



CS 30700
Design Document

Team 32:

Ali Fareed Shirazi
Sara Xiao
Jun Shern Lim
Ming Yan Ti



Index

■ Purpose	2
○ Functional Requirements	2
○ Nonfunctional Requirements	4
■ Design Outline	6
○ High-Level Overview	6
○ Sequence of Events Overview	8
■ Design Issues	10
○ Functional Issues	10
○ Non-Functional Issues	12
■ Design Details	14
○ Class Design	14
○ Descriptions of Classes and Interaction between Classes	15
○ UI Mockups	17
○ Activity Diagram	21
○ Sequence Diagrams	22



Purpose

Consider a situation where a university student needs a scientific calculator for a single exam; purchasing one makes no sense. Hence, people might want to use certain items for just a fixed period of time instead of buying them. Similarly, there are people who want to rent their items so that these items can be purposeful for other people. Hence, there are demands to rent items as well as the supply for these items.

The purpose of this project is to develop a web application as a platform to connect lenders and borrowers. Platforms like Facebook marketplace offer a similar interface, but their purpose is for selling/buying. This application will be a one-stop-station for users to borrow, lend, and chat with the borrowers/lenders for information. With Itembnb, users can offer up their items for borrowing and borrow other people's items. Our project comes with a calendar and chat system to facilitate communication between users, making the borrowing/lending process more smooth and efficient.

Functional Requirements

*Note: A user is both a lender and a borrower.

1. User account

As a user,

- a. I would like to register for an Itembnb account.
- b. I would like to log in to my account once I log out.
- c. I would like to have the option to delete my account.
- d. I would like to view/edit my own user profile.
- e. I would like to view other users' profiles, showing a list of their posted items.

2. Landing page

As a borrower,

- a. I would like to see a feed for items up for borrowing.
- b. I would like to search for available items.
- c. I would like to filter search items by category, tags, price range, etc.

As a lender,

- a. I would like to see a feed for items that people have requested.

3. Item postings

As a borrower,

- a. I would like to view a posted item's details - description, lender profile, date, etc.
- b. I would like to favorite a posted item to save it for later.
- c. I would like to see a list of all my favorited item postings.



- d. I would like to view a posted item's availability calendar.
- e. I would like to add a borrowing request to the calendar.
- f. I would like to view my borrowing history.

As a lender,

- a. I would like to post an item.
- b. I would like to add a price for borrowing my item to my posted item.
- c. I would like to add a thumbnail image and multiple preview images to my posted item.
- d. I would like to delete my posted item.
- e. I would like to edit my posted item.
- f. I would like to set up my posted item's calendar.
- g. I would like to edit my posted item's calendar.
- h. I would like my calendar to be automatically updated when an item is booked.
- i. I would like to view the list of borrowing requests for my posted items.
- j. I would like to be notified when a borrower has requested to borrow my item.

4. Item requests

As a borrower,

- a. I would like to create an item request for an item I want to borrow that is not listed on the site.
- b. I would like to edit my item request.
- c. I would like to delete my item request.
- d. I would like to be notified when a lender has linked one of their posted items to my item request.

As a lender,

- a. I would like to link one of my posted items to an existing item request.

5. Item reservations

As a borrower,

- a. I would like to receive a notification one hour prior to my borrowing deadline.
- b. I would like to confirm that I have received the item from the lender.

As a lender,

- a. I would like to be able to accept or deny a borrower's request.
- b. I would like to view my lending history.
- c. I would like to update the posted item's status.
- d. I would like to confirm that I have received the item back from the borrower.



6. Chat system

As a user,

- a. I would like to be able to view a simple chat interface between myself and other users.
- b. I would like to send and receive chat messages with another user for in-progress lending activities (items I have borrowed but have yet to return) in real time, as well as get notified if someone messages me.

7. Rating system

As a user,

- a. I would like to rate and review a lender or borrower after they've completed a transaction with me.

As a borrower,

- a. I would like to rate and review an item I have borrowed.

8. Map System (if time allows)

As a borrower,

- a. I would like to see a map view of items up for borrowing near me.

9. Payment System (if time allows)

As a borrower,

- a. I would like to send a payment to a lender for lending an item to me.
- b. I would like to make a deposit so that I can borrow a lender's item.
- c. I would like to receive my deposit back for returning the item to the lender.

As a lender,

- a. I would like to receive payment from a borrower for lending my item to them.
- b. I would like to receive a deposit for lending my item.
- c. I would like to reimburse the deposit to the borrower for returning my item to me.

Nonfunctional Requirements

1. Server

As a developer,

- a. I would like the server to store and retrieve data from a cloud database (MongoDB).
- b. I would like the server to be able to communicate with third-party API servers.
- c. I would like the server to handle exceptions properly instead of crashing.
- d. I would like the server to support real-time communication with the client.



2. Client

As a developer,

- a. I would like the client to be any up-to-date web browser, e.g., Chrome, Safari, Firefox, etc.
- b. I would like the client to support real-time communication with the server.

3. Performance

As a developer,

- a. I would like the server to be efficient at processing big data so that it performs well with scale.
- b. I would like the website to have fast load times of < 3000ms for each web page.
- c. I would like the server to support multiple clients concurrently.

4. Usability

As a developer,

- a. I would like the website to adapt to screens of different sizes.
- b. I would like the website to be compatible with any up-to-date web browser client.
- c. I would like the website to have an intuitive and user-friendly design.
- d. I would like the website to be easy to navigate.

5. Security

As a developer,

- a. I would like to prevent any NoSQL injections.
- b. I would like to authenticate all users upon logging in.
- c. I would like to authenticate all API calls to the server.
- d. I would like to protect user privacy by encrypting user details in the database.
- e. I would like to safeguard chat history between users.

6. Hosting and Deployment (If time allows)

As a developer,

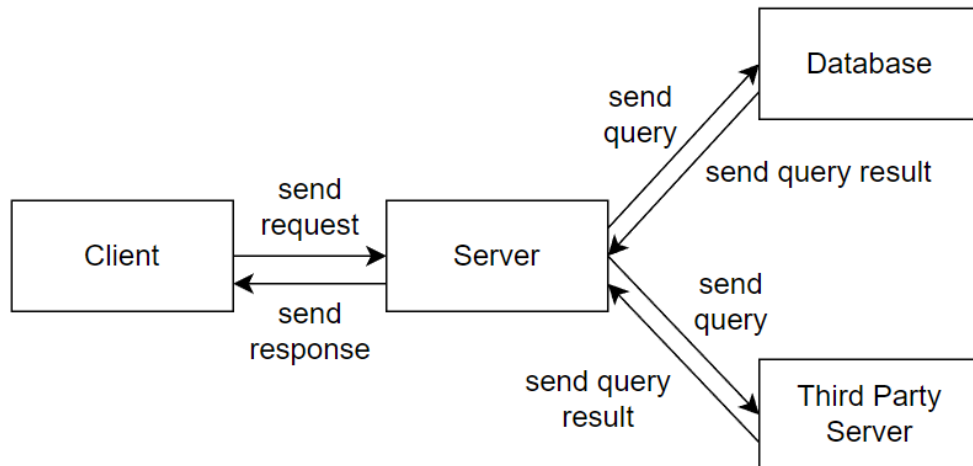
- a. I would like to host our application's front and back end separately on Heroku.



