CS 319 - 2023 Fall Semester
S3T12: PAKAD

FINAL REPORT
CAMPUS CONNECT

Ahmet Kaan Sever - 22102278
Atakan Şerifoğlu - 22102313
Dila Tosun - 2212100
Konuralp Kılınç - 22101791
Perit Dinçer - 22103102
Selimhan Tokat - 22003145

1. Introduction:

The completed CampusConnect application offers Bilkent students to open an account with Bilkent email, log in, look through the listings made by other students and, if they wish to do so, put up their own item for sale, lend, donate or post a lost or found listing. Bilkent students or academic staff logged in with a Bilkent web-mail account are able to use all the functionalities of the application.

All important functions of the app are separated into their own pages and form. All listings on the app are visible through their respective dashboard access buttons, and the listings of the user are visible on their dashboard. Adding a listing is done through the respective page that lists it. In the settings page, users can change their password, logout or delete their account. Users can also see their previous messages or send new ones within the messages page.

CampusConnect offers a Bilkent internal e-commerce website with messaging, donation, sales, lending and many more features.

1.1. State of the System:

- The frontend and backend were successfully connected.
- The messaging client for real-time messaging was implemented.
- Payment thorough Iyzico was not implemented due to legal obstructions.
- All pages work as intended.
- The database for holding users, items and messages was implemented.

2. Lessons Learned

Our first challenge was learning how to work as a cohesive team. We wanted to share the workload equally without violating the time or efforts of the others. To this end, we decided to conduct weekly meetings for everyone to share their work, voice their concerns and ask for help in specific areas. This approach helped us a lot for upcoming development, as we were able to visualize how the frontend and backend would come together, how the class structure would work and how users would be able to navigate the app.

We also learned that it was important to determine a scope and stick to it. Initially we were overwhelmed with ideas on how to expand and improve the functionality of the app; but we quickly found out that was an unwise use of our time. So we decided on the core and most important features and focused on polishing them instead of adding new but janky features. Additionally, the tech stack we selected was not familiar to all of us, so we had to do a lot of learning; especially for React and .NET. We had to figure out how to portion our time to allow

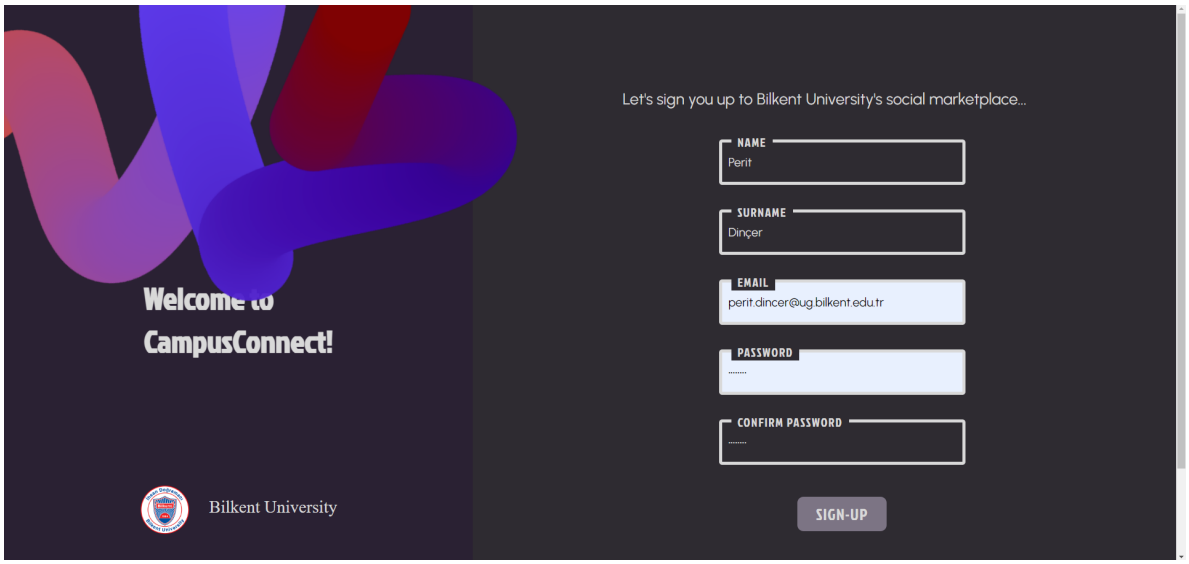
for both steady development and learning. While this took us some time to get right, we eventually did so.

We also had to learn the hard way on how work fills the time allocated to it (Parkinson's law). A lot of our time issues were due to us underestimating the complexity of what we had to implement, but we managed to overcome this hurdle towards the end of the development cycle.

3. User's Guide

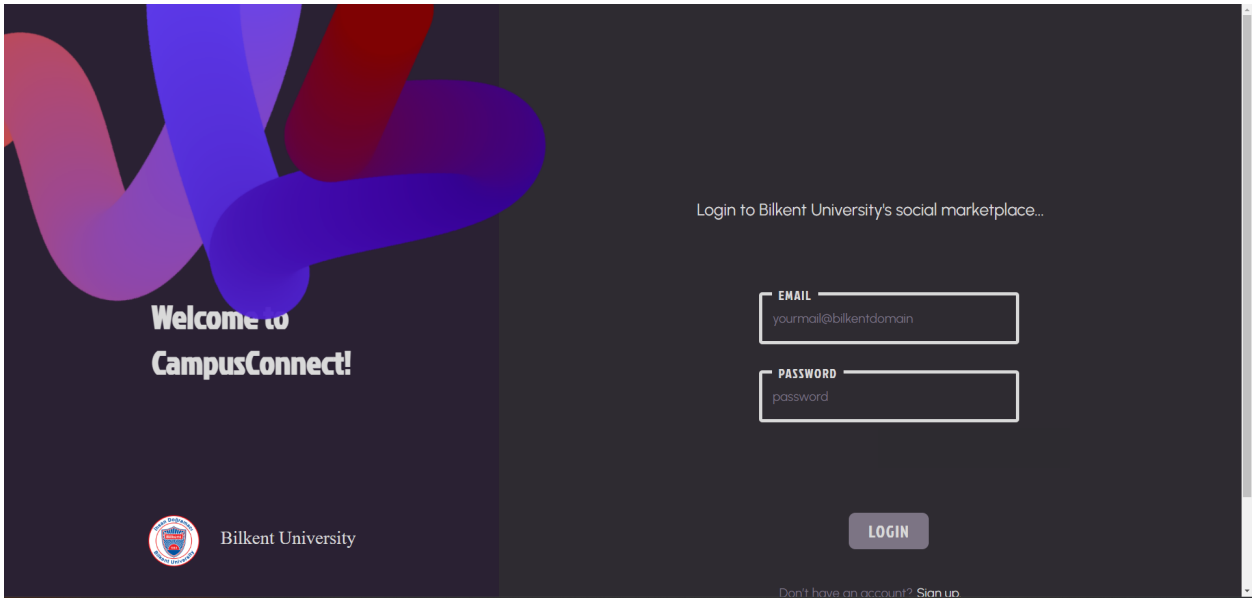
Signup Page:

On the signup page, users will be asked for their name, surname, email, password and password confirmation. After signing up, the user will be directed to the login page.



