CMPE 451 - Milestone 3 Report
Group 2

Abdullah Yıldız
A. Furkan Varol
Aleyna Kara
Aslı Aykan
Bekir Yıldırım
Burak Çetin
Furkan Nane
Hande Şirikçi
Mehdi Saffar (Communicator)
Mehmet Umut Öksüz
Murat Can Bayraktar
Ömer Faruk Deniz
Özdeniz Dolu

January 31, 2021

Contents

1 Final project assessment

1.1 Introduction

Getflix is an e-commerce platform that connects vendors and customers in one single platform with a great user experience. Our platform lets vendors signup to our system and add their products. They are able to later edit or delete their products through their vendor page. When a product of theirs is bought, they can display the order in the orders section on profile page. Then, they can change the current status of any order.

Getflix is also a nice and friendly place for buyers. Customers can signup and login into the system and immediately start shopping. The homepage displays our best products in certain categories. As customers make orders, Getflix recommends more personalized products. Customers can add products to their shopping cart and decide to continue their shopping later. Customers may decide to add products to their wishlist and get notified of any price changes. This system also notifies buyers as their orders' status changes.

Getflix cares about customer satisfaction. Thus, offers much functionality to make customers sure about their buying choice. Getflix uses a rating and reviewing system for products, however, only the people who have bought the item can make reviews and rates. This way, we make sure products are rated fairly. Also, customers can contact the vendors about their questions and any problems. Getflix has a nice messaging functionality and supports sending media files.

1.2 Milestone 1 and Milestone 2 Assessment

For the first milestone, we have focused on building our foundation firm and clean. Our main prioritization was to keep our backend endpoints documentation up-to-date, to fix our meeting notes format, pull request templates and pull request review-approve-merge cycle, and to complete the CI-CD process to make things easier for further milestones. Team members followed our best practices and the management became much easier.

In terms of functionalities, we have completed the sign-up page, login page, auto-login using tokens, search page, product page, profile page, shopping cart page. We created a trending products section in our landing page, created a product card to display products in packed form, designed the header to show current user's information.

For the second milestone, we were much confident about our performance and productivity since milestone 1 was showing that our team was working hard and communicating efficiently. Our work started working on the components that received feedback. We focused on fixing our bugs,

We have delivered our major components timely in accordance with our plan. We have completed purchasing a product including adding and editing credit cards and addresses, search page functionality in the backend, product reviewing system and displaying in the product page, managing customer orders for vendors, profile page with addresses, credit cards, messages sections, product page with detailed information, email verification system and displaying a warning to unverified accounts, wishlist functionality with mock data.

Although we have created many pages and components, all teams focused greatly on User Experience when designing the subcomponents. We have changed our homepage to display products in categorical format because of feedbacks in milestone 1. We have improved our init process by showing a user-friendly loading page, started to increase using status update messages at the right-top corner of the page, enhanced token management system

by injecting tokens in HTTP headers, created a mock proxy server in frontend for faster development experience.

1.3 Final Milestone

For the final milestone, our team knew that we were going to be busy at this time of the semester, so we have greatly advantaged our decisions in previous milestones like implementing continuous integration pipeline in milestone 1, or focusing on user experience and not overlooking small details and hoping our customer would not notice.

Teams were confident in their acquired skills obtained through milestones 1 and 2, so we did not experience technical slowness. After milestone 2, we all have gathered in the lab session and discussed the deliverables for the final milestone. As we concretized the tasks, we made sure that our customers and we were on the same page. Then, teams started arranging their own meetings.

Previous milestones were focusing on components from the customer's point of view. However, for the final milestone, we focus more on vendor-related pages and components.

The functionalities we have implemented are as follows:

- We have started using personalized product recommendation system, this is the first time we have included data science into our system. We have displayed these recommended products on our homepage.
- We have created a notifications section on the profile page to let customers know about the change in their order status and any price change of products that are added to the wishlist.
- We have created a vendor page to display fundamental vendor information like title, description, image, and rating. Also, the page displays the main products of the vendor in a grid-based view. The page is designed for both customers and vendors so it normally shows vendor's info but it enables the vendor owner of the page to edit it. The owner can add new products, edit existing products, edit its basic information in place, etc.
- To edit and add products, we have prioritized the user experience. A pop-up modal is shown and fills the information automatically from the clicked product.
- Messaging system that enables customers and vendors to connect and share information. Messaging functionality modal made accessible via Orders section and Vendor's page. Previous messages can be accessed on the Profile page.
- Created an Orders page for Vendors that shows the current status of orders and it comes with functionality to change the status of orders.

Overall, we spent such a productive and amazing semester since we have practiced teamwork, realized the importance of communication, gained knowledge about software best practices, and obtained hands-on experience in many latest technologies.

2 List and Status of the Deliverables

Deliverable	Delivery Date	Status
Web interface	31.01.2021	Complete ✓
Android interface	31.01.2021	Complete ✓
Database and endpoints	31.01.2021	Complete ✓
Requirements	31.01.2021	Complete ✓
API documentation	31.01.2021	Complete ✓
Project plan	31.01.2021	Complete ✓
User scenarios presented during the demonstration	31.01.2021	Complete ✓

- Web Interface: The frontend application of our project was flawless in the presentation. We have showcased multiple components without any bugs and problems. Planning and implementation are designed by taking user experience into account as discussed in the lectures and lab hours. The functionalities that are presented were in line with our customer's needs.
- Android Interface: Android application of our project was ready and fully functional in the presentation. Components in the Android application were in sync with the Web Interface in terms of design and functionality.
- Database and endpoints: Database and endpoints worked as intended during the presentation. We have created data, such as some orders and reviews, to be used in the presentation specifically for our scenario.
- Requirements: Requirements were already cleared and updated in Milestone 1 so there was no need for any other change. A link to requirements can be found in our Wiki page
- API Documentation: API documentation is up to date with all the endpoints. Thus, Android and Frontend teams had been working without the constant need for the Backend team.
- Project Plan: Our project plan is ready and can be found on our Wiki page. The project plan has been a directive in planning our work, distributing tasks, and completing the aforementioned deliverables.
- User Scenarios: User scenarios that we used in the presentation can be found in the report. The scenarios for Frontend and Android teams are arranged by the team leaders and got approved by all team members. Scenarios are arranged to be in line with each other and to complement the flow of the presentation and present the most important features we have implemented.

3 Coding Work Done by Each Team Member

Aleyna Kara	I have designed the layout, namely fragmentvendor.xml after creating VendorPageFragment.kt with its view model VendorPageViewModel.kt. Since my initial design has included just a recycler view that shows which products are currently retailed by the vendor, I have also created cardvendorproduct.xml followed by VendorPageProductAdapter.kt. With a view of improving this auspicious design, I have watched a video explaining the implementation of ViewPager2 and also examined the documentation of ViewPager2 written by Google. I have created three fragments - NByTwoGridFragment.kt, NByThreeGridFragment.kt and VendorReviewsFragment- and their view models NByTwoViewModel.kt, NByThreeViewModel.kt and VendorReviewsModel.kt so that The use of ViewPager2, ways and means to transfer my knowledge into practice, has been provided by creating VendorPageFragmentAdapter.kt in order for the page of the vendor to have horizontal paging which is a navigation pattern frequently used in modern apps. Furthermore, it is clear that fragmentnbytwogrid.xml, fragmentnbythreegrid.xml and fragmentvendorreviews.xml have been created at the same time. Now, let me explain these three fragments in detail : 1. NByTwoGridFragmen.kt : My motive to create this fragment is to show all data which a user probably wants to look for first when he or she is viewing a product. Thus, it has been sufficient to update the binding of VendorPageProductAdapter. 2. NByThreeFragment.kt : This fragment is a shot in the arm for UX since it enables a user to glance at the products of the vendor thanks to the design of the view holder, namely cardvendorproductsmal.xml. I have also created VendorProductSmallAdapter.kt. 3. VendorReviewsFragment.kt : With the aim of the ability to view all reviews regarding the vendor, VendorCommentAdapter.kt and cardvendorreview.xml has been created. I have taken the responsibility of the implementation of VendorOrdersFragment.kt in for which the vendor user of our system is able to change the status of the order while viewing all orders within the same page. Furthermore, a vendor is able to click any order to get more information about a specific order. It has been a task which has included another research topic for me since I had not enabled Google Maps SDK for any Android project. Video resources and the guides provided by Google have been sufficient to grasp how to use it. In addition to Google Maps API, I have made researches about shared element activity transitions to make the navigation from the list of orders to the fragment of a single order more continuous and smooth. From the perspective of the implementation, my initial attempt was to write non-existing PUT and GET request to /vendor/order and create necessary data models to handle the response and the body of the request. Then, I have implemented VendorOrderAdapter.kt for the list of orders and cardvendororder.xml as the layout of the adapter's view holder. For the maintenance of MVVM pattern, I have controlled actions taken after click events and requests to the /vendor/order endpoint in VendorOrdersViewModel.kt after creating it. Speaking of the single page of the order, it is where I have used Google Maps in order for the vendor to estimate the address of the destination to determine the delivery time and the status of the order, i.e. whether or not he or she is able to send a cargo to the specified address by viewing the order address visually. Clearly, I have also implemented VendorOrderFragment.kt, VendorOrderViewModel.kt and fragmentvendororder.xml. Since I have served the functionality to change the order status up via spinners, I have customized the spinner by creating customspinnerdropdownitem.xml. Mail verification and change of password have been my other tasks which has required the search as regards Firebase. After checking the blogs, videos and documentation about Firebase out, I have created MailVerificationFragment.kt, MailVerificationViewModel.kt and updated the functions in both LoginViewModel.kt and RegisterViewModel.kt. It should be mentioned that I have implemented POST request to /changepassword which has never been used. My last task has been to implement both POST request to /review for the customer to make a comment to a product which he or she already bought and data model of the body of this request. I have also created a comment dialog fragment for the customer to be able to propose a review directly when he or she is viewing its orders and added a section in the product page itself. In addition to adding new features to our app, I have fixed six bugs related to the empty shopping cart, empty list of recommended products, empty address fragment, empty list of credit cards, empty shopping cart and change the status of the notification. I have, at the end, refactored our code via the removal of all println statements and I have reviewed a couple of PRs of my teammates during the process up to Milestone 3.