Design Outline

High-Level Overview

This project will be a web application that allows users to post items and requests, manage their lists of reservations and requests, and chat with other users. This application will use the client-server model. The server will be the abstraction layer for the client. The client(s) will send fetch requests to the server; then, the server will send queries to the database, as well as make fetch requests to third-party APIs for the chat system. The server will then send the responses back to the client(s).



1. Client
 - a. Client provides a user interface to our system.
 - b. Client sends fetch requests to the server via JSON format.
 - c. Client receives response from the server, format the response, and display the changes or results in the user interface.
2. Server
 - a. Server receives and handles fetch requests from clients.
 - b. Server will send queries to the database or to the third party server for chats, based on the clients' requests.
 - c. The server will receive responses from the database or the third party server, implement the business logic portion needed for parsing the response.



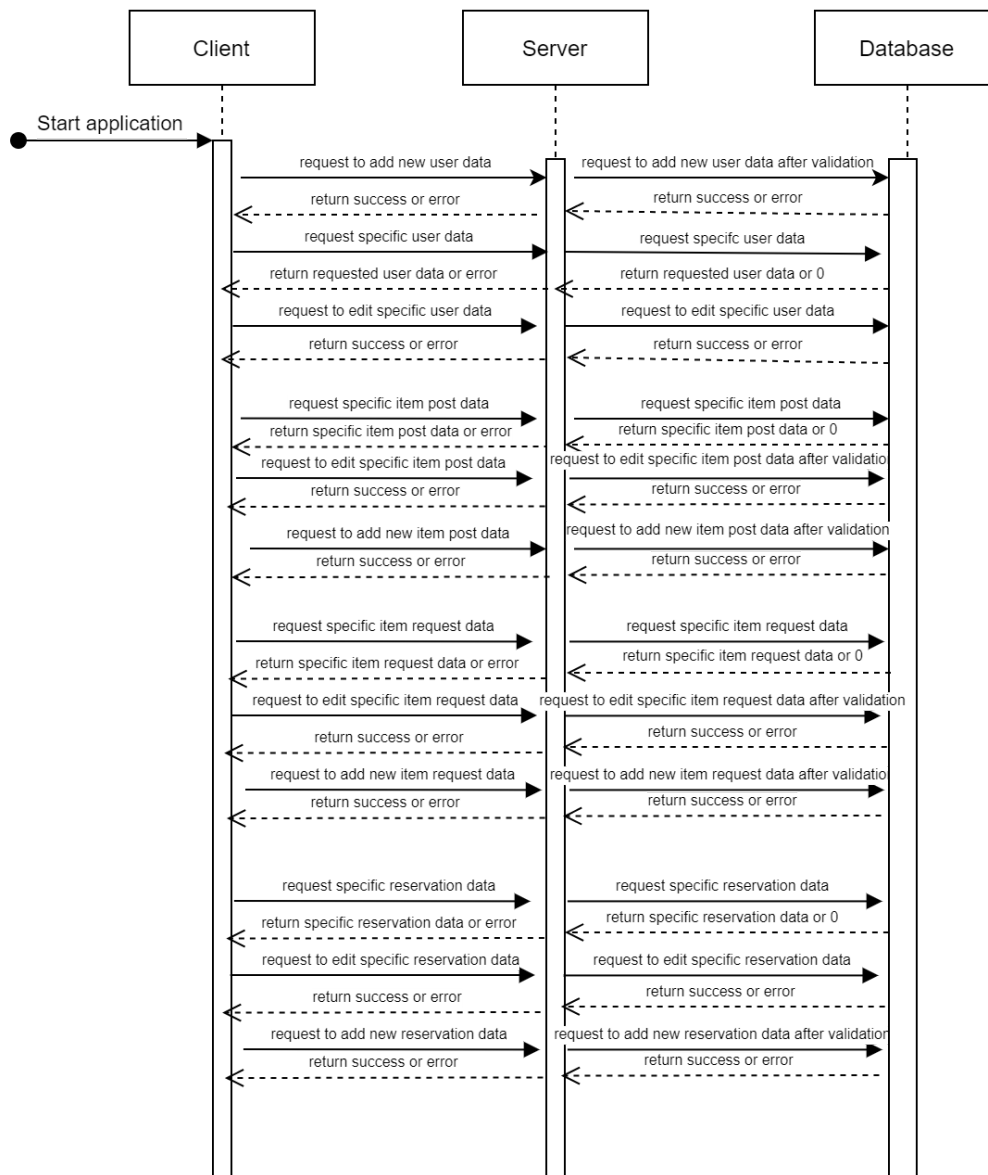
-
- d. The server will send the responses back to the clients.
3. Database
- a. Database stores all data and information used in the application.
 - b. Database receives queries from the server and sends the query results back to the server.
4. Third-party server for chat API (Talk.js)
- a. This server is a cloud server that stores all the users' chat information securely.
 - b. This server receives queries from the application server and sends the extracted data back to the application server.



