

BoilerTrack



CS 30700 Design Document

Team 19

- Anitej Waghlay
- Daivik Ghosh
- Daniel Benes-Magana
- Mihika Sharma
- Shlok Bairagi
- Shreya Laxmi Nagendra

INDEX

Purpose	3
Functional Requirements	4
Non-Functional Requirements	7
Design Outline	9
High Level Overview	9
Design Issues	11
Functional Issues	11
Non Functional Issues	13
Design Details	15
Class Design	15
Descriptions of Classes and Interaction between Classes	16
Sequence of Events Overview	19
Sequence diagrams	20
Navigation Flow Map	26
UI Mockups	28

Purpose:

Currently, Purdue University's lost and found system is fragmented, requiring students to physically visit different help desks in hopes of locating their lost belongings. This inefficient process leads to delays and frustration for those attempting to recover their items. Our proposed web app will serve as a centralized platform where staff can log found items and students can quickly search for their lost possessions. By streamlining this process, we aim to significantly reduce the time and effort spent on tracking down lost items, offering a more effective solution than the existing physical approach.

In addition to benefiting students, our web app will also improve efficiency for Purdue's staff by consolidating all lost and found information in one place. Current platforms, such as ileftmystuff.com, cater only to specific groups (like hotel guests), and Purdue's central lost and found lacks a comprehensive online listing. This often results in unclaimed items being sent to the surplus store after a short period. Our app not only offers a searchable database for students but also ensures proper tracking of claimed items, reducing the risk of misplacement or theft and providing a mechanism for resolving disputes.

Functional Requirements:

Student Side -

1. As a user, I want to have the option to create an account with my email.
2. As a user, I want my details (preferred name and pronouns) to be assigned to my account.
3. As a user, I want to have a reset password option.
4. As a user, I want to be able to deactivate my account.
5. As a user, I want to view a list of all lost items on the platform.
6. As a user, I want to search for a lost item by keywords.
7. As a user, I want to filter lost items by category.
8. As a user, I want to see detailed information about a lost item.
9. As a user, I want to see images of the found items uploaded by help desk staff.
10. As a user, I want to be able to filter items by location (where they were lost or found).
11. As a user, I want to be able to filter by the date the item was lost.
12. As a user, I want to flag an item that I believe to be mine for further verification.
13. As a user, I want to view the current status of my flagged items (claimed, in dispute, etc.).
14. As a user, I want to pre-input details like description, date, and location about my lost item to help match found items.
15. As a user, I want to receive a notification when an item matching my description of a lost item is found.
16. As a user, I want to be able to post a template of my lost item request on social media (if time allows).
17. As a user, I want to claim my item through a verification process (e.g., providing additional details or proof).
18. As a user, I want to report an item I have found.
19. As a user, I want to see a map of the nearby help desks to drop off the items I found.
20. As a user, I want to view the items I have reported as found.
21. As a user, I want to modify my claim on a lost item in case of error.
22. As a user, I want to get a map view of the locations where lost items were reported.
23. As a user, I want to see the top items being reported as lost (based on trends).
24. As a user, I want to be able to “register” a personal item: generate a unique identifier/QR code for a personal item that maps directly to my user account and item details.
25. As a user, I want to be able to save the QR as a printable sticker and be provided steps on printing at a designated location (WALC).
26. As a user, I want to be able to view all, add and delete my registered items.
27. As a user, I want to see a list of all previously claimed items.
28. As a user, I want to have my recently claimed items to be automatically registered in my account.

29. As a user, I want to start a claim dispute form on wrongly claimed items.
30. As a user, I want to be able to communicate with the help desk in case of disputes.
31. As a user, I want to leave a review or feedback on the lost and found service.
32. As a user, I want to be able to view a listing of items moved to the surplus store for sale (if time allows).
33. As a user, I want to be able to buy items online from the surplus store (if time allows).

Front Desk/Staff Side -

1. As a staff member, I want to log in with a secure staff-only authentication.
2. As a staff member, I want to enter a new found item into the system with details (location, description).
3. As a staff member, I want to upload an image of a found item to the system.
4. As a staff member, I want to tag the item with specific characteristics (e.g., size, brand, color).
5. As a staff member, I would like to generate searchable keyword descriptions for listings using visual search AI.
6. As a staff member, I want to transfer an item to the central lost and found facility in the system.
7. As a staff member, I want to notify the system when an item has been claimed by its rightful owner.
8. As a staff member, I want to update the status of an item when it moves from one location to another.
9. As a staff member, I want to flag items for special attention (e.g., high-value items).
10. As a staff member, I want to log disputes between students over claimed items.
11. As a staff member, I want to see a list of all unclaimed items that have been in the system for more than 1 week.
12. As a staff member, I want to track the history of the movement of an item across help desks.
13. As a staff member, I want to archive items that have been claimed or moved to the surplus store for sale.
14. As a staff member, I want to manage listings on the online surplus store marketplace (if time allows).
15. As a staff member, I want to print a receipt for students when they claim a lost item.
16. As a staff member, I want to see reports on lost and found activity (number of items, most common locations).
17. As a staff member, I want to be able to scan pre registered items for easy verification of ownership.
18. As a staff member, I want to send alerts to students who have claimed items for pickup.

19. As a staff member, I want to verify students' identification before releasing an item.
20. As a staff member, I want to delete or edit incorrect entries in the system.
21. As a staff member, I want to categorize found items in bulk (for batch uploads).
22. As a staff member, I want to use filters to quickly find items that have been reported as lost by a specific student.
23. As a staff member, I want to see the real-time status of all items at my location.
24. As a staff member, I want to see which items have been flagged for review by students.
25. As a staff member, I want to receive automated reminders to transfer items to the central lost and found.
26. As a staff member, I want to view activity logs for each item to ensure accountability.
27. As a staff member, I want to post weekly unclaimed item listings to social media (if time allows).

Non-Functional Requirements:

- **Architecture and Performance**

Database: sqlite3

Frontend: React

Backend: Flask

BoilerTrack will use a modern web architecture with a React frontend and a Flask backend. We will use SQLite for data storage. The frontend built with React will provide a user-friendly interface for the reporting and claiming of lost items. The Flask backend, powered by Python, will handle the core application logic, manage data processing, API requests, and integrations with the database. The lightweight SQLite database will store all item-related data and user-account related data. RESTful APIs will ensure smooth communication between the front and back ends, handling tasks like user authentication and claim management. This architecture will provide an efficient and scalable solution to centralize lost and found operations across Purdue University.

