Boiler Buy

Design Document

Team Members: Joseph Chen, Christopher Hyman, Trenton Reeves, Chris Hyman, Sanjith Pabbisetty, Michelle Song, Richard Stump

Purpose

While there are endless ways to buy and sell products already, many of them come with their flaws. Facebook marketplace for example, bounds the listable prices for their items, which could result in an inaccurate post when sellers want to sell something such as a used car, which would fall out of the price interval facebook marketplace allows you to list. Another example would be Craigslist, where the UI is much too simple and outdated. Along with this, something that all the buy, sell, trade websites lack is safety, as any user can create an account to post and buy items. We plan to create a buy, sell, trade website that provides a good UI, less restrictions when it comes to selling, and limiting this marketplace to only Purdue students as this can increase the safety of this marketplace.

Project Objectives

- Develop a web application utilizing both a frontend and backend
- Features to post and manage ads for the sale, purchase, and trade of goods/services
- Features to buy, sell, and trade goods/services
- Features to categorize and search for specific items
- Features to message a seller for meet-up locations and bartering of prices
- Database to store user account information and store items
- Improve upon the UI and features of Facebook Marketplace and Craigslist to provide a smoother user experience
- Features to ensure users are Purdue students

Project Requirements

Functional Requirements:

Id:	User Story:
1	As a user, I would like to access the website.
2	As a user, I would like to register for a Boiler Buy account.
3	As a user, I would like to access my Boiler Buy profile.
4	As a user, I would like to verify that I am a Purdue student.
5	As a user, I would like to change my password.
6	As a user, I would like to set a profile picture.
7	As a user, I would like to change my username.
8	As a user, I would like to reset my password if I forget.
9	As a user, I would like to retrieve my username if I forget.
10	As a user, I would like to block other users from sending me messages.
11	As a user, I would like to be able to view the website from my phone.
12	As a user, I would like to be able to upload my Venmo tag so that when I come to an agreement with a buyer/seller, it will automatically send it via private message.
13	As a user, I would like to toggle between light dark modes.
14	As a user, I would like to delete my account.
15	As a buyer, I would like to shop for products.
16	As a buyer, I would like to filter by product type.
17	As a buyer, I would like to filter by a price range.
18	As a buyer, I would like to filter by a seller rating range.
19	As a buyer, I would like to see the seller’s rating.
20	As a buyer, I would like to see the seller’s reviews.
21	As a buyer, I would like to leave a review for the seller.
22	As a buyer, I would like to purchase a product from a seller.
23	As a buyer, I would like to choose for my purchase to be shipped.
24	As a buyer, I would like to choose to meet up for my purchase.
25	As a buyer, I would like to message the seller.
26	As a buyer, I would like to save product ads to view later.
27	As a buyer, I would like to see a history of all the products I have bought.
28	As a buyer, I would like to see a history of the products that I have recently viewed.
29	As a buyer, I would like to be able to set a safe place as a meeting location.

30	As a buyer, I would like to be able to search for items using a search bar.
31	As a seller, I would like to post ads for products to sell.
32	As a seller, I would like to add pictures of the product to my ad.
33	As a seller, I would like to access my shop (what I have for sale).
34	As a seller, I would like to set the product’s price.
35	As a seller, I would like to write a description of the product.
36	As a seller, I would like to provide a shipping option.
37	As a seller, I would like to provide a shipping price.
38	As a seller, I would like to provide a meetup option.
39	As a seller, I would like to respond to the buyer’s message.
40	As a seller, I would like to delete product ads.
41	As a seller, I would like to mark products as sold.
42	As a seller, I would like to group multiple product ads into a “grouped ad”
43	As a seller, I would like to see my selling history.
44	As a seller, I would like to modify ads that I’ve posted.
45	As a seller, I would like to indicate that I have multiple of the same product available.
46	As a seller, I would like to customize my shop front
47	As a seller, I would like to set tags for my product
48	As a buyer, I would like to save/favorite tags
49	As a buyer, I would like to filter by tags
50	As a seller, I would like to set the brand of my product
51	As a buyer, I would like to filter by brand of product

Non-Functional Requirements:

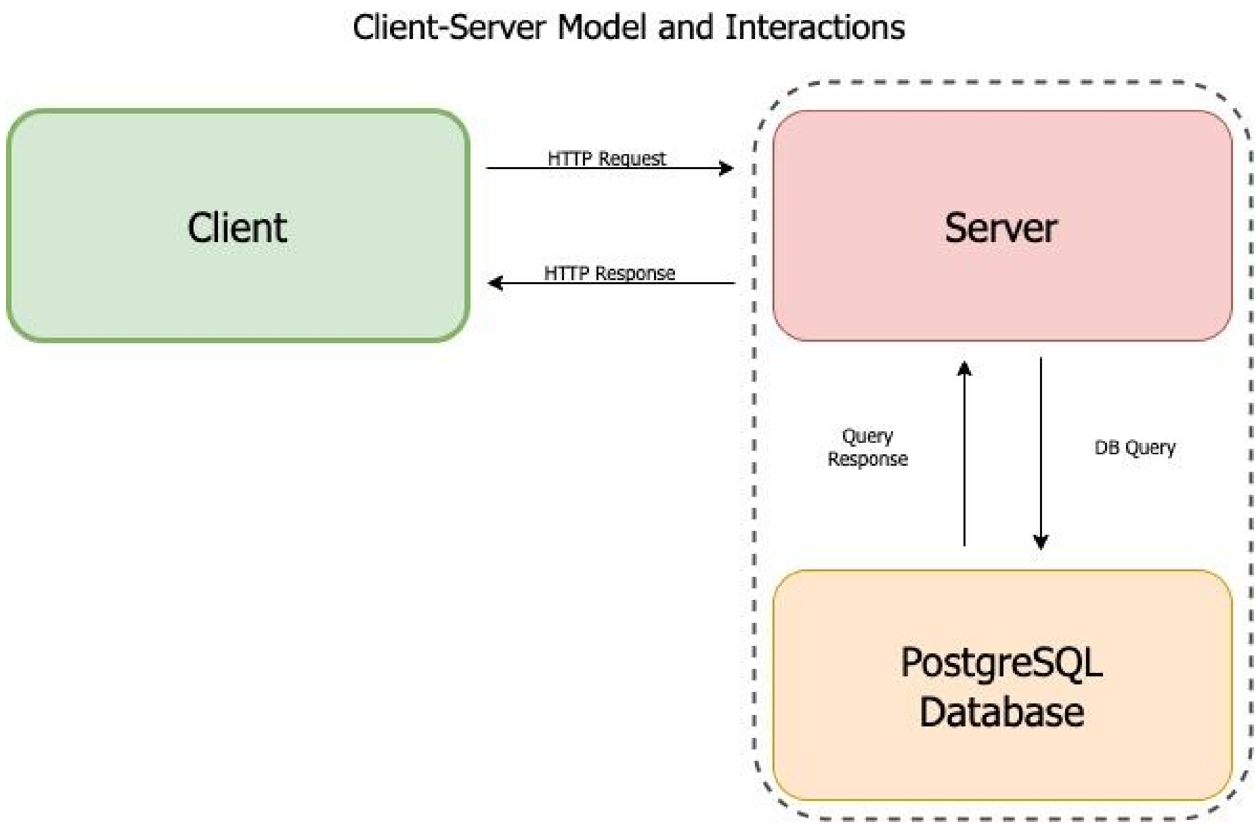
User Experience:

The primary goal of any application is to make a product that is both featureful and easy to use. Our user interface should be smooth, pleasing to look at, and simple. Navigating the web page should be effortless and straightforward. The appearance of the application should be appealing to look at while not being cluttered; moreover, it should be adaptive to both desktop and mobile devices. Additionally, the application should be responsive. The application should never freeze and search results should appear quickly because there is nothing more frustrating as a user than waiting a long time for pages and/or search results to appear.

Security:
Any application involving personal information such as names, numbers, emails, and locations must ensure security. Boiler Buy will achieve this through the encryption of sensitive information such as passwords and emails within our database. Additionally, we will take precautions to test for and fix any bugs which could be exploited.

Scalability:
For projects that aim to survive long-term, scalability can be very important to this goal. We have chosen to follow a model, view, controller architecture to facilitate this. This allows us to update one of the three components with minimal changes to the others. Apart from this architecture decision, we will ensure that the Boiler Buy codebase and design scales easily for future maintenance and additions.

Design Outline:



We have an overall pretty simple design. When the user interacts with the UI, the front end will send a HTTP request to the backend. This HTTP request can vary depending on what the user wants, such as a GET request when a user wants to see someone's profile, or a POST request when the user wants to sell something. Once the backend receives this HTTP request, it sends a request to the database to query whatever is needed. The database responds with the query response, in which the response is formatted and returned to the frontend as JSON to be parsed and displayed.

Components:

Component:	Responsibilities:
Frontend:	● Sends HTTP POST requests to backend ● Receives HTTP POST responses from backend as JSON ● Displays the website
Backend:	● Receives HTTP POST requests from frontend ● Sends requests to database ● Receives responses from database ● Forwards the database's responses to the frontend ● Hosts the server
Database::	● Stores account information ● Stores shop information

Design Issues:

Functional Issues:

1. What information do we need for signing up?
   - Option 1: Username and password
   - **Option 2: Username, password, and Purdue email**
   - Option 3: Username, password, Purdue email, and phone number

Justification: When creating an account, a username and password are obviously needed to separate each unique user from others. Along with this, an email is usually not necessary for a lot of sign ups, but we will require a Purdue email in order to guarantee that the user is a Purdue student. This will allow the user to receive a verification email to their Purdue email, and also allows for username and password recovery. A phone number could be another good way for recovering lost information, however this costs the user when receiving a text message which is not very beneficial when the user can already recover information using their Purdue email.

2. Can the seller set any price they want?
- Option 1: No
- **Option 2: Yes**

Justification: By setting a price cap ensures buyers have more safety when dealing with scams, but this brings another issue where the price cap is set too low for the listed product. This would require sellers to list the correct price in the product description. To solve this issue, we allow the buyer to set the price. Since the Purdue community is known to be safe, there won't be a need to deal with an influx of scams.

3. How will sellers and buyers communicate?
- Option 1: Voice Call
- **Option 2: Messaging**
- Option 3: Emails

Justification: Messaging is the most optimal way to communicate since it allows for users to respond whenever they have the time. Voice calling requires both users to be free for a certain period of time, which could result in lots of conflicts when scheduling time. Along with this, communicating with email is much too slow and inefficient.