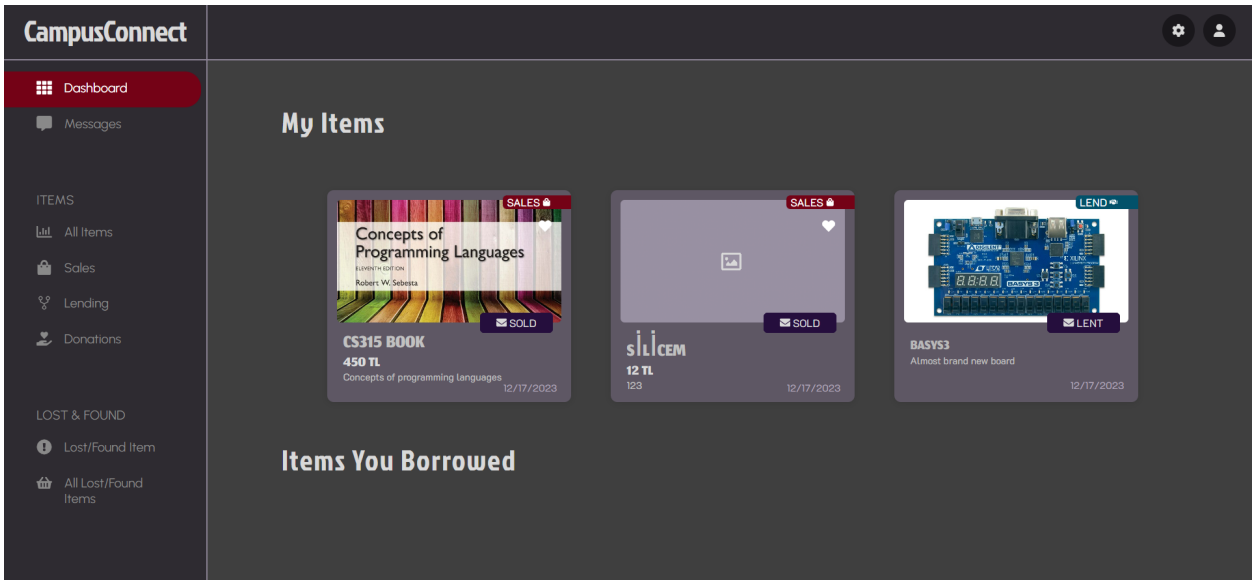
Login Page:

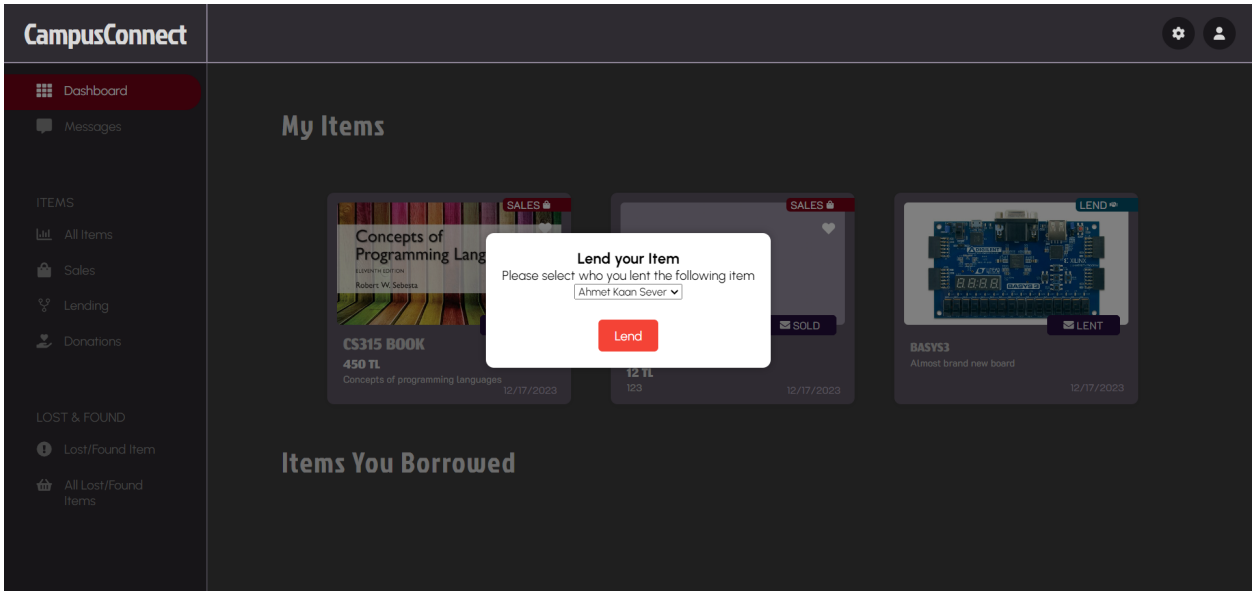
On the login page users will enter their email and password to gain access to the application.


Welcome to
CampusConnect!
Bilkent University
Login to Bilkent University's social marketplace...
EMAIL
yourmail@bilkentdomain
PASSWORD
password
LOGIN
Don't have an account? Sign up

Dashboard:

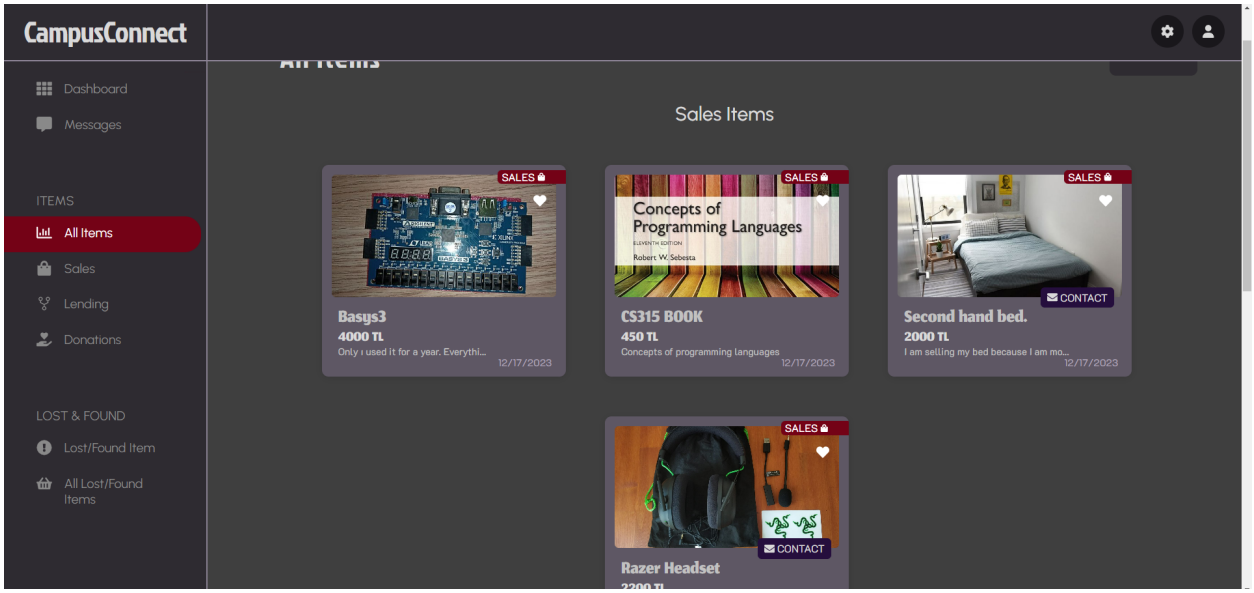
On this page, the user will see many different types of items; items they have listed to sell, donate, lend. The lost listings posted by the user are also here. The navbar that is always present on the right side of the screen also appears first on this page. Using the navbar, users can navigate to all the different pages of the application. The user can mark their items as sold, lended etc. on the cards. The items borrowed by the user are also visible below with their remaining days to return. Users can also press LEND on a item they have listed for lending to complete the lending process to someone in their messaging history. This sends an email notification to both the lender and the lendee; and also sends the lendee a notification when 1 day is left to return the item.