- **Security**

For BoilerTrack, we need to securely store user accounts. We plan on using Supabase for security of user account information. We will use OAuth with authentik. University Staff will have special permissions to add/update/delete items that regular users will not have.

- **Usability**

The UI should be minimalist, showing only essential information like search fields and item lists. We want the main page to load in 500 ms. Navigation must be easy, with users able to browse, search, and report items within a few clicks. Consistent design across all pages is key. Feedback should include clear success and error messages. Search should allow filtering by date, location, and item type, with fast, relevant results. A “Help” or FAQ section will address common questions. The app must stay under 1 GB by limiting image sizes and unnecessary code, ensuring no lag with efficient API calls and database management. We want to load 100 concurrent instances.

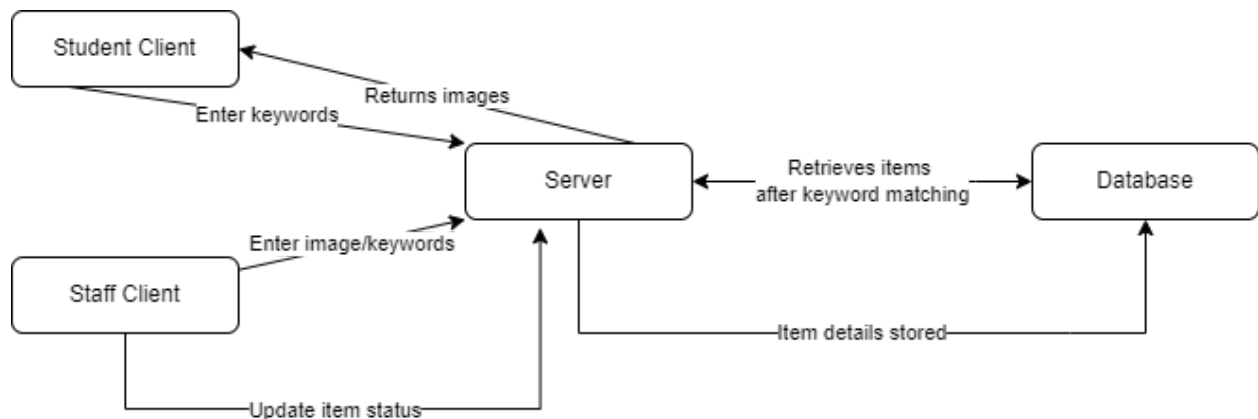
- **Hosting/Deployment**

We will be hosting in containers for easy reproducibility using docker-compose on a free Oracle vps. This will allow us to bring up all the services (like the container for authentik, and its supporting entourage) for the application in one command.

Design Outline:

High Level Overview

BoilerTrack will have two main interfaces through the web application: the student client and the staff client. When an item is given to staff, they start the application on the staff client and can create a listing for the item by inputting keywords and images. They can also update a listing based on its claim status. This information is then sent to the server to create the listing and store the item and its attributes (keywords and images) in the SQLite database. Then, a student who has lost an item will start the application on the student client and enter keywords in order to find their lost item. These are sent to the server and it attempts to retrieve the item listing based on the keywords from the database. If found, the server returns the listing to the student client.



1. Student Client

- The student client provides the user an interface to search through listings for their lost item(s) using keywords that they enter.
- The student client validates and sends the keyword input to the server in order to find the item.

2. Staff Client

- The staff client provides the staff an interface to create listings using images and keywords the staff enters.
- The staff client validates and sends the image and keywords to the server in order to create the listing.

3. Server

- The server stores item listing details entered through the staff client in the database.
- The server retrieves listings from the database based on keywords entered by the student client.
- The server returns the listings that it retrieved to the student client.

4. Database

- An SQLite database stores whether an item has been claimed by a student.
- The database stores keywords, images, and claim status associated with listings.

Functional Issues:

1. Account Creation and Verification

Issue: What method should be used for verifying user identities when they create an account?

- Option 1: Name, university-issued email, password
- Option 2: Name, personal/any email, password
- Option 3: Name, phone number, password

Choice: Option 1

Justification: Since the system is tied to Purdue University, ensuring that only valid university members can create accounts is essential for security. Additionally, this makes contacting students involved in a dispute easier because the contact information put down is credible.

2. Lost Item Description Input

Issue: How should staff input details for a lost item?

- Option 1: Allow manual input of descriptions and characteristics.
- Option 2: Provide a guided input with dropdowns for item types, colors, and other features.
- Option 3: Allow for auto-generation of keywords for item using ML Model/AI and optional manual input of description

Choice: Option 3

Justification: Option 3 provides an efficient balance between automation and flexibility. AI-generated keywords streamline item entry, ensuring consistency and reducing staff workload, while optional manual input allows for additional customization when necessary. This hybrid approach improves both accuracy and scalability in handling lost item descriptions.

3. Claim Dispute Process

Issue: How should disputes over claimed items be resolved?

- Option 1: Fully automated resolution based on pre-set rules (e.g., time of report, details provided).
- Option 2: Involve human intervention from staff to manually review disputes via scheduled meetings between staff and all parties involved.
- Option 3: Allow for direct communication between all parties involved in the dispute via in-app messaging.

Choice: Option 2

Justification: Given the potential for sensitive personal belongings, a manual review process provides better assurance of fairness and accuracy in resolving disputes. Additionally, having a third party (staff) present would speed up the process and lead to faster resolution.

4. Notification System for Found Items

Issue: How should notifications be delivered to users about found items that match their lost item description?

- Option 1: Immediate email when an item matching the entered description (beyond a specified % threshold) is reported to be returned to a help desk.
- Option 2: Daily or weekly digest with a list of found items matching user queries.

Choice: Option 1

Justification: Immediate notification ensures faster claim times and increases user satisfaction with the system by reducing delays in retrieving important items.

5. Listing Recently Claimed Items

Issue: How should recently claimed items be displayed to students for review?

- Option 1: List all recently claimed items in a tabular form
- Option 2: Feed of items recently returned to help desk
- Option 3: List of items that can be filtered by date and item category

Choice: Option 1

Justification: Filtering helps students focus on items relevant to them, improving the efficiency of dispute resolution and item review. Additionally, we chose a tabular form over a feed for user's to be able to quickly scan over items recently returned to a help desk.

