CMPE451 - Milestone 1 Report

Group Members

Burak Çuhadar

Mehmet Erdinç Oğuz

Koray Çetin

Eylül Yalçınkaya

Emre Girgin

Berkay Alkan

M. Olcayto Türker

Çağrı Çiftçi

Alperen Bağ

Veli Can Ünal

Meriç Üngör

Berke Can Gürer

Fall, 2020

Table of Contents

Executive Summary

Introduction

Buyo is an e-commerce platform where customers can find a variety of products from different vendors. The platform provides a functional interface for customers to search for and buy products as well as vendors to sell their products. Intuitive user interactions and global standards are important for the platform to be successful.

Work Done So Far

This semester, we started by remembering and going over what we did last semester. We explained the project to new group members. First, we focused on updating our requirements and design documents from last year because our project is based on them. We discussed the necessary changes with the customer and got their approval. We also updated our project plan as our project timeline became more clear.

Before starting to implement, we split into three groups: backend, frontend, android. We decided to implement the most basic parts of the project in order to deliver an MVP to the first milestone. We implemented login, sign up, home page, products page, wishlist page and product details page in frontend and android. Backend team also prepared the related API endpoints and deployed the backend side. Thanks to good cooperation and communication within the team, we were able to deliver our product to the first customer meeting.

Road Ahead

We implemented our interfaces for the customer user before the milestone. Since we got more acquainted with the technologies we use, we plan to speed up our implementation process and prepare interfaces for the vendor user as well. We also want to develop the payment flow for the customer.

Deliverable Status and Evaluation

For the first milestone, we had a platform where you can sign-up / login, then surf through the products and categories and add them to your wishlist as the main (might be called as umbrella deliverable) deliverable.

Backend

Deployment of the Backend	Done
Functional Database	Done
Mock Data	Done
Updated Project Details	Done
Login/Signup as User/Vendor	Done
Get Category/Products Endpoints	Done
Wishlist Endpoint	Done
Search	Partially Done
Shopping Cart	Postponed
Forgot My Password	Postponed

1. **Deployment of the Backend - DONE:** It was necessary for the first milestone as we needed to provide a working backend for the customer.
2. **Functional Database - DONE:** To add actual products in the future, we made sure that we have a database working with the endpoints we wrote.
3. **Add Mock Data - DONE:** We added the mock data for vendors and their products for our demos. We also have products obtained from some public APIs.
4. **Update Project Details from Last Semester - DONE:** We changed the diagrams and requirements this semester since what we have in mind as a platform has evolved until the first milestone.
5. **Login/Signup as User/Vendor - DONE:** Login/signup functionality is required for the users to later experience the other features, such as buying products or handling the orders as a vendor, of the platform.
6. **Get Category/Products Endpoints - DONE:** Our platform should be able to show the products and categories available to the user in our UI, so these endpoints were

mandatory for our first milestone. Also, the user interacts with products a lot, so we decided that it would speed up the backend implementation process in the future if we started building it ASAP.

7. **Wishlist Endpoint - DONE:** We wanted to have something similar to the shopping cart but not as complicated as such; so, we decided to have a working wishlist for the user.
8. **Search - PARTIALLY DONE:** We wanted to have search functionality for the first milestone, but we decided that it would be overhead to have semantic search for the first milestone. So, we only made a keyword search for the first milestone.
9. **Shopping Cart - POSTPONED:** Since we had the wishlist, we decided to postpone the shopping cart to the next milestone, and integrate it to the platform along with the payment system.
10. **Forgot My Password - POSTPONED:** We thought it would be better to have it along with the authorization of the users when they login.

Mobile

Base App	Done
Navigation Tabs	Done
Homepage	Done
Product List	Done
Category List	Done
Product Detail Page	Done
Wishlist	Done
Login/Signup	Done
Search	Partially Done
User Authentication	Done

Base App - DONE: The base app consists of a set of classes, interfaces, and functions that allow us to develop the mobile app in a more robust and consistent way. It has wrapper classes for endpoint request, fragment navigation, and user authentication. Also, we used the MVMM design pattern for development. These features need to be implemented before we start any implementation so it was the first step for our plan. For the following milestones, probably there will be only minor changes on this and everything else is going to be built on the base app.

Navigation Tabs - DONE: The navigation tabs represent the main 5 sections of our app. Fragments are stacked as one on top of each other in each tab and each tab has its own fragment stack. This feature allowed us to manage tabs separately like different apps. Since the flow of user scenarios require the interaction of more fragments and tabs, this feature had to be implemented in advance, too, like the base app. Thus, this deliverable was placed at the first steps of our project plan, and everything after that implemented regarding this design choice.

Homepage - DONE: The Homepage is implemented in terms of both UI and functionality, except the recommendation and discounted products logic. These sections only provide a random list of products for the concerns of the first milestone. This is the landing page for our app when someone starts to use the app. Thus, it is important to have in the first milestone, it will be enhanced for the following milestones, like campaign banners and recommended products.

Product List - DONE: The Product List shows a list of products for the desired search query and category. The customer user can add products to his/her wishlist and remove them. For the following milestones, there will be a “Buy Directly” button.

Category List - DONE: Category List shows the main and sub-categories of the products present in the database. It directs to the Product List page after a category is selected. It has its final version and there is no plan to update it later on.

Product Detail Page - DONE: On this page, all types of users may explore a certain product. The user may see the details and comments on that product and add it to its wishlist and cart. For the first milestone, it is planned to have only the “Like” button and UI of comments fragment but for the following milestones, there will add to cart and see comments functionality.

Wishlist - DONE: The customer may add a product to its wishlist and remove it. For the first milestone, there is not a system called a collection. At the final milestone, there will be collections and add to cart directly options. Also, the customer will be able to sort the products in his/her wishlist.

Login/Signup - DONE: For the first milestone we only have signup and login for customer type of user and after login, the user is directed to an empty fragment. For the following milestones, there will be a vendor signup/login section getting the address of the vendor for instance. There will not be an admin login in the mobile app. Also, after login, the user will be directed to the appropriate fragment or profile page directly.

Search - PARTIALLY DONE: Search was not a part of the first milestones scenarios but it works in terms of UI and functionally. However, it only does a syntactic search, rather than a semantic search. After the search, the user is directed to the product list page listing the products returned by the search endpoint. For the following milestones, it is planned to conduct a semantic search.

User Authentication - DONE: For the user authentication, the login endpoint returns a user id and it is stored in a global variable. Using the technique called shared pref, it is checked if a user is logged in the system when a registered user action is taken. For the following milestones, a tokenization mechanism may be used, instead of just user id.

Frontend

Homepage	Done
Navigation Bar	Done
User Authentication	Done
Product List	Postponed
Product Details Page	Done
Deployment, Dockerization, CI/CD	Partially Done
Wishlist	Done
Search	Done
Best Sellers, Recommended, etc.	Done

Homepage - DONE: The Homepage was planned for the first milestone because we decided to go with the order of a user's journey. To demonstrate something for the first milestone, we felt like the customer needed to see something else than a blank page when we first landed on the website. Besides from the navigation bar seen on every page, we implemented a deals section and modular product block component we can use for Best Sellers, Recommended, Featured, etc. sections. Even though we populated these sections with mock data since we planned these on the backend for the second milestone, the UI and functional implementation is done.

Navigation Bar - DONE: The Navigation Bar is among the most essential deliverables for the first milestone. We need it to navigate through any new page we implement throughout the project. Also since it is present on every page it would be really hard to implement the user interface of our pages imagining that the navigation bar is there compared to it being actually there. It consists of an icon that takes the user to the homepage, a search bar, wishlist, profile and cart icon and product categories. The categories are pulled from the backend when the page first mounts and rendered accordingly. Their subcategories are rendered as submenus that open when we click on the categories. Besides from redirection to unimplemented pages, we can say that this deliverable is completely done.

User Authentication - DONE: We decided to plan the user authentication before anything else because a lot of functionality on our website depends on a user account. So it would be foolish to implement those before the user authentication. We have a Sign in and a Sign Up page implemented. These pages ask the user for the necessary information, then send them to our backend and act according to the response we get. Then if the response is positive, we update our redux state with the signed in user so the functionalities that need it can use it easily.

Product List - POSTPONED: Since our website is about buying products, showing these products is essential in most of its flow. Like adding products to cart, wishlist or seeing the details of a product to add comments, etc. So it is only logical that we prioritized implementing a product list in the first milestone. Unfortunately we could not start on this and had to postpone it to the second milestone. But since we planned the whole project by giving ourselves plenty of breathing room, we were able to pull this deliverable to the second milestone without compromising anything else from the second to the third milestone. Also having a really similar functionality with the wishlist page really helped in this manner.

Product Details Page - DONE: The Product Details Page is the next page in our user journey. Also since we primarily execute adding products to the wishlist, adding products to the cart, commenting on products actions on this page, it made sense to give it priority. The general UI implementation is completed along with adding the product to the wishlist and displaying comments with mock data. Adding products to the cart and adding comments are yet to be implemented because we need more functionality from the backend for those.

Deployment, Dockerization, CI/CD - PARTIALLY DONE: By prioritizing deployment, dockerization and CI/CD we achieve a medium where we can test everyone's work getting combined with minimal effort. This really increases a software team's efficiency. Also it provides the customer a medium to follow our progress with no technical effort. So we realized that sooner we dealt with this, more time we would save. We achieved deployment on the AWS servers, but unfortunately we bumped into some problems and spent a lot of time solving them but we couldn't yet solve Dockerization and couldn't even start looking into CI/CD in time for the milestone 1. So we postponed those parts for the first action items of milestone 2.

Wishlist - DONE: Even though this functionality does not block most of the other features on the website, we decided to be done with the next step of our user journey. Also if we got started with Cart, Checkout flow, Filter, etc. we probably wouldn't have time to show the customer something meaningful. So we decided to go with the wishlist so that we can get input on additional functionality. The backend endpoints were implemented as well so we were able to fully implement the wishlist besides from the sublists in the wishlist page, which wasn't planned for the first milestone anyway.

Search - DONE: The search functionality we are talking about here is only the user interface logic. The actual searching was planned for the second milestone on the backend so we implemented this just so our navigation bar would be complete and it wouldn't be extra work to integrate it later.

Best Sellers, Recommended, etc - DONE: Similar to the search bar, the backend side for this deliverable was planned for the second milestone. But a UI implementation of a modular Product Block Component was done which can be filled with any group of products and a title to create Best Sellers, Recommended or something else we decide in the future easily. And we filled it with mock data so that we can get a proper input on the homepage in the first milestone.