Sequence of Events Overview

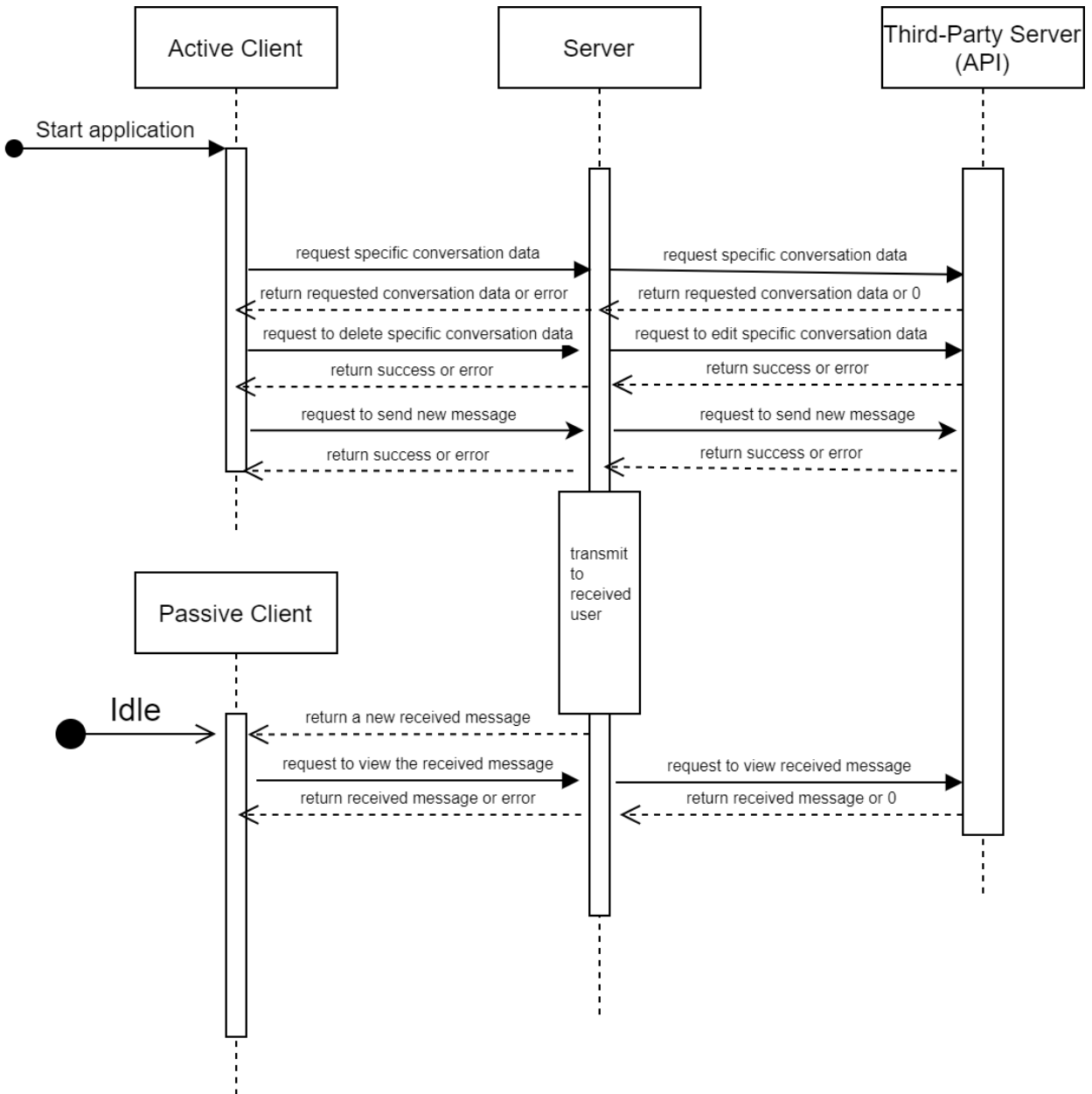
The sequence diagram below shows the typical interaction among clients, the application server, the database, and the third-party server. Once a user starts the web application, the sequence is initiated. As a user interacts with data on the user interface, the client will interact with the server requesting to either read, edit, or create data. Based on the client's request, the server will interact with either the database or the third-party server. The database and the third-party server will retrieve the relevant data or update the data, then either send the data back or a success signal to the server. The server then formats the data and sends the requested data back to the client.

A typical client-server-database interaction is as below.





A typical client-server-third-party-server interaction is as below.





Design Issues

Functional Issues

1. How should reservation history be preserved?

- Option 1: Delete automatically after X period of time
- Option 2: Reservation history is preserved for an indefinite period of time
- Option 3: Users can delete specific reservations (but only on their end)

Choice: Option 2

Discussion: Preserving reservation histories will allow borrowers to easily refer to items that they have previously borrowed. It will also help lenders see users who have previously borrowed their items. While choosing option 1 seems reasonable, we can't be sure how much reservation history each user would like to view. Hence, our decided time frame may satisfy some users and disappoint others. Option 3 is not a good idea either. To implement it, we would need to store two instances of a Reservation (one on the borrower's side and one on the lender's side) so that deleting one would not affect the other. For example, if Lender Alice deletes the reservation history between Lender Alice and Borrower Bob, Borrower Bob should still be able to see the reservation. This approach can be messy to implement and would take up a significant amount of space in the database as well.

2. How should ratings and reviews be configured?

- Option 1: Reviews/ratings for lenders only
- Option 2: Reviews/ratings for lenders and items
- Option 3: Reviews/ratings for borrowers and items
- Option 4: Reviews/ratings for lenders and borrowers only
- Option 5: Reviews/ratings for lenders, borrowers, and items

Choice: Option 5

Discussion: Providing a rating for borrowers, lenders, and items will benefit all users. From the lender's perspective, viewing a borrower's rating will allow the lender to determine if that borrower is trustworthy. From a borrower's perspective, viewing an item's and its lender's reviews will help the borrower determine if the item is of good quality and whether the lender is reliable. Thus, a user will have two ratings: a lender rating, which is the average of every one of their items, and a borrower rating, created from interactions with lenders where lenders review the borrowers after a reservation



period is complete. Items will have their own ratings that are given by borrowers also after the reservation period is complete.

3. How should items be categorized?

- Option 1: Custom user-defined tags
- Option 2: Predetermined list of categories
- Option 3: Both user-defined tags and predetermined categories

Choice: Option 2

Discussion: Having a predetermined list of categories to choose from will enable users to easily categorize their items. Searching under predefined categories may also make searches quicker and more productive. Option 1 may make it difficult for users to find items depending on how they tag them. It may also allow users to misuse the tag system by intentionally putting misleading tags on their items. Option 3 has similar problems as Option 1, and it may also overload users with choices and procedures that may dissuade them from using our application.

4. How should the UI allow lenders to satisfy borrower requests?

- Option 1: Drop-down menu with list of all your items; select the appropriate item
- Option 2: Lenders comment on the borrower request's post, linking their items
- Option 3: Multiple-select menu with list of all your items; select all items that could fulfill the borrower's request

Choice: Option 3

Discussion: A multiple-select menu like Option 3 offers a streamlined and simple interface that eliminates the need for separate storage of text or comments. This option also offers the lender the advantage of being able to choose from multiple items, which caters to situations where the borrower's request can be met through a combination of items rather than being limited to just one.

5. Should we automatically open a chat between a lender and a borrower once a reservation request has been approved?

- Option 1: Automatically open a chat between lender and borrower.
- Option 2: Let lenders and borrowers reach out to each other on their own.

Choice: Option 1

Discussion: It would be more convenient for both lenders and borrowers if a chat was opened automatically between the two parties. This way, both parties won't have to



expend additional effort to make plans for the item exchange, making the reservation experience more streamlined and user-friendly.

Non-Functional Issues

1. What project management tool should we use?

- Option 1: GitHub Issues
- Option 2: Jira
- Option 3: Trello

Choice: Option 1

Discussion: We will use Github Issues as our primary means of collaboration and task management. This platform offers the advantage of not requiring the creation of separate accounts or the need to constantly switch between multiple websites. Furthermore, sharing information and delegating tasks with team members becomes a seamless process as access to the root repository, where the issues reside, can be easily granted to any authorized individual. Its simplicity and ease of use make it a convenient choice for our project.