4. How will our filter categories work?
- Option 1: Dropdown list
- **Option 2: Checkboxes**

Justification: Checkboxes are a lot more convenient when it comes to filtering systems for the user. They see what filters they currently have checked. Along with this, they can easily check and uncheck filters they want rather than having a dropdown list that they have to click on every time to find the filters they want.

5. How will users verify their account?
- Option 1: Regex matching
- **Option 2: Purdue Database**

Justification: We would like to only provide our services to users with a valid purdue account. Simply verifying if the email ends with "@purdue.edu" would be insufficient as this could include inactive accounts. For an extra layer of security, we will verify users using the Purdue student database to make sure a user has a valid Purdue account.

6. Can the seller create custom tags for their ads?
- Option 1: Yes
- **Option 2: No**

Choice: Option 2
Justification: Creating custom tags may be a nice feature, however it will be hard for us to create a system that filters ads by tags if this were the case. There would be too many possibilities on

our filter list and would create an extremely long filter list for the user which would be very hard for them to use. By only allowing users to choose from premade tags, this makes our filter system much cleaner and easier for the user to use.

Non-Functional Issues

1. What frontend will Boiler Buy use?

- **Option 1: Angular**
- Option 2: React

Justification: We are choosing to use Angular for this project because Angular uses TypeScript and it supports dependency injection. Both options would allow for component based development, but using Angular would allow the project to grow much larger than if we were to use React, which lets the project have more scalability.

2. What database should Boiler Buy use?

- Option 1: MySQL
- Option 2: NoSQL
- **Option 3: postgreSQL**

Justification: The objects to be stored are relatively simple. Hence, a relational database would be sufficient for the purposes of the project. However, postgreSQL offers much faster reads and writes and supports additional types. PostgreSQL offers the ability to scale up much better than mySQL and NoSQL as well.

3. What language will the backend be?

- Option 1: Java
- **Option 2: Python**

Justification: Python offers much more dynamic syntax than Java. Python generally is easier to operate than Java. Alongside these benefits, the team has prior experience with using Python as a backend through its various frameworks like Flask and Django.

4. What testing framework will be used?

- **Option 1: PyTest**
- Option 2: unittest

Justification: PyTest is a lot simpler to use than unittest since PyTest has a lot more built in features, which allows for less code written for the test class. All we have to do is to define the test functions and assert the conditions inside it while for unittest there are a lot more steps which complicates things.

5. What backend framework will we use?

- **Option 1: Django**
- Option 2: Flask

Justification: Since Boiler Buy is a web app, we choose to go with Django which is a full-stack web framework. Django is versatile as it can be used with content in any format. In addition, Django has built-in security features and is scalable.

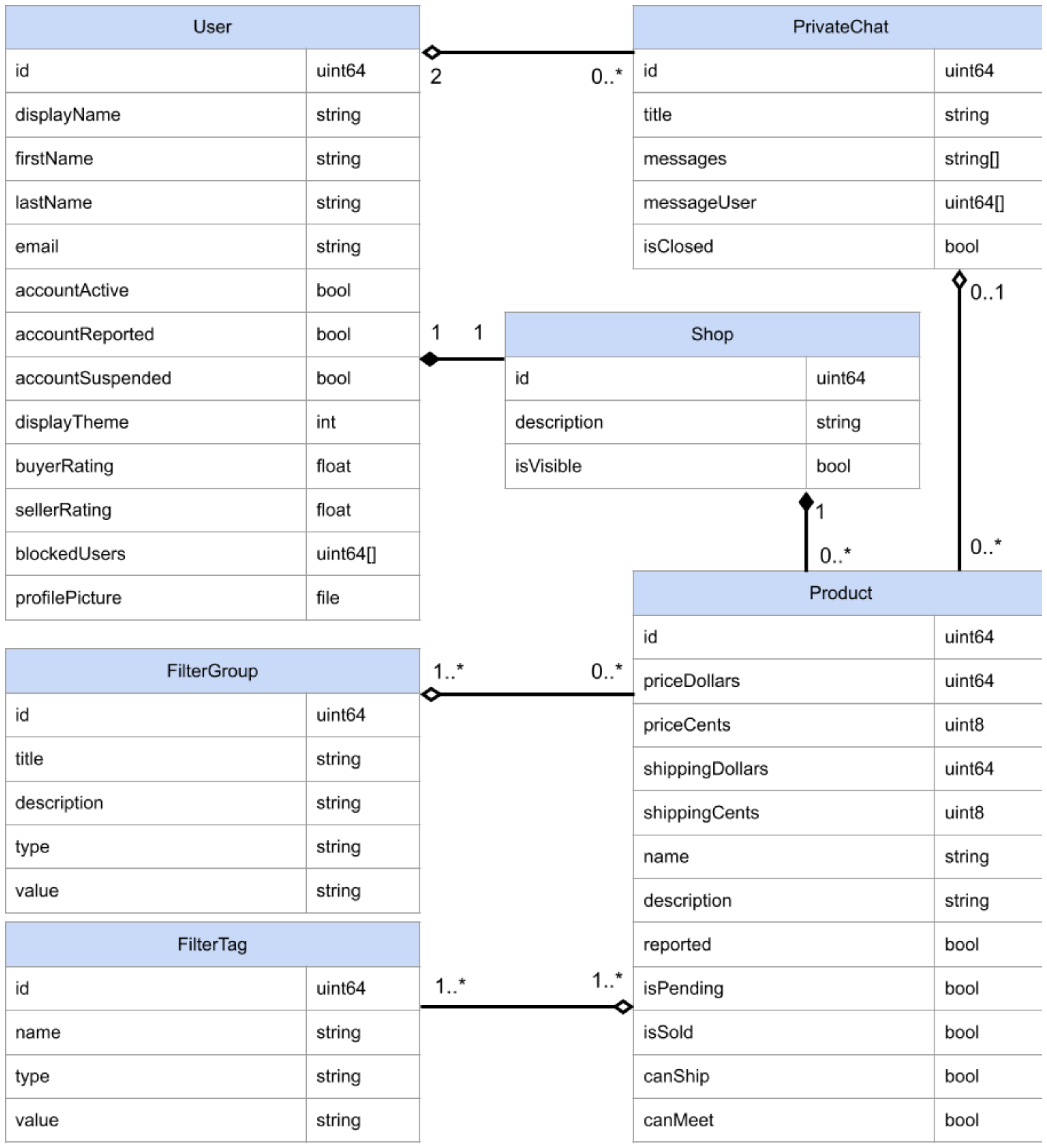
6. What architecture should we use?