Summary of Coding Work

Eylül Yalçınkaya (Frontend)	I created high fidelity designs for frontend pages, and the frontend team used these designs' .css files to implement the UI's. I implemented the UI for sign up and sign in. I implemented the API calls to the backend for these pages. I also joined troubleshooting sessions with the frontend team to fix bugs and review the code.
Meriç Üngör (Frontend)	I constructed the initial structure of the React project. I have set up the folder structure, some utilities like history management and routing, Redux and Reduxthunk frameworks. I also implemented the UI, functionality and API calls necessary for the homepage, header and a modular product card component used throughout the website. Finally I managed dockerization and deployment of our website on AWS servers.
Mehmet Erdinc Oguz (Backend)	Me and Veli Can together have acted as the DevOps of our team and shared tasks until the first milestone. I deployed the backend code to our AWS EC2 server. I researched the CI/CD process but haven't applied it to our codebase yet.(will be done until the end of the week) I took part in the decision process for the app architecture. I wrote the endpoint returning the product information. I also joined the "initialization of the

	backend" sessions where we collaboratively created the backend structure. Finally, I helped to solve bugs related to Docker, signup and login endpoints, and deployment.
Alperen Bağ (Android)	I created the UI of the homepage and login/signup page. I also implemented the product API connection. In addition to my individual work, I also joined coding sessions (on zoom) with my team for bug fixing and the initialization of the base application structure.
Koray Cetin (Backend)	We've created our express app with mongoose connection with Olcayto. Also I've taken a significant part in manipulating and adding mock products to our database, creating the endpoints for the first milestone and configuring the app architecture. And I took a small part in docker configuration, about the issue we encounter in mongodb connection.
Emre Girgin (Android)	I prepared the UI of the Product List page and Product Detail page according to the design prepared in advance, in XML. I coded a Repository and ViewModel for products. I rendered the information retrieved from the backend to the product list page, homepage recommendation / discount (mock data). I also connected the like/dislike feature to the wishlist modelView and the product detail and product list pages. I also attended all general meetings and android team meetings (about planning, debugging, branch merging, etc.) I prepared the requirements and list of deliverables and evaluation of the Android part of this report.
Olcayto Türker (Backend)	We created our first endpoints and database with Koray. For the database part, I

	contributed in getting data from our mock products and integration of them to our database including assigning vendors, points and stock information to each of the products. Also we created databases for users(customer and vendor) and wishlists. I took part in development of our first endpoints such as login, signup, basic search function, addition/deletion to wishlist, displaying products from a category/subcategories.
Burak Çuhadar (Frontend)	I created the product page, where the details of the product and comments on the product is displayed. I developed adding to and removing from wishlist functionality. I also created the wishlist page where the user can view the products in the wishlist. At these pages all the data is requested through API calls from the backend except the comments. I also helped to fix a bug about saving state when the user logs in.
Berkay Alkan (Android)	I prepared the UI design of the wish list page by writing the XML file of it. I created a repository, view model, adapter and fragment classes for the wish list. I created the endpoints for the wish list end liking or disliking a product. I did the API connection part of the products in the wish list and the half of the dislike feature in the wish list page. In addition I joined all coding meetings with my team where we made discussions on the project and bug fixes.
Veli Can Ünal (Backend)	I worked on creating the first deployed system. However my aws server was not free. We created a new server with Erdinç. Erdinç and I worked as DevOps for the first steps of our project. We also searched for the CI/CD processes and how to use Jenkins. I dockerize the backend code for deploying our systems. I checked the sequence diagrams and created a sequence diagram for searching. I documented the backend development page for the other developers. I fixed signing up bugs with my friends' help.
Çağrı Çiftçi(Android)	I prepared the base structure of the app.

	I added dagger(dependency injection) and retrofit(api calls). I implemented base class and some helper class for fragment transactions and api calls. I also prepared some example classes to show some basic concepts about android to my teammates. I created the UI of the categories page. I also implemented categories API connection. Also we made some android meetings for coding and debugging with my teammates.

Challenges Met During Deployment, Dockerizing and CI/CD

Frontend

- The first challenge we faced was to choose the environment we were going to deploy in. Since services that simplifies this process for frontend like Heroku, AWS Amplify were off-limits, we decided to go with good-old AWS EC2 servers. The reason was because they had a really decent free-tier option, and since AWS is arguably the industry-leader in web services and cloud computing, it has really good documentation and great community support.
- The second challenge was to get acquainted with Docker. This was fairly simple since Docker has decent documentation and informative videos. So we quickly produced a Dockerfile that installs the necessary packages on our project and generates a build file for our React app. Then we needed to figure out how exactly those build files of our website would appear on our web browser when we entered the dns address of our EC2. After some research, we realized that AWS automatically redirects any http request coming into the IP address to the port 80. So we needed to set up a server that exposes web files to port 80. And we decided to use nginx for this. Since we are not really experts on networks and we can’t really judge the functionality of nginx, our decision was mostly caused by again documentation and community support.

- Manually building the react application and then serving it with nginx was working pretty fine so we pretty much handled deployment of our website. But for some reason integrating nginx into our Dockerfile didn’t seem to work no matter what. We were going through a lot of different resources and even found a lot of example Dockerfiles written for the exact same scenario as us but they didn’t even work. At first docker could not locate the build folder it created before, then it succeeded to locate it and seemed like everything was working but nothing seemed to happen. Finally we decided to get rid of nginx and just use the serve command of npm. That worked when we manually executed every step as well but didn’t seem to work when we defined every step in the Dockerfile, even though the commands gave the proper outputs of working fine.

Backend

- The first challenge we faced was to use the AWS EC2 server. The machine failed to recognize our .pem file due to setting the file’s ownership wrongly. We used all of our free-trial duration in our first month for the first machine we created, so we needed to create another one from a different account. Our new machine didn’t recognize our .pemfile, so our old authorization key needed to be added to the new machine’s key register.
- The second challenge was to dockerize MongoDB. There was an error when we created the database with the mock data causing us to lose all the data. So we created a separate endpoint for initializing the database with mock data.
- The final challenge is to get used to CI/CD implementation. We learned how to do it but weren't able to decide on how to implement it on our codebase. We finally decided on having one machine holding all the codebase, and having separate branches for each team so that when a team’s member pushes to that branch, the automation system will update the master and deploy it to the EC2 server. We had trouble at the first time because we had separate machines for each team and we lost some time to apply CI/CD to our workflow. Now, collaboratively we will manage to apply CI/CD to our workflow in a short time.

Glossary and Requirements

Glossary

- Campaign: Campaigns include discounts for buying in bulk (discount on a single product is not included) such as but not limited to: buying 3 items but paying for 2, discount of 10% for orders more than 100 TL, free shipment.
- Cart: A functionality for customer users that enables them to save the products they want to buy. The customers can then order the products saved in the cart to buy them all at once.
- Combinative search: Searching mechanism that allows users to filter the results of a search by multiple attributes. For example, a user searching for 'hat' can filter the results by criteria A and B, and then sort by Y.
- Comment: Text displayed on product pages written by customers. Customers explain and discuss their shopping experience via this functionality.
- Customer: A logged-in user with the special privileges of creating wishlists, ordering products and tracking orders.
- Guest user: A non-logged in user with the ability to search, display and read comments on products.
- GDPR: Abbreviation(General Data Protection Regulation) for the regulation in European Union law on data protection and privacy in the European Union and the European Economic Area. Its primary goal is to give control to individuals over their personal data.
- KVKK: Abbreviation(Kişisel Verilerin Korunması Kanunu) for the law introduced by Republic of Turkey to regulate how personal data should be respected, used and processed by individuals and organizations to protect basic rights and liberties of citizens.
- **Wishlist: A page for customers to keep track of the products they are interested in.**
- **Collection A subcategory of wishlist where the customers can add their products they are interested in.**
  (*The name "list" was a bit confusing and not clear about what it wanted to mean. Hence, we introduced the notion of wishlist and collection - as a subsection of a wishlist - instead of list, to remove this misconception.***)**
- Notification: Message sent by the platform to notify users of an event (i.e. item price changes, direct messages, etc.)
- Order processing stage: Time-frame between a customer ordering product and them receiving it.

- Orders Page: A special page for customers and vendors. Customers can track their orders and interact with their orders on this page (e.g. cancel an order, message a vendor, etc.). Vendors can also track the orders made for their products and cancel the orders in case of a problem (such as stock issues).
- Platform: The software product as a whole.
- Product: Any item that is for sale on the platform.
- Product Page: A page for details about the product. Price of the product, comments and rating on the product made by customers, and details of the product are displayed on this page.
- PWA: A website that looks and behaves as if it is a mobile app. PWAs are built to take advantage of native mobile device features, without requiring the end user to visit an app store, make a purchase and download software locally.
- Semantic Search: Searching mechanism where the results are generated by considering searcher's intent, context of the query and the relationships between words. A semantic search engine tries to understand the meaning of the query to improve search accuracy. For example in this project's context when a customer searches for "clothes for summer", search engines should understand what is meant by the query and generate appropriate results.
- User: People that interact with the platform.
- Vendor: Class of users that are offering products in exchange for money.

Requirements

1. Functional Requirements

1.1 User Requirements

- **1.1.1 User Types**
    - **1.1.1.1 Customer**
        - 1.1.1.1.1 Customers shall be able to search for products.
        - 1.1.1.1.2 Customers shall be able to filter searched products based on average customer review, brand, **category,** vendor, price range and **available filters for that category.**
          (*We have to implement a category system for the user to explore for the sake of Project Description. Thus we decided to add the category as a filtering option to improve the user experience.***)**
        - 1.1.1.1.3 Customers shall be able to sort searched products based on best-selling, latest arrival time, price, average customer review, number of comments, and rating.
        - 1.1.1.1.4 Customers shall have a **Wishlist and Collection.;**
          (*As mentioned in the Glossary part, we introduced "Wishlist" and*

*"Collection" notions as a replacement for List. More detailed explanations are present in the section of Wishlist.(**1.2.9**)*)

    - 1.1.1.1.4.1 Customers shall be able to save collections, to keep track of items they wish to buy and to purchase the same items multiple times.
- 1.1.1.1.5 Customers shall have a Cart page.
    - 1.1.1.1.5.1 Customers shall be able to add products to their cart.
    - 1.1.1.1.5.2 Customers shall be able to remove products from their cart.
    - 1.1.1.1.5.3 Customers shall be able to purchase all items from their cart at once.
- 1.1.1.1.6 Customers shall have an Orders page.
    - 1.1.1.1.6.1 Customers shall be able to see all their purchased items in the Orders page.
    - 1.1.1.1.6.2 Customers shall be able to cancel their active orders in the Orders page before they have been shipped.
    - 1.1.1.1.6.3 After the items have been delivered, customers shall be able to return the delivered orders in the Orders page.
    - 1.1.1.1.6.4 Customers shall be able to see the total amount of money they have spent so far in the Orders page. The total differs based on the actions the customer has taken, such as returning an item or cancelling an order.
    - **1.1.1.1.6.5 Customers shall be able to file a complaint about the order to inform the vendor and the admin.**
      (*In the project description, it is said that the vendor can communicate with Admins about a certain product or order. We thought that customers also have the right to complain to the vendor or the admin if a problem arises.***)**
- 1.1.1.1.7 Customers shall be able to receive notifications.
    - 1.1.1.1.7.1 Customers shall be notified when an order is delivered.
    - 1.1.1.1.7.2 Customers shall be able to choose to be notified if the price of a chosen product changes.
    - 1.1.1.1.7.3 Customers shall be able to choose to be notified if a product that is out of stock is available again.
- 1.1.1.1.8 Customers shall be able to comment on the products that they have purchased, after the products have been delivered.
- 1.1.1.1.9 Customers shall be able to rate the products that they have purchased, after the products have been delivered.
- 1.1.1.1.10 Customers shall be able to read user comments about products.
- **1.1.1.1.11 Customers shall give their names and address information before ordering their product.**
- **1.1.1.1.12 Customers shall be able to edit their address during the ordering process or at their profile page.**
  (*The address of the customer was not mentioned clearly, although it is an essential part of the ordering process. Thus, we specified it clearly.* **)**