2. How should we build our front end?

- Option 1: Vanilla HTML5 and JavaScript
- Option 2: React
- Option 3: Angular
- Option 4: Vue
- Option 5: JQuery

Choice: Option 2

Discussion: React is easy to learn and accessible to those without much knowledge of the intricacies of vanilla Javascript. It is widely used in industry, which means documentation and resources on its use will be extensive. Its component system is easy and effective for managing parts of the UI, streamlining the development process. In contrast, JQuery is outdated, while Angular and Vue have a steeper learning curve and limited compatibility. Vanilla HTML5 and Javascript are rarely used and become difficult to manage with large projects.

3. What database should we use?

- Option 1: MySQL
- Option 2: MongoDB
- Option 3: Firebase



Choice: Option 2

Discussion: MongoDB and Firebase will be a better option than MySQL because they are document-based databases, which fits our need to have flexible schemas. MongoDB is the better option compared to Firebase because it is more robust, known for high performance and has a better security features as well. Hence, it is a more reliable and dependable database platform to store all of the information needed for our application.

4. What chat API should we use for communication between borrowers and lenders?

- Option 1: CometChat
- Option 2: Twilio
- Option 3: Sendbird
- Option 4: TalkJS

Choice: Option 4

Discussion: TalkJS offers a permanent free model, which allows us to run a beta version of our app at no cost. It does not require payment per message or impose a costly fee after a free trial expires, like CometChat or Twilio. Furthermore, it is designed to be more lightweight than other APIs, which are typically geared toward offering comprehensive customer service and call center functionality for well-established businesses.

5. How should we build the calendar feature to track item availability?

- Option 1: FullCalendar
- Option 2: Google Calendar API
- Option 3: Calendar Javascript API
- Option 4: Build our own calendar in React

Choice: Option 4

Discussion: Creating our own calendar will give us greater flexibility in customization and the opportunity to incorporate unique features that cater to our specific requirements. Conventional calendar APIs, such as Google Calendar, offer a wide range of functionalities that may not align with our goals. In our case, we only require the capability of scheduling item reservations instead of a full-fledged user calendar with additional features, like recurring events and setting time slots, that may not be necessary.



Design Details

Class Design





Descriptions of Classes and Interaction between Classes

User

- A **User** object is created every time someone signs up for our application.
- Each **User** object will have its own unique id and chat id (for the chat API).
- Each **User's** name, email, password, profile picture, and profile description will be set by their respective user during registration.
- Each **User** will have a list of favorited items and posted items storing **Item** ids.
- Each **User** will have a list of requested items storing **Request** ids.
- Each **User** will have a list of **Reservation** ids to keep track of the history of reservations they have made.
- Each **User** will have a lender and borrower rating, which will be on a scale of 0 to 5.

Reservation

- A **Reservation** object is created every time a borrower decides to reserve an item posted by a lender.
- Each **Reservation** object has its own unique id.
- Each **Reservation** will store the **Item's** id as well as two **User** ids: the borrower's and lender's ids.
- A **Reservation** may or may not have a **Request** id. If the item being reserved was recommended to fulfill a request post, then a **Request** id would be present. Else, it would be null.
- Each **Reservation** will have a start date and end date.
- Each **Reservation** will have an approval status stored as a number to signify different statuses, e.g., 0: pending, 1: approved, 2: denied, etc.

Post

- The **Post** class is a superclass of the **Item** and **Request** classes.
- A **Post** object is created every time a user creates an **Item** post or a **Request** post.
- A user cannot directly create a **Post** object.
- Each **Post** object will have its own unique id.
- Each **Post** will store a name, description, date of when it was created, and its owner's id.



Item

- The **Item** class is a subclass of the **Post** class.
- An **Item** object is created whenever a user wants to put an item up for borrowing.
- Each **Item** will store a list of **Reservation** ids, which show users that have borrowed the **Item** in the past.
- Each **Item** will store a list of images, which will be provided by the lender.
- Each **Item** will have a rent price, which will be determined by the lender.
- Each **Item** will have a list of unavailabilities, which will be represented as tuples: (start date, end date, id of borrower).
- Each **Item** will have a list of **ItemReview** objects.
- Each **Item** will have a list of **Category** enums, which will be selected by the lender.

Request

- The **Request** class is a subclass of the **Post** class.
- A **Request** object is created whenever a user wants to make request post for an item they would to borrow.
- Each **Request** object will have resolved status to indicate whether it's been fulfilled.
- Each **Request** will store a list of **Item** ids that represent items that other users have recommended to fulfill the request.

ItemReview

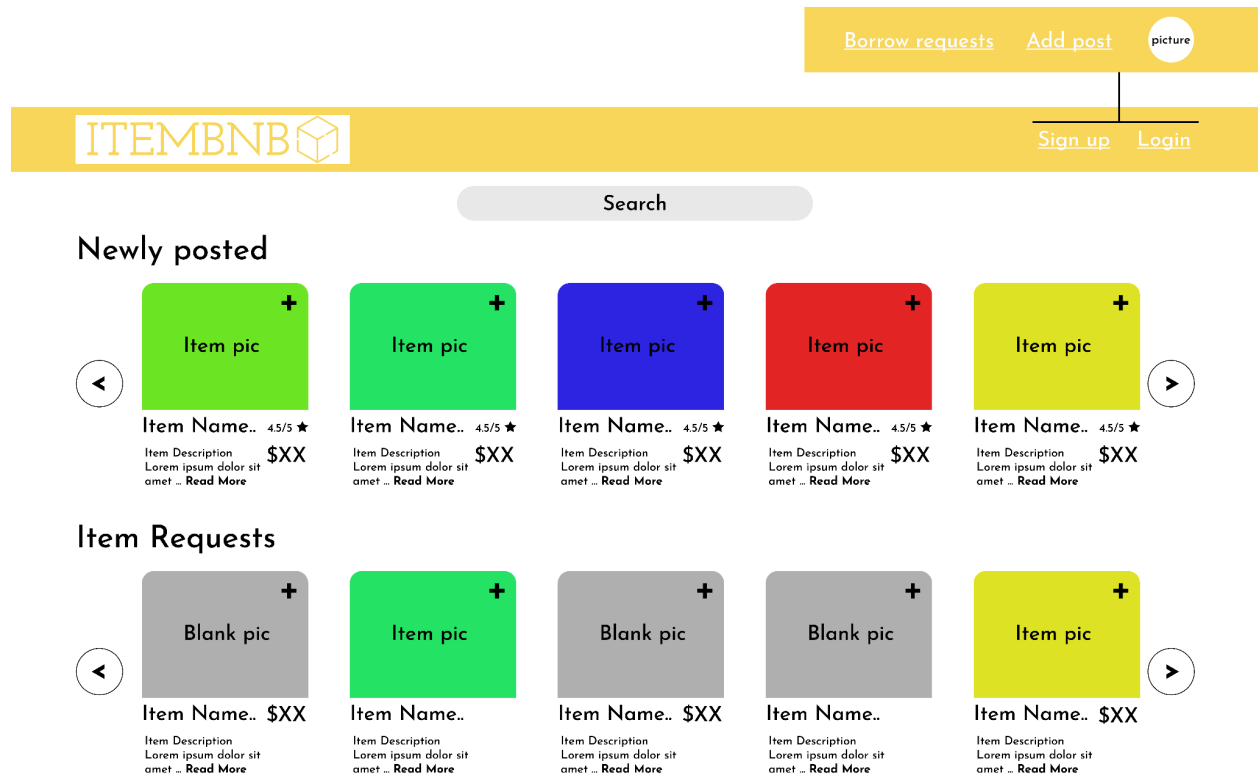
- An **ItemReview** object is created whenever a user leaves a review for a borrowed **Item**.
- Each **ItemReview** will have its own unique id.
- Each **ItemReview** will contain the reviewer's (**User**) id as well as the **Item's** id.
- Each **ItemReview** will store a rating (on a scale from 0 to 5), a review text, and the date the review was last modified.

