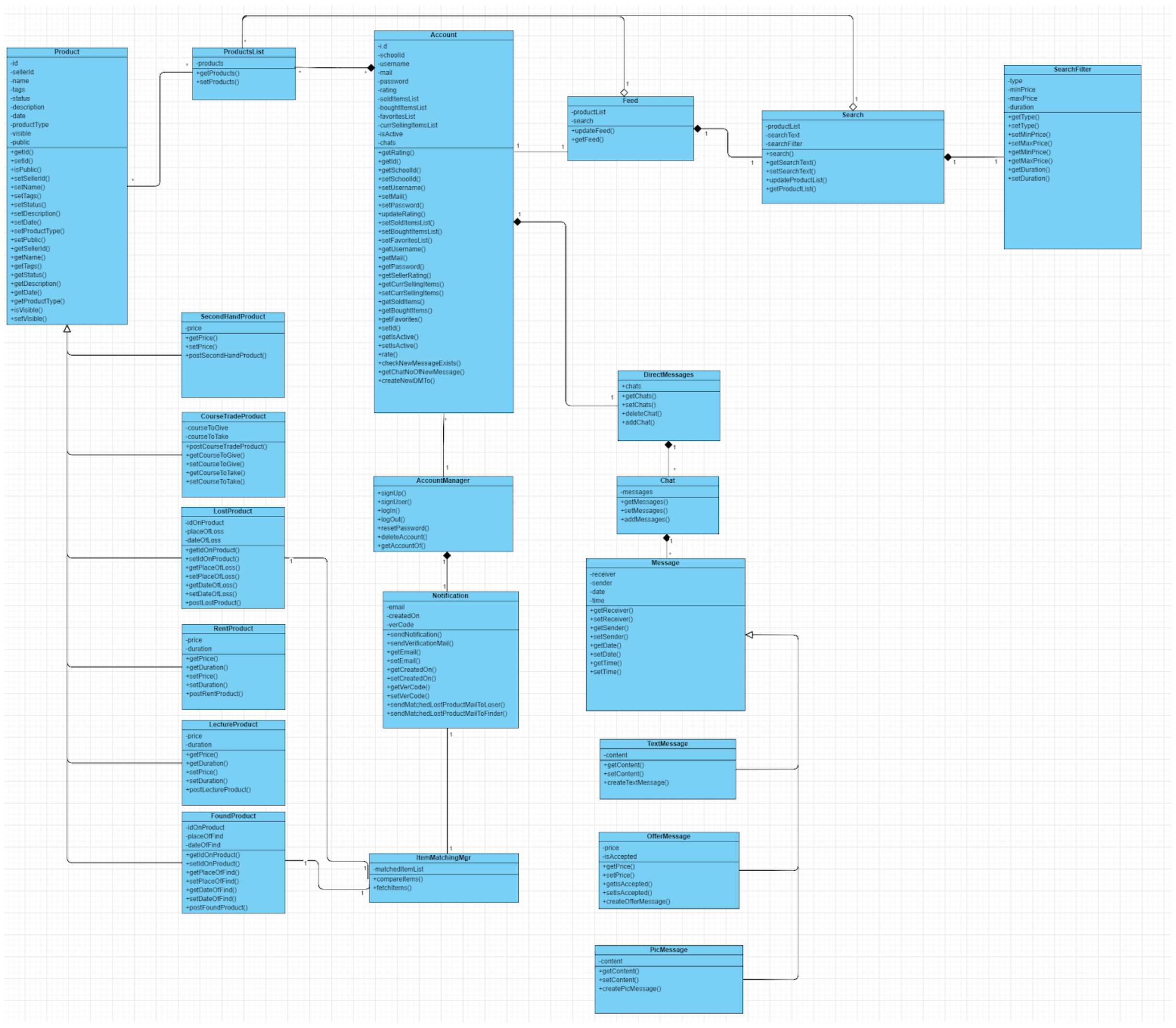


Fig. 2: Class Diagram for Bilcon Web Application

Product: The *Product* class serves as a pivotal cornerstone, encompassing an array of vital attributes that are essential in the world of commerce. Within its framework, you'll find critical data elements, including the unique product identifier, the seller's identification, the product's name, an array of descriptive tags, its current status, comprehensive product details, the date of publication, visibility preferences, and a discerning classification of its product type. This robust and versatile class forms the very backbone of the platform, providing a robust foundation for a wide range of products, and ensuring a dynamic and information-rich environment where buyers and sellers can interact with confidence and clarity.

SecondHandProduct: The *SecondHandProduct* class is a specialized child class derived from the broader *Product* class, tailored specifically for items that have previously been owned and are now being resold. In addition to inheriting all the attributes from its parent class, the *SecondHandProduct* class introduces an essential and distinguishing characteristic: the *price*. This new attribute reflects the cost at which the second-hand product is offered, providing potential buyers with crucial pricing information. By incorporating this extra attribute, the *SecondHandProduct* class caters to a unique subset of products in the marketplace, enhancing the platform's ability to facilitate the buying and selling of pre-owned items while maintaining consistency with the broader product data structure.

CourseTradeProduct: The *CourseTradeProduct* class represents a specialized child class derived from the parent *Product* class, designed to cater to the specific needs of users looking to exchange educational courses. In addition to inheriting all the attributes of the parent class, *CourseTradeProduct* introduces two pivotal attributes: *courseToGive* and *courseToTake*. These additional attributes enable users to specify the course they have available for trade and the course they are seeking in return. This distinct feature facilitates educational resource exchange on the platform, empowering users to connect and barter knowledge effectively. By introducing these extra attributes, *CourseTradeProduct* enhances the platform's functionality and serves as a valuable tool for users interested in skill development through course trading.

LostProduct: The *LostProduct* class is a specialized child class that extends the capabilities of the parent *Product* class, specifically tailored for instances where users need to report lost items. In addition to inheriting all the core attributes from its parent class, the *LostProduct* class introduces three crucial supplementary attributes: *idOnProduct*, *dateOfLoss*, and *placeOfLoss*. These extra attributes allow users to document essential information about the lost item, such as a unique identifier for the product, the date it was lost, and the specific location where the loss occurred. This unique feature equips the platform to efficiently assist users in tracking and potentially recovering their lost belongings, providing a valuable service while maintaining the platform's core product data structure.

RentProduct: The *RentProduct* class is a specialized child class derived from the broader *Product* class, customized to accommodate rental offerings. In addition to inheriting all the core attributes from its parent class, the *RentProduct* class introduces two vital supplementary attributes: *price* and *duration*. These additional features empower users to specify the cost associated with renting the product and the duration for which it is available for rent. This distinctive addition enhances the platform's capacity to facilitate rental transactions, offering users a convenient means to both list and rent items for a specified period. By introducing these extra attributes, the *RentProduct* class contributes to the platform's versatility, making it a valuable tool for users engaged in rental-related activities and transactions.

LectureProduct: The *LectureProduct* class is a specialized child class that extends the functionality of the parent *Product* class, catering specifically to the domain of educational resources. In addition to inheriting all the fundamental attributes from the parent class, the *LectureProduct* class introduces two key supplementary attributes: *price* and *duration*. These additional features enable users to specify the cost associated with accessing the lecture or educational content and the duration of the lecture itself. This specialized addition enriches the platform's capacity to host educational materials, making it a valuable resource for users looking to share and access educational content in an organized and transparent manner. By introducing these extra attributes, the *LectureProduct* class fosters a conducive environment for learning and knowledge sharing, aligning with the broader product data structure while catering to the unique needs of educational content providers and seekers.

FoundProducts: The *FoundProduct* class is a specialized child class that extends the functionality of the parent *Product* class, serving as a dedicated category for items that users have found and wish to report or return. In addition to inheriting all the core attributes from its parent class, the *FoundProduct* class introduces three essential supplementary attributes: *idOnProduct*, *dateOfFind*, and *placeOfFind*. These extra attributes allow users to document key information about the found item, including a unique identifier, the date it was discovered, and the specific location where it was found. This distinct feature enhances the platform's ability to facilitate the reporting and potential return of lost belongings, fostering a sense of community and goodwill among its users while retaining the core product data structure.

ProductList: The *ProductList* class is a fundamental class specifically designed to manage and organize lists of products within a broader system or application. Its primary attribute is *products*, which essentially serves as an array, collection, or container for storing and categorizing various products. This class allows for the aggregation and structured management of products, making it easier for users or the system to access and manipulate these product entries. Whether used for personal preferences like favorites or for administrative purposes in an e-commerce platform, the *ProductList* class is a versatile tool that streamlines product organization and retrieval, contributing to a more efficient and user-friendly experience within the system.

AccountManager: The *AccountManager* is a vital component within a system or application, designed to oversee user account-related functionalities. This class is responsible for handling crucial operations such as *signUp*, *logIn*, *logOut*, *resetPassword*, and *deleteAccount*. By incorporating these functionalities, the *AccountManager* streamlines the user experience by enabling users to create and access their accounts securely, recover their account access if needed, and opt for account removal when necessary. It plays a pivotal role in ensuring the security and accessibility of user accounts, providing a comprehensive set of tools to manage the user's interaction with the system, enhancing user satisfaction, and maintaining data integrity.

Notifiction: *Notification* is a crucial class within a system that handles the communication of important information to users. This class includes attributes such as *email*, *createdOn*, and *verificationCode*. The *email* attribute stores the recipient's email address, which serves as the destination for the notification. *createdOn* captures the date and time when the notification was generated, helping users track when the information was sent. The *verificationCode* attribute, on the other hand, plays a vital role in security and confirmation processes. It is often utilized to verify a user's identity or authorize certain actions, such as account registration or password reset, by providing a unique code. In sum, the *Notification* class is essential for efficient and secure communication, ensuring that the right information reaches the right recipient while also maintaining a record of when the notification was created and any necessary verification codes.