Non-functional issues:

1. What backend should be used?

- Nodejs/any other javascript runtime
- Flask
- Apache httpd

Choice: Flask

Justification: Most of our team is familiar with python, and the language has many libraries that are written in faster languages if something is performance sensitive. It also has a lot of debugging tools like PDB, and an integrated debugger. This application does not require too much asynchronous communication, so javascript isn't a necessity, and flask is lighter weight than apache httpd.

2. What front end should be used?

- React
- Angular
- htmx

Choice: React

Justification: React is the most popular web framework, so it should have good community support as well as rich documentation. The skills should also apply in the future owing to its popularity.

3. Where will the app be hosted?

- Raspberry pi in a closet
- Onion service
- Aws resellers like vercel
- Virtual server from the likes of oracle or akamai

Choice: VPS from oracle

Justification: Price. Oracle offers a free tier, so it will be more appealing than aws resellers. The VPS also has a public and static Ip address, so unlike self hosting, it will divert risk of ddos or similar away from a team member's ISP. Having a "raw" server will also let us develop our containerization skills.

4. What database will we use?

- Postgres
- Mysql
- Microsoft Sql server
- Sqlite
- Database provider like planetscale

Choice: Sqlite

Justification: Sqlite is a single file solution, so it is more simple to host, wipe, and backup than something like postgres. It is also self hosted, so it reduces the risk of data breaches if the provider gets compromised. There may be performance issues, but at this scale, the simplicity is a bigger benefit than scalability or performance.

5. What image classification model will we use?

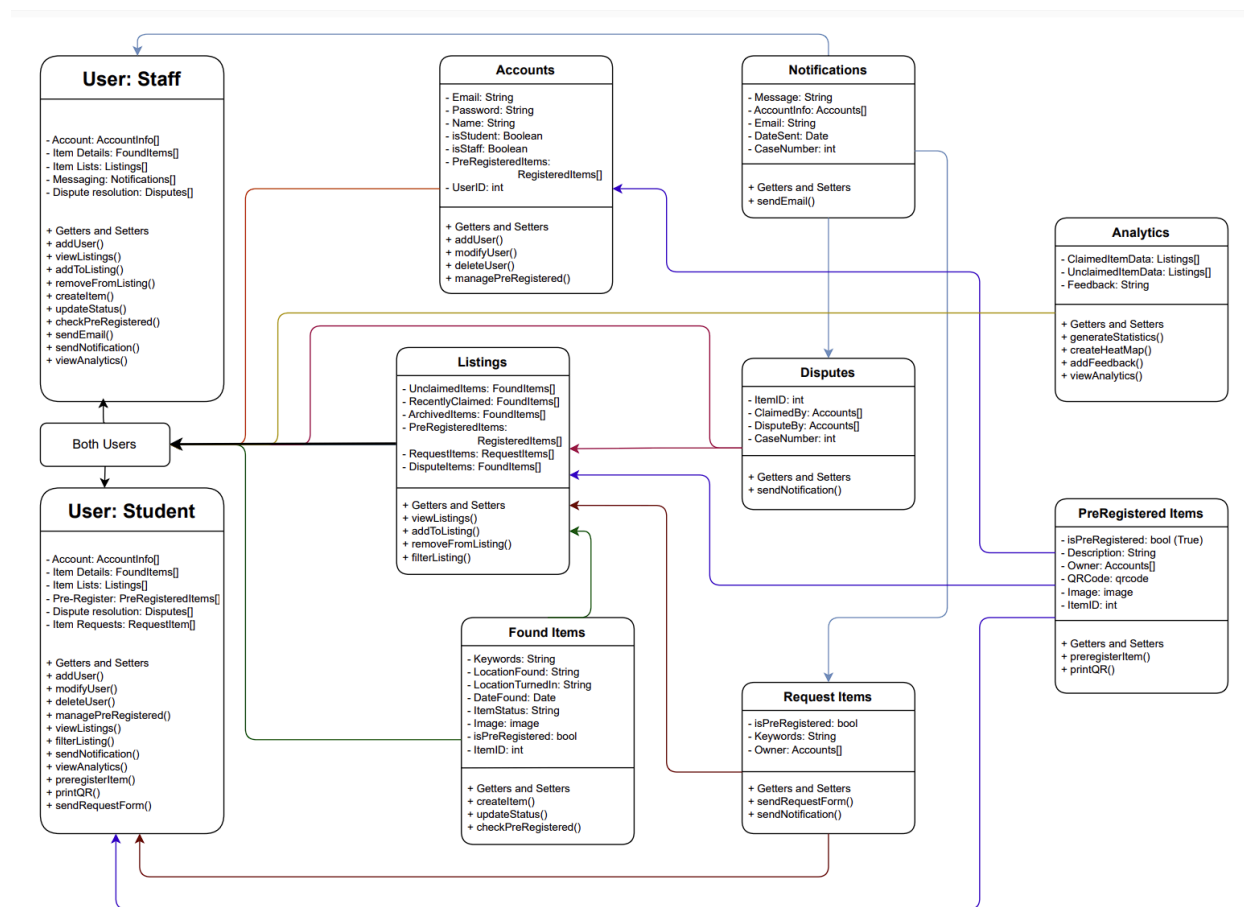
- CLIP
- YOLOv8
- Google API

Choice: Google api

Justification; The google api is much more rich than the small self hosted models, as well as offering \$300 in credits, so cost is not an issue. Offloading the AI to another will also help to minimize hosting cost because of not using as much cpu time.

Design Details:

Class design



Descriptions of Classes and Interaction between Classes

In the class design diagram, we structured the classes based on the various objects our project needs. Each class has a list of necessary attributes and methods.

Accounts:

- An account is created when someone signs up with their @purdue.edu email.
- Each user will have an email and password for login purposes.
- Each user will have a userId, name (desk name for Staff User), isStaff, and isStudent attribute.
- At any time, the user will be able to modify their password and name, but they cannot modify their email
- The user will be able to delete their account.

User: Staff

- Once signed up or logged in, the user is taken to the home page where they can add a found item into the database of items.
- The user will be able to view the items that were turned into their desk, active disputes, and claimed items.
- The user will be able to archive items after a month of them being claimed.
- The user will be able to view a master list of all unclaimed items, claimed items, archived items.