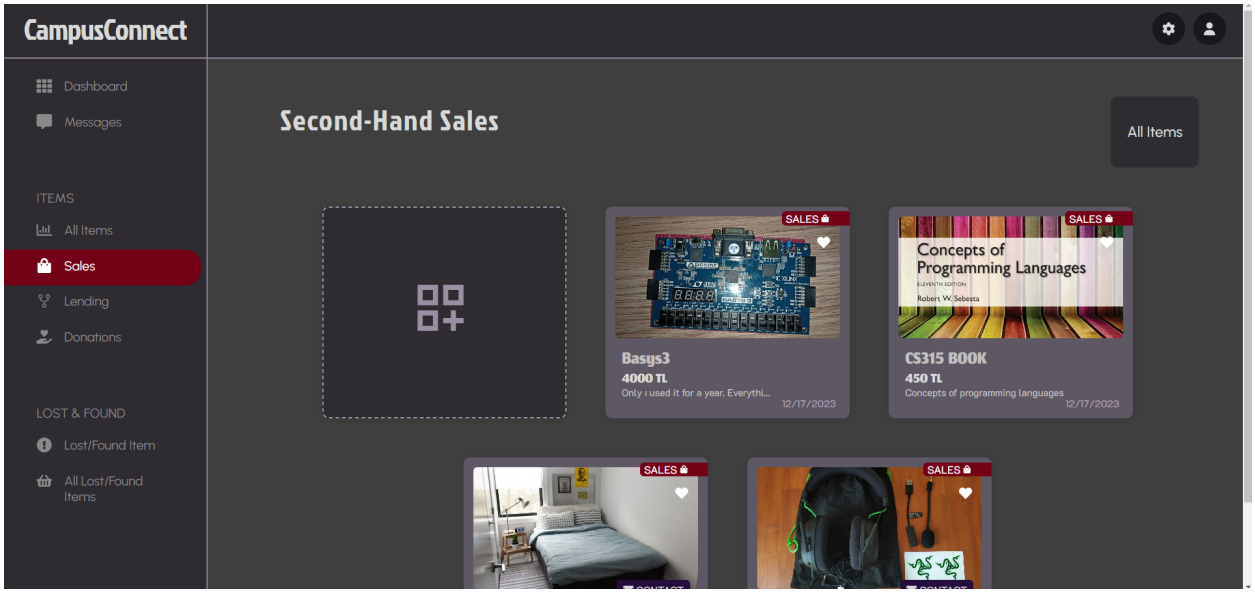
All Items/Main Page:

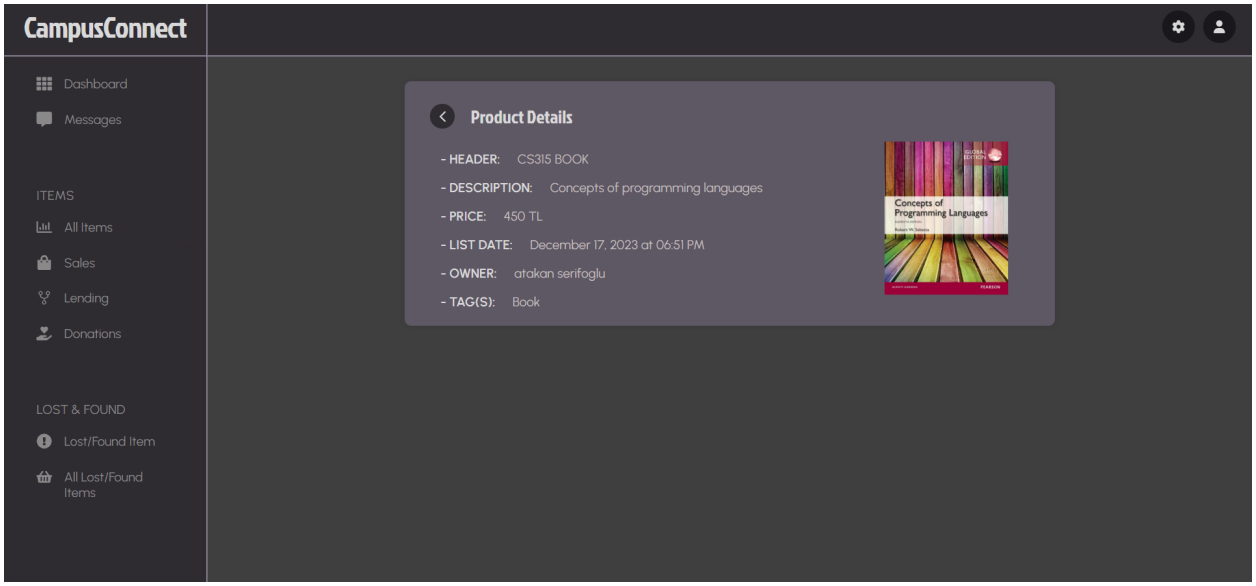
All item listings on the application will be available to the user here.



Sales Page:

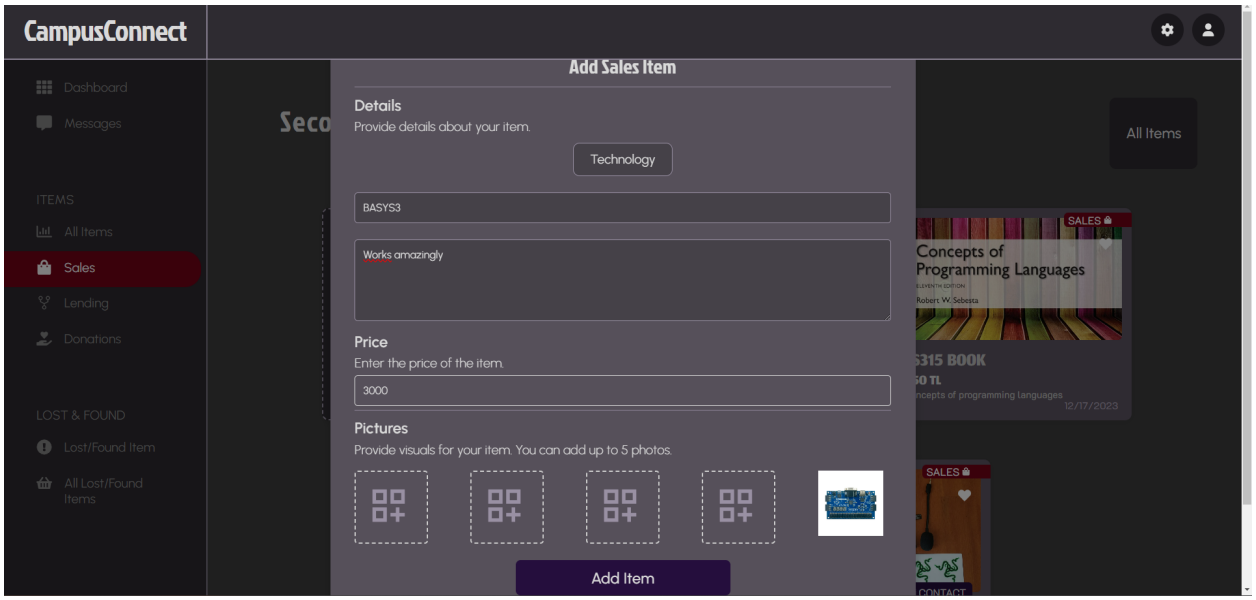
All the sale listings in the application are shown here. Users can click the + button to add a listing directly. By clicking on the cards, extra information about the item can be seen.