ItemMatchingMgr: The *ItemMatchingMgr* class is a critical component within a system, responsible for managing and facilitating the matching of items. Its primary attribute, *matchedItemList*, acts as a repository to store and organize product pairings or matches. This class plays a pivotal role in streamlining and automating the process of identifying and presenting related lost and found items to users, improving their experience by suggesting relevant products based on their preferences or previous interactions. The *ItemMatchingMgr* class not only enhances user engagement but also increases the efficiency of product discovery and recommendation, ultimately contributing to a more personalized and satisfying user experience within the system.

Account: *Account* is a fundamental class serving as a repository for essential user data. Within this class, one can find a comprehensive compilation of user information, including but not limited to their full names, email addresses, school IDs, unique user IDs, securely encrypted passwords, contact phone numbers, and user-generated ratings. Furthermore, the *Account* class also encompasses a repository for personal preferences, encapsulating product lists that cater to both the user's favorites and the items they have on offer in the realm of online commerce. This versatile class not only centralizes user data but also lays the foundation for a rich and interactive digital experience, facilitating seamless interactions and personalization within the platform.

Feed: The *Feed* class is a central element within a system, designed to curate and present a dynamic collection of content to users. It includes attributes such as *productList* and *search*, along with a crucial function called *updateFeed*. The *productList* attribute acts as a repository for a selection of products or items to be displayed to users, often tailored to their preferences or needs. The *search* attribute helps users narrow down and find specific items within the feed, enhancing their ability to discover products of interest. The *updateFeed* function is a key feature that enables real-time or periodic updates to the content displayed, ensuring that users receive fresh and relevant products. Overall, the *Feed* class plays a fundamental role in providing a curated and up-to-date stream of content, improving user engagement and product discovery within the system.

Search: The *Search* class serves as a fundamental component within a system, dedicated to facilitating effective product discovery and information retrieval. It encompasses several key attributes, including *productList*, *searchFilter*, and *searchText*, along with a critical function called *search*. The *productList* attribute functions as a container for storing and presenting search results, while *searchFilter* allows users to refine their search criteria, such as filtering by price, category, or other parameters. *searchText* captures the user's query or keywords, guiding the search process. The *search* function is the engine that orchestrates the search operation, matching user queries with relevant products and returning the results for display. The *Search* class plays a pivotal role in enhancing the system's functionality, making it easier for users to find specific products or information, ultimately improving their overall experience and usability within the platform.

SearchFilter: The *SearchFilter* class is an essential component designed to empower users in refining and customizing their search queries within a system or application. It incorporates key attributes such as *type*, *minPrice*, *maxPrice*, and *duration*. The *type* attribute allows users to specify the category or type of items they are interested in, providing a structured way to narrow down their search. *minPrice* and *maxPrice* enable users to set price range constraints, helping them to find products within their budget. *duration* is especially useful for time-sensitive searches, allowing users to define a specific timeframe for items with temporal relevance, such as rentals or events. In sum, the *SearchFilter* class enriches the search experience by offering users the flexibility to fine-tune their queries, making it easier to discover precisely what they are looking for and enhancing the overall usability and satisfaction of the system.

DirectMessages: The *DirectMessages* class plays a pivotal role in enabling private and real-time communication within a system or application. Its primary attribute, *chat*, serves as a repository for storing individual or group conversations, allowing users to engage in direct messaging. The class also features two essential functions: *addChat* and *deleteChat*. The *addChat* function facilitates the initiation of new conversations, enabling users to connect and exchange messages with others. Conversely, the *deleteChat* function provides the means to remove or archive conversations when they are no longer needed. The *DirectMessages* class enhances the platform's capacity for private communication, ensuring that users can seamlessly connect, converse, and manage their messages, contributing to a more dynamic and interactive user experience.

Chat: The *Chat* class is a fundamental component within a messaging system, designed to facilitate real-time conversations between users. Its central attribute, *messages*, serves as a container to store and manage the messages exchanged within a chat. This class offers several critical functions: *addMessages* allows users to append new messages to the chat, ensuring the ongoing flow of conversation. *setMessages* provides the ability to update or replace existing messages in the chat, helping users manage their chat history. The *getMessages* function enables users to retrieve and view the messages within the chat, ensuring that they can access their conversation history. The *Chat* class is pivotal in enhancing communication capabilities, offering a structured means to send, receive, and manage messages, contributing to an efficient and seamless messaging experience within the system.

Message: The *Message* mother class serves as a foundational element for managing communication within a system or application, offering a structured framework for messages. It includes essential attributes such as *receiver*, *sender*, *date*, and *time*. The *receiver* and *sender* attributes capture the identities of the message's intended recipient and sender, facilitating the proper routing of messages. The *date* and *time* attributes record the moment when the message was sent, enabling users to maintain a chronological record of their conversations. The *Message* class is instrumental in ensuring orderly and efficient communication, providing the necessary information for messages to reach their destination while also keeping a comprehensive record of communication history for users. It underpins the functionality of messaging systems and is integral to the user experience.

TextMessage: The *TextMessage* class is a specialized child class of *Message* tailored to handle text-based communication. In addition to inheriting the essential attributes of its parent class, such as *receiver*, *sender*, *date*, and *time* from its parent class, the *TextMessage* class introduces a crucial additional attribute, *content*. This attribute is responsible for storing the actual textual message content, allowing users to exchange written information. Furthermore, the *TextMessage* class comes equipped with a dedicated functionality, *createTextMessage*, which enables users to compose and send text messages seamlessly. This function streamlines the process of crafting and delivering text-based messages, enhancing the efficiency of text communication within the system. By introducing these enhancements, the *TextMessage* class offers a tailored solution for handling and managing textual content within messages, providing a more comprehensive and user-friendly communication experience.

OfferMessage: The *OfferMessage* class is a specialized child class of *Message*, designed to facilitate offer-based communication within a messaging system. In addition to inheriting the core attributes like *receiver*, *sender*, *date*, and *time* from its parent class, the *OfferMessage* class introduces two critical supplementary attributes: *price* and *isAccepted*. The *price* attribute allows users to specify and negotiate the monetary value associated with an offer, while *isAccepted* serves as a flag to indicate whether the offer has been accepted or not. Moreover, the *OfferMessage* class includes a dedicated function called *createOfferMessage*. This function simplifies the process of composing and sending offer-related messages, streamlining negotiations and enhancing user interactions. By incorporating these additional attributes and functionality, the *OfferMessage* class is tailored to handle offer-based communication effectively, ensuring users can engage in transparent and organized negotiations while maintaining a record of important offer details. This specialization contributes to a more efficient and structured exchange of offers within the messaging system.

PicMessage: The *PicMessage* class is a specialized child class of the *Message* mother class, specifically designed for handling image-based communication. In addition to inheriting the foundational attributes such as *receiver*, *sender*, *date*, and *time* from its parent class, the *PicMessage* class introduces a crucial extra attribute: *content*. This attribute is responsible for storing and delivering image-based content, allowing users to share and receive pictures within their messages. Furthermore, the *PicMessage* class incorporates a dedicated functionality known as *createPicMessage*. This function simplifies the process of creating and sending image-based messages, making it easy for users to share visual content seamlessly. By introducing these enhancements, the *PicMessage* class streamlines image-based communication, enriching the user experience within the messaging system and ensuring that users can effortlessly exchange pictures and images as part of their conversations.

2.3.3 Dynamic Model

2.3.3.1 Activity Diagram

Below is the Activity Diagram for Bilcon Web Application and its Authentication System.

Link of the Diagram (Visual Paradigm Online Link):
https://lszibzhkon.de-05.visual-paradigm.com/rest/diagrams/shares/diagram/7dd60a6f-e10d-4b03-98a3-ba8d6ac25bcc/preview?p=1