        - **1.1.1.2 Vendor**
            - 1.1.1.2.1 Vendors shall be able to keep track of their orders via the orders page.
            - 1.1.1.2.2 Vendors shall specify at least one store location through Google Maps.
            - 1.1.1.2.3 Vendors shall be able to cancel an order which is in the order processing stage.
            - 1.1.1.2.4 Vendors shall be able to specify a reason while cancelling an order.
            - 1.1.1.2.5 Vendors shall be notified if an order is canceled.
            - 1.1.1.2.6 Vendors shall be able to make a discount on a product for a certain time span.
            - 1.1.1.2.7 Vendors shall be able to create campaigns for certain products for a certain time span.
            - 1.1.1.2.8 Vendors shall not be able to make purchases using their vendor account.
            - 1.1.1.2.9 Vendors should specify their stock information while adding a product to the platform.
            - 1.1.1.2.10 Vendors shall be able to read user comments about products.
        - **1.1.1.3 Guest User**
            - 1.1.1.3.1 Guests shall be able to search for products.
            - 1.1.1.3.2 Guests shall be able to view the prices.
            - 1.1.1.3.3 Guests shall be able to read user comments about products.
            - 1.1.1.3.4 Guests shall be required to sign up during their purchase process.
        - **1.1.1.4 Admin**
            - 1.1.1.4.1 Admins shall be able to ban customers or vendors.
            - 1.1.1.4.2 Admins shall be able to remove malicious comments.
- **1.1.2 Social Interactions**
    - 1.1.2.1 Customer users shall send messages to vendors who are sellers of the products they buy.
    - 1.1.2.2 Vendors shall be able to communicate with platform admins about a certain product/order.
- **1.1.3 Sign up**
    - 1.1.3.1 Customer and vendor users shall provide an email, an username and a password to sign up.
    - 1.1.3.2 Vendors shall provide their location to sign up.
    - 1.1.3.3 E-mail address and user type pairs shall be unique. (i.e. there can be a customer and a vendor account using the same email.
    - 1.1.3.4 User passwords should be in the form of alphanumeric and at least 6 characters long.
    - 1.1.3.5 Google account should be used for signing up/in.
    - 1.1.3.6 During registration, the email address should be verified with a confirmation mail.
    - 1.1.3.7 All user types shall accept a KVKK agreement to sign up.

- **1.1.3.8 Users shall be able to change their passwords without signing in by providing their email addresses.**
  (*This is a common feature in most of the projects requiring accounts for the user. It is commonly known as the feature of "Forgot Password". If a user forgets his or her password, then s/he can have a new one by email verification.***)**

1.2 System Requirements

- **1.2.1 Search**
  - **1.2.1.1 Searching Mechanism**
    - 1.2.1.1.1 Searching Mechanism shall utilize product names, features, vendor names,vendor location, customer reviews and ratings.
    - 1.2.1.1.2 Searching Mechanism shall include semantic search (search for similar products).
    - 1.2.1.1.3 Searching Mechanism shall include combinative search.
  - **1.2.1.2 Filtering Mechanism**
    - 1.2.1.2.1 Products shall be filtered by average customer review.
    - 1.2.1.2.2 Products shall be filtered by brand.
    - 1.2.1.2.3 Products shall be filtered by vendor.
    - 1.2.1.2.4 Products shall be filtered by price range.
    - 1.2.1.2.5 Products shall be filtered by product specified features (size, color, technical features).
  - **1.2.1.3 Sorting Mechanism**
    - 1.2.1.3.1 System shall sort products by popularity (bestsellers).
    - 1.2.1.3.2 System shall sort products by the time of release (new arrivals).
    - 1.2.1.3.3 System shall sort products by price.
    - 1.2.1.3.4 System shall sort products by customer review.
    - 1.2.1.3.5 System shall sort products by number of comments.
    - 1.2.1.3.6 System shall sort products by rating.
- **1.2.2 Interactions**
  - 1.2.2.1 Customers shall be able to comment on the products that they have purchased, after the products have been delivered.
  - 1.2.2.2 Customers shall be able to rate the products that they have purchased, after the products have been delivered.
  - 1.2.2.3 After making an order, customers shall be able to message the vendor(s) of the item(s) directly.
  - 1.2.2.3 Vendors shall be able to message admins, e.g. to cancel an order.
  - 1.2.2.4 Customer users shall send messages to vendors who are sellers of the products they buy.
  - **1.2.2.5 The names of the customers shall be visible only with their initials on top of the comments they make for privacy. ex: M**** U******
    (*To sort a list of products by their comments, we also need to show the comments publicly for the sake of transparency. However, due to privacy concerns, we decided to hide the name of the reviewer, except the initials so that the reviewer is ensured that his/her review is counted and displayed.***)**

- **1.2.3 Interface**
    - 1.2.3.1 System shall have categories for discovering new products.
- **1.2.4 Product**
    - 1.2.4.1 When the same product is sold by different vendors, product data shall be distinguished.
- **1.2.5 Order Process**
    - **1.2.5.1 Customer's Orders Page**
        - 1.2.5.1.1 Orders shall be exactly in one of those states, such as Pending, Being Prepared, Being Delivered, Pending for Customer Approval, Completed, Cancelled, Refund.
        - 1.2.5.1.2 For both active or completed cases, there shall be product information, order date, product price, cargo information, and delivery date.
        - 1.2.5.1.3 There shall be options for canceling active orders that are not at the Being Delivered stage or returning completed ones.
        - 1.2.5.1.4 For each case (active, canceled, completed, returned), the total amount of money spent shall be available. In terms of canceled and returned products belonging to a shopping basket, users shall see the amount of money they get back when they check their basket.
        - **1.2.5.1.5 There shall be only one shipment option, which is BUYO Express.**
        - **1.2.5.1.6 Customers shall pay with Turkish Lira.**
          (*The shipment and payment currency options were fuzzy. We specified them to be more clear.***)**
    - **1.2.5.2 Vendor's Orders Page**
        - 1.2.5.2.1 Orders shall be exactly in one of those states, such as Pending, Being Prepared, Being Delivered, Completed, Cancelled, Refund.
        - 1.2.5.2.2 For both active or completed cases, there shall be product information, order date, product price, cargo information, and delivery date.
        - 1.2.5.2.3 Vendors shall be able to communicate with platform admins about a certain product/order.
        - 1.2.5.2.4 Vendors shall be able to cancel an order (e.g. if out of stock) during order processing stage.
        - 1.2.5.2.5 Vendors shall be able to see all orders and the total earnings.
        - 1.2.5.2.6 Vendors shall be able to disable and re-enable new orders for a product (e.g. if out of stock).
- **1.2.6 Notifications**
    - 1.2.6.1 Customers shall be able to choose notified if the price of a **wishlisted** product changes. **(***Wishlist is the replacement of List.***)**
    - 1.2.6.2 Customers shall be able to choose notified if a product that is out of stock is available again.
    - 1.2.6.3 Customers shall be notified when the order is delivered.
    - 1.2.6.4 Vendors shall be notified if an order is canceled.
- **1.2.7 Recommendation**
    - 1.2.7.1 The platform shall recommend certain products to the users based on their interactions on the platform.

- **1.2.8 Payment**
    - 1.2.8.1 When a customer makes a purchase, the platform shall keep the money for three days. When the user approves that they got the product, money shall be transferred to the vendor.
    - 1.2.8.2 If the cargo has not arrived to the customer yet, the customer shall extend the period for approval of the product.
    - 1.2.8.3 System shall not accept the payment if the customer does not approve an e-commerce shopping agreement.
    - 1.2.8.4 Customers shall approve an KVKK agreement after writing the payment information.
- **1.2.9 Wishlist**
    - **1.2.9.1 Customers shall be able to create private collections that they can name, edit, delete, add products to, e.g. to keep track of the items' prices.**
    - **1.2.9.2 Customers shall create empty collections before adding any products.**
    - **1.2.9.3 Products that get removed from a wishlist, are also removed from all the collections.**
    - **1.2.9.4 Products that get removed from a collection, do not get removed from the wishlist.**
    - **1.2.9.5 Wishlist includes all the products from all the collections.**
      (*Most of the specifications about Wishlist - List as the previous name - were under the section of Product (1.2.4). After we changed it to Wishlist and Collection we wanted to create a separate section for it and specify the relation between Wishlist and Collection.***)**

2. Non-functional Requirements

2.1 Availability

- 2.1.1 There shall be a web application and native Android applications.
- 2.1.2 The mobile app should not be a PWA or hybrid application like Angular & Ionic combination or Cordova.
- 2.1.3 Platform shall be deployable to a configurable server, Amazon EC2 or Digital Ocean.
- 2.1.4 Platform shall be Dockerized.
- 2.1.5 Open-source software with appropriate use permissions shall be used, as long as it is properly attributed and documented, and unless otherwise specified.
- 2.1.6 System shall support Turkish and English characters.
- 2.1.7 Direct messaging features implemented between different user types shall be in real-time.
- 2.1.8 Users shall be able to access the platform via a web browser or Android mobile device.

2.2 Standards

- 2.2.1 System shall support W3C Activity Streams protocol so that the activities on the platform are expressed as a stream – including the actions taken by vendors e.g. adding a new product, discounts.
- 2.2.2 System shall follow W3C standards and the standards of the used packages.

2.3 Privacy

- 2.3.1 System shall follow rules defined by GDPR and KVKK.
- 2.3.2 The personal information, contact information, copyrighted contents, license issues and everything related to these paradigms should be respected and considered.

2.4 Security

- 2.4.1 User passwords should be encrypted before stored in the database.
- 2.4.2 Users should be sent a verification mail when they want to change their passwords. Until the users click the link in the mail, their passwords should not be changed.

2.5 Performance

- 2.5.1 The maximum response time of the system should be 2 seconds.
- 2.5.2. The system should be able to respond up to 1000 users simultaneously.