User: Student

- Once signed up or logged in, the user is taken to the home page where they can search for their item, pre-register their item, dispute an item.
- The user will be able to view their pre registered items, active disputes, claimed items, and the found items that they turned in.

Found Items:

- An Item object is created when the Staff User inputs a found item into the BoilerTrack system.
- For an item, the Staff user will input:
 - Keywords
 - Location found
 - Location turned in

- Date found
- Item status: claimed, unclaimed, in dispute
- Image
- isRegistered
- When a staff user adds a new item, it will be added to the list of unclaimed items.
- The Staff user can modify the item details at any time, archive it, or delete it.

PreRegistered Items:

- When a student user wants to pre register an item, they can navigate to the pre register button on the homepage.
- This would prompt them to enter the item details such as:
 - Keywords
 - Email and name of the person it belongs to
 - Image of the item
- Once these details are entered, an API will generate a QR code to be downloaded and printed out by the user.
- This item is then added to the preRegisteredItems array in the allListings class.

Request Items:

- When a student user has lost an item, they can choose to enter the into the system as a “requestedItem” with the following details
 - Keywords
 - Email and name of the person it belongs to
- This item is added to the requestedItems array in the allListings class.
- When an unclaimed items keywords matched the keywords for a requested item, then a notification is sent to the person whose requestedItem it is.

Listings:

- Every single item, regardless of status will be a part of the allListings class.
- When an item is added as unclaimed, it is added into the unclaimed array of allListings class
- When an item is modified at claimed, the item is moved from the unclaimed to the claimed array in the allListings class
- When a user clicks the dispute button on an item, the item is moved from the claimed array in the allListings class to the disputes array in the allListings class.
- After 6 months of the item being in the archive array of the allListings class, the item is deleted from the database.

Notifications:

- When a pre registered item is in the unclaimed items array, a notification is sent to the owner of the item via an email API
- When a student user claims an item, a notification is sent to them to ask them to pick it up from the respective desk.
- When a requested item is in the unclaimed items array, a notification is sent to the owner of the item via an email API

Sequence of Events Overview

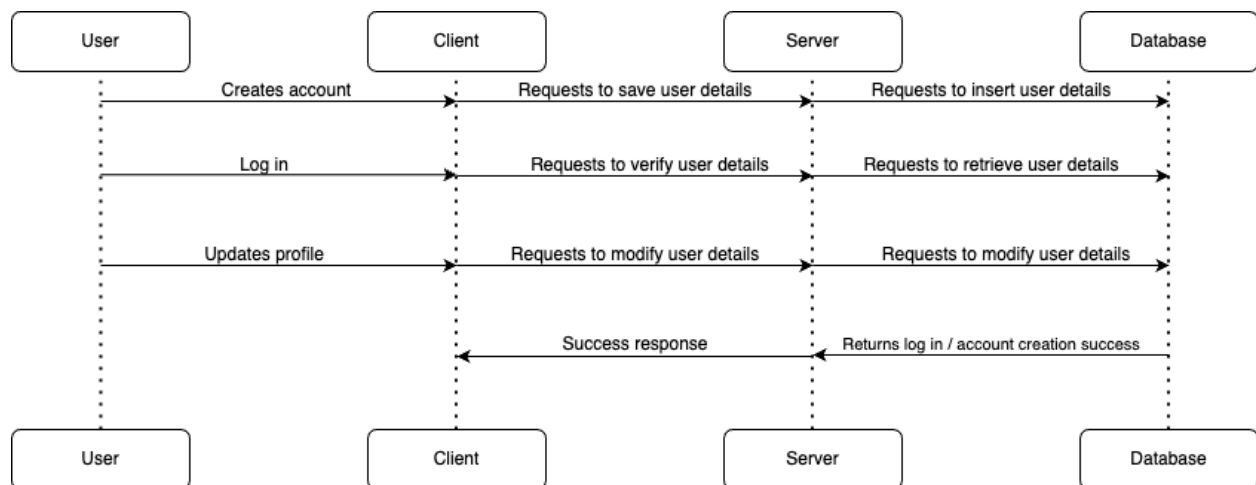
In the following diagrams, we have detailed the interactions between the clients (student user and staff user), server, and database.

When the user creates a new account or logs into their existing account, the user information is sent from the client to the server. The server then creates a query to retrieve or store the user's details in the SQLite database. After account creation or user log in, the user can search the catalog for their lost item, browse a directory of all lost items, report found items, claim items, and dispute items. Each item-found report, modification, claim, or dispute starts at the client and is sent as a request to the server. The server then processes this information and generates a query to store or update the database. Upon successful modification of the database with the new information, the database sends a success message to the client. When the client is searching for a lost item and wants to retrieve information from the database, the request goes from the client to the server, which queries the database and returns the relevant data to the client.

Sequence diagrams:

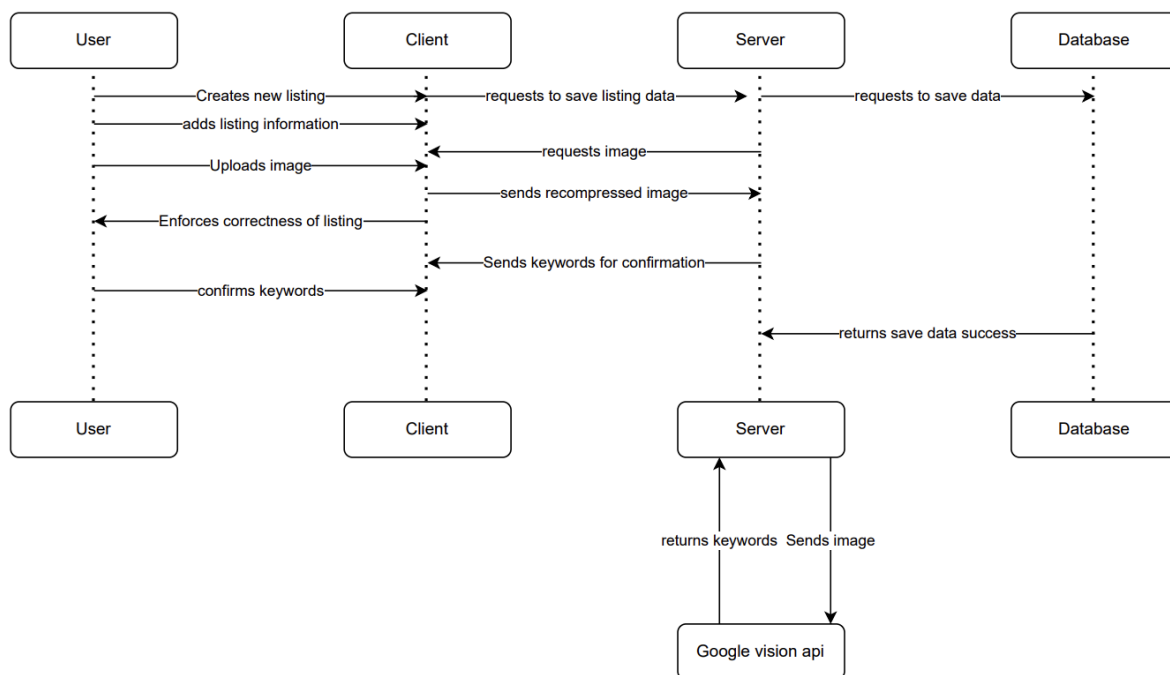
User account creation and login

When a user goes to the BoilerTrack web app, they have the option to either log in or create an account. In the event that the user chooses to create an account, the client sends a request to the server to store the user details in the database. In this case, the server will pass the user details to store in the database. In the event that the user chooses to log in, the client requests the server to verify user details. In this case, the server will retrieve the user details from the database. In the event that the user chooses to update their profile, the client requests the server to modify user details. In this case, the server will modify the user details from the database. In these situations, if the operation is successful, the database returns success message to the server and the server returns a success message to the client.



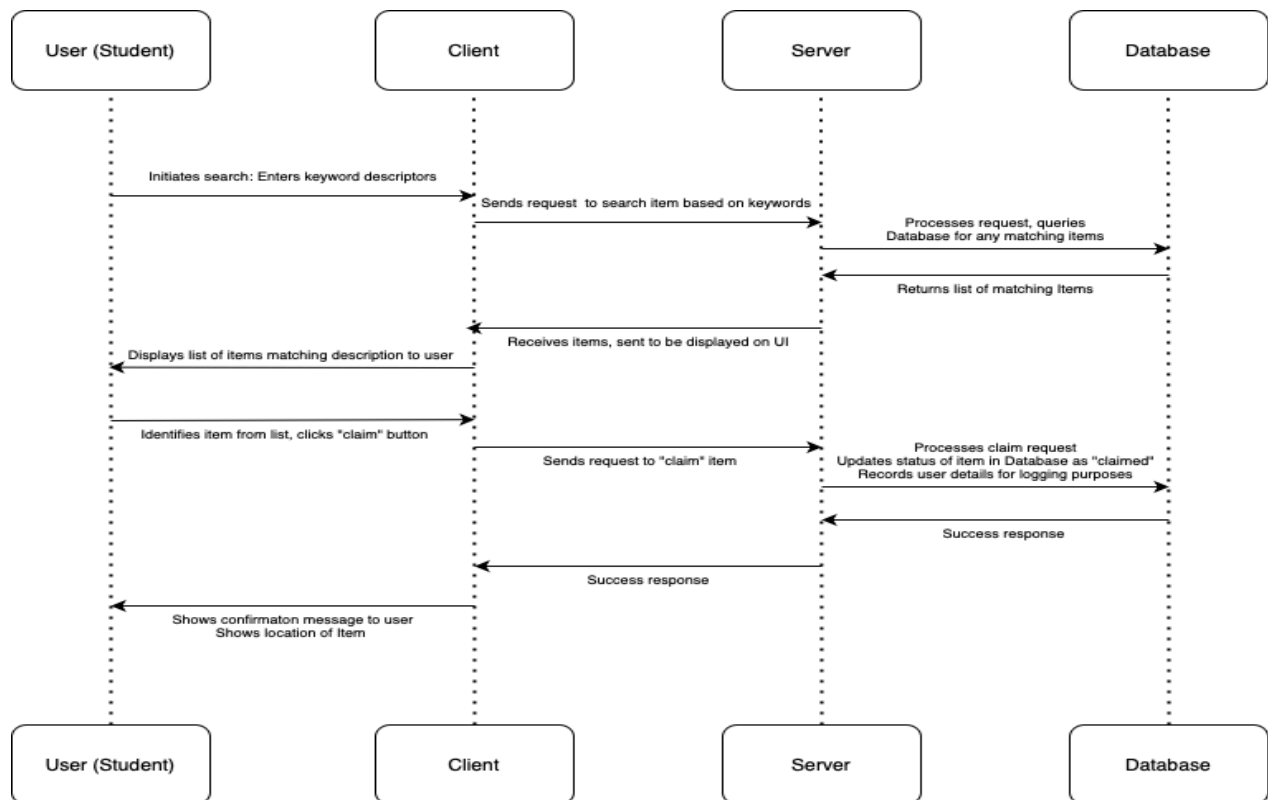
Listing item staff side

Staff will be presented with a page to add a listing to the database. They will be able to categorize it by if it is electronic, a food container, or otherwise. The client will enforce that all fields are filled, and will then send the text data to the server. The user will upload an image with the client to the server, where the image will be recompressed, and converted to a compatible format for the google api, and to save space on the database. The server will then send the image to the google api which will then return generated keywords. The client will then present these keywords to the user, and the user can confirm their correctness. After that, the client will send the rest of the data to the server, which will save it in the database.



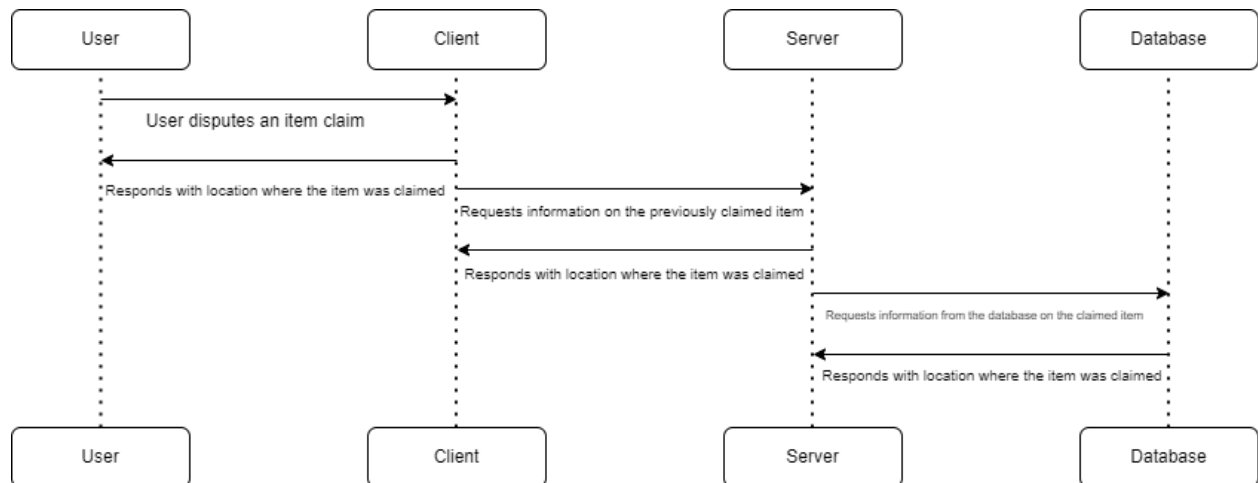
User searching for item by keywords and claiming