ACTIVITY DIAGRAM OF BILCON WEB APP AND ITS AUTHENTICATOR

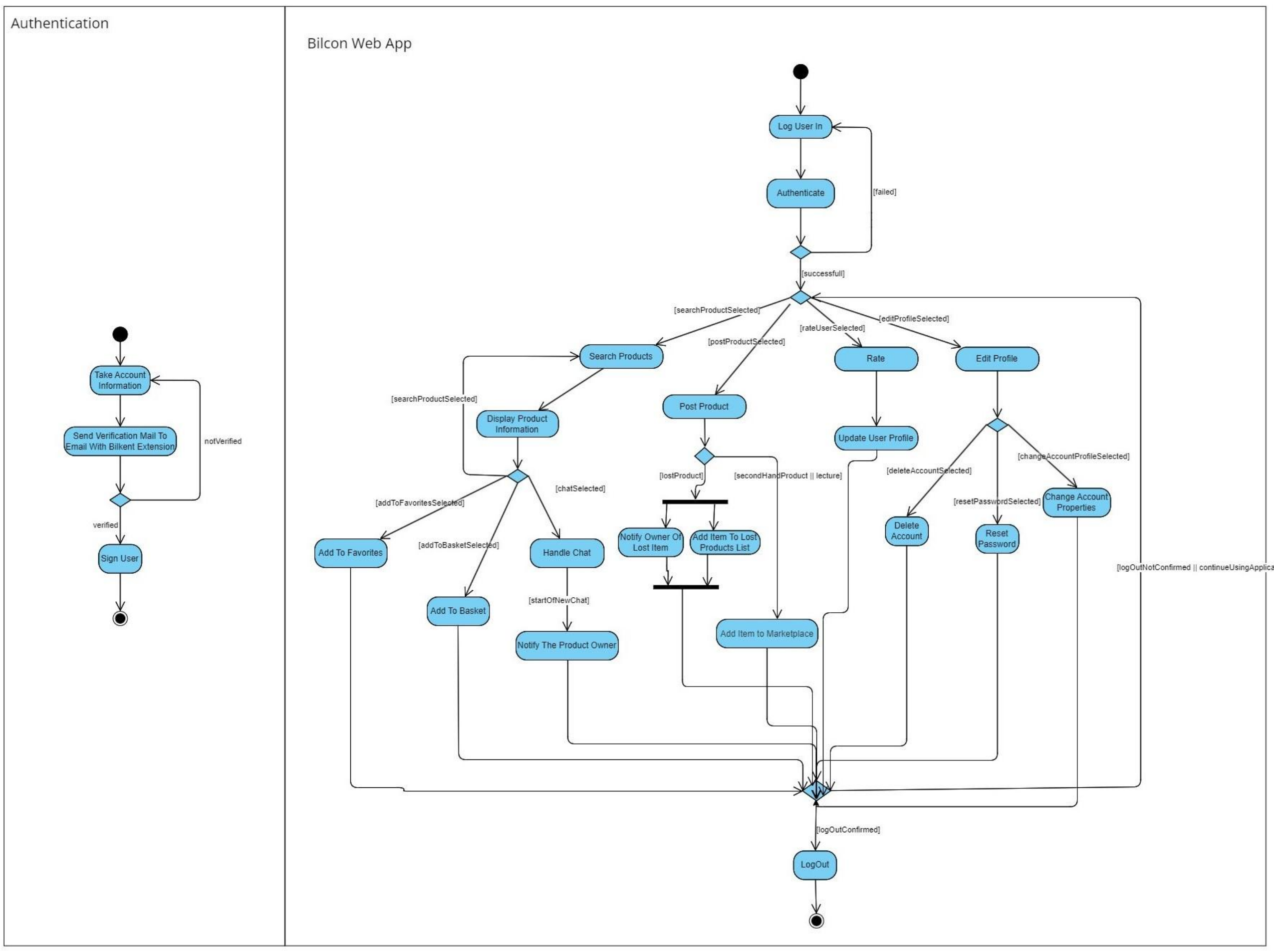


Fig. 3: Activity Diagram for Bilcon Web Application and its Authentication System

The first activity diagram shows the authentication system activities related signing up a user to Bilcon Web Application. The authentication system firstly takes the Bilkent member's account information (Bilkent email address, name, Bilkent ID). The system sends a verification mail to the Bilkent email address, if the signing up activity is verified by the user the systems signs the user to the application, if it is not verified, the system does not register the user.

The second activity diagram shows the activities of general Bilcon Web Application system. The application logs user in to the application by its authentication system. If the log in is successful, the user can search a product, post a product, rate a user, edit profile, and the system will act accordingly to the selection of the user. If search product is selected, the system displays products and their information to the user. The user can add the product to their favorites list, add it to their basket, chat with its seller or search for a product again. The system handles chat by starting a chat between the potential buyer and the product's seller and notifying them. If post product is selected, and if the product is a lost item, the system notifies the owner of the lost item, and at the same time, adds this product to the lost products list. If the product is a second hand product, the system adds it to the marketplace. If rate is selected by the user, the system updates the rated user profile. If the user chooses to edit their profile, the system handles the activities of the user such as changing account properties, resetting password and deleting the account. If the user chooses to log out of the application and confirms it, the system logs the user out. The user can continue to use the application if she/he does not log out.

2.3.3.2 Sequence Diagrams

Below are 3 sequence diagrams for 3 given scenarios. Link for each diagram is given for easier inspection.

1)

https://online.visual-paradigm.com/share/book/registersequencediagram-1k170qx4z3

Scenario name: Register to Bilcon

Participating Actors: User:User

Flow of events:
1. User tries to enter the system
2. User does not have an previous instance of account
3. User enters ID, mail and password for his/her account

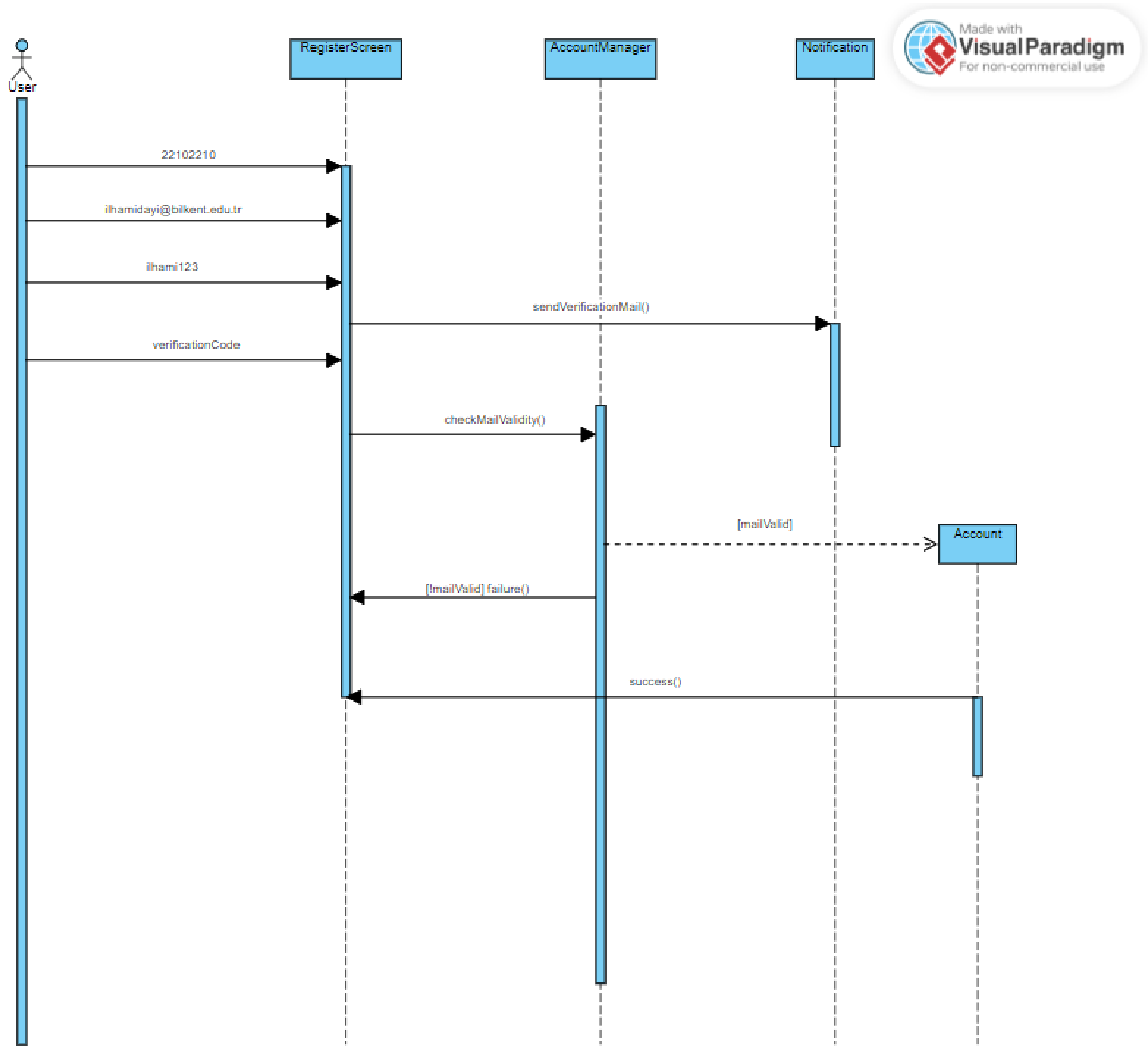


Fig. 4: Sequence Diagram for Bilcon Web Application Registration

2)

https://online.visual-paradigm.com/share/book/communicatewithsellersequence-1k1759mcl4

Scenario name: Communicate with seller

Participating Actors: Seller, Buyer:Account

Flow of events:
1.Buyer is interested in a product that the seller is actively selling within the system
2. Buyer decides to send a offer to the seller
3. Seller receives the offer
4. Seller accepts or declines the offer message
5. Buyer views the state of the sent message, that is, if it is accepted or declined.