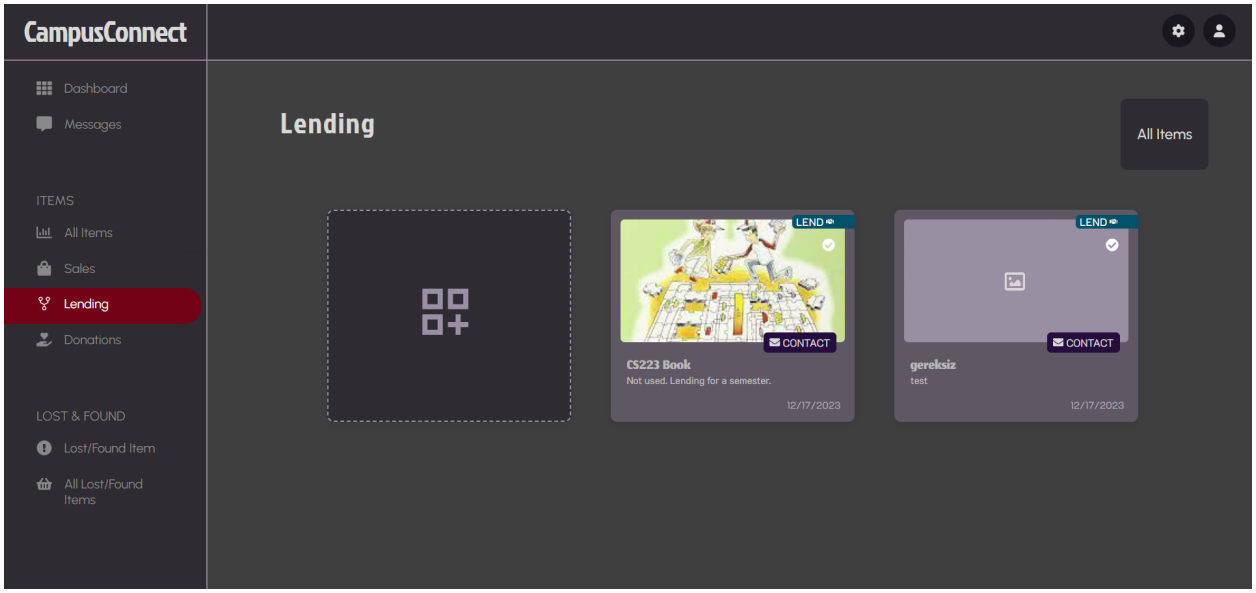
Sales Form:

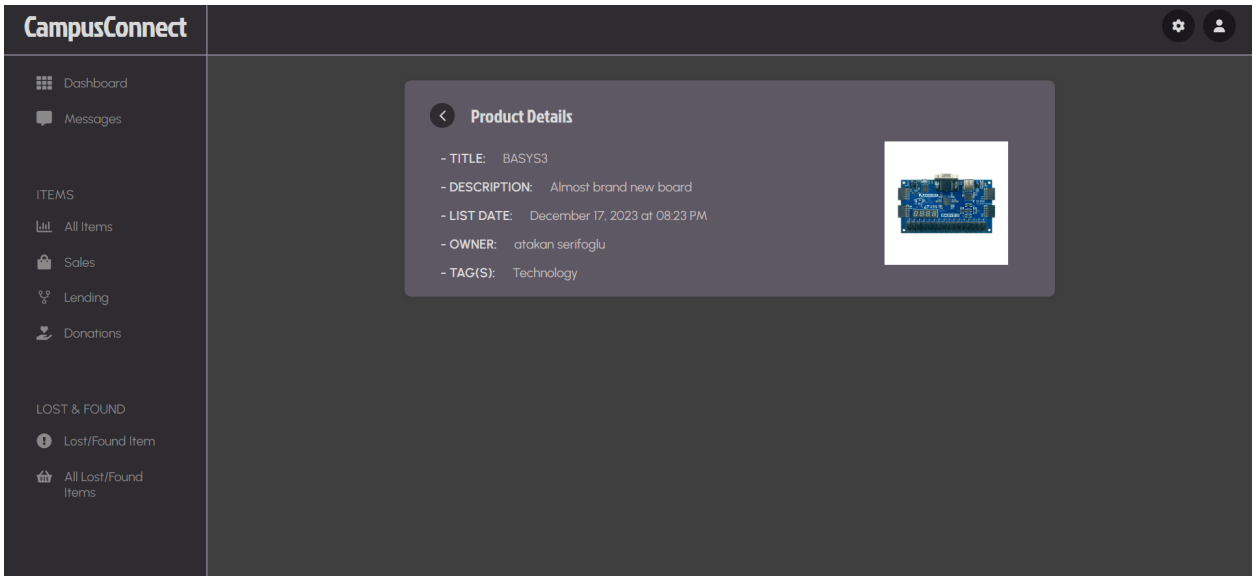
All the details for the item to be sold are entered by the user here. The name, description, type and the price of the item will be entered, along with 5 optional photos.



Lending Page:

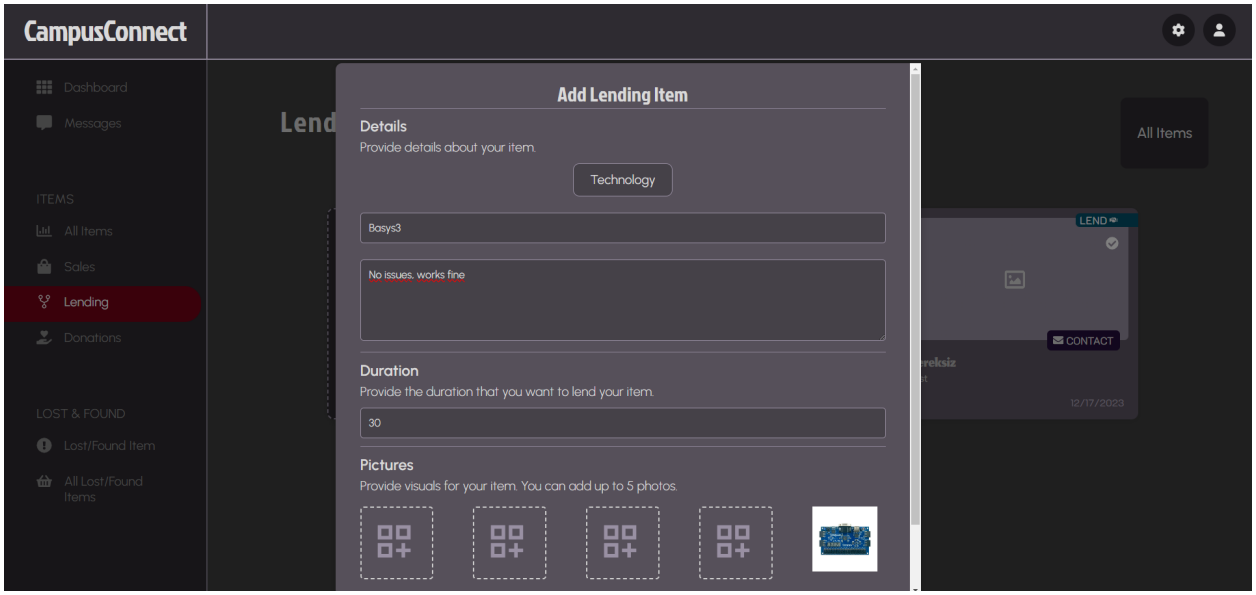
All the lending listings in the application are shown here. Users can click the + button to add a listing directly. By clicking on the cards, extra information about the item can be seen.