A. Furkan Varol	At the start of Milestone 3, I created the Recommendations component. This is a component that shows a list of recommended products for the specific user. Here, I used the same structure that I implemented previously for Best Sellers components on the Home Page. Then, I refactored and glanced over the whole project. Removed the necessary imports from every .js file. Altered the design of the Best Sellers components on the Home Page, so that they did not overflow vertically. Changed some error messages and notification text with more appropriate ones. At last, I tried to get the updating profile information functionality up and running before the milestone and implemented the frontend part but we could not stay in sync, thus I was not able to merge this into master.
Furkan Nane	Implemented the personalized product recommendation endpoint and wrote the unit test to test it. Attended the backend meetings as well as team meetings. Also reviewed a lot of pr's.
Hande Şirikci	Changed one of the five main fragments in application from favorites to lists by constructing all the structure for creating new lists and to display them in order and also clicking on lists to see products in lists. Implemented lists in a recycler view, created view model for list item then created navigation feature for each list item in the list. Designed and implemented the card views in list fragments displayed product and enabled user to add it directly into her basket. Implemented notification page by creating recycler view and view model for the notification item. Displayed the notifications in a list in notification page designed noticeable view for the items. Updated register form to took the address information of the vendor. Created a new part in register form of vendor under the relevant title to made register form more organized. Redesigned the orders page which i have completed for second milestone by adding a review option according to status of the orders and also implemented navigation from order items to a fragment which displays all the products in that order in a list. Implemented recycler view, view model for order items and designed horizontal card view to display products with their amount information, price for one of the product and also price for all the products in that specific order. I have always followed MVVM pattern and used data binding for all the tasks i have completed since the beginning of the term. I have reviewed many of the pull requests and attended all meetings.
Mehdi Saffar	refactor the CI/CD pipeline workflow files to work with only one single instance, implement the frontend interface of the messaging system, implement the vendor user interface focusing on the profile page and the vendor orders, report a few bugs in backend (image url confusion, vendorid vs userid confusion), co-author vendor orders page with Özdeniz Dolu, co-author add/edit product modal with Özdeniz Dolu, continue implementing missing functionalities of Product List with Özdeniz Dolu, hide customer-related actions from Vendor user interface (such as add to shopping cart, add to product list, checkout...), co-author the vendor homepage with Özdeniz Dolu and Abdullah Yıldız, due to our EC2 instance being hacked the day before the demo, refactor backend and github actions to hide django SECRETKEY from github and pass it as an environment variable to the CI/CD pipeline, collect a list of bugs in the frontend (more than 15) and fix them before the demo, build Docker images of backend and frontend for the Submission Delivery Package, write the System Manual with Ömer Faruk Deniz for deploying locally and to AWS
Mehmet Umut Öksüz	Implemented price change notifications and order status change notifications at backend, implemented endpoints to make a specific notification or all notifications seen. Fixed minor bugs (image adding problem at vendor edit product, changed something in delete product). After frontend team requested, added username to user serializer. Attended meetings and distributed tasks at backend team. Also join some frontend meetings to listen their requirements from backend and fix problems. Helped my teammates wherever they had a problem.
Murat Can Bayraktar	I've learned Kotlin programming throughout semester and individually implemented a few minor apps but they weren't used in the GetFlix project. I've designed the product page. The design later replaced with a better version. I couldn't be supportive to the team in the code field. Instead, I've participated more in documentation issues.

Özdeniz Dolu	Co-authored orders page for vendors with Mehdi Saffar, co-authored edit/add product modal with Mehdi Saffar, co-authored some of the functionalities related to Product Lists (not all), co-authored vendor shop page with both Mehdi Saffar and Abdullah Yıldız, towards the last days of the milestone Mehdi Saffar and I spent countless hours on small stuff, polishing, bug fixing and refactoring
Abdullah Yıldız	Created a notification system where customers are getting a message about a price change in their wishlist and any order status change. Designed a page to display notifications in vertical list format. Designed notification's layout to show image, link, title, date and description message. I kept in coordination with Android team to keep in harmony. I created a common message description template for both price change and order status changes and shared with Android team. I created buttons to mark notifications and mark specific notification seen. Created Vendor's page to display fundamental information of vendors. Designed the page editable according to current signed in user. Changed product cards editable to work with vendor's page. Designed the Vendor Splash page, arranged the layout, selected a background color. I created Vendor's Main products page in grid based view by getting help from our previous components. I created Vendor Reviews page from our previous User Reviews component in case we can display different information. I have integrated Adding product and Editing product modals with my Vendor page.Created a notification system where customers are getting a message about a price change in their wishlist and any order status change. Designed a page to display notifications in vertical list format. Designed notification's layout to show image, link, title, date and description message. I kept in coordination with Android team to keep in harmony. I created a common message description template for both price change and order status changes and shared with Android team. I created buttons to mark notifications and mark specific notification seen. Created Vendor's page to display fundamental information of vendors. Designed the page editable according to current signed in user. Changed product cards editable to work with vendor's page. Designed the Vendor Splash page, arranged the layout, selected a background color. I created Vendor's Main products page in grid based view by getting help from our previous components. I created Vendor Reviews page from our previous User Reviews component in case we can display different information. I have integrated Adding product and Editing product modals with my Vendor page.
Ömer Faruk Deniz	Implemented the password change endpoint, send message and get conversations endpoints both of which compose the direct-messaging system of the platform, implemented the semantic search functionality, fixed the userid and vendorid in the system. Fixed the system before the Milestone 3 due to an attack to the server and wrote the System Manual with Mehdi Saffar. Attended all of the meetings of backend, arranged meetings for the specific needs of other teams. Reviewed and refactored nearly all of the pull requests opened in backend for the Milestone 3.
Bekir Yıldırım	After the Milestone 2, I have worked on ProductList table and implemented relevant endpoints. By list endpoints, user can create, get and delete product list and also put product into list or delete product from list. To make ProductList table more functional, I have arranged table to ease table manipulation. With the backend team, We have gathered many times as a team and generally our meeting's main subject is refactor and bug fix. Also I have reviewed many PR and give suggestion or change request to PR's owner. We exchanged ideas with Ömer Faruk Deniz on the tools we will use to make the currently written search endpoint more functional and the roadmap we will follow.

Aslı Aykan	After Milestone 2, I firstly dealt with the filter, sort, and search functionalities which were missing in Milestone 2. I added an edit text to the toolbar on the homepage, and added the search functionality by refactoring the related endpoint function and handling necessary navigations. I added a rating bar to FilterOptionsFragment, and made the options for the filter on this page work correctly by also adding necessary fragment arguments for it. I also edited the vertical product layout, by adding a rating bar and removing unused amount section for the user to see the ratings of the products. I connected all the sort options of the related endpoint to the layout. I removed the toolbar on ProductFragment and added a small back button for the user to see the full image of the product. For the message functionality, I firstly made a research about chat libraries in Android. After also discussing with my subteam, I decided to use the ChatKit library. I needed to examine the sample Java code of this library on its GitHub page, and find out what functionsdata models are necessary and how to adapt them to our app. After that, I created CustMessagesFragment(by also adding its navigation to the ProfileFragment), CustChatFragment and VendorChatFragment from scratch, and redesigned the VendorMessagesFragment. To use the library functions correctly, I added the necessary data classes which are Message, MessageModel, MessageListModel, DialogModel and AuthorModel. For the classes that I created, adapted and converted to Kotlin language for this library, I gave their related GitHub links as references in comment sections of my code (I also informed stfalcon-studio that we are using its library for our school project via email). I added the necessary endpoints functions for GET /messages and POST /messages, by also adding their necessary request and response classes. Termsconditions texts for register and completing order were also missing in Milestone 2, so I took these texts from frontend and added them to our app. I also added the necessary checkbox for it on the complete order page and made them clickable so that pop-ups open with these texts on it. For the list selection on product fragment, I created two dialogs: one with the radio buttons(with MaterialDialog builder) and the other one with an edit text(custom dialog). For a more realistic app design, instead of showing empty pages and only a floating action button when the user has no credit cards, addresses or product lists, I created three empty page designs for ListsFragment, AddressFragment and BankAccountFragment with an image view, text view and add button by also adding related navigations to these buttons. I disabled navigating to the complete order page if the shopping cart is empty, with a dialog that warns the user. I also handled the bugs that occur when the guest option is enabled. I made navigating to the complete order page, adding products to the shopping cart and navigating to the bank accounts page disabled for the user with related dialogs that warn the user. I added the necessary endpoint functions for GET /lists, POST /lists, DELETE /lists/id, DELETE /lists/id/product/productid, POST /lists/id/product/productid, GET /recommendation, GET /notification, POST /notification/seen and POST /notification/seen/notificationid by also adding necessary request and response classes. I connected all these endpoints with the related layouts. For delete endpoints, I created necessary SwipeToDelete classes. After the notification endpoint connection, I made the necessary design updates for NotificationFragment (with product images, seenunseen buttons and same text formats with frontend). I also made design improvements on OrderInfoFragment and OrderProductsFragment, after realizing our misunderstanding for the json format of order. I connected the add to cart button on list products page to the related endpoint function. I refactored all the product images according to the update in the backend. After the presentation, I also added POST /vendor/product and DELETE /vendor/product endpoints, refactored the vendor product navigation, connected these endpoints to related layouts, and created AddProductFragment from scratch. I also handled some small bugs during the last refactoring. I merged all the commits for Android while also resolving all the conflicts. I also reviewed all of the pull requests for Android.
Burak Çetin	Impemented vendor profile and vendor details endpoints and updated the database models accordingly. Fixed a critical bug on the search endpoint.