The user (student) can search for items by keyword descriptors. On entering keywords into a search bar, a search request is sent by the client to the server. The server queries the database for items that match the keywords entered by the student and sends this list of items to the user. On viewing this list of items, the student can “claim” an item they believe is theirs. This request to claim is sent to the Server, which updates the status of the item and logs the users details against the item in the database. The Server retrieves the current help-desk location of the item and sends it to the Client to be viewed by the user.



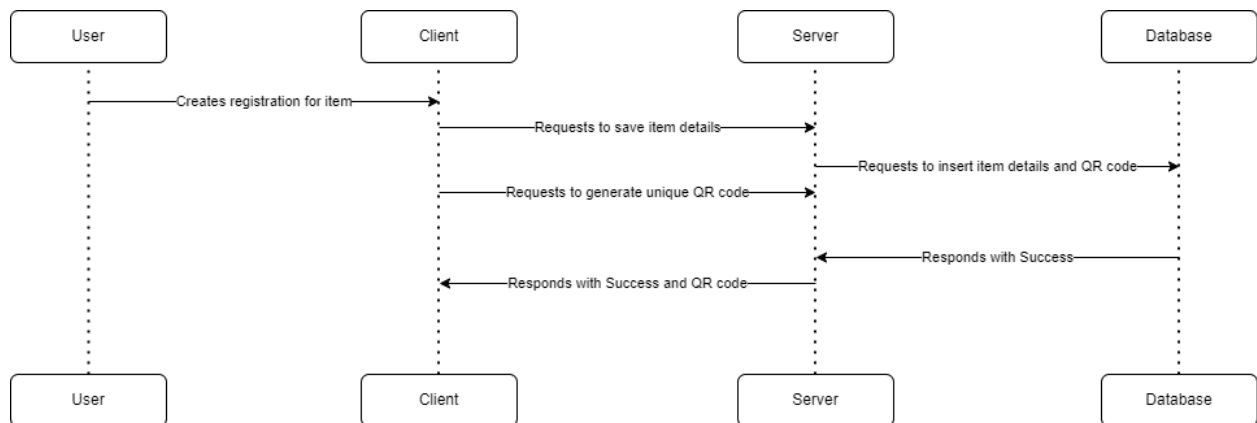
Dispute system

Users can dispute a past claimed item by entering their information on the app. The server will then send the location where the item was claimed so the user can make further inquiries in person.



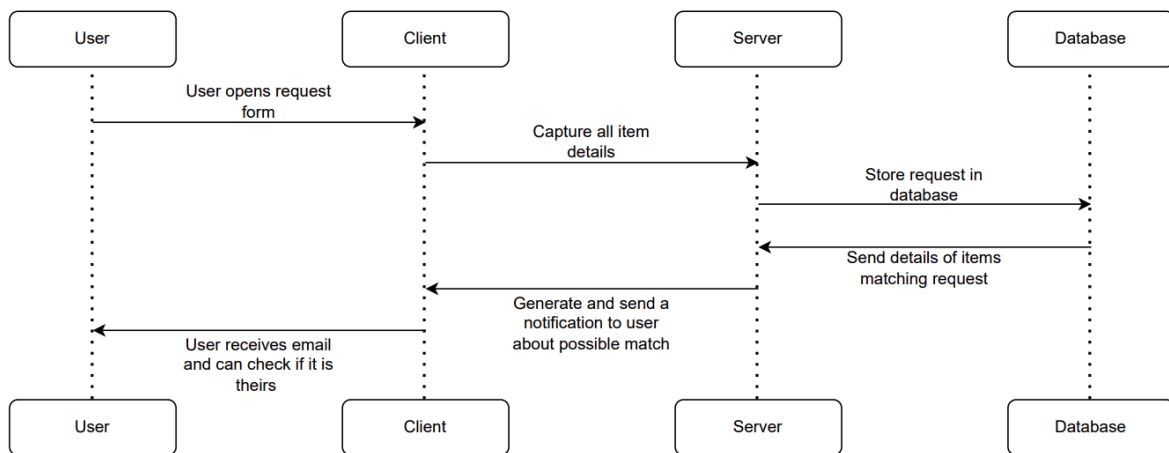
Preregistration

In order to find items easily when lost, users can pre register their items. They can register their item and the client requests the server to save the item details and generate a unique QR code. The server then requests to insert the item details and QR code associated with it in the database. The database then responds with success to the server. The server then responds with success and the generated QR code to the client. The user can print this QR code as a sticker and place it on their items.