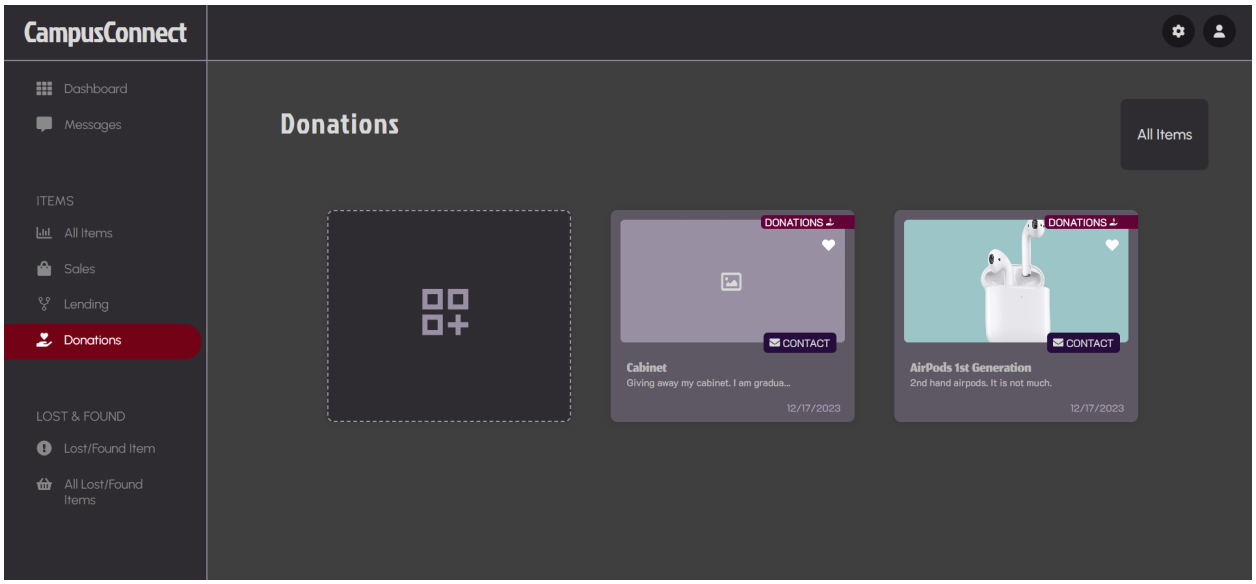
Lending Form:

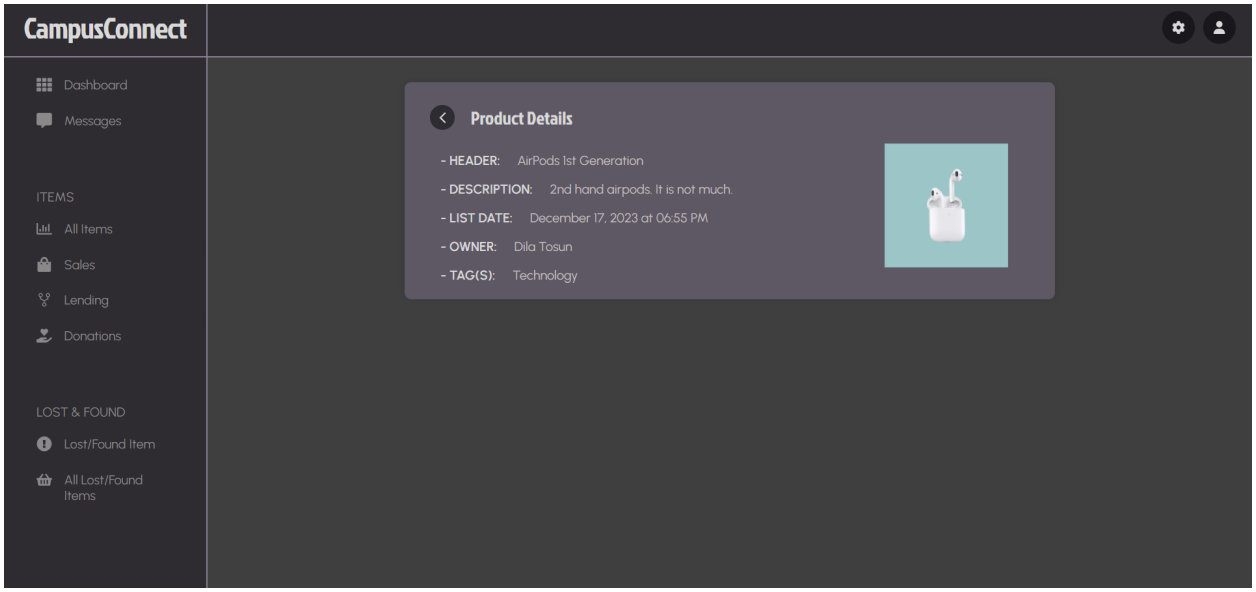
All the details for the item to be lended are entered by the user here. The name, description, type and the lending duration will be entered, along with 5 optional photos.



Donations Page:

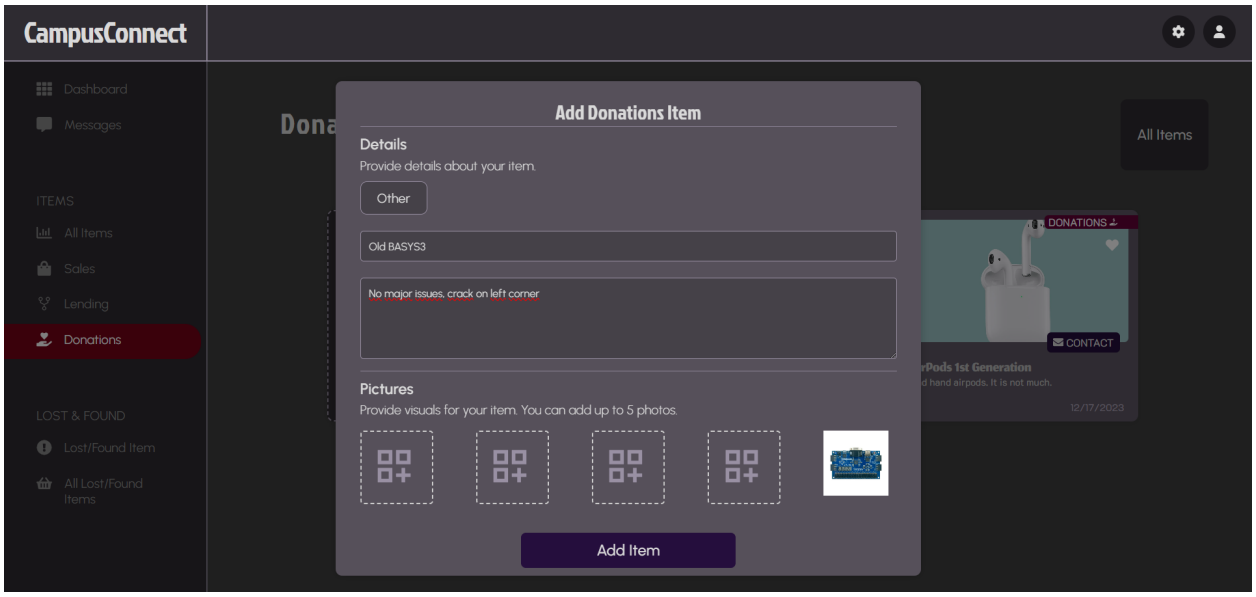
All the donation listings in the application are shown here. Users can click the + button to add a listing directly. By clicking on the cards, extra information about the item can be seen.