- **Option 1: Client-Server**
- Option 2: Client-Queue-Client
- Option 3: Peer-to-Peer

Justification: We choose to use a client-server architecture because we need a separation between the client and server. The client will send a request to the server for information. For example when searching for certain products the server will return the output of the query from the database to the client. In addition, using client-server architecture helps applications improve performance in scalability.

Design Details:

Class Diagram:



User	
id	uint64
displayName	string
firstName	string
lastName	string
email	string
accountActive	bool
accountReported	bool
accountSuspended	bool
displayTheme	int
buyerRating	float
sellerRating	float
blockedUsers	uint64[]
profilePicture	file

PrivateChat	
id	uint64
title	string
messages	string[]
messageUser	uint64[]
isClosed	bool

Shop	
id	uint64
description	string
isVisible	bool

Product	
id	uint64
priceDollars	uint64
priceCents	uint8
shippingDollars	uint64
shippingCents	uint8
name	string
description	string
reported	bool
isPending	bool
isSold	bool
canShip	bool
canMeet	bool

FilterGroup	
id	uint64
title	string
description	string
type	string
value	string

FilterTag	
id	uint64
name	string
type	string
value	string

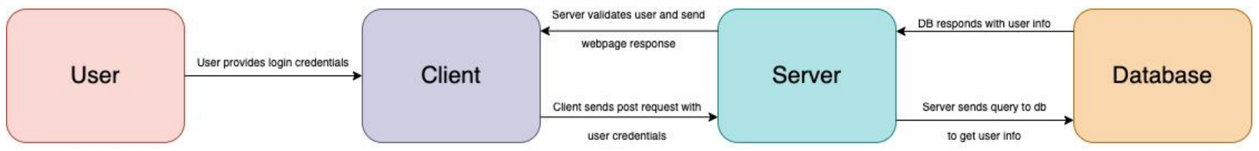
The User class represents a single user registered for Boiler Buy. This class contains information about each user that isn’t applicable directly to products or chats. The Shop class contains all the information needed for a single user's storefront of all the items they are currently selling. The Product class stores all the information about a single item for sale, and the PrivateChat class contains all the necessary info for messaging between two users. We have decided that a PrivateChat can only be associated with two Users, as there would be little to no reason for 2 or more users to be involved in buying or selling an item.

Each user can only have one shop, and each shop can have any number of items. Some users may not sell items, so it makes sense that a shop does not require any items to be in it. We have chosen to associate exactly one shop with each user, regardless of whether they sell items or not to simplify implementation. The shop class can be used to manage items, so if no shop class exists for a user, extra logic would be needed in the User class for users to add items. If a user does not have any items for sale, their shop can be hidden but will always be there if the user decides to sell something.

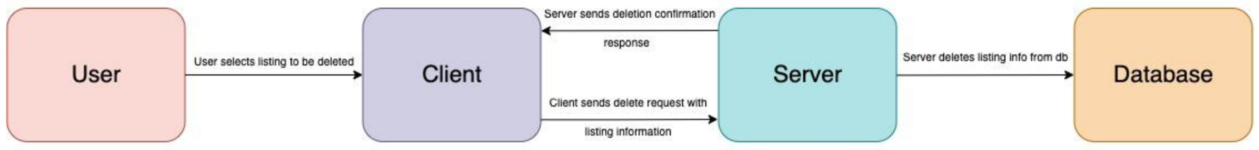
Alone, this would suffice for our app's intentions; however, searching may require sorting through most or all of the items from all sellers to provide all listings. To solve this, the FilterGroup class will associate a trait, such as the make or model of an object, to a group of products. When products are added to the marketplace they are added to the various filters that exist, and when a user searches with these filters, little work needs to be done due to the item already being filtered.