<Enum> Category

- The *Category* enum stores predefined categories that users can select for items they would like to put up for borrowing.
- Currently, the *Category* enum has six defined constants: ACADEMICS, HOUSEHOLD, ENTERTAINMENT, OUTDOOR, ELECTRONIC, and MISC.



UI Mockups



Homepage

This page will appear upon visiting the website and will either display “sign up/login” in the top right if the user is not logged in, or display the user’s profile picture with buttons to create a new post or view any incoming requests to borrow the user’s items.



This page will appear once a user clicks on an item listing, allowing them to view more details regarding the item and place a borrow request. Depending on the amount of time selected, a price estimate will appear.



ITEMBNB

Borrow requestsAdd post

Search term

Results for "Search term"

+

Item pic

Item Name.. 4.5/5 ★
Item Description
Lorem ipsum dolor sit
amet ... Read More
\$XX

+

Item pic

Item Name.. 4.5/5 ★
Item Description
Lorem ipsum dolor sit
amet ... Read More
\$XX

+

Item pic

Item Name.. 4.5/5 ★
Item Description
Lorem ipsum dolor sit
amet ... Read More
\$XX

+

Item pic

Item Name.. 4.5/5 ★
Item Description
Lorem ipsum dolor sit
amet ... Read More
\$XX

+

Item pic

Item Name.. 4.5/5 ★
Item Description
Lorem ipsum dolor sit
amet ... Read More
\$XX

+

Item pic

Item Name.. 4.5/5 ★
Item Description
Lorem ipsum dolor sit
amet ... Read More
\$XX

+

Item pic

Item Name.. 4.5/5 ★
Item Description
Lorem ipsum dolor sit
amet ... Read More
\$XX

+

Item pic

Item Name.. 4.5/5 ★
Item Description
Lorem ipsum dolor sit
amet ... Read More
\$XX

+

Item pic

Item Name.. 4.5/5 ★
Item Description
Lorem ipsum dolor sit
amet ... Read More
\$XX

+

Item pic

Item Name.. 4.5/5 ★
Item Description
Lorem ipsum dolor sit
amet ... Read More
\$XX

+

Item pic

Item Name.. 4.5/5 ★
Item Description
Lorem ipsum dolor sit
amet ... Read More
\$XX


+

Item pic

Item Name.. 4.5/5 ★
Item Description
Lorem ipsum dolor sit
amet ... Read More
\$XX

Search Results Page

This page will appear and show all relevant items when a search term is entered.



Sign Up

Name:

Email:

Purdue:

Password:

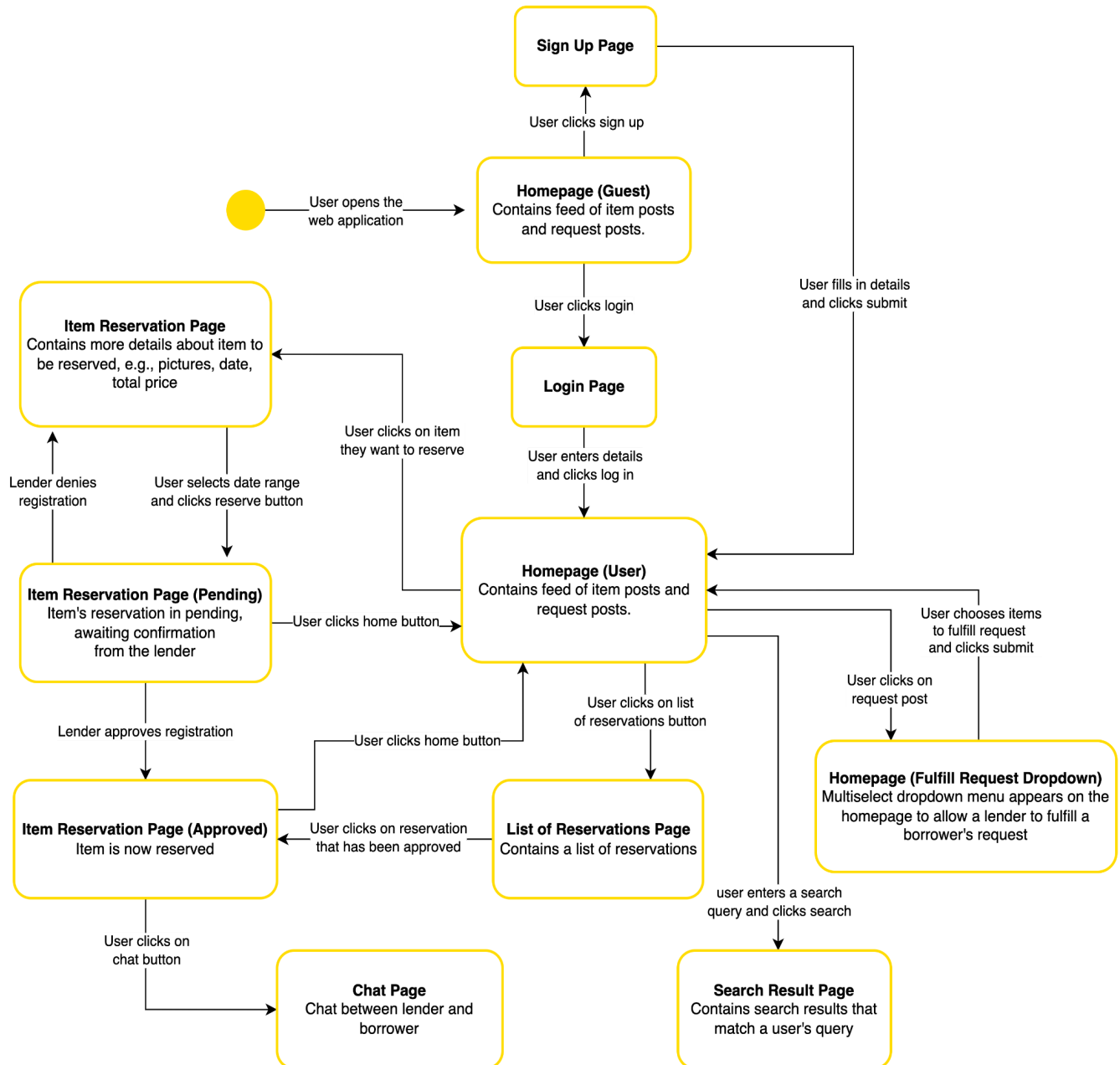
Confirm password:

Sign Up Page

When users click the “sign up” button, they are redirected to this page with a signup form.



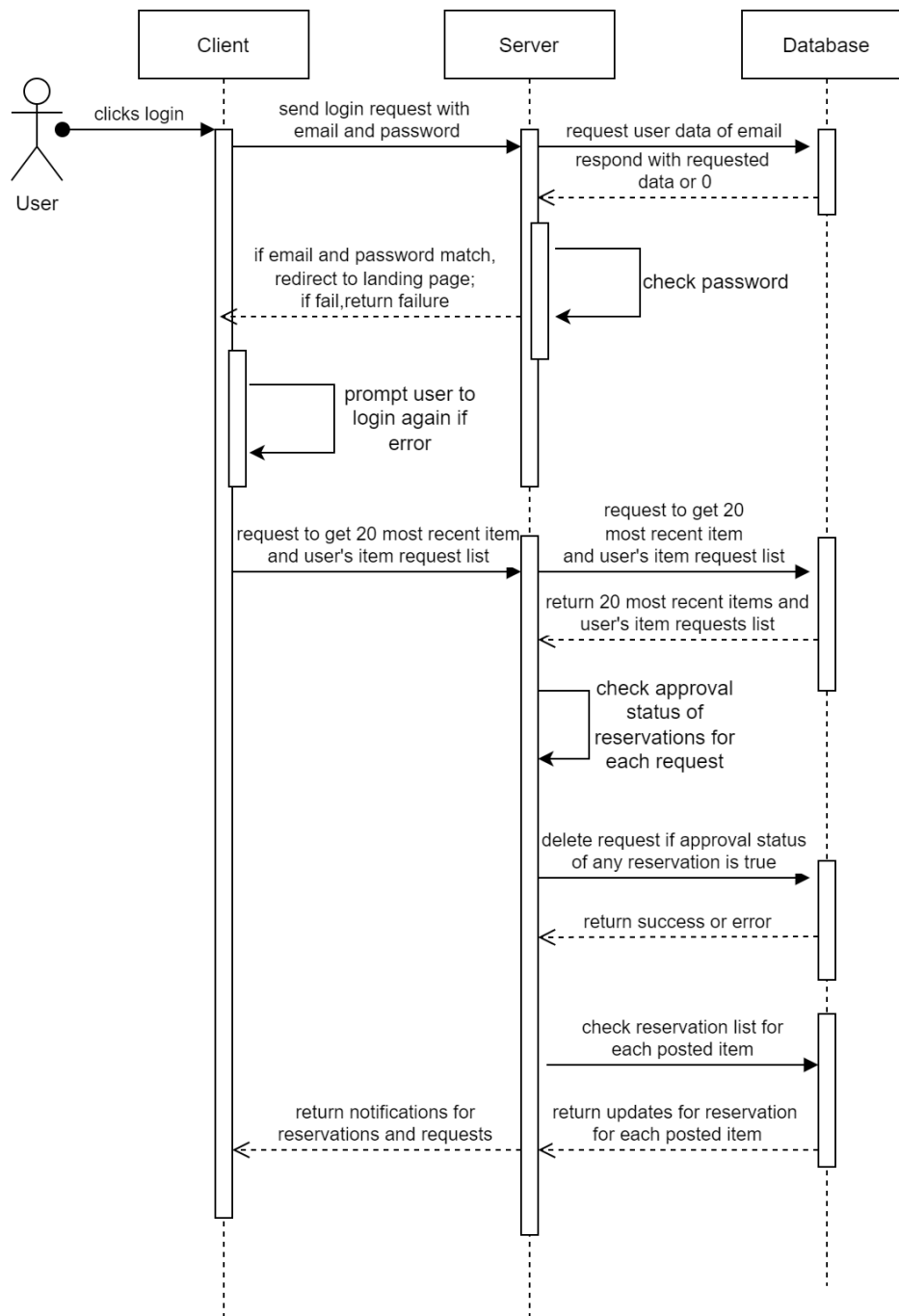
Activity Diagram





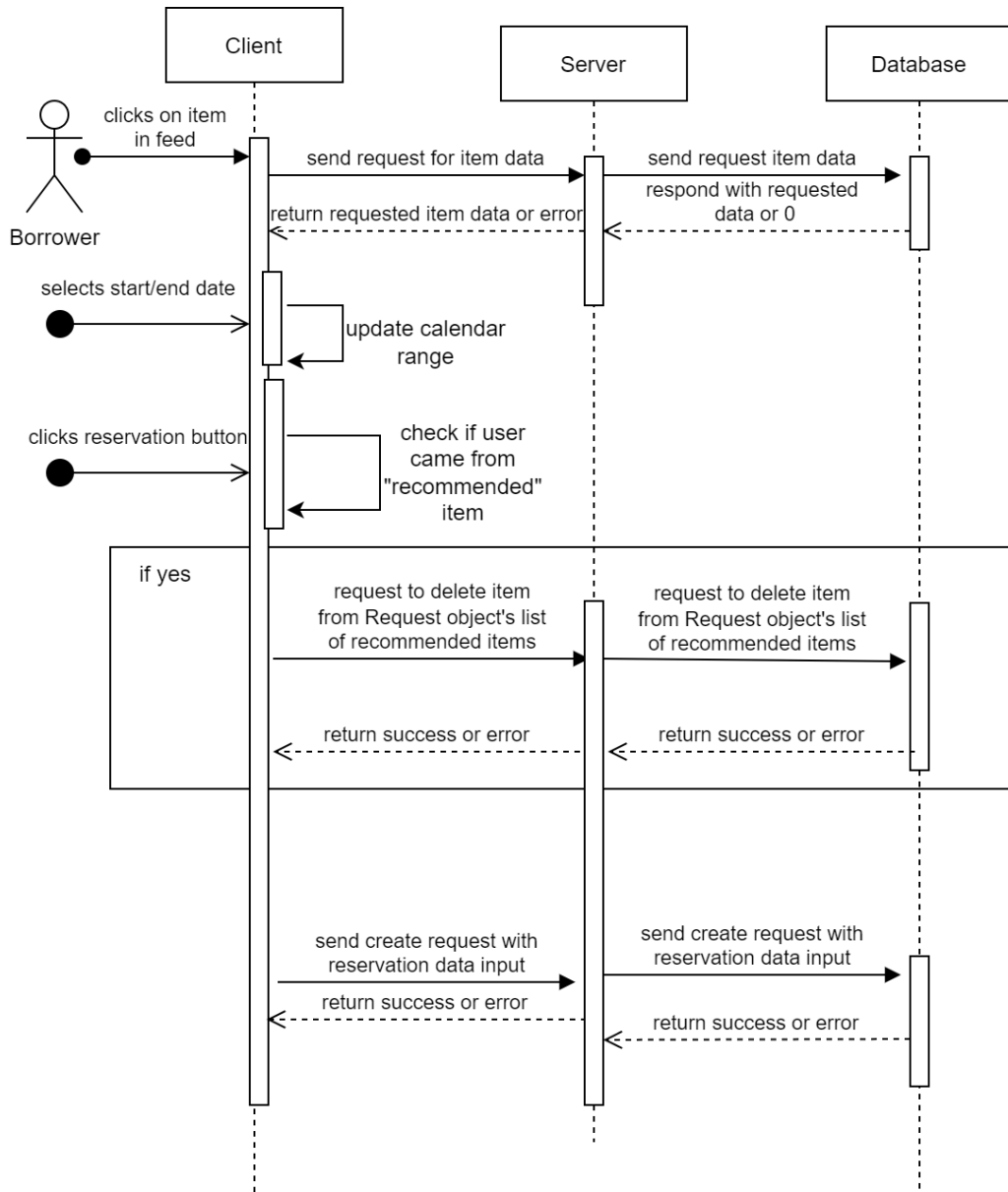
Sequence Diagrams

1. The sequence of events when a user logs in.



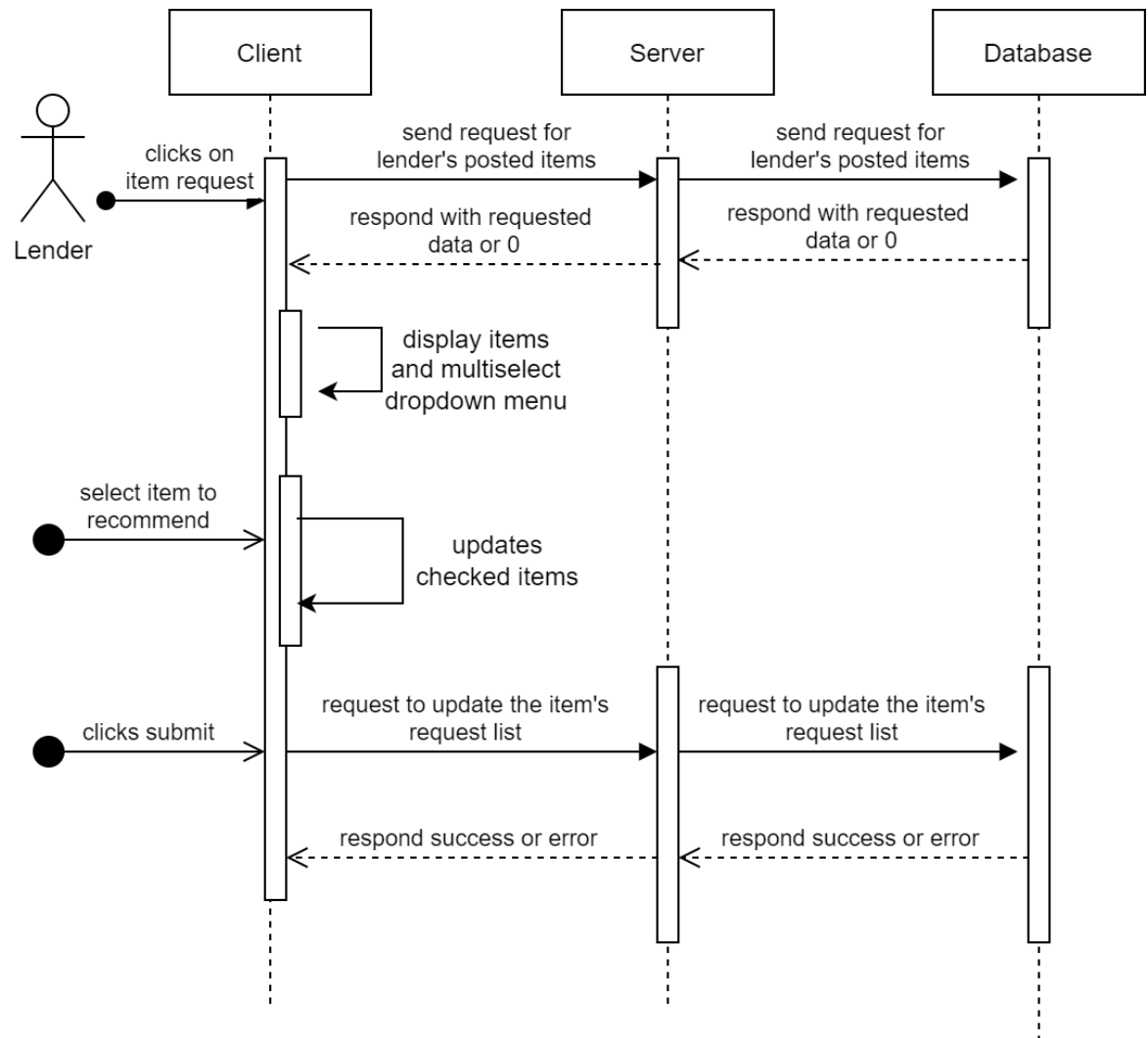