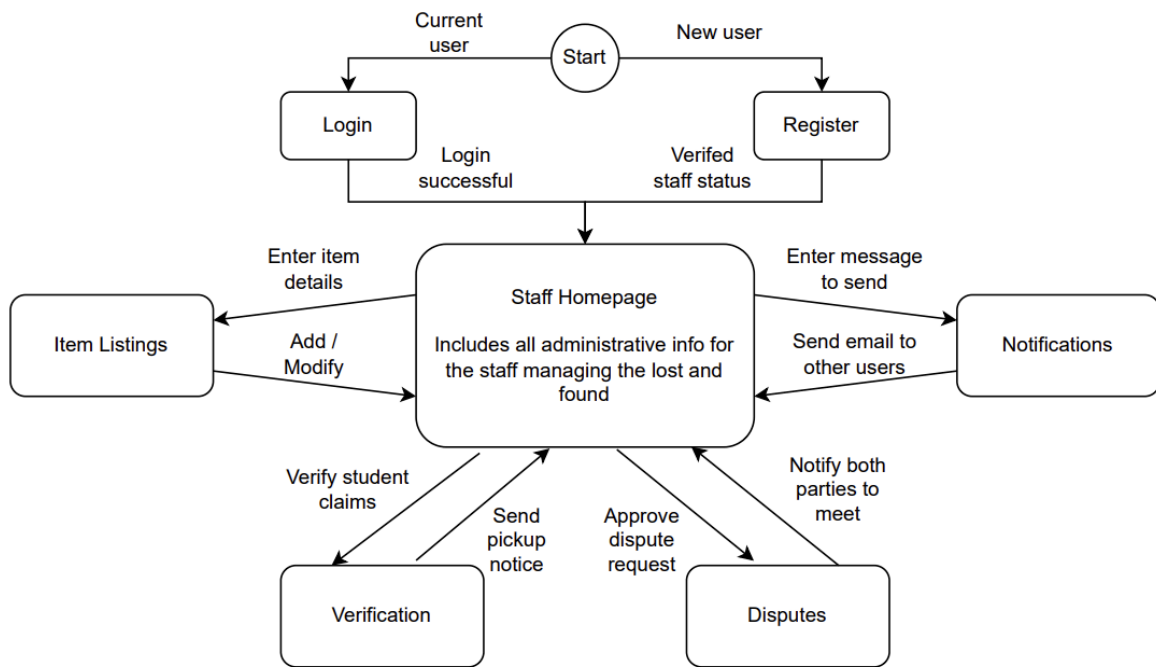
Requesting lost item

The sequence diagram shows how a student user enters details of a requested item into the system, which is stored in the database. When an unclaimed item's keywords match a requested item's keywords, the system generates a notification. The user receives an email alerting them to the potential match, allowing them to check if the item belongs to them.

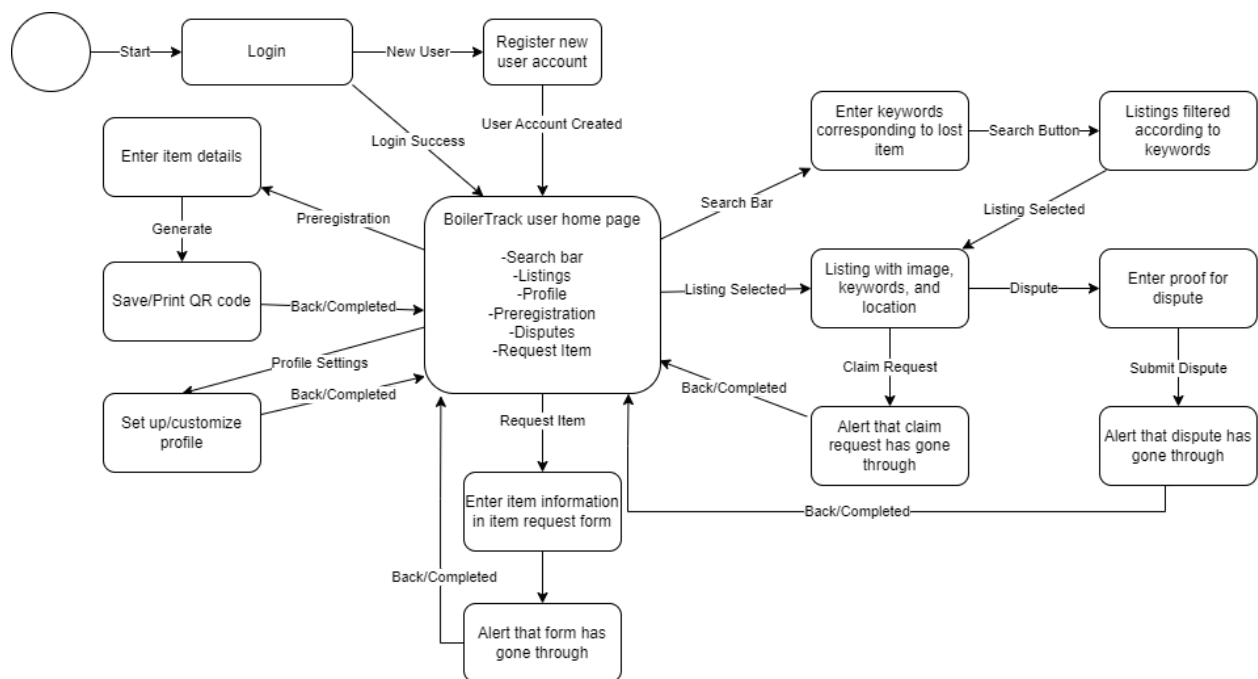


Navigation Flow Map:

Staff side



Student side



UI Mockups:

The image displays two UI mockups for a login and account creation interface, set against a dark background with a gold header bar.

Log In Mockup:

- Title:** Log In
- Email Address (@purdue.edu):** Input field.
- Your Password:** Input field with a **Hide** toggle.
- Log in:** Button.
- Forget your password?:** Link.
- Don't have an account? Sign up:** Link.

Create an account Mockup:

- Title:** Create an account
- Email (@purdue.edu):** Input field.
- Phone:** Input field.
- We strongly recommend adding a phone number. This will help verify your account and keep it safe.** (Text below phone field)
- Password:** Input field with a **Hide** toggle.
- Sign in:** Button.
- Already have an account? Log in:** Link.

Additional Elements:

- We will use your email as your user ID.** (Text box on the left side of the Create an account form)
- Sign in** (Text on the left side of the Create an account form)

Pre-Register

Upload

Analytics


Q

Lost smartphone

Past Items X X

Include past items


Feed



Samsung Galaxy

red, orange, yellow, green, blue,....