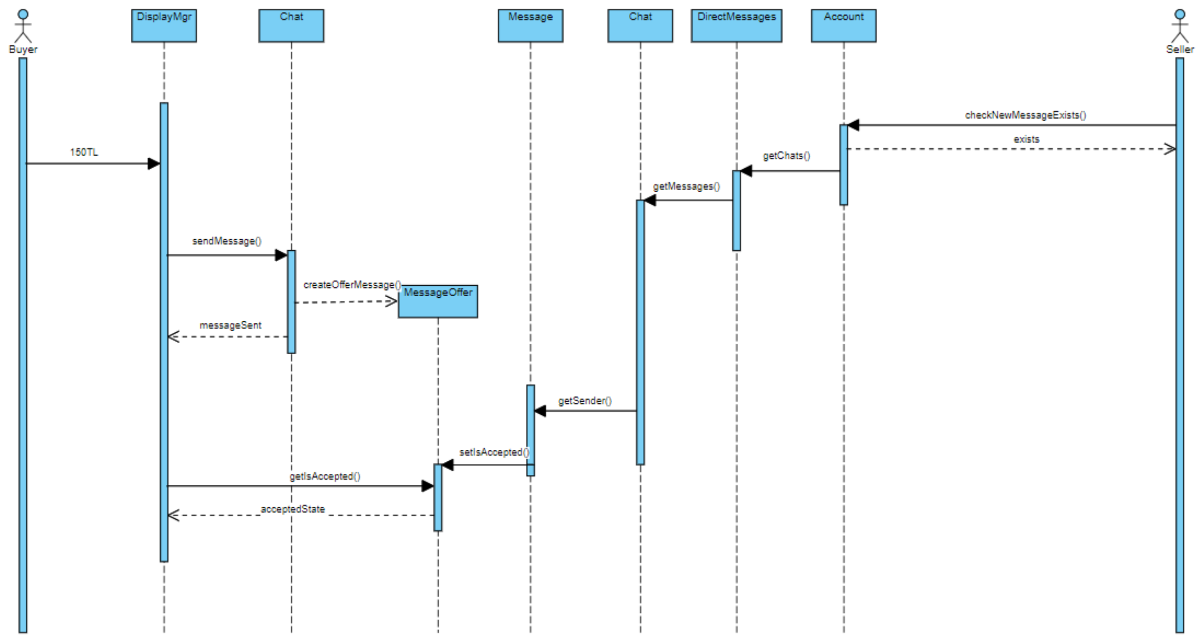


Fig. 5: Sequence Diagram for Communicating With Seller

3)

https://online.visual-paradigm.com/share/book/searchitemsequence-1k1r0mvey1

Scenario Name: searchCS223Book

Participating Actors: User: Customer

Flow of events:
1) User opens Bilcon in his web-browser and confronts the log in page. He logs into Bilcon with his email address: User@bilkent.edu.tr and password: ilhami123.
2) After logging in, User confronts the account page.
3) User goes to customer page from the account page.
4) User searches for CS 223 Book.
5) User confronts the related products.
6) User filters the search by setting the max price as 1000.
7) User confronts the filtered version of products.

Users can log in by entering their Bilkent mail and passwords, to the log in page. After that, they see their account page which consists of the information of the user. If they want to search for a good or service, they can go to customer page, search for an item by it's name and see the relevant items. Also, they can filter the search results by setting price floors or ceilings, or other properties.

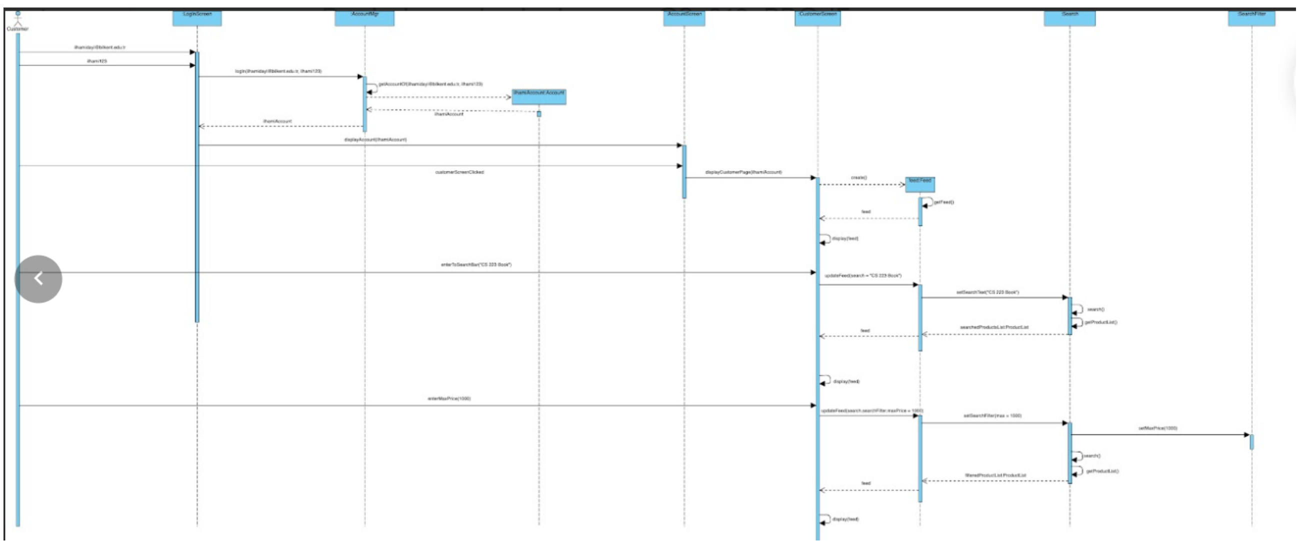

:LoginScreen
:AccountMgr
:AccountScreen
:CustomerScreen
:Search
:SearchFilter
Customer
ilhamiayi@bilkent.edu.tr
ilhami123
login(ilhamiayi@bilkent.edu.tr, ilhami123)
getAccountOf(ilhamiayi@bilkent.edu.tr, ilhami123)
ilhamiAccount:Account
ilhamiAccount
ilhamiAccount
displayAccount(ilhamiAccount)
customerScreenClicked
displayCustomerPage(ilhamiAccount)
create()
feed:Feed
getFeed()
feed
display(feed)
enterToSearchBar("CS 223 Book")
updateFeed(search = "CS 223 Book")
setSearchText("CS 223 Book")
search()
getProductList()
searchedProductsList:ProductList
feed
display(feed)
enterMaxPrice(1000)
updateFeed(search.searchFilter.maxPrice = 1000)
setSearchFilter(max = 1000)
setMaxPrice(1000)
search()
getProductList()
filteredProductList:ProductList
feed
display(feed)