Sequence Diagrams:

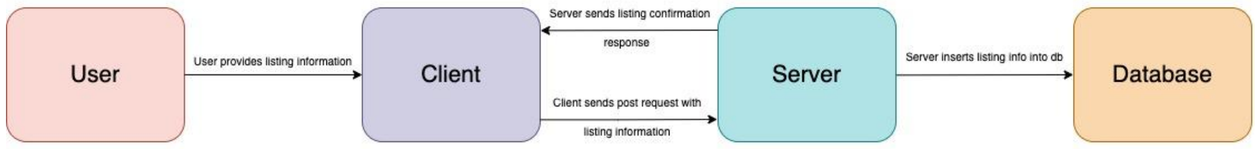
Login Sequence


User
User provides login credentials
Client
Server validates user and send webpage response
Client sends post request with user credentials
Server
DB responds with user info
Server sends query to db to get user info
Database

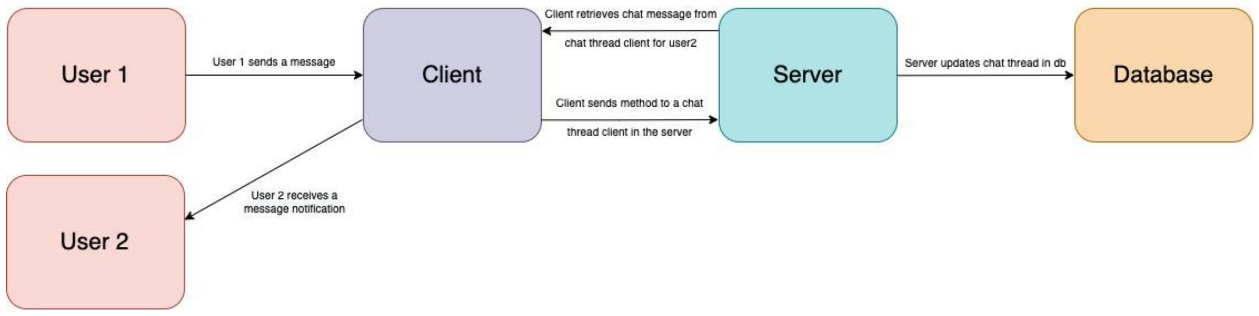
Delete Listing Sequence


User
User selects listing to be deleted
Client
Server sends deletion confirmation response
Client sends delete request with listing information
Server
Server deletes listing info from db
Database

Post Listing Sequence

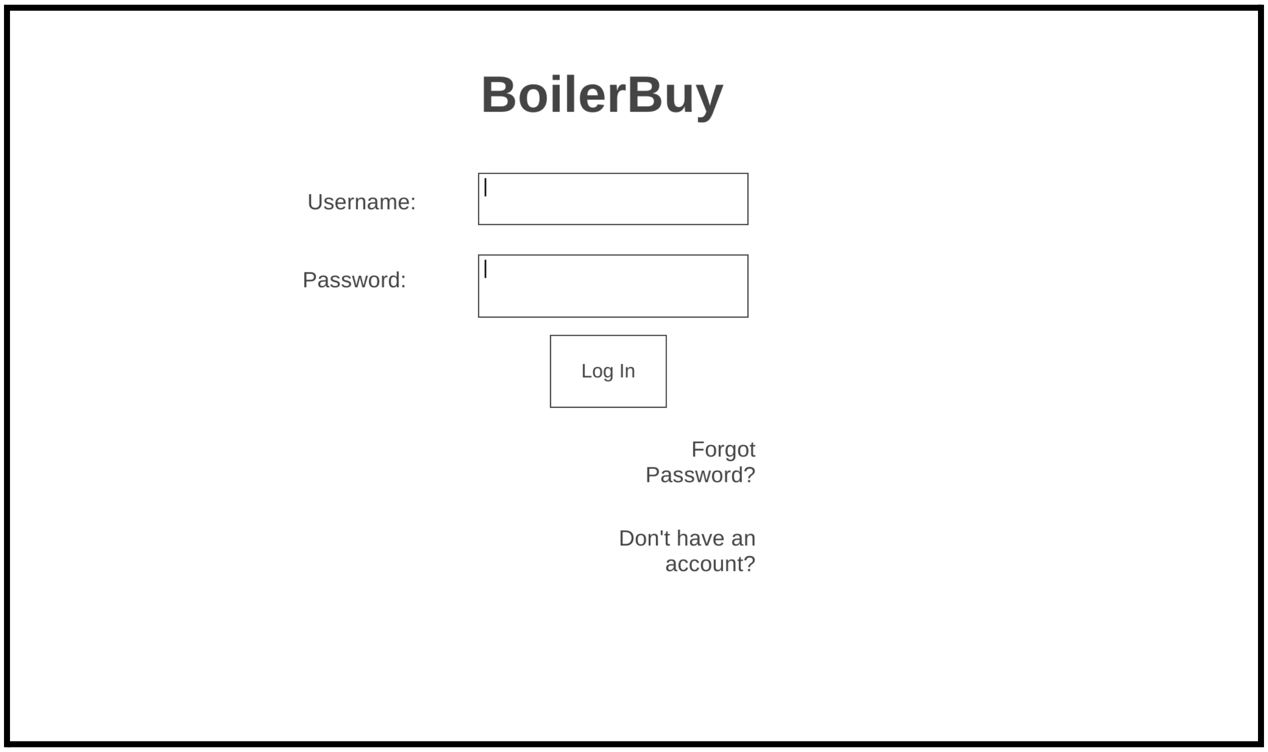

User
User provides listing information
Client
Server sends listing confirmation response
Client sends post request with listing information
Server
Server inserts listing info into db
Database