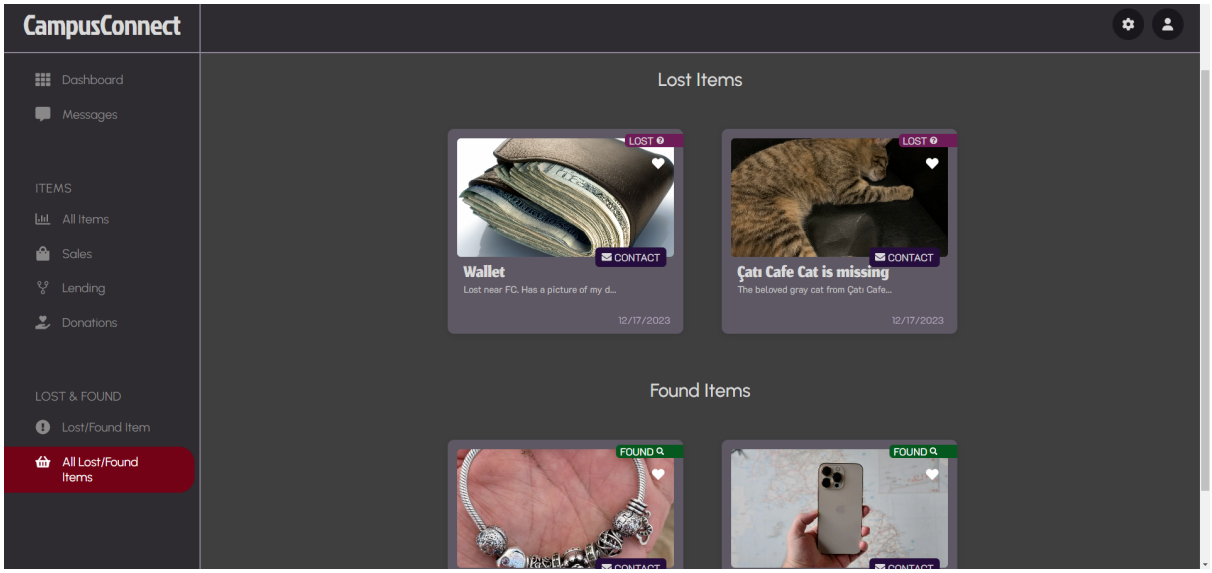
Donations Form:

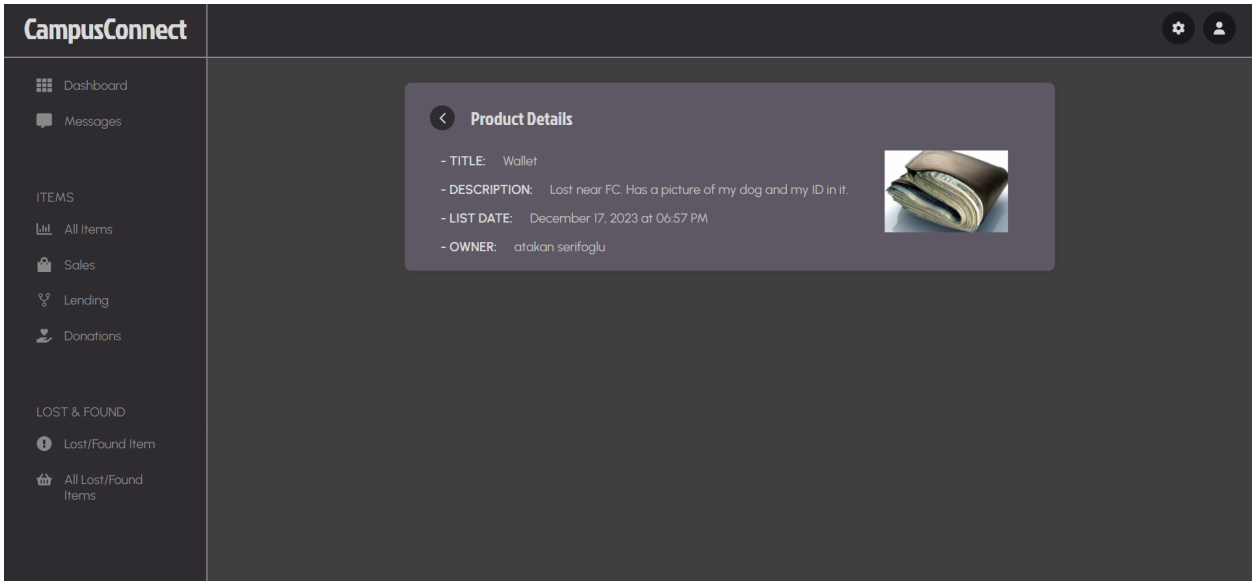
All the details for the item to be donated are entered by the user here. The name, description and type will be entered, along with 5 optional photos.



All Lost and Found Items Page:

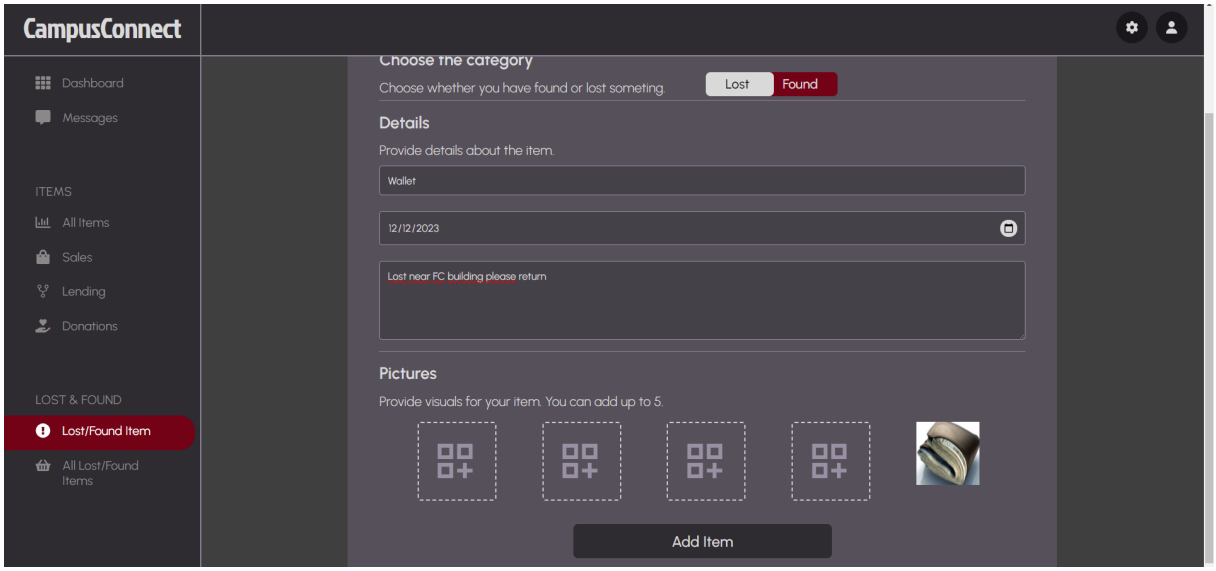
All lost/found items of the application are found here. Users can contact the lister of the item through the contact button on the cards. By clicking on the cards, extra information about the item can be seen.





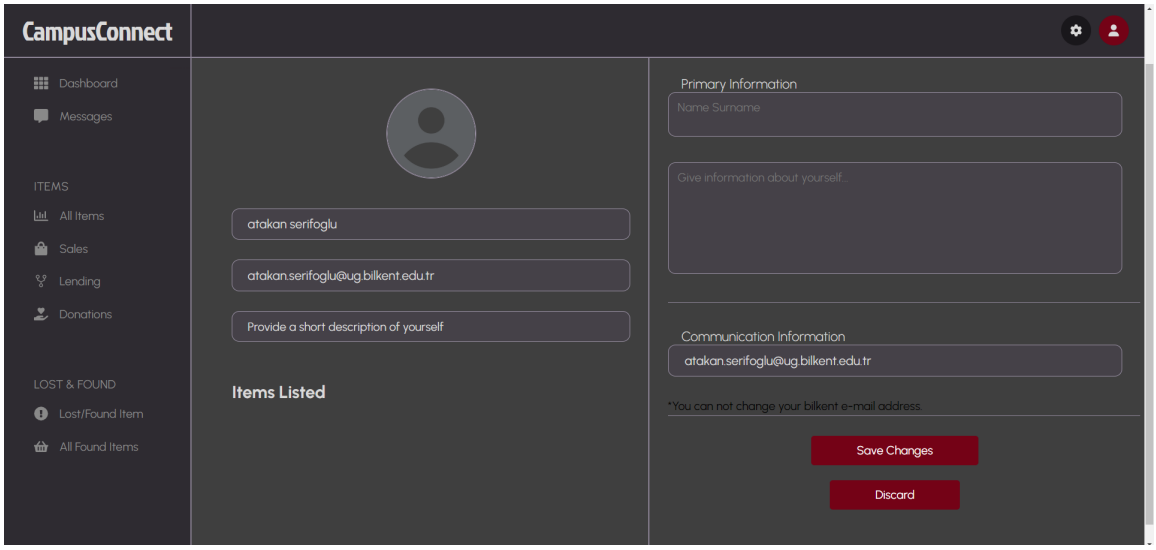
Lost/Found Form:

All the details for the lost/found item are entered by the user here. The name, description and date of when the item was lost/found will be entered, along with 5 optional photos.