4 Project Plan

Name	Start Date	Finish Date	Assignee Group
Creating initial architecture of Django App	11/03/20 8:00	11/17/20 17:00	Backend Team
Setting up JWT authentication	11/03/20 8:00	11/17/20 17:00	Backend Team
Database design	11/03/20 8:00	11/17/20 17:00	Backend Team
Implementation of database model classes in code side.	11/03/20 8:00	11/17/20 17:00	Backend Team
Implementing register, login and token validation for customers	11/17/20 8:00	11/24/20 17:00	Backend Team
Implementing shopping cart get and add items functionalities	11/17/20 8:00	11/24/20 17:00	Backend Team
Implementing Google login	11/17/20 8:00	11/24/20 17:00	Backend Team
Implementing products/homepage and product details	11/17/20 8:00	11/24/20 17:00	Backend Team
Unit test development	11/17/20 8:00	11/24/20 17:00	Backend Team
Dockerizing backend and database	11/17/20 8:00	11/24/20 17:00	Backend Team
Initialize the app	11/03/20 8:00	11/10/20 17:00	Android Team
Implement the bottom navigation bar and necessary navigation	11/03/20 8:00	11/10/20 17:00	Android Team
Implement the splash screen	11/03/20 8:00	11/10/20 17:00	Android Team
Implement the localization of the app	11/03/20 8:00	11/10/20 17:00	Android Team
Implement the categories page	11/10/20 8:00	11/17/20 17:00	Android Team
Implement the favorites page	11/10/20 8:00	11/17/20 17:00	Android Team
Implement the cart page	11/10/20 8:00	11/17/20 17:00	Android Team
Implement the profile page	11/10/20 8:00	11/17/20 17:00	Android Team
Implement the category page	11/10/20 8:00	11/17/20 17:00	Android Team
Implement the sign in form	11/17/20 8:00	11/24/20 17:00	Android Team
Implement the sign up form	11/17/20 8:00	11/24/20 17:00	Android Team
Implement the login endpoint connection	11/17/20 8:00	11/24/20 17:00	Android Team
Initialize frontend folder, setup React	11/03/20 8:00	11/10/20 17:00	Frontend Team
Create initial layout, global state management, routing	11/03/20 8:00	11/10/20 17:00	Frontend Team
Add product card component	11/03/20 8:00	11/10/20 17:00	Frontend Team
Add signup form component	11/03/20 8:00	11/10/20 17:00	Frontend Team
Add search input component	11/03/20 8:00	11/10/20 17:00	Frontend Team
Create login page, login form	11/03/20 8:00	11/10/20 17:00	Frontend Team
Add "Trending Products' grid in home page	11/10/20 8:00	11/17/20 17:00	Frontend Team
Add search product page	11/10/20 8:00	11/17/20 17:00	Frontend Team

Name	Start Date	Finish Date	Assignee Group
Add logged in user info on top bar	11/10/20 8:00	11/17/20 17:00	Frontend Team
Add shopping cart page	11/10/20 8:00	11/17/20 17:00	Frontend Team
Add profile page	11/17/20 8:00	11/24/20 17:00	Frontend Team
Add Axios instance, token management, login, logout	11/17/20 8:00	11/24/20 17:00	Frontend Team
Design improvements	11/17/20 8:00	11/24/20 17:00	Frontend Team
Integrate necessary pages with Backend	11/17/20 8:00	11/24/20 17:00	Frontend Team
Add Product page	11/17/20 8:00	11/24/20 17:00	Frontend Team
Finalize dockerization and deployment pipeline	11/17/20 8:00	11/24/20 17:00	Frontend Team
Milestone 1 - 11/24/20			
Implementing Search-Sort Mechanism	11/30/20 8:00	12/03/20 17:00	Backend Team
Detailed Product and User Profiles	11/30/20 8:00	12/03/20 17:00	Backend Team
Implementing Comment Mechanism	12/03/20 8:00	12/10/20 17:00	Backend Team
Implementing Shopping Cart	12/10/20 8:00	12/16/20 17:00	Backend Team
Implementing Order Mechanism	12/18/20 8:00	12/24/20 17:00	Backend Team
Implementing Search-Sort Mechanism	11/30/20 8:00	12/03/20 17:00	Android Team
Detailed Product and User Profiles	11/30/20 8:00	12/03/20 17:00	Android Team
Implementing Comment Mechanism	12/03/20 8:00	12/10/20 17:00	Android Team
Implementing Shopping Cart	12/10/20 8:00	12/16/20 17:00	Android Team
Implementing Order Mechanism	12/18/20 8:00	12/24/20 17:00	Android Team
Implementing Search-Sort Mechanism	11/30/20 8:00	12/03/20 17:00	Frontend Team
Detailed Product and User Profiles	11/30/20 8:00	12/03/20 17:00	Frontend Team
Implementing Shopping Cart	12/10/20 8:00	12/16/20 17:00	Frontend Team
Implementing Comment Mechanism	12/03/20 8:00	12/10/20 17:00	Frontend Team
Implementing Order Mechanism	12/18/20 8:00	12/24/20 17:00	Frontend Team
Milestone 2 - 12/29/20			

Name	Start Date	Finish Date	Assignee Group
Implementing Recommendation Mechanism	01/06/21 8:00	01/15/21 17:00	Backend Team
Implementing Notification-Alarm	01/06/21 8:00	01/15/21 17:00	Backend Team
Implementing Lists	01/06/21 8:00	01/15/21 17:00	Backend Team
Implementing Direct Message Mechanism	01/06/21 8:00	01/15/21 17:00	Backend Team
Implementing Recommendation Mechanism	01/06/21 8:00	01/15/21 17:00	Android Team
Implementing Vendor Functionalities	01/06/21 8:00	01/15/21 17:00	Android Team
Implementing Notification-Alarm	01/06/21 8:00	01/15/21 17:00	Android Team
Implementing Lists	01/06/21 8:00	01/15/21 17:00	Android Team
Implementing Direct Message Mechanism	01/06/21 8:00	01/15/21 17:00	Android Team
Implementing Recommendation Mechanism	01/06/21 8:00	01/15/21 17:00	Frontend Team
Implementing Vendor Functionalities	01/06/21 8:00	01/15/21 17:00	Frontend Team
Implementing Notification-Alarm	01/06/21 8:00	01/15/21 17:00	Frontend Team
Implementing Lists	01/06/21 8:00	01/15/21 17:00	Frontend Team
Implementing Direct Message Mechanism	01/06/21 8:00	01/15/21 17:00	Frontend Team
Test Deployment	01/18/21 8:00	01/25/21 17:00	Whole Team
Milestone 3 - 01/26/21			

5 Activity stream implementation and W3C standard compliance

We didn't strictly apply ot the ActivityStreams protocol in GetFlix. GetFlix is also have no strict compliance with W3C Standards. However the platform behaves in similar patterns with these standards. Notification, messaging, and the other features are instinctually implemented coherent to these protocols and standards.