Fig. 6: Sequence Diagram for Searching Second Hand Product

2.3.4 User Interface

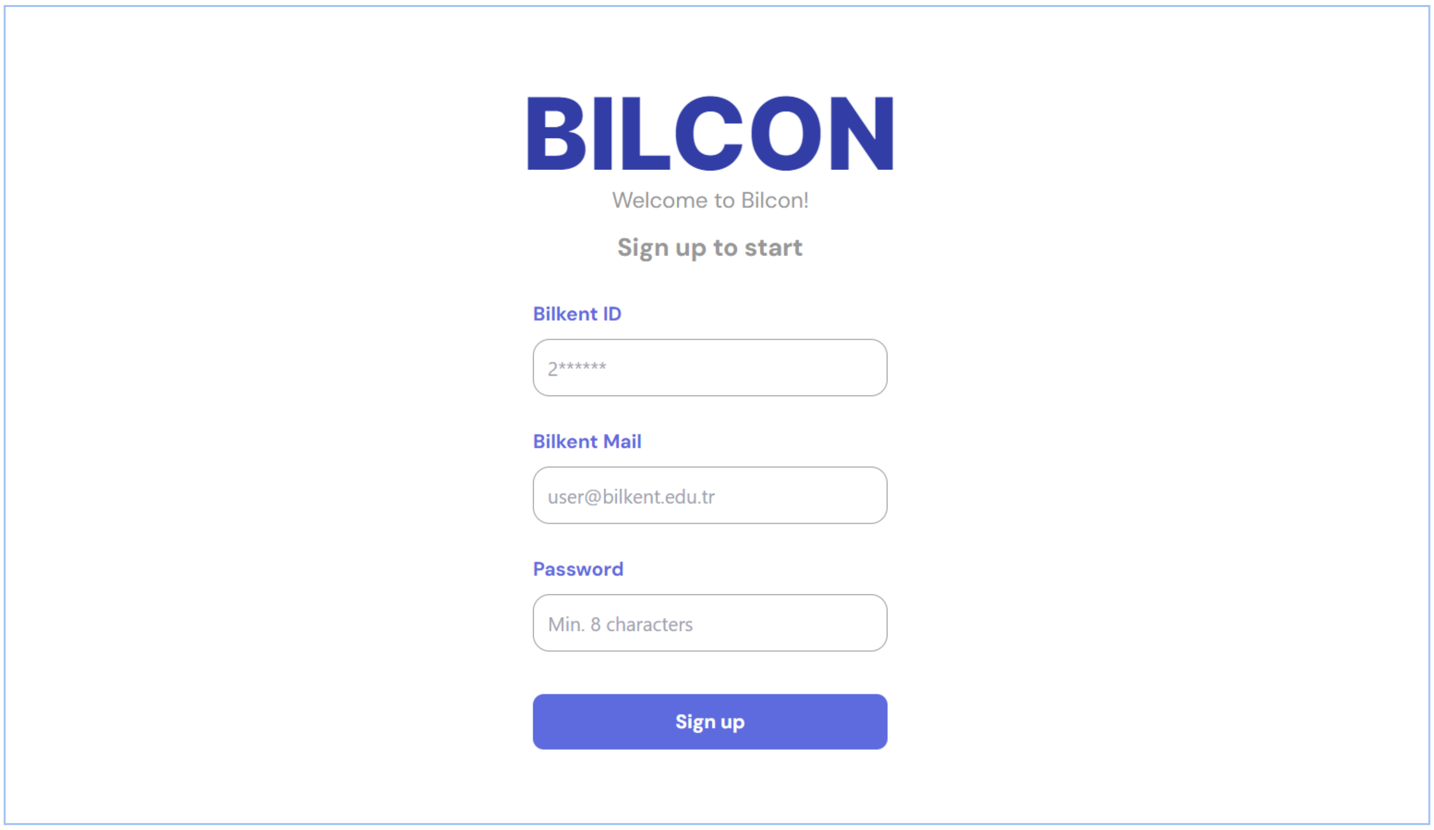


Fig. 7: Register Screen

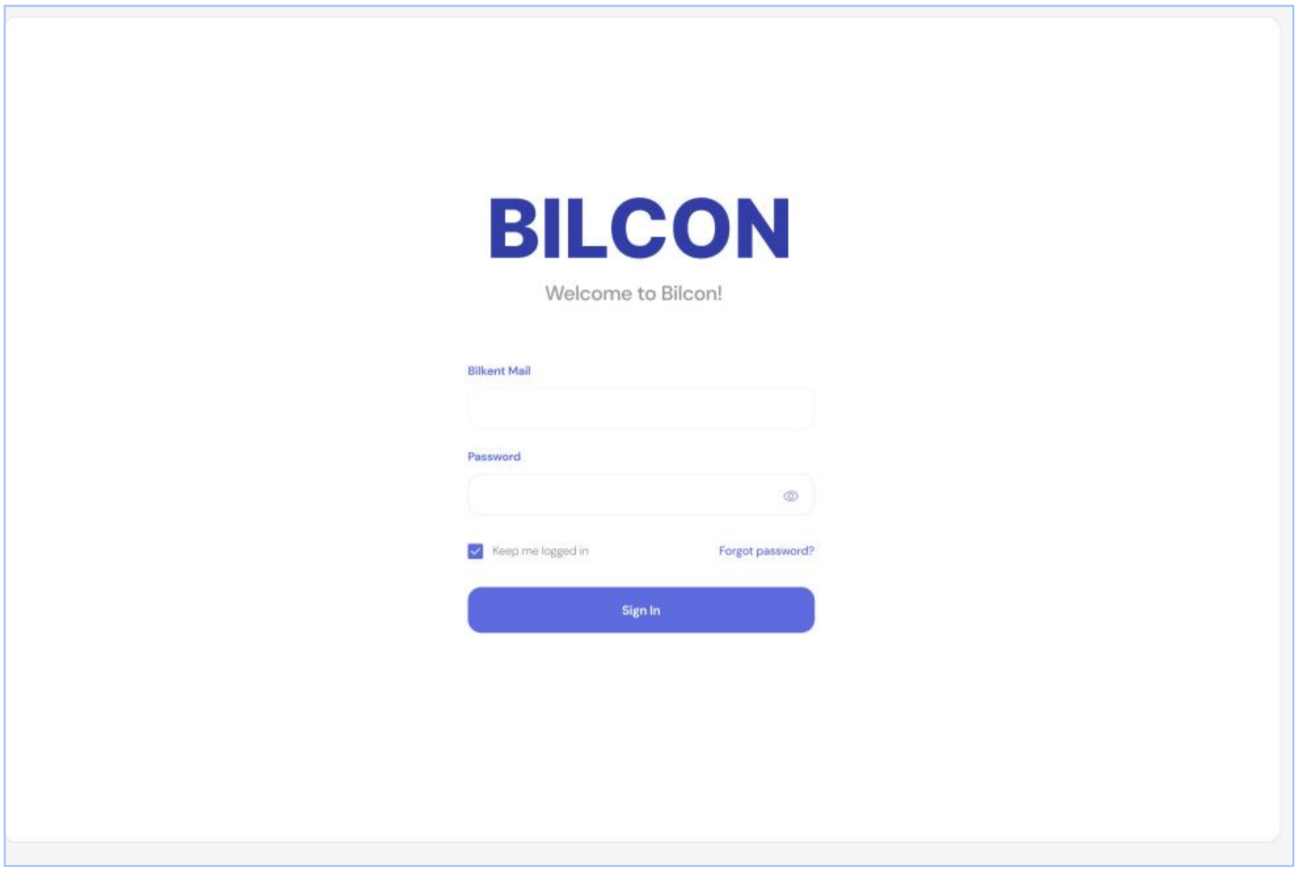


Fig. 8: Login Screen

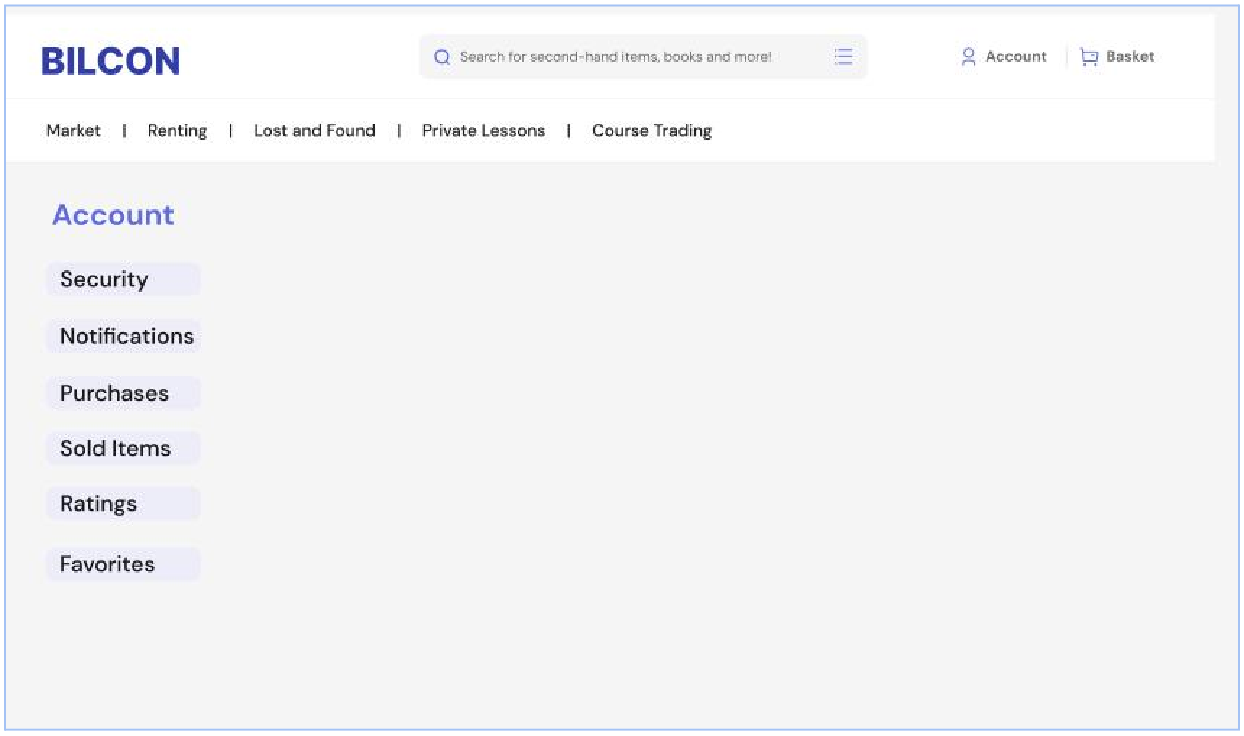


Fig. 9: Account Screen

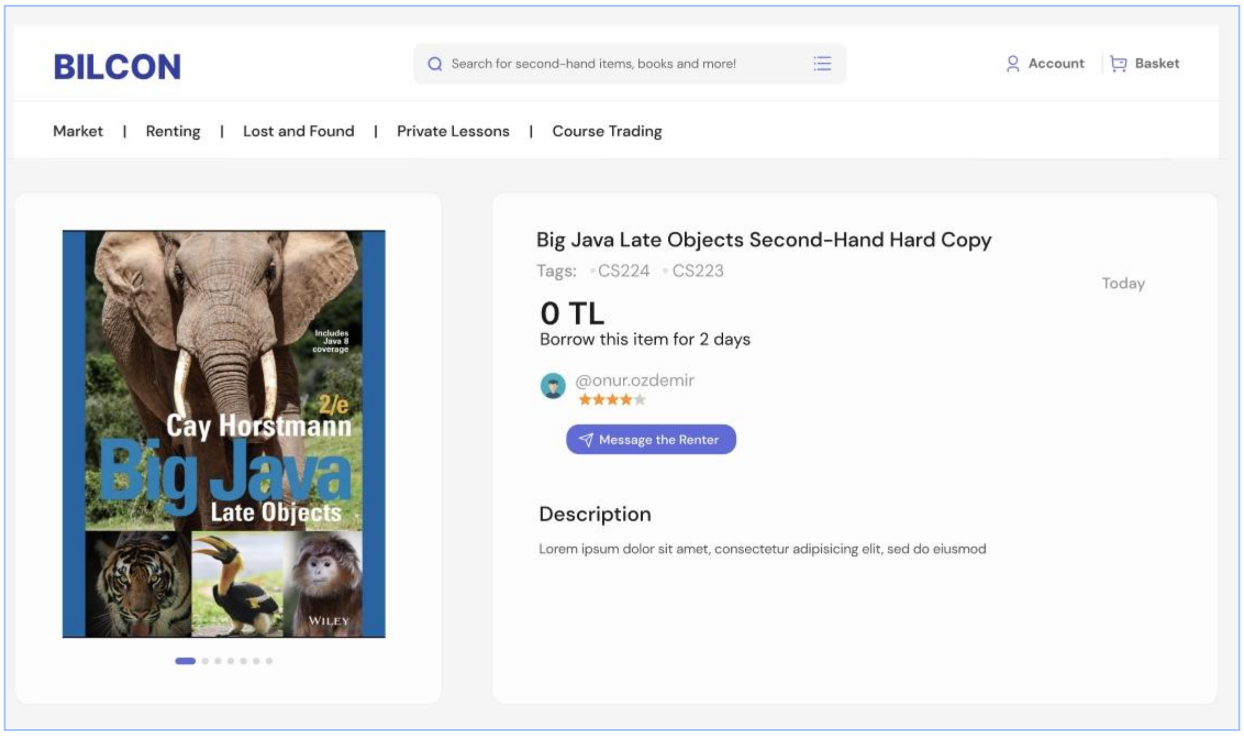

BILCON
Search for second-hand items, books and more!
Account
Basket
Market | Renting | Lost and Found | Private Lessons | Course Trading
Big Java Late Objects Second-Hand Hard Copy
Tags: CS224 CS223
Today
0 TL
Borrow this item for 2 days
@onur.ozdemir
Message the Renter
Description
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod