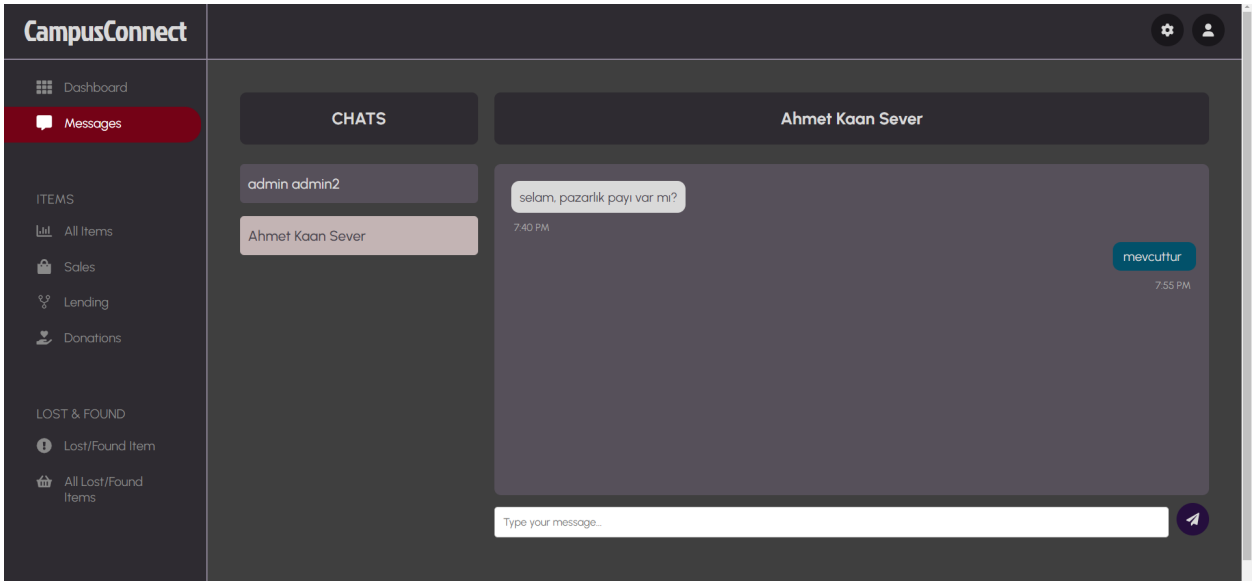
Profile Page:

On the profile page, users can see their personal information such as name, email, profile picture and their description. They can change all of these fields except for the email address for security purposes.



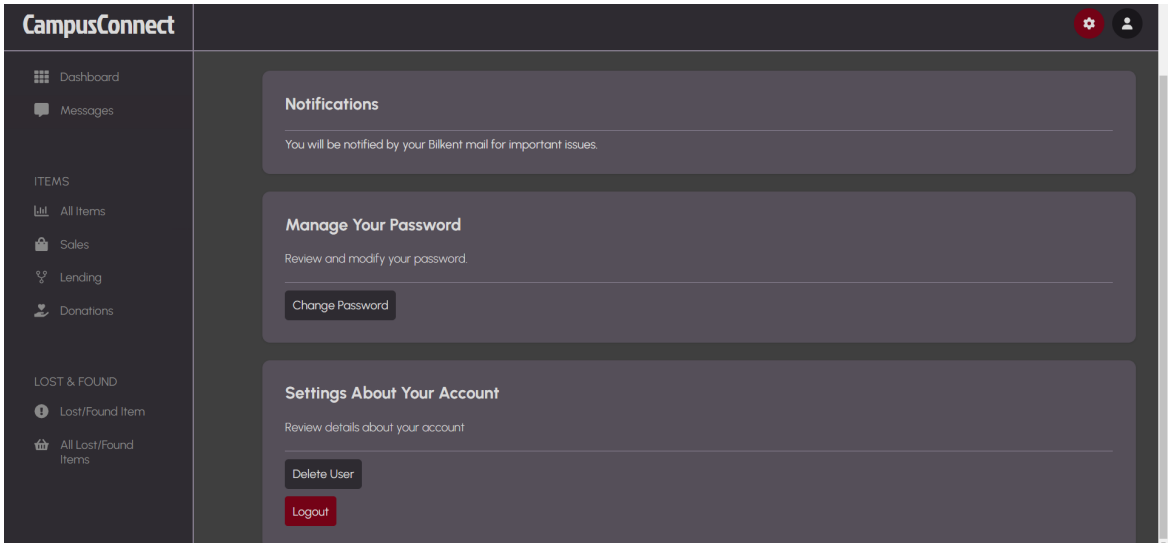
Messaging Page:

The messages page allows user to message any students they have contacted through an item listing previously. The messages are delivered in real-time and possess time signatures.



Settings Page

Users can change their passwords, delete their profile or logout of the application.



4. High Level Description

CampusConnect provides a way for Bilkent students to connect by allowing them to sell, lend, donate and list their lost/found items, as a more polished alternative to the @bilkentitiraf ecosystem. Users are required to sign up with their Bilkent email to ensure only students are able to access the system. Users can peruse the marketplace that contains all listings, make their own listings and message other students through the messaging feature. Users can access the application through the link http://bilcampusconnect.azurewebsites.net/

5. Developer Manual

5.1. How to obtain the source code:

Fork/Clone this repository (https://github.com/CS319-23-FA/S3T12-CampusConnect.git) into your local machine. Make sure you are on the "main" branch. Open the project folder using Visual Studio 2022 Enterprise edition (which should be free to obtain with an educational e-mail). Visual Studio 2022 Community edition will probably work as well if you cannot obtain the Enterprise edition for some reason, but please note that we have not tested it. Developers can run the backend by two options using VS' pre-built run functionality or use a terminal to run the dotnet, respectively.

5.2. How to build the software:

5.2.1. Backend instruction:

The database and Web API is hosted on Azure Cloud. If you want to host the entire app locally you have to change API Call URL's in frontend from https://bilcampusconnect.azurewebsites.net to https://localhost:7094. After doing the conversion, to open the project at VS navigate to open local folder, navigate to S3T12-CampusConnect folder and find my-new-app.csproj or my-new-app.sln and click it.In other words, find the solution to open it in the VS. To run the backend using VS it can be simply run by pressing the run button of the VS. In contrast, to run the backend of the project using a terminal, developers should navigate inside the project folder "cd S3T12-CampusConnect." When navigated developer can run the backend by simply writing:

dotnet run

,to compile the backend of the project. The localhost will be available after the dotnet command. If the localhost didn't open automatically after the "dotnet run" command, the swagger website (debug interface of the project) can be achieved by using the url of https://localhost:7094/swagger/index.html. As a reminder, the project will fully work after the both backend and frontend compiles.

5.2.2. Frontend Instructions:

Running the frontend part requires some packages. Initially install node.js and npm and make sure you can use the npm command.

Navigate to ClientApp folder at my-new-app and write the following commands

npm install @microsoft/signalr

npm install

After installment of the node_modules the react app can be run by writing “npm start”

After following the above installment processes, now the project is ready to launch at localhost. If the frontend of the app doesn’t launch automatically you can open it manually by using the following url: https://localhost:44442/. After following the commands carefully both backend and frontend will work seamlessly and the project will be available to use.