Design Documents

Changes in Class Diagram

Registered User

We added new fields to class such as kvkk accepted, address, name and surname. Since we should keep users' information according to KVKK, we realized that we should ask users to accept it to proceed to next functions in our app. Also, when we checked requirements, registered users must give their name and surname to us to sign-up, so we added these fields. Also we added an address field to registered users to save. We added one function to the registered user class named forgotMyPassword for sending them an email if they forget their password.

Vendor User

We didn’t change fields in that class but added one more function named createTag for adding tags to vendor’s products.

Location

We deleted the user field and added addressName and postalCode to Location class. Also, we added updateLocation and deleteLocation functions to Location class.

Customer User

We deleted cart and comments fields from customer user class and added a wishlist field to Customer User because we decided to move cart and comments to other classes and it would be sufficient to add a wishlist field.

Product

We added size, original price, tagList, campaignId and color fields to product class for providing information about previous price before a campaign or discount, tags a product has, campaign details if applied on that product and color. We changed the stockValue field to make it easier to get stock information by giving only color or size information of that product. Also we added one function named addTag to that class for assigning a tag for that product.

Order

We added an id field to the order class for reaching them with only id info and address field because all individual orders have one address and we might need to reach that information.

Notification Type

We added new fields to Notification Type class such as shippingSuccess, shippingFailed, orderShipped, orderReceived and discountWarning. We added these fields because we realized that we would need to notify people about these issues.

Comment

We deleted boughtProduct a user can comment if he/she bought this product so this field was redundant. Also we changed the product field with productId for reaching that product's information by using its id.

Cart

We created a new class Cart for indicating user's cart information as a whole. In this class, there is a products array consisting of the products in the customer's cart, userId indicating which user that shopping cart belongs to and totalCartAmount which keeps the total price of that cart. We put one function named calculate price for calculating this cart's total price.

Campaign

We added Campaign class for separating campaigns with each other. So, we added id, title, bannerUrl and campaignInfo fields to campaign class. Id, title and bannerUrl is given in the constructor class but campaignInfo field is filled with the information came from campaignInfo class as a parameter in applyCampaign function in Campaign class.

Campaign Type

We created a new enum class named Campaign Type. This class has three fields indicating campaign type enumerations such as free shipping, buy x pay y and x% discount. We added these three because these are the all campaign types we stated in our requirements.

Campaign Info

We added CampaignInfo class for getting campaign type from Campaign Type class and giving that information to Campaign class. So, Campaign Info class works like a bridge class for campaigns in our app.

Payment

We created a Payment class to use in payment operations in our app. This class has four fields such as amount, date, code and creditCartNumber. It has a constructor function as its only function.

Feature

We deleted Feature class from our previous class diagram because we decided we can manage all features from Product class and moved Feature Class' fields and functions to product class.

Image

We deleted Image class from our previous class diagram because turning this class to a field in Product class is much easier and there is no need to call two functions with this method.

Changes in Sequence Diagrams

We didn't delete anything from our previous Sequence Diagrams, but we added Login, Signup, Search Product, Show Product and Show Category Content sequence diagrams to our list. We decided to add these sequence diagrams after determining the final scenario which will be demonstrated to the customer. These sequence diagrams were in our scenario but not in our sequence diagrams, so we added them.

Changes in Use Case Diagram

We didn't delete anything from our previous Use Case Diagram, but we added forgot password use case for the registered user, entering name and address before making an order and filing a complaint to the admin about the order for the customer.

API Documentation

Base URL: http://localhost:8080/

Base URL on Amazon server: http://3.138.113.101:8080/

About the App

This is the backend of an e-commerce product which serves web and android applications.

How to Run?

Once you are inside the backend/product_service folder, install docker and run

```
docker-compose build
docker-compose up
```

Web app is available on the base URL, you can query the app with any request tool.

Endpoints

All of the endpoints return responses in JSON format.

Get Categories

Returns an inspirational quote from a famous, historical person.

Endpoint: /categories

Method: GET

Authorization: None

Parameters: None

Response: `{ status: { success: true, code: 200 }, data: { categories: [{ name: "Woman", path: "Woman", subcategories: [{ name: "Shoes", path: "Woman,Shoes", subcategories: []}] } ]}}`

Get Products of a Category

Gets the products of a category.

Endpoint: /products

Method: GET

Authorization: None

Parameters(Query): categories=["Kiz Cocuk", "Cocuk", "Kazak"]

Response: { status: { code: 200, message: "OK", }, data: { products: [{ category: ["Cocuk", "Kiz Cocuk", "Kazak"], sizes: null, colors: ["Red", "Blue", "White", "Purple", "Orange", "Black", "Grey", "Green"], name: "Kiz Cocuk Kalin Triko Kazak", id: 10002, imageUrl: "https://img-lcwaikiki.mncdn.com/mnresize/230/-/productimages/20192/4/3870373/l_20192-9wp531z4-gzn_a.jpg", rating: 4.92, price: 19.99, originalPrice: 39.99, brand: "Nike", stockValue: { Red: 82, Blue: 37, White: 35, Purple: 15, Orange: 53, Black: 26, Grey: 49, Green: 17, }, vendor: { name: "Ahmet", rating: 3.22, }, }, { category: ["Cocuk", "Kiz Cocuk", "Kazak"], sizes: null, colors: ["Red"], name: "Kiz Cocuk Kalin Triko Kazak", id: 10102, imageUrl: "https://img-lcwaikiki.mncdn.com/mnresize/230/-/productimages/20192/4/3870373/l_20192-9wp531z4-gzn_a.jpg", rating: 0.83, price: 19.99, originalPrice: 39.99, brand: "Jack & Jones", stockValue: { Red: 95, }, vendor: { name: "AyseTeyze", rating: 3.21, }, },], }, };

Get Products with Keyword

Returns the products whose title is the keyword.

Endpoint: /products

Method: GET

Authorization: None

Parameters(Query): keyword=cocuk

Response: { status: { code: 200, message: "OK", }, data: { products: [{ category: ["Cocuk", "Erkek Cocuk", "Ic Giyim"], sizes: null, colors: ["Red", "Blue", "White", "Purple", "Orange", "Black"], name: "Erkek Cocuk Pamuklu Atlet 2'li", id: 10001, imageUrl: "https://img-lcwaikiki.mncdn.com/mnresize/230/-/pim/productimages/20201/4041406/l_20201-0w1007z4-jyx_a.jpg", rating: 0.78, price: 22.99, originalPrice: 22.99, brand: "Adidas", stockValue: { Red: 17, Blue: 69, White: 72, Purple: 57, Orange: 71, Black: 44, }, vendor: { name: "Ahmet", rating: 3.22, }, }, { category: ["Cocuk", "Kiz Cocuk", "Kazak"], sizes: null, colors: ["Red", "Blue", "White", "Purple", "Orange", "Black", "Grey", "Green"], name: "Kiz Cocuk Kalin Triko Kazak", id: 10002, imageUrl: "https://img-lcwaikiki.mncdn.com/mnresize/230/-/productimages/20192/4/3870373/l_20192-9wp531z4-gzn_a.jpg", rating: 4.92, price: 19.99, originalPrice: 39.99, brand: "Nike", stockValue: { Red: 82, Blue: 37, White: 35, Purple: 15, Orange: 53, Black: 26, Grey: 49, Green: 17, }, vendor: { name: "Ahmet", rating: 3.22, }, },], }, };

Get Product Detail

Returns the detailed information of a product.

Endpoint: /products

Method: GET

Authorization: None

Parameters(Query): id=10003

Response: { "status": { "code": 200, "message": "OK" }, "data": { "result": { "category": ["Bebek", "Erkek Bebek", "Tulum"], "sizes": null, "colors": [], "name": "Erkek Bebek Baskili Uyku Tulumu", "id": 10003, "imageUrl": "https://img-lcwaikiki.mncdn.com/mnresize/230/-/pim/productimages/20202/4457947/l_20202-0wg929z1-lsf_a.jpg", "rating": 0.53, "price": 89.99, "originalPrice": 89.99, "brand": "Watsons", "vendor": { "name": "Ahmet", "rating": 3.22 } } } }

Like / Dislike

Triggers the like or dislike event from a user to a product.

Endpoint: /like

Method: POST

Authorization: None

Parameters(Query): customerId=12341&productId=10003

Response: { "status": { "code": 200, "message": "OK" } }

Get Wishlist

Gets the wishlist of a user.

Endpoint: /wishlist

Method: GET

Authorization: None

Parameters: customerId=12341

Response: { "status": { "code": 200, "message": "OK" }, "data": { "products": [{ "category": ["Cilt", "Yüz Bakımı", "Yüz Maskesi"], "sizes": null, "colors": null, "name": "Saf Kil Detoks Maskesi 50 ml", "id": 12312, "imageUrl": "https://img-watsons.mncdn.com/Content/Images/Thumbs/0320616_saf-kil-detoks-maskesi-

50-ml.png", "rating": 1.2, "price": 69.95, "originalPrice": 69.95, "brand": "Nike", "stockValue": { "self": 45 }, "vendor": { "name": "AyseTeyze", "rating": 3.21 } }] } }

Login

Performs the login event of a user.

Endpoint: /covid19

Method: POST

Authorization: None

Parameters(Query): userType=customer&email=mark@zucker.org&password=1234

Response: { "status": { "code": 200, "message": "OK" }, "data": { "userId": 10 } }

Sign-up

Performs the sign-up event for a user.

Endpoint: /covid/news

Method: POST

Authorization: API key from .env file

Parameters(Query): userType=customer&email=mark@zucker.org&password=1234

Response: { "status": { "code": 200, "message": "OK" }, "data": { "userId": 10 } }

Reset the Mock Database

This endpoint is created for internal purposes, it can be used when the database is needed to be initialized or reset.