Claim

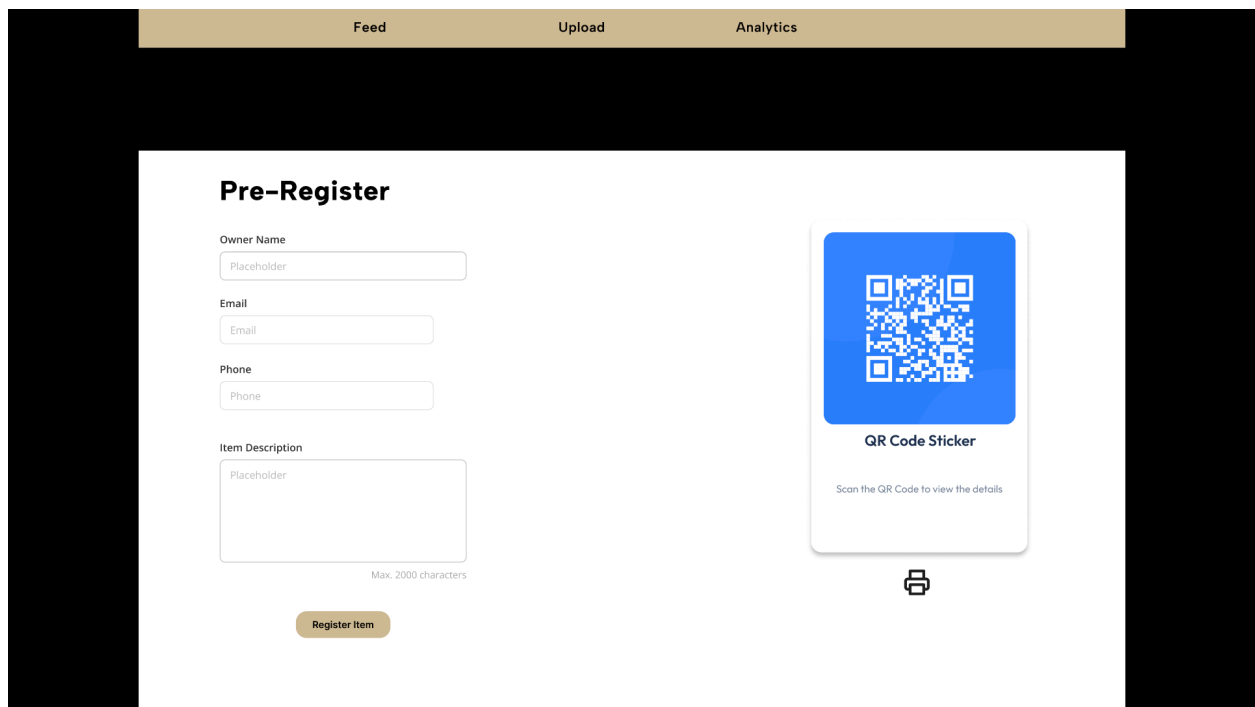
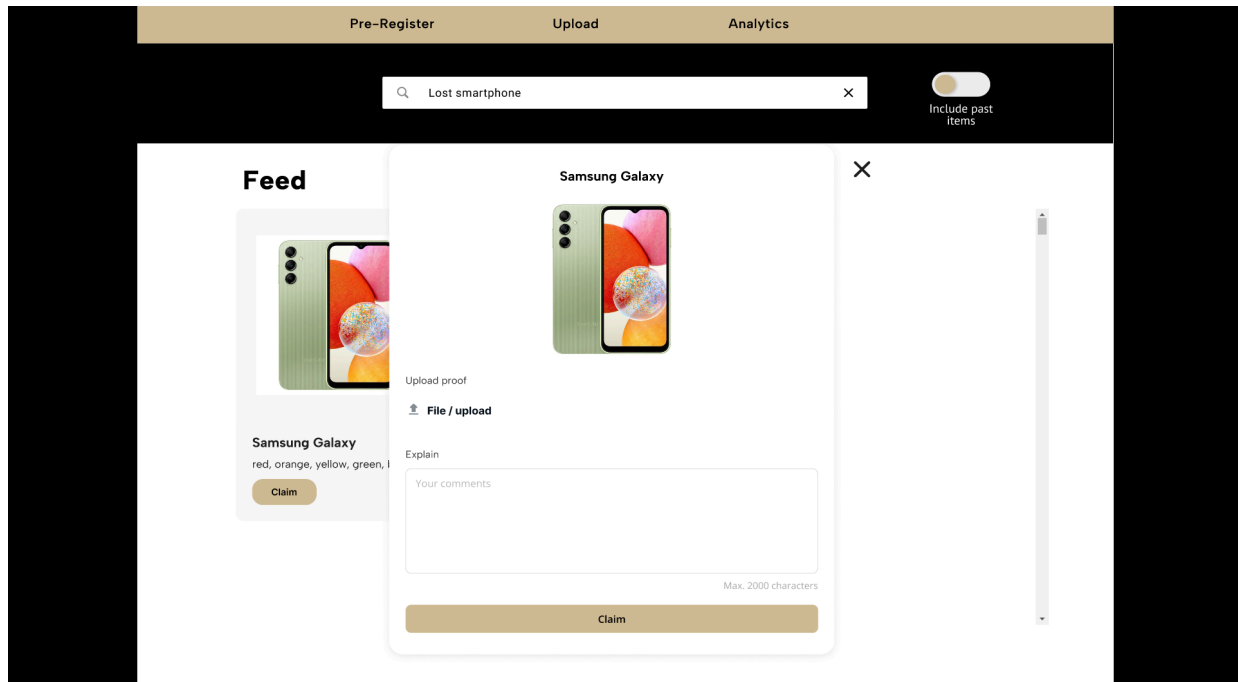


Apple iPhone

✓ Claimed

red, orange, yellow, green, blue,....

Dispute



Pre-Register

Feed

Analytics

Upload



Item Name

Placeholder

Item Description

Placeholder

Max. 2000 characters

Keywords

Please Select

Generate with AI

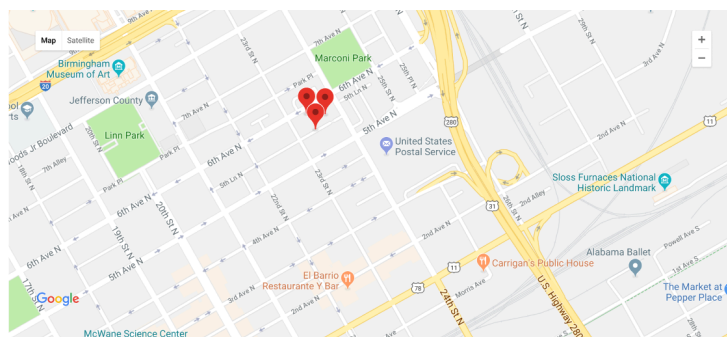
Upload

Pre-Register

Feed

Upload

Analytics

Claimed
4Unclaimed
34