6 Unit Test

6.1 Mehmet Umut Öksüz

1- Commit SHA: b107198f9c05a195beee72d662253d8f82341a19

- test for make all notifications seen
- test for make single notification seen
- test for single notification seen with invalid token

2- Commit SHA: ed92c3975d4c9deed2c113fc6b417d924689b468

- test for price change notifications
- test for price change notifications (same price no notification tested)

3- Commit SHA: d1b16496618eab622e748934f0662e509fb9af21

- test for order status change notifications

6.2 Bekir Yıldırım

1- Tests of the Create List Endpoint

- test for create list : 676a57424b46a9abd0c3c8a6391fc1065f4eb7bc
- test for duplicate list name error : 3140b9c15bb13b680d2173d658640c4284fd0211

2- Tests of the Get List Endpoint

- test for getting list : 2fc1d2931f8c990671d0870b3931bb97ea1f5326

3- Tests of the Delete List Endpoint

- test for deleting list : c96181d5f52ce98913035aa57b15e140262c34dc

4- Tests of the Add Product to List Endpoint

- test for adding product to list : 596cf79cda83bab682c5c583151d7e702f404100
- test for adding duplicate product to list : c8463c5c97477c260966a207af86715aae97ecfe

5- Tests of the Delete Product from List Endpoint

- test for deleting product from list : b44653225c04f0f04047b82e0e5a8fc84c648c33

6.3 Ömer Faruk Deniz

1-Tests of the Password Change Endpoint

- test for password change : 8b348a4a551ebb967c3cfdd3cc0e0e5d60aefd96
- test for password change with wrong current password : b4930d301a18528df92b30af38179c00bbf5cf73
- test for password change without logging in : 19198e743ba7513396defcf45ec0f66223e8cd94

2-Tests of the Send Message Endpoint

- test for sending message : 1da3c3286e99a08087eff525a2a9aac536bc0070
- test for sending message without login : 4c17a891b4fe51a9f481bf6d3c970365156746d1
- test for sending message with invalid data : d1153266ce1266e8400168a083312977ed059120

3-Tests of the Get Conversations Endpoint

- test for getting all conversations : c62b6a718d13cfcd4d4c2ca9193c14fd92a642f1
- test for getting all conversations without logging in : 1a6c0e0d98942294d9792ca422dab4a7df1ce5ee

4-Tests of the Semantic Search Endpoint

- test for reading from Database API for similar queries : d184c47c7cb8bd0b6d4c4c2f5e2cf9c225b6be61

6.4 Furkan Nane

1-Commit SHA: bb86f3c077630e6e66c6d6d5305df5a2ffbe0579

- This test is for checking whether the number of recommended products are true

7 User and System Manuals, Requirements and Design Documents

The links for the user manual, the system , requirements and design Documents can be found here:

- User manual: https://github.com/bounswe/bounswe2020group2/wiki/User-Manual
- System manual: https://github.com/bounswe/bounswe2020group2/wiki/System-manual
- Requirements: https://github.com/bounswe/bounswe2020group2/wiki/Project-Requirements
- Design Documents:
  - Use case diagram : https://github.com/bounswe/bounswe2020group2/wiki/Use-Case-Diagram
  - Class Diagram : https://github.com/bounswe/bounswe2020group2/wiki/Class-Diagram
  - Sequence Diagram : https://github.com/bounswe/bounswe2020group2/wiki/Sequence-Diagrams

8 Assessment of the final customer presentation

As a group, we successfully presented our web and android interfaces as well as backend endpoints. Presentation was all about sequence of user activities to demonstrate the usefulness of our system. We were able to buy products, rate products, message vendors, see the recommended products, see the active vendor and customer orders etc. Feedback was very positive aside some small points like "does your system supports user purchases from abroad?". We felt like we have convinced our customers in terms of having a website and android application that is working as intended.

9 Requirements

- Glossary
- Requirements
- Functional Requirements
- Non-functional Requirements

9.1 Glossary

- **Active order:** an order whose payment has been done and is being processed to be delivered
- **Admin:** a registered user who manages or organizes the platform
- **Customer:** a registered user who wants to buy some products
- **Delivered order:** an order that has been processed and delivered to the customer
- **Google Sign Up**: a sign up method where the user uses their existing Google account to signup.
- **Guest:** an unregistered user who wants to look for some products
- **List:** a list of products that is created by a user
- **Order:** an order contains information about a purchase of a list of products that were in the shopping cart.
- **Product:** a thing that can be bought and sold
- **Quick Registration**: when a Guest user attempts to checkout their shopping cart, the system asks the user to signup (through a popup/modal form) before being able to continue with the checkout process. This lets the user proceed with the checkout smoothly without any hassle.
- **Registered User:** a user who is using this application or web service after sign up to the system
- **Regular Sign Up**: a sign up method where the user enters their information manually
- **Semantic Search:** a type of search where the contextual information is prioritized over the exact sentence that is being searched. The given string is searched in the following fields:
  - Product Description
  - Product Details
  - Product Title Writing `cat food` in the search might also give search results about `dog food` since `cat` and `dog` are similar to each other semantically (both are domestic animals)
- **Shopping Cart:** a section where the users add products they want to buy and checkout all together, analogous to a real shopping cart.
- **User:** a person who is using this application or web service
- **Vendor:** a registered user who wants to sell some products
- **Verified Account**: is an account whose email has been verified to be valid. When the user signs up and enters their email address, they receive an email from the system asking them to click on a link. By clicking on that link, the user has proven that the email account is valid and reachable by the user.

9.2 Requirements

9.3 1. Functional Requirements

1.1. User Requirements

- **1.1.1 Sign Up**
- **1.1.1.1** Customer and Vendor users shall be able to sign up to the system via one of the following methods:

  - **1.1.1.1.1** Regular sign up
  - **1.1.1.1.2** Google sign up

- **1.1.1.2** Customer, Vendor and Admin users shall be able to use regular sign up by providing the following required information and edit the information whenever they like:

  - **1.1.1.2.1** Username
  - **1.1.1.2.2** Password
  - **1.1.1.2.3** E-mail
  - **1.1.1.2.4** Name
  - **1.1.1.2.5** Surname
  - **1.1.1.2.6** Location (not mandatory for customer and admin users)
  - **1.1.1.2.7** Profile Picture (not mandatory)

- **1.1.1.3** Vendor users shall enter at least one location, while Customer shall be able to enter location information if they wish to

- **1.1.1.4** Customer and Vendor users shall verify their account via a link sent by the system to their e-mail account before being able to checkout a cart.

- **1.1.1.5** Guest users must read and accept the Terms of Services to be able to sign up as customer users.

- **1.1.2 Login**

- **1.1.2.1** Registered Users who signed up with their Google account shall be able to login with their Google account

- **1.1.2.2** Registered Users who signed up with a regular signup shall be able to login by providing the following information (password field will have an show/hide option) .

  - **1.1.2.1.1** Username
  - **1.1.2.1.2** Password

- **1.1.3 Search**

- **1.1.3.1** Guest and Registered Users shall be able to perform a semantic search for a product over:

  - **1.1.3.1.1** Name of the product
  - **1.1.3.1.2** Description of the product
  - **1.1.3.1.3** Type of the product
  - **1.1.3.1.4** Vendor of the product

- **1.1.3.2 Filter**

  - **1.1.3.2.1** Guest and Registered Users shall be able to filter the products according to:

  - **1.1.3.2.1.1** Product's Average customer review (Product rating)
  - **1.1.3.2.1.2** Product vendor
  - **1.1.3.2.1.3** Product brand
  - **1.1.3.2.1.4** Product price range

- **1.1.3.3 Sort**

  - **1.1.3.3.1** Guest and Registered Users shall be able to sort the products according to:
  - **1.1.3.3.1.1** Best sellers
  - **1.1.3.3.1.2** Newest arrivals
  - **1.1.3.3.1.3** Price
  - **1.1.3.3.1.4** Average customer review (Rating)
  - **1.1.3.3.1.5** Number of comments

- **1.1.3.4** Guest and Registered Users shall be able to see the information below about the products they searched:

  - **1.1.3.4.1** Product type
  - **1.1.3.4.2** Product picture
  - **1.1.3.4.3** Price
  - **1.1.3.4.4** Product vendor
  - **1.1.3.4.5** Average Customer Review (Rating)
  - **1.1.3.4.6** Comments about the product
  - **1.1.3.4.6** Description about the product

- **1.1.3.5** Guest and Registered Users shall be able to see the information below about the vendors they searched:

  - **1.1.3.5.1** Company name or personal name
  - **1.1.3.5.2** Company logo picture or personal picture
  - **1.1.3.5.3** Average customer review (Rating)
  - **1.1.3.5.4** Products of this vendor
  - **1.1.3.5.5** Comments about the vendor

- **1.1.3.6** Guest and Registered Users shall be able to search a product by entering their name or type.

- **1.1.3.7** Guest and Registered Users shall be able to search a vendor by entering their name.

- **1.1.4 Product Details**

- **1.1.4.1** Guest and Customer users shall be able to see a product's or a vendor's information even if they did not search for the product (these products and vendors are shown to the customer users in the main page or offered to the customer users at the side of the page. **1.1.3.4** and **1.1.3.5** specifies what information users shall be able to see.

- **1.1.4.2** Vendor users shall be able to update the product's detail information
- **1.1.4.3** Vendor users shall specify the stock information of a product. This requires:
  - Stock status information: `Available`, `Out of Stock`, `Online only`
  - Locations to which the product is deliverable.
- **1.1.5 Comment**
- **1.1.5.1** Customer users shall be able to comment on a product only if they have purchased that product.
- **1.1.5.2** Customer users shall be able to comment on a vendor only if they have purchased a product from that vendor.
- **1.1.5.3** All users shall be able to see the comments about a product or a vendor.
- **1.1.5.4** Vendor users shall be able to reply to comments.
- **1.1.5.5** Customer users shall be able to like or dislike comments.
- **1.1.6 List**
- **1.1.6.1** Customer users shall be able to create a list of products they would like to keep an eye on.
- **1.1.6.2** Customer users shall be able to name their lists.
- **1.1.6.3** Customer users shall be able to edit their lists.
- **1.1.6.4** Customer users shall be able to delete their lists.
- **1.1.7 Shopping Cart**
- **1.1.7.1** Guest and Customer users shall be able to add products to their shopping carts.
- **1.1.7.2** Guest and Customer users shall be able to edit their shopping carts.
- **1.1.7.3** Guest and Customer users shall be able to view their shopping carts.
- **1.1.7.4** Guest users shall be able to checkout their shopping cart only after creating a Customer user account with a Quick registration option.
- **1.1.7.5** Customer users shall be able to checkout their shopping cart only after verifying their account. Please look at **1.1.1.4** and the glossary for details about Account verification.
- **1.1.8 Order**
- **1.1.8.1** Customers shall be able to see their active (not delivered) orders with their status.
  - **1.1.8.1.1** Status shall be `At store` or `At cargo`.