Endpoint: /db/init

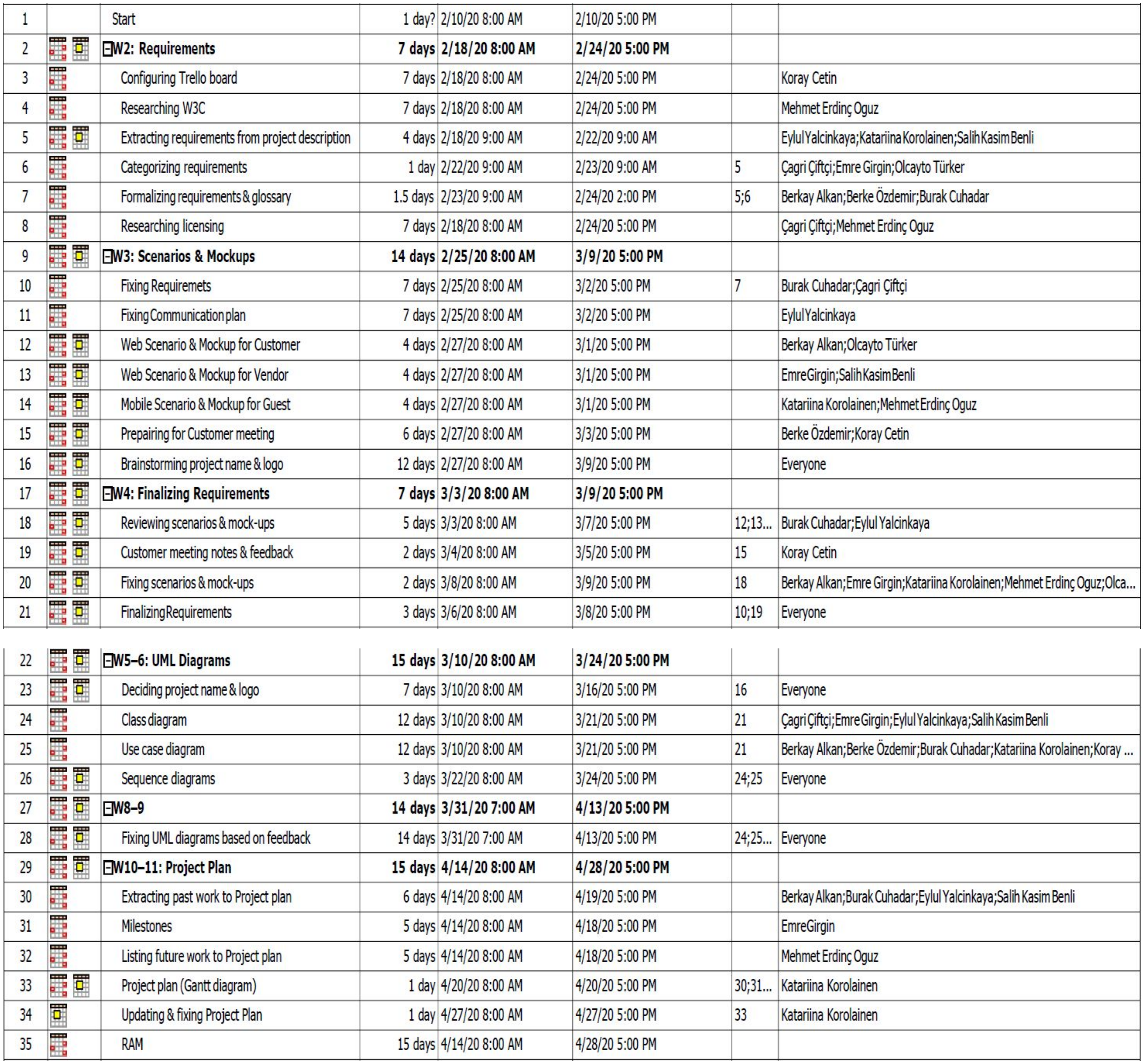
Method: POST

Authorization: API key from .env file

Parameters: None

Response: { "status": { "code": 200, "message": "OK" } }

Project Plan

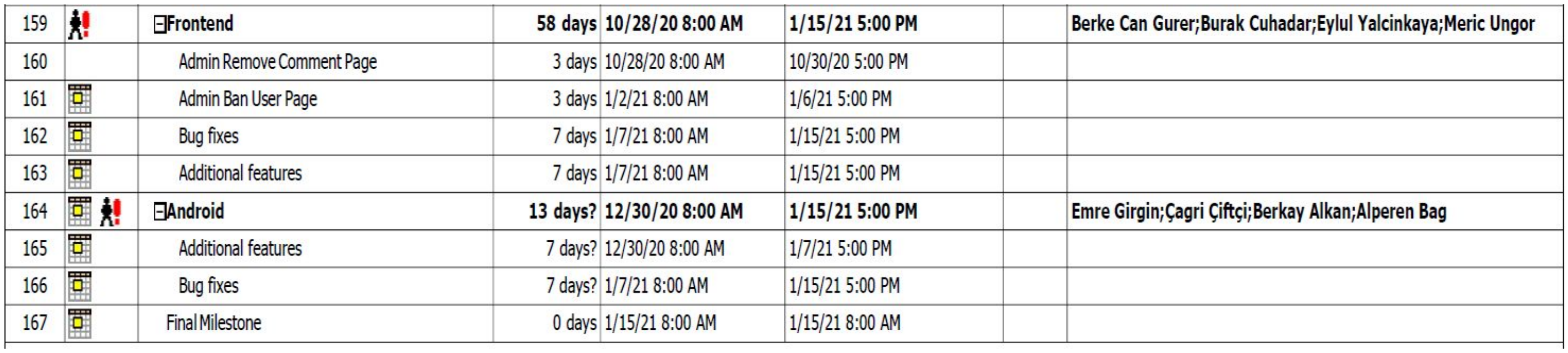


1		Start	1 day?	2/10/20 8:00 AM	2/10/20 5:00 PM		
2	▦ ▣	**⊟W2: Requirements**	**7 days**	**2/18/20 8:00 AM**	**2/24/20 5:00 PM**		
3	▦	Configuring Trello board	7 days	2/18/20 8:00 AM	2/24/20 5:00 PM		Koray Cetin
4	▦	Researching W3C	7 days	2/18/20 8:00 AM	2/24/20 5:00 PM		Mehmet Erdinç Oguz
5	▦ ▣	Extracting requirements from project description	4 days	2/18/20 9:00 AM	2/22/20 9:00 AM		Eylul Yalcinkaya;Katariina Korolainen;Salih Kasim Benli
6	▦	Categorizing requirements	1 day	2/22/20 9:00 AM	2/23/20 9:00 AM	5	Çagri Çiftçi;Emre Girgin;Olcayto Türker
7	▦	Formalizing requirements & glossary	1.5 days	2/23/20 9:00 AM	2/24/20 2:00 PM	5;6	Berkay Alkan;Berke Özdemir;Burak Cuhadar
8	▦	Researching licensing	7 days	2/18/20 8:00 AM	2/24/20 5:00 PM		Çagri Çiftçi;Mehmet Erdinç Oguz
9	▦ ▣	**⊟W3: Scenarios & Mockups**	**14 days**	**2/25/20 8:00 AM**	**3/9/20 5:00 PM**		
10	▦	Fixing Requiremets	7 days	2/25/20 8:00 AM	3/2/20 5:00 PM	7	Burak Cuhadar;Çagri Çiftçi
11	▦	Fixing Communication plan	7 days	2/25/20 8:00 AM	3/2/20 5:00 PM		Eylul Yalcinkaya
12	▦ ▣	Web Scenario & Mockup for Customer	4 days	2/27/20 8:00 AM	3/1/20 5:00 PM		Berkay Alkan;Olcayto Türker
13	▦ ▣	Web Scenario & Mockup for Vendor	4 days	2/27/20 8:00 AM	3/1/20 5:00 PM		Emre Girgin;Salih Kasim Benli
14	▦ ▣	Mobile Scenario & Mockup for Guest	4 days	2/27/20 8:00 AM	3/1/20 5:00 PM		Katariina Korolainen;Mehmet Erdinç Oguz
15	▦ ▣	Prepairing for Customer meeting	6 days	2/27/20 8:00 AM	3/3/20 5:00 PM		Berke Özdemir;Koray Cetin
16	▦ ▣	Brainstorming project name & logo	12 days	2/27/20 8:00 AM	3/9/20 5:00 PM		Everyone
17	▦ ▣	**⊟W4: Finalizing Requirements**	**7 days**	**3/3/20 8:00 AM**	**3/9/20 5:00 PM**		
18	▦ ▣	Reviewing scenarios & mock-ups	5 days	3/3/20 8:00 AM	3/7/20 5:00 PM	12;13...	Burak Cuhadar;Eylul Yalcinkaya
19	▦ ▣	Customer meeting notes & feedback	2 days	3/4/20 8:00 AM	3/5/20 5:00 PM	15	Koray Cetin
20	▦ ▣	Fixing scenarios & mock-ups	2 days	3/8/20 8:00 AM	3/9/20 5:00 PM	18	Berkay Alkan;Emre Girgin;Katariina Korolainen;Mehmet Erdinç Oguz;Olca...
21	▦ ▣	Finalizing Requirements	3 days	3/6/20 8:00 AM	3/8/20 5:00 PM	10;19	Everyone
22	▦ ▣	**⊟W5–6: UML Diagrams**	**15 days**	**3/10/20 8:00 AM**	**3/24/20 5:00 PM**		
23	▦ ▣	Deciding project name & logo	7 days	3/10/20 8:00 AM	3/16/20 5:00 PM	16	Everyone
24	▦	Class diagram	12 days	3/10/20 8:00 AM	3/21/20 5:00 PM	21	Çagri Çiftçi;Emre Girgin;Eylul Yalcinkaya;Salih Kasim Benli
25	▦	Use case diagram	12 days	3/10/20 8:00 AM	3/21/20 5:00 PM	21	Berkay Alkan;Berke Özdemir;Burak Cuhadar;Katariina Korolainen;Koray ...
26	▦ ▣	Sequence diagrams	3 days	3/22/20 8:00 AM	3/24/20 5:00 PM	24;25	Everyone
27	▦ ▣	**⊟W8–9**	**14 days**	**3/31/20 7:00 AM**	**4/13/20 5:00 PM**		
28	▦ ▣	Fixing UML diagrams based on feedback	14 days	3/31/20 7:00 AM	4/13/20 5:00 PM	24;25...	Everyone
29	▦ ▣	**⊟W10–11: Project Plan**	**15 days**	**4/14/20 8:00 AM**	**4/28/20 5:00 PM**		
30	▦	Extracting past work to Project plan	6 days	4/14/20 8:00 AM	4/19/20 5:00 PM		Berkay Alkan;Burak Cuhadar;Eylul Yalcinkaya;Salih Kasim Benli
31	▦	Milestones	5 days	4/14/20 8:00 AM	4/18/20 5:00 PM		Emre Girgin
32	▦	Listing future work to Project plan	5 days	4/14/20 8:00 AM	4/18/20 5:00 PM		Mehmet Erdinç Oguz
33	▦ ▣	Project plan (Gantt diagram)	1 day	4/20/20 8:00 AM	4/20/20 5:00 PM	30;31...	Katariina Korolainen
34	▣	Updating & fixing Project Plan	1 day	4/27/20 8:00 AM	4/27/20 5:00 PM	33	Katariina Korolainen
35	▦	RAM	15 days	4/14/20 8:00 AM	4/28/20 5:00 PM		