Fig. 10: Borrow Post Screen

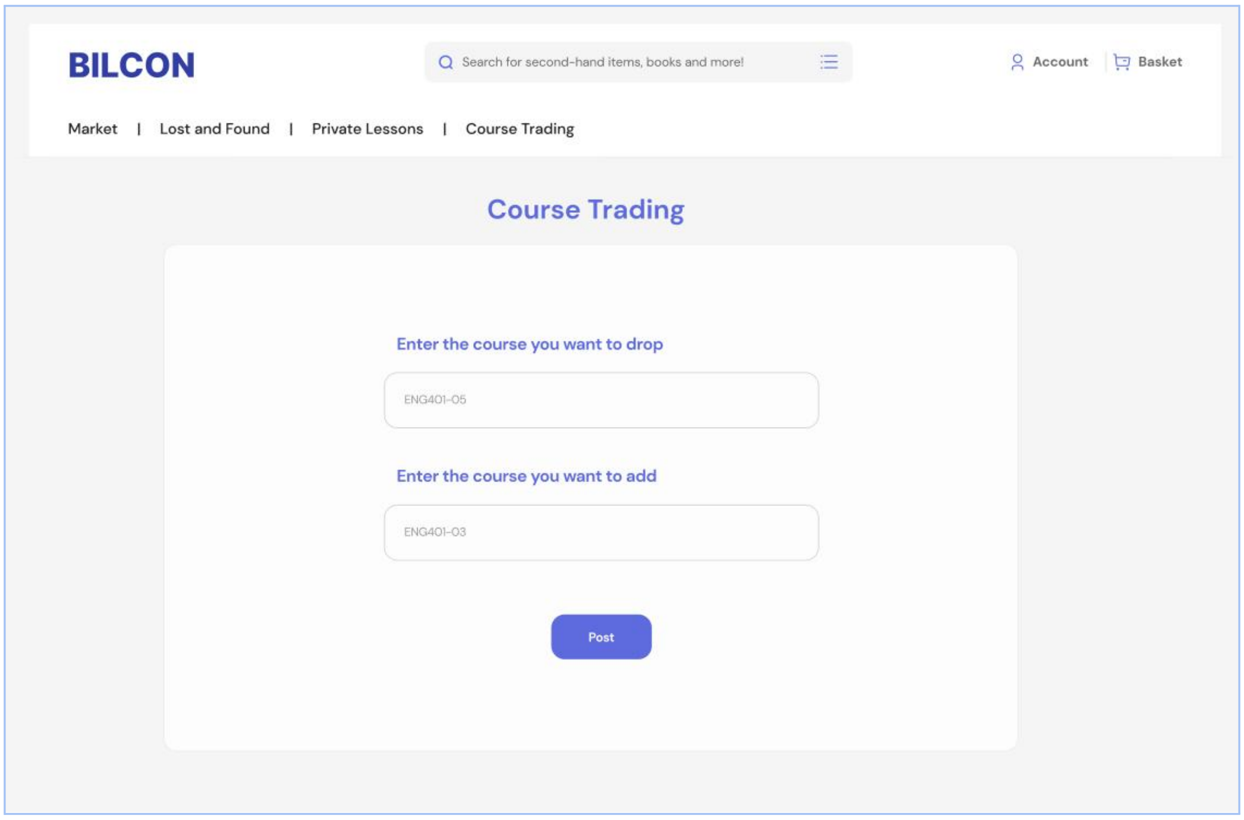
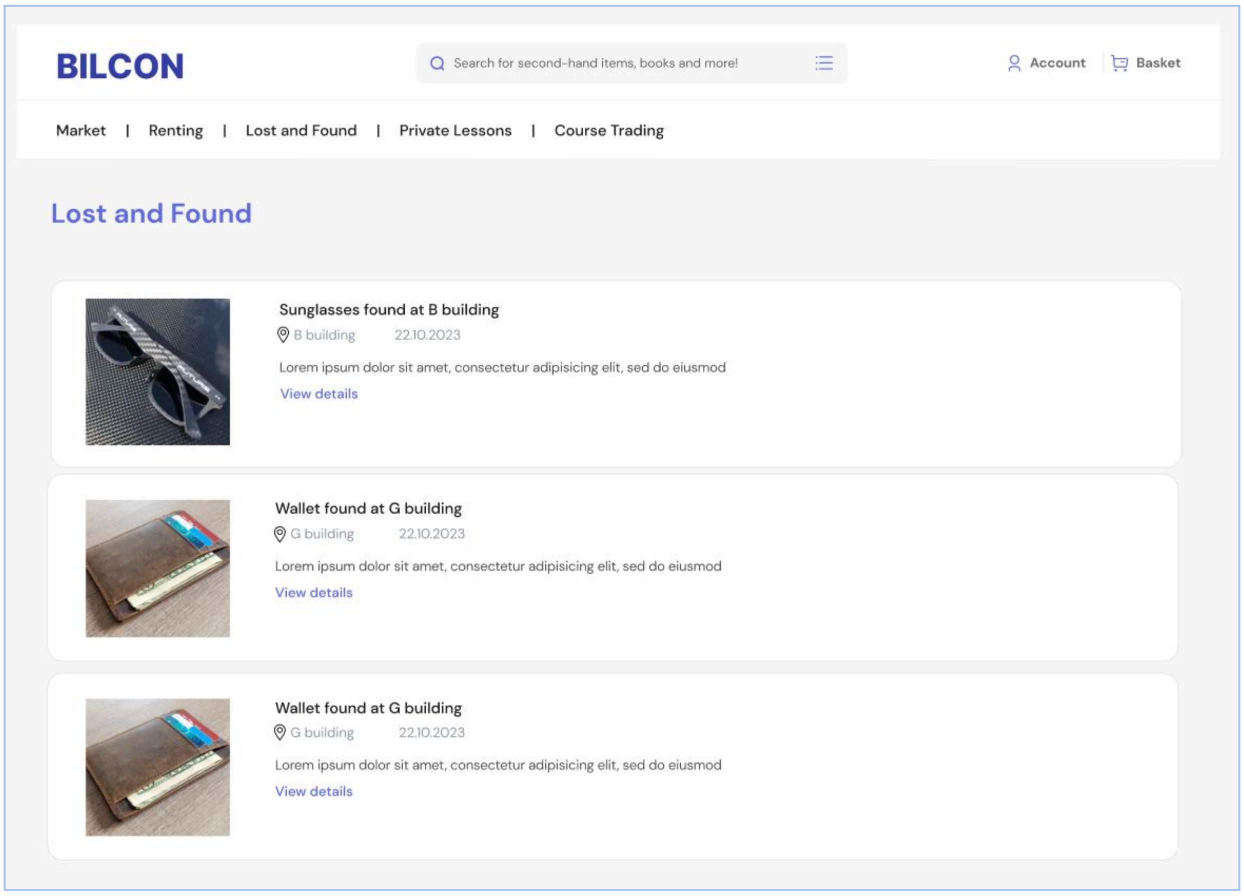

BILCON
Search for second-hand items, books and more!
Account
Basket
Market | Lost and Found | Private Lessons | Course Trading
Course Trading
Enter the course you want to drop
ENG401-05
Enter the course you want to add
ENG401-03
Post

Fig. 11: Course Trading Screen


BILCON

Search for second-hand items, books and more!