- **1.1.8.2** Guest and Customers shall be able to see their delivered orders with information according to:
  - **1.1.8.2.1** Order date
  - **1.1.8.2.2** Selling price
  - **1.1.8.2.3** Cargo tracking ID
  - **1.1.8.2.4** Cargo tracking link
  - **1.1.8.2.5** Estimated delivery date
- **1.1.8.3** Customers shall be able to cancel their active orders that have the `At store` status.
- **1.1.8.4** Customers shall be able to return their delivered orders.
- **1.1.8.5** Customers shall be able to see the total amount they spent on orders.
- **1.1.8.6** Vendors shall be able to cancel an order during the order processing stage when vendors face a problem related to their order. In such case, the Vendor is required to:
  - specify the reason of the cancellation
  - specify their refund policy
- **1.1.8.7** Vendors shall accept the returns from customers within the 3 days after the purchase date if a customer wants to return a product. Customers do not need to specify any reason for the return.
- **1.1.9 Direct Messaging**
- **1.1.9.1** Customers shall be able to communicate with the vendors or admins about orders via direct message.
- **1.1.9.2** Vendors shall be able to communicate with the customers or admins about a certain product or order via direct message.
- **1.1.10 Notifications**
- **1.1.10.1** Customers shall be able to choose to get notified whenever the price of any product in their list changes.
- **1.1.10.2** Customers shall be able to choose to set an alarm for a certain price threshold in order to be notified about whether the price of selected product downs below the threshold price.
- **1.1.11 Customer Review**
- **1.1.11.1** Customers shall be able to rate a product they have "bought and received".
- **1.1.11.2** Customers shall be able to rate a vendor from which they have "bought and received" a product.
- **1.1.12 Purchase**
- **1.1.12.1** Customer users shall be able to purchase the products in their shopping cart by using one of the following payment methods:.

  - **1.1.12.1.1** Debit card
  - **1.1.12.1.2** Credit card

- **1.1.12.2** Customer users shall agree to the contract of sale before proceeding with the payment.

- **1.1.12.3** Customer users shall be able to see if the payment is successfully completed or not.

- **1.1.11 Online Shop**

- **1.1.11.1** Vendor users shall be able to open an online shop

  - **1.1.11.1.1** Vendor users shall be able to put the following information in their online shop (for reference: https://www.n11.com/magaza/trendimbu):
    * **1.1.11.1.1.1** Vendor information
    * **1.1.11.1.1.2** Active products
    * **1.1.11.1.1.3** Location of physical stores (if any)
  - **1.1.11.1.2** Vendor users should be able to customize some features of the online shop to their liking like changing the colour of the background but in general online shops of the different vendors can be generic so how much a vendor can customize their online shop is a design choice.

1.2. System requirements

- **1.2.1** The system shall have 4 different user types:

- **1.2.1.1** Guest

- **1.2.1.2** Customer

- **1.2.1.3** Vendor

- **1.2.1.4** Admin

- **1.2.2 Signup Functionality**

- **1.2.2.1** The system shall provide Regular signup and Google signup.

- **1.2.2.2** The system shall require acceptance of Terms of Service by the user before continuing with the signup processing.

- **1.2.2.3** The system shall not allow Guest users to checkout their shopping cart

- **1.2.2.4** The system shall confirm the user email by sending a link to the email account and receiving a confirmation from the user.

- **1.2.2.5** The system shall not allow unverified Customer account to checkout their shopping cart

- **1.2.2.6** The system shall only let confirmed users make transactions. Confirmation is via Google or e-mail.

- **1.2.3 Search Functionality**
- **1.2.4.1** The system shall provide a semantic search functionality which finds products and vendors based on contextual information. The information shall be looked for in
  - **1.2.4.1.1** Product title
  - **1.2.4.1.1** Product description
  - **1.2.4.1.1** Product details
- **1.2.3.2** The system shall be able to filter search results according to:
  - **1.2.3.2.1** Average customer review of the product
  - **1.2.3.2.2** Vendor of the product
  - **1.2.3.2.3** Brand of the product
  - **1.2.3.2.4** Price range of the product
- **1.2.3.3** The system shall be able to sort search results according to:
  - **1.2.3.3.1** The rate of sale
  - **1.2.3.3.2** Arrival time
  - **1.2.3.3.3** Price
  - **1.2.3.3.4** Average customer review
  - **1.2.3.3.5** Number of comments
- **1.2.3** Products in the system may be discovered in various categories:
- **1.2.3.1** Arts & Crafts
- **1.2.3.2** Automotive
- **1.2.3.3** Baby
- **1.2.3.4** Beauty & Personal Care
- **1.2.3.6** Books
- **1.2.3.7** Boys' Fashion
- **1.2.3.8** Computers
- **1.2.3.9** Camera & Photo
- **1.2.3.10** Cell Phones & Accessories
- **1.2.3.11** Digital Videos
- **1.2.3.12** Electronics
- **1.2.3.13** Furniture
- **1.2.3.14** Girls' Fashion
- **1.2.3.15** Health & Households

- **1.2.3.16** Home & Garden
- **1.2.3.17** Industrial & Scientific
- **1.2.3.18** Luggage
- **1.2.3.19** Men's Fashion
- **1.2.3.20** Movies & TVs
- **1.2.3.21** Music & CDs
- **1.2.3.22** Pet Supplies
- **1.2.3.23** Software
- **1.2.3.24** Sports & Outdoor
- **1.2.3.25** Toys & Games
- **1.2.3.26** Video Games
- **1.2.3.27** Women's Fashion
- **1.2.4** The system shall distinguish between the same products that are sold by different vendors.
- **1.2.5 Product List Functionality**
- **1.2.5.1** Product Lists created by users shall be private (meaning that only its owner and admins can view the list).
- **1.2.6** The system shall have a shopping cart functionality.
- **1.2.7 Order Functionality**
- **1.2.7.1** System shall show the following information about past orders:
  - **1.2.7.1.1** Order date
  - **1.2.7.1.2** Product price
  - **1.2.7.1.3** Cargo information (Cargo tracking ID and a link to the tracking page)
  - **1.2.7.1.4** Delivery date
  - **1.2.7.1.5** Order status
- **1.2.7.2** The system shall accept transactions made only with Credit Card or Debit Card.
- **1.2.8 Price Tracking and alarm Functionality**
- **1.2.8.1** The system shall track changes in the prices of the products and alert users if requested.
- **1.2.9 Notification Functionality**
- **1.2.9.1** The system shall notify users of direct messages.

- **1.2.9.2** The system shall -if requested- notify users of price changes.
- **1.2.9.3** The system shall -if it is set- notify users of price alarms.
- **1.2.9.4** The system shall notify users of changes in order status.
- **1.2.9.5** The system shall -if requested- notify users of sales and campaigns.
- **1.2.10 Recommendation Functionality**
- **1.2.10.1** The system shall recommend certain products to the users based on their interactions on the platform so recommendation will be personalized.
- **1.2.11 Direct Messaging Functionality**
- **1.2.11.1** The system shall provide direct messaging functionality between customers and vendors and admins via inbox and mail system.
- **1.2.12** The system provides a like and comment functionality but only to the users whose products has been delivered.
- **1.2.13 Product Details Functionality**
- **1.2.13.1** The system must keep the comments and reviews for a product even if that product is editted.
- **1.2.14 Language Functionality**
- **1.2.14.1** The system must offer a language switch option.

9.4 2. Non-Functional requirements

2.1. Availability and Accessibility

- **2.1.1** The system should work on wide range of web browsers (Chrome, Firefox, Safari, Microsoft Edge, Opera).
- **2.1.2** The mobile app shall be developed natively for Android 7 and newer versions.
- **2.1.3** Customer and Vendor users shall be able to verify their account via a link sent by the system to their e-mails.

2.2. Performance

- **2.2.1** System should be responsive (5 seconds in the worst case).
- **2.2.2** Android application should be optimized well to work smoothly with lower hardware resources.

2.3. Security

- **2.3.1** Authentication and authorization shall always be done in the server side.
- **2.3.2** Passwords should be hashed and salted to prevent accounts to be stolen in a database injection.
- **2.3.3** Input to the system shall be properly validated.
- **2.3.4** Sensitive data (such as passwords) should always be encrypted in requests.

- **2.3.5** After 3 unsuccessful login attempts, owner of the account shall be informed via e-mail by the system.

2.4. Protocols and Privacy

- **2.4.1** Rules defined by GDPR and KVKK and shall be followed strictly when developing the system.
- **2.4.2** The implementation of the system should follow the standards introduced by the World Wide Web Consortium (W3C)
- **2.4.3** W3C Activity Streams protocol should be supported by the system.
- **2.4.4** Privacy policy should be defined exactly and shown to users in registration process and users will be forced to accept it to complete registration.
- **2.4.5** Distant sales contract should be defined clearly according to regulations, and customers shall be forced to accept it before purchasing any goods.