36	**⊟W12**	**7 days**	**4/28/20 8:00 AM**	**5/4/20 5:00 PM**		
37	Executive Summary	4 days	4/28/20 8:00 AM	5/1/20 5:00 PM		
38	List & status of deliverables	4 days	4/28/20 8:00 AM	5/1/20 5:00 PM		
39	Evaluating the status of the deliverables	4 days	4/28/20 8:00 AM	5/1/20 5:00 PM		
40	Evaluating tools & processes	4 days	4/28/20 8:00 AM	5/1/20 5:00 PM		
41	Summary of work done by each member	4 days	4/30/20 8:00 AM	5/3/20 5:00 PM	35	
42	Communication plan	4 days	4/30/20 8:00 AM	5/3/20 5:00 PM	11	
43	Requirements	4 days	4/30/20 8:00 AM	5/3/20 5:00 PM	21	
44	Scenarios & Mockups	4 days	4/30/20 8:00 AM	5/3/20 5:00 PM	20	
45	UMLDiagrams	4 days	4/30/20 8:00 AM	5/3/20 5:00 PM	28	
46	Project Plan	4 days	4/30/20 8:00 AM	5/3/20 5:00 PM	34	
47	RAM	4 days	4/30/20 8:00 AM	5/3/20 5:00 PM		
48	Milestone 1	0 days	5/4/20 5:00 PM	5/4/20 5:00 PM	37;38...	
49	**⊟W13: API Assignment**	**14 days**	**5/2/20 8:00 AM**	**5/15/20 5:00 PM**		
50	Finding APIs	3 days	5/2/20 8:00 AM	5/4/20 5:00 PM		Everyone
51	Creating RESTful API	9 days	5/5/20 8:00 AM	5/13/20 5:00 PM	50	Everyone
52	Unit testing	9 days	5/5/20 8:00 AM	5/13/20 5:00 PM		Everyone
53	Pull requests & reviews	11 days	5/5/20 8:00 AM	5/15/20 5:00 PM		Everyone
54	**⊟W14**	**6 days?**	**5/12/20 7:00 AM**	**5/18/20 8:00 AM**		
55	Executive Summary	4 days	5/12/20 7:00 AM	5/15/20 5:00 PM		
56	List & status of deliverables	4 days	5/12/20 7:00 AM	5/15/20 5:00 PM		
57	Evaluating the status of the deliverables	4 days	5/12/20 7:00 AM	5/15/20 5:00 PM		
58	Evaluating tools & processes	4 days	5/12/20 8:00 AM	5/15/20 5:00 PM		
59	Summary of work done by each member	4 days	5/14/20 7:00 AM	5/17/20 5:00 PM		
60	API Documentation	4 days	5/14/20 8:00 AM	5/17/20 5:00 PM		
61	Milestone 2	0 days?	5/18/20 7:00 AM	5/18/20 8:00 AM	55;56...	
62	BREAK	100 days?	5/18/20 9:00 AM	10/5/20 9:00 AM		
63	**⊟SECONDSEMESTER**	**60 days?**	**10/28/20 8:00 AM**	**1/19/21 5:00 PM**		
64	Update the requirements	3 days?	10/28/20 8:00 AM	10/30/20 5:00 PM		
65	Extract features from the requirements	5 days?	10/30/20 5:00 PM	11/6/20 5:00 PM		
66	Discuss possible new requirements	8 days?	10/28/20 8:00 AM	11/6/20 5:00 PM		
67	Teamdistribution	8 days?	10/28/20 8:00 AM	11/6/20 5:00 PM		
68	Create the design of the product	8 days?	10/28/20 8:00 AM	11/6/20 5:00 PM		
69	**⊟Backend**	**10 days?**	**11/3/20 8:00 AM**	**11/16/20 5:00 PM**		**Koray Cetin;Mehmet Erdinç Oguz;Olcayto Türker;Veli Can Unal**
70	Research: Docker	4 days	11/3/20 8:00 AM	11/6/20 5:00 PM		
71	Research: AWS	4 days	11/3/20 8:00 AM	11/6/20 5:00 PM		
72	Get some mock data	4 days?	11/3/20 8:00 AM	11/6/20 5:00 PM		
73	Deployment	3 days?	11/12/20 8:00 AM	11/16/20 5:00 PM		
74	Integrate Jenkins	3 days?	11/12/20 8:00 AM	11/16/20 5:00 PM		
75	Integrate Docker	3 days?	11/12/20 8:00 AM	11/16/20 5:00 PM		
76	Update Class Diagrams	3 days?	11/12/20 8:00 AM	11/16/20 5:00 PM		
77	Update Sequence Diagrams	3 days?	11/12/20 8:00 AM	11/16/20 5:00 PM		
78	Initialize DB	3 days?	11/12/20 8:00 AM	11/16/20 5:00 PM		
79	Add some endpoints for testing	3 days?	11/12/20 8:00 AM	11/16/20 5:00 PM		

80		Connect endpoints with database	3 days?	11/12/20 8:00 AM	11/16/20 5:00 PM		
81		Research: Kubernetes	3 days?	11/12/20 8:00 AM	11/16/20 5:00 PM		
82		**⊟Backend**	**6 days**	**11/17/20 8:00 AM**	**11/24/20 5:00 PM**		**Mehmet Erdinç Oguz;Koray Cetin;Olcayto Türker;Veli Can Unal**
83		Homepage Recommendation Endpoint	5 days	11/17/20 8:00 AM	11/23/20 5:00 PM		
84		Login/SignUp as Vendor	4 days	11/17/20 8:00 AM	11/20/20 5:00 PM		
85		Login/SignUp as User	2 days	11/17/20 8:00 AM	11/18/20 5:00 PM		
86		Forgot My Password	1 day	11/17/20 8:00 AM	11/17/20 5:00 PM		
87		Category/Product Endpoint	6 days	11/17/20 8:00 AM	11/24/20 5:00 PM		
88		Wishlist for User	2 days	11/17/20 8:00 AM	11/18/20 5:00 PM		
89		Search	6 days	11/17/20 8:00 AM	11/24/20 5:00 PM		
90		Vendor Data	6 days	11/17/20 8:00 AM	11/24/20 5:00 PM		
91		Shopping Cart	2 days	11/17/20 8:00 AM	11/18/20 5:00 PM		
92		**⊟Frontend**	**17 days?**	**11/3/20 8:00 AM**	**11/25/20 5:00 PM**		**Berke Can Gurer;Burak Cuhadar;Eylul Yalcinkaya;Meric Ungor**
93		React and Redux research	7 days	11/3/20 8:00 AM	11/11/20 5:00 PM		
94		Project Setup	1 day?	11/10/20 8:00 AM	11/10/20 5:00 PM		
95		Integrate Redux	1 day?	11/11/20 8:00 AM	11/11/20 5:00 PM		
96		Homepage UI	2 days	11/12/20 8:00 AM	11/13/20 5:00 PM		
97		Navbar UI	3 days	11/14/20 8:00 AM	11/18/20 5:00 PM		
98		Login/Signup Page UI	7 days	11/10/20 8:00 AM	11/18/20 5:00 PM		
99		Category Products Page UI	7 days	11/10/20 8:00 AM	11/18/20 5:00 PM		
100		Product Details Page UI	7 days	11/10/20 8:00 AM	11/18/20 5:00 PM		
101		Deployment	3 days	11/17/20 5:00 PM	11/20/20 5:00 PM		
102		Setup CI/CD	2 days	11/20/20 8:00 AM	11/23/20 5:00 PM		
103		Best Sellers, On Sale, New Releases, etc. UI	3 days	11/22/20 8:00 AM	11/25/20 5:00 PM		
104		Wishlist	6 days	11/17/20 5:00 PM	11/25/20 5:00 PM		
105		**⊟Android**	**15 days?**	**11/3/20 8:00 AM**	**11/23/20 5:00 PM**		**Emre Girgin;Çagri Çiftçi;Berkay Alkan;Alperen Bag**
106		Android core topics research	7 days?	11/3/20 8:00 AM	11/11/20 5:00 PM		
107		Project Setup	3 days?	11/10/20 8:00 AM	11/12/20 5:00 PM		
108		Homepage UI-Logic	10 days?	11/10/20 8:00 AM	11/23/20 5:00 PM		
109		Categories Page UI-Logic	10 days?	11/10/20 8:00 AM	11/23/20 5:00 PM		
110		Wishlist Page UI-Logic	10 days?	11/10/20 8:00 AM	11/23/20 5:00 PM		
111		Product List Page UI-Logic	10 days?	11/10/20 8:00 AM	11/23/20 5:00 PM		
112		Product Detail Page UI-Logic	10 days?	11/10/20 8:00 AM	11/23/20 5:00 PM		