Chat Sequence

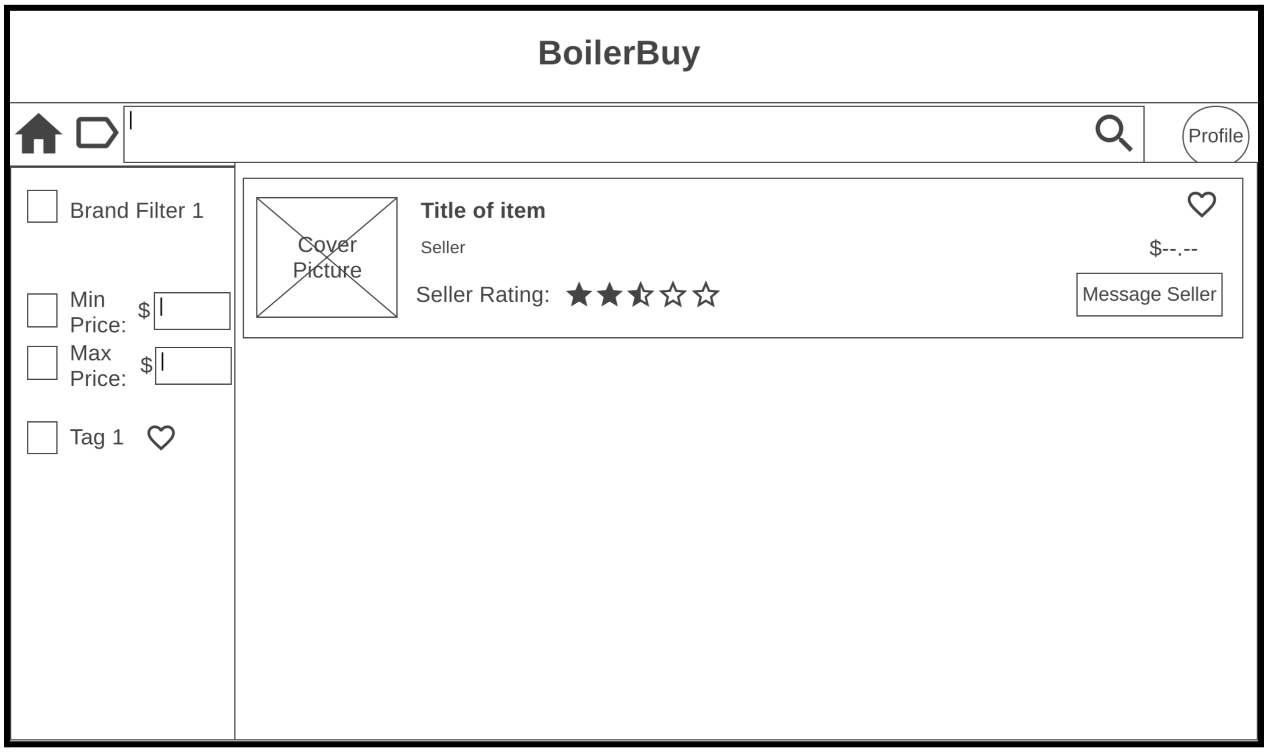

User 1
User 1 sends a message
Client
Client retrieves chat message from chat thread client for user2
Client sends method to a chat thread client in the server
Server
Server updates chat thread in db
Database
User 2 receives a message notification
User 2

UI Mockups:

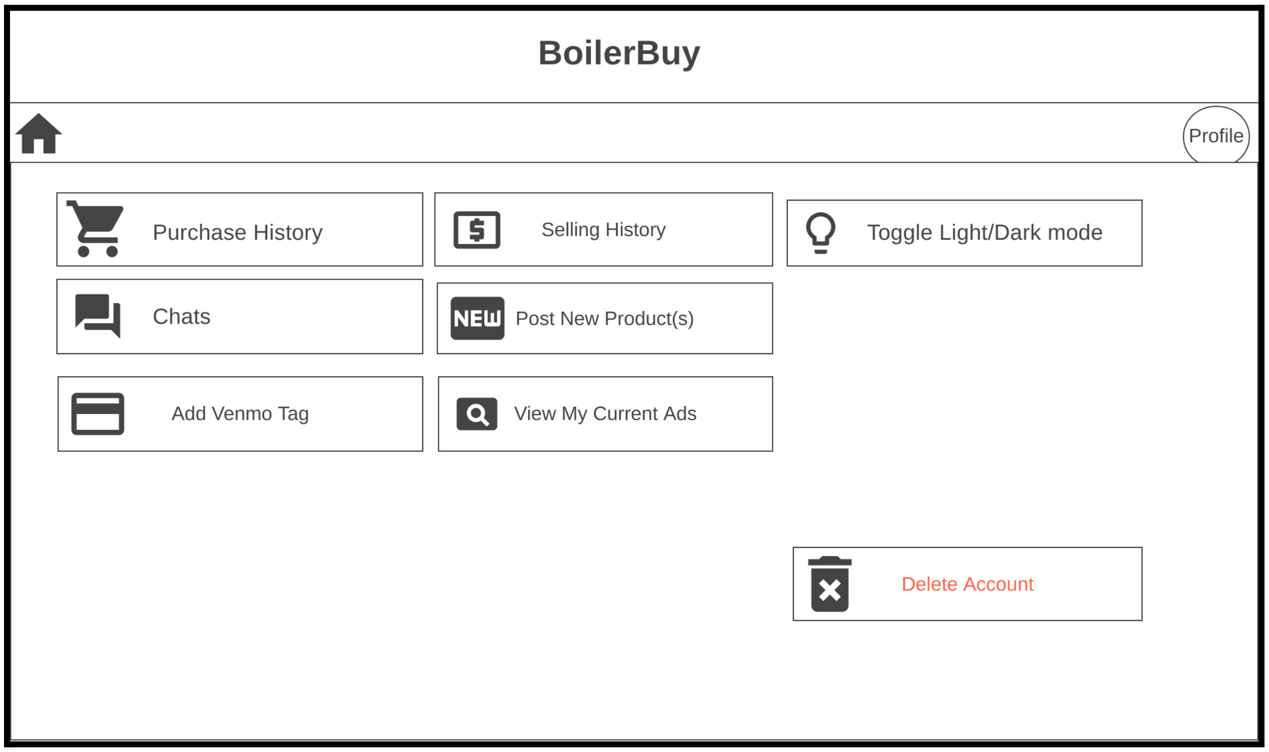
Login/Register Page:



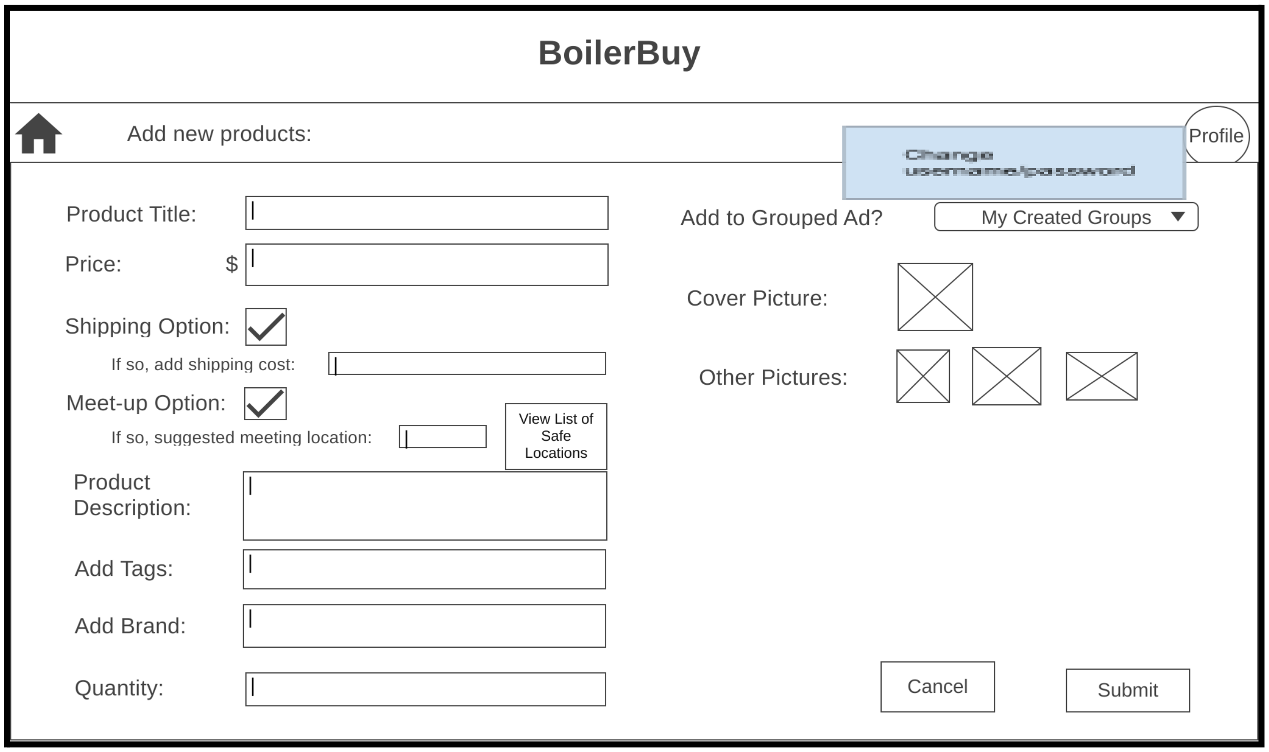
Product Search Page:



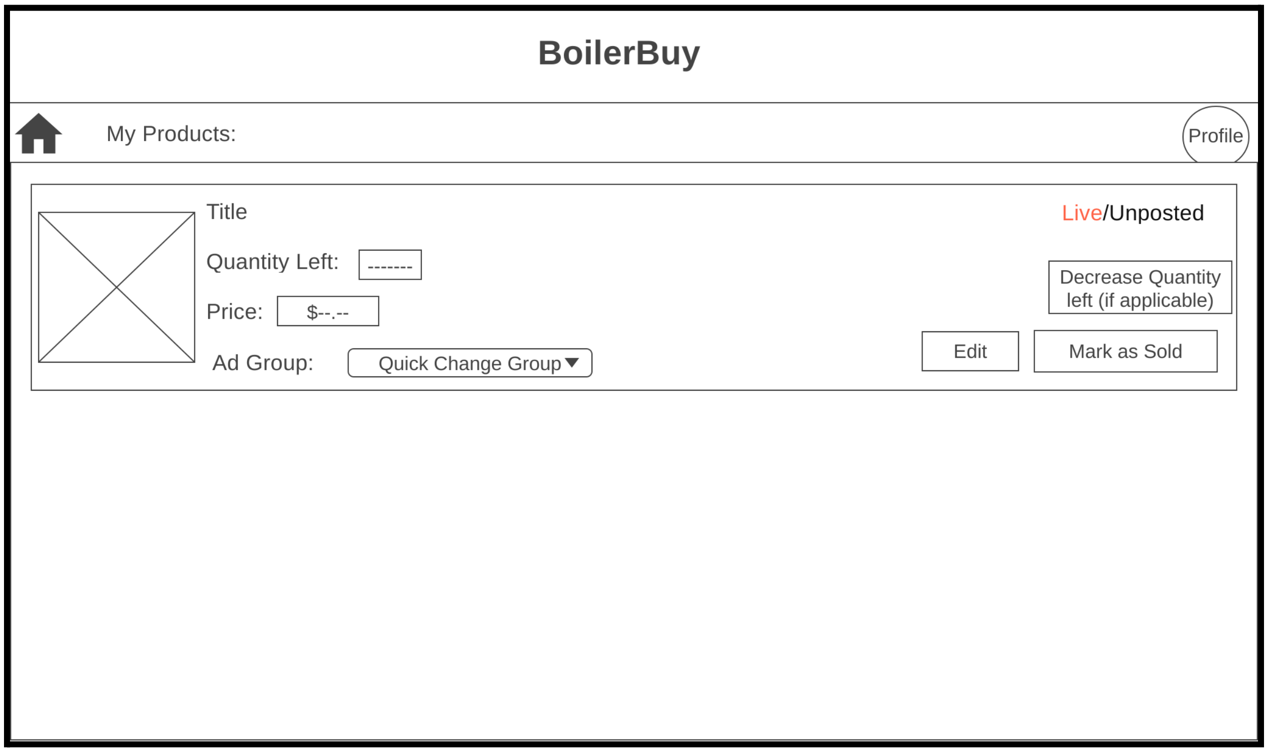
Profile/Settings Page:



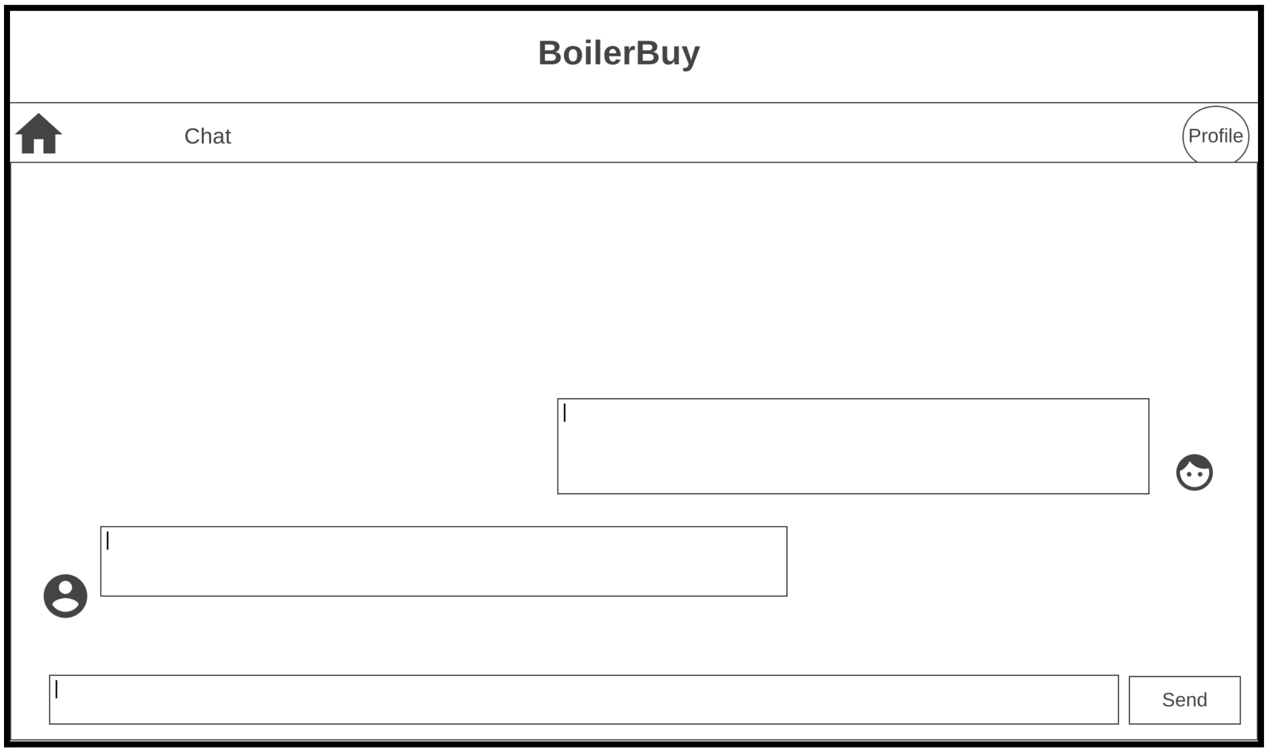
Add Product Page:



My Products Page:



Private Chat Page:


BoilerBuy
Chat
Profile
Send