Account | Basket

Market | Renting | Lost and Found | Private Lessons | Course Trading

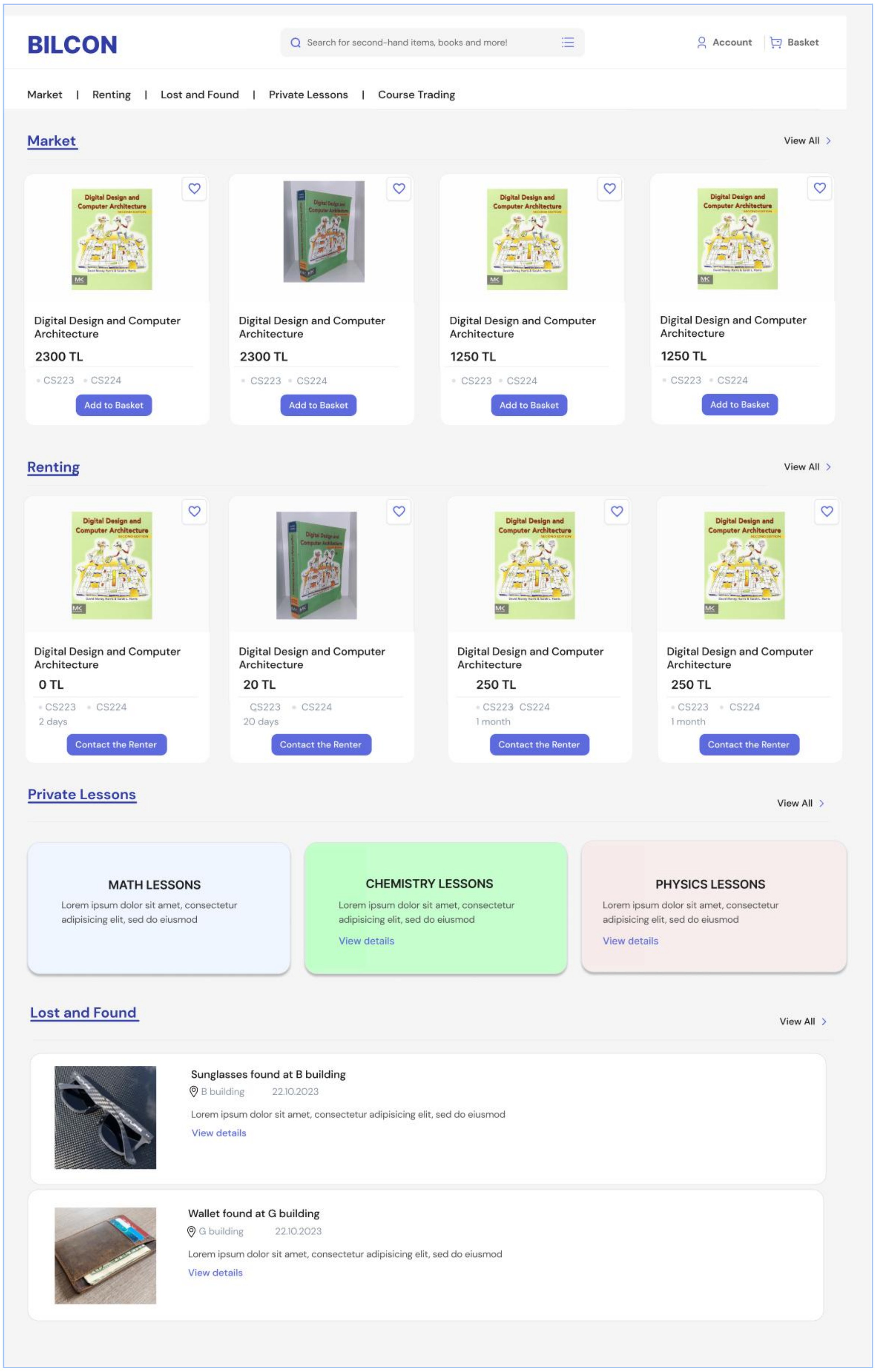
Lost and Found

Sunglasses found at B building
B building 22.10.2023
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod
View details

Wallet found at G building
G building 22.10.2023
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod
View details

Wallet found at G building
G building 22.10.2023
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod
View details

Fig. 12: Lost and Found Screen


BILCON
Search for second-hand items, books and more!
Account
Basket
Market | Renting | Lost and Found | Private Lessons | Course Trading

Market

View All

Digital Design and Computer Architecture	Digital Design and Computer Architecture	Digital Design and Computer Architecture	Digital Design and Computer Architecture
2300 TL	2300 TL	1250 TL	1250 TL
CS223 CS224	CS223 CS224	CS223 CS224	CS223 CS224
Add to Basket	Add to Basket	Add to Basket	Add to Basket

Renting

View All

Digital Design and Computer Architecture	Digital Design and Computer Architecture	Digital Design and Computer Architecture	Digital Design and Computer Architecture
0 TL	20 TL	250 TL	250 TL
CS223 CS224	CS223 CS224	CS223 CS224	CS223 CS224
2 days	20 days	1 month	1 month
Contact the Renter	Contact the Renter	Contact the Renter	Contact the Renter

Private Lessons

View All

MATH LESSONS

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod

CHEMISTRY LESSONS

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod

View details

PHYSICS LESSONS

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod

View details

Lost and Found

View All

Sunglasses found at B building

B building 22.10.2023

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod

View details

Wallet found at G building

G building 22.10.2023

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod

View details

Fig. 13: Home Screen

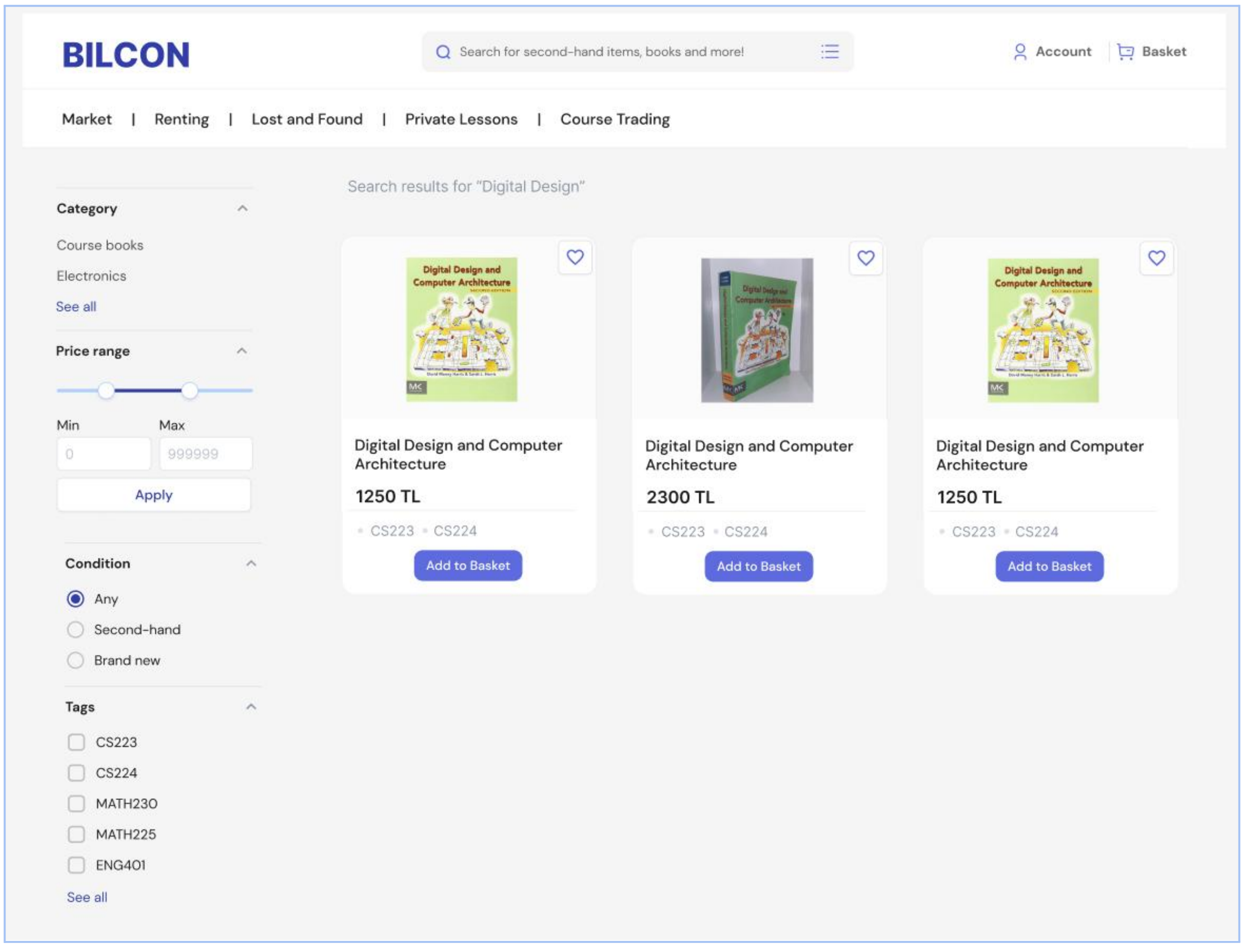

BILCON
Search for second-hand items, books and more!
Account
Basket
Market | Renting | Lost and Found | Private Lessons | Course Trading
Search results for "Digital Design"
Category
Course books
Electronics
See all
Price range
Min
Max
0
999999
Apply
Condition
Any
Second-hand
Brand new
Tags
CS223
CS224
MATH230
MATH225
ENG401
See all
Digital Design and Computer Architecture
1250 TL
CS223 • CS224
Add to Basket
Digital Design and Computer Architecture
2300 TL
CS223 • CS224
Add to Basket
Digital Design and Computer Architecture
1250 TL
CS223 • CS224
Add to Basket