113		Log-in/Sign-up UI-Logic	10 days?	11/10/20 8:00 AM	11/23/20 5:00 PM		
114		Final review and test	1 day?	11/23/20 8:00 AM	11/23/20 5:00 PM		
115		Milestone 1	0 days?	11/24/20 8:00 AM	11/24/20 8:00 AM		
116		**⊟Backend**	**17 days**	**11/24/20 8:00 AM**	**12/16/20 5:00 PM**		**Koray Cetin;Mehmet Erdinç Oguz;Olcayto Türker;Veli Can Unal**
117		Profile Pages of Users	5 days	11/24/20 8:00 AM	11/30/20 5:00 PM		
118		Orders and Payment	4 days	11/30/20 5:00 PM	12/4/20 5:00 PM		
119		Filtering and Sorting Mechanism	4 days	12/4/20 5:00 PM	12/10/20 5:00 PM		
120		Shopping Cart	4 days	12/10/20 5:00 PM	12/16/20 5:00 PM		
121		**⊟Frontend**	**17 days**	**11/24/20 8:00 AM**	**12/16/20 5:00 PM**		**Berke Can Gurer;Burak Cuhadar;Eylul Yalcinkaya;Meric Ungor**
122		Connect hardcoded functionality to backend	7 days	11/24/20 8:00 AM	12/2/20 5:00 PM		
123		Shopping Cart	5 days	12/3/20 8:00 AM	12/9/20 5:00 PM		
124		Profile Information Page	5 days	12/3/20 8:00 AM	12/9/20 5:00 PM		
125		Orders Page	5 days	12/10/20 8:00 AM	12/16/20 5:00 PM		
126		Filter	5 days	12/10/20 8:00 AM	12/16/20 5:00 PM		
127		**⊟Android**	**17 days?**	**11/24/20 8:00 AM**	**12/16/20 5:00 PM**		**Emre Girgin;Çagri Çiftçi;Berkay Alkan;Alperen Bag**
128		Connect hardcoded funtionality to backend	7 days?	11/24/20 8:00 AM	12/2/20 5:00 PM		
129		Shopping Cart Page UI-Logic	8 days?	11/28/20 8:00 AM	12/9/20 5:00 PM		
130		Profile Page UI-Logic	8 days?	11/28/20 8:00 AM	12/9/20 5:00 PM		
131		Vendor Page UI	5 days?	11/28/20 8:00 AM	12/4/20 5:00 PM		
132		Payment Page UI	5 days?	11/28/20 8:00 AM	12/4/20 5:00 PM		
133		Orders Page UI-Logic	8 days?	11/28/20 8:00 AM	12/9/20 5:00 PM		
134		Filter and Sort Logic	5 days?	12/9/20 8:00 AM	12/15/20 5:00 PM		
135		Final review and test	1 day?	12/16/20 8:00 AM	12/16/20 5:00 PM		
136		Milestone 2	0 days	12/17/20 8:00 AM	12/17/20 8:00 AM		
137		**⊟Backend**	**8 days**	**12/17/20 8:00 AM**	**12/28/20 5:00 PM**		**Veli Can Unal;Olcayto Türker**
138		Update Orders	3 days	12/17/20 8:00 AM	12/21/20 5:00 PM		
139		Most Purchased, Most Liked, Discounted Prod...	4 days	12/21/20 5:00 PM	12/25/20 5:00 PM		
140		Complaints	2 days	12/23/20 5:00 PM	12/25/20 5:00 PM		
141		Update Profile for Users	1 day	12/25/20 5:00 PM	12/28/20 5:00 PM		
142		**⊟Frontend**	**8 days**	**12/17/20 8:00 AM**	**12/28/20 5:00 PM**		**Berke Can Gurer;Burak Cuhadar;Eylul Yalcinkaya;Meric Ungor**
143		Payment Flow	4 days	12/17/20 8:00 AM	12/22/20 5:00 PM		
144		Complaint about past order	2 days	12/23/20 8:00 AM	12/24/20 5:00 PM		
145		Admin Panel Login	4 days	12/23/20 8:00 AM	12/28/20 5:00 PM		
146		Vendor Page	8 days	12/17/20 8:00 AM	12/28/20 5:00 PM		
147		**⊟Android**	**30 days?**	**11/17/20 8:00 AM**	**12/28/20 5:00 PM**		**Emre Girgin;Çagri Çiftçi;Berkay Alkan;Alperen Bag**
148		Old milestones review	3 days?	11/17/20 8:00 AM	11/19/20 5:00 PM		
149		Payment Logic	5 days?	11/19/20 8:00 AM	11/25/20 5:00 PM		
150		Vendor Logic	6 days?	12/19/20 8:00 AM	12/28/20 5:00 PM		
151		Complaint Logic	5 days?	11/19/20 8:00 AM	11/25/20 5:00 PM		
152		Final review and test	1 day?	12/28/20 8:00 AM	12/28/20 5:00 PM		
153		Milestone 3	0 days	12/29/20 5:00 PM	12/29/20 5:00 PM		
154		**⊟Backend**	**18 days**	**12/25/20 8:00 AM**	**1/19/21 5:00 PM**		**Koray Cetin;Olcayto Türker;Veli Can Unal;Mehmet Erdinç Oguz**
155		Admin Login	2 days	12/25/20 8:00 AM	12/28/20 5:00 PM		
156		Admin Search	2 days	12/31/20 5:00 PM	1/4/21 5:00 PM		
157		Messaging Framework	4 days	1/4/21 5:00 PM	1/8/21 5:00 PM		
158		Notification Infrastructure	7 days	1/8/21 5:00 PM	1/19/21 5:00 PM		



159	❗	⊟Frontend	58 days	10/28/20 8:00 AM	1/15/21 5:00 PM		Berke Can Gurer;Burak Cuhadar;Eylul Yalcinkaya;Meric Ungor
160		Admin Remove Comment Page	3 days	10/28/20 8:00 AM	10/30/20 5:00 PM		
161	📅	Admin Ban User Page	3 days	1/2/21 8:00 AM	1/6/21 5:00 PM		
162	📅	Bug fixes	7 days	1/7/21 8:00 AM	1/15/21 5:00 PM		
163	📅	Additional features	7 days	1/7/21 8:00 AM	1/15/21 5:00 PM		
164	📅 ❗	⊟Android	13 days?	12/30/20 8:00 AM	1/15/21 5:00 PM		Emre Girgin;Çagri Çiftçi;Berkay Alkan;Alperen Bag
165	📅	Additional features	7 days?	12/30/20 8:00 AM	1/7/21 5:00 PM		
166	📅	Bug fixes	7 days?	1/7/21 8:00 AM	1/15/21 5:00 PM		
167	📅	Final Milestone	0 days	1/15/21 8:00 AM	1/15/21 8:00 AM		

User Scenarios

Android Scenario

Persona: Ahmet is a 20-year-old student studying at Bogazici University. Recently, because of the pandemic, he could not go shopping. He wants to buy himself perfume and a pyjama.

Preconditions: Ahmet has an android phone and internet connection.

Goals: His goal is to buy nice perfume and a nice pyjama.

Actions:

1. Ahmet downloads the application from Google Play, then opens it.
2. He directly clicks on the profile tab, because he knows that he should sign up before buying something, since he is very familiar with mobile applications.
3. He enters his e-mail and a password, then clicks the sign-up button and sees the approval notification.
4. He clicks the log-in button and is directed to his profile page.
5. Then he returns to the homepage to check if he can find some discounted products, however cannot find anything interesting.
6. He clicks on the categories tab, then he clicks on the perfume category.
7. He takes a glance at the perfume products, however he finds them expensive.
8. He clicks on the back button and returns to the categories list page.
9. He clicks on the arrow that is on the “Men” category to go into the subcategories of the “Men” category.
10. He clicks on the “Pyjama” category and sees the pyjamas.
11. He finds two nice pyjamas and adds these to his wishlist.

12. He also finds another nice pyjama. He enters the product detail page for this pyjama and adds that one to his wishlist as well.
13. Then he clicks on the wishlist tab to see his selections.
14. One of the pyjamas in the list seems expensive, so he removes that one.
15. He cannot decide which one of the other two pyjamas to buy, therefore he decides to think about it later and closes the app.

Frontend Scenario

Persona: Grandfather Battal is 70 years old. He has 3 sons and 8 grandchildren. Recently one of his sons bought him a laptop so his children and grandchildren can zoom with him. Grandfather Battal loved using it and started trying to solve all of his problems with his laptop. His youngest baby grandchild's birthday is coming up so he wonders if he can find a present online.

Preconditions: Battal has a laptop with an internet connection. A web browser is installed in his laptop.

Goals:

- Battal wants to add products he likes to his wishlist.
- He wants to see the prices of the products in his wishlist.
- He wants to remove the products from his wishlist if they are too expensive.

Actions:

1. Battal starts from the homepage.
2. He checks the categories but doesn't know which one to select
3. He clicks a baby product from best seller products.
4. He tries to add it to his wishlist.
5. He is redirected to sign in page.
6. He signs up to the platform.
7. He signs in.
8. He goes back to the product page.
9. He adds the product to his wishlist.
10. He goes to his wishlist page.
11. He goes to the homepage.
12. He finds another product from recommended products.
13. He adds it to wishlist.
14. He goes to his wishlist page.
15. He removes the more expensive product.

Acceptance Criteria:

- 1.2.3.1 System shall have categories for discovering new products.
- 1.1.1.1.4 Customers shall have a Wishlist and Collection.
- 1.1.3.1 Customer and vendor users shall provide an email, an username and a password to sign up.
- 1.2.7.1 The platform shall recommend certain products to the users based on their interactions on the platform.

Code Structure

Frontend: Frontend team's main branch is *web-dev*. This branch is where we merge all the personal branches and solve merge conflicts. Everyone has their own branch: *web-eylul, web-meric, web-burak*. These personal branches are used for developing the assigned features. We use Github's issues system to assign and manage tasks. We try to assign independent features to different people so as to prevent merge conflicts and bugs. Once a feature is done, a pull request is opened from the personal branch to web-dev. The pull request is reviewed by at least one other person from the team. If the code is well written and works fine, it is then merged.

Code structure is as follows:

- **web**
    - **public**: public images
    - **src**
        - **components**: components that are used in pages
        - **images**: icons, images that need to be in src folder
        - **pages**: webpages
        - **redux**: redux related - actions and reducers
        - **services**: API calls
        - **util**: utility functions

Backend: Backend team creates branches according to the issue name. For instance if our issue is "#153 endpoint implementation" our branch name will be the same. Our purpose is reaching the mini document why we need this change. If there is a problem with that code , anyone can change it from this branch by reading the definition of issue. After the development process, we are creating a pull request. At least one person checks the developer's code. If it is OK, s/he merges with the master branch.

Code structure:

- **product_service**
  - **db**: This part creates db
  - **Mock-product-data:** We have mock data for sending frontend side
  - **Models:** This part defines table columns for every area that we need to work such as customer, product, vendor etc.
  - **Routes:** This part send data to frontend side by using endpoints
  - **Tests:** This file includes tests that we wrote.
  - **Views:** This file returns data to the routes file, it basically gets data from the database.

Android: Android team's main branch is *android-dev*. This branch is where we merge all the personal branches and solve merge conflicts. Everyone has their own branches such as *android-dev-login, android-dev-homepage, android-dev-products etc*. These personal branches are used for developing the assigned features. We use Github's issues system to assign and manage tasks. We try to assign independent features to different people so as to prevent merge conflicts and bugs. Once a feature is done, a pull request is opened from the personal branch to *android-dev*. The pull requests are reviewed by all team members in a zoom meeting. If the code is well written and works fine, it is then merged.

Code structure is as follows:

- **app -> src -> main**
  - **res**: It is a conventional android folder to store xmls, images, drawables, colors, styles etc.
  - **buyo**
    - **api**: Endpoints Interface and Service wrapper (Api calls 1)
    - **base**: Base classes and fragment transactions helpers
    - **datamanager**
      - **network** api calls helper classes (Retrofit)
      - **repository**: (Api calls 2)
      - **shared_pref**: Storing primitive data in local
    - **dependencyinjection**: Some module classes about di

    - **ui**: Fragments(Pages in the app)
    - **util**: Some utility classes and functions such as base dialog
      viewmodel: Storing data coming from api
    - **vo**: Data models according to api
    - **widget**: Custom views such as navigation bar

- **Buyo -> app -> src -> test/AndroidTest:** All tests will be written in this folder
- **Buyo -> Gradle:** Gradle files such as version.gradle, build.gradle

Evaluation of the Tools and Managing the Project

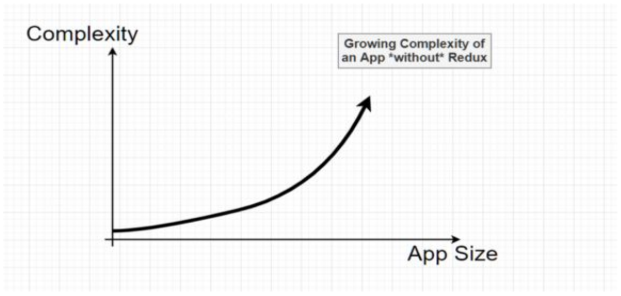
Evaluation of the Tools Used by the General Team:

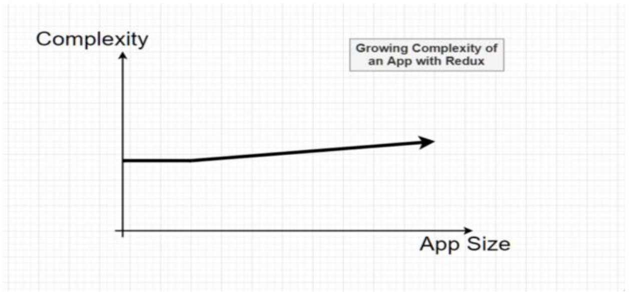
- Our communication channels are Slack, Whatsapp and Zoom. We are holding general weekly meetings on Zoom as well as weekly team meetings to go over what we did the previous week and plan the week ahead. Also we hold additional meetings for helping each other solve some problems and whenever necessary like when we are close to a milestone and need to figure some stuff out. Most of the other communication goes over Slack where we have a lot of dedicated channels. Whatsapp is mostly for critical messages to each other, like organizing a meeting for the same day.
- For planning, we are using ProjectLibre which is an open-source project management tool. It can be difficult and painful to export it to PDF properly and doesn’t have much support, but it is still the best free alternative to Microsoft Project. The tool is easy to use and get started with and has all of the functionality that we need for now. So we don’t have any regrets with our decision to use ProjectLibre.