5.3. The layout of your directory structure:

There are several directories with functionalities varying from backend and frontend. Dependencies hold every dependency for example: Hangfire, SignalR or AzureBlob. For ClientApp directory: it holds every frontend logic class that makes use of backend functionalities to display elements with the help of React. The controllers folder holds, as the name suggests, all the controllers that are used to control, send, mutate and write all the data that exists within the system. All major functionalities of the app are consolidated into their isolated controllers, to ensure data and function separation. For example, the AuthController and MailController classes are in different folders; as they do not overlap in functionality, and so should be separate. Data directory holds the database content class which is the schema for creating migrations. In other words dbcontext at the Data directory controls how items to be mapped and stored at AzureDB. The DTOs folder holds all the data transfer objects used by the different item types in the backend. They ensure only the most important data is sent where it needs to and reduced data cluttering in the project in general. The migrations folder holds the migration files for all the data within the database. They are an integral part of the project, and so are separated into their own folder. The models folder contains old data models used for the authentication parts of the application. They ensure proper data transfer from the frontend to the backend and increase code understandability. The reports folder holds all the reports for the project. The Hubs directory holds SignalR hub that allows real time messaging functionality. Properties directory holds launch setting that creates swagger interface at launch. Finally in the main directory Program.cs file is the “main” function of the app which creates and initializes the backend classes and database, maps crucial urls and finally runs the code. The services folder holds all the services

used by the controllers within the application. They serve as individual functionality chunks of the application, such as the AuthServices or

5.4. Coding Conventions

All variables and functions must be named succinctly and descriptively. Private variables are names with a _ preceding them with camel-case, parameters are with camel-case, public and protected variables are Pascal case. To ensure data safety and ease of management, the project prioritized on reducing coupling as much as possible. DTOs and similar data transfer conventions were used to securely transfer, store and write data. We maintained a feature based branch structure, so that we could collaborate and help our teammates with their problems, and we recommend you do the same for future development. We also reviewed and resolved merges and conflicts as a team to ensure no errors or bad practices found their way into important branches.

5.5. Known Bugs

For 2-way messaging, when attempting to display the time of the last message, an erroneous empty message bubble appears. This is due to a yet unresolved issue, but it is important to point out that this bug does NOT lessen or break the messaging functionality to any meaningful extent, and that it persists only because we did not have the requisite time to resolve it.

5.6. Bug Report Template

For reporting bugs through GitHub issues, please use the following template:

Description: Describe the bug or issue with enough detail.

Reproduction: Explain how the issue can be recreated by developers.

Expected Result: Describe how the functionality should actually work.

Actual Result: Describe what actually happens on the app.

Screenshots/Images: Include screenshots or images that can help in diagnosing the problem..

Device Specifications: Provide data about your device and operating system where the issue occured.

Browser: Specify the browser where the bug occurred .

Severity/Priority: Assign a severity and priority level to the bug based on its impact and urgency.

Date and Time: Specify the date and time when the bug was first encountered.

6. Work Allocations

Ahmet Kaan Sever - 22102278

Analysis Report:

- Worked on the Sequence diagrams with Perit
- Worked on the Use Case Diagram with Perit and Konuralp
- Wrote the package details for packages such as Sell-Buy and Borrow-Lend in the textual specifications

Design Report:

- Designed the subsystem decomposition layer diagram with Perit and Konuralp
- Wrote the layer explanations for all 6 layers with Perit
- Wrote the Persistent Data Management and Access Control and Security paragraphs
- Especially did work on the Firewall and Database layers

Implementation:

- Worked on the backend of the project
- Implemented the Item system logic
- Used Services, Controllers and Repositories to accurately and efficiently store and transfer item and user action data
- Created models and DTOs for taking and writing user data
- Used MSSQL-Azure and Blob storage to store item data and images
- Co-designed messaging system with Konuralp

Final Report:

- Wrote work allocation for myself
- Corrected eros in the report

Atakan Şerifoğlu - 22102313

Analysis Report:

- Wrote non-functional requirements.
- Draw the class diagram and wrote the explanations for it.
- Helped wrote textual descriptions for use case diagram.

Design Report:

- Helped write design goals.
- Helped review the access control matrix.
- Helped to design the presentation layer diagram.
- Wrote hardware-software mapping.
- Wrote the details for decorator design and drew its diagram.
- Listed some of the pages and their methods for the front end.

Implementation:

- Designed and implemented routes and created private routes.
- Implemented the navbar and sidebar
- Co-implemented the card component.
- Implemented the login-signup logics for frontend.
- Implemented the backend-frontend connection logic and wrote callbacks.
- Implemented the messaging logic and co-implemented the chat design.
- Implemented the user-seller interaction logic in the frontend.
- Co-implemented the designs for all of the pages.
- Co-implemented the jsx codes for all of the pages.
- Wrote the GeneralItemController for the backend.

Final Report:

- Wrote work allocation for myself
- Helped write the developer diagram

Dila Tosun - 2212100

Analysis Report:

- Helped edit the use case textual specification and provided some of them.
- Designed all of the mockups.

Design Report:

- Wrote design goals.
- Provided some rows of the access control matrix.
- Wrote boundary conditions other than the backend specific conditions.
- Wrote the factory design pattern and provided the diagram.
- Listed some of the pages and their methods for the front end.

Implementation:

- Designed the UI for the pages.
- Co-implemented the card component.
- Co-implemented the designs for all of the pages.

- Co-implemented the jsx codes for all of the pages.
- Implemented filter for All Items, All Found Items, Dashboard, Sales, Donations, and Lending pages.
- Implemented time logic for messaging.
- Implemented product detail logic for some card components.

Final Report:

- Wrote work allocation.
- Edited the design screenshots.
- Made the teaser video.

Konuralp Kılınç - 22101791

Analysis Report:

- Worked on Activity diagrams.
- Worked on the Use Case Diagram with Ahmet and Perit

Design Report:

- Designed the subsystem decomposition layer diagram with Ahmet and Perit
- Prepared layer diagrams of the report and backend class diagrams with Ahmet and Perit

Implementation:

- Worked on the backend of the project
- Designed messaging system with Ahmet
- Designed notification system with Ahmet and Perit
- Published project using Azure Publish with Ahmet

Final Report:

- Wrote work allocation for himself
- Wrote build instructions of the application

Perit Dinçer - 22103102

Analysis Report:

- Worked on the Sequence diagrams with Ahmet
- Worked on the Use Case Diagram with Ahmet and Konuralp
- Wrote the authentication package in the textual specifications

Design Report:

- Designed the subsystem decomposition layer diagram with Ahmet and Konuralp
- Wrote the layer explanations for all 6 layers with Ahmet
- Wrote the Persistent Data Management and Access Control and Security paragraphs

Implementation:

- Worked on the backend of the project
- Designed the authentication and user information systems
- Used .NET Identity to store user data securely on MSSQL-Azure database

- Utilized JWT access and refresh tokens to keep users logged in securely to the application
- Created models and DTOs for taking and writing user data
- Created controller and service architecture to authentication
- Built mail notification functionality

Final Report:

- Wrote work allocation for himself
- Took the screenshots and wrote the explanations used for the user manual

Selimhan Tokat - 22003145

Analysis Report:

- Wrote Use Case textual specifications for User Profile, Money Transaction, New Item, Marketplace, and Donate & Receive packages.
- Wrote the explanations of the Activity Diagrams.
- Reviewed the Use Case textual specifications for the second iteration for the renewed Use Case diagram.

Design Report:

- Designed the User Interface Management Layer.
- Designed the Access Control Matrix
- Wrote Object Design Trade-offs for the project.

Implementation:

- Worked on the frontend of the project.
- Worked on some of the UI.
- Checked and fixed the edge cases.
- Fixed issues regarding front end codes.

Final Report:

- Wrote work allocation for himself.