2.5. Development and Publish

- **2.5.1** System shall be deployed to Amazon EC2 servers or Digital Ocean.
- **2.5.2** System shall be dockerized to ease development process.
- **2.5.3** System shall have monthly periodic maintenance operations.

10 System manual

10.1 Deploy on AWS

10.1.1 Requirements

- Docker
- Linux
- AWS
- gpg (GPG_PASSWORD is stored in Github Secrets)
- git
- Create an EC2 instance

Steps

- Make sure that the EC2 instance has for name “instance”
- Make sure that it has the ports 80 (frontend), 8000 (backend) and 5432 (postgres) open in its Security Policy
- SSH into the instance and install Docker
- CD into the repo

- Execute the following:

```
./copy_backend_secrets.sh && ./encrypt_secrets.sh
```

- Merge master to deploy and push
- The .github/workflows will execute when pushed to deploy

10.2 Deploy on your local

10.2.1 Requirements

- Docker
- Linux
- git

10.2.2 Steps

To build the images from scratch please follow the commands below: **IMPORTANT**: For the purposes of testing, you can download the docker images from the drive link and continue from Step 4.

1. Clone the repo to your local and switch into it

```
git clone https://github.com/bounswe/bounswe2020group2 && cd bounswe2020group2
```

2. Download and copy .env.production and .env.development to ecommerce/backend

```
cp .env.development ecommerce/backend
cp .env.production ecommerce/backend
```

3. Build the backend and frontend images

```
docker build ecommerce/frontend -t frontend --build-arg REACT_APP_API_URL="http://localhost:8000"
docker build ecommerce/backend -t backend
```

4. Load docker images (skip if you are not in testing)

```
docker pull postgres
docker load -i frontend.tar.gz
docker load -i backend.tar.gz
```

5. Run and setup/restore database

```
docker run --name database --net=host -e POSTGRES_PASSWORD=admin postgres
docker cp dump.sql database:/
docker exec database psql -U postgres -f /dump.sql
```

6. Run backend

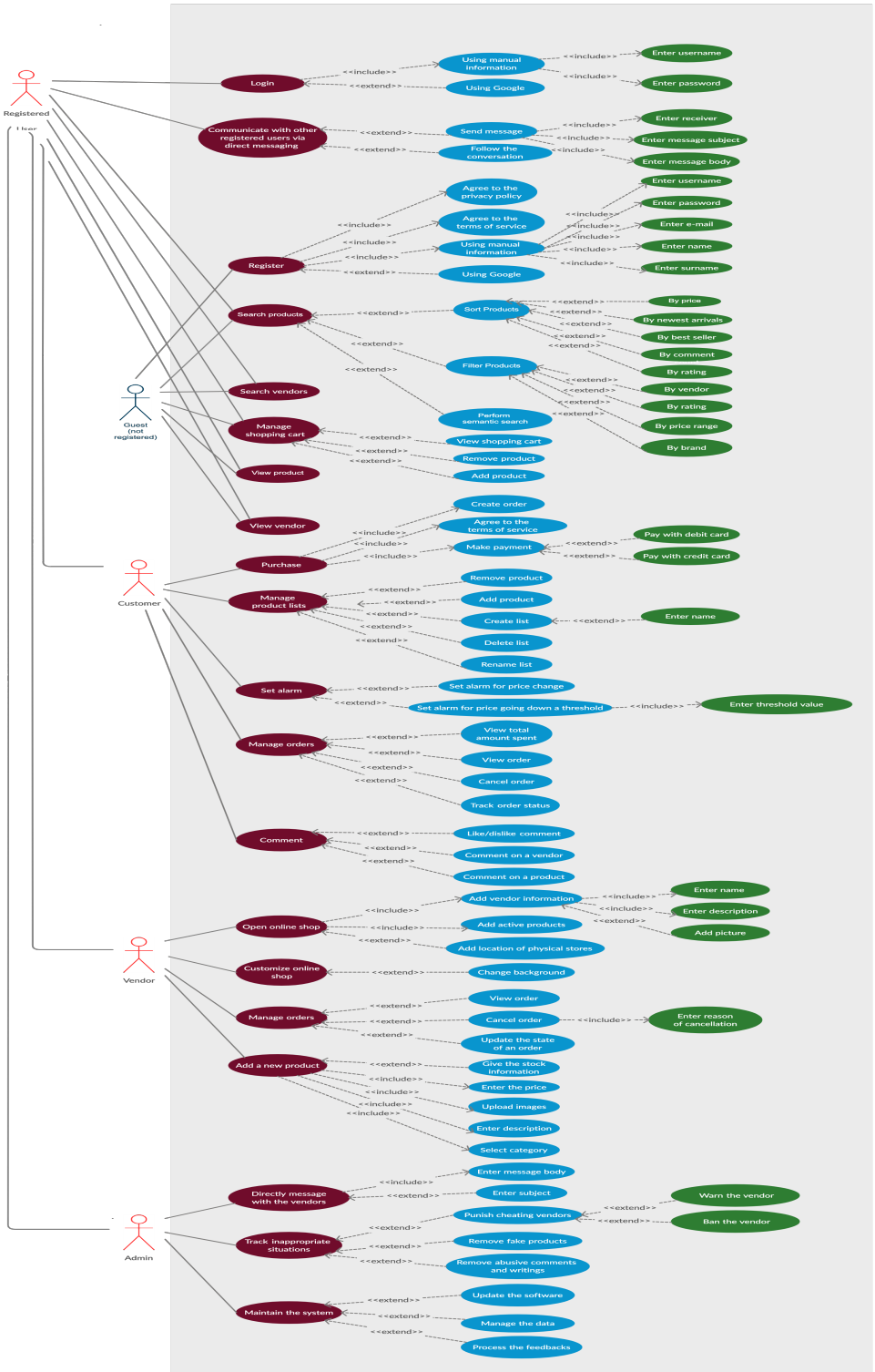
```
docker run --name backend --net=host -e DEBUG=False -e DB_HOST=localhost backend
```

7. Run frontend

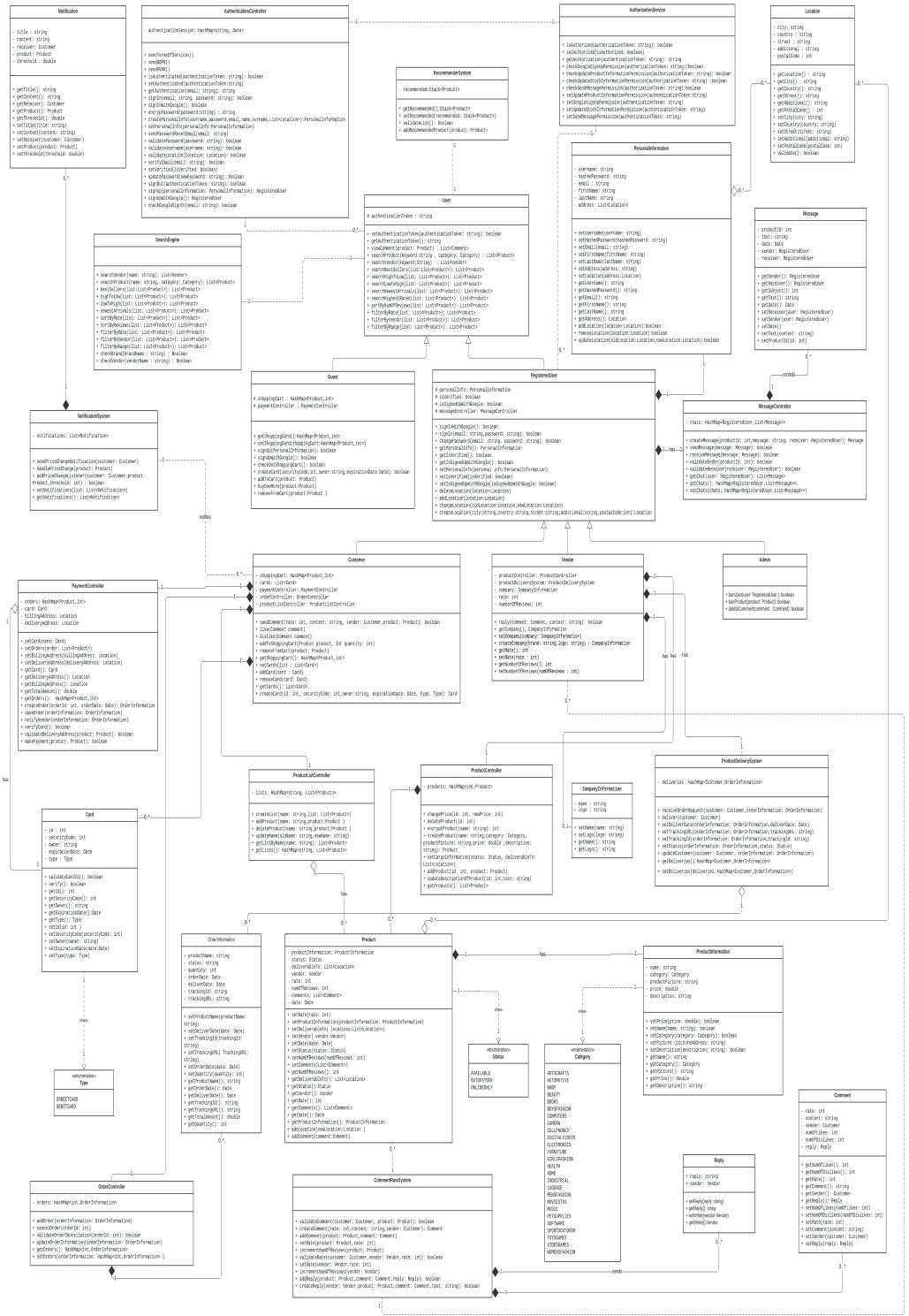
```
docker run --name frontend --net=host frontend
```