Evaluation of the Tools Used by the Frontend Team:

- For the frontend, we are using ReactJs, which is a Javascript Library for building web applications. Since it is written with a more app-like approach rather than our plain html, css, javascript, a lot of functionality on your web application is implemented much easier if your aim is not to create a mostly static website. Even though it increased our research and learning times at the beginning of our project plan a bit, we are already glad we went in this direction and we will only keep eating the fruits of this choice. But we can still use React more effectively. There are a couple of best practises to write React like using functions or classes for components. We talked about the obvious parts but since we are learning as well we couldn't figure out everything and there are systematic differences in some parts of the code which increases the complexity of the overall project. But we solve these differences as we come across them and we believe it is not a big problem.

- We are also using Redux in the frontend, which is a predictable state container for javascript applications. Normally in React, any information on a component is accessible only from its parent or child components. Redux makes it so that any information on any component can be written onto the Redux state and then accessed from any other component. This is extremely helpful in complex web applications like ours where we need to access any information from anywhere like accessing which user is signed in from the cart page so that we can send an endpoint accordingly. Directing this information from the sign in page through only child and parent components would be a nightmare and really increase the complexity of our code. The only downside on this decision was that we didn't realize Redux states





vanished when the user refreshed the page. After some research we decided to use redux-persist which is an addition to redux that saves the redux state on the local storage of the browser. If we knew this, we would still go with Redux. Nonetheless, this showed us that more research can be helpful when we are making important decisions like which tools to use.

- Another tool we use on the frontend is Bootstrap. Bootstrap is a CSS-framework for developing responsive websites. When it is used properly, it deals with most aspects of responsiveness, which can be excruciating and time-consuming. Also its predefined and coherent classes provide us a medium where the team can generate a coherent design. Even though we design the pages before implementing them, with the absence of a UI/UX designer, conforming on a coherent design can be a real challenge in communication. Bootstrap really simplifies this. So we are extremely happy with this tool and the many hours it has gifted us.

- For collaborating on UI designs as a team, we use Figma. Figma makes collaborating very easy on interface designs. Also it generates CSS codes for our designs and provides the necessary information for web development like margin, padding sizes in pixels, which really fast tracks dealing with CSS.
- For deployment of frontend, we tried to use nginx since it is one of the most common options but we ran into a lot of problems when we tried to integrate it with docker and realized its documentation is not great to help us. So we weren't really happy with that decision and decided to just deploy with the serve package of npm. We couldn't still integrate *serve with docker* functionality either but it seems much simpler and straightforward so we are more optimistic about it for now.
- Besides from these, as the frontend team, we are happy with the management of our frontend project. We can say we effectively divide work among us since we didn't come across a proper merge conflict yet. We have a development branch on github where we merge our work and make sure everything works fine before deploying features in bulk, which prevents messy problems we would deal with on the master branch. We planned the project with a lot of tolerance and we mostly stick to the initial plan which promises a successful and easy-going project for us throughout the semester.

Evaluation of the Tools Used by the Android Team:

- For the android, we used Android Studio which is the official IDE for Android. We chose it due to its flexible build system, real-time analyzer, intuitive visual layout, and provided virtual Android machines where the user can test the application without need for anything else. These features made it easier to develop the project for us.
- We used XML in the UI implementation of Android. It is easier to describe a layout in the XML than in Java directly. We created the elements in the UI as XML, then we specified the constraints of them with the visualizer of the Android Studio where the corresponding XML code is automatically created. For the logic part initially we decided on Java because all team members have experience with it. But we came across some problems when implementing the base of the project. Then we changed it to Kotlin programming language which is developed by Jetbrains. Some members of

the team did not know Kotlin thus, we spared some time to learn the aspects and syntax of it and it was worth it. We completed the implementation of the base of project and logic parts much easier with it. Kotlin checks for NullPointerExceptions everywhere in your code which was a lifesaver for us. To add packages we used Gradle which is an open source build automation tool. We used Retrofit which is a REST client for Android.

- Overall, as the Android team, we are satisfied with the management process of ours. Since some members did not have any experience with Android, initially we learned the concepts that we used before starting the implementation. We used Android-development branch as our main branch in Github and merged other branches that we implement functionalities to it. We defined the works needed to be done and shared them among the team members in the group meetings. We helped each other when someone came across a bug or had difficulty in the implementation phase by sometimes arranging extra meetings. Generally we got the job done before the deadlines we defined.

Evaluation of the Tools Used by Backend Team:

Node JS:
Node js is an environment that executes Javascript code outside of a web browser. It is built on V8. Node JS is a high-performance open-source Javascript engine.

- Why are we using it ?
    - It is easy-to-learn.
    - It is supported by huge community
    - It has a very reliable package system which is NPM(Node Package Manager).
    - It is compatible with almost all types of databases.
    - It is very suitable for web server and backend operations.
    - It is suitable for I/O operations.
- Do we have any problem with Node JS?
    - No, everyone adapted the concept really fast and wrote code by using Node JS.
- How about the other alternatives?
    - All developers at the backend team voted for Node Js. We also discussed using Python. Even if python might have some other advantages, we preferred to use Node JS. The reasons can be seen above. Additionally we had more experience on Node js The backendside .

MongoDB:
MongoDB is a document-oriented NoSQL database.

- Why are we using it?

    - Since it is a Nosql database, MongoDB makes the representation in the form of JSON documents. We decided to use Node JS. All our objects will be JSON. That's why it is very logical to use MongoDB
    - MongoDB can store any kind of file
    - MongoDB can run on different servers.
- Do we have any problem with MongoDB?
    - We don't spend so much effort using it. However we had several issues while installing it. We think it was normal to have some problems at the installation step.
- How about the other alternatives?
    - The other alternative of CmpE451 is PostgreSQL. PostgreSQL is a relational database. It is hard to maintain JSON objects for PostgreSQL by comparison to MongoDB

Mongoose:
Mongoose is an Object Data Modeling (ODM) library. It is used for connection between Node JS and MongoDB. The developers can handle database management by using mongoose.

- Why are we using it?
    - It is really easy to use. We can update our database like adding an element to a list or updating it on our NodeJs server thanks to mongoose.
    - We have experience on it from our internships.
- Did we have any problem with Mongoose?
    - No, we are glad to use it.
- How about the other alternatives?
    - There are lots of object data modeling libraries for NodeJs and MongoDB such as sift,mongoist,mongoJs etc. We preferred to use the most popular library. We can find some sources if we have a problem. Additionally, we have some experience on Mongoose as well.

Docker:
Docker is a tool for making it easier to create, deploying, and running applications by using containers.Docker containers encapsulate everything an application needs to run.

- Why are we using it?
    - The backend team member uses a different kind of operating system. We need to find a common development base. Docker provides this common development base to us
    - In the end, we have to deploy our codes to a server. By using docker configurations, we can deploy our code easily
    - We want to use microservices architecture .Microservices architecture is a system that has independent services from each other. If a service is down, the other ones will continue to work thanks to microservice architecture. In a monolithic system, if there is an error in the code , the system shut down itself. Monolithic systems are really error prone. Docker allows us to apply microservices architecture
    - We can check our logs if there is a problem with the current container.

- Did we have any problem with Docker?
    - We had to turn on hyper virtualization for windows systems . This was the main problem while we were using docker.
- How about the other alternatives?
    - We didn't consider other alternatives. Docker is the most dominant actor for container technology. We can create all types of environment by using its libraries.

AWS:
Amazon Web Services(AWS) is the cloud platform provided by Amazon.

- Why are we using it?
    - AWS is the most reliable and easy-to-use cloud environment according to our research
    - It has a high-functional dashboard. We can create or terminate our servers easily
    - It has lots of cloud computing tools for storage, networking & content delivery, customer engagement etc. In our opinion, we might need them in the future.
    - AWS is the most popular cloud platform. We can find support easily if we have a problem
- Did we have any problem with AWS?
    - At first, we didn't create a proper server for our system, but we solved it immediately.
- How about the other alternatives?
    - The other choice was Google. However, the software engineers community doesn't use Google servers as much as Amazon. It might be difficult to find our way if we have an issue.

Jenkins:
Jenkins is an open-source automation. It is used for Continuous Integration and Continuous delivery (CI/CD). The development and test phases are considered as continuous integration. The developers are integrating their code to the previous code.
Consider delivery is making the recent product available to the customer. The purpose of a development team is delivering the most useful functions for the customer. Applying CI/CD has crucial importance for reaching out the customer.

- Why do we want to use it?
    - Easy to install
    - It is free
    - All development process can be automated
    - It can generate test reports
    - It is compatible with different kind of OS
    - It is compatible with dockerized systems
- How about the other alternatives?
    - Travis CI can be considerable as our second alternative. Jenkins is more customizable than Travis CI. In our opinion we might need this ability in the

future. As a third alternative we may think to use Gitlab. Since Travis has a bigger community than Gitlab and we can find plugins that we want to use easily, we preferred Jenkins.

Assessment of the Presentation

Our presentation was overall good. It was fluent and not boring. Especially presenting the android application on a real phone was a good idea, since it prevented the lagging that comes from the android emulator, making the presentation fluent and realistic. We also could find the chance of asking some questions to the customer and we became sure that we are on the right direction about the project.

Moreover, It was a good experience for us in terms of presentation. What we learnt from this meeting is that we should ask more questions and make the presentation much more interactive with the customer. In addition to this, we focused mostly on the functionalities of our application, therefore we tried to implement the functionalities as much as we could for the first customer meeting. However, for the next meeting, we should beautify our UI and fix the bugs we have.

Lastly, we couldn’t deploy our frontend, therefore we had to present the frontend part on our local machine. We couldn’t set up a CI/CD process for our frontend yet and had encountered difficulties while trying to dockerize the app and deploy it to Amazon EC2 machine. This was our major weakness in the presentation. We will make sure that we solve these issues until the next customer presentation.