2. The sequence of events when a borrower wants to reserve an item





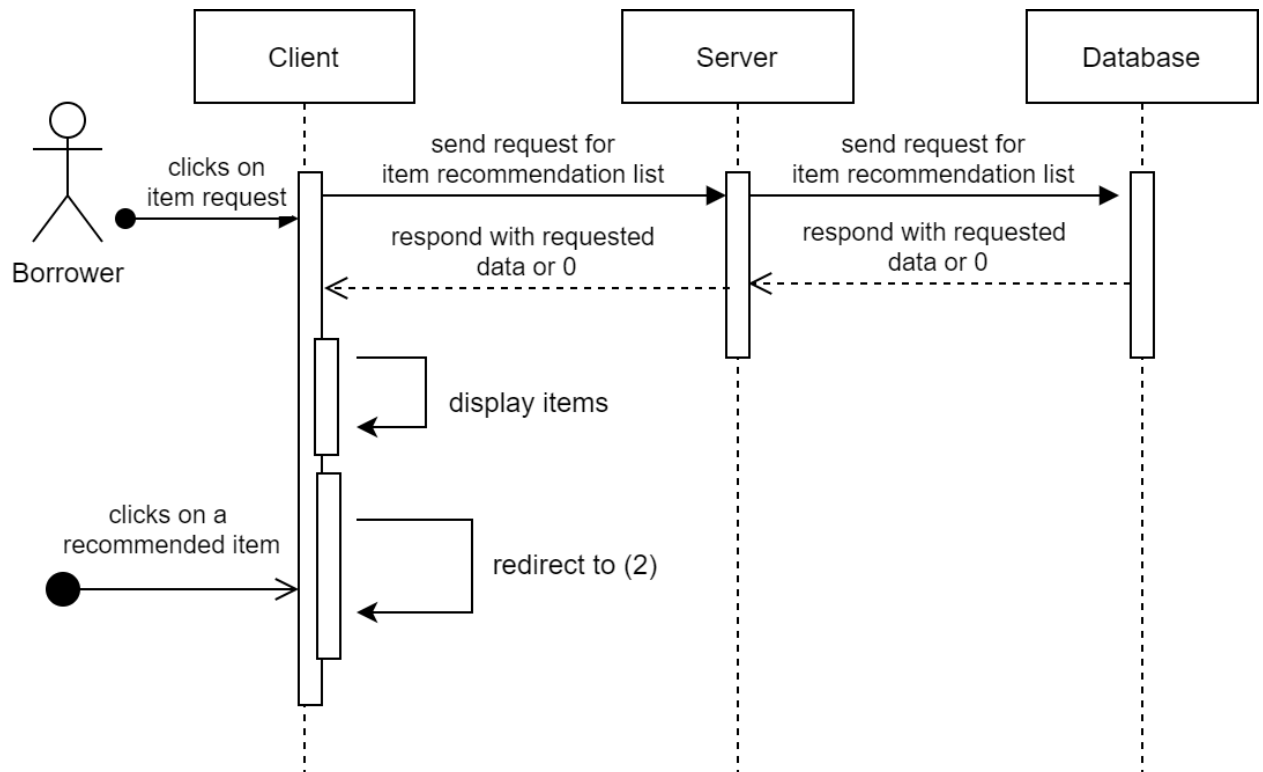
3. The sequence of events when lender responds to an item request



* Then the user with this item request will get a notification on their side

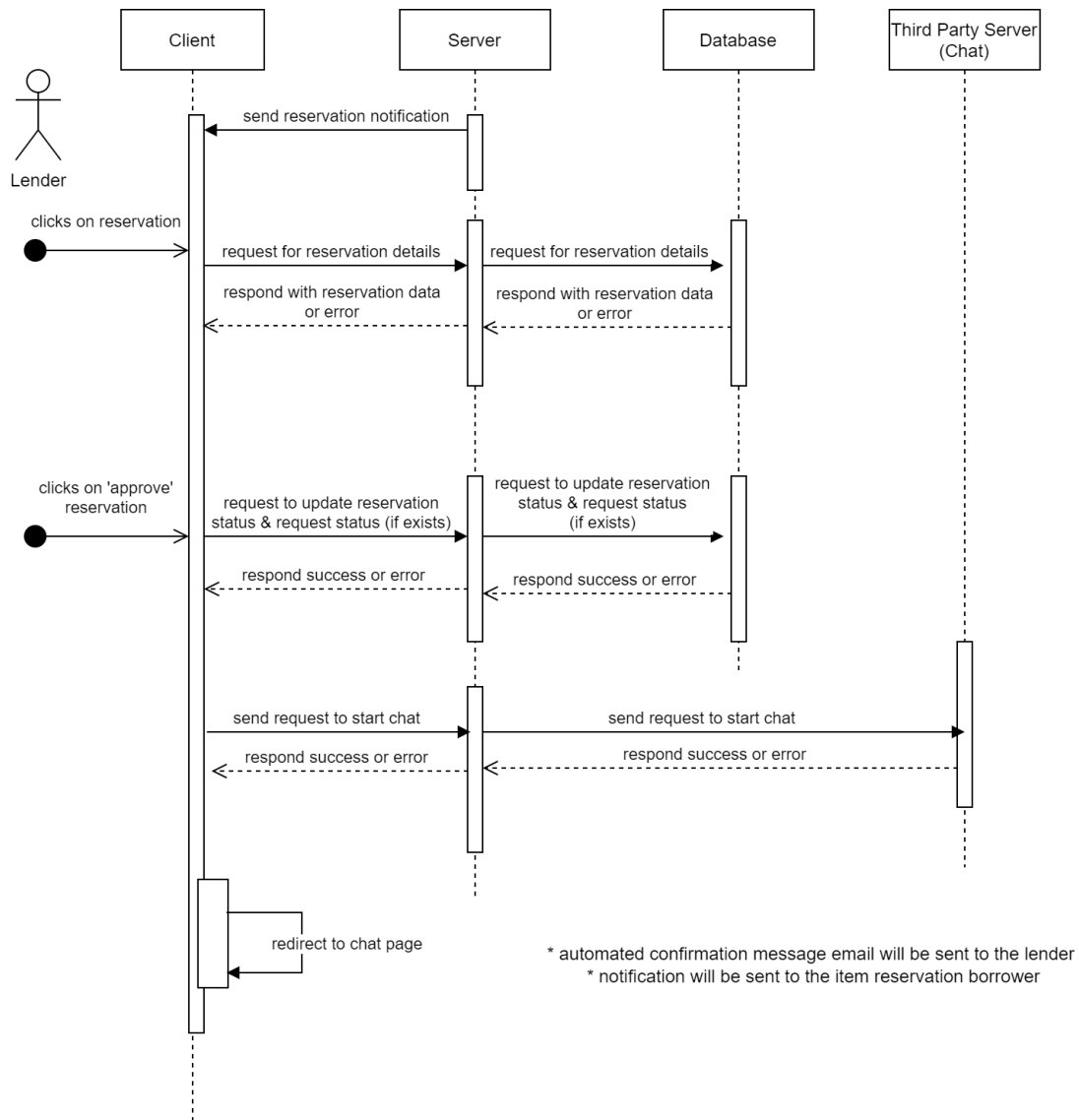


4. The sequence of events when a borrower responds to an item recommendation





5. The sequence of events when a lender is notified of their posted item's new reservation (and approves it).





6. The sequence of events when a lender is notified of their posted item's new reservation (and denies it).