8. Open the browser and go to http://localhost

11 Use Case Diagram



12 Class Diagram



13 API Documentation

Link for API documentation can be found at the wiki page. Here are the API endpoints implemented:

URL	Method	Description	Parameters	Response

/init	GET	Initially checks if token is valid	Authorization(header)	{
				"token": "string",
				"id": 0,
				"email": "string",
				"firstname": "string",
				"lastname": "string",
				"is_verified": true,
				"role": "string"
				}

/regularsignup	POST	Signup for customers	body: {
			"username": "string",
			"email": "string",
			"password": "string",
			"firstname": "string",
			"lastname": "string",
			"phonenumber": "string"
			}
				{
				"successful": true,
				"message": "string"
				}

/vendor/signup	POST	Signup for vendors	body: {
			"username": "string",
			"email": "string",
			"password": "string",
			"firstname": "string",
			"lastname": "string",
			"phonenumber": "string",
			"address": {
			"title": "string",
			"name": "string",
			"surname": "string",
			"address": "string",
			"province": "string",
			"city": "string",
			"country": "string",
			"phone": {
			"country_code": "string",
			"number": "string"
			}
			"zip_code": "string"
			}
			}
				{
				"successful": true,
				"message": "string"
				}

/regularlogin	POST	Login with email and password	{
			"email": "string",
			"password": "string"
			}
				{
				"status": {
				"successful": true,
				"message": "string"
				},
				"user":
				{"token": "string",
				"id": 0,
				"email": "string",
				"firstname": "string",
				"lastname": "string",
				"is_verified": true,
				"role": "string"
				}
				}

URL	Method	Description	Parameters	Response

/googlelogin	GET	Login using google apis	id_token(header)	{
				"status": {
				"successful": true,
				"message": "string"
				},
				"user":
				{"token": "string",
				"id": 0,
				"email": "string",
				"firstname": "string",
				"lastname": "string",
				"is_verified": true,
				"role": "string"
				}
				}

/changepassword	POST	*Change password of the user*	body : {
			"password": "string",
			"new_password": "string"
			}
				{
				"status": {
				"successful": true,
				"message": "string"
				}
				}

/search/products	POST	All search done over products	query(header)	{
				results: ["string"]
				}

/products/homepage/{numberOfProducts}	GET	Get products to be shown in home page	numberOfProducts(path)	[{
				"id": 0,
				"name": "string",
				"creation_date": "2021-01-31",
				"discount": 0,
				"price": 0,
				"price_after_discount": 0,
				"total_rating": 0,
				"rating_count": 0,
				"rating": 0,
				"stock_amount": 0,
				"short_description": "string",
				"long_description": "string",
				"category": {
				"id": 0,
				"name": "string"
				},
				"subcategory": {
				"id": 0,
				"name": "string"
				},
				"brand": {
				"id": 0,
				"name": "string"
				},
				"vendor": {
				"id": 0,
				"name": "string",
				"rating": 0
				},
				"images": [
				"url1",
				"url2",
				"url3"
				]
				}
				]

URL	Method	Description	Parameters	Response

/product/{product_id}	GET	Get product details by id	product_id(path)	[{
				"id": 0,
				"name": "string",
				"creation_date": "2021-01-31",
				"discount": 0,
				"price": 0,
				"price_after_discount": 0,
				"total_rating": 0,
				"rating_count": 0,
				"rating": 0,
				"stock_amount": 0,
				"short_description": "string",
				"long_description": "string",
				"category": {
				"id": 0,
				"name": "string"
				},
				"subcategory": {
				"id": 0,
				"name": "string"
				},
				"brand": {
				"id": 0,
				"name": "string"
				},
				"vendor": {
				"id": 0,
				"name": "string",
				"rating": 0
				},
				"images": [
				"url1",
				"url2",
				"url3"
				]
				}
				]

URL	Method	Description	Parameters	Response

/customer/{customer_id}/shoppingcart	GET	get items in shoping cart	Authorization(header)
			customer_id(path)
				{
				"status": {
				"successful": true,
				"message": "string"
				},
				"sc_items": [
				{
				"id": 0,
				"amount": 0,
				"product":{
				"id": 0,
				"name": "string",
				"creation_date": "2021-01-31",
				"discount": 0,
				"price": 0,
				"price_after_discount": 0,
				"total_rating": 0,
				"rating_count": 0,
				"rating": 0,
				"stock_amount": 0,
				"short_description": "string",
				"long_description": "string",
				"category": {
				"id": 0,
				"name": "string"
				},
				"subcategory": {
				"id": 0,
				"name": "string"
				},
				"brand": {
				"id": 0,
				"name": "string"
				},
				"vendor": {
				"id": 0,
				"name": "string",
				"rating": 0
				},
				"images": [
				"url1",
				"url2",
				"url3"
				]
				}
				}
				]
				}

/customer/{customer_id}/shoppingcart	POST	add a new item into shopping cart	Authorization(header)
			customer_id(path)
			body : {
			"password": "string",
			"new_password": "string"
			}
				{
				"sc_item_id:": 0,
				"status": {
				"successful": true,
				"message": "string"
				}
				}

URL	Method	Description	Parameters	Response

/customer/{customer_id}/shoppingcart/{sc_item_id}	GET	get specific item from the shopping cart	Authorization(header)
			customer_id(path)
			sc_item_id(path)
				{
				"status": {
				"successful": true,
				"message": "string"
				},
				"sc_items": [
				{
				"id": 0,
				"amount": 0,
				"product":{
				"id": 0,
				"name": "string",
				"creation_date": "2021-01-31",
				"discount": 0,
				"price": 0,
				"price_after_discount": 0,
				"total_rating": 0,
				"rating_count": 0,
				"rating": 0,
				"stock_amount": 0,
				"short_description": "string",
				"long_description": "string",
				"category": {
				"id": 0,
				"name": "string"
				},
				"subcategory": {
				"id": 0,
				"name": "string"
				},
				"brand": {
				"id": 0,
				"name": "string"
				},
				"vendor": {
				"id": 0,
				"name": "string",
				"rating": 0
				},
				"images": [
				"url1",
				"url2",
				"url3"
				]
				}
				}
				]
				}

/customer/{customer_id}/shoppingcart/{sc_item_id}	PUT	update existing item in the shopping cart	Authorization(header)
			customer_id(path)
			sc_item_id(path)
			body: {
			"product_id": 0,
			"amount": 0
			}
				{
				"status": {
				"successful": true,
				"message": "string"
				}
				}

/customer/{customer_id}/shoppingcart/{sc_item_id}	DELETE	delete an existing item from shopping cart	Authorization(header)
			customer_id(path)
			sc_item_id(path)
				{
				"status": {
				"successful": true,
				"message": "string"
				}
				}

URL	Method	Description	Parameters	Response

/customer/{customer_id}/addresses	GET	get all addresses of the customer	Authorization(header)
			customer_id(path)
				{
				"status": {
				"successful": true,
				"message": "string"
				},
				"addresses": [
				{
				"id": 0,
				"title": "string",
				"name": "string",
				"surname": "string",
				"address": "string",
				"province": "string",
				"city": "string",
				"country": "string",
				"phone": {
				"country_code": "string",
				"number":"string"
				},
				"zip_code": "string"
				}
				]
				}

/customer/{customer_id}/addresses	POST	add a new address	Authorization(header)
			customer_id(path)
			body : {
			"title": "string",
			"name": "string",
			"surname": "string",
			"address": "string",
			"province": "string",
			"city": "string",
			"country": "string",
			"phone": {
			"country_code": "string",
			"number":"string"
			},
			"zip_code": "string"
			}
				{
				"address_id": 0,
				"status": {
				"successful": true,
				"message": "string"
				}
				}

/customer/{customer_id}/addresses/{address_id}	GET	get specific address of the user	Authorization(header)
			customer_id(path)
			address_id(path)
				{
				"status": {
				"successful": true,
				"message": "string"
				},
				"address": {
				"id": 0,
				"title": "string",
				"name": "string",
				"surname": "string",
				"address": "string",
				"province": "string",
				"city": "string",
				"country": "string",
				"phone": {
				"country_code": "string",
				"number":"string"
				},
				"zip_code": "string"
				}
				}

/customer/{customer_id}/addresses/{address_id}	PUT	update existing address	Authorization(header)
			customer_id(path)
			address_id(path)
			body: { "id": 0,
			"title": "string",
			"name": "string",
			"surname": "string",
			"address": "string",
			"province": "string",
			"city": "string",
			"country": "string",
			"phone": {
			"country_code": "string",
			"number":"string"
			},
			"zip_code": "string"
			}
				{
				"status": {
				"successful": true,
				"message": "string"
				}
				}

URL	Method	Description	Parameters	Response

/customer/{customer_id}/addresses/{address_id}	DELETE	delete an existing address	Authorization(header)
			customer_id(path)
			address_id(path)
				{
				”status”: {
				”successful”: true,
				”message”: ”string”
				}
				}