Fig. 14: Market Search Results Screen

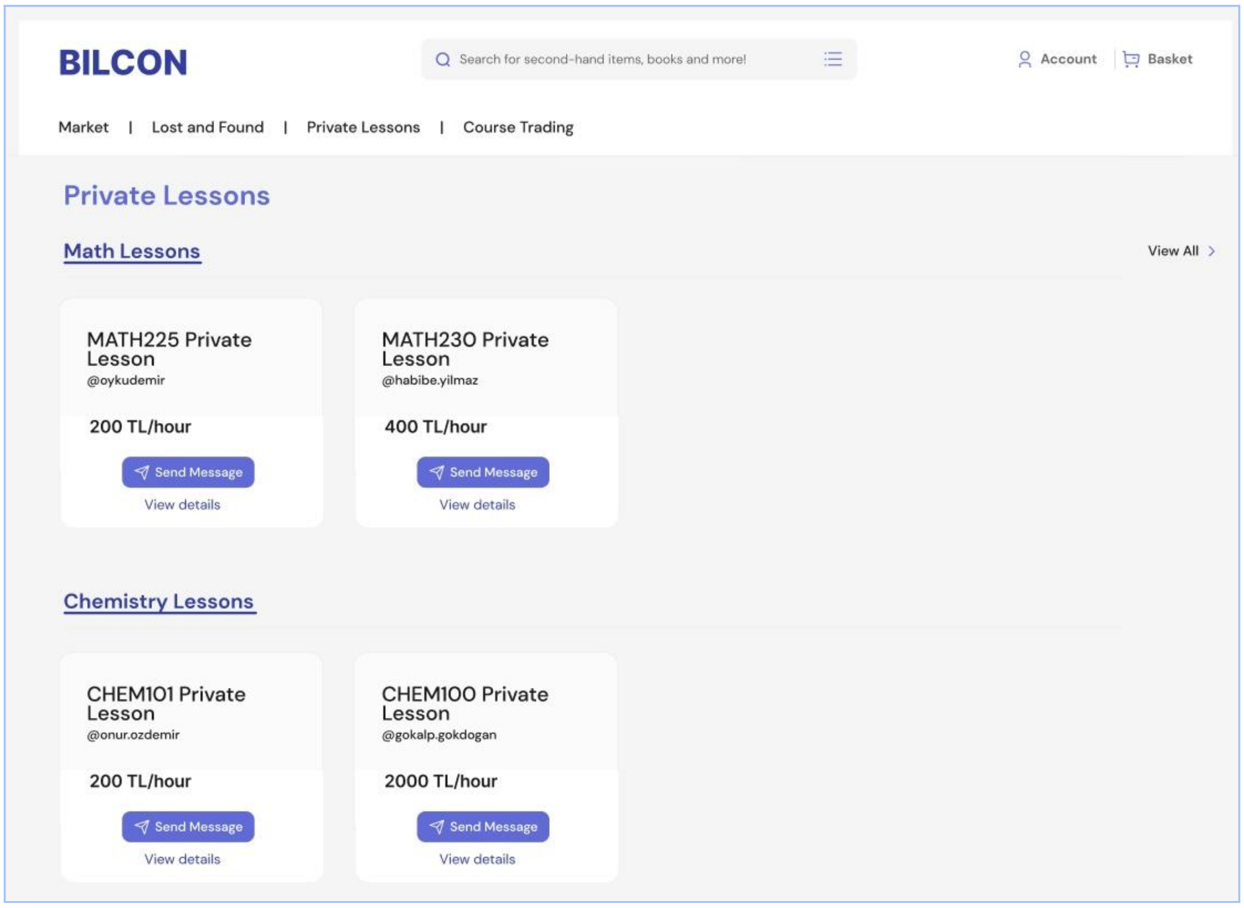
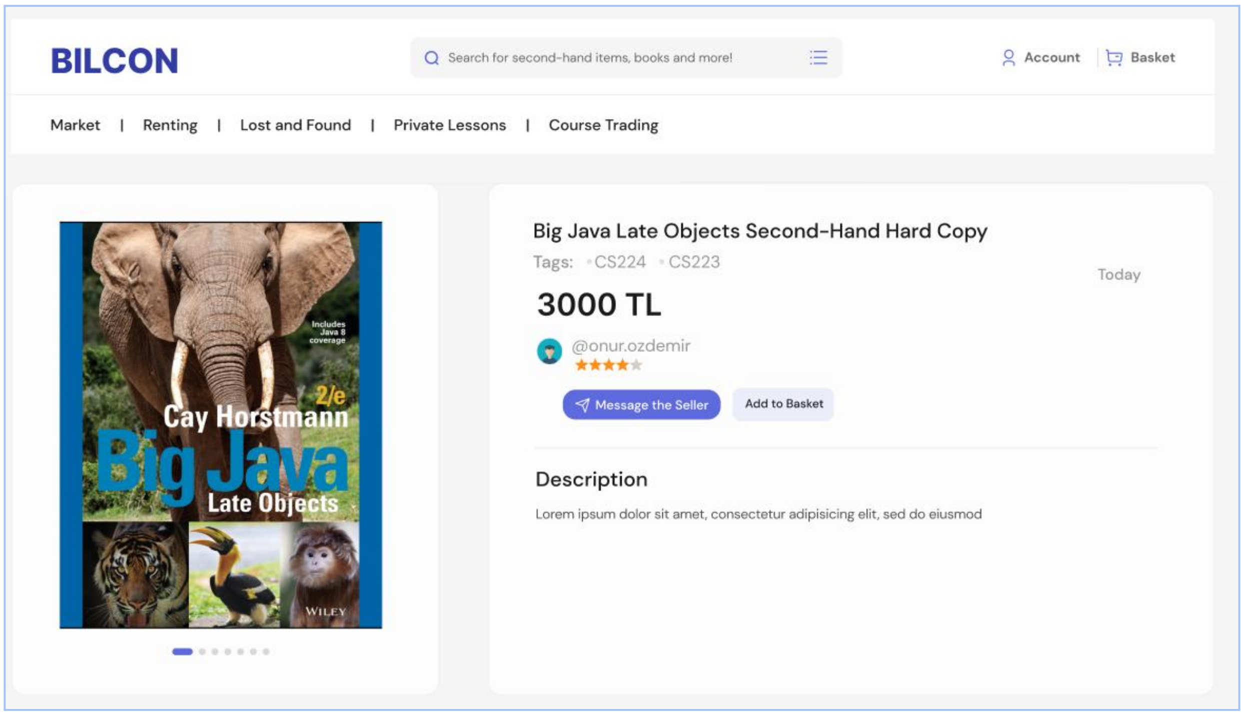


Fig. 15: Private Lessons Screen


BILCON
Search for second-hand items, books and more!
Account
Basket
Market | Renting | Lost and Found | Private Lessons | Course Trading
Message the Seller
Add to Basket

Big Java Late Objects Second-Hand Hard Copy

Tags: CS224 CS223

Today

3000 TL

@onur.ozdemir

Description

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod

Fig. 16: Market Post Screen

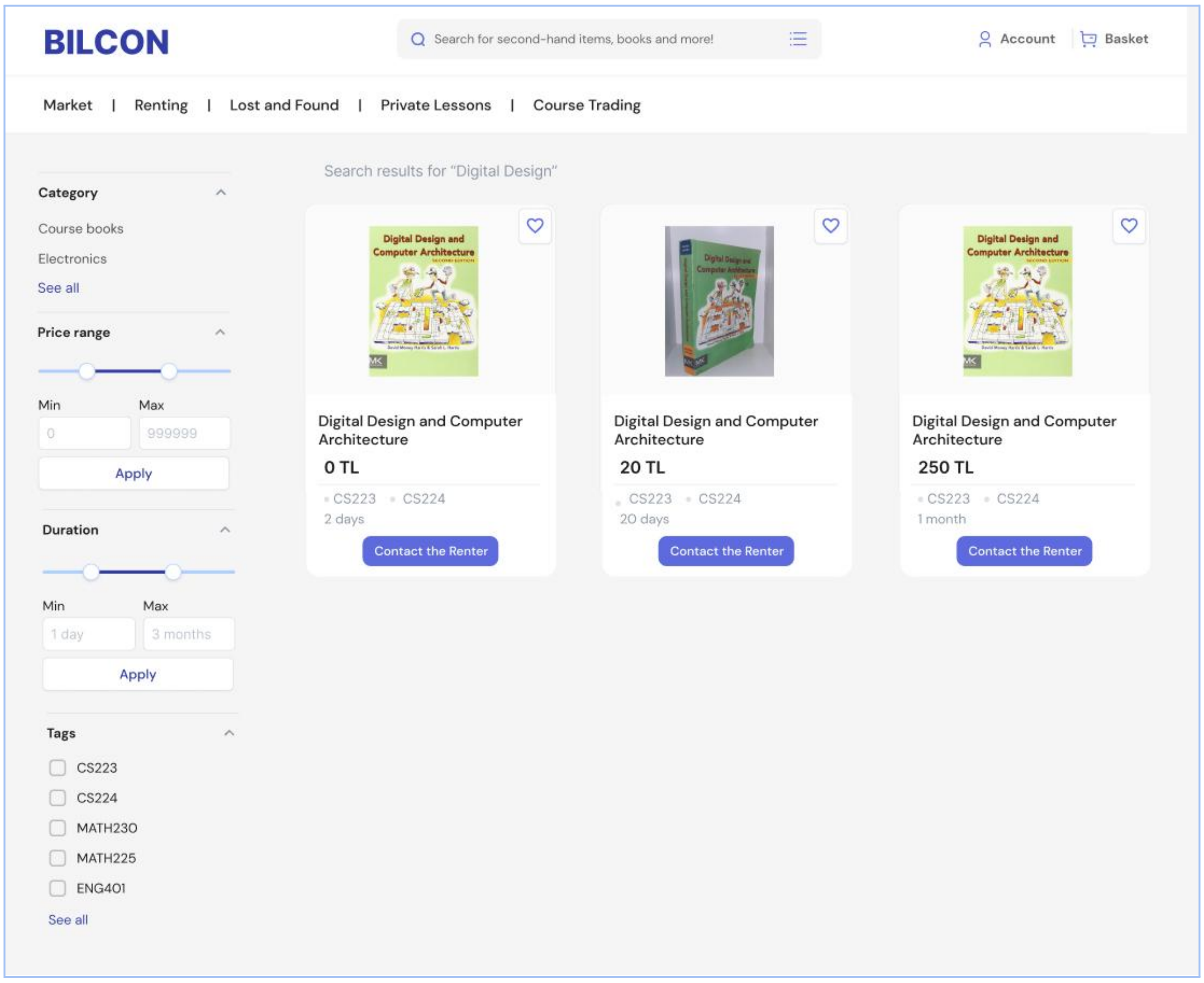


Fig. 17: Renting Search Results Screen