/customer/{customer_id}/cards	GET	get all cards of the customer	Authorization(header)
			customer_id(path)
				{
				”status”: {
				”successful”: true,
				”message”: ”string”
				},
				”cards”: [
				{
				”id”: 0,
				”name”: ”string”,
				”owner_name”: ”string”,
				”serial_number”: ”string”,
				”expiration_data”: {
				”month”: 0,
				”year”: 0
				},
				”cvv”: 0
				}
				
				]
				}

/customer/{customer_id}/cards	POST	add a new card	Authorization(header)
			customer_id(path)
			body: {
			”name”: ”string”,
			”owner_name”: ”string”,
			”serial_number”: ”string”,
			”expiration_data”: {
			”month”: 0,
			”year”: 0
			},
			”cvv”: 0
			}
				{
				”card_id”: 0,
				”status”: {
				”successful”: true,
				”message”: ”string”
				}
				}

/customer/{customer_id}/cards/{card_id}	GET	get specific card of the user	Authorization(header)
			customer_id(path)
			card_id(path)
				{
				”status”: {
				”successful”: true,
				”message”: ”string”
				},
				”card”: {
				”id”: 0,
				”name”: ”string”,
				”owner_name”: ”string”,
				”serial_number”: ”string”,
				”expiration_data”: {
				”month”: 0,
				”year”: 0
				},
				”cvv”: 0
				}
				}

/customer/{customer_id}/cards/{card_id}	PUT	update existing card	Authorization(header)
			customer_id(path)
			card_id(path)
			body:
			{
			”name”: ”string”,
			”owner_name”: ”string”,
			”serial_number”: ”string”,
			”expiration_data”: {
			”month”: 0,
			”year”: 0
			},
			”cvv”: 0
			}
				{
				”status”: {
				”successful”: true,
				”message”: ”string”
				}
				}

/customer/{customer_id}/cards/{card_id}	DELETE	delete an existing card	Authorization(header)
			customer_id(path)
			card_id(path)
				{
				”status”: {
				”successful”: true,
				”message”: ”string”
				}
				}

URL	Method	Description	Parameters	Response

/categories	GET	get the whole category and subcategory information		{
				”categories”: [
				{
				”id”: 0,
				”name”: ”string”,
				”subcategories”: [
				{
				”id”: 0,
				”name”: ”string”
				}
				]
				}
				]
				}

/review	GET	get review items by query	review(query)
			product(query)
			vendor(query)
				{
				”reviews”: [
				{
				”id”: 0,
				”rating”: 0,
				”comment”: 0,
				”product”: {
				”id”: 0,
				”name”: ”string”
				},
				”vendor”: {
				”id”: 0,
				”firstname”: ”string”,
				”lastname”: ”string”
				},
				”review_date”: ”2021-01-31”,
				”reviewed_by”: {
				”id”: 0,
				”firstname”: ”string”,
				”lastname”: ”string”
				}
				
				}
				],
				”status”: {
				”successful”: true,
				”message”: ”string”
				}
				}

/review	POST	post a new review for a vendor or a product	body: {
			”user_id”: 0,
			”product_id”: 0,
			”vendor_id”: 0,
			”rating”: 0,
			”comment”: ”string”
			}
				{
				”review”: {
				”id”: 0,
				”rating”: 0,
				”comment”: 0,
				”product”: {
				”id”: 0,
				”name”: ”string”
				},
				”vendor”: {
				”id”: 0,
				”firstname”: ”string”,
				”lastname”: ”string”
				},
				”review_date”: ”2021-01-31”,
				”reviewed_by”: {
				”id”: 0,
				”firstname”: ”string”,
				”lastname”: ”string”
				}
				},
				”status”: {
				”successful”: true,
				”message”: ”string”
				}
				}

URL	Method	Description	Parameters	Response

/review	DELETE	delete a existing review using its id	body : {
			"id": 0
			}
				{
				"review": {
				"id": 0,
				"rating": 0,
				"comment": 0,
				"product": {
				"id": 0,
				"name": "string"
				},
				"vendor": {
				"id": 0,
				"firstname": "string",
				"lastname": "string"
				},
				"review_date": "2021-01-31",
				"reviewed_by": {
				"id": 0,
				"firstname": "string",
				"lastname": "string"
				}
				},
				"status": {
				"successful": true,
				"message": "string"
				}
				}

/vendor/product	POST	adds a new product to the shop of vendor	body: {
			"name": "string",
			"price": 0,
			"stock_amount": 0,
			"short_description": "string",
			"long_description": "string",
			"discount": 0,
			"brand_id": 0,
			"subcategory_id": 0,
			"images": [
			"string"
			]
			}
				{
				"id": 0,
				"status": {
				"successful": true,
				"message": "string"
				}
				}

/vendor/product	PUT	updates product of a vendor	body: {
			"name": "string",
			"price": 0,
			"stock_amount": 0,
			"short_description": "string",
			"long_description": "string",
			"discount": 0,
			"brand_id": 0,
			"subcategory_id": 0,
			}
				{
				"id": 0,
				"status": {
				"successful": true,
				"message": "string"
				}
				}

/vendor/product	DELETE	delete a existing product	body : {
			"id": 0
			}
				{
				"id": 0,
				"status": {
				"successful": true,
				"message": "string"
				}
				}

/checkout/details	GET	get prices of shopping cart	Authorization(header)	{
				"products_price": 0,
				"delivery_price": 0,
				"discount": 0,
				"total_price": 0
				}

URL	Method	Description	Parameters	Response

/checkout/payment	POST	put shopping cart items into purchase and order table	Authorization(header)
			body: {
			"address_id": 0,
			"card_id": 0
			}
				{
				"status": {
				"successful": true,
				"message": "string"
				}
				}

/checkout/cancelorder/{id}	POST	cancel specific order	Authorization(header)
			id(path)
				{
				"status": {
				"successful": true,
				"message": "string"
				}
				}

/customer/orders	GET	get orders of customer	Authorization(header)	{
				"status": {
				"successful": true,
				"message": "string"
				} }

/messages	GET	get all messages of the logged in user	Authorization(header)	{
				"status": {
				"successful": true,
				"message": "string"
				
				},
				"conversations": [
				{
				"counterpart": {
				"id": 0,
				"name": "string"
				
				},
				
				"messages": [
				{
				"id": 0,
				"text": "string",
				"sent_by_me": true,
				"date": "string",
				"attachment_url": "string"
				}
				]
				}
				]
				}

/messages	POST	send a message for the logged in user	Authorization(header)
			body: {
			"receiver_id": 0,
			"text": "string",
			"attachment": "string"
			}
				{
				"status": {
				"successful": true,
				"message": "string"
				}
				}

/lists	POST	create product list	Authorization(header)
			body: {
			"name": "string"
			}
				{
				"id": 0,
				"status": {
				"successful": true,
				"message": "string"
				}
				}

URL	Method	Description	Parameters	Response

/lists	GET	get product list	Authorization(header)	{
				”status”: {
				”successful”: true,
				”message”: ”string”
				},
				”lists”: [
				{
				”list_id”: 0,
				”name”: ”string”,
				”products”: [
				{
				”id”: 0,
				”name”: ”string”,
				”creation_date”: ”2021-01-31”,
				”discount”: 0,
				”price”: 0,
				”price_after_discount”: 0,
				”total_rating”: 0,
				”rating_count”: 0,
				”rating”: 0,
				”stock_amount”: 0,
				”short_description”: ”string”,
				”long_description”: ”string”,
				”category”: {
				”id”: 0,
				”name”: ”string”
				},
				”subcategory”: {
				”id”: 0,
				”name”: ”string”
				},
				”brand”: {
				”id”: 0,
				”name”: ”string”
				},
				”vendor”: {
				”id”: 0,
				”name”: ”string”,
				”rating”: 0
				},
				”images”: [
				”url1”,
				”url2”,
				”url3”
				]
				}
				]
				}
				]
				}

/lists/{id}	DELETE	delete product list	Authorization(header)
			id(path)
				{
				”status”: {
				”successful”: true,
				”message”: ”string”
				}
				}

/lists/{id}/product/{product_id}	DELETE	delete product from list	Authorization(header)
			id(path)
			product_id(path)
				{
				”status”: {
				”successful”: true,
				”message”: ”string”
				}
				}

/lists/{id}/product/{product_id}	POST	add product to list	Authorization(header)
			id(path)
			product_id(path)
				{
				”status”: {
				”successful”: true,
				”message”: ”string”
				}
				}

URL	Method	Description	Parameters	Response

/recommendation	GET	Get products to recommended	Authorization(header)	{
				"status": {
				"successful": true,
				"message": "string"
				},
				"products": [
				{
				"id": 0,
				"name": "string",
				"creation_date": "2021-01-31",
				"discount": 0,
				"price": 0,
				"price_after_discount": 0,
				"total_rating": 0,
				"rating_count": 0,
				"rating": 0,
				"stock_amount": 0,
				"short_description": "string",
				"long_description": "string",
				"category": {
				"id": 0,
				"name": "string"
				},
				"subcategory": {
				"id": 0,
				"name": "string"
				},
				"brand": {
				"id": 0,
				"name": "string"
				},
				"vendor": {
				"id": 0,
				"name": "string",
				"rating": 0
				},
				"images": [
				"url1",
				"url2",
				"url3"
				]
				}
